Attachment 1 (Appendix 9 from master thesis)

Below follows the full code we developed in MATLAB, for our master thesis.

```
%% Import data from spreadsheet
clc
clear all
close all
tic

% Setup the Import Options
opts = spreadsheetImportOptions("NumVariables", 15);

% Specify sheet and range
opts.Sheet = "Sheet1";
opts.DataRange = "A2:O209709";

% Specify column names and types
opts.VariableNames = ["BoligID", "Var2", "Salgsdato", "Var4", "Bydel", "Pris",
"Fellesgjeld", "Prisantydning", "Eierform", "Byggeaar", "prom", "Var12", "Etasje",
"Soverom", "Boligtype"];
opts.SelectedVariableNames = ["BoligID", "Salgsdato", "Bydel", "Pris", "Fellesgjeld",
"Prisantydning", "Eierform", "Byggeaar", "prom", "Etasje", "Soverom", "Boligtype"];
opts.VariableTypes = ["categorical", "char", "datetime", "char", "categorical",
"double", "double", "double", "categorical", "double", "double", "char", "double",
"double", "categorical"];
opts = setvaropts(opts, 3, "InputFormat", "");
opts = setvaropts(opts, [2, 4, 12], "WhitespaceRule", "preserve");
opts = setvaropts(opts, [1, 2, 4, 5, 9, 12, 15], "EmptyFieldRule", "auto");

% Import the data, adjust to personal folder
Property_Data = readtable('Data_Eiendomsverdi.xlsx', opts, "UseExcel", false);
clear opts

% Notes
% 1. All values in class "Number" that are originally "NULL" are converted
% to NaN, while other classes are still labeled "NULL"
%% Create new variables
% Create year variable
[y,m,d] = ymd(Property_Data.Salgsdato);
Property_Data.year = y+(((10/12)/10)*m);

% Create quarter, Q, variable
Property_Data.Q = string(year(Property_Data.Salgsdato)) + 'Q' +
string(quarter(Property_Data.Salgsdato));
Property_Data.Q = categorical(Property_Data.Q);

%% Edit "BoligID"
% Count number of observations/real estate tranbsactions originally dataset
numobsorg = numel(Property_Data.BoligID);
fprintf('\nNumber of observations in the original dataset: %.f\n ', numobsorg);

% Count number of observations/real estate transactions related to unique
% homes
unique_homesorg = numel(unique(Property_Data.BoligID));
fprintf('\nNumber of unique observations in the original dataset: %.f\n ',
unique_homesorg);

%% Edit "Salgsdato"
% No editing necessary
unique(Property_Data.Salgsdato)

%% Edit "Bydel"
unique(Property_Data.Bydel)
% Remove properties with "Bydel" labeled "NULL"
NULL = Property_Data.Bydel == 'NULL';
% Stats after editing bydel
fprintf('\nNumber of errors in "bydel" variable: %.f\n ', nnz(NULL));
Property_Data(NULL,:) = [];

numobsorg1 = numel(Property_Data.BoligID);
fprintf('\nNumber of observations in dataset after "bydel" edit: %.f\n ', numobsorg1);
```

```
unique_homesorg1 = numel(unique(Property_Data.BoligID));
fprintf('\nNumber of unique observations in dataset after "bydel" edit: %.f\n ',
unique_homesorg1);

%% Edit "Pris"
% Remove NaN values in "Pris"
nan_pris = isnan(Property_Data.Pris);
fprintf('\nNumber of NaN-values in "pris" variable: %.f\n ', nnz(nan_pris));
Property_Data(nan_pris,:) = [];

% Remove properties with prices lower than 500000 (SSB)
P_less500k = (Property_Data.Pris + Property_Data.Fellesgjeld) < 500000;
fprintf('\nNumber of properties with prices lower than 500.000: %.f\n ',
nnz(P_less500k));
Property_Data(P_less500k,:) = [];

numobsorg2 = numel(Property_Data.BoligID);
fprintf('\nNumber of observations in dataset after "pris" edit: %.f\n ', numobsorg2);

unique_homesorg2 = numel(unique(Property_Data.BoligID));
fprintf('\nNumber of unique observations in dataset after "pris" edit: %.f\n ',
unique_homesorg2);

%% Edit "Fellesgjeld"
fprintf('\nNumber of NaN-values in "fellesgjeld" variable: %.f\n ',
nnz(isnan(Property_Data.Fellesgjeld)));

%% Edit "Prisantydning"
fprintf('\nNumber of NaN-values in "prisantydning" variable: %.f\n ',
nnz(isnan(Property_Data.Prisantydning)));

%% Edit "Eierform"
per_eierform = groupsummary(Property_Data, 'Eierform')

% Remove {Eierform = "Aksjeleilighet", "Obl.leilighet" and "ukjent"}
fprintf('\nNumber of StOUs: %.f\n ', length(find(Property_Data.Eierform ==
'Aksjeleilighet')));
Aksjeleilighet = Property_Data.Eierform == 'Aksjeleilighet';
Property_Data(Aksjeleilighet,:) = [];

numobsorg3 = numel(Property_Data.BoligID);
fprintf('\nNumber of observations in dataset after removing StOUs: %.f\n ',
numobsorg3);

unique_homesorg3 = numel(unique(Property_Data.BoligID));
fprintf('\nNumber of unique observations in dataset after removing StOUs: %.f\n ',
unique_homesorg3);

fprintf('\nNumber of BOUs: %.f\n ', length(find(Property_Data.Eierform ==
'Obl.leilighet')));

Obl_leilighet = Property_Data.Eierform == 'Obl.leilighet';
Property_Data(Obl_leilighet,:) = [];

fprintf('\nNumber of unknowns: %.f\n ', length(find(Property_Data.Eierform ==
'Ukjent')));

Ukjent = Property_Data.Eierform == 'Ukjent';
Property_Data(Ukjent,:) = [];

numobsorg4 = numel(Property_Data.BoligID);
fprintf('\nNumber of observations in dataset: %.f\n ', numobsorg4);

unique_homesorg4 = numel(unique(Property_Data.BoligID));
fprintf('\nNumber of unique observations in dataset: %.f\n ', unique_homesorg4);

% Check the code above
unique(Property_Data.Eierform)

%% Edit "Byggeaar"
% Remove NaN values in "Byggeaar"
nan_byggeaar = isnan(Property_Data.Byggeaar);
fprintf('\nNumber of NaN-values in "Byggeaar" variable: %.f\n ', nnz(nan_byggeaar));
Property_Data(nan_byggeaar,:) = [];
```

```
% Remove properties with "Bygge√•r" labeled 0
construction_year_zero = Property_Data.Byggeaar == 0;
nnz(construction_year_zero)
Property_Data(construction_year_zero,:)=[];

numobsorg5 = numel(Property_Data.BoligID);
fprintf('\nNumber of observations in dataset: %.f\n ', numobsorg5);

unique_homesorg5 = numel(unique(Property_Data.BoligID));
fprintf('\nNumber of unique observations in dataset: %.f\n ', unique_homesorg5);

% Make categories for age of properties
% Bins for period of ages in "Byggeaar"
X = Property_Data.Byggeaar;

edges = [0, 1985, 2001, 2010, 2021];
Alder = discretize(X,edges, 'categorical', ...
    {'Age4', 'Age3', 'Age2', 'Age1'});

% Create variable related to age of dwelling
Property_Data.Alder = Alder;
per_alder = groupsummary(Property_Data, 'Alder');

%% Edit "Boligtype"
unique(Property_Data.Boligtype)
per_boligtype = groupsummary(Property_Data, 'Boligtype');

% Label the dwelling types "Rekkehus" and "Tomannsbolig" as "Smaahus"
Property_Data.Boligtype = mergecats(Property_Data.Boligtype,{'Rekkehus'
'Tomannsbolig'},'Smaahus');
per_boligtype = groupsummary(Property_Data, 'Boligtype');

%% Edit "prom"
% Remove NaN values in "prom"
nan_prom = isnan(Property_Data.prom);
fprintf('\nNumber of NaN-values in "prom" variable: %.f\n ', nnz(nan_prom));
Property_Data(nan_prom,:) = [];

% Set limitations to sizes of dwellings
% For apartments
apartmentrec = Property_Data.Boligtype == 'Leilighet';
remove_apartments(length(Property_Data.Boligtype),1) = 0;

for i = 1:length(Property_Data.Boligtype)
    if apartmentrec(i) == 1
        if Property_Data.prom(i) < 15
           remove_apartments(i) = 1;
        elseif Property_Data.prom(i) > 250
           remove_apartments(i) = 1;
        else
           remove_apartments(i) = 0;
        end
    else
        continue
    end
end
remove_apartments = logical(remove_apartments);
nnz(remove_apartments);
Property_Data(remove_apartments,:) = [];

% For eneboliger
houserec = Property_Data.Boligtype == 'Enebolig';
remove_houses(length(Property_Data.Boligtype),1) = 0;

for i = 1:length(Property_Data.Boligtype)
    if houserec(i) == 1
        if Property_Data.prom(i) < 50
           remove_houses(i) = 1;
        elseif Property_Data.prom(i) > 500
           remove_houses(i) = 1;
        else
           remove_houses(i) = 0;
        end
```

```
    else
        continue
    end
end
remove_houses = logical(remove_houses);
nnz(remove_houses);
Property_Data(remove_houses,:) = [];

% For small houses
small_houserec = Property_Data.Boligtype == 'Smaahus';
remove_small_houses(length(Property_Data.Boligtype),1) = 0;

for i = 1:length(Property_Data.Boligtype)
    if small_houserec(i) == 1
        if Property_Data.prom(i) < 40
           remove_small_houses(i) = 1;
        elseif Property_Data.prom(i) > 350
           remove_small_houses(i) = 1;
        else
           remove_small_houses(i) = 0;
        end
    else
        continue
    end
end
remove_small_houses = logical(remove_small_houses);
nnz(remove_small_houses);
Property_Data(remove_small_houses,:) = [];

numobsorg6 = numel(Property_Data.BoligID);
fprintf('\nNumber of observations in dataset after "prom" edit: %.f\n ', numobsorg6);

unique_homesorg6 = numel(unique(Property_Data.BoligID));
fprintf('\nNumber of unique observations in dataset after "prom" edit: %.f\n ',
unique_homesorg6);

%% Edit "Etasje"
% Remove observations with floor level greater than 15
F_15 = Property_Data.Etasje > 15;
nnz(F_15)
f15 = Property_Data(F_15,:);
unique(f15.BoligID);
Property_Data(F_15,:) = [];

% Remove "NaN" in "Etasje" for "Boligtype" == "Leilighet"
NaN_apartments(length(Property_Data.Boligtype),:) = 0;
rec2 = Property_Data.Boligtype == 'Leilighet';

for i = 1:length(Property_Data.Boligtype)
    if rec2(i) == 1
        if isnan(Property_Data.Etasje(i)) == 1
           NaN_apartments(i) = 1;
        else
           NaN_apartments(i) = 0;
        end
    else
        continue
    end
end

nnz(NaN_apartments)
NaN_apartments = logical(NaN_apartments);
Property_Data(NaN_apartments,:) = [];

% Remove apartments ("Leilighet") with "Etasje" equals "0"
rec = Property_Data.Boligtype == 'Leilighet';
zero_apartments(length(Property_Data.Boligtype),:) = 0;

for i = 1:length(Property_Data.Boligtype)
    if rec(i) == 1
        if Property_Data.Etasje(i) == 0
           zero_apartments(i) = 1;
        else
           zero_apartments(i) = 0;
```

```
        end
    else
        continue
    end
end

zero_apartments = logical(zero_apartments);
nnz(zero_apartments);
Property_Data(zero_apartments,:) = [];

numobsorg7 = numel(Property_Data.BoligID);
fprintf('\nNumber of observations in dataset: %.f\n ', numobsorg7);

unique_homesorg7 = numel(unique(Property_Data.BoligID));
fprintf('\nNumber of unique observations in dataset: %.f\n ', unique_homesorg7);

%% Edit "Soverom"
% Remove NaN values in "Soverom"
nan_soverom = isnan(Property_Data.Soverom);
fprintf('\nNumber of NaN-values in "soverom" variable: %.f\n ', nnz(nan_soverom));

Property_Data(nan_soverom,:) = [];

% Remove properties with more than 6 bedrooms
B_6 = Property_Data.Soverom > 6;
nnz(B_6);
Property_Data(B_6,:) = [];

numobsorg8 = numel(Property_Data.BoligID);
fprintf('\nNumber of observations in dataset: %.f\n ', numobsorg8);

unique_homesorg8 = numel(unique(Property_Data.BoligID));
fprintf('\nNumber of unique observations in dataset: %.f\n ', unique_homesorg8);

%% Further adjustments according to SSB
Property_Data.kvmpris =
(Property_Data.Pris+Property_Data.Fellesgjeld)./Property_Data.prom;

%% Transactions removed in cleaning process
% Count number of observations/real estate tranbsactions in cleaned dataset
numobs = numel(Property_Data.BoligID);
fprintf('\nNumber of observations in the cleaned dataset: %.f\n ', numobs);

% Count number of observations/real estate transactions related to unique
% homes in cleaned dataset
unique_homes = numel(unique(Property_Data.BoligID));
fprintf('\nNumber of unique observations in the cleaned dataset: %.f\n ',
unique_homes);

% Observations removed in the cleaning process
removed = numobsorg - numobs;
fprintf('\nNumber of observations removed in the cleaning process: %.f\n ', removed);
share = (removed/numobsorg)*100;
fprintf('\nIn the cleaning process we removed %.4f percent of the dataset\n ', share);

% Observations removed in the cleaning process (related to unique
% transactions)
removed2 = unique_homesorg - unique_homes;
fprintf('\nNumber of unique observations removed in the cleaning process: %.f\n ',
removed2);
share2 = (removed2/unique_homesorg)*100;
fprintf('\nIn the cleaning process we removed %.4f percent of the unique
observations\n ', share2);

%% Summary Statistics
Property_Data.Apartment = Property_Data.Boligtype == 'Leilighet';

format bank
% Testing years as group
Data_by_year = Property_Data;
X = Data_by_year.year;
edges = [2010, 2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019, 2020, 2021];
Data_by_year.year = discretize(X, edges, 'categorical', ...
```

```
    {'2010', '2011', '2012', '2013', '2014', '2015', '2016', '2017', '2018', '2019',
'2020'});
per_year = groupsummary(Data_by_year, 'year', {'mean', 'std'}, {'Apartment', 'Pris',
'prom', 'kvmpris', 'Soverom'});
%per_year = removevars(per_year, [4, 5, 10, 11, 14, 16, 17]);
per_year.Properties.VariableNames{'year'} = 'Group';
per_year.Group = nominal(per_year.Group);

per_boligtype = groupsummary(Property_Data, 'Boligtype', {'mean', 'std'},
{'Apartment', 'Pris', 'prom', 'kvmpris', 'Soverom'});
%per_boligtype = removevars(per_boligtype, [4, 5, 10, 11, 14, 16, 17]);
per_boligtype.Properties.VariableNames{'Boligtype'} = 'Group';
per_boligtype.Group = nominal(per_boligtype.Group);

per_bydel = groupsummary(Property_Data, 'Bydel', {'mean', 'std'}, {'Apartment',
'Pris', 'prom', 'kvmpris', 'Soverom'});
%per_bydel = removevars(per_bydel, [4, 5, 10, 11, 14, 16, 17]);
per_bydel.Properties.VariableNames{'Bydel'} = 'Group';
per_bydel.Group = nominal(per_bydel.Group);

summary_table = [per_boligtype; per_year; per_bydel];
writetable(summary_table, 'Summary_table.xlsx','Sheet',1)

Property_Data.Apartment = [];

%% Import risk-free rates (innskuddsrente)
opts = spreadsheetImportOptions("NumVariables", 2);

% Specify sheet and range
opts.Sheet     = "Bankinnskuddrente";
opts.DataRange = "A4:B14";

% Specify column names and types
opts.VariableNames         = ["year", "rf"];
opts.SelectedVariableNames = ["year", "rf"];
opts.VariableTypes         = ["categorical", "double"];
opts                       = setvaropts(opts, 1, "EmptyFieldRule", "auto");

% Setup rules for import
opts.ImportErrorRule = "error";
opts                 = setvaropts(opts, 2, "TreatAsMissing", '');

% Import the data, adjust to personal folder
Rf = readtable('Bankinnskuddrente.xlsx', opts, "UseExcel", false);

% Clear temporary variables
clear opts

Rf.rf = Rf.rf./100;

%% Import rental prices
% Setup the Import Options
opts = spreadsheetImportOptions("NumVariables", 6);

% Specify sheet and range
opts.Sheet = "Sheet1";
opts.DataRange = "A3:F46";

% Specify column names and types
opts.VariableNames = ["Time", "room1", "room2", "room3", "room4", "room5_"];
opts.SelectedVariableNames = ["Time", "room1", "room2", "room3", "room4", "room5_"];
opts.VariableTypes = ["categorical", "double", "double", "double", "double",
"double"];
opts = setvaropts(opts, 1, "EmptyFieldRule", "auto");

% % Import the data, adjust to personal folder
Rent_Data = readtable("rental_price_Oslo", opts, "UseExcel", false);

% Clear temporary variables
clear opts

% Clear all variables except the ones needed for further analysis
clearvars -except Property_Data summary_table Rf Rent_Data
```

```
%% HEDONIC REGRESSION FOR APARTMENTS
% 1. PREPERATION OF DATASET
apartments     = Property_Data.Boligtype == 'Leilighet';
apartment_set  = Property_Data(apartments,:);

% Make categorical for dummy-purposes
apartment_set.Bydel    = categorical(apartment_set.Bydel);
apartment_set.Alder    = categorical(apartment_set.Alder);
apartment_set.Q        = categorical(apartment_set.Q);
apartment_set.Eierform = categorical(apartment_set.Eierform);

% Reorder categories/dummies to find reference variables
apartment_set.Bydel    = reordercats(apartment_set.Bydel,    {'Gr√°nerl√∏kka', 'Alna',
'Bjerke', 'Frogner', 'Gamle Oslo', 'Grorud', 'Nordre Aker', 'Nordstrand', 'Sagene',
'Sentrum', 'St. Hanshaugen', 'Stovner', 'S√∏ndre Nordstrand', 'Ullern', 'Vestre Aker',
'√òstensj√∏', 'NULL', 'Marka'});
apartment_set.Alder    = reordercats(apartment_set.Alder,    {'Age1', 'Age2', 'Age3',
'Age4'});
apartment_set.Eierform = reordercats(apartment_set.Eierform, {'Selveier',
'Borettslag', 'Obl.leilighet', 'Aksjeleilighet', 'Ukjent'});

% Remove unused categories
apartment_set.Bydel    = removecats(apartment_set.Bydel);
apartment_set.Eierform = removecats(apartment_set.Eierform);

% Check that we in fact removed unused categories
categories(apartment_set.Bydel)
categories(apartment_set.Alder)
categories(apartment_set.Eierform)

% Make log prices and sizes
apartment_set.log_pris = log(apartment_set.Pris);
apartment_set.log_prom = log(apartment_set.prom);

% 2. FIND PERFECT REGRESSION MODEL
hedonic_apartments                          = zeros(8,5);
hedonic_apartments                          = array2table(hedonic_apartments);
hedonic_apartments.Properties.VariableNames = {'key_stats', 'LTD_Model',
'LOG_TD_Model', 'LOG_TD_Int_Q_District', 'LOG_TD_Int_Size_District'};
hedonic_apartments.key_stats                =
categorical(hedonic_apartments.key_stats);

hedonic_apartments.key_stats(1) = 'Nu_Obs';
hedonic_apartments.key_stats(2) = 'Nu_Coeff';
hedonic_apartments.key_stats(3) = 'R_squared';
hedonic_apartments.key_stats(4) = 'Adj_R_squared';
hedonic_apartments.key_stats(5) = 'RMSE';

% Histogram plot of observed prices and sizes of apartments
h = histogram(apartment_set.Pris);
xlim([0 20000000]);
ax = ancestor(h, 'axes');
ax.XAxis.Exponent = 0;
xtickformat('%,0.f');
ylabel('Number of apartments', 'FontSize', 12);
xlabel('(a) Observed price', 'FontSize', 12);

skewness(apartment_set.Pris)
kurtosis(apartment_set.Pris)

h2 = histogram(apartment_set.prom);
ylabel('Number of apartments', 'FontSize', 12);
xlabel('(b) Observed size', 'FontSize', 12);

skewness(apartment_set.prom)
kurtosis(apartment_set.prom)

% Histogram plot of observed log(prices) and log(sizes) of apartments
h = histogram(apartment_set.log_pris);
xlim([11 18]);
ax = ancestor(h, 'axes');
ax.XAxis.Exponent = 0;
xtickformat('%,0.f');
```

```
ylabel('Number of apartments', 'FontSize', 12);
xlabel('(a) Observed log(price)', 'FontSize', 12);

skewness(apartment_set.log_pris)
kurtosis(apartment_set.log_pris)

h2 = histogram(apartment_set.log_prom);
ylabel('Number of apartments', 'FontSize', 12);
xlabel('(b) Observed log(size)', 'FontSize', 12);

skewness(apartment_set.log_prom)
kurtosis(apartment_set.log_prom)

% 2.1 Run first-try regression - Linear model with prom, Q, Bydel, Soverom, Alder on
Pris
% Explanatory variables: prom, Soverom, Etasje - continous
% Explanatory variables: Q, Bydel, Eierform and Alder - dummies

fit_test1 = fitlm(apartment_set,'Pris~prom+Q+Bydel+Eierform+Soverom+Alder+Etasje');
plotResiduals(fit_test1,'fitted')
xlabel('(a) Estimated prices, LTD model')

hedonic_apartments.LTD_Model(1) = fit_test1.NumObservations;
hedonic_apartments.LTD_Model(2) = fit_test1.NumCoefficients;
hedonic_apartments.LTD_Model(3) = fit_test1.Rsquared.Ordinary;
hedonic_apartments.LTD_Model(4) = fit_test1.Rsquared.Adjusted;
hedonic_apartments.LTD_Model(5) = fit_test1.RMSE;

% 2.2 Run second-try regression - Log model with log(prom), Q, Bydel, Soverom, Alder
on log(pris)
% Explanatory variables log(prom), soverom - continous
% Explanatory variables Q, Bydel, Eierform, Alder and Etasje - dummies
apartment_set.Etasje  = categorical(apartment_set.Etasje);
apartment_set.Etasje  = reordercats(apartment_set.Etasje, {'1', '2', '3', '4', '5',
'6', '7', '8', '9', '10', '11', '12', '13', '14', '15', '-1', '-2', '-3', '-4'});

fit_test2 =
fitlm(apartment_set,'log_pris~log_prom+Q+Bydel+Eierform+Soverom+Alder+Etasje');
plotResiduals(fit_test2,'fitted')
xlabel('(b) Estimated prices, LOG-TD model')

hedonic_apartments.LOG_TD_Model(1) = fit_test2.NumObservations;
hedonic_apartments.LOG_TD_Model(2) = fit_test2.NumCoefficients;
hedonic_apartments.LOG_TD_Model(3) = fit_test2.Rsquared.Ordinary;
hedonic_apartments.LOG_TD_Model(4) = fit_test2.Rsquared.Adjusted;
hedonic_apartments.LOG_TD_Model(5) = fit_test2.RMSE;

% 2.3 Run third-try regression - Log model with log(prom), Q, Bydel, Soverom, Alder on
log(pris)
% Explanatory variables log(prom) and soverom - continous
% Explanatory variables Q, Bydel, Eierform, Alder, Etasje - dummies
% Interaction terms between bydel and tid

fit_test3 =
fitlm(apartment_set,'log_pris~log_prom+Q+Bydel+Eierform+Soverom+Alder+Etasje+Bydel:Q')
;
plotResiduals(fit_test3,'fitted')

hedonic_apartments.LOG_TD_Int_Q_District(1) = fit_test3.NumObservations;
hedonic_apartments.LOG_TD_Int_Q_District(2) = fit_test3.NumCoefficients;
hedonic_apartments.LOG_TD_Int_Q_District(3) = fit_test3.Rsquared.Ordinary;
hedonic_apartments.LOG_TD_Int_Q_District(4) = fit_test3.Rsquared.Adjusted;
hedonic_apartments.LOG_TD_Int_Q_District(5) = fit_test3.RMSE;

% 2.4 Run fourth-try regression - Log model with log(prom), Q, Bydel, Soverom, Alder
on log(pris)
% Explanatory variables log(prom) and soverom - continous
% Explanatory variables Q, Bydel, Eierform, Alder, Etasje - dummies
% Interaction terms between bydel and prom

fit_test4 =
fitlm(apartment_set,'log_pris~log_prom+Q+Bydel+Eierform+Soverom+Alder+Etasje+log_prom:
Bydel');
plotResiduals(fit_test4,'fitted')
```

```
hedonic_apartments.LOG_TD_Int_Size_District(1) = fit_test4.NumObservations;
hedonic_apartments.LOG_TD_Int_Size_District(2) = fit_test4.NumCoefficients;
hedonic_apartments.LOG_TD_Int_Size_District(3) = fit_test4.Rsquared.Ordinary;
hedonic_apartments.LOG_TD_Int_Size_District(4) = fit_test4.Rsquared.Adjusted;
hedonic_apartments.LOG_TD_Int_Size_District(5) = fit_test4.RMSE;

% We continue to use the LOG-TD model without interaction terms
apartment_model = fit_test2;

% 3. ALLOCATE TABLE FOR ESTIMATED PRICES (BASED ON MODEL FROM PT. 2) FOR EVERY QUARTER
FOR EACH ID
apartment_set    = sortrows(apartment_set,'BoligID','ascend');
[C,ia,ic]        = unique(apartment_set.BoligID,'first');
price_apartments = zeros(length(C)*44, width(apartment_set));
price_apartments = array2table(price_apartments);

price_apartments.Properties.VariableNames = {'BoligID', 'Salgsdato', 'Bydel', 'Pris',
'Fellesgjeld', 'Prisantydning', 'Eierform', 'Byggeaar', 'prom', 'Etasje', 'Soverom',
'Boligtype', 'year', 'Q', 'Alder', 'kvmpris', 'log_pris', 'log_prom' };

Q_rep   = repmat(["Q1"; "Q2"; "Q3";"Q4"],(length(C)*44)/4,1);
x       = (2010:2020)';
n       = 4;
newx    =
[repmat(x(1),n,1);repmat(x(2),n,1);repmat(x(3),n,1);repmat(x(4),n,1);repmat(x(5),n,1);
repmat(x(6),n,1);repmat(x(7),n,1);repmat(x(8),n,1);repmat(x(9),n,1);repmat(x(10),n,1);
repmat(x(11),n,1)];
Y_order = repmat(newx,(length(C)),1);

price_apartments.Q_order = string(Y_order(:,1)) + Q_rep(:,1);
price_apartments.Q       = price_apartments.Q_order;
price_apartments.Q_order = [];

price_apartments.BoligID = categorical(price_apartments.BoligID);
price_apartments.BoligID = repelem(C, 44);

price_apartments           = convertvars(price_apartments,{'BoligID', 'Bydel',
'Eierform', 'Boligtype', 'Etasje', 'Alder', 'Q'},'categorical');
price_apartments.Salgsdato =
datetime(price_apartments.Salgsdato,'ConvertFrom','yyyymmdd');

b = 1:length(newx):height(price_apartments);

% Create set containing only the unique observations of house_set. When
% multiple unique values it stores the first observation
uniq_apartment_set = apartment_set(ia,:);

% Allocate variables into price_houses to use for regression model
price_apartments(:,[3 5 7 9 10 11 15 18]) = repelem(uniq_apartment_set(:,[3 5 7 9 10
11 15 18]), 44, 1);

% Store relevant variables for regression model in new table
x3 = price_apartments(:,[18 14 3 7 11 15 10]);

% GET COLUMN OF ESTIMATED PRICES BASED ON REGRESSION MODEL
price_apartments.log_estimated_price    = predict(apartment_model, x3);
price_apartments.estimated_price        = exp(price_apartments.log_estimated_price);
price_apartments.lagged_estimated_price =
lagmatrix(price_apartments.estimated_price,1);

% 4. IMPLEMENT RELEVANT COSTS RELATED TO HOUSING INVESTMENTS
% Transaction costs buyer, b
gebyr_b             = zeros(height(price_apartments),1);
dokavg_b            = zeros(height(price_apartments),1);
insurance_b         = zeros(height(price_apartments),1);
price_apartments.t_cost = zeros(height(price_apartments),1);

% If type of ownership = 'Selveier'
idx_selv             = price_apartments.Eierform == "Selveier";
gebyr_b(idx_selv)    = 540 + 540 + 172;
dokavg_b(idx_selv)   =
0.025.*(price_apartments.estimated_price(idx_selv)+price_apartments.Fellesgjeld(idx_se
lv));
```

```
insurance_b(idx_selv) = 7400;

% If type of ownership = 'Borettslag'
idx_andel              = price_apartments.Eierform == "Borettslag";
gebyr_b(idx_andel)     = 440 + 440 + 172;
dokavg_b(idx_andel)    = 0;
insurance_b(idx_andel) = 4100;

% Compute total transaction cost for buyer, b, for each observation/quarter
price_apartments.t_cost        = gebyr_b+dokavg_b+insurance_b;
price_apartments.lagged_t_cost = lagmatrix(price_apartments.t_cost,1);

% Transaction costs seller, s
eierskiftegebyr_s     = zeros(height(price_apartments),1);
public_fees_s         = zeros(height(price_apartments),1);
insurance_s           = zeros(height(price_apartments),1);
commission_s          = zeros(height(price_apartments),1);
price_apartments.t_cost_s = zeros(height(price_apartments),1);
indices_selv          = zeros(height(price_apartments),1);
indices_andel         = zeros(height(price_apartments),1);

% If type of ownership = 'Selveier'
idx_selv                    = price_apartments.Eierform == "Selveier";
public_fees_s(idx_selv)     = 8430;
eierskiftegebyr_s(idx_selv) = 0;
insurance_s(idx_selv)       =
max(0.0033*(price_apartments.estimated_price(idx_selv)+price_apartments.Fellesgjeld(id
x_selv)), 4000);
indices_selv                = insurance_s > 24000;
indices_selv                = logical(indices_selv);
insurance_s(indices_selv)   = 24000;

% If type of ownership = 'Borettslag'
idx_andel                    = price_apartments.Eierform == "Borettslag";
public_fees_s(idx_andel)     = 14319;
eierskiftegebyr_s(idx_andel) = 4858;
insurance_s(idx_andel)       =
max(0.002*(price_apartments.estimated_price(idx_andel)+price_apartments.Fellesgjeld(id
x_andel)), 2500);
indices_andel                = zeros(height(price_apartments),1);
indices_andel(idx_andel)     = insurance_s(idx_andel) > 24000;
indices_andel                = logical(indices_andel);
insurance_s(indices_andel)   = 24000;

% Commission to broker independent of type of ownership
commission_s =
max(0.02*(price_apartments.estimated_price+price_apartments.Fellesgjeld),40000);

% Compute total transaction cost for seller, s, for each observation/quarter
price_apartments.t_cost_s = public_fees_s+eierskiftegebyr_s+insurance_s+commission_s;

% Compute transaction cost rates for buyer and seller
% For buyer
t_cost_rate_apartm   =
mean(price_apartments.t_cost./price_apartments.estimated_price);
t_cost_rate_Q = t_cost_rate_apartm/4;

% For seller
t_cost_s_rate   = mean(price_apartments.t_cost_s./price_apartments.estimated_price);
t_cost_s_rate_Q = t_cost_s_rate/4;

% Define holding period for each dwelling
holding_period = 11;

% Define living expenses per quarter - reference apartment is 70 sqm
liv_exp_a_70_Q = ([76734/4, 82530/4, 83430/4, 89095/4, 88857/4, 83084/4, 83769/4,
86976/4, 91419/4, 103712/4, 87905/4])';

% Allocate correct liwing expenses to each quarter in price_houses set
n = 4;
liv_exp_apartments =
[repmat(liv_exp_a_70_Q(1),n,1);repmat(liv_exp_a_70_Q(2),n,1);repmat(liv_exp_a_70_Q(3),
n,1);repmat(liv_exp_a_70_Q(4),n,1);repmat(liv_exp_a_70_Q(5),n,1);repmat(liv_exp_a_70_Q
```

```
(6),n,1);repmat(liv_exp_a_70_Q(7),n,1);repmat(liv_exp_a_70_Q(8),n,1);repmat(liv_exp_a_
70_Q(9),n,1);repmat(liv_exp_a_70_Q(10),n,1);repmat(liv_exp_a_70_Q(11),n,1)];
price_apartments.ordered_liv_exp_70 = repmat(liv_exp_apartments,(length(C)),1);

% Compute living expenses based on a reference apartment of 70 sqm
price_apartments.living_exp =
(price_apartments.prom./70).*price_apartments.ordered_liv_exp_70;
avg_living_exp_Q =
mean(price_apartments.living_exp./price_apartments.estimated_price);

% Allocate correct implicit rents
implicit_rent     = zeros(height(price_apartments),1);
rental_allocation = repmat(Rent_Data, length(C), 1);

implicit_rent(find(price_apartments.Soverom == 0)) =
rental_allocation.room1(find(price_apartments.Soverom == 0));
implicit_rent(find(price_apartments.Soverom == 1)) =
rental_allocation.room1(find(price_apartments.Soverom == 1));
implicit_rent(find(price_apartments.Soverom == 2)) =
rental_allocation.room2(find(price_apartments.Soverom == 2));
implicit_rent(find(price_apartments.Soverom == 3)) =
rental_allocation.room3(find(price_apartments.Soverom == 3));
implicit_rent(find(price_apartments.Soverom == 4)) =
rental_allocation.room4(find(price_apartments.Soverom == 4));
implicit_rent(find(price_apartments.Soverom >= 5)) =
rental_allocation.room5_(find(price_apartments.Soverom >= 5));

price_apartments.implicit_rent = implicit_rent;
avg_implicit_rent_Q =
mean(price_apartments.implicit_rent./price_apartments.estimated_price);

% Implement tax rates for sample period
tax_rates = ([ 0.28, 0.28, 0.28, 0.28, 0.27, 0.27, 0.25, 0.24, 0.23, 0.22, 0.22])';
n = 4;
tax_rates_order =
[repmat(tax_rates(1),n,1);repmat(tax_rates(2),n,1);repmat(tax_rates(3),n,1);repmat(tax
_rates(4),n,1);repmat(tax_rates(5),n,1);repmat(tax_rates(6),n,1);repmat(tax_rates(7),n
,1);repmat(tax_rates(8),n,1);repmat(tax_rates(9),n,1);repmat(tax_rates(10),n,1);repmat
(tax_rates(11),n,1)];

% Allocate correct tax rates to each quarter in price_houses set
price_apartments.tax_rates = repmat(tax_rates_order,(length(C)),1);

% Compute transaction costs for different ownership types (apartments)
SOUs_apartments         = price_apartments((price_apartments.Eierform ==
'Selveier'),:);
t_cost_rate_SOUs        =
mean(SOUs_apartments.t_cost./SOUs_apartments.estimated_price);
SOU_share_apartments    = height(SOUs_apartments)/height(price_apartments);

co_ops_apartments       = price_apartments((price_apartments.Eierform ==
'Borettslag'),:);
t_cost_rate_co_ops      =
mean(co_ops_apartments.t_cost./co_ops_apartments.estimated_price);
Co_ops_share_apartments = height(co_ops_apartments)/height(price_apartments);

t_cost_rate_s_SOUs      =
mean(SOUs_apartments.t_cost_s./SOUs_apartments.estimated_price);
t_cost_rate_s_co_ops    =
mean(co_ops_apartments.t_cost_s./co_ops_apartments.estimated_price);

avg_living_exp_SOUs_Q   =
mean(SOUs_apartments.living_exp./SOUs_apartments.estimated_price);
avg_living_exp_co_ops_Q =
mean(co_ops_apartments.living_exp./co_ops_apartments.estimated_price);

% Create a table to store different cost rates
avg_cost_rates_apartments                          = zeros(4,5);
avg_cost_rates_apartments                          =
array2table(avg_cost_rates_apartments);
avg_cost_rates_apartments.Properties.VariableNames = {'Average_cost_rates', 'Yearly',
'Quarterly', 'SOUs_Y', 'Co_ops_Y'};
avg_cost_rates_apartments.Average_cost_rates       =
categorical(avg_cost_rates_apartments.Average_cost_rates);
```

```
% Import stats for cost rates
avg_cost_rates_apartments.Average_cost_rates(1) = 't_cost_buyer';
avg_cost_rates_apartments.Yearly(1)             = t_cost_rate_apartm;
avg_cost_rates_apartments.Quarterly(1)          = t_cost_rate_Q;
avg_cost_rates_apartments.SOUs_Y(1)             = t_cost_rate_SOUs;
avg_cost_rates_apartments.Co_ops_Y(1)           = t_cost_rate_co_ops;

avg_cost_rates_apartments.Average_cost_rates(2) = 't_cost_seller';
avg_cost_rates_apartments.Yearly(2)             = t_cost_s_rate;
avg_cost_rates_apartments.Quarterly(2)          = t_cost_s_rate_Q;
avg_cost_rates_apartments.SOUs_Y(2)             = t_cost_rate_s_SOUs;
avg_cost_rates_apartments.Co_ops_Y(2)           = t_cost_rate_s_co_ops;

avg_cost_rates_apartments.Average_cost_rates(3) = 'living_expenses';
avg_cost_rates_apartments.Yearly(3)             = avg_living_exp_Q * 4;
avg_cost_rates_apartments.Quarterly(3)          = avg_living_exp_Q;
avg_cost_rates_apartments.SOUs_Y(3)             = avg_living_exp_SOUs_Q * 4;
avg_cost_rates_apartments.Co_ops_Y(3)           = avg_living_exp_co_ops_Q * 4;

avg_cost_rates_apartments.Average_cost_rates(4) = 'implicit_rent';
avg_cost_rates_apartments.Yearly(4)             = avg_implicit_rent_Q * 4;
avg_cost_rates_apartments.Quarterly(4)          = avg_implicit_rent_Q;

% 5. ALLOCATE TABLE FOR PRICE INDICES
priceindex_apartments          = table();
Y_index                        = Y_order(1:44);
Q_index                        = Q_rep(1:44);
priceindex_apartments.Q_order  = string(Y_index(:,1)) + Q_index(:,1);
priceindex_apartments.Q_order  = categorical(priceindex_apartments.Q_order);

% Create a table to store returns, std, risk free, sharpe
sum_stats_apartments                          = zeros(10,7);
sum_stats_apartments                          = array2table(sum_stats_apartments);
sum_stats_apartments.Properties.VariableNames = {'House_Indices', 'Return',
'std_dev','Excess_return','Sharpe_ratio', 'Skewness', 'Kurtosis'};
sum_stats_apartments.House_Indices            =
categorical(sum_stats_apartments.House_Indices);

% Import risk-free rates (innskuddsrente)
n = 4;
priceindex_apartments.rf = zeros(44,1);
priceindex_apartments.rf =
[repmat(Rf.rf(1),n,1);repmat(Rf.rf(2),n,1);repmat(Rf.rf(3),n,1);repmat(Rf.rf(4),n,1);r
epmat(Rf.rf(5),n,1);repmat(Rf.rf(6),n,1);repmat(Rf.rf(7),n,1);repmat(Rf.rf(8),n,1);rep
mat(Rf.rf(9),n,1);repmat(Rf.rf(10),n,1);repmat(Rf.rf(11),n,1)];
priceindex_apartments.rf = priceindex_apartments.rf./4;

% 6. COMPUTATION OF MEAN PRICES FOR HOUSES AND PRICE INDEX BASED ON MEAN PRICES
% Calculate mean prices for houses
priceindex_apartments.mean_prices = zeros(height(priceindex_apartments), end);
for i = 1:length(priceindex_apartments.Q_order)
    priceindex_apartments.mean_prices(i) =
mean(price_apartments.estimated_price(find(price_apartments.Q ==
priceindex_apartments.Q_order(i))));
end

% Calculate price index for houses based on mean prices
priceindex_apartments.index_values = zeros(height(priceindex_apartments), end);
for i = 1:length(priceindex_apartments.index_values)
    priceindex_apartments.index_values(i) =
priceindex_apartments.mean_prices(i)/priceindex_apartments.mean_prices(1);
end

% 7. COMPUTATION OF QUARTERLY RETURNS ON ASSET/HOUSE FROM PERIOD TO PERIOD, NO COSTS
% Compute quarterly returns for each unique ID
price_apartments.Q_ret_no_costs =
(price_apartments.estimated_price./price_apartments.lagged_estimated_price)-1;
price_apartments.Q_ret_no_costs(1:44:end) = NaN;

% Compute mean quarterly returns for houses (mean of all IDs per quarter) (no costs)
priceindex_apartments.Q_ret_no_costs = zeros(height(priceindex_apartments), end);

for i = 1:height(priceindex_apartments)
```

```
    priceindex_apartments.Q_ret_no_costs(i) =
nanmean(price_apartments.Q_ret_no_costs(find(price_apartments.Q ==
priceindex_apartments.Q_order(i))));
end

% Compute mean of mean quarterly returns for houses (no costs)
mean_Q_ret_no_costs = nanmean(priceindex_apartments.Q_ret_no_costs);

% Compute excess quarterly returns for houses (no costs)
priceindex_apartments.excess_Q_ret_no_costs = priceindex_apartments.Q_ret_no_costs-
priceindex_apartments.rf;

% Compute mean excess quarterly returns for houses (no costs)
mean_excess_Q_ret_no_costs = nanmean(priceindex_apartments.excess_Q_ret_no_costs);

% Compute standard deviation for each unique ID
st_Q_ret_no_costs= zeros(length(C),1);
time(1,:) = clock;
for i = 1:length(C)
    if b(i)<b(length(C))
        st_Q_ret_no_costs(i) = std(price_apartments.Q_ret_no_costs(((b(i)+1):(b(i+1)-
1))));
    else
        st_Q_ret_no_costs(i) =
std(price_apartments.Q_ret_no_costs((b(length(C))+1):height(price_apartments)));
    end
time(i+1,:) = clock;
fprintf('\nTo complete %.f/%.f iterations (unique IDs) it took %.2f seconds\n', i,
length(C), etime(time(i+1,:), time(1,:)));
end

% Standard deviation of all quarterly returns
std_Q_ret_no_costs = mean(st_Q_ret_no_costs);

% Import stats for 'Q_ret_no_costs' in summary table
sum_stats_apartments.House_Indices(1) = 'Q_ret_no_costs';
sum_stats_apartments.Return(1)        = mean_Q_ret_no_costs;
sum_stats_apartments.std_dev(1)       = std_Q_ret_no_costs;
sum_stats_apartments.Excess_return(1) = mean_excess_Q_ret_no_costs;
sum_stats_apartments.Sharpe_ratio(1)  =
sum_stats_apartments.Excess_return(1)/sum_stats_apartments.std_dev(1);
sum_stats_apartments.Skewness(1)      =
skewness(priceindex_apartments.Q_ret_no_costs(2:44));
sum_stats_apartments.Kurtosis(1)      =
kurtosis(priceindex_apartments.Q_ret_no_costs(2:44));

% COMPUTE CUMULATIVE ROA(NC)
first_obs_price = price_apartments.estimated_price(1:44:end);
price_apartments.first_obs_price_repeated = repelem(first_obs_price, 44);

price_apartments.cum_Q_ret_no_costs =
(price_apartments.estimated_price./price_apartments.first_obs_price_repeated)-1;
price_apartments.cum_Q_ret_no_costs(1:44:end) = NaN;

priceindex_apartments.cum_Q_ret_no_costs = zeros(height(priceindex_apartments),end);
% Compute mean cumulative ROA(NC)
for i = 1:height(priceindex_apartments)
    priceindex_apartments.cum_Q_ret_no_costs(i) =
nanmean(price_apartments.cum_Q_ret_no_costs(find(price_apartments.Q ==
priceindex_apartments.Q_order(i))));
end

% 8. COMPUTATION OF QUARTERLY RETURNS ON ASSET/HOUSE FROM PERIOD TO PERIOD (BUY/SELL),
INCL TOTAL COSTS
% Compute quarterly returns for each unique ID
price_apartments.Q_ret_tot_costs = (((price_apartments.estimated_price -
price_apartments.t_cost_s - price_apartments.lagged_t_cost -
price_apartments.lagged_estimated_price).*(1-price_apartments.tax_rates)) -
price_apartments.living_exp +
price_apartments.implicit_rent)./(price_apartments.lagged_estimated_price);
price_apartments.Q_ret_tot_costs(1:44:end) = NaN;

% Compute mean quarterly returns for houses (mean of all IDs per quarter) (inc tot
costs)
```

```
priceindex_apartments.Q_ret_tot_costs = zeros(height(priceindex_apartments), end);

for i = 1:length(priceindex_apartments.mean_prices)
    priceindex_apartments.Q_ret_tot_costs(i) =
mean(price_apartments.Q_ret_tot_costs(find(price_apartments.Q ==
priceindex_apartments.Q_order(i))));
end

% Compute mean of mean quarterly returns for houses (inc tot costs)
mean_Q_ret_tot_costs = nanmean(priceindex_apartments.Q_ret_tot_costs);

% Compute excess quarterly returns for houses (inc tot costs)
priceindex_apartments.excess_Q_ret_tot_costs = priceindex_apartments.Q_ret_tot_costs-
priceindex_apartments.rf;

% Compute mean excess quarterly returns for houses (inc tot costs)
mean_excess_Q_ret_tot_costs = nanmean(priceindex_apartments.excess_Q_ret_tot_costs);

% Compute standard deviation for each unique IDs returns
st_Q_ret_tot_costs= zeros(length(C),1);
time(1,:) = clock;
for i = 1:length(C)
    if b(i)<b(length(C))
       st_Q_ret_tot_costs(i) = std(price_apartments.Q_ret_tot_costs(((b(i)+1):(b(i+1)-
1))));
    else
       st_Q_ret_tot_costs(i) =
std(price_apartments.Q_ret_tot_costs((b(length(C))+1):height(price_apartments)));
    end
time(i+1,:) = clock;
fprintf('\nTo complete %.f/%.f iterations (unique IDs) it took %.2f seconds\n', i,
length(C), etime(time(i+1,:), time(1,:)));
end

% Standard deviation of all quarterly returns
std_Q_ret_tot_costs = mean(st_Q_ret_tot_costs);

% Import stats for 'Q_ret_tot_costs' in summary table
sum_stats_apartments.House_Indices(2) = 'Q_ret_tot_costs';
sum_stats_apartments.Return(2)        = mean_Q_ret_tot_costs;
sum_stats_apartments.std_dev(2)       = std_Q_ret_tot_costs;
sum_stats_apartments.Excess_return(2) = mean_excess_Q_ret_tot_costs;
sum_stats_apartments.Sharpe_ratio(2) =
sum_stats_apartments.Excess_return(2)/sum_stats_apartments.std_dev(2);
sum_stats_apartments.Skewness(2)      =
skewness(priceindex_apartments.Q_ret_tot_costs(2:44));
sum_stats_apartments.Kurtosis(2)      =
kurtosis(priceindex_apartments.Q_ret_tot_costs(2:44));

% COMPUTE CUMULATIVE ROA (TC)
taxes_TC = (price_apartments.estimated_price - price_apartments.t_cost_s -
price_apartments.lagged_t_cost -
price_apartments.lagged_estimated_price).*price_apartments.tax_rates;
taxes_TC(1:44:end) = NaN;

net_costs_TC = (price_apartments.living_exp + price_apartments.lagged_t_cost +
price_apartments.t_cost_s + taxes_TC - price_apartments.implicit_rent);
net_costs_TC(1:44:end) = NaN;

% Compute cumulative costs
cumcosts_TC = zeros(height(price_apartments), end);
time(1,:) = clock;
for i = 1:length(C)
    if b(i)<b(length(C))
       cumcosts_TC((b(i)):(b(i+1)-1)) = cumsum(net_costs_TC((b(i)):(b(i+1)-
1)),'omitnan');
    else
       cumcosts_TC(b(length(C)):height(price_apartments)) =
cumsum(net_costs_TC(b(length(C)):height(price_apartments)),'omitnan');
    end
time(i+1,:) = clock;
fprintf('\nTo complete %.f/%.f iterations (unique IDs) it took %.2f seconds\n', i,
length(C), etime(time(i+1,:), time(1,:)));
end
```

```
price_apartments.cum_Q_ret_tot_costs = (price_apartments.estimated_price - cumcosts_TC
-
price_apartments.first_obs_price_repeated)./price_apartments.first_obs_price_repeated;
price_apartments.cum_Q_ret_tot_costs(1:44:end) = NaN;

priceindex_apartments.cum_Q_ret_tot_costs = zeros(height(priceindex_apartments),end);
% Compute mean cumulative returns for houses (mean of all IDs per quarter) (no costs)
for i = 1:height(priceindex_apartments)
    priceindex_apartments.cum_Q_ret_tot_costs(i) =
nanmean(price_apartments.cum_Q_ret_tot_costs(find(price_apartments.Q ==
priceindex_apartments.Q_order(i))));
end

all_lost = priceindex_apartments.cum_Q_ret_tot_costs < -1;
priceindex_apartments.cum_Q_ret_tot_costs(all_lost) = -1;

% 9. COMPUTATION OF QUARTERLY RETURNS ON ASSET/HOUSE FROM PERIOD TO PERIOD , INCL
SMOOTHED COSTS
% Locate buyer and sellers cost when buying in 2010Q1 and selling in 2020Q4
% Find buyer and seller cost per quarter for sample period
cost_2010Q1 = price_apartments.t_cost(1:44:end);
cost_2010Q1 = cost_2010Q1/44;
price_apartments.cost_2010Q1 = repelem(cost_2010Q1, 44);

cost_2020Q4 = price_apartments.t_cost_s(44:44:end);
cost_2020Q4 = cost_2020Q4/44;
price_apartments.cost_2020Q4 = repelem(cost_2020Q4, 44);

% Compute quarterly returns given that you buy in 2010Q1 and sell in 2020Q4
% and spred the relevant costs on the sample period.
price_apartments.Q_ret_smooth_costs = (price_apartments.estimated_price -
price_apartments.cost_2010Q1 - price_apartments.cost_2020Q4 -
price_apartments.living_exp - price_apartments.lagged_estimated_price +
price_apartments.implicit_rent)./price_apartments.lagged_estimated_price;
price_apartments.Q_ret_smooth_costs(1:44:end) = NaN;

% Compute mean quarterly returns for houses (mean of all IDs per quarter) (inc smooth
costs)
priceindex_apartments.Q_ret_smooth_costs = zeros(height(priceindex_apartments), end);

for i = 1:length(priceindex_apartments.mean_prices)
    priceindex_apartments.Q_ret_smooth_costs(i) =
mean(price_apartments.Q_ret_smooth_costs(find(price_apartments.Q ==
priceindex_apartments.Q_order(i))));
end

% Compute mean of mean quarterly returns for houses (inc smooth costs)
mean_Q_ret_smooth_costs = nanmean(priceindex_apartments.Q_ret_smooth_costs);

% Compute excess quarterly returns for houses (inc smooth costs)
priceindex_apartments.excess_Q_ret_smooth_costs =
priceindex_apartments.Q_ret_smooth_costs - priceindex_apartments.rf;

% Compute mean excess quarterly returns for houses (inc smooth costs)
mean_excess_Q_ret_smooth_costs =
nanmean(priceindex_apartments.excess_Q_ret_smooth_costs);

% Compute standard deviation for each unique IDs returns
st_Q_ret_smooth_costs= zeros(length(C),1);
time(1,:) = clock;
for i = 1:length(C)
    if b(i)<b(length(C))
       st_Q_ret_smooth_costs(i) =
std(price_apartments.Q_ret_smooth_costs(((b(i)+1):(b(i+1)-1))));
    else
       st_Q_ret_smooth_costs(i) =
std(price_apartments.Q_ret_smooth_costs((b(length(C))+1):height(price_apartments)));
    end
time(i+1,:) = clock;
fprintf('\nTo complete %.f/%.f iterations (unique IDs) it took %.2f seconds\n', i,
length(C), etime(time(i+1,:), time(1,:)));
end
```

```
% Standard deviation of all quarterly returns
std_Q_ret_smooth_costs = mean(st_Q_ret_smooth_costs);

% Import stats for 'Q_ret_smooth_costs' in summary table
sum_stats_apartments.House_Indices(3) = 'Q_ret_smooth_costs';
sum_stats_apartments.Return(3)        = mean_Q_ret_smooth_costs;
sum_stats_apartments.std_dev(3)       = std_Q_ret_smooth_costs;
sum_stats_apartments.Excess_return(3) = mean_excess_Q_ret_smooth_costs;
sum_stats_apartments.Sharpe_ratio(3)  =
sum_stats_apartments.Excess_return(3)/sum_stats_apartments.std_dev(3);
sum_stats_apartments.Skewness(3)      =
skewness(priceindex_apartments.Q_ret_smooth_costs(2:44));
sum_stats_apartments.Kurtosis(3)      =
kurtosis(priceindex_apartments.Q_ret_smooth_costs(2:44));

% COMPUTE CUMULATIVE ROA(SC)
net_costs_SC = (price_apartments.living_exp + price_apartments.cost_2010Q1 +
price_apartments.cost_2020Q4 - price_apartments.implicit_rent);
net_costs_SC(1:44:end) = NaN;

cumcosts_SC = zeros(height(price_apartments), end);
time(1,:) = clock;
for i = 1:length(C)
    if b(i)<b(length(C))
       cumcosts_SC((b(i)):(b(i+1)-1)) = cumsum(net_costs_SC((b(i)):(b(i+1)-
1)),'omitnan');
    else
       cumcosts_SC(b(length(C)):height(price_apartments)) =
cumsum(net_costs_SC(b(length(C)):height(price_apartments)),'omitnan');
    end
time(i+1,:) = clock;
fprintf('\nTo complete %.f/%.f iterations (unique IDs) it took %.2f seconds\n', i,
length(C), etime(time(i+1,:), time(1,:)));
end

price_apartments.cum_Q_ret_smooth_costs = (price_apartments.estimated_price -
cumcosts_SC -
price_apartments.first_obs_price_repeated)./price_apartments.first_obs_price_repeated;
price_apartments.cum_Q_ret_smooth_costs(1:44:end) = NaN;

% Compute mean cumulative returns for houses (mean of all IDs per quarter) (no costs)
priceindex_apartments.cum_Q_ret_smooth_costs =
zeros(height(priceindex_apartments),end);
for i = 1:height(priceindex_apartments)
    priceindex_apartments.cum_Q_ret_smooth_costs(i) =
nanmean(price_apartments.cum_Q_ret_smooth_costs(find(price_apartments.Q ==
priceindex_apartments.Q_order(i))));
end

% Identify average loan-to-value ratio for housing investments
l = ([0.6957, 0.6957, 0.6957, 0.6957, 0.71, 0.76, 0.67, 0.66, 0.68, 0.70, 0.69])';
n = 4;
ltv_ratio_order =
[repmat(l(1),n,1);repmat(l(2),n,1);repmat(l(3),n,1);repmat(l(4),n,1);repmat(l(5),n,1);
repmat(l(6),n,1);repmat(l(7),n,1);repmat(l(8),n,1);repmat(l(9),n,1);repmat(l(10),n,1);
repmat(l(11),n,1)];

% Allocate correct ltv-rates to each quarter in price_houses set
price_apartments.ltvr        = repmat(ltv_ratio_order,(length(C)),1);
price_apartments.lagged_ltvr = lagmatrix(price_apartments.ltvr,1);

% Allocate ltv ratio as of 2010Q1 to each quarter in price_houses set
price_apartments.ltvr_2010Q1(:) = price_apartments.ltvr(1);

% 10. WE DONT USE THE RETURN SERIES FROM SECTION 10
% Compute quarterly equity returns for each unique ID (no costs)
price_apartments.Q_eq_ret_no_costs = ((price_apartments.estimated_price -
price_apartments.lagged_estimated_price)./(price_apartments.lagged_estimated_price.*(1
-price_apartments.lagged_ltvr)));
price_apartments.Q_eq_ret_no_costs(1:44:end) = NaN;

% Compute mean quarterly equity returns for houses (mean of all IDs per quarter) (no
costs)
priceindex_apartments.Q_eq_ret_no_costs = zeros(height(priceindex_apartments), end);
```

```
for i = 1:height(priceindex_apartments)
    priceindex_apartments.Q_eq_ret_no_costs(i) =
mean(price_apartments.Q_eq_ret_no_costs(find(price_apartments.Q ==
priceindex_apartments.Q_order(i))));
end

% Compute mean of mean quarterly equity returns for houses (no costs)
mean_Q_eq_ret_no_costs = nanmean(priceindex_apartments.Q_eq_ret_no_costs);

% Compute excess quarterly equity returns for houses (no costs)
priceindex_apartments.excess_Q_eq_ret_no_costs =
priceindex_apartments.Q_eq_ret_no_costs - priceindex_apartments.rf;

% Compute mean excess quarterly equity returns for houses (no costs)
mean_excess_Q_eq_ret_no_costs =
nanmean(priceindex_apartments.excess_Q_eq_ret_no_costs);

% Compute standard deviation for each unique IDs returns
st_Q_eq_ret_no_costs= zeros(length(C),1);
time(1,:) = clock;
for i = 1:length(C)
    if b(i)<b(length(C))
       st_Q_eq_ret_no_costs(i) =
std(price_apartments.Q_eq_ret_no_costs(((b(i)+1):(b(i+1)-1))));
    else
       st_Q_eq_ret_no_costs(i) =
std(price_apartments.Q_eq_ret_no_costs((b(length(C))+1):height(price_apartments)));
    end
time(i+1,:) = clock;
fprintf('\nTo complete %.f/%.f iterations (unique IDs) it took %.2f seconds\n', i,
length(C), etime(time(i+1,:), time(1,:)));
end

% Standard deviation of all quarterly equity returns
std_Q_eq_ret_no_costs = mean(st_Q_eq_ret_no_costs);

% Import stats for 'Q_eq_ret_no_costs' in summary table
sum_stats_apartments.House_Indices(4) = 'Q_eq_ret_no_costs';
sum_stats_apartments.Return(4)        = mean_Q_eq_ret_no_costs;
sum_stats_apartments.std_dev(4)       = std_Q_eq_ret_no_costs;
sum_stats_apartments.Excess_return(4) = mean_excess_Q_eq_ret_no_costs;
sum_stats_apartments.Sharpe_ratio(4)  =
sum_stats_apartments.Excess_return(4)/sum_stats_apartments.std_dev(4);
sum_stats_apartments.Skewness(4)      =
skewness(priceindex_apartments.Q_eq_ret_no_costs(2:44));
sum_stats_apartments.Kurtosis(4)      =
kurtosis(priceindex_apartments.Q_eq_ret_no_costs(2:44));

% COMPUTE CUMULATIVE ROE (NC)
price_apartments.cum_Q_eq_ret_no_costs = ((price_apartments.estimated_price -
price_apartments.first_obs_price_repeated)./(price_apartments.first_obs_price_repeated
.*(1-price_apartments.ltvr_2010Q1)));
price_apartments.cum_Q_eq_ret_no_costs(1:44:end) = NaN;

% Compute mean cumulative returns for houses (mean of all IDs per quarter) (no costs)
priceindex_apartments.cum_Q_eq_ret_no_costs =
zeros(height(priceindex_apartments),end);
for i = 1:height(priceindex_apartments)
    priceindex_apartments.cum_Q_eq_ret_no_costs(i) =
nanmean(price_apartments.cum_Q_eq_ret_no_costs(find(price_apartments.Q ==
priceindex_apartments.Q_order(i))));
end

% 11. WE DONT USE THE RETURN SERIES FROM SECTION 11
% Compute quarterly equity returns for each unique ID (inc tot costs)
price_apartments.Q_eq_ret_tot_costs = (((price_apartments.estimated_price -
price_apartments.t_cost_s - price_apartments.lagged_t_cost -
price_apartments.lagged_estimated_price).*(1-price_apartments.tax_rates)) -
price_apartments.living_exp +
price_apartments.implicit_rent)./(price_apartments.lagged_estimated_price.*(1-
price_apartments.lagged_ltvr));
price_apartments.Q_eq_ret_tot_costs(1:44:end) = NaN;
```

```
% Compute mean quarterly equity returns for houses (mean of all IDs per quarter) (inc
tot costs)
priceindex_apartments.Q_eq_ret_tot_costs = zeros(height(priceindex_apartments), end);

for i = 1:length(priceindex_apartments.mean_prices)
    priceindex_apartments.Q_eq_ret_tot_costs(i) =
mean(price_apartments.Q_eq_ret_tot_costs(find(price_apartments.Q ==
priceindex_apartments.Q_order(i))));
end

% Compute mean of mean quarterly equity returns for houses (inc tot costs)
mean_Q_eq_ret_tot_costs = nanmean(priceindex_apartments.Q_eq_ret_tot_costs);

% Compute excess quarterly equity returns for houses (inc tot costs)
priceindex_apartments.excess_Q_eq_ret_tot_costs =
priceindex_apartments.Q_eq_ret_tot_costs - priceindex_apartments.rf;

% Compute mean excess quarterly equity returns for houses (inc tot costs)
mean_excess_Q_eq_ret_tot_costs =
nanmean(priceindex_apartments.excess_Q_eq_ret_tot_costs);

% Compute standard deviation for each unique IDs returns
st_Q_eq_ret_tot_costs= zeros(length(C),1);
time(1,:) = clock;
for i = 1:length(C)
    if b(i)<b(length(C))
       st_Q_eq_ret_tot_costs(i) =
std(price_apartments.Q_eq_ret_tot_costs(((b(i)+1):(b(i+1)-1))));
    else
       st_Q_eq_ret_tot_costs(i) =
std(price_apartments.Q_eq_ret_tot_costs((b(length(C))+1):height(price_apartments)));
    end
time(i+1,:) = clock;
fprintf('\nTo complete %.f/%.f iterations (unique IDs) it took %.2f seconds\n', i,
length(C), etime(time(i+1,:), time(1,:)));
end

% Standard deviation of all quarterly equity returns
std_Q_eq_ret_tot_costs = mean(st_Q_eq_ret_tot_costs);

% Import stats for 'Q_eq_ret_tot_costs' in summary table
sum_stats_apartments.House_Indices(5) = 'Q_eq_ret_tot_costs';
sum_stats_apartments.Return(5)        = mean_Q_eq_ret_tot_costs;
sum_stats_apartments.std_dev(5)       = std_Q_eq_ret_tot_costs;
sum_stats_apartments.Excess_return(5) = mean_excess_Q_eq_ret_tot_costs;
sum_stats_apartments.Sharpe_ratio(5)  =
sum_stats_apartments.Excess_return(5)/sum_stats_apartments.std_dev(5);
sum_stats_apartments.Skewness(5)      =
skewness(priceindex_apartments.Q_eq_ret_tot_costs(2:44));
sum_stats_apartments.Kurtosis(5)      =
kurtosis(priceindex_apartments.Q_eq_ret_tot_costs(2:44));

% COMPUTE CUMULATIVE RETURNS BASED ON QUARTERLY EQUITY RETURNS FROM PRICE INDEX (INC
TOT COSTS)
priceindex_apartments.cum_Q_eq_ret_tot_costs =
zeros(height(priceindex_apartments),end);
priceindex_apartments.cum_Q_eq_ret_tot_costs =
cumprod((1+priceindex_apartments.Q_eq_ret_tot_costs),'omitnan') -1;

% 12. COMPUTATION OF QUARTERLY EQUITY RETURNS ON ASSET/HOUSE FROM PERIOD TO PERIOD ,
INCL SMOOTHED COSTS
% Compute quarterly EQUITY returns given that you buy in 2010Q1 and sell in 2020Q4
% and spread the relevant costs on the sample period.
price_apartments.Q_eq_ret_smooth_costs = (price_apartments.estimated_price -
price_apartments.cost_2010Q1 - price_apartments.cost_2020Q4 -
price_apartments.living_exp - price_apartments.lagged_estimated_price +
price_apartments.implicit_rent)./(price_apartments.lagged_estimated_price.*(1-
price_apartments.ltvr_2010Q1));
price_apartments.Q_eq_ret_smooth_costs(1:44:end) = NaN;

% Compute mean quarterly equity returns for houses (mean of all IDs per quarter) (inc
smooth costs)
priceindex_apartments.Q_eq_ret_smooth_costs = zeros(height(priceindex_apartments),
end);
```

```
for i = 1:length(priceindex_apartments.mean_prices)
    priceindex_apartments.Q_eq_ret_smooth_costs(i) =
mean(price_apartments.Q_eq_ret_smooth_costs(find(price_apartments.Q ==
priceindex_apartments.Q_order(i))));
end

% Compute mean of mean quarterly equity returns for houses (inc smooth costs)
mean_Q_eq_ret_smooth_costs = nanmean(priceindex_apartments.Q_eq_ret_smooth_costs);

% Compute excess quarterly equity returns for houses (inc smooth costs)
priceindex_apartments.excess_Q_eq_ret_smooth_costs =
priceindex_apartments.Q_eq_ret_smooth_costs - priceindex_apartments.rf;

% Compute mean excess quarterly equity returns for houses (inc smooth costs)
mean_excess_Q_eq_ret_smooth_costs =
nanmean(priceindex_apartments.excess_Q_eq_ret_smooth_costs);

% Compute standard deviation for each unique IDs returns
st_Q_eq_ret_smooth_costs= zeros(length(C),1);
time(1,:) = clock;
for i = 1:length(C)
    if b(i)<b(length(C))
       st_Q_eq_ret_smooth_costs(i) =
std(price_apartments.Q_eq_ret_smooth_costs(((b(i)+1):(b(i+1)-1))));
    else
       st_Q_eq_ret_smooth_costs(i) =
std(price_apartments.Q_eq_ret_smooth_costs((b(length(C))+1):height(price_apartments)))
;
    end
time(i+1,:) = clock;
fprintf('\nTo complete %.f/%.f iterations (unique IDs) it took %.2f seconds\n', i,
length(C), etime(time(i+1,:), time(1,:)));
end

% Standard deviation of all quarterly equity returns
std_Q_eq_ret_smooth_costs = mean(st_Q_eq_ret_smooth_costs);

% Import stats for 'Q_eq_ret_smooth_costs' in summary table
sum_stats_apartments.House_Indices(6) = 'Q_eq_ret_smooth_costs';
sum_stats_apartments.Return(6)        = mean_Q_eq_ret_smooth_costs;
sum_stats_apartments.std_dev(6)       = std_Q_eq_ret_smooth_costs;
sum_stats_apartments.Excess_return(6) = mean_excess_Q_eq_ret_smooth_costs;
sum_stats_apartments.Sharpe_ratio(6)  =
sum_stats_apartments.Excess_return(6)/sum_stats_apartments.std_dev(6);
sum_stats_apartments.Skewness(6)      =
skewness(priceindex_apartments.Q_eq_ret_smooth_costs(2:44));
sum_stats_apartments.Kurtosis(6)      =
kurtosis(priceindex_apartments.Q_eq_ret_smooth_costs(2:44));

% COMPUTE CUMULATIVE ROE(SC)
price_apartments.cum_Q_eq_ret_smooth_costs = (price_apartments.estimated_price -
cumcosts_SC -
price_apartments.first_obs_price_repeated)./(price_apartments.first_obs_price_repeated
.*(1-price_apartments.ltvr_2010Q1));
price_apartments.cum_Q_eq_ret_smooth_costs(1:44:end) = NaN;

% Compute mean cumulative returns for houses (mean of all IDs per quarter) (smooth
costs)
priceindex_apartments.cum_Q_eq_ret_smooth_costs =
zeros(height(priceindex_apartments),end);
for i = 1:height(priceindex_apartments)
    priceindex_apartments.cum_Q_eq_ret_smooth_costs(i) =
nanmean(price_apartments.cum_Q_eq_ret_smooth_costs(find(price_apartments.Q ==
priceindex_apartments.Q_order(i))));
end
% MAKE TABLES STORING INDICES AND QUARTELRY CHANGES
apartment_price_changes = priceindex_apartments(:,[1 5 8 11 14 17 20]);

apartment_price_indices = priceindex_apartments(:,[1 7 10 13 16 19 22]);
apartment_price_indices.cum_Q_ret_no_costs(1) = 0;
apartment_price_indices.cum_Q_ret_tot_costs(1) = 0;
apartment_price_indices.cum_Q_ret_smooth_costs(1) = 0;
apartment_price_indices.cum_Q_eq_ret_no_costs(1) = 0;
```

```
apartment_price_indices.cum_Q_eq_ret_tot_costs(1) = 0;
apartment_price_indices.cum_Q_eq_ret_smooth_costs(1) = 0;

apartment_price_indices.cum_Q_ret_no_costs        =
apartment_price_indices.cum_Q_ret_no_costs + 1;
apartment_price_indices.cum_Q_ret_tot_costs       =
apartment_price_indices.cum_Q_ret_tot_costs + 1;
apartment_price_indices.cum_Q_ret_smooth_costs    =
apartment_price_indices.cum_Q_ret_smooth_costs + 1;
apartment_price_indices.cum_Q_eq_ret_no_costs     =
apartment_price_indices.cum_Q_eq_ret_no_costs + 1;
apartment_price_indices.cum_Q_eq_ret_tot_costs    =
apartment_price_indices.cum_Q_eq_ret_tot_costs + 1;
apartment_price_indices.cum_Q_eq_ret_smooth_costs =
apartment_price_indices.cum_Q_eq_ret_smooth_costs + 1;

clearvars -except Property_Data summary_table Rf Rent_Data...
                  house_set hedonic_houses housing_model...
                  price_houses uniq_house_set avg_cost_rates_houses...
                  priceindex_houses sum_stats_houses houseprice_changes...
                  houseprice_indices...
                  smallhouse_set hedonic_smallhouses smallhouse_model...
                  price_smallhouses uniq_smallhouse_set avg_cost_rates_smallhouses...
                  priceindex_smallhouses sum_stats_smallhouses...
                  smallhouse_price_changes smallhouse_price_indices...
                  apartment_set hedonic_apartments apartment_model price_apartments...
                  uniq_apartment_set avg_cost_rates_apartments
priceindex_apartments...
                  sum_stats_apartments apartment_price_changes...
                  apartment_price_indices...
                  inner_eastern_set hedonic_inner_eastern inner_eastern_model...
                  price_inner_eastern uniq_inner_eastern_set
avg_cost_rates_inner_eastern...
                  priceindex_inner_eastern sum_stats_inner_eastern
inner_eastern_price_changes...
                  inner_eastern_price_indices...
                  inner_western_set hedonic_inner_western inner_western_model...
                  price_inner_western uniq_inner_western_set
avg_cost_rates_inner_western...
                  priceindex_inner_western sum_stats_inner_western
inner_western_price_changes...
                  inner_western_price_indices...
                  outer_eastern_set hedonic_outer_eastern outer_eastern_model...
                  price_outer_eastern uniq_outer_eastern_set
avg_cost_rates_outer_eastern...
                  priceindex_outer_eastern sum_stats_outer_eastern
outer_eastern_price_changes...
                  outer_eastern_price_indices...
                  outer_western_set hedonic_outer_western outer_western_model...
                  price_outer_western uniq_outer_western_set
avg_cost_rates_outer_western...
                  priceindex_outer_western sum_stats_outer_western
outer_western_price_changes...
                  outer_western_price_indices...
                  outer_southern_set hedonic_outer_southern outer_southern_model...
                  price_outer_southern uniq_outer_southern_set
avg_cost_rates_outer_southern...
                  priceindex_outer_southern sum_stats_outer_southern
outer_southern_price_changes...
                  outer_southern_price_indices...
                  property_price_changes property_price_indices hedonic_total...
                  ROA_NC_stats ROE_NC_stats ROA_TC_stats ROE_TC_stats...
                  ROA_SC_stats ROE_SC_stats sum_stats_OSEFX OSEFX...
                  ROA_NC_sharpe_test ROA_TC_sharpe_test ROA_SC_sharpe_test...
                  ROE_SC_sharpe_test

%% Regression model for Houses
% 1. PREPERATION OF DATASET
houses    = Property_Data.Boligtype == 'Enebolig';
house_set = Property_Data(houses,:);

house_set.Bydel = categorical(house_set.Bydel);
house_set.Alder = categorical(house_set.Alder);
house_set.Q     = categorical(house_set.Q);
```

```
house_set.Bydel = reordercats(house_set.Bydel, {'Gr√°nerl√∏kka', 'Alna', 'Bjerke',
'Frogner', 'Gamle Oslo', 'Grorud', 'Nordre Aker', 'Nordstrand', 'Sagene', 'Sentrum',
'St. Hanshaugen', 'Stovner', 'S√∏ndre Nordstrand', 'Ullern', 'Vestre Aker',
'√òstensj√∏', 'NULL', 'Marka'});
house_set.Alder = reordercats(house_set.Alder, {'Age1', 'Age2', 'Age3', 'Age4'});
house_set.Bydel = removecats(house_set.Bydel);

house_set.log_pris = log(house_set.Pris);
house_set.log_prom = log(house_set.prom);

categories(house_set.Bydel)
categories(house_set.Alder)

% Estimate regression model - Log model with log(prom), Q, Bydel, Soverom, Alder on
log(pris)
% Explanatory variables log(prom) and Soverom - continous
% Explanatory variables Q, Bydel and Alder - dummies
housing_model = fitlm(house_set,'log_pris~log_prom+Q+Bydel+Soverom+Alder');

% 3. ALLOCATE TABLE FOR ESTIMATED PRICES (BASED ON MODEL FROM PT. 2) FOR EVERY QUARTER
FOR EACH ID
house_set    = sortrows(house_set,'BoligID','ascend');
[C,ia,ic]    = unique(house_set.BoligID,'first');
price_houses = zeros(length(C)*44, width(house_set));
price_houses = array2table(price_houses);

price_houses.Properties.VariableNames = {'BoligID', 'Salgsdato', 'Bydel', 'Pris',
'Fellesgjeld', 'Prisantydning', 'Eierform', 'Byggeaar', 'prom', 'Etasje', 'Soverom',
'Boligtype', 'year', 'Q', 'Alder', 'kvmpris', 'log_pris', 'log_prom' };

Q_rep   = repmat(["Q1"; "Q2"; "Q3";"Q4"],(length(C)*44)/4,1);
x       = (2010:2020)';
n       = 4;
newx    =
[repmat(x(1),n,1);repmat(x(2),n,1);repmat(x(3),n,1);repmat(x(4),n,1);repmat(x(5),n,1);
repmat(x(6),n,1);repmat(x(7),n,1);repmat(x(8),n,1);repmat(x(9),n,1);repmat(x(10),n,1);
repmat(x(11),n,1)];
Y_order = repmat(newx,(length(C)),1);

price_houses.Q_order   = string(Y_order(:,1)) + Q_rep(:,1);
price_houses.Q         = price_houses.Q_order;
price_houses.Q_order   = [];
price_houses.BoligID   = categorical(price_houses.BoligID);
price_houses.BoligID   = repelem(C, 44);
price_houses           = convertvars(price_houses,{'BoligID', 'Bydel', 'Eierform',
'Boligtype', 'Alder', 'Q'},'categorical');
price_houses.Salgsdato = datetime(price_houses.Salgsdato,'ConvertFrom','yyyymmdd');

b = 1:length(newx):height(price_houses);

% Create set containing only the unique observations of house_set. When
% multiple unique values it stores the first observation
uniq_house_set = house_set(ia,:);

% Allocate variables into price_houses to use for regression model
price_houses(:,[3 5 7 9 11 15 18]) = repelem(uniq_house_set(:,[3 5 7 9 11 15 18]), 44,
1);

% Store relevant variables for regression model in new table
x3 = price_houses(:,[18 14 3 11 15]);

% GET COLUMN OF ESTIMATED PRICES BASED ON REGRESSION MODEL
price_houses.log_estimated_price    = predict(housing_model, x3);
price_houses.estimated_price        = exp(price_houses.log_estimated_price);
price_houses.lagged_estimated_price = lagmatrix(price_houses.estimated_price,1);

% Store risk-free rates in price_houses
n = 4;
rf_ = zeros(44,1);
rf_ =
[repmat(Rf.rf(1),n,1);repmat(Rf.rf(2),n,1);repmat(Rf.rf(3),n,1);repmat(Rf.rf(4),n,1);r
epmat(Rf.rf(5),n,1);repmat(Rf.rf(6),n,1);repmat(Rf.rf(7),n,1);repmat(Rf.rf(8),n,1);rep
mat(Rf.rf(9),n,1);repmat(Rf.rf(10),n,1);repmat(Rf.rf(11),n,1)];
```

```
rf_ = rf_./4;
price_houses.rf = repmat(rf_, length(C),1);

% 4. IMPLEMENT RELEVANT COSTS RELATED TO HOUSING INVESTMENTS
% Transaction costs buyer, b
gebyr_b             = zeros(height(price_houses),1);
dokavg_b            = zeros(height(price_houses),1);
insurance_b         = zeros(height(price_houses),1);
price_houses.t_cost = zeros(height(price_houses),1);

% If type of ownership = 'Selveier'
idx_selv              = price_houses.Eierform == "Selveier";
gebyr_b(idx_selv)     = 540 + 540 + 172;
dokavg_b(idx_selv)    =
0.025.*(price_houses.estimated_price(idx_selv)+price_houses.Fellesgjeld(idx_selv));
insurance_b(idx_selv) = 11500;

% If type of ownership = 'Borettslag'
idx_andel             = price_houses.Eierform == "Borettslag";
gebyr_b(idx_andel)    = 440 + 440 + 172;
dokavg_b(idx_andel)   = 0;
insurance_b(idx_andel) = 4100;

% Compute total transaction cost for buyer, b, for each observation/quarter
price_houses.t_cost        = gebyr_b+dokavg_b+insurance_b;
price_houses.lagged_t_cost = lagmatrix(price_houses.t_cost,1);

% Transaction costs seller, s
eierskiftegebyr_s     = zeros(height(price_houses),1);
public_fees_s         = zeros(height(price_houses),1);
insurance_s           = zeros(height(price_houses),1);
commission_s          = zeros(height(price_houses),1);
price_houses.t_cost_s = zeros(height(price_houses),1);
indices_selv          = zeros(height(price_houses),1);
indices_andel         = zeros(height(price_houses),1);

% If type of ownership = 'Selveier'
idx_selv                    = price_houses.Eierform == "Selveier";
public_fees_s(idx_selv)     = 4930;
eierskiftegebyr_s(idx_selv) = 0;
insurance_s(idx_selv)       =
max(0.005*(price_houses.estimated_price(idx_selv)+price_houses.Fellesgjeld(idx_selv)),
10250);
indices_selv                = insurance_s > 48750;
indices_selv                = logical(indices_selv);
insurance_s(indices_selv)   = 48750;

% If type of ownership = 'Borettslag'
idx_andel                   = price_houses.Eierform == "Borettslag";
public_fees_s(idx_andel)    = 10819;
eierskiftegebyr_s(idx_andel) = 4858;
insurance_s(idx_andel)      =
max(0.002*(price_houses.estimated_price(idx_andel)+price_houses.Fellesgjeld(idx_andel)
), 2500);
indices_andel               = zeros(height(price_houses),1);
indices_andel(idx_andel)    = insurance_s(idx_andel) > 24000;
indices_andel               = logical(indices_andel);
insurance_s(indices_andel)  = 24000;

% Commission to broker independent of type of ownership
commission_s =
max(0.02*(price_houses.estimated_price+price_houses.Fellesgjeld),40000);

% Compute total transaction cost for seller, s, for each observation/quarter
price_houses.t_cost_s = public_fees_s+eierskiftegebyr_s+insurance_s+commission_s;

% Compute transaction cost rates for buyer and seller
% For buyer
t_cost_rate_houses   = mean(price_houses.t_cost./price_houses.estimated_price);
t_cost_rate_Q = t_cost_rate_houses/4;

% For seller
t_cost_s_rate   = mean(price_houses.t_cost_s./price_houses.estimated_price);
t_cost_s_rate_Q = t_cost_s_rate/4;
```

```
% Define holding period for each dwelling
holding_period = 11;

% Define living expenses per quarter - reference house is 120 sqm
liv_exp_h_120_Q = ([125463/4, 130884/4, 134339/4, 138418/4, 137016/4, 129402/4,
127842/4, 136811/4, 146624/4, 165266/4, 142411/4])';

% Allocate correct liwing expenses to each quarter in price_houses set
n = 4;
liv_exp_h =
[repmat(liv_exp_h_120_Q(1),n,1);repmat(liv_exp_h_120_Q(2),n,1);repmat(liv_exp_h_120_Q(
3),n,1);repmat(liv_exp_h_120_Q(4),n,1);repmat(liv_exp_h_120_Q(5),n,1);repmat(liv_exp_h
_120_Q(6),n,1);repmat(liv_exp_h_120_Q(7),n,1);repmat(liv_exp_h_120_Q(8),n,1);repmat(li
v_exp_h_120_Q(9),n,1);repmat(liv_exp_h_120_Q(10),n,1);repmat(liv_exp_h_120_Q(11),n,1)]
;
price_houses.ordered_liv_exp_120 = repmat(liv_exp_h,(length(C)),1);

% Compute living expenses based on a reference house of 120 sqm
price_houses.living_exp = (price_houses.prom./120).*price_houses.ordered_liv_exp_120;
avg_living_exp_Q = mean(price_houses.living_exp./price_houses.estimated_price);

% Allocate correct implicit rents
implicit_rent     = zeros(height(price_houses),1);
rental_allocation = repmat(Rent_Data, length(C), 1);

implicit_rent(find(price_houses.Soverom == 0)) =
rental_allocation.room1(find(price_houses.Soverom == 0));
implicit_rent(find(price_houses.Soverom == 1)) =
rental_allocation.room1(find(price_houses.Soverom == 1));
implicit_rent(find(price_houses.Soverom == 2)) =
rental_allocation.room2(find(price_houses.Soverom == 2));
implicit_rent(find(price_houses.Soverom == 3)) =
rental_allocation.room3(find(price_houses.Soverom == 3));
implicit_rent(find(price_houses.Soverom == 4)) =
rental_allocation.room4(find(price_houses.Soverom == 4));
implicit_rent(find(price_houses.Soverom >= 5)) =
rental_allocation.room5_(find(price_houses.Soverom >= 5));

price_houses.implicit_rent = implicit_rent;
avg_implicit_rent_Q = mean(price_houses.implicit_rent./price_houses.estimated_price);

% Implement tax rates for sample period
tax_rates = ([ 0.28, 0.28, 0.28, 0.28, 0.27, 0.27, 0.25, 0.24, 0.23, 0.22, 0.22])';
n = 4;
tax_rates_order =
[repmat(tax_rates(1),n,1);repmat(tax_rates(2),n,1);repmat(tax_rates(3),n,1);repmat(tax
_rates(4),n,1);repmat(tax_rates(5),n,1);repmat(tax_rates(6),n,1);repmat(tax_rates(7),n
,1);repmat(tax_rates(8),n,1);repmat(tax_rates(9),n,1);repmat(tax_rates(10),n,1);repmat
(tax_rates(11),n,1)];

% Allocate correct tax rates to each quarter in price_houses set
price_houses.tax_rates = repmat(tax_rates_order,(length(C)),1);

% Compute transaction costs and living expenses for different ownership types (houses)
SOUs_houses           = price_houses((price_houses.Eierform == 'Selveier'),:);
t_cost_rate_SOUs      = mean(SOUs_houses.t_cost./SOUs_houses.estimated_price);
SOU_share_houses      = height(SOUs_houses)/height(price_houses);

co_ops_houses         = price_houses((price_houses.Eierform == 'Borettslag'),:);
t_cost_rate_co_ops    = mean(co_ops_houses.t_cost./co_ops_houses.estimated_price);
Co_ops_share_houses   = height(co_ops_houses)/height(price_houses);

t_cost_rate_s_SOUs    = mean(SOUs_houses.t_cost_s./SOUs_houses.estimated_price);
t_cost_rate_s_co_ops  = mean(co_ops_houses.t_cost_s./co_ops_houses.estimated_price);

avg_living_exp_SOUs_Q   = mean(SOUs_houses.living_exp./SOUs_houses.estimated_price);
avg_living_exp_co_ops_Q =
mean(co_ops_houses.living_exp./co_ops_houses.estimated_price);

% Create a table to store different cost rates
avg_cost_rates_houses                          = zeros(4,5); %JUSTERE DENNE ETTERSOM
HVORDAN ST√òRRELSEN VIL BLI
avg_cost_rates_houses                          = array2table(avg_cost_rates_houses);
```

```
avg_cost_rates_houses.Properties.VariableNames = {'Average_cost_rates', 'Yearly',
'Quarterly', 'SOUs_Y', 'Co_ops_Y'};
avg_cost_rates_houses.Average_cost_rates       =
categorical(avg_cost_rates_houses.Average_cost_rates);

% Import stats for cost rates
avg_cost_rates_houses.Average_cost_rates(1) = 't_cost_buyer';
avg_cost_rates_houses.Yearly(1)             = t_cost_rate_houses;
avg_cost_rates_houses.Quarterly(1)          = t_cost_rate_Q;
avg_cost_rates_houses.SOUs_Y(1)             = t_cost_rate_SOUs;
avg_cost_rates_houses.Co_ops_Y(1)           = t_cost_rate_co_ops;

avg_cost_rates_houses.Average_cost_rates(2) = 't_cost_seller';
avg_cost_rates_houses.Yearly(2)             = t_cost_s_rate;
avg_cost_rates_houses.Quarterly(2)          = t_cost_s_rate_Q;
avg_cost_rates_houses.SOUs_Y(2)             = t_cost_rate_s_SOUs;
avg_cost_rates_houses.Co_ops_Y(2)           = t_cost_rate_s_co_ops;

avg_cost_rates_houses.Average_cost_rates(3) = 'living_expenses';
avg_cost_rates_houses.Yearly(3)             = avg_living_exp_Q * 4;
avg_cost_rates_houses.Quarterly(3)          = avg_living_exp_Q;
avg_cost_rates_houses.SOUs_Y(3)             = avg_living_exp_SOUs_Q * 4;
avg_cost_rates_houses.Co_ops_Y(3)           = avg_living_exp_co_ops_Q * 4;

avg_cost_rates_houses.Average_cost_rates(4) = 'implicit_rent';
avg_cost_rates_houses.Yearly(4)             = avg_implicit_rent_Q * 4;
avg_cost_rates_houses.Quarterly(4)          = avg_implicit_rent_Q;

% 5. ALLOCATE TABLE FOR PRICE INDICES
priceindex_houses         = table();
Y_index                   = Y_order(1:44);
Q_index                   = Q_rep(1:44);
priceindex_houses.Q_order = string(Y_index(:,1)) + Q_index(:,1);
priceindex_houses.Q_order = categorical(priceindex_houses.Q_order);

% Create a table to store returns, std, risk free, sharpe
sum_stats_houses                          = zeros(10,7);
sum_stats_houses                          = array2table(sum_stats_houses);
sum_stats_houses.Properties.VariableNames = {'House_Indices', 'Return',
'std_dev','Excess_return','Sharpe_ratio', 'Skewness', 'Kurtosis'};
sum_stats_houses.House_Indices            =
categorical(sum_stats_houses.House_Indices);

% Import risk-free rates (innskuddsrente)
n = 4;
priceindex_houses.rf = zeros(44,1);
priceindex_houses.rf =
[repmat(Rf.rf(1),n,1);repmat(Rf.rf(2),n,1);repmat(Rf.rf(3),n,1);repmat(Rf.rf(4),n,1);r
epmat(Rf.rf(5),n,1);repmat(Rf.rf(6),n,1);repmat(Rf.rf(7),n,1);repmat(Rf.rf(8),n,1);rep
mat(Rf.rf(9),n,1);repmat(Rf.rf(10),n,1);repmat(Rf.rf(11),n,1)];
priceindex_houses.rf = priceindex_houses.rf./4;

% 6. COMPUTATION OF MEAN PRICES FOR HOUSES AND PRICE INDEX BASED ON MEAN PRICES
% Calculate mean prices for houses
priceindex_houses.mean_prices = zeros(height(priceindex_houses), end);
for i = 1:length(priceindex_houses.Q_order)
    priceindex_houses.mean_prices(i) =
mean(price_houses.estimated_price(find(price_houses.Q ==
priceindex_houses.Q_order(i))));
end

% Calculate price index for houses based on mean prices
priceindex_houses.index_values = zeros(height(priceindex_houses), end);
for i = 1:length(priceindex_houses.index_values)
    priceindex_houses.index_values(i) =
priceindex_houses.mean_prices(i)/priceindex_houses.mean_prices(1);
end

% 7. COMPUTATION OF QUARTERLY RETURNS ON ASSET/HOUSE FROM PERIOD TO PERIOD, NO COSTS
% Compute quarterly returns for each unique ID
price_houses.Q_ret_no_costs =
(price_houses.estimated_price./price_houses.lagged_estimated_price)-1;
price_houses.Q_ret_no_costs(1:44:end) = NaN;
```

```
% Compute excess returns for each unique ID
price_houses.excess_Q_ret_no_costs = price_houses.Q_ret_no_costs - price_houses.rf;

% Compute mean quarterly returns for houses (mean of all IDs per quarter) (no costs)
priceindex_houses.Q_ret_no_costs = zeros(height(priceindex_houses), end);

for i = 1:height(priceindex_houses)
    priceindex_houses.Q_ret_no_costs(i) =
nanmean(price_houses.Q_ret_no_costs(find(price_houses.Q ==
priceindex_houses.Q_order(i))));
end

% Compute mean of mean quarterly returns for houses (no costs)
mean_Q_ret_no_costs = nanmean(priceindex_houses.Q_ret_no_costs);

% Compute excess quarterly returns for houses (no costs)
priceindex_houses.excess_Q_ret_no_costs = priceindex_houses.Q_ret_no_costs-
priceindex_houses.rf;

% Compute mean excess quarterly returns for houses (no costs)
mean_excess_Q_ret_no_costs = nanmean(priceindex_houses.excess_Q_ret_no_costs);

% Compute standard deviation for each unique ID
st_Q_ret_no_costs= zeros(length(C),1);
time(1,:) = clock;
for i = 1:length(C)
    if b(i)<b(length(C))
       st_Q_ret_no_costs(i) = std(price_houses.Q_ret_no_costs(((b(i)+1):(b(i+1)-1))));
    else
       st_Q_ret_no_costs(i) =
std(price_houses.Q_ret_no_costs((b(length(C))+1):height(price_houses)));
    end
time(i+1,:) = clock;
fprintf('\nTo complete %.f/%.f iterations (unique IDs) it took %.2f seconds\n', i,
length(C), etime(time(i+1,:), time(1,:)));
end

% Standard deviation of all quarterly returns
std_Q_ret_no_costs = mean(st_Q_ret_no_costs);

% Import stats for 'Q_ret_no_costs' in summary table
sum_stats_houses.House_Indices(1) = 'Q_ret_no_costs';
sum_stats_houses.Return(1)        = mean_Q_ret_no_costs;
sum_stats_houses.std_dev(1)       = std_Q_ret_no_costs;
sum_stats_houses.Excess_return(1) = mean_excess_Q_ret_no_costs;
sum_stats_houses.Sharpe_ratio(1)  =
sum_stats_houses.Excess_return(1)/sum_stats_houses.std_dev(1);
sum_stats_houses.Skewness(1)      = skewness(priceindex_houses.Q_ret_no_costs(2:44));
sum_stats_houses.Kurtosis(1)      = kurtosis(priceindex_houses.Q_ret_no_costs(2:44));

% COMPUTE CUMULATIVE ROA(NC)
first_obs_price = price_houses.estimated_price(1:44:end);
price_houses.first_obs_price_repeated = repelem(first_obs_price, 44);

price_houses.cum_Q_ret_no_costs =
(price_houses.estimated_price./price_houses.first_obs_price_repeated)-1;
price_houses.cum_Q_ret_no_costs(1:44:end) = NaN;

priceindex_houses.cum_Q_ret_no_costs = zeros(height(priceindex_houses),end);

% Compute mean cumulative ROA(NC)
for i = 1:height(priceindex_houses)
    priceindex_houses.cum_Q_ret_no_costs(i) =
nanmean(price_houses.cum_Q_ret_no_costs(find(price_houses.Q ==
priceindex_houses.Q_order(i))));
end

% 8. COMPUTATION OF QUARTERLY RETURNS ON ASSET/HOUSE FROM PERIOD TO PERIOD (BUY/SELL),
INCL TOTAL COSTS
% Compute quarterly returns for each unique ID
price_houses.Q_ret_tot_costs = (((price_houses.estimated_price - price_houses.t_cost_s
- price_houses.lagged_t_cost - price_houses.lagged_estimated_price).*(1-
price_houses.tax_rates)) - price_houses.living_exp +
price_houses.implicit_rent)./(price_houses.lagged_estimated_price);
```

```
price_houses.Q_ret_tot_costs(1:44:end) = NaN;

% Compute mean quarterly returns for houses (mean of all IDs per quarter) (inc tot
costs)
priceindex_houses.Q_ret_tot_costs = zeros(height(priceindex_houses), end);

for i = 1:length(priceindex_houses.mean_prices)
    priceindex_houses.Q_ret_tot_costs(i) =
mean(price_houses.Q_ret_tot_costs(find(price_houses.Q ==
priceindex_houses.Q_order(i))));
end

% Compute mean of mean quarterly returns for houses (inc tot costs)
mean_Q_ret_tot_costs = nanmean(priceindex_houses.Q_ret_tot_costs);

% Compute excess quarterly returns for houses (inc tot costs)
priceindex_houses.excess_Q_ret_tot_costs = priceindex_houses.Q_ret_tot_costs-
priceindex_houses.rf;

% Compute mean excess quarterly returns for houses (inc tot costs)
mean_excess_Q_ret_tot_costs = nanmean(priceindex_houses.excess_Q_ret_tot_costs);

% Compute standard deviation for each unique IDs returns
st_Q_ret_tot_costs= zeros(length(C),1);
time(1,:) = clock;
for i = 1:length(C)
    if b(i)<b(length(C))
       st_Q_ret_tot_costs(i) = std(price_houses.Q_ret_tot_costs(((b(i)+1):(b(i+1)-
1))));
    else
       st_Q_ret_tot_costs(i) =
std(price_houses.Q_ret_tot_costs((b(length(C))+1):height(price_houses)));
    end
time(i+1,:) = clock;
fprintf('\nTo complete %.f/%.f iterations (unique IDs) it took %.2f seconds\n', i,
length(C), etime(time(i+1,:), time(1,:)));
end

% Standard deviation of all quarterly returns
std_Q_ret_tot_costs = mean(st_Q_ret_tot_costs);

% Import stats for 'Q_ret_tot_costs' in summary table
sum_stats_houses.House_Indices(2) = 'Q_ret_tot_costs';
sum_stats_houses.Return(2)        = mean_Q_ret_tot_costs;
sum_stats_houses.std_dev(2)       = std_Q_ret_tot_costs;
sum_stats_houses.Excess_return(2) = mean_excess_Q_ret_tot_costs;
sum_stats_houses.Sharpe_ratio(2)  =
sum_stats_houses.Excess_return(2)/sum_stats_houses.std_dev(2);
sum_stats_houses.Skewness(2)      = skewness(priceindex_houses.Q_ret_tot_costs(2:44));
sum_stats_houses.Kurtosis(2)      = kurtosis(priceindex_houses.Q_ret_tot_costs(2:44));

% COMPUTE CUMULATIVE ROA (TC)
taxes_TC = (price_houses.estimated_price - price_houses.t_cost_s -
price_houses.lagged_t_cost -
price_houses.lagged_estimated_price).*price_houses.tax_rates;
taxes_TC(1:44:end) = NaN;

net_costs_TC = (price_houses.living_exp + price_houses.lagged_t_cost +
price_houses.t_cost_s + taxes_TC - price_houses.implicit_rent);
net_costs_TC(1:44:end) = NaN;

% Compute cumulative costs
cumcosts_TC = zeros(height(price_houses), end);
time(1,:) = clock;
for i = 1:length(C)
    if b(i)<b(length(C))
       cumcosts_TC((b(i)):(b(i+1)-1)) = cumsum(net_costs_TC((b(i)):(b(i+1)-
1)),'omitnan');
    else
       cumcosts_TC(b(length(C)):height(price_houses)) =
cumsum(net_costs_TC(b(length(C)):height(price_houses)),'omitnan');
    end
time(i+1,:) = clock;
```

```
fprintf('\nTo complete %.f/%.f iterations (unique IDs) it took %.2f seconds\n', i,
length(C), etime(time(i+1,:), time(1,:)));
end

price_houses.cum_Q_ret_tot_costs = (price_houses.estimated_price - cumcosts_TC -
price_houses.first_obs_price_repeated)./price_houses.first_obs_price_repeated;
price_houses.cum_Q_ret_tot_costs(1:44:end) = NaN;

priceindex_houses.cum_Q_ret_tot_costs = zeros(height(priceindex_houses),end);
% Compute mean cumulative returns for houses (mean of all IDs per quarter) (no costs)
for i = 1:height(priceindex_houses)
    priceindex_houses.cum_Q_ret_tot_costs(i) =
nanmean(price_houses.cum_Q_ret_tot_costs(find(price_houses.Q ==
priceindex_houses.Q_order(i))));
end

all_lost = priceindex_houses.cum_Q_ret_tot_costs < -1;
priceindex_houses.cum_Q_ret_tot_costs(all_lost) = -1;

% 9. COMPUTATION OF QUARTERLY RETURNS ON ASSET/HOUSE FROM PERIOD TO PERIOD , INCL
SMOOTHED COSTS
% Locate buyer and sellers cost when buying in 2010Q1 and selling in 2020Q4
% Find buyer and seller cost per quarter for sample period
cost_2010Q1 = price_houses.t_cost(1:44:end);
cost_2010Q1 = cost_2010Q1/44;
price_houses.cost_2010Q1 = repelem(cost_2010Q1, 44);

cost_2020Q4 = price_houses.t_cost_s(44:44:end);
cost_2020Q4 = cost_2020Q4/44;
price_houses.cost_2020Q4 = repelem(cost_2020Q4, 44);

% Compute quarterly returns given that you buy in 2010Q1 and sell in 2020Q4
% and spred the relevant costs on the sample period.
price_houses.Q_ret_smooth_costs = (price_houses.estimated_price -
price_houses.cost_2010Q1 - price_houses.cost_2020Q4 - price_houses.living_exp -
price_houses.lagged_estimated_price +
price_houses.implicit_rent)./price_houses.lagged_estimated_price;
price_houses.Q_ret_smooth_costs(1:44:end) = NaN;

% Compute mean quarterly returns for houses (mean of all IDs per quarter) (inc smooth
costs)
priceindex_houses.Q_ret_smooth_costs = zeros(height(priceindex_houses), end);

for i = 1:length(priceindex_houses.mean_prices)
    priceindex_houses.Q_ret_smooth_costs(i) =
mean(price_houses.Q_ret_smooth_costs(find(price_houses.Q ==
priceindex_houses.Q_order(i))));
end

% Compute mean of mean quarterly returns for houses (inc smooth costs)
mean_Q_ret_smooth_costs = nanmean(priceindex_houses.Q_ret_smooth_costs);

% Compute excess quarterly returns for houses (inc smooth costs)
priceindex_houses.excess_Q_ret_smooth_costs = priceindex_houses.Q_ret_smooth_costs -
priceindex_houses.rf;

% Compute mean excess quarterly returns for houses (inc smooth costs)
mean_excess_Q_ret_smooth_costs = nanmean(priceindex_houses.excess_Q_ret_smooth_costs);

% Compute standard deviation for each unique IDs returns
st_Q_ret_smooth_costs= zeros(length(C),1);
time(1,:) = clock;
for i = 1:length(C)
    if b(i)<b(length(C))
       st_Q_ret_smooth_costs(i) =
std(price_houses.Q_ret_smooth_costs(((b(i)+1):(b(i+1)-1))));
    else
       st_Q_ret_smooth_costs(i) =
std(price_houses.Q_ret_smooth_costs((b(length(C))+1):height(price_houses)));
    end
time(i+1,:) = clock;
fprintf('\nTo complete %.f/%.f iterations (unique IDs) it took %.2f seconds\n', i,
length(C), etime(time(i+1,:), time(1,:)));
end
```

```
% Standard deviation of all quarterly returns
std_Q_ret_smooth_costs = mean(st_Q_ret_smooth_costs);

% Import stats for 'Q_ret_smooth_costs' in summary table
sum_stats_houses.House_Indices(3) = 'Q_ret_smooth_costs';
sum_stats_houses.Return(3)        = mean_Q_ret_smooth_costs;
sum_stats_houses.std_dev(3)       = std_Q_ret_smooth_costs;
sum_stats_houses.Excess_return(3) = mean_excess_Q_ret_smooth_costs;
sum_stats_houses.Sharpe_ratio(3)  =
sum_stats_houses.Excess_return(3)/sum_stats_houses.std_dev(3);
sum_stats_houses.Skewness(3)      =
skewness(priceindex_houses.Q_ret_smooth_costs(2:44));
sum_stats_houses.Kurtosis(3)      =
kurtosis(priceindex_houses.Q_ret_smooth_costs(2:44));

% COMPUTE CUMULATIVE ROA(SC)
net_costs_SC = (price_houses.living_exp + price_houses.cost_2010Q1 +
price_houses.cost_2020Q4 - price_houses.implicit_rent);
net_costs_SC(1:44:end) = NaN;

cumcosts_SC = zeros(height(price_houses), end);
time(1,:) = clock;
for i = 1:length(C)
    if b(i)<b(length(C))
       cumcosts_SC((b(i)):(b(i+1)-1)) = cumsum(net_costs_SC((b(i)):(b(i+1)-
1)),'omitnan');
    else
       cumcosts_SC(b(length(C)):height(price_houses)) =
cumsum(net_costs_SC(b(length(C)):height(price_houses)),'omitnan');
    end
time(i+1,:) = clock;
fprintf('\nTo complete %.f/%.f iterations (unique IDs) it took %.2f seconds\n', i,
length(C), etime(time(i+1,:), time(1,:)));
end

price_houses.cum_Q_ret_smooth_costs = (price_houses.estimated_price - cumcosts_SC -
price_houses.first_obs_price_repeated)./price_houses.first_obs_price_repeated;
price_houses.cum_Q_ret_smooth_costs(1:44:end) = NaN;

% Compute mean cumulative returns for houses (mean of all IDs per quarter) (no costs)
priceindex_houses.cum_Q_ret_smooth_costs = zeros(height(priceindex_houses),end);
for i = 1:height(priceindex_houses)
    priceindex_houses.cum_Q_ret_smooth_costs(i) =
nanmean(price_houses.cum_Q_ret_smooth_costs(find(price_houses.Q ==
priceindex_houses.Q_order(i))));
end

% Identify average loan-to-value ratio for housing investments
l = ([0.6957, 0.6957, 0.6957, 0.6957, 0.71, 0.76, 0.67, 0.66, 0.68, 0.70, 0.69])';
n = 4;
ltv_ratio_order =
[repmat(l(1),n,1);repmat(l(2),n,1);repmat(l(3),n,1);repmat(l(4),n,1);repmat(l(5),n,1);
repmat(l(6),n,1);repmat(l(7),n,1);repmat(l(8),n,1);repmat(l(9),n,1);repmat(l(10),n,1);
repmat(l(11),n,1)];

% Allocate correct ltv-rates to each quarter in price_houses set
price_houses.ltvr        = repmat(ltv_ratio_order,(length(C)),1);
price_houses.lagged_ltvr = lagmatrix(price_houses.ltvr,1);

% Allocate ltv ratio as of 2010Q1 to each quarter in price_houses set
price_houses.ltvr_2010Q1(:) = price_houses.ltvr(1);

% 10. WE DONT USE THE RETURN SERIES FROM SECTION 10
% Compute quarterly equity returns for each unique ID (no costs)
price_houses.Q_eq_ret_no_costs = ((price_houses.estimated_price -
price_houses.lagged_estimated_price)./(price_houses.lagged_estimated_price.*(1-
price_houses.lagged_ltvr)));
price_houses.Q_eq_ret_no_costs(1:44:end) = NaN;

% Compute mean quarterly equity returns for houses (mean of all IDs per quarter) (no
costs)
priceindex_houses.Q_eq_ret_no_costs = zeros(height(priceindex_houses), end);
```

```
for i = 1:height(priceindex_houses)
    priceindex_houses.Q_eq_ret_no_costs(i) =
mean(price_houses.Q_eq_ret_no_costs(find(price_houses.Q ==
priceindex_houses.Q_order(i))));
end

% Compute mean of mean quarterly equity returns for houses (no costs)
mean_Q_eq_ret_no_costs = nanmean(priceindex_houses.Q_eq_ret_no_costs);

% Compute excess quarterly equity returns for houses (no costs)
priceindex_houses.excess_Q_eq_ret_no_costs = priceindex_houses.Q_eq_ret_no_costs -
priceindex_houses.rf;

% Compute mean excess quarterly equity returns for houses (no costs)
mean_excess_Q_eq_ret_no_costs = nanmean(priceindex_houses.excess_Q_eq_ret_no_costs);

% Compute standard deviation for each unique IDs returns
st_Q_eq_ret_no_costs= zeros(length(C),1);
time(1,:) = clock;
for i = 1:length(C)
    if b(i)<b(length(C))
       st_Q_eq_ret_no_costs(i) = std(price_houses.Q_eq_ret_no_costs(((b(i)+1):(b(i+1)-
1))));
    else
       st_Q_eq_ret_no_costs(i) =
std(price_houses.Q_eq_ret_no_costs((b(length(C))+1):height(price_houses)));
    end
time(i+1,:) = clock;
fprintf('\nTo complete %.f/%.f iterations (unique IDs) it took %.2f seconds\n', i,
length(C), etime(time(i+1,:), time(1,:)));
end

% Standard deviation of all quarterly equity returns
std_Q_eq_ret_no_costs = mean(st_Q_eq_ret_no_costs);

% Import stats for 'Q_eq_ret_no_costs' in summary table
sum_stats_houses.House_Indices(4) = 'Q_eq_ret_no_costs';
sum_stats_houses.Return(4)        = mean_Q_eq_ret_no_costs;
sum_stats_houses.std_dev(4)       = std_Q_eq_ret_no_costs;
sum_stats_houses.Excess_return(4) = mean_excess_Q_eq_ret_no_costs;
sum_stats_houses.Sharpe_ratio(4)  =
sum_stats_houses.Excess_return(4)/sum_stats_houses.std_dev(4);
sum_stats_houses.Skewness(4)      =
skewness(priceindex_houses.Q_eq_ret_no_costs(2:44));
sum_stats_houses.Kurtosis(4)      =
kurtosis(priceindex_houses.Q_eq_ret_no_costs(2:44));

% COMPUTE CUMULATIVE ROE (NC)
price_houses.cum_Q_eq_ret_no_costs = ((price_houses.estimated_price -
price_houses.first_obs_price_repeated)./(price_houses.first_obs_price_repeated.*(1-
price_houses.ltvr_2010Q1)));
price_houses.cum_Q_eq_ret_no_costs(1:44:end) = NaN;

% Compute mean cumulative returns for houses (mean of all IDs per quarter) (no costs)
priceindex_houses.cum_Q_eq_ret_no_costs = zeros(height(priceindex_houses),end);
for i = 1:height(priceindex_houses)
    priceindex_houses.cum_Q_eq_ret_no_costs(i) =
nanmean(price_houses.cum_Q_eq_ret_no_costs(find(price_houses.Q ==
priceindex_houses.Q_order(i))));
end

% 11. WE DONT USE THE RETURN SERIES FROM SECTION 11
% Compute quarterly equity returns for each unique ID (inc tot costs)
price_houses.Q_eq_ret_tot_costs = (((price_houses.estimated_price -
price_houses.t_cost_s - price_houses.lagged_t_cost -
price_houses.lagged_estimated_price).*(1-price_houses.tax_rates)) -
price_houses.living_exp +
price_houses.implicit_rent)./(price_houses.lagged_estimated_price.*(1-
price_houses.lagged_ltvr));
price_houses.Q_eq_ret_tot_costs(1:44:end) = NaN;

% Compute mean quarterly equity returns for houses (mean of all IDs per quarter) (inc
tot costs)
priceindex_houses.Q_eq_ret_tot_costs = zeros(height(priceindex_houses), end);
```

```
for i = 1:length(priceindex_houses.mean_prices)
    priceindex_houses.Q_eq_ret_tot_costs(i) =
mean(price_houses.Q_eq_ret_tot_costs(find(price_houses.Q ==
priceindex_houses.Q_order(i))));
end

% Compute mean of mean quarterly equity returns for houses (inc tot costs)
mean_Q_eq_ret_tot_costs = nanmean(priceindex_houses.Q_eq_ret_tot_costs);

% Compute excess quarterly equity returns for houses (inc tot costs)
priceindex_houses.excess_Q_eq_ret_tot_costs = priceindex_houses.Q_eq_ret_tot_costs -
priceindex_houses.rf;

% Compute mean excess quarterly equity returns for houses (inc tot costs)
mean_excess_Q_eq_ret_tot_costs = nanmean(priceindex_houses.excess_Q_eq_ret_tot_costs);

% Compute standard deviation for each unique IDs returns
st_Q_eq_ret_tot_costs= zeros(length(C),1);
time(1,:) = clock;
for i = 1:length(C)
    if b(i)<b(length(C))
       st_Q_eq_ret_tot_costs(i) =
std(price_houses.Q_eq_ret_tot_costs(((b(i)+1):(b(i+1)-1))));
    else
       st_Q_eq_ret_tot_costs(i) =
std(price_houses.Q_eq_ret_tot_costs((b(length(C))+1):height(price_houses)));
    end
time(i+1,:) = clock;
fprintf('\nTo complete %.f/%.f iterations (unique IDs) it took %.2f seconds\n', i,
length(C), etime(time(i+1,:), time(1,:)));
end

% Standard deviation of all quarterly equity returns
std_Q_eq_ret_tot_costs = mean(st_Q_eq_ret_tot_costs);

% Import stats for 'Q_eq_ret_tot_costs' in summary table
sum_stats_houses.House_Indices(5) = 'Q_eq_ret_tot_costs';
sum_stats_houses.Return(5)        = mean_Q_eq_ret_tot_costs;
sum_stats_houses.std_dev(5)       = std_Q_eq_ret_tot_costs;
sum_stats_houses.Excess_return(5) = mean_excess_Q_eq_ret_tot_costs;
sum_stats_houses.Sharpe_ratio(5)  =
sum_stats_houses.Excess_return(5)/sum_stats_houses.std_dev(5);
sum_stats_houses.Skewness(5)      =
skewness(priceindex_houses.Q_eq_ret_tot_costs(2:44));
sum_stats_houses.Kurtosis(5)      =
kurtosis(priceindex_houses.Q_eq_ret_tot_costs(2:44));

% COMPUTE CUMULATIVE RETURNS BASED ON QUARTERLY EQUITY RETURNS FROM PRICE INDEX (INC
TOT COSTS)
priceindex_houses.cum_Q_eq_ret_tot_costs = zeros(height(priceindex_houses),end);
priceindex_houses.cum_Q_eq_ret_tot_costs =
cumprod((1+priceindex_houses.Q_eq_ret_tot_costs),'omitnan') -1;

% 12. COMPUTATION OF QUARTERLY EQUITY RETURNS ON ASSET/HOUSE FROM PERIOD TO PERIOD ,
INCL SMOOTHED COSTS
% Compute quarterly EQUITY returns given that you buy in 2010Q1 and sell in 2020Q4
% and spread the relevant costs on the sample period.
price_houses.Q_eq_ret_smooth_costs = (price_houses.estimated_price -
price_houses.cost_2010Q1 - price_houses.cost_2020Q4 - price_houses.living_exp -
price_houses.lagged_estimated_price +
price_houses.implicit_rent)./(price_houses.lagged_estimated_price.*(1-
price_houses.ltvr_2010Q1));
price_houses.Q_eq_ret_smooth_costs(1:44:end) = NaN;

% Compute mean quarterly equity returns for houses (mean of all IDs per quarter) (inc
smooth costs)
priceindex_houses.Q_eq_ret_smooth_costs = zeros(height(priceindex_houses), end);

for i = 1:length(priceindex_houses.mean_prices)
    priceindex_houses.Q_eq_ret_smooth_costs(i) =
mean(price_houses.Q_eq_ret_smooth_costs(find(price_houses.Q ==
priceindex_houses.Q_order(i))));
end
```

```
% Compute mean of mean quarterly equity returns for houses (inc smooth costs)
mean_Q_eq_ret_smooth_costs = nanmean(priceindex_houses.Q_eq_ret_smooth_costs);

% Compute excess quarterly equity returns for houses (inc smooth costs)
priceindex_houses.excess_Q_eq_ret_smooth_costs =
priceindex_houses.Q_eq_ret_smooth_costs - priceindex_houses.rf;

% Compute mean excess quarterly equity returns for houses (inc smooth costs)
mean_excess_Q_eq_ret_smooth_costs =
nanmean(priceindex_houses.excess_Q_eq_ret_smooth_costs);

% Compute standard deviation for each unique IDs returns
st_Q_eq_ret_smooth_costs= zeros(length(C),1);
time(1,:) = clock;
for i = 1:length(C)
    if b(i)<b(length(C))
       st_Q_eq_ret_smooth_costs(i) =
std(price_houses.Q_eq_ret_smooth_costs(((b(i)+1):(b(i+1)-1))));
    else
       st_Q_eq_ret_smooth_costs(i) =
std(price_houses.Q_eq_ret_smooth_costs((b(length(C))+1):height(price_houses)));
    end
time(i+1,:) = clock;
fprintf('\nTo complete %.f/%.f iterations (unique IDs) it took %.2f seconds\n', i,
length(C), etime(time(i+1,:), time(1,:)));
end

% Standard deviation of all quarterly equity returns
std_Q_eq_ret_smooth_costs = mean(st_Q_eq_ret_smooth_costs);

% Import stats for 'Q_eq_ret_smooth_costs' in summary table
sum_stats_houses.House_Indices(6) = 'Q_eq_ret_smooth_costs';
sum_stats_houses.Return(6)        = mean_Q_eq_ret_smooth_costs;
sum_stats_houses.std_dev(6)       = std_Q_eq_ret_smooth_costs;
sum_stats_houses.Excess_return(6) = mean_excess_Q_eq_ret_smooth_costs;
sum_stats_houses.Sharpe_ratio(6)  =
sum_stats_houses.Excess_return(6)/sum_stats_houses.std_dev(6);
sum_stats_houses.Skewness(6)      =
skewness(priceindex_houses.Q_eq_ret_smooth_costs(2:44));
sum_stats_houses.Kurtosis(6)      =
kurtosis(priceindex_houses.Q_eq_ret_smooth_costs(2:44));

% COMPUTE CUMULATIVE ROE(SC)
price_houses.cum_Q_eq_ret_smooth_costs = (price_houses.estimated_price - cumcosts_SC -
price_houses.first_obs_price_repeated)./(price_houses.first_obs_price_repeated.*(1-
price_houses.ltvr_2010Q1));
price_houses.cum_Q_eq_ret_smooth_costs(1:44:end) = NaN;

% Compute mean cumulative returns for houses (mean of all IDs per quarter) (smooth
costs)
priceindex_houses.cum_Q_eq_ret_smooth_costs = zeros(height(priceindex_houses),end);
for i = 1:height(priceindex_houses)
    priceindex_houses.cum_Q_eq_ret_smooth_costs(i) =
nanmean(price_houses.cum_Q_eq_ret_smooth_costs(find(price_houses.Q ==
priceindex_houses.Q_order(i))));
end

% MAKE PLOT OF THE RELEVANT INDICES TOGHETER, MAKE MULTIPLE PLOTS FOR COMPARISON
REASONS
houseprice_changes = priceindex_houses(:,[1 5 8 11 14 17 20]);

houseprice_indices = priceindex_houses(:,[1 7 10 13 16 19 22]);
houseprice_indices.cum_Q_ret_no_costs(1) = 0;
houseprice_indices.cum_Q_ret_tot_costs(1) = 0;
houseprice_indices.cum_Q_ret_smooth_costs(1) = 0;
houseprice_indices.cum_Q_eq_ret_no_costs(1) = 0;
houseprice_indices.cum_Q_eq_ret_tot_costs(1) = 0;
houseprice_indices.cum_Q_eq_ret_smooth_costs(1) = 0;

houseprice_indices.cum_Q_ret_no_costs        = houseprice_indices.cum_Q_ret_no_costs +
1;
houseprice_indices.cum_Q_ret_tot_costs       = houseprice_indices.cum_Q_ret_tot_costs
+ 1;
```

```
houseprice_indices.cum_Q_ret_smooth_costs    =
houseprice_indices.cum_Q_ret_smooth_costs + 1;
houseprice_indices.cum_Q_eq_ret_no_costs     =
houseprice_indices.cum_Q_eq_ret_no_costs + 1;
houseprice_indices.cum_Q_eq_ret_tot_costs    =
houseprice_indices.cum_Q_eq_ret_tot_costs + 1;
houseprice_indices.cum_Q_eq_ret_smooth_costs =
houseprice_indices.cum_Q_eq_ret_smooth_costs + 1;

clearvars -except Property_Data summary_table Rf Rent_Data...
                  house_set hedonic_houses housing_model...
                  price_houses uniq_house_set avg_cost_rates_houses...
                  priceindex_houses sum_stats_houses houseprice_changes...
                  houseprice_indices...
                  smallhouse_set hedonic_smallhouses smallhouse_model...
                  price_smallhouses uniq_smallhouse_set avg_cost_rates_smallhouses...
                  priceindex_smallhouses sum_stats_smallhouses...
                  smallhouse_price_changes smallhouse_price_indices...
                  apartment_set hedonic_apartments apartment_model price_apartments...
                  uniq_apartment_set avg_cost_rates_apartments
priceindex_apartments...
                  sum_stats_apartments apartment_price_changes...
                  apartment_price_indices...
                  inner_eastern_set hedonic_inner_eastern inner_eastern_model...
                  price_inner_eastern uniq_inner_eastern_set
avg_cost_rates_inner_eastern...
                  priceindex_inner_eastern sum_stats_inner_eastern
inner_eastern_price_changes...
                  inner_eastern_price_indices...
                  inner_western_set hedonic_inner_western inner_western_model...
                  price_inner_western uniq_inner_western_set
avg_cost_rates_inner_western...
                  priceindex_inner_western sum_stats_inner_western
inner_western_price_changes...
                  inner_western_price_indices...
                  outer_eastern_set hedonic_outer_eastern outer_eastern_model...
                  price_outer_eastern uniq_outer_eastern_set
avg_cost_rates_outer_eastern...
                  priceindex_outer_eastern sum_stats_outer_eastern
outer_eastern_price_changes...
                  outer_eastern_price_indices...
                  outer_western_set hedonic_outer_western outer_western_model...
                  price_outer_western uniq_outer_western_set
avg_cost_rates_outer_western...
                  priceindex_outer_western sum_stats_outer_western
outer_western_price_changes...
                  outer_western_price_indices...
                  outer_southern_set hedonic_outer_southern outer_southern_model...
                  price_outer_southern uniq_outer_southern_set
avg_cost_rates_outer_southern...
                  priceindex_outer_southern sum_stats_outer_southern
outer_southern_price_changes...
                  outer_southern_price_indices...
                  property_price_changes property_price_indices hedonic_total...
                  ROA_NC_stats ROE_NC_stats ROA_TC_stats ROE_TC_stats...
                  ROA_SC_stats ROE_SC_stats sum_stats_OSEFX OSEFX...
                  ROA_NC_sharpe_test ROA_TC_sharpe_test ROA_SC_sharpe_test...
                  ROE_SC_sharpe_test

%% Regression model for Small Houses
% 1. PREPERATION OF DATASET
small_houses    = Property_Data.Boligtype == 'Smaahus';
smallhouse_set  = Property_Data(small_houses,:);

% Make categorical for dummy-purposes
smallhouse_set.Bydel    = categorical(smallhouse_set.Bydel);
smallhouse_set.Alder    = categorical(smallhouse_set.Alder);
smallhouse_set.Q        = categorical(smallhouse_set.Q);
smallhouse_set.Eierform = categorical(smallhouse_set.Eierform);

% Reorder categories/dummies to find reference variables
smallhouse_set.Bydel    = reordercats(smallhouse_set.Bydel,    {'Gr√°nerl√∏kka',
'Alna', 'Bjerke', 'Frogner', 'Gamle Oslo', 'Grorud', 'Nordre Aker', 'Nordstrand',
```

```
'Sagene', 'Sentrum', 'St. Hanshaugen', 'Stovner', 'S√∏ndre Nordstrand', 'Ullern',
'Vestre Aker', '√òstensj√∏', 'NULL', 'Marka'});
smallhouse_set.Alder    = reordercats(smallhouse_set.Alder,    {'Age1', 'Age2',
'Age3', 'Age4'});
smallhouse_set.Eierform = reordercats(smallhouse_set.Eierform, {'Selveier',
'Borettslag', 'Obl.leilighet', 'Aksjeleilighet', 'Ukjent'});

% Remove unused categories
smallhouse_set.Bydel    = removecats(smallhouse_set.Bydel);
smallhouse_set.Eierform = removecats(smallhouse_set.Eierform);

% Check that we in fact removed unused categories
categories(smallhouse_set.Bydel)
categories(smallhouse_set.Alder)
categories(smallhouse_set.Eierform)

% Make log prices and sizes
smallhouse_set.log_pris = log(smallhouse_set.Pris);
smallhouse_set.log_prom = log(smallhouse_set.prom);

% 2. Estimate regression model - Log model with log(prom), Q, Bydel, Soverom, Alder on
log(pris)
% Explanatory variables log(prom) and Soverom - continous
% Explanatory variables Q, Bydel and Alder - dummies
smallhouse_model =
fitlm(smallhouse_set,'log_pris~log_prom+Q+Bydel+Eierform+Soverom+Alder');

% 3. ALLOCATE TABLE FOR ESTIMATED PRICES (BASED ON MODEL FROM PT. 2) FOR EVERY QUARTER
FOR EACH ID
smallhouse_set    = sortrows(smallhouse_set,'BoligID','ascend');
[C,ia,ic]    = unique(smallhouse_set.BoligID,'first');
price_smallhouses = zeros(length(C)*44, width(smallhouse_set));
price_smallhouses = array2table(price_smallhouses);

price_smallhouses.Properties.VariableNames = {'BoligID', 'Salgsdato', 'Bydel', 'Pris',
'Fellesgjeld', 'Prisantydning', 'Eierform', 'Byggeaar', 'prom', 'Etasje', 'Soverom',
'Boligtype', 'year', 'Q', 'Alder', 'kvmpris', 'log_pris', 'log_prom' };

Q_rep   = repmat(["Q1"; "Q2"; "Q3";"Q4"],(length(C)*44)/4,1);
x       = (2010:2020)';
n       = 4;
newx    =
[repmat(x(1),n,1);repmat(x(2),n,1);repmat(x(3),n,1);repmat(x(4),n,1);repmat(x(5),n,1);
repmat(x(6),n,1);repmat(x(7),n,1);repmat(x(8),n,1);repmat(x(9),n,1);repmat(x(10),n,1);
repmat(x(11),n,1)];
Y_order = repmat(newx,(length(C)),1);

price_smallhouses.Q_order   = string(Y_order(:,1)) + Q_rep(:,1);
price_smallhouses.Q         = price_smallhouses.Q_order;
price_smallhouses.Q_order   = [];
price_smallhouses.BoligID   = categorical(price_smallhouses.BoligID);
price_smallhouses.BoligID   = repelem(C, 44);
price_smallhouses           = convertvars(price_smallhouses,{'BoligID', 'Bydel',
'Eierform', 'Boligtype', 'Alder', 'Q'},'categorical');
price_smallhouses.Salgsdato =
datetime(price_smallhouses.Salgsdato,'ConvertFrom','yyyymmdd');

b = 1:length(newx):height(price_smallhouses);

% Create set containing only the unique observations
uniq_smallhouse_set = smallhouse_set(ia,:);

% Allocate variables into price_smallhouses to use for regression model
price_smallhouses(:,[3 5 7 9 11 15 18]) = repelem(uniq_smallhouse_set(:,[3 5 7 9 11 15
18]), 44, 1);

% Store relevant variables for regression model in new table
x3 = price_smallhouses(:,[18 14 3 7 11 15]);

% GET COLUMN OF ESTIMATED PRICES BASED ON REGRESSION MODEL
price_smallhouses.log_estimated_price    = predict(smallhouse_model, x3);
price_smallhouses.estimated_price        = exp(price_smallhouses.log_estimated_price);
price_smallhouses.lagged_estimated_price =
lagmatrix(price_smallhouses.estimated_price,1);
```

```
% 4. IMPLEMENT RELEVANT COSTS RELATED TO HOUSING INVESTMENTS
% Transaction costs buyer, b
gebyr_b                  = zeros(height(price_smallhouses),1);
dokavg_b                 = zeros(height(price_smallhouses),1);
insurance_b              = zeros(height(price_smallhouses),1);
price_smallhouses.t_cost = zeros(height(price_smallhouses),1);

% If type of ownership = 'Selveier'
idx_selv              = price_smallhouses.Eierform == "Selveier";
gebyr_b(idx_selv)     = 540 + 540 + 172;
dokavg_b(idx_selv)    =
0.025.*(price_smallhouses.estimated_price(idx_selv)+price_smallhouses.Fellesgjeld(idx_
selv));
insurance_b(idx_selv) = 9450;

% If type of ownership = 'Borettslag'
idx_andel              = price_smallhouses.Eierform == "Borettslag";
gebyr_b(idx_andel)     = 440 + 440 + 172;
dokavg_b(idx_andel)    = 0;
insurance_b(idx_andel) = 4100;

% Compute total transaction cost for buyer, b, for each observation/quarter
price_smallhouses.t_cost        = gebyr_b+dokavg_b+insurance_b;
price_smallhouses.lagged_t_cost = lagmatrix(price_smallhouses.t_cost,1);

% Transaction costs seller, s
eierskiftegebyr_s          = zeros(height(price_smallhouses),1);
public_fees_s              = zeros(height(price_smallhouses),1);
insurance_s                = zeros(height(price_smallhouses),1);
commission_s               = zeros(height(price_smallhouses),1);
price_smallhouses.t_cost_s = zeros(height(price_smallhouses),1);
indices_selv               = zeros(height(price_smallhouses),1);
indices_andel              = zeros(height(price_smallhouses),1);

% If type of ownership = 'Selveier'
idx_selv                    = price_smallhouses.Eierform == "Selveier";
public_fees_s(idx_selv)     = 8430;
eierskiftegebyr_s(idx_selv) = 0;
insurance_s(idx_selv)       =
max(0.005*(price_smallhouses.estimated_price(idx_selv)+price_smallhouses.Fellesgjeld(i
dx_selv)), 10250);
indices_selv                = insurance_s > 48750;
indices_selv                = logical(indices_selv);
insurance_s(indices_selv)   = 48750;

% If type of ownership = 'Borettslag'
idx_andel                    = price_smallhouses.Eierform == "Borettslag";
public_fees_s(idx_andel)     = 14319;
eierskiftegebyr_s(idx_andel) = 4858;
insurance_s(idx_andel)       =
max(0.002*(price_smallhouses.estimated_price(idx_andel)+price_smallhouses.Fellesgjeld(
idx_andel)), 2500);
indices_andel                = zeros(height(price_smallhouses),1);
indices_andel(idx_andel)     = insurance_s(idx_andel) > 24000;
indices_andel                = logical(indices_andel);
insurance_s(indices_andel)   = 24000;

% Commission to broker independent of type of ownership
commission_s =
max(0.02*(price_smallhouses.estimated_price+price_smallhouses.Fellesgjeld),40000);

% Compute total transaction cost for seller, s, for each observation/quarter
price_smallhouses.t_cost_s = public_fees_s+eierskiftegebyr_s+insurance_s+commission_s;

% Compute transaction cost rates for buyer and seller
% For buyer
t_cost_rate   = mean(price_smallhouses.t_cost./price_smallhouses.estimated_price);
t_cost_rate_Q = t_cost_rate/4;

% For seller
t_cost_s_rate   = mean(price_smallhouses.t_cost_s./price_smallhouses.estimated_price);
t_cost_s_rate_Q = t_cost_s_rate/4;
```

```
% Define holding period for each dwelling
holding_period = 11;

% Define living expenses per quarter - reference apartment is 70 sqm
liv_exp_a_70_Q = ([76734/4, 82530/4, 83430/4, 89095/4, 88857/4, 83084/4, 83769/4,
86976/4, 91419/4, 103712/4, 87905/4])';

% Allocate correct liwing expenses to each quarter in price_houses set
n = 4;
liv_exp_smallh =
[repmat(liv_exp_a_70_Q(1),n,1);repmat(liv_exp_a_70_Q(2),n,1);repmat(liv_exp_a_70_Q(3),
n,1);repmat(liv_exp_a_70_Q(4),n,1);repmat(liv_exp_a_70_Q(5),n,1);repmat(liv_exp_a_70_Q
(6),n,1);repmat(liv_exp_a_70_Q(7),n,1);repmat(liv_exp_a_70_Q(8),n,1);repmat(liv_exp_a_
70_Q(9),n,1);repmat(liv_exp_a_70_Q(10),n,1);repmat(liv_exp_a_70_Q(11),n,1)];
price_smallhouses.ordered_liv_exp_70 = repmat(liv_exp_smallh,(length(C)),1);

% Compute living expenses based on a reference apartment of 70 sqm
price_smallhouses.living_exp =
(price_smallhouses.prom./70).*price_smallhouses.ordered_liv_exp_70;
avg_living_exp_Q =
mean(price_smallhouses.living_exp./price_smallhouses.estimated_price);

% Allocate correct implicit rents
implicit_rent     = zeros(height(price_smallhouses),1);
rental_allocation = repmat(Rent_Data, length(C), 1);

implicit_rent(find(price_smallhouses.Soverom == 0)) =
rental_allocation.room1(find(price_smallhouses.Soverom == 0));
implicit_rent(find(price_smallhouses.Soverom == 1)) =
rental_allocation.room1(find(price_smallhouses.Soverom == 1));
implicit_rent(find(price_smallhouses.Soverom == 2)) =
rental_allocation.room2(find(price_smallhouses.Soverom == 2));
implicit_rent(find(price_smallhouses.Soverom == 3)) =
rental_allocation.room3(find(price_smallhouses.Soverom == 3));
implicit_rent(find(price_smallhouses.Soverom == 4)) =
rental_allocation.room4(find(price_smallhouses.Soverom == 4));
implicit_rent(find(price_smallhouses.Soverom >= 5)) =
rental_allocation.room5_(find(price_smallhouses.Soverom >= 5));

price_smallhouses.implicit_rent = implicit_rent;
avg_implicit_rent_Q =
mean(price_smallhouses.implicit_rent./price_smallhouses.estimated_price);

% Implement tax rates for sample period
tax_rates = ([ 0.28, 0.28, 0.28, 0.28, 0.27, 0.27, 0.25, 0.24, 0.23, 0.22, 0.22])';
n = 4;
tax_rates_order =
[repmat(tax_rates(1),n,1);repmat(tax_rates(2),n,1);repmat(tax_rates(3),n,1);repmat(tax
_rates(4),n,1);repmat(tax_rates(5),n,1);repmat(tax_rates(6),n,1);repmat(tax_rates(7),n
,1);repmat(tax_rates(8),n,1);repmat(tax_rates(9),n,1);repmat(tax_rates(10),n,1);repmat
(tax_rates(11),n,1)];

% Allocate correct tax rates to each quarter in price_houses set
price_smallhouses.tax_rates = repmat(tax_rates_order,(length(C)),1);

% Compute transaction costs for different ownership types (small houses)
SOUs_smallhouses      = price_smallhouses((price_smallhouses.Eierform ==
'Selveier'),:);
t_cost_rate_SOUs      =
mean(SOUs_smallhouses.t_cost./SOUs_smallhouses.estimated_price);
SOU_share_smallhouses = height(SOUs_smallhouses)/height(price_smallhouses);

co_ops_smallhouses       = price_smallhouses((price_smallhouses.Eierform ==
'Borettslag'),:);
t_cost_rate_co_ops       =
mean(co_ops_smallhouses.t_cost./co_ops_smallhouses.estimated_price);
Co_ops_share_smallhouses = height(co_ops_smallhouses)/height(price_smallhouses);

t_cost_rate_s_SOUs     =
mean(SOUs_smallhouses.t_cost_s./SOUs_smallhouses.estimated_price);
t_cost_rate_s_co_ops   =
mean(co_ops_smallhouses.t_cost_s./co_ops_smallhouses.estimated_price);
```

```
avg_living_exp_SOUs_Q   =
mean(SOUs_smallhouses.living_exp./SOUs_smallhouses.estimated_price);
avg_living_exp_co_ops_Q =
mean(co_ops_smallhouses.living_exp./co_ops_smallhouses.estimated_price);

% Create a table to store different cost rates
avg_cost_rates_smallhouses                          = zeros(4,5); %JUSTERE DENNE
ETTERSOM HVORDAN ST√òRRELSEN VIL BLI
avg_cost_rates_smallhouses                          =
array2table(avg_cost_rates_smallhouses);
avg_cost_rates_smallhouses.Properties.VariableNames = {'Average_cost_rates', 'Yearly',
'Quarterly', 'SOUs_Y', 'Co_ops_Y'};
avg_cost_rates_smallhouses.Average_cost_rates       =
categorical(avg_cost_rates_smallhouses.Average_cost_rates);

% Import stats for cost rates
avg_cost_rates_smallhouses.Average_cost_rates(1) = 't_cost_buyer';
avg_cost_rates_smallhouses.Yearly(1)             = t_cost_rate;
avg_cost_rates_smallhouses.Quarterly(1)          = t_cost_rate_Q;
avg_cost_rates_smallhouses.SOUs_Y(1)             = t_cost_rate_SOUs;
avg_cost_rates_smallhouses.Co_ops_Y(1)           = t_cost_rate_co_ops;

avg_cost_rates_smallhouses.Average_cost_rates(2) = 't_cost_seller';
avg_cost_rates_smallhouses.Yearly(2)             = t_cost_s_rate;
avg_cost_rates_smallhouses.Quarterly(2)          = t_cost_s_rate_Q;
avg_cost_rates_smallhouses.SOUs_Y(2)             = t_cost_rate_s_SOUs;
avg_cost_rates_smallhouses.Co_ops_Y(2)           = t_cost_rate_s_co_ops;

avg_cost_rates_smallhouses.Average_cost_rates(3) = 'living_expenses';
avg_cost_rates_smallhouses.Yearly(3)             = avg_living_exp_Q * 4;
avg_cost_rates_smallhouses.Quarterly(3)          = avg_living_exp_Q;
avg_cost_rates_smallhouses.SOUs_Y(3)             = avg_living_exp_SOUs_Q * 4;
avg_cost_rates_smallhouses.Co_ops_Y(3)           = avg_living_exp_co_ops_Q * 4;

avg_cost_rates_smallhouses.Average_cost_rates(4) = 'implicit_rent';
avg_cost_rates_smallhouses.Yearly(4)             = avg_implicit_rent_Q * 4;
avg_cost_rates_smallhouses.Quarterly(4)          = avg_implicit_rent_Q;

% 5. ALLOCATE TABLE FOR PRICE INDICES
priceindex_smallhouses         = table();
Y_index                        = Y_order(1:44);
Q_index                        = Q_rep(1:44);
priceindex_smallhouses.Q_order = string(Y_index(:,1)) + Q_index(:,1);
priceindex_smallhouses.Q_order = categorical(priceindex_smallhouses.Q_order);

% Create a table to store returns, std, risk free, sharpe
sum_stats_smallhouses                           = zeros(10,7); %JUSTERE DENNE ETTERSOM
HVORDAN ST√òRRELSEN VIL BLI
sum_stats_smallhouses                           = array2table(sum_stats_smallhouses);
sum_stats_smallhouses.Properties.VariableNames = {'House_Indices', 'Return',
'std_dev','Excess_return','Sharpe_ratio', 'Skewness', 'Kurtosis'};
sum_stats_smallhouses.House_Indices             =
categorical(sum_stats_smallhouses.House_Indices);

% Import risk-free rates (innskuddsrente)
n = 4;
priceindex_smallhouses.rf = zeros(44,1);
priceindex_smallhouses.rf =
[repmat(Rf.rf(1),n,1);repmat(Rf.rf(2),n,1);repmat(Rf.rf(3),n,1);repmat(Rf.rf(4),n,1);r
epmat(Rf.rf(5),n,1);repmat(Rf.rf(6),n,1);repmat(Rf.rf(7),n,1);repmat(Rf.rf(8),n,1);rep
mat(Rf.rf(9),n,1);repmat(Rf.rf(10),n,1);repmat(Rf.rf(11),n,1)];
priceindex_smallhouses.rf = priceindex_smallhouses.rf./4;

% 6. COMPUTATION OF MEAN PRICES FOR HOUSES AND PRICE INDEX BASED ON MEAN PRICES
% Calculate mean prices for houses
priceindex_smallhouses.mean_prices = zeros(height(priceindex_smallhouses), end);
for i = 1:length(priceindex_smallhouses.Q_order)
    priceindex_smallhouses.mean_prices(i) =
mean(price_smallhouses.estimated_price(find(price_smallhouses.Q ==
priceindex_smallhouses.Q_order(i))));
end

% Calculate price index for houses based on mean prices
priceindex_smallhouses.index_values = zeros(height(priceindex_smallhouses), end);
```

```
for i = 1:length(priceindex_smallhouses.index_values)
    priceindex_smallhouses.index_values(i) =
priceindex_smallhouses.mean_prices(i)/priceindex_smallhouses.mean_prices(1);
end

% 7. COMPUTATION OF QUARTERLY RETURNS ON ASSET/HOUSE FROM PERIOD TO PERIOD, NO COSTS
% Compute quarterly returns for each unique ID
price_smallhouses.Q_ret_no_costs =
(price_smallhouses.estimated_price./price_smallhouses.lagged_estimated_price)-1;
price_smallhouses.Q_ret_no_costs(1:44:end) = NaN;

% Compute mean quarterly returns for houses (mean of all IDs per quarter) (no costs)
priceindex_smallhouses.Q_ret_no_costs = zeros(height(priceindex_smallhouses), end);

for i = 1:height(priceindex_smallhouses)
    priceindex_smallhouses.Q_ret_no_costs(i) =
nanmean(price_smallhouses.Q_ret_no_costs(find(price_smallhouses.Q ==
priceindex_smallhouses.Q_order(i))));
end

% Compute mean of mean quarterly returns for houses (no costs)
mean_Q_ret_no_costs = nanmean(priceindex_smallhouses.Q_ret_no_costs);

% Compute excess quarterly returns for houses (no costs)
priceindex_smallhouses.excess_Q_ret_no_costs = priceindex_smallhouses.Q_ret_no_costs-
priceindex_smallhouses.rf;

% Compute mean excess quarterly returns for houses (no costs)
mean_excess_Q_ret_no_costs = nanmean(priceindex_smallhouses.excess_Q_ret_no_costs);

% Compute standard deviation for each unique ID
st_Q_ret_no_costs= zeros(length(C),1);
time(1,:) = clock;
for i = 1:length(C)
    if b(i)<b(length(C))
       st_Q_ret_no_costs(i) = std(price_smallhouses.Q_ret_no_costs(((b(i)+1):(b(i+1)-
1))));
    else
       st_Q_ret_no_costs(i) =
std(price_smallhouses.Q_ret_no_costs((b(length(C))+1):height(price_smallhouses)));
    end
time(i+1,:) = clock;
fprintf('\nTo complete %.f/%.f iterations (unique IDs) it took %.2f seconds\n', i,
length(C), etime(time(i+1,:), time(1,:)));
end

% Standard deviation of all quarterly returns
std_Q_ret_no_costs = mean(st_Q_ret_no_costs);

% Import stats for 'Q_ret_no_costs' in summary table
sum_stats_smallhouses.House_Indices(1) = 'Q_ret_no_costs';
sum_stats_smallhouses.Return(1)        = mean_Q_ret_no_costs;
sum_stats_smallhouses.std_dev(1)       = std_Q_ret_no_costs;
sum_stats_smallhouses.Excess_return(1) = mean_excess_Q_ret_no_costs;
sum_stats_smallhouses.Sharpe_ratio(1) =
sum_stats_smallhouses.Excess_return(1)/sum_stats_smallhouses.std_dev(1);
sum_stats_smallhouses.Skewness(1)      =
skewness(priceindex_smallhouses.Q_ret_no_costs(2:44));
sum_stats_smallhouses.Kurtosis(1)      =
kurtosis(priceindex_smallhouses.Q_ret_no_costs(2:44));

% COMPUTE CUMULATIVE ROA(NC)
first_obs_price = price_smallhouses.estimated_price(1:44:end);
price_smallhouses.first_obs_price_repeated = repelem(first_obs_price, 44);

price_smallhouses.cum_Q_ret_no_costs =
(price_smallhouses.estimated_price./price_smallhouses.first_obs_price_repeated)-1;
price_smallhouses.cum_Q_ret_no_costs(1:44:end) = NaN;

priceindex_smallhouses.cum_Q_ret_no_costs = zeros(height(priceindex_smallhouses),end);
% Compute mean cumulative ROA(NC)
for i = 1:height(priceindex_smallhouses)
```

```
    priceindex_smallhouses.cum_Q_ret_no_costs(i) =
nanmean(price_smallhouses.cum_Q_ret_no_costs(find(price_smallhouses.Q ==
priceindex_smallhouses.Q_order(i))));
end

% 8. COMPUTATION OF QUARTERLY RETURNS ON ASSET/HOUSE FROM PERIOD TO PERIOD (BUY/SELL),
INCL TOTAL COSTS
% Compute quarterly returns for each unique ID
price_smallhouses.Q_ret_tot_costs = (((price_smallhouses.estimated_price -
price_smallhouses.t_cost_s - price_smallhouses.lagged_t_cost -
price_smallhouses.lagged_estimated_price).*(1-price_smallhouses.tax_rates)) -
price_smallhouses.living_exp +
price_smallhouses.implicit_rent)./(price_smallhouses.lagged_estimated_price);
price_smallhouses.Q_ret_tot_costs(1:44:end) = NaN;

% Compute mean quarterly returns for houses (mean of all IDs per quarter) (inc tot
costs)
priceindex_smallhouses.Q_ret_tot_costs = zeros(height(priceindex_smallhouses), end);

for i = 1:length(priceindex_smallhouses.mean_prices)
    priceindex_smallhouses.Q_ret_tot_costs(i) =
mean(price_smallhouses.Q_ret_tot_costs(find(price_smallhouses.Q ==
priceindex_smallhouses.Q_order(i))));
end

% Compute mean of mean quarterly returns for houses (inc tot costs)
mean_Q_ret_tot_costs = nanmean(priceindex_smallhouses.Q_ret_tot_costs);

% Compute excess quarterly returns for houses (inc tot costs)
priceindex_smallhouses.excess_Q_ret_tot_costs =
priceindex_smallhouses.Q_ret_tot_costs-priceindex_smallhouses.rf;

% Compute mean excess quarterly returns for houses (inc tot costs)
mean_excess_Q_ret_tot_costs = nanmean(priceindex_smallhouses.excess_Q_ret_tot_costs);

% Compute standard deviation for each unique IDs returns
st_Q_ret_tot_costs= zeros(length(C),1);
time(1,:) = clock;
for i = 1:length(C)
    if b(i)<b(length(C))
       st_Q_ret_tot_costs(i) =
std(price_smallhouses.Q_ret_tot_costs(((b(i)+1):(b(i+1)-1))));
    else
       st_Q_ret_tot_costs(i) =
std(price_smallhouses.Q_ret_tot_costs((b(length(C))+1):height(price_smallhouses)));
    end
time(i+1,:) = clock;
fprintf('\nTo complete %.f/%.f iterations (unique IDs) it took %.2f seconds\n', i,
length(C), etime(time(i+1,:), time(1,:)));
end

% Standard deviation of all quarterly returns
std_Q_ret_tot_costs = mean(st_Q_ret_tot_costs);

% Import stats for 'Q_ret_tot_costs' in summary table
sum_stats_smallhouses.House_Indices(2) = 'Q_ret_tot_costs';
sum_stats_smallhouses.Return(2)        = mean_Q_ret_tot_costs;
sum_stats_smallhouses.std_dev(2)       = std_Q_ret_tot_costs;
sum_stats_smallhouses.Excess_return(2) = mean_excess_Q_ret_tot_costs;
sum_stats_smallhouses.Sharpe_ratio(2)  =
sum_stats_smallhouses.Excess_return(2)/sum_stats_smallhouses.std_dev(2);
sum_stats_smallhouses.Skewness(2)      =
skewness(priceindex_smallhouses.Q_ret_tot_costs(2:44));
sum_stats_smallhouses.Kurtosis(2)      =
kurtosis(priceindex_smallhouses.Q_ret_tot_costs(2:44));

% COMPUTE CUMULATIVE ROA (TC)
taxes_TC = (price_smallhouses.estimated_price - price_smallhouses.t_cost_s -
price_smallhouses.lagged_t_cost -
price_smallhouses.lagged_estimated_price).*price_smallhouses.tax_rates;
taxes_TC(1:44:end) = NaN;

net_costs_TC = (price_smallhouses.living_exp + price_smallhouses.lagged_t_cost +
price_smallhouses.t_cost_s + taxes_TC - price_smallhouses.implicit_rent);
```

```
net_costs_TC(1:44:end) = NaN;

% Compute cumulative costs
cumcosts_TC = zeros(height(price_smallhouses), end);
time(1,:) = clock;
for i = 1:length(C)
    if b(i)<b(length(C))
       cumcosts_TC((b(i)):(b(i+1)-1)) = cumsum(net_costs_TC((b(i)):(b(i+1)-
1)),'omitnan');
    else
       cumcosts_TC(b(length(C)):height(price_smallhouses)) =
cumsum(net_costs_TC(b(length(C)):height(price_smallhouses)),'omitnan');
    end
time(i+1,:) = clock;
fprintf('\nTo complete %.f/%.f iterations (unique IDs) it took %.2f seconds\n', i,
length(C), etime(time(i+1,:), time(1,:)));
end

price_smallhouses.cum_Q_ret_tot_costs = (price_smallhouses.estimated_price -
cumcosts_TC -
price_smallhouses.first_obs_price_repeated)./price_smallhouses.first_obs_price_repeate
d;
price_smallhouses.cum_Q_ret_tot_costs(1:44:end) = NaN;

priceindex_smallhouses.cum_Q_ret_tot_costs =
zeros(height(priceindex_smallhouses),end);
% Compute mean cumulative returns for houses (mean of all IDs per quarter) (no costs)
for i = 1:height(priceindex_smallhouses)
    priceindex_smallhouses.cum_Q_ret_tot_costs(i) =
nanmean(price_smallhouses.cum_Q_ret_tot_costs(find(price_smallhouses.Q ==
priceindex_smallhouses.Q_order(i))));
end

all_lost = priceindex_smallhouses.cum_Q_ret_tot_costs < -1;
priceindex_smallhouses.cum_Q_ret_tot_costs(all_lost) = -1;

% 9. COMPUTATION OF QUARTERLY RETURNS ON ASSET/HOUSE FROM PERIOD TO PERIOD , INCL
SMOOTHED COSTS
% Locate buyer and sellers cost when buying in 2010Q1 and selling in 2020Q4
% Find buyer and seller cost per quarter for sample period
cost_2010Q1 = price_smallhouses.t_cost(1:44:end);
cost_2010Q1 = cost_2010Q1/44;
price_smallhouses.cost_2010Q1 = repelem(cost_2010Q1, 44);

cost_2020Q4 = price_smallhouses.t_cost_s(44:44:end);
cost_2020Q4 = cost_2020Q4/44;
price_smallhouses.cost_2020Q4 = repelem(cost_2020Q4, 44);

% Compute quarterly returns given that you buy in 2010Q1 and sell in 2020Q4
% and spred the relevant costs on the sample period.
price_smallhouses.Q_ret_smooth_costs = (price_smallhouses.estimated_price -
price_smallhouses.cost_2010Q1 - price_smallhouses.cost_2020Q4 -
price_smallhouses.living_exp - price_smallhouses.lagged_estimated_price +
price_smallhouses.implicit_rent)./price_smallhouses.lagged_estimated_price;
price_smallhouses.Q_ret_smooth_costs(1:44:end) = NaN;

% Compute mean quarterly returns for houses (mean of all IDs per quarter) (inc smooth
costs)
priceindex_smallhouses.Q_ret_smooth_costs = zeros(height(priceindex_smallhouses),
end);

for i = 1:length(priceindex_smallhouses.mean_prices)
    priceindex_smallhouses.Q_ret_smooth_costs(i) =
mean(price_smallhouses.Q_ret_smooth_costs(find(price_smallhouses.Q ==
priceindex_smallhouses.Q_order(i))));
end

% Compute mean of mean quarterly returns for houses (inc smooth costs)
mean_Q_ret_smooth_costs = nanmean(priceindex_smallhouses.Q_ret_smooth_costs);

% Compute excess quarterly returns for houses (inc smooth costs)
priceindex_smallhouses.excess_Q_ret_smooth_costs =
priceindex_smallhouses.Q_ret_smooth_costs - priceindex_smallhouses.rf;
```

```
% Compute mean excess quarterly returns for houses (inc smooth costs)
mean_excess_Q_ret_smooth_costs =
nanmean(priceindex_smallhouses.excess_Q_ret_smooth_costs);

% Compute standard deviation for each unique IDs returns
st_Q_ret_smooth_costs= zeros(length(C),1);
time(1,:) = clock;
for i = 1:length(C)
    if b(i)<b(length(C))
       st_Q_ret_smooth_costs(i) =
std(price_smallhouses.Q_ret_smooth_costs(((b(i)+1):(b(i+1)-1))));
    else
       st_Q_ret_smooth_costs(i) =
std(price_smallhouses.Q_ret_smooth_costs((b(length(C))+1):height(price_smallhouses)));
    end
time(i+1,:) = clock;
fprintf('\nTo complete %.f/%.f iterations (unique IDs) it took %.2f seconds\n', i,
length(C), etime(time(i+1,:), time(1,:)));
end

% Standard deviation of all quarterly returns
std_Q_ret_smooth_costs = mean(st_Q_ret_smooth_costs);

% Import stats for 'Q_ret_smooth_costs' in summary table
sum_stats_smallhouses.House_Indices(3) = 'Q_ret_smooth_costs';
sum_stats_smallhouses.Return(3)        = mean_Q_ret_smooth_costs;
sum_stats_smallhouses.std_dev(3)       = std_Q_ret_smooth_costs;
sum_stats_smallhouses.Excess_return(3) = mean_excess_Q_ret_smooth_costs;
sum_stats_smallhouses.Sharpe_ratio(3) =
sum_stats_smallhouses.Excess_return(3)/sum_stats_smallhouses.std_dev(3);
sum_stats_smallhouses.Skewness(3)      =
skewness(priceindex_smallhouses.Q_ret_smooth_costs(2:44));
sum_stats_smallhouses.Kurtosis(3)      =
kurtosis(priceindex_smallhouses.Q_ret_smooth_costs(2:44));

% COMPUTE CUMULATIVE ROA(SC)
net_costs_SC = (price_smallhouses.living_exp + price_smallhouses.cost_2010Q1 +
price_smallhouses.cost_2020Q4 - price_smallhouses.implicit_rent);
net_costs_SC(1:44:end) = NaN;

cumcosts_SC = zeros(height(price_smallhouses), end);
time(1,:) = clock;
for i = 1:length(C)
    if b(i)<b(length(C))
       cumcosts_SC((b(i)):(b(i+1)-1)) = cumsum(net_costs_SC((b(i)):(b(i+1)-
1)),'omitnan');
    else
       cumcosts_SC(b(length(C)):height(price_smallhouses)) =
cumsum(net_costs_SC(b(length(C)):height(price_smallhouses)),'omitnan');
    end
time(i+1,:) = clock;
fprintf('\nTo complete %.f/%.f iterations (unique IDs) it took %.2f seconds\n', i,
length(C), etime(time(i+1,:), time(1,:)));
end

price_smallhouses.cum_Q_ret_smooth_costs = (price_smallhouses.estimated_price -
cumcosts_SC -
price_smallhouses.first_obs_price_repeated)./price_smallhouses.first_obs_price_repeate
d;
price_smallhouses.cum_Q_ret_smooth_costs(1:44:end) = NaN;

% Compute mean cumulative returns for houses (mean of all IDs per quarter) (no costs)
priceindex_smallhouses.cum_Q_ret_smooth_costs =
zeros(height(priceindex_smallhouses),end);
for i = 1:height(priceindex_smallhouses)
    priceindex_smallhouses.cum_Q_ret_smooth_costs(i) =
nanmean(price_smallhouses.cum_Q_ret_smooth_costs(find(price_smallhouses.Q ==
priceindex_smallhouses.Q_order(i))));
end

% Identify average loan-to-value ratio for housing investments
l = ([0.6957, 0.6957, 0.6957, 0.6957, 0.71, 0.76, 0.67, 0.66, 0.68, 0.70, 0.69])';
n = 4;
```

```
ltv_ratio_order =
[repmat(l(1),n,1);repmat(l(2),n,1);repmat(l(3),n,1);repmat(l(4),n,1);repmat(l(5),n,1);
repmat(l(6),n,1);repmat(l(7),n,1);repmat(l(8),n,1);repmat(l(9),n,1);repmat(l(10),n,1);
repmat(l(11),n,1)];

% Allocate correct ltv-rates to each quarter in price_houses set
price_smallhouses.ltvr        = repmat(ltv_ratio_order,(length(C)),1);
price_smallhouses.lagged_ltvr = lagmatrix(price_smallhouses.ltvr,1);

% Allocate ltv ratio as of 2010Q1 to each quarter in price_houses set
price_smallhouses.ltvr_2010Q1(:) = price_smallhouses.ltvr(1);

% 10. WE DONT USE THE RETURN SERIES FROM SECTION 10
% Compute quarterly equity returns for each unique ID (no costs)
price_smallhouses.Q_eq_ret_no_costs = ((price_smallhouses.estimated_price -
price_smallhouses.lagged_estimated_price)./(price_smallhouses.lagged_estimated_price.*
(1-price_smallhouses.lagged_ltvr)));
price_smallhouses.Q_eq_ret_no_costs(1:44:end) = NaN;

% Compute mean quarterly equity returns for houses (mean of all IDs per quarter) (no
costs)
priceindex_smallhouses.Q_eq_ret_no_costs = zeros(height(priceindex_smallhouses), end);

for i = 1:height(priceindex_smallhouses)
    priceindex_smallhouses.Q_eq_ret_no_costs(i) =
mean(price_smallhouses.Q_eq_ret_no_costs(find(price_smallhouses.Q ==
priceindex_smallhouses.Q_order(i))));
end

% Compute mean of mean quarterly equity returns for houses (no costs)
mean_Q_eq_ret_no_costs = nanmean(priceindex_smallhouses.Q_eq_ret_no_costs);

% Compute excess quarterly equity returns for houses (no costs)
priceindex_smallhouses.excess_Q_eq_ret_no_costs =
priceindex_smallhouses.Q_eq_ret_no_costs - priceindex_smallhouses.rf;

% Compute mean excess quarterly equity returns for houses (no costs)
mean_excess_Q_eq_ret_no_costs =
nanmean(priceindex_smallhouses.excess_Q_eq_ret_no_costs);

% Compute standard deviation for each unique IDs returns
st_Q_eq_ret_no_costs= zeros(length(C),1);
time(1,:) = clock;
for i = 1:length(C)
    if b(i)<b(length(C))
       st_Q_eq_ret_no_costs(i) =
std(price_smallhouses.Q_eq_ret_no_costs(((b(i)+1):(b(i+1)-1))));
    else
       st_Q_eq_ret_no_costs(i) =
std(price_smallhouses.Q_eq_ret_no_costs((b(length(C))+1):height(price_smallhouses)));
    end
time(i+1,:) = clock;
fprintf('\nTo complete %.f/%.f iterations (unique IDs) it took %.2f seconds\n', i,
length(C), etime(time(i+1,:), time(1,:)));
end

% Standard deviation of all quarterly equity returns
std_Q_eq_ret_no_costs = mean(st_Q_eq_ret_no_costs);

% Import stats for 'Q_eq_ret_no_costs' in summary table
sum_stats_smallhouses.House_Indices(4) = 'Q_eq_ret_no_costs';
sum_stats_smallhouses.Return(4)        = mean_Q_eq_ret_no_costs;
sum_stats_smallhouses.std_dev(4)       = std_Q_eq_ret_no_costs;
sum_stats_smallhouses.Excess_return(4) = mean_excess_Q_eq_ret_no_costs;
sum_stats_smallhouses.Sharpe_ratio(4) =
sum_stats_smallhouses.Excess_return(4)/sum_stats_smallhouses.std_dev(4);
sum_stats_smallhouses.Skewness(4)      =
skewness(priceindex_smallhouses.Q_eq_ret_no_costs(2:44));
sum_stats_smallhouses.Kurtosis(4)      =
kurtosis(priceindex_smallhouses.Q_eq_ret_no_costs(2:44));

% COMPUTE CUMULATIVE ROE (NC)
```

```
price_smallhouses.cum_Q_eq_ret_no_costs = ((price_smallhouses.estimated_price -
price_smallhouses.first_obs_price_repeated)./(price_smallhouses.first_obs_price_repeat
ed.*(1-price_smallhouses.ltvr_2010Q1)));
price_smallhouses.cum_Q_eq_ret_no_costs(1:44:end) = NaN;

% Compute mean cumulative returns for houses (mean of all IDs per quarter) (no costs)
priceindex_smallhouses.cum_Q_eq_ret_no_costs =
zeros(height(priceindex_smallhouses),end);
for i = 1:height(priceindex_smallhouses)
    priceindex_smallhouses.cum_Q_eq_ret_no_costs(i) =
nanmean(price_smallhouses.cum_Q_eq_ret_no_costs(find(price_smallhouses.Q ==
priceindex_smallhouses.Q_order(i))));
end

% 11. WE DONT USE THE RETURN SERIES FROM SECTION 11
% Compute quarterly equity returns for each unique ID (inc tot costs)
price_smallhouses.Q_eq_ret_tot_costs = (((price_smallhouses.estimated_price -
price_smallhouses.t_cost_s - price_smallhouses.lagged_t_cost -
price_smallhouses.lagged_estimated_price).*(1-price_smallhouses.tax_rates)) -
price_smallhouses.living_exp +
price_smallhouses.implicit_rent)./(price_smallhouses.lagged_estimated_price.*(1-
price_smallhouses.lagged_ltvr));
price_smallhouses.Q_eq_ret_tot_costs(1:44:end) = NaN;

% Compute mean quarterly equity returns for houses (mean of all IDs per quarter) (inc
tot costs)
priceindex_smallhouses.Q_eq_ret_tot_costs = zeros(height(priceindex_smallhouses),
end);

for i = 1:length(priceindex_smallhouses.mean_prices)
    priceindex_smallhouses.Q_eq_ret_tot_costs(i) =
mean(price_smallhouses.Q_eq_ret_tot_costs(find(price_smallhouses.Q ==
priceindex_smallhouses.Q_order(i))));
end

% Compute mean of mean quarterly equity returns for houses (inc tot costs)
mean_Q_eq_ret_tot_costs = nanmean(priceindex_smallhouses.Q_eq_ret_tot_costs);

% Compute excess quarterly equity returns for houses (inc tot costs)
priceindex_smallhouses.excess_Q_eq_ret_tot_costs =
priceindex_smallhouses.Q_eq_ret_tot_costs - priceindex_smallhouses.rf;

% Compute mean excess quarterly equity returns for houses (inc tot costs)
mean_excess_Q_eq_ret_tot_costs =
nanmean(priceindex_smallhouses.excess_Q_eq_ret_tot_costs);

% Compute standard deviation for each unique IDs returns
st_Q_eq_ret_tot_costs= zeros(length(C),1);
time(1,:) = clock;
for i = 1:length(C)
    if b(i)<b(length(C))
       st_Q_eq_ret_tot_costs(i) =
std(price_smallhouses.Q_eq_ret_tot_costs(((b(i)+1):(b(i+1)-1))));
    else
       st_Q_eq_ret_tot_costs(i) =
std(price_smallhouses.Q_eq_ret_tot_costs((b(length(C))+1):height(price_smallhouses)));
    end
time(i+1,:) = clock;
fprintf('\nTo complete %.f/%.f iterations (unique IDs) it took %.2f seconds\n', i,
length(C), etime(time(i+1,:), time(1,:)));
end

% Standard deviation of all quarterly equity returns
std_Q_eq_ret_tot_costs = mean(st_Q_eq_ret_tot_costs);

% Import stats for 'Q_eq_ret_tot_costs' in summary table
sum_stats_smallhouses.House_Indices(5) = 'Q_eq_ret_tot_costs';
sum_stats_smallhouses.Return(5)        = mean_Q_eq_ret_tot_costs;
sum_stats_smallhouses.std_dev(5)       = std_Q_eq_ret_tot_costs;
sum_stats_smallhouses.Excess_return(5) = mean_excess_Q_eq_ret_tot_costs;
sum_stats_smallhouses.Sharpe_ratio(5) =
sum_stats_smallhouses.Excess_return(5)/sum_stats_smallhouses.std_dev(5);
sum_stats_smallhouses.Skewness(5)      =
skewness(priceindex_smallhouses.Q_eq_ret_tot_costs(2:44));
```

```
sum_stats_smallhouses.Kurtosis(5)      =
kurtosis(priceindex_smallhouses.Q_eq_ret_tot_costs(2:44));

% COMPUTE CUMULATIVE RETURNS BASED ON QUARTERLY EQUITY RETURNS FROM PRICE INDEX (INC
TOT COSTS)
priceindex_smallhouses.cum_Q_eq_ret_tot_costs =
zeros(height(priceindex_smallhouses),end);
priceindex_smallhouses.cum_Q_eq_ret_tot_costs =
cumprod((1+priceindex_smallhouses.Q_eq_ret_tot_costs),'omitnan') -1;

% 12. COMPUTATION OF QUARTERLY EQUITY RETURNS ON ASSET/HOUSE FROM PERIOD TO PERIOD ,
INCL SMOOTHED COSTS
% Compute quarterly EQUITY returns given that you buy in 2010Q1 and sell in 2020Q4
% and spread the relevant costs on the sample period.
price_smallhouses.Q_eq_ret_smooth_costs = (price_smallhouses.estimated_price -
price_smallhouses.cost_2010Q1 - price_smallhouses.cost_2020Q4 -
price_smallhouses.living_exp - price_smallhouses.lagged_estimated_price +
price_smallhouses.implicit_rent)./(price_smallhouses.lagged_estimated_price.*(1-
price_smallhouses.ltvr_2010Q1));
price_smallhouses.Q_eq_ret_smooth_costs(1:44:end) = NaN;

% Compute mean quarterly equity returns for houses (mean of all IDs per quarter) (inc
smooth costs)
priceindex_smallhouses.Q_eq_ret_smooth_costs = zeros(height(priceindex_smallhouses),
end);

for i = 1:length(priceindex_smallhouses.mean_prices)
    priceindex_smallhouses.Q_eq_ret_smooth_costs(i) =
mean(price_smallhouses.Q_eq_ret_smooth_costs(find(price_smallhouses.Q ==
priceindex_smallhouses.Q_order(i))));
end

% Compute mean of mean quarterly equity returns for houses (inc smooth costs)
mean_Q_eq_ret_smooth_costs = nanmean(priceindex_smallhouses.Q_eq_ret_smooth_costs);

% Compute excess quarterly equity returns for houses (inc smooth costs)
priceindex_smallhouses.excess_Q_eq_ret_smooth_costs =
priceindex_smallhouses.Q_eq_ret_smooth_costs - priceindex_smallhouses.rf;

% Compute mean excess quarterly equity returns for houses (inc smooth costs)
mean_excess_Q_eq_ret_smooth_costs =
nanmean(priceindex_smallhouses.excess_Q_eq_ret_smooth_costs);

% Compute standard deviation for each unique IDs returns
st_Q_eq_ret_smooth_costs= zeros(length(C),1);
time(1,:) = clock;
for i = 1:length(C)
    if b(i)<b(length(C))
       st_Q_eq_ret_smooth_costs(i) =
std(price_smallhouses.Q_eq_ret_smooth_costs(((b(i)+1):(b(i+1)-1))));
    else
       st_Q_eq_ret_smooth_costs(i) =
std(price_smallhouses.Q_eq_ret_smooth_costs((b(length(C))+1):height(price_smallhouses)
));
    end
time(i+1,:) = clock;
fprintf('\nTo complete %.f/%.f iterations (unique IDs) it took %.2f seconds\n', i,
length(C), etime(time(i+1,:), time(1,:)));
end

% Standard deviation of all quarterly equity returns
std_Q_eq_ret_smooth_costs = mean(st_Q_eq_ret_smooth_costs);

% Import stats for 'Q_eq_ret_smooth_costs' in summary table
sum_stats_smallhouses.House_Indices(6) = 'Q_eq_ret_smooth_costs';
sum_stats_smallhouses.Return(6)        = mean_Q_eq_ret_smooth_costs;
sum_stats_smallhouses.std_dev(6)       = std_Q_eq_ret_smooth_costs;
sum_stats_smallhouses.Excess_return(6) = mean_excess_Q_eq_ret_smooth_costs;
sum_stats_smallhouses.Sharpe_ratio(6) =
sum_stats_smallhouses.Excess_return(6)/sum_stats_smallhouses.std_dev(6);
sum_stats_smallhouses.Skewness(6)      =
skewness(priceindex_smallhouses.Q_eq_ret_smooth_costs(2:44));
sum_stats_smallhouses.Kurtosis(6)      =
kurtosis(priceindex_smallhouses.Q_eq_ret_smooth_costs(2:44));
```

```
% COMPUTE CUMULATIVE ROE(SC)
price_smallhouses.cum_Q_eq_ret_smooth_costs = (price_smallhouses.estimated_price -
cumcosts_SC -
price_smallhouses.first_obs_price_repeated)./(price_smallhouses.first_obs_price_repeat
ed.*(1-price_smallhouses.ltvr_2010Q1));
price_smallhouses.cum_Q_eq_ret_smooth_costs(1:44:end) = NaN;

% Compute mean cumulative returns for houses (mean of all IDs per quarter) (smooth
costs)
priceindex_smallhouses.cum_Q_eq_ret_smooth_costs =
zeros(height(priceindex_smallhouses),end);
for i = 1:height(priceindex_smallhouses)
    priceindex_smallhouses.cum_Q_eq_ret_smooth_costs(i) =
nanmean(price_smallhouses.cum_Q_eq_ret_smooth_costs(find(price_smallhouses.Q ==
priceindex_smallhouses.Q_order(i))));
end

% MAKE PLOT OF THE RELEVANT INDICES TOGHETER, MAKE MULTIPLE PLOTS FOR COMPARISON
REASONS
smallhouse_price_changes = priceindex_smallhouses(:,[1 5 8 11 14 17 20]);

smallhouse_price_indices = priceindex_smallhouses(:,[1 7 10 13 16 19 22]);
smallhouse_price_indices.cum_Q_ret_no_costs(1) = 0;
smallhouse_price_indices.cum_Q_ret_tot_costs(1) = 0;
smallhouse_price_indices.cum_Q_ret_smooth_costs(1) = 0;
smallhouse_price_indices.cum_Q_eq_ret_no_costs(1) = 0;
smallhouse_price_indices.cum_Q_eq_ret_tot_costs(1) = 0;
smallhouse_price_indices.cum_Q_eq_ret_smooth_costs(1) = 0;

smallhouse_price_indices.cum_Q_ret_no_costs        =
smallhouse_price_indices.cum_Q_ret_no_costs + 1;
smallhouse_price_indices.cum_Q_ret_tot_costs       =
smallhouse_price_indices.cum_Q_ret_tot_costs + 1;
smallhouse_price_indices.cum_Q_ret_smooth_costs    =
smallhouse_price_indices.cum_Q_ret_smooth_costs + 1;
smallhouse_price_indices.cum_Q_eq_ret_no_costs     =
smallhouse_price_indices.cum_Q_eq_ret_no_costs + 1;
smallhouse_price_indices.cum_Q_eq_ret_tot_costs    =
smallhouse_price_indices.cum_Q_eq_ret_tot_costs + 1;
smallhouse_price_indices.cum_Q_eq_ret_smooth_costs =
smallhouse_price_indices.cum_Q_eq_ret_smooth_costs + 1;

clearvars -except Property_Data summary_table Rf Rent_Data...
                  house_set hedonic_houses housing_model...
                  price_houses uniq_house_set avg_cost_rates_houses...
                  priceindex_houses sum_stats_houses houseprice_changes...
                  houseprice_indices...
                  smallhouse_set hedonic_smallhouses smallhouse_model...
                  price_smallhouses uniq_smallhouse_set avg_cost_rates_smallhouses...
                  priceindex_smallhouses sum_stats_smallhouses...
                  smallhouse_price_changes smallhouse_price_indices...
                  apartment_set hedonic_apartments apartment_model price_apartments...
                  uniq_apartment_set avg_cost_rates_apartments
priceindex_apartments...
                  sum_stats_apartments apartment_price_changes...
                  apartment_price_indices...
                  inner_eastern_set hedonic_inner_eastern inner_eastern_model...
                  price_inner_eastern uniq_inner_eastern_set
avg_cost_rates_inner_eastern...
                  priceindex_inner_eastern sum_stats_inner_eastern
inner_eastern_price_changes...
                  inner_eastern_price_indices...
                  inner_western_set hedonic_inner_western inner_western_model...
                  price_inner_western uniq_inner_western_set
avg_cost_rates_inner_western...
                  priceindex_inner_western sum_stats_inner_western
inner_western_price_changes...
                  inner_western_price_indices...
                  outer_eastern_set hedonic_outer_eastern outer_eastern_model...
                  price_outer_eastern uniq_outer_eastern_set
avg_cost_rates_outer_eastern...
                  priceindex_outer_eastern sum_stats_outer_eastern
outer_eastern_price_changes...
```

```
                outer_eastern_price_indices...
                outer_western_set hedonic_outer_western outer_western_model...
                price_outer_western uniq_outer_western_set
avg_cost_rates_outer_western...
                priceindex_outer_western sum_stats_outer_western
outer_western_price_changes...
                outer_western_price_indices...
                outer_southern_set hedonic_outer_southern outer_southern_model...
                price_outer_southern uniq_outer_southern_set
avg_cost_rates_outer_southern...
                priceindex_outer_southern sum_stats_outer_southern
outer_southern_price_changes...
                outer_southern_price_indices...
                property_price_changes property_price_indices hedonic_total...
                ROA_NC_stats ROE_NC_stats ROA_TC_stats ROE_TC_stats...
                ROA_SC_stats ROE_SC_stats sum_stats_OSEFX OSEFX...
                ROA_NC_sharpe_test ROA_TC_sharpe_test ROA_SC_sharpe_test...
                ROE_SC_sharpe_test

%% Regression model for Inner Eastern
% 1. PREPERATION OF DATASET
inner_eastern      = (apartment_set.Bydel == 'Gamle Oslo') + (apartment_set.Bydel ==
'Gr√°nerl√∏kka') + (apartment_set.Bydel == 'Sagene');
inner_eastern      = logical(inner_eastern);
inner_eastern_set  = apartment_set(inner_eastern,:);

% Make categorical for dummy-purposes
inner_eastern_set.Alder    = categorical(inner_eastern_set.Alder);
inner_eastern_set.Q        = categorical(inner_eastern_set.Q);
inner_eastern_set.Eierform = categorical(inner_eastern_set.Eierform);
inner_eastern_set.Etasje   = categorical(inner_eastern_set.Etasje);

% Reorder categories/dummies to find reference variables
inner_eastern_set.Alder    = reordercats(inner_eastern_set.Alder,    {'Age1', 'Age2',
'Age3', 'Age4'});
inner_eastern_set.Eierform = reordercats(inner_eastern_set.Eierform, {'Selveier',
'Borettslag'});
inner_eastern_set.Etasje   = reordercats(inner_eastern_set.Etasje,   {'1', '2', '3',
'4', '5', '6', '7', '8', '9', '10', '11', '12', '13', '14', '15', '-1', '-2', '-3', '-
4'});

% Remove unused categories
inner_eastern_set.Eierform = removecats(inner_eastern_set.Eierform);

% Check that we in fact removed unused categories
categories(inner_eastern_set.Alder)
categories(inner_eastern_set.Eierform)

% Make log prices and sizes
inner_eastern_set.log_pris = log(inner_eastern_set.Pris);
inner_eastern_set.log_prom = log(inner_eastern_set.prom);

% 2. Run regression - Log model with log(prom), Q, Bydel, Soverom, Alder on log(pris)
% Explanatory variables log(prom), soverom - continous
% Explanatory variables Q, Bydel, Eierform, Alder and Etasje - dummies
inner_eastern_model =
fitlm(inner_eastern_set,'log_pris~log_prom+Q+Eierform+Soverom+Alder+Etasje');

% 3. ALLOCATE TABLE FOR ESTIMATED PRICES (BASED ON MODEL FROM PT. 2) FOR EVERY QUARTER
FOR EACH ID
inner_eastern_set   = sortrows(inner_eastern_set,'BoligID','ascend');
[C,ia,ic]          = unique(inner_eastern_set.BoligID,'first');
price_inner_eastern = zeros(length(C)*44, width(inner_eastern_set));
price_inner_eastern = array2table(price_inner_eastern);

price_inner_eastern.Properties.VariableNames = {'BoligID', 'Salgsdato', 'Bydel',
'Pris', 'Fellesgjeld', 'Prisantydning', 'Eierform', 'Byggeaar', 'prom', 'Etasje',
'Soverom', 'Boligtype', 'year', 'Q', 'Alder', 'kvmpris', 'log_pris', 'log_prom' };

Q_rep   = repmat(["Q1"; "Q2"; "Q3";"Q4"],(length(C)*44)/4,1);
x       = (2010:2020)';
n       = 4;
newx    =
[repmat(x(1),n,1);repmat(x(2),n,1);repmat(x(3),n,1);repmat(x(4),n,1);repmat(x(5),n,1);
```

```
repmat(x(6),n,1);repmat(x(7),n,1);repmat(x(8),n,1);repmat(x(9),n,1);repmat(x(10),n,1);
repmat(x(11),n,1)];
Y_order = repmat(newx,(length(C)),1);

price_inner_eastern.Q_order = string(Y_order(:,1)) + Q_rep(:,1);
price_inner_eastern.Q       = price_inner_eastern.Q_order;
price_inner_eastern.Q_order = [];

price_inner_eastern.BoligID = categorical(price_inner_eastern.BoligID);
price_inner_eastern.BoligID = repelem(C, 44);

price_inner_eastern           = convertvars(price_inner_eastern,{'BoligID', 'Bydel',
'Eierform', 'Boligtype', 'Etasje', 'Alder', 'Q'},'categorical');
price_inner_eastern.Salgsdato =
datetime(price_inner_eastern.Salgsdato,'ConvertFrom','yyyymmdd');

b = 1:length(newx):height(price_inner_eastern);

% Create set containing only the unique observations
uniq_inner_eastern_set = inner_eastern_set(ia,:);

% Allocate variables into price_houses to use for regression model
% Remember to adjust according to model
price_inner_eastern(:,[3 5 7 9 10 11 15 18]) = repelem(uniq_inner_eastern_set(:,[3 5 7
9 10 11 15 18]), 44, 1);

% Store relevant variables for regression model in new table
x3 = price_inner_eastern(:,[18 14 7 11 15 10]);

% GET COLUMN OF ESTIMATED PRICES BASED ON REGRESSION MODEL
price_inner_eastern.log_estimated_price    = predict(inner_eastern_model, x3);
price_inner_eastern.estimated_price        =
exp(price_inner_eastern.log_estimated_price);
price_inner_eastern.lagged_estimated_price =
lagmatrix(price_inner_eastern.estimated_price,1);

% 4. IMPLEMENT RELEVANT COSTS RELATED TO HOUSING INVESTMENTS
% Transaction costs buyer, b
gebyr_b             = zeros(height(price_inner_eastern),1);
dokavg_b            = zeros(height(price_inner_eastern),1);
insurance_b         = zeros(height(price_inner_eastern),1);
price_inner_eastern.t_cost = zeros(height(price_inner_eastern),1);

% If type of ownership = 'Selveier'
idx_selv              = price_inner_eastern.Eierform == "Selveier";
gebyr_b(idx_selv)     = 540 + 540 + 172;
dokavg_b(idx_selv)    =
0.025.*(price_inner_eastern.estimated_price(idx_selv)+price_inner_eastern.Fellesgjeld(
idx_selv));
insurance_b(idx_selv) = 7400;

% If type of ownership = 'Borettslag'
idx_andel             = price_inner_eastern.Eierform == "Borettslag";
gebyr_b(idx_andel)    = 440 + 440 + 172;
dokavg_b(idx_andel)   = 0;
insurance_b(idx_andel) = 4100;

% Compute total transaction cost for buyer, b, for each observation/quarter
price_inner_eastern.t_cost        = gebyr_b+dokavg_b+insurance_b;
price_inner_eastern.lagged_t_cost = lagmatrix(price_inner_eastern.t_cost,1);

% Transaction costs seller, s
eierskiftegebyr_s     = zeros(height(price_inner_eastern),1);
public_fees_s         = zeros(height(price_inner_eastern),1);
insurance_s           = zeros(height(price_inner_eastern),1);
commission_s          = zeros(height(price_inner_eastern),1);
price_inner_eastern.t_cost_s = zeros(height(price_inner_eastern),1);
indices_selv          = zeros(height(price_inner_eastern),1);
indices_andel         = zeros(height(price_inner_eastern),1);

% If type of ownership = 'Selveier'
idx_selv                  = price_inner_eastern.Eierform == "Selveier";
public_fees_s(idx_selv)     = 8430;
eierskiftegebyr_s(idx_selv) = 0;
```

```
insurance_s(idx_selv)       =
max(0.0033*(price_inner_eastern.estimated_price(idx_selv)+price_inner_eastern.Fellesgj
eld(idx_selv)), 4000);
indices_selv                = insurance_s > 24000;
indices_selv                = logical(indices_selv);
insurance_s(indices_selv)   = 24000;

% If type of ownership = 'Borettslag'
idx_andel                    = price_inner_eastern.Eierform == "Borettslag";
public_fees_s(idx_andel)     = 14319;
eierskiftegebyr_s(idx_andel) = 4858;
insurance_s(idx_andel)       =
max(0.002*(price_inner_eastern.estimated_price(idx_andel)+price_inner_eastern.Fellesgj
eld(idx_andel)), 2500);
indices_andel                = zeros(height(price_inner_eastern),1);
indices_andel(idx_andel)     = insurance_s(idx_andel) > 24000;
indices_andel                = logical(indices_andel);
insurance_s(indices_andel)   = 24000;

% Commission to broker independent of type of ownership
commission_s =
max(0.02*(price_inner_eastern.estimated_price+price_inner_eastern.Fellesgjeld),40000);

% Compute total transaction cost for seller, s, for each observation/quarter
price_inner_eastern.t_cost_s =
public_fees_s+eierskiftegebyr_s+insurance_s+commission_s;

% Compute transaction cost rates for buyer and seller
% For buyer
t_cost_rate   = mean(price_inner_eastern.t_cost./price_inner_eastern.estimated_price);
t_cost_rate_Q = t_cost_rate/4;

% For seller
t_cost_s_rate   =
mean(price_inner_eastern.t_cost_s./price_inner_eastern.estimated_price);
t_cost_s_rate_Q = t_cost_s_rate/4;

% Define holding period for each dwelling
holding_period = 11;

% Define living expenses per quarter - reference apartment is 70 sqm
liv_exp_a_70_Q = ([76734/4, 82530/4, 83430/4, 89095/4, 88857/4, 83084/4, 83769/4,
86976/4, 91419/4, 103712/4, 87905/4])';

% Allocate correct liwing expenses to each quarter in price_houses set
n = 4;
liv_exp_apartments =
[repmat(liv_exp_a_70_Q(1),n,1);repmat(liv_exp_a_70_Q(2),n,1);repmat(liv_exp_a_70_Q(3),
n,1);repmat(liv_exp_a_70_Q(4),n,1);repmat(liv_exp_a_70_Q(5),n,1);repmat(liv_exp_a_70_Q
(6),n,1);repmat(liv_exp_a_70_Q(7),n,1);repmat(liv_exp_a_70_Q(8),n,1);repmat(liv_exp_a_
70_Q(9),n,1);repmat(liv_exp_a_70_Q(10),n,1);repmat(liv_exp_a_70_Q(11),n,1)];
price_inner_eastern.ordered_liv_exp_70 = repmat(liv_exp_apartments,(length(C)),1);

% Compute living expenses based on a reference apartment of 70 sqm
price_inner_eastern.living_exp =
(price_inner_eastern.prom./70).*price_inner_eastern.ordered_liv_exp_70;
avg_living_exp_Q =
mean(price_inner_eastern.living_exp./price_inner_eastern.estimated_price);

% Allocate correct implicit rents
implicit_rent     = zeros(height(price_inner_eastern),1);
rental_allocation = repmat(Rent_Data, length(C), 1);

implicit_rent(find(price_inner_eastern.Soverom == 0)) =
rental_allocation.room1(find(price_inner_eastern.Soverom == 0));
implicit_rent(find(price_inner_eastern.Soverom == 1)) =
rental_allocation.room1(find(price_inner_eastern.Soverom == 1));
implicit_rent(find(price_inner_eastern.Soverom == 2)) =
rental_allocation.room2(find(price_inner_eastern.Soverom == 2));
implicit_rent(find(price_inner_eastern.Soverom == 3)) =
rental_allocation.room3(find(price_inner_eastern.Soverom == 3));
implicit_rent(find(price_inner_eastern.Soverom == 4)) =
rental_allocation.room4(find(price_inner_eastern.Soverom == 4));
```

```
implicit_rent(find(price_inner_eastern.Soverom >= 5)) =
rental_allocation.room5_(find(price_inner_eastern.Soverom >= 5));

price_inner_eastern.implicit_rent = implicit_rent;
avg_implicit_rent_Q =
mean(price_inner_eastern.implicit_rent./price_inner_eastern.estimated_price);

% Implement tax rates for sample period
tax_rates = ([ 0.28, 0.28, 0.28, 0.28, 0.27, 0.27, 0.25, 0.24, 0.23, 0.22, 0.22])';
n = 4;
tax_rates_order =
[repmat(tax_rates(1),n,1);repmat(tax_rates(2),n,1);repmat(tax_rates(3),n,1);repmat(tax
_rates(4),n,1);repmat(tax_rates(5),n,1);repmat(tax_rates(6),n,1);repmat(tax_rates(7),n
,1);repmat(tax_rates(8),n,1);repmat(tax_rates(9),n,1);repmat(tax_rates(10),n,1);repmat
(tax_rates(11),n,1)];

% Allocate correct tax rates to each quarter in price_houses set
price_inner_eastern.tax_rates = repmat(tax_rates_order,(length(C)),1);

% Create a table to store different cost rates
avg_cost_rates_inner_eastern                          = zeros(4,3); %JUSTERE DENNE
ETTERSOM HVORDAN ST√òRRELSEN VIL BLI
avg_cost_rates_inner_eastern                          =
array2table(avg_cost_rates_inner_eastern);
avg_cost_rates_inner_eastern.Properties.VariableNames = {'Average_cost_rates',
'Yearly', 'Quarterly'};
avg_cost_rates_inner_eastern.Average_cost_rates       =
categorical(avg_cost_rates_inner_eastern.Average_cost_rates);

% Import stats for cost rates
avg_cost_rates_inner_eastern.Average_cost_rates(1) = 't_cost_buyer';
avg_cost_rates_inner_eastern.Yearly(1)             = t_cost_rate;
avg_cost_rates_inner_eastern.Quarterly(1)          = t_cost_rate_Q;

avg_cost_rates_inner_eastern.Average_cost_rates(2) = 't_cost_seller';
avg_cost_rates_inner_eastern.Yearly(2)             = t_cost_s_rate;
avg_cost_rates_inner_eastern.Quarterly(2)          = t_cost_s_rate_Q;

avg_cost_rates_inner_eastern.Average_cost_rates(3) = 'living_expenses';
avg_cost_rates_inner_eastern.Yearly(3)             = avg_living_exp_Q * 4;
avg_cost_rates_inner_eastern.Quarterly(3)          = avg_living_exp_Q;

avg_cost_rates_inner_eastern.Average_cost_rates(4)    = 'implicit_rent';
avg_cost_rates_inner_eastern.Yearly(4)                = avg_implicit_rent_Q * 4;
avg_cost_rates_inner_eastern.Quarterly(4)             = avg_implicit_rent_Q;

% 5. ALLOCATE TABLE FOR PRICE INDICES
priceindex_inner_eastern          = table();
Y_index                           = Y_order(1:44);
Q_index                           = Q_rep(1:44);
priceindex_inner_eastern.Q_order  = string(Y_index(:,1)) + Q_index(:,1);
priceindex_inner_eastern.Q_order  = categorical(priceindex_inner_eastern.Q_order);

% Create a table to store returns, std, risk free, sharpe
sum_stats_inner_eastern                           = zeros(10,7); %JUSTERE DENNE
ETTERSOM HVORDAN ST√òRRELSEN VIL BLI
sum_stats_inner_eastern                           =
array2table(sum_stats_inner_eastern);
sum_stats_inner_eastern.Properties.VariableNames = {'House_Indices', 'Return',
'std_dev','Excess_return','Sharpe_ratio', 'Skewness', 'Kurtosis'};
sum_stats_inner_eastern.House_Indices             =
categorical(sum_stats_inner_eastern.House_Indices);

% Import risk-free rates (innskuddsrente)
n = 4;
priceindex_inner_eastern.rf = zeros(44,1);
priceindex_inner_eastern.rf =
[repmat(Rf.rf(1),n,1);repmat(Rf.rf(2),n,1);repmat(Rf.rf(3),n,1);repmat(Rf.rf(4),n,1);r
epmat(Rf.rf(5),n,1);repmat(Rf.rf(6),n,1);repmat(Rf.rf(7),n,1);repmat(Rf.rf(8),n,1);rep
mat(Rf.rf(9),n,1);repmat(Rf.rf(10),n,1);repmat(Rf.rf(11),n,1)];
priceindex_inner_eastern.rf = priceindex_inner_eastern.rf./4;

% 6. COMPUTATION OF MEAN PRICES FOR HOUSES AND PRICE INDEX BASED ON MEAN PRICES
% Calculate mean prices for houses
```

```
priceindex_inner_eastern.mean_prices = zeros(height(priceindex_inner_eastern), end);
for i = 1:length(priceindex_inner_eastern.Q_order)
    priceindex_inner_eastern.mean_prices(i) =
mean(price_inner_eastern.estimated_price(find(price_inner_eastern.Q ==
priceindex_inner_eastern.Q_order(i))));
end

% Calculate price index for houses based on mean prices
priceindex_inner_eastern.index_values = zeros(height(priceindex_inner_eastern), end);
for i = 1:length(priceindex_inner_eastern.index_values)
    priceindex_inner_eastern.index_values(i) =
priceindex_inner_eastern.mean_prices(i)/priceindex_inner_eastern.mean_prices(1);
end

% 7. COMPUTATION OF QUARTERLY RETURNS ON ASSET/HOUSE FROM PERIOD TO PERIOD, NO COSTS
% Compute quarterly returns for each unique ID
price_inner_eastern.Q_ret_no_costs =
(price_inner_eastern.estimated_price./price_inner_eastern.lagged_estimated_price)-1;
price_inner_eastern.Q_ret_no_costs(1:44:end) = NaN;

% Compute mean quarterly returns for houses (mean of all IDs per quarter) (no costs)
priceindex_inner_eastern.Q_ret_no_costs = zeros(height(priceindex_inner_eastern),
end);

for i = 1:height(priceindex_inner_eastern)
    priceindex_inner_eastern.Q_ret_no_costs(i) =
nanmean(price_inner_eastern.Q_ret_no_costs(find(price_inner_eastern.Q ==
priceindex_inner_eastern.Q_order(i))));
end

% Compute mean of mean quarterly returns for houses (no costs)
mean_Q_ret_no_costs = nanmean(priceindex_inner_eastern.Q_ret_no_costs);

% Compute excess quarterly returns for houses (no costs)
priceindex_inner_eastern.excess_Q_ret_no_costs =
priceindex_inner_eastern.Q_ret_no_costs-priceindex_inner_eastern.rf;

% Compute mean excess quarterly returns for houses (no costs)
mean_excess_Q_ret_no_costs = nanmean(priceindex_inner_eastern.excess_Q_ret_no_costs);

% Compute standard deviation for each unique ID
st_Q_ret_no_costs= zeros(length(C),1);
time(1,:) = clock;
for i = 1:length(C)
    if b(i)<b(length(C))
       st_Q_ret_no_costs(i) =
std(price_inner_eastern.Q_ret_no_costs(((b(i)+1):(b(i+1)-1))));
    else
       st_Q_ret_no_costs(i) =
std(price_inner_eastern.Q_ret_no_costs((b(length(C))+1):height(price_inner_eastern)));
    end
time(i+1,:) = clock;
fprintf('\nTo complete %.f/%.f iterations (unique IDs) it took %.2f seconds\n', i,
length(C), etime(time(i+1,:), time(1,:)));
end

% Standard deviation of all quarterly returns
std_Q_ret_no_costs = mean(st_Q_ret_no_costs);

% Import stats for 'Q_ret_no_costs' in summary table
sum_stats_inner_eastern.House_Indices(1) = 'Q_ret_no_costs';
sum_stats_inner_eastern.Return(1)        = mean_Q_ret_no_costs;
sum_stats_inner_eastern.std_dev(1)       = std_Q_ret_no_costs;
sum_stats_inner_eastern.Excess_return(1) = mean_excess_Q_ret_no_costs;
sum_stats_inner_eastern.Sharpe_ratio(1)  =
sum_stats_inner_eastern.Excess_return(1)/sum_stats_inner_eastern.std_dev(1);
sum_stats_inner_eastern.Skewness(1)      =
skewness(priceindex_inner_eastern.Q_ret_no_costs(2:44));
sum_stats_inner_eastern.Kurtosis(1)      =
kurtosis(priceindex_inner_eastern.Q_ret_no_costs(2:44));

% COMPUTE CUMULATIVE ROA(NC)
first_obs_price = price_inner_eastern.estimated_price(1:44:end);
price_inner_eastern.first_obs_price_repeated = repelem(first_obs_price, 44);
```

```
price_inner_eastern.cum_Q_ret_no_costs =
(price_inner_eastern.estimated_price./price_inner_eastern.first_obs_price_repeated)-1;
price_inner_eastern.cum_Q_ret_no_costs(1:44:end) = NaN;

priceindex_inner_eastern.cum_Q_ret_no_costs =
zeros(height(priceindex_inner_eastern),end);
% Compute mean cumulative ROA(NC)
for i = 1:height(priceindex_inner_eastern)
    priceindex_inner_eastern.cum_Q_ret_no_costs(i) =
nanmean(price_inner_eastern.cum_Q_ret_no_costs(find(price_inner_eastern.Q ==
priceindex_inner_eastern.Q_order(i))));
end

% 8. COMPUTATION OF QUARTERLY RETURNS ON ASSET/HOUSE FROM PERIOD TO PERIOD (BUY/SELL),
INCL TOTAL COSTS
% Compute quarterly returns for each unique ID
price_inner_eastern.Q_ret_tot_costs = (((price_inner_eastern.estimated_price -
price_inner_eastern.t_cost_s - price_inner_eastern.lagged_t_cost -
price_inner_eastern.lagged_estimated_price).*(1-price_inner_eastern.tax_rates)) -
price_inner_eastern.living_exp +
price_inner_eastern.implicit_rent)./(price_inner_eastern.lagged_estimated_price);
price_inner_eastern.Q_ret_tot_costs(1:44:end) = NaN;

% Compute mean quarterly returns for houses (mean of all IDs per quarter) (inc tot
costs)
priceindex_inner_eastern.Q_ret_tot_costs = zeros(height(priceindex_inner_eastern),
end);

for i = 1:length(priceindex_inner_eastern.mean_prices)
    priceindex_inner_eastern.Q_ret_tot_costs(i) =
mean(price_inner_eastern.Q_ret_tot_costs(find(price_inner_eastern.Q ==
priceindex_inner_eastern.Q_order(i))));
end

% Compute mean of mean quarterly returns for houses (inc tot costs)
mean_Q_ret_tot_costs = nanmean(priceindex_inner_eastern.Q_ret_tot_costs);

% Compute excess quarterly returns for houses (inc tot costs)
priceindex_inner_eastern.excess_Q_ret_tot_costs =
priceindex_inner_eastern.Q_ret_tot_costs-priceindex_inner_eastern.rf;

% Compute mean excess quarterly returns for houses (inc tot costs)
mean_excess_Q_ret_tot_costs =
nanmean(priceindex_inner_eastern.excess_Q_ret_tot_costs);

% Compute standard deviation for each unique IDs returns
st_Q_ret_tot_costs= zeros(length(C),1);
time(1,:) = clock;
for i = 1:length(C)
    if b(i)<b(length(C))
       st_Q_ret_tot_costs(i) =
std(price_inner_eastern.Q_ret_tot_costs(((b(i)+1):(b(i+1)-1))));
    else
       st_Q_ret_tot_costs(i) =
std(price_inner_eastern.Q_ret_tot_costs((b(length(C))+1):height(price_inner_eastern)))
;
    end
time(i+1,:) = clock;
fprintf('\nTo complete %.f/%.f iterations (unique IDs) it took %.2f seconds\n', i,
length(C), etime(time(i+1,:), time(1,:)));
end

% Standard deviation of all quarterly returns
std_Q_ret_tot_costs = mean(st_Q_ret_tot_costs);

% Import stats for 'Q_ret_tot_costs' in summary table
sum_stats_inner_eastern.House_Indices(2) = 'Q_ret_tot_costs';
sum_stats_inner_eastern.Return(2)        = mean_Q_ret_tot_costs;
sum_stats_inner_eastern.std_dev(2)       = std_Q_ret_tot_costs;
sum_stats_inner_eastern.Excess_return(2) = mean_excess_Q_ret_tot_costs;
sum_stats_inner_eastern.Sharpe_ratio(2) =
sum_stats_inner_eastern.Excess_return(2)/sum_stats_inner_eastern.std_dev(2);
```

```
sum_stats_inner_eastern.Skewness(2)      =
skewness(priceindex_inner_eastern.Q_ret_tot_costs(2:44));
sum_stats_inner_eastern.Kurtosis(2)      =
kurtosis(priceindex_inner_eastern.Q_ret_tot_costs(2:44));

% COMPUTE CUMULATIVE ROA (TC)
taxes_TC = (price_inner_eastern.estimated_price - price_inner_eastern.t_cost_s -
price_inner_eastern.lagged_t_cost -
price_inner_eastern.lagged_estimated_price).*price_inner_eastern.tax_rates;
taxes_TC(1:44:end) = NaN;

net_costs_TC = (price_inner_eastern.living_exp + price_inner_eastern.lagged_t_cost +
price_inner_eastern.t_cost_s + taxes_TC - price_inner_eastern.implicit_rent);
net_costs_TC(1:44:end) = NaN;

% Compute cumulative costs
cumcosts_TC = zeros(height(price_inner_eastern), end);
time(1,:) = clock;
for i = 1:length(C)
    if b(i)<b(length(C))
       cumcosts_TC((b(i)):(b(i+1)-1)) = cumsum(net_costs_TC((b(i)):(b(i+1)-
1)),'omitnan');
    else
       cumcosts_TC(b(length(C)):height(price_inner_eastern)) =
cumsum(net_costs_TC(b(length(C)):height(price_inner_eastern)),'omitnan');
    end
time(i+1,:) = clock;
fprintf('\nTo complete %.f/%.f iterations (unique IDs) it took %.2f seconds\n', i,
length(C), etime(time(i+1,:), time(1,:)));
end

price_inner_eastern.cum_Q_ret_tot_costs = (price_inner_eastern.estimated_price -
cumcosts_TC -
price_inner_eastern.first_obs_price_repeated)./price_inner_eastern.first_obs_price_rep
eated;
price_inner_eastern.cum_Q_ret_tot_costs(1:44:end) = NaN;

priceindex_inner_eastern.cum_Q_ret_tot_costs =
zeros(height(priceindex_inner_eastern),end);
% Compute mean cumulative returns for houses (mean of all IDs per quarter) (no costs)
for i = 1:height(priceindex_inner_eastern)
    priceindex_inner_eastern.cum_Q_ret_tot_costs(i) =
nanmean(price_inner_eastern.cum_Q_ret_tot_costs(find(price_inner_eastern.Q ==
priceindex_inner_eastern.Q_order(i))));
end

all_lost = priceindex_inner_eastern.cum_Q_ret_tot_costs < -1;
priceindex_inner_eastern.cum_Q_ret_tot_costs(all_lost) = -1;

% 9. COMPUTATION OF QUARTERLY RETURNS ON ASSET/HOUSE FROM PERIOD TO PERIOD , INCL
SMOOTHED COSTS
% Locate buyer and sellers cost when buying in 2010Q1 and selling in 2020Q4
% Find buyer and seller cost per quarter for sample period
cost_2010Q1 = price_inner_eastern.t_cost(1:44:end);
cost_2010Q1 = cost_2010Q1/44;
price_inner_eastern.cost_2010Q1 = repelem(cost_2010Q1, 44);

cost_2020Q4 = price_inner_eastern.t_cost_s(44:44:end);
cost_2020Q4 = cost_2020Q4/44;
price_inner_eastern.cost_2020Q4 = repelem(cost_2020Q4, 44);

% Compute quarterly returns given that you buy in 2010Q1 and sell in 2020Q4
% and spred the relevant costs on the sample period.
price_inner_eastern.Q_ret_smooth_costs = (price_inner_eastern.estimated_price -
price_inner_eastern.cost_2010Q1 - price_inner_eastern.cost_2020Q4 -
price_inner_eastern.living_exp - price_inner_eastern.lagged_estimated_price +
price_inner_eastern.implicit_rent)./price_inner_eastern.lagged_estimated_price;
price_inner_eastern.Q_ret_smooth_costs(1:44:end) = NaN;

% Compute mean quarterly returns for houses (mean of all IDs per quarter) (inc smooth
costs)
priceindex_inner_eastern.Q_ret_smooth_costs = zeros(height(priceindex_inner_eastern),
end);
```

```
for i = 1:length(priceindex_inner_eastern.mean_prices)
    priceindex_inner_eastern.Q_ret_smooth_costs(i) =
mean(price_inner_eastern.Q_ret_smooth_costs(find(price_inner_eastern.Q ==
priceindex_inner_eastern.Q_order(i))));
end

% Compute mean of mean quarterly returns for houses (inc smooth costs)
mean_Q_ret_smooth_costs = nanmean(priceindex_inner_eastern.Q_ret_smooth_costs);

% Compute excess quarterly returns for houses (inc smooth costs)
priceindex_inner_eastern.excess_Q_ret_smooth_costs =
priceindex_inner_eastern.Q_ret_smooth_costs - priceindex_inner_eastern.rf;

% Compute mean excess quarterly returns for houses (inc smooth costs)
mean_excess_Q_ret_smooth_costs =
nanmean(priceindex_inner_eastern.excess_Q_ret_smooth_costs);

% Compute standard deviation for each unique IDs returns
st_Q_ret_smooth_costs= zeros(length(C),1);
time(1,:) = clock;
for i = 1:length(C)
    if b(i)<b(length(C))
       st_Q_ret_smooth_costs(i) =
std(price_inner_eastern.Q_ret_smooth_costs(((b(i)+1):(b(i+1)-1))));
    else
       st_Q_ret_smooth_costs(i) =
std(price_inner_eastern.Q_ret_smooth_costs((b(length(C))+1):height(price_inner_eastern
)));
    end
time(i+1,:) = clock;
fprintf('\nTo complete %.f/%.f iterations (unique IDs) it took %.2f seconds\n', i,
length(C), etime(time(i+1,:), time(1,:)));
end

% Standard deviation of all quarterly returns
std_Q_ret_smooth_costs = mean(st_Q_ret_smooth_costs);

% Import stats for 'Q_ret_smooth_costs' in summary table
sum_stats_inner_eastern.House_Indices(3) = 'Q_ret_smooth_costs';
sum_stats_inner_eastern.Return(3)        = mean_Q_ret_smooth_costs;
sum_stats_inner_eastern.std_dev(3)       = std_Q_ret_smooth_costs;
sum_stats_inner_eastern.Excess_return(3) = mean_excess_Q_ret_smooth_costs;
sum_stats_inner_eastern.Sharpe_ratio(3) =
sum_stats_inner_eastern.Excess_return(3)/sum_stats_inner_eastern.std_dev(3);
sum_stats_inner_eastern.Skewness(3)      =
skewness(priceindex_inner_eastern.Q_ret_smooth_costs(2:44));
sum_stats_inner_eastern.Kurtosis(3)      =
kurtosis(priceindex_inner_eastern.Q_ret_smooth_costs(2:44));

% COMPUTE CUMULATIVE ROA(SC)
net_costs_SC = (price_inner_eastern.living_exp + price_inner_eastern.cost_2010Q1 +
price_inner_eastern.cost_2020Q4 - price_inner_eastern.implicit_rent);
net_costs_SC(1:44:end) = NaN;

cumcosts_SC = zeros(height(price_inner_eastern), end);
time(1,:) = clock;
for i = 1:length(C)
    if b(i)<b(length(C))
       cumcosts_SC((b(i)):(b(i+1)-1)) = cumsum(net_costs_SC((b(i)):(b(i+1)-
1)),'omitnan');
    else
       cumcosts_SC(b(length(C)):height(price_inner_eastern)) =
cumsum(net_costs_SC(b(length(C)):height(price_inner_eastern)),'omitnan');
    end
time(i+1,:) = clock;
fprintf('\nTo complete %.f/%.f iterations (unique IDs) it took %.2f seconds\n', i,
length(C), etime(time(i+1,:), time(1,:)));
end

price_inner_eastern.cum_Q_ret_smooth_costs = (price_inner_eastern.estimated_price -
cumcosts_SC -
price_inner_eastern.first_obs_price_repeated)./price_inner_eastern.first_obs_price_rep
eated;
price_inner_eastern.cum_Q_ret_smooth_costs(1:44:end) = NaN;
```

```
% Compute mean cumulative returns for houses (mean of all IDs per quarter) (no costs)
priceindex_inner_eastern.cum_Q_ret_smooth_costs =
zeros(height(priceindex_inner_eastern),end);
for i = 1:height(priceindex_inner_eastern)
    priceindex_inner_eastern.cum_Q_ret_smooth_costs(i) =
nanmean(price_inner_eastern.cum_Q_ret_smooth_costs(find(price_inner_eastern.Q ==
priceindex_inner_eastern.Q_order(i))));
end

% Identify average loan-to-value ratio for housing investments
l = ([0.6957, 0.6957, 0.6957, 0.6957, 0.71, 0.76, 0.67, 0.66, 0.68, 0.70, 0.69])';
n = 4;
ltv_ratio_order =
[repmat(l(1),n,1);repmat(l(2),n,1);repmat(l(3),n,1);repmat(l(4),n,1);repmat(l(5),n,1);
repmat(l(6),n,1);repmat(l(7),n,1);repmat(l(8),n,1);repmat(l(9),n,1);repmat(l(10),n,1);
repmat(l(11),n,1)];

% Allocate correct ltv-rates to each quarter in price_houses set
price_inner_eastern.ltvr        = repmat(ltv_ratio_order,(length(C)),1);
price_inner_eastern.lagged_ltvr = lagmatrix(price_inner_eastern.ltvr,1);

% Allocate ltv ratio as of 2010Q1 to each quarter in price_houses set
price_inner_eastern.ltvr_2010Q1(:) = price_inner_eastern.ltvr(1);

% 10. WE DONT USE THE SERIES COMPUTED IN SECTION 10
% Compute quarterly equity returns for each unique ID (no costs)
price_inner_eastern.Q_eq_ret_no_costs = ((price_inner_eastern.estimated_price -
price_inner_eastern.lagged_estimated_price)./(price_inner_eastern.lagged_estimated_pri
ce.*(1-price_inner_eastern.lagged_ltvr)));
price_inner_eastern.Q_eq_ret_no_costs(1:44:end) = NaN;

% Compute mean quarterly equity returns for houses (mean of all IDs per quarter) (no
costs)
priceindex_inner_eastern.Q_eq_ret_no_costs = zeros(height(priceindex_inner_eastern),
end);

for i = 1:height(priceindex_inner_eastern)
    priceindex_inner_eastern.Q_eq_ret_no_costs(i) =
mean(price_inner_eastern.Q_eq_ret_no_costs(find(price_inner_eastern.Q ==
priceindex_inner_eastern.Q_order(i))));
end

% Compute mean of mean quarterly equity returns for houses (no costs)
mean_Q_eq_ret_no_costs = nanmean(priceindex_inner_eastern.Q_eq_ret_no_costs);

% Compute excess quarterly equity returns for houses (no costs)
priceindex_inner_eastern.excess_Q_eq_ret_no_costs =
priceindex_inner_eastern.Q_eq_ret_no_costs - priceindex_inner_eastern.rf;

% Compute mean excess quarterly equity returns for houses (no costs)
mean_excess_Q_eq_ret_no_costs =
nanmean(priceindex_inner_eastern.excess_Q_eq_ret_no_costs);

% Compute standard deviation for each unique IDs returns
st_Q_eq_ret_no_costs= zeros(length(C),1);
time(1,:) = clock;
for i = 1:length(C)
    if b(i)<b(length(C))
       st_Q_eq_ret_no_costs(i) =
std(price_inner_eastern.Q_eq_ret_no_costs(((b(i)+1):(b(i+1)-1))));
    else
       st_Q_eq_ret_no_costs(i) =
std(price_inner_eastern.Q_eq_ret_no_costs((b(length(C))+1):height(price_inner_eastern)
));
    end
time(i+1,:) = clock;
fprintf('\nTo complete %.f/%.f iterations (unique IDs) it took %.2f seconds\n', i,
length(C), etime(time(i+1,:), time(1,:)));
end

% Standard deviation of all quarterly equity returns
std_Q_eq_ret_no_costs = mean(st_Q_eq_ret_no_costs);
```

```
% Import stats for 'Q_eq_ret_no_costs' in summary table
sum_stats_inner_eastern.House_Indices(4) = 'Q_eq_ret_no_costs';
sum_stats_inner_eastern.Return(4)        = mean_Q_eq_ret_no_costs;
sum_stats_inner_eastern.std_dev(4)       = std_Q_eq_ret_no_costs;
sum_stats_inner_eastern.Excess_return(4) = mean_excess_Q_eq_ret_no_costs;
sum_stats_inner_eastern.Sharpe_ratio(4) =
sum_stats_inner_eastern.Excess_return(4)/sum_stats_inner_eastern.std_dev(4);
sum_stats_inner_eastern.Skewness(4)      =
skewness(priceindex_inner_eastern.Q_eq_ret_no_costs(2:44));
sum_stats_inner_eastern.Kurtosis(4)      =
kurtosis(priceindex_inner_eastern.Q_eq_ret_no_costs(2:44));

% COMPUTE CUMULATIVE ROE (NC)
price_inner_eastern.cum_Q_eq_ret_no_costs = ((price_inner_eastern.estimated_price -
price_inner_eastern.first_obs_price_repeated)./(price_inner_eastern.first_obs_price_re
peated.*(1-price_inner_eastern.ltvr_2010Q1)));
price_inner_eastern.cum_Q_eq_ret_no_costs(1:44:end) = NaN;

% Compute mean cumulative returns for houses (mean of all IDs per quarter) (no costs)
priceindex_inner_eastern.cum_Q_eq_ret_no_costs =
zeros(height(priceindex_inner_eastern),end);
for i = 1:height(priceindex_inner_eastern)
    priceindex_inner_eastern.cum_Q_eq_ret_no_costs(i) =
nanmean(price_inner_eastern.cum_Q_eq_ret_no_costs(find(price_inner_eastern.Q ==
priceindex_inner_eastern.Q_order(i))));
end

% 11. WE DONT USE THE SERIES COMPUTED IN SECTION 11
% Compute quarterly equity returns for each unique ID (inc tot costs)
price_inner_eastern.Q_eq_ret_tot_costs = (((price_inner_eastern.estimated_price -
price_inner_eastern.t_cost_s - price_inner_eastern.lagged_t_cost -
price_inner_eastern.lagged_estimated_price).*(1-price_inner_eastern.tax_rates)) -
price_inner_eastern.living_exp +
price_inner_eastern.implicit_rent)./(price_inner_eastern.lagged_estimated_price.*(1-
price_inner_eastern.lagged_ltvr));
price_inner_eastern.Q_eq_ret_tot_costs(1:44:end) = NaN;

% Compute mean quarterly equity returns for houses (mean of all IDs per quarter) (inc
tot costs)
priceindex_inner_eastern.Q_eq_ret_tot_costs = zeros(height(priceindex_inner_eastern),
end);

for i = 1:length(priceindex_inner_eastern.mean_prices)
    priceindex_inner_eastern.Q_eq_ret_tot_costs(i) =
mean(price_inner_eastern.Q_eq_ret_tot_costs(find(price_inner_eastern.Q ==
priceindex_inner_eastern.Q_order(i))));
end

% Compute mean of mean quarterly equity returns for houses (inc tot costs)
mean_Q_eq_ret_tot_costs = nanmean(priceindex_inner_eastern.Q_eq_ret_tot_costs);

% Compute excess quarterly equity returns for houses (inc tot costs)
priceindex_inner_eastern.excess_Q_eq_ret_tot_costs =
priceindex_inner_eastern.Q_eq_ret_tot_costs - priceindex_inner_eastern.rf;

% Compute mean excess quarterly equity returns for houses (inc tot costs)
mean_excess_Q_eq_ret_tot_costs =
nanmean(priceindex_inner_eastern.excess_Q_eq_ret_tot_costs);

% Compute standard deviation for each unique IDs returns
st_Q_eq_ret_tot_costs= zeros(length(C),1);
time(1,:) = clock;
for i = 1:length(C)
    if b(i)<b(length(C))
       st_Q_eq_ret_tot_costs(i) =
std(price_inner_eastern.Q_eq_ret_tot_costs(((b(i)+1):(b(i+1)-1))));
    else
       st_Q_eq_ret_tot_costs(i) =
std(price_inner_eastern.Q_eq_ret_tot_costs((b(length(C))+1):height(price_inner_eastern
)));
    end
time(i+1,:) = clock;
fprintf('\nTo complete %.f/%.f iterations (unique IDs) it took %.2f seconds\n', i,
length(C), etime(time(i+1,:), time(1,:)));
```

```
end

% Standard deviation of all quarterly equity returns
std_Q_eq_ret_tot_costs = mean(st_Q_eq_ret_tot_costs);

% Import stats for 'Q_eq_ret_tot_costs' in summary table
sum_stats_inner_eastern.House_Indices(5) = 'Q_eq_ret_tot_costs';
sum_stats_inner_eastern.Return(5)        = mean_Q_eq_ret_tot_costs;
sum_stats_inner_eastern.std_dev(5)       = std_Q_eq_ret_tot_costs;
sum_stats_inner_eastern.Excess_return(5) = mean_excess_Q_eq_ret_tot_costs;
sum_stats_inner_eastern.Sharpe_ratio(5) =
sum_stats_inner_eastern.Excess_return(5)/sum_stats_inner_eastern.std_dev(5);
sum_stats_inner_eastern.Skewness(5)      =
skewness(priceindex_inner_eastern.Q_eq_ret_tot_costs(2:44));
sum_stats_inner_eastern.Kurtosis(5)      =
kurtosis(priceindex_inner_eastern.Q_eq_ret_tot_costs(2:44));

% COMPUTE CUMULATIVE RETURNS BASED ON QUARTERLY EQUITY RETURNS FROM PRICE INDEX (INC
TOT COSTS)
priceindex_inner_eastern.cum_Q_eq_ret_tot_costs =
zeros(height(priceindex_inner_eastern),end);
priceindex_inner_eastern.cum_Q_eq_ret_tot_costs =
cumprod((1+priceindex_inner_eastern.Q_eq_ret_tot_costs),'omitnan') -1;

% 12. COMPUTATION OF QUARTERLY EQUITY RETURNS ON ASSET/HOUSE FROM PERIOD TO PERIOD ,
INCL SMOOTHED COSTS
% Compute quarterly EQUITY returns given that you buy in 2010Q1 and sell in 2020Q4
% and spread the relevant costs on the sample period.
price_inner_eastern.Q_eq_ret_smooth_costs = (price_inner_eastern.estimated_price -
price_inner_eastern.cost_2010Q1 - price_inner_eastern.cost_2020Q4 -
price_inner_eastern.living_exp - price_inner_eastern.lagged_estimated_price +
price_inner_eastern.implicit_rent)./(price_inner_eastern.lagged_estimated_price.*(1-
price_inner_eastern.ltvr_2010Q1));
price_inner_eastern.Q_eq_ret_smooth_costs(1:44:end) = NaN;

% Compute mean quarterly equity returns for houses (mean of all IDs per quarter) (inc
smooth costs)
priceindex_inner_eastern.Q_eq_ret_smooth_costs =
zeros(height(priceindex_inner_eastern), end);

for i = 1:length(priceindex_inner_eastern.mean_prices)
    priceindex_inner_eastern.Q_eq_ret_smooth_costs(i) =
mean(price_inner_eastern.Q_eq_ret_smooth_costs(find(price_inner_eastern.Q ==
priceindex_inner_eastern.Q_order(i))));
end

% Compute mean of mean quarterly equity returns for houses (inc smooth costs)
mean_Q_eq_ret_smooth_costs = nanmean(priceindex_inner_eastern.Q_eq_ret_smooth_costs);

% Compute excess quarterly equity returns for houses (inc smooth costs)
priceindex_inner_eastern.excess_Q_eq_ret_smooth_costs =
priceindex_inner_eastern.Q_eq_ret_smooth_costs - priceindex_inner_eastern.rf;

% Compute mean excess quarterly equity returns for houses (inc smooth costs)
mean_excess_Q_eq_ret_smooth_costs =
nanmean(priceindex_inner_eastern.excess_Q_eq_ret_smooth_costs);

% Compute standard deviation for each unique IDs returns
st_Q_eq_ret_smooth_costs= zeros(length(C),1);
time(1,:) = clock;
for i = 1:length(C)
    if b(i)<b(length(C))
       st_Q_eq_ret_smooth_costs(i) =
std(price_inner_eastern.Q_eq_ret_smooth_costs(((b(i)+1):(b(i+1)-1))));
    else
       st_Q_eq_ret_smooth_costs(i) =
std(price_inner_eastern.Q_eq_ret_smooth_costs((b(length(C))+1):height(price_inner_east
ern)));
    end
time(i+1,:) = clock;
fprintf('\nTo complete %.f/%.f iterations (unique IDs) it took %.2f seconds\n', i,
length(C), etime(time(i+1,:), time(1,:)));
end
```

```
% Standard deviation of all quarterly equity returns
std_Q_eq_ret_smooth_costs = mean(st_Q_eq_ret_smooth_costs);

% Import stats for 'Q_eq_ret_smooth_costs' in summary table
sum_stats_inner_eastern.House_Indices(6) = 'Q_eq_ret_smooth_costs';
sum_stats_inner_eastern.Return(6)        = mean_Q_eq_ret_smooth_costs;
sum_stats_inner_eastern.std_dev(6)       = std_Q_eq_ret_smooth_costs;
sum_stats_inner_eastern.Excess_return(6) = mean_excess_Q_eq_ret_smooth_costs;
sum_stats_inner_eastern.Sharpe_ratio(6) =
sum_stats_inner_eastern.Excess_return(6)/sum_stats_inner_eastern.std_dev(6);
sum_stats_inner_eastern.Skewness(6)      =
skewness(priceindex_inner_eastern.Q_eq_ret_smooth_costs(2:44));
sum_stats_inner_eastern.Kurtosis(6)      =
kurtosis(priceindex_inner_eastern.Q_eq_ret_smooth_costs(2:44));

% COMPUTE CUMULATIVE ROE(SC)
price_inner_eastern.cum_Q_eq_ret_smooth_costs = (price_inner_eastern.estimated_price -
cumcosts_SC -
price_inner_eastern.first_obs_price_repeated)./(price_inner_eastern.first_obs_price_re
peated.*(1-price_inner_eastern.ltvr_2010Q1));
price_inner_eastern.cum_Q_eq_ret_smooth_costs(1:44:end) = NaN;

% Compute mean cumulative returns for houses (mean of all IDs per quarter) (smooth
costs)
priceindex_inner_eastern.cum_Q_eq_ret_smooth_costs =
zeros(height(priceindex_inner_eastern),end);
for i = 1:height(priceindex_inner_eastern)
    priceindex_inner_eastern.cum_Q_eq_ret_smooth_costs(i) =
nanmean(price_inner_eastern.cum_Q_eq_ret_smooth_costs(find(price_inner_eastern.Q ==
priceindex_inner_eastern.Q_order(i))));
end

% MAKE PLOT OF THE RELEVANT INDICES TOGHETER, MAKE MULTIPLE PLOTS FOR COMPARISON
REASONS
inner_eastern_price_changes = priceindex_inner_eastern(:,[1 5 8 11 14 17 20]);

inner_eastern_price_indices = priceindex_inner_eastern(:,[1 7 10 13 16 19 22]);
inner_eastern_price_indices.cum_Q_ret_no_costs(1) = 0;
inner_eastern_price_indices.cum_Q_ret_tot_costs(1) = 0;
inner_eastern_price_indices.cum_Q_ret_smooth_costs(1) = 0;
inner_eastern_price_indices.cum_Q_eq_ret_no_costs(1) = 0;
inner_eastern_price_indices.cum_Q_eq_ret_tot_costs(1) = 0;
inner_eastern_price_indices.cum_Q_eq_ret_smooth_costs(1) = 0;

inner_eastern_price_indices.cum_Q_ret_no_costs        =
inner_eastern_price_indices.cum_Q_ret_no_costs + 1;
inner_eastern_price_indices.cum_Q_ret_tot_costs       =
inner_eastern_price_indices.cum_Q_ret_tot_costs + 1;
inner_eastern_price_indices.cum_Q_ret_smooth_costs    =
inner_eastern_price_indices.cum_Q_ret_smooth_costs + 1;
inner_eastern_price_indices.cum_Q_eq_ret_no_costs     =
inner_eastern_price_indices.cum_Q_eq_ret_no_costs + 1;
inner_eastern_price_indices.cum_Q_eq_ret_tot_costs    =
inner_eastern_price_indices.cum_Q_eq_ret_tot_costs + 1;
inner_eastern_price_indices.cum_Q_eq_ret_smooth_costs =
inner_eastern_price_indices.cum_Q_eq_ret_smooth_costs + 1;

clearvars -except Property_Data summary_table Rf Rent_Data...
                  house_set hedonic_houses housing_model...
                  price_houses uniq_house_set avg_cost_rates_houses...
                  priceindex_houses sum_stats_houses houseprice_changes...
                  houseprice_indices...
                  smallhouse_set hedonic_smallhouses smallhouse_model...
                  price_smallhouses uniq_smallhouse_set avg_cost_rates_smallhouses...
                  priceindex_smallhouses sum_stats_smallhouses...
                  smallhouse_price_changes smallhouse_price_indices...
                  apartment_set hedonic_apartments apartment_model price_apartments...
                  uniq_apartment_set avg_cost_rates_apartments
priceindex_apartments...
                  sum_stats_apartments apartment_price_changes...
                  apartment_price_indices...
                  inner_eastern_set hedonic_inner_eastern inner_eastern_model...
                  price_inner_eastern uniq_inner_eastern_set
avg_cost_rates_inner_eastern...
```

```
                priceindex_inner_eastern sum_stats_inner_eastern
inner_eastern_price_changes...
                inner_eastern_price_indices...
                inner_western_set hedonic_inner_western inner_western_model...
                price_inner_western uniq_inner_western_set
avg_cost_rates_inner_western...
                priceindex_inner_western sum_stats_inner_western
inner_western_price_changes...
                inner_western_price_indices...
                outer_eastern_set hedonic_outer_eastern outer_eastern_model...
                price_outer_eastern uniq_outer_eastern_set
avg_cost_rates_outer_eastern...
                priceindex_outer_eastern sum_stats_outer_eastern
outer_eastern_price_changes...
                outer_eastern_price_indices...
                outer_western_set hedonic_outer_western outer_western_model...
                price_outer_western uniq_outer_western_set
avg_cost_rates_outer_western...
                priceindex_outer_western sum_stats_outer_western
outer_western_price_changes...
                outer_western_price_indices...
                outer_southern_set hedonic_outer_southern outer_southern_model...
                price_outer_southern uniq_outer_southern_set
avg_cost_rates_outer_southern...
                priceindex_outer_southern sum_stats_outer_southern
outer_southern_price_changes...
                outer_southern_price_indices...
                property_price_changes property_price_indices hedonic_total...
                ROA_NC_stats ROE_NC_stats ROA_TC_stats ROE_TC_stats...
                ROA_SC_stats ROE_SC_stats sum_stats_OSEFX OSEFX...
                ROA_NC_sharpe_test ROA_TC_sharpe_test ROA_SC_sharpe_test...
                ROE_SC_sharpe_test

%% Regression for Inner Western
% 1. PREPERATION OF DATASET
inner_western      = (apartment_set.Bydel == 'St. Hanshaugen') + (apartment_set.Bydel
== 'Frogner') + (apartment_set.Bydel == 'Sentrum');
inner_western      = logical(inner_western);
inner_western_set  = apartment_set(inner_western,:);

% Make categorical for dummy-purposes
inner_western_set.Alder    = categorical(inner_western_set.Alder);
inner_western_set.Q        = categorical(inner_western_set.Q);
inner_western_set.Eierform = categorical(inner_western_set.Eierform);
inner_western_set.Etasje   = categorical(inner_western_set.Etasje);

% Reorder categories/dummies to find reference variables
inner_western_set.Alder    = reordercats(inner_western_set.Alder,    {'Age1', 'Age2',
'Age3', 'Age4'});
inner_western_set.Eierform = reordercats(inner_western_set.Eierform, {'Selveier',
'Borettslag'});
inner_western_set.Etasje   = reordercats(inner_western_set.Etasje,   {'1', '2', '3',
'4', '5', '6', '7', '8', '9', '10', '11', '12', '13', '14', '15', '-1', '-2', '-3', '-
4'});

% Remove unused categories
inner_western_set.Eierform = removecats(inner_western_set.Eierform);

% Check that we in fact removed unused categories
categories(inner_western_set.Alder)
categories(inner_western_set.Eierform)

% Make log prices and sizes
inner_western_set.log_pris = log(inner_western_set.Pris);
inner_western_set.log_prom = log(inner_western_set.prom);

% 2. Run regression - Log model with log(prom), Q, Bydel, Soverom, Alder on log(pris)
% Explanatory variables log(prom), soverom - continous
% Explanatory variables Q, Bydel, Eierform, Alder and Etasje - dummies
inner_western_model =
fitlm(inner_western_set,'log_pris~log_prom+Q+Eierform+Soverom+Alder+Etasje');

% 3. ALLOCATE TABLE FOR ESTIMATED PRICES (BASED ON MODEL FROM PT. 2) FOR EVERY QUARTER
FOR EACH ID
```

```
inner_western_set   = sortrows(inner_western_set,'BoligID','ascend');
[C,ia,ic]           = unique(inner_western_set.BoligID,'first');
price_inner_western = zeros(length(C)*44, width(inner_western_set));
price_inner_western = array2table(price_inner_western);

price_inner_western.Properties.VariableNames = {'BoligID', 'Salgsdato', 'Bydel',
'Pris', 'Fellesgjeld', 'Prisantydning', 'Eierform', 'Byggeaar', 'prom', 'Etasje',
'Soverom', 'Boligtype', 'year', 'Q', 'Alder', 'kvmpris', 'log_pris', 'log_prom' };

Q_rep   = repmat(["Q1"; "Q2"; "Q3";"Q4"],(length(C)*44)/4,1);
x       = (2010:2020)';
n       = 4;
newx    =
[repmat(x(1),n,1);repmat(x(2),n,1);repmat(x(3),n,1);repmat(x(4),n,1);repmat(x(5),n,1);
repmat(x(6),n,1);repmat(x(7),n,1);repmat(x(8),n,1);repmat(x(9),n,1);repmat(x(10),n,1);
repmat(x(11),n,1)];
Y_order = repmat(newx,(length(C)),1);

price_inner_western.Q_order = string(Y_order(:,1)) + Q_rep(:,1);
price_inner_western.Q       = price_inner_western.Q_order;
price_inner_western.Q_order = [];

price_inner_western.BoligID = categorical(price_inner_western.BoligID);
price_inner_western.BoligID = repelem(C, 44);

price_inner_western           = convertvars(price_inner_western,{'BoligID', 'Bydel',
'Eierform', 'Boligtype', 'Etasje', 'Alder', 'Q'},'categorical');
price_inner_western.Salgsdato =
datetime(price_inner_western.Salgsdato,'ConvertFrom','yyyymmdd');

b = 1:length(newx):height(price_inner_western);

% Create set containing only the unique observations of house_set. When
% multiple unique values it stores the first observation
uniq_inner_western_set = inner_western_set(ia,:);

% Allocate variables into price_inner_western to use for regression model
price_inner_western(:,[3 5 7 9 10 11 15 18]) = repelem(uniq_inner_western_set(:,[3 5 7
9 10 11 15 18]), 44, 1);

% Store relevant variables for regression model in new table
x3 = price_inner_western(:,[18 14 7 11 15 10]);

% GET COLUMN OF ESTIMATED PRICES BASED ON REGRESSION MODEL
price_inner_western.log_estimated_price    = predict(inner_western_model, x3);
price_inner_western.estimated_price        =
exp(price_inner_western.log_estimated_price);
price_inner_western.lagged_estimated_price =
lagmatrix(price_inner_western.estimated_price,1);

% 4. IMPLEMENT RELEVANT COSTS RELATED TO HOUSING INVESTMENTS
% Transaction costs buyer, b
gebyr_b             = zeros(height(price_inner_western),1);
dokavg_b            = zeros(height(price_inner_western),1);
insurance_b         = zeros(height(price_inner_western),1);
price_inner_western.t_cost = zeros(height(price_inner_western),1);

% If type of ownership = 'Selveier'
idx_selv              = price_inner_western.Eierform == "Selveier";
gebyr_b(idx_selv)     = 540 + 540 + 172;
dokavg_b(idx_selv)    =
0.025.*(price_inner_western.estimated_price(idx_selv)+price_inner_western.Fellesgjeld(
idx_selv));
insurance_b(idx_selv) = 7400;

% If type of ownership = 'Borettslag'
idx_andel             = price_inner_western.Eierform == "Borettslag";
gebyr_b(idx_andel)    = 440 + 440 + 172;
dokavg_b(idx_andel)   = 0;
insurance_b(idx_andel) = 4100;

% Compute total transaction cost for buyer, b, for each observation/quarter
price_inner_western.t_cost        = gebyr_b+dokavg_b+insurance_b;
price_inner_western.lagged_t_cost = lagmatrix(price_inner_western.t_cost,1);
```

```
% Transaction costs seller, s
eierskiftegebyr_s     = zeros(height(price_inner_western),1);
public_fees_s         = zeros(height(price_inner_western),1);
insurance_s           = zeros(height(price_inner_western),1);
commission_s          = zeros(height(price_inner_western),1);
price_inner_western.t_cost_s = zeros(height(price_inner_western),1);
indices_selv          = zeros(height(price_inner_western),1);
indices_andel         = zeros(height(price_inner_western),1);

% If type of ownership = 'Selveier'
idx_selv                    = price_inner_western.Eierform == "Selveier";
public_fees_s(idx_selv)     = 8430;
eierskiftegebyr_s(idx_selv) = 0;
insurance_s(idx_selv)       =
max(0.0033*(price_inner_western.estimated_price(idx_selv)+price_inner_western.Fellesgj
eld(idx_selv)), 4000);
indices_selv                = insurance_s > 24000;
indices_selv                = logical(indices_selv);
insurance_s(indices_selv)   = 24000;

% If type of ownership = 'Borettslag'
idx_andel                    = price_inner_western.Eierform == "Borettslag";
public_fees_s(idx_andel)     = 14319;
eierskiftegebyr_s(idx_andel) = 4858;
insurance_s(idx_andel)       =
max(0.002*(price_inner_western.estimated_price(idx_andel)+price_inner_western.Fellesgj
eld(idx_andel)), 2500);
indices_andel                = zeros(height(price_inner_western),1);
indices_andel(idx_andel)     = insurance_s(idx_andel) > 24000;
indices_andel                = logical(indices_andel);
insurance_s(indices_andel)   = 24000;

% Commission to broker independent of type of ownership
commission_s =
max(0.02*(price_inner_western.estimated_price+price_inner_western.Fellesgjeld),40000);

% Compute total transaction cost for seller, s, for each observation/quarter
price_inner_western.t_cost_s =
public_fees_s+eierskiftegebyr_s+insurance_s+commission_s;

% Compute transaction cost rates for buyer and seller
% For buyer
t_cost_rate   = mean(price_inner_western.t_cost./price_inner_western.estimated_price);
t_cost_rate_Q = t_cost_rate/4;

% For seller
t_cost_s_rate   =
mean(price_inner_western.t_cost_s./price_inner_western.estimated_price);
t_cost_s_rate_Q = t_cost_s_rate/4;

% Define holding period for each dwelling
holding_period = 11;

% Define living expenses per quarter - reference apartment is 70 sqm
liv_exp_a_70_Q = ([76734/4, 82530/4, 83430/4, 89095/4, 88857/4, 83084/4, 83769/4,
86976/4, 91419/4, 103712/4, 87905/4])';

% Allocate correct liwing expenses to each quarter in price_houses set
n = 4;
liv_exp_apartments =
[repmat(liv_exp_a_70_Q(1),n,1);repmat(liv_exp_a_70_Q(2),n,1);repmat(liv_exp_a_70_Q(3),
n,1);repmat(liv_exp_a_70_Q(4),n,1);repmat(liv_exp_a_70_Q(5),n,1);repmat(liv_exp_a_70_Q
(6),n,1);repmat(liv_exp_a_70_Q(7),n,1);repmat(liv_exp_a_70_Q(8),n,1);repmat(liv_exp_a_
70_Q(9),n,1);repmat(liv_exp_a_70_Q(10),n,1);repmat(liv_exp_a_70_Q(11),n,1)];
price_inner_western.ordered_liv_exp_70 = repmat(liv_exp_apartments,(length(C)),1);

% Compute living expenses based on a reference apartment of 70 sqm
price_inner_western.living_exp =
(price_inner_western.prom./70).*price_inner_western.ordered_liv_exp_70;
avg_living_exp_Q =
mean(price_inner_western.living_exp./price_inner_western.estimated_price);

% Allocate correct implicit rents
```

```
implicit_rent     = zeros(height(price_inner_western),1);
rental_allocation = repmat(Rent_Data, length(C), 1);

implicit_rent(find(price_inner_western.Soverom == 0)) =
rental_allocation.room1(find(price_inner_western.Soverom == 0));
implicit_rent(find(price_inner_western.Soverom == 1)) =
rental_allocation.room1(find(price_inner_western.Soverom == 1));
implicit_rent(find(price_inner_western.Soverom == 2)) =
rental_allocation.room2(find(price_inner_western.Soverom == 2));
implicit_rent(find(price_inner_western.Soverom == 3)) =
rental_allocation.room3(find(price_inner_western.Soverom == 3));
implicit_rent(find(price_inner_western.Soverom == 4)) =
rental_allocation.room4(find(price_inner_western.Soverom == 4));
implicit_rent(find(price_inner_western.Soverom >= 5)) =
rental_allocation.room5_(find(price_inner_western.Soverom >= 5));

price_inner_western.implicit_rent = implicit_rent;
avg_implicit_rent_Q =
mean(price_inner_western.implicit_rent./price_inner_western.estimated_price);

% Implement tax rates for sample period
tax_rates = ([ 0.28, 0.28, 0.28, 0.28, 0.27, 0.27, 0.25, 0.24, 0.23, 0.22, 0.22])';
n = 4;
tax_rates_order =
[repmat(tax_rates(1),n,1);repmat(tax_rates(2),n,1);repmat(tax_rates(3),n,1);repmat(tax
_rates(4),n,1);repmat(tax_rates(5),n,1);repmat(tax_rates(6),n,1);repmat(tax_rates(7),n
,1);repmat(tax_rates(8),n,1);repmat(tax_rates(9),n,1);repmat(tax_rates(10),n,1);repmat
(tax_rates(11),n,1)];

% Allocate correct tax rates to each quarter in price_houses set
price_inner_western.tax_rates = repmat(tax_rates_order,(length(C)),1);

% Create a table to store different cost rates
avg_cost_rates_inner_western                          = zeros(4,3); %JUSTERE DENNE
ETTERSOM HVORDAN ST√òRRELSEN VIL BLI
avg_cost_rates_inner_western                          =
array2table(avg_cost_rates_inner_western);
avg_cost_rates_inner_western.Properties.VariableNames = {'Average_cost_rates',
'Yearly', 'Quarterly'};
avg_cost_rates_inner_western.Average_cost_rates       =
categorical(avg_cost_rates_inner_western.Average_cost_rates);

% Import stats for cost rates
avg_cost_rates_inner_western.Average_cost_rates(1) = 't_cost_buyer';
avg_cost_rates_inner_western.Yearly(1)             = t_cost_rate;
avg_cost_rates_inner_western.Quarterly(1)          = t_cost_rate_Q;

avg_cost_rates_inner_western.Average_cost_rates(2) = 't_cost_seller';
avg_cost_rates_inner_western.Yearly(2)             = t_cost_s_rate;
avg_cost_rates_inner_western.Quarterly(2)          = t_cost_s_rate_Q;

avg_cost_rates_inner_western.Average_cost_rates(3) = 'living_expenses';
avg_cost_rates_inner_western.Yearly(3)             = avg_living_exp_Q * 4;
avg_cost_rates_inner_western.Quarterly(3)          = avg_living_exp_Q;

avg_cost_rates_inner_western.Average_cost_rates(4) = 'implicit_rent';
avg_cost_rates_inner_western.Yearly(4)             = avg_implicit_rent_Q * 4;
avg_cost_rates_inner_western.Quarterly(4)          = avg_implicit_rent_Q;

% 5. ALLOCATE TABLE FOR PRICE INDICES
priceindex_inner_western          = table();
Y_index                           = Y_order(1:44);
Q_index                           = Q_rep(1:44);
priceindex_inner_western.Q_order  = string(Y_index(:,1)) + Q_index(:,1);
priceindex_inner_western.Q_order  = categorical(priceindex_inner_western.Q_order);

% Create a table to store returns, std, risk free, sharpe
sum_stats_inner_western                           = zeros(10,7); %JUSTERE DENNE
ETTERSOM HVORDAN ST√òRRELSEN VIL BLI
sum_stats_inner_western                           =
array2table(sum_stats_inner_western);
sum_stats_inner_western.Properties.VariableNames = {'House_Indices', 'Return',
'std_dev','Excess_return','Sharpe_ratio', 'Skewness', 'Kurtosis'};
```

```
sum_stats_inner_western.House_Indices            =
categorical(sum_stats_inner_western.House_Indices);

% Import risk-free rates (innskuddsrente)
n = 4;
priceindex_inner_western.rf = zeros(44,1);
priceindex_inner_western.rf =
[repmat(Rf.rf(1),n,1);repmat(Rf.rf(2),n,1);repmat(Rf.rf(3),n,1);repmat(Rf.rf(4),n,1);r
epmat(Rf.rf(5),n,1);repmat(Rf.rf(6),n,1);repmat(Rf.rf(7),n,1);repmat(Rf.rf(8),n,1);rep
mat(Rf.rf(9),n,1);repmat(Rf.rf(10),n,1);repmat(Rf.rf(11),n,1)];
priceindex_inner_western.rf = priceindex_inner_western.rf./4;

% 6. COMPUTATION OF MEAN PRICES FOR HOUSES AND PRICE INDEX BASED ON MEAN PRICES
% Calculate mean prices for houses
priceindex_inner_western.mean_prices = zeros(height(priceindex_inner_western), end);
for i = 1:length(priceindex_inner_western.Q_order)
    priceindex_inner_western.mean_prices(i) =
mean(price_inner_western.estimated_price(find(price_inner_western.Q ==
priceindex_inner_western.Q_order(i))));
end

% Calculate price index for houses based on mean prices
priceindex_inner_western.index_values = zeros(height(priceindex_inner_western), end);
for i = 1:length(priceindex_inner_western.index_values)
    priceindex_inner_western.index_values(i) =
priceindex_inner_western.mean_prices(i)/priceindex_inner_western.mean_prices(1);
end

% 7. COMPUTATION OF QUARTERLY RETURNS ON ASSET/HOUSE FROM PERIOD TO PERIOD, NO COSTS
% Compute quarterly returns for each unique ID
price_inner_western.Q_ret_no_costs =
(price_inner_western.estimated_price./price_inner_western.lagged_estimated_price)-1;
price_inner_western.Q_ret_no_costs(1:44:end) = NaN;

% Compute mean quarterly returns for houses (mean of all IDs per quarter) (no costs)
priceindex_inner_western.Q_ret_no_costs = zeros(height(priceindex_inner_western),
end);

for i = 1:height(priceindex_inner_western)
    priceindex_inner_western.Q_ret_no_costs(i) =
nanmean(price_inner_western.Q_ret_no_costs(find(price_inner_western.Q ==
priceindex_inner_western.Q_order(i))));
end

% Compute mean of mean quarterly returns for houses (no costs)
mean_Q_ret_no_costs = nanmean(priceindex_inner_western.Q_ret_no_costs);

% Compute excess quarterly returns for houses (no costs)
priceindex_inner_western.excess_Q_ret_no_costs =
priceindex_inner_western.Q_ret_no_costs-priceindex_inner_western.rf;

% Compute mean excess quarterly returns for houses (no costs)
mean_excess_Q_ret_no_costs = nanmean(priceindex_inner_western.excess_Q_ret_no_costs);

% Compute standard deviation for each unique ID
st_Q_ret_no_costs= zeros(length(C),1);
time(1,:) = clock;
for i = 1:length(C)
    if b(i)<b(length(C))
       st_Q_ret_no_costs(i) =
std(price_inner_western.Q_ret_no_costs(((b(i)+1):(b(i+1)-1))));
    else
       st_Q_ret_no_costs(i) =
std(price_inner_western.Q_ret_no_costs((b(length(C))+1):height(price_inner_western)));
    end
time(i+1,:) = clock;
fprintf('\nTo complete %.f/%.f iterations (unique IDs) it took %.2f seconds\n', i,
length(C), etime(time(i+1,:), time(1,:)));
end

% Standard deviation of all quarterly returns
std_Q_ret_no_costs = mean(st_Q_ret_no_costs);

% Import stats for 'Q_ret_no_costs' in summary table
```

```
sum_stats_inner_western.House_Indices(1) = 'Q_ret_no_costs';
sum_stats_inner_western.Return(1)        = mean_Q_ret_no_costs;
sum_stats_inner_western.std_dev(1)       = std_Q_ret_no_costs;
sum_stats_inner_western.Excess_return(1) = mean_excess_Q_ret_no_costs;
sum_stats_inner_western.Sharpe_ratio(1) =
sum_stats_inner_western.Excess_return(1)/sum_stats_inner_western.std_dev(1);
sum_stats_inner_western.Skewness(1)      =
skewness(priceindex_inner_western.Q_ret_no_costs(2:44));
sum_stats_inner_western.Kurtosis(1)      =
kurtosis(priceindex_inner_western.Q_ret_no_costs(2:44));

% COMPUTE CUMULATIVE ROA(NC)
first_obs_price = price_inner_western.estimated_price(1:44:end);
price_inner_western.first_obs_price_repeated = repelem(first_obs_price, 44);

price_inner_western.cum_Q_ret_no_costs =
(price_inner_western.estimated_price./price_inner_western.first_obs_price_repeated)-1;
price_inner_western.cum_Q_ret_no_costs(1:44:end) = NaN;

priceindex_inner_western.cum_Q_ret_no_costs =
zeros(height(priceindex_inner_western),end);
% Compute mean cumulative ROA(NC)
for i = 1:height(priceindex_inner_western)
    priceindex_inner_western.cum_Q_ret_no_costs(i) =
nanmean(price_inner_western.cum_Q_ret_no_costs(find(price_inner_western.Q ==
priceindex_inner_western.Q_order(i))));
end

% 8. COMPUTATION OF QUARTERLY RETURNS ON ASSET/HOUSE FROM PERIOD TO PERIOD (BUY/SELL),
INCL TOTAL COSTS
% Compute quarterly returns for each unique ID
price_inner_western.Q_ret_tot_costs = (((price_inner_western.estimated_price -
price_inner_western.t_cost_s - price_inner_western.lagged_t_cost -
price_inner_western.lagged_estimated_price).*(1-price_inner_western.tax_rates)) -
price_inner_western.living_exp +
price_inner_western.implicit_rent)./(price_inner_western.lagged_estimated_price);
price_inner_western.Q_ret_tot_costs(1:44:end) = NaN;

% Compute mean quarterly returns for houses (mean of all IDs per quarter) (inc tot
costs)
priceindex_inner_western.Q_ret_tot_costs = zeros(height(priceindex_inner_western),
end);

for i = 1:length(priceindex_inner_western.mean_prices)
    priceindex_inner_western.Q_ret_tot_costs(i) =
mean(price_inner_western.Q_ret_tot_costs(find(price_inner_western.Q ==
priceindex_inner_western.Q_order(i))));
end

% Compute mean of mean quarterly returns for houses (inc tot costs)
mean_Q_ret_tot_costs = nanmean(priceindex_inner_western.Q_ret_tot_costs);

% Compute excess quarterly returns for houses (inc tot costs)
priceindex_inner_western.excess_Q_ret_tot_costs =
priceindex_inner_western.Q_ret_tot_costs-priceindex_inner_western.rf;

% Compute mean excess quarterly returns for houses (inc tot costs)
mean_excess_Q_ret_tot_costs =
nanmean(priceindex_inner_western.excess_Q_ret_tot_costs);

% Compute standard deviation for each unique IDs returns
st_Q_ret_tot_costs= zeros(length(C),1);
time(1,:) = clock;
for i = 1:length(C)
    if b(i)<b(length(C))
       st_Q_ret_tot_costs(i) =
std(price_inner_western.Q_ret_tot_costs(((b(i)+1):(b(i+1)-1))));
    else
       st_Q_ret_tot_costs(i) =
std(price_inner_western.Q_ret_tot_costs((b(length(C))+1):height(price_inner_western)))
;
    end
time(i+1,:) = clock;
```

```
fprintf('\nTo complete %.f/%.f iterations (unique IDs) it took %.2f seconds\n', i,
length(C), etime(time(i+1,:), time(1,:)));
end

% Standard deviation of all quarterly returns
std_Q_ret_tot_costs = mean(st_Q_ret_tot_costs);

% Import stats for 'Q_ret_tot_costs' in summary table
sum_stats_inner_western.House_Indices(2) = 'Q_ret_tot_costs';
sum_stats_inner_western.Return(2)        = mean_Q_ret_tot_costs;
sum_stats_inner_western.std_dev(2)       = std_Q_ret_tot_costs;
sum_stats_inner_western.Excess_return(2) = mean_excess_Q_ret_tot_costs;
sum_stats_inner_western.Sharpe_ratio(2) =
sum_stats_inner_western.Excess_return(2)/sum_stats_inner_western.std_dev(2);
sum_stats_inner_western.Skewness(2)      =
skewness(priceindex_inner_western.Q_ret_tot_costs(2:44));
sum_stats_inner_western.Kurtosis(2)      =
kurtosis(priceindex_inner_western.Q_ret_tot_costs(2:44));

% COMPUTE CUMULATIVE ROA (TC)
taxes_TC = (price_inner_western.estimated_price - price_inner_western.t_cost_s -
price_inner_western.lagged_t_cost -
price_inner_western.lagged_estimated_price).*price_inner_western.tax_rates;
taxes_TC(1:44:end) = NaN;

net_costs_TC = (price_inner_western.living_exp + price_inner_western.lagged_t_cost +
price_inner_western.t_cost_s + taxes_TC - price_inner_western.implicit_rent);
net_costs_TC(1:44:end) = NaN;

% Compute cumulative costs
cumcosts_TC = zeros(height(price_inner_western), end);
time(1,:) = clock;
for i = 1:length(C)
    if b(i)<b(length(C))
       cumcosts_TC((b(i)):(b(i+1)-1)) = cumsum(net_costs_TC((b(i)):(b(i+1)-
1)),'omitnan');
    else
       cumcosts_TC(b(length(C)):height(price_inner_western)) =
cumsum(net_costs_TC(b(length(C)):height(price_inner_western)),'omitnan');
    end
time(i+1,:) = clock;
fprintf('\nTo complete %.f/%.f iterations (unique IDs) it took %.2f seconds\n', i,
length(C), etime(time(i+1,:), time(1,:)));
end

price_inner_western.cum_Q_ret_tot_costs = (price_inner_western.estimated_price -
cumcosts_TC -
price_inner_western.first_obs_price_repeated)./price_inner_western.first_obs_price_rep
eated;
price_inner_western.cum_Q_ret_tot_costs(1:44:end) = NaN;

priceindex_inner_western.cum_Q_ret_tot_costs =
zeros(height(priceindex_inner_western),end);
% Compute mean cumulative returns for houses (mean of all IDs per quarter) (no costs)
for i = 1:height(priceindex_inner_western)
    priceindex_inner_western.cum_Q_ret_tot_costs(i) =
nanmean(price_inner_western.cum_Q_ret_tot_costs(find(price_inner_western.Q ==
priceindex_inner_western.Q_order(i))));
end

all_lost = priceindex_inner_western.cum_Q_ret_tot_costs < -1;
priceindex_inner_western.cum_Q_ret_tot_costs(all_lost) = -1;

% 9. COMPUTATION OF QUARTERLY RETURNS ON ASSET/HOUSE FROM PERIOD TO PERIOD , INCL
SMOOTHED COSTS
% Locate buyer and sellers cost when buying in 2010Q1 and selling in 2020Q4
% Find buyer and seller cost per quarter for sample period
cost_2010Q1 = price_inner_western.t_cost(1:44:end);
cost_2010Q1 = cost_2010Q1/44;
price_inner_western.cost_2010Q1 = repelem(cost_2010Q1, 44);

cost_2020Q4 = price_inner_western.t_cost_s(44:44:end);
cost_2020Q4 = cost_2020Q4/44;
price_inner_western.cost_2020Q4 = repelem(cost_2020Q4, 44);
```

```
% Compute quarterly returns given that you buy in 2010Q1 and sell in 2020Q4
% and spred the relevant costs on the sample period.
price_inner_western.Q_ret_smooth_costs = (price_inner_western.estimated_price -
price_inner_western.cost_2010Q1 - price_inner_western.cost_2020Q4 -
price_inner_western.living_exp - price_inner_western.lagged_estimated_price +
price_inner_western.implicit_rent)./price_inner_western.lagged_estimated_price;
price_inner_western.Q_ret_smooth_costs(1:44:end) = NaN;

% Compute mean quarterly returns for houses (mean of all IDs per quarter) (inc smooth
costs)
priceindex_inner_western.Q_ret_smooth_costs = zeros(height(priceindex_inner_western),
end);

for i = 1:length(priceindex_inner_western.mean_prices)
    priceindex_inner_western.Q_ret_smooth_costs(i) =
mean(price_inner_western.Q_ret_smooth_costs(find(price_inner_western.Q ==
priceindex_inner_western.Q_order(i))));
end

% Compute mean of mean quarterly returns for houses (inc smooth costs)
mean_Q_ret_smooth_costs = nanmean(priceindex_inner_western.Q_ret_smooth_costs);

% Compute excess quarterly returns for houses (inc smooth costs)
priceindex_inner_western.excess_Q_ret_smooth_costs =
priceindex_inner_western.Q_ret_smooth_costs - priceindex_inner_western.rf;

% Compute mean excess quarterly returns for houses (inc smooth costs)
mean_excess_Q_ret_smooth_costs =
nanmean(priceindex_inner_western.excess_Q_ret_smooth_costs);

% Compute standard deviation for each unique IDs returns
st_Q_ret_smooth_costs= zeros(length(C),1);
time(1,:) = clock;
for i = 1:length(C)
    if b(i)<b(length(C))
       st_Q_ret_smooth_costs(i) =
std(price_inner_western.Q_ret_smooth_costs(((b(i)+1):(b(i+1)-1))));
    else
       st_Q_ret_smooth_costs(i) =
std(price_inner_western.Q_ret_smooth_costs((b(length(C))+1):height(price_inner_western
)));
    end
time(i+1,:) = clock;
fprintf('\nTo complete %.f/%.f iterations (unique IDs) it took %.2f seconds\n', i,
length(C), etime(time(i+1,:), time(1,:)));
end

% Standard deviation of all quarterly returns
std_Q_ret_smooth_costs = mean(st_Q_ret_smooth_costs);

% Import stats for 'Q_ret_smooth_costs' in summary table
sum_stats_inner_western.House_Indices(3) = 'Q_ret_smooth_costs';
sum_stats_inner_western.Return(3)        = mean_Q_ret_smooth_costs;
sum_stats_inner_western.std_dev(3)       = std_Q_ret_smooth_costs;
sum_stats_inner_western.Excess_return(3) = mean_excess_Q_ret_smooth_costs;
sum_stats_inner_western.Sharpe_ratio(3) =
sum_stats_inner_western.Excess_return(3)/sum_stats_inner_western.std_dev(3);
sum_stats_inner_western.Skewness(3)      =
skewness(priceindex_inner_western.Q_ret_smooth_costs(2:44));
sum_stats_inner_western.Kurtosis(3)      =
kurtosis(priceindex_inner_western.Q_ret_smooth_costs(2:44));

% COMPUTE CUMULATIVE ROA(SC)
net_costs_SC = (price_inner_western.living_exp + price_inner_western.cost_2010Q1 +
price_inner_western.cost_2020Q4 - price_inner_western.implicit_rent);
net_costs_SC(1:44:end) = NaN;

cumcosts_SC = zeros(height(price_inner_western), end);
time(1,:) = clock;
for i = 1:length(C)
    if b(i)<b(length(C))
       cumcosts_SC((b(i)):(b(i+1)-1)) = cumsum(net_costs_SC((b(i)):(b(i+1)-
1)),'omitnan');
```

```
    else
       cumcosts_SC(b(length(C)):height(price_inner_western)) =
cumsum(net_costs_SC(b(length(C)):height(price_inner_western)),'omitnan');
    end
time(i+1,:) = clock;
fprintf('\nTo complete %.f/%.f iterations (unique IDs) it took %.2f seconds\n', i,
length(C), etime(time(i+1,:), time(1,:)));
end

price_inner_western.cum_Q_ret_smooth_costs = (price_inner_western.estimated_price -
cumcosts_SC -
price_inner_western.first_obs_price_repeated)./price_inner_western.first_obs_price_rep
eated;
price_inner_western.cum_Q_ret_smooth_costs(1:44:end) = NaN;

% Compute mean cumulative returns for houses (mean of all IDs per quarter) (no costs)
priceindex_inner_western.cum_Q_ret_smooth_costs =
zeros(height(priceindex_inner_western),end);
for i = 1:height(priceindex_inner_western)
    priceindex_inner_western.cum_Q_ret_smooth_costs(i) =
nanmean(price_inner_western.cum_Q_ret_smooth_costs(find(price_inner_western.Q ==
priceindex_inner_western.Q_order(i))));
end

% Identify average loan-to-value ratio for housing investments
l = ([0.6957, 0.6957, 0.6957, 0.6957, 0.71, 0.76, 0.67, 0.66, 0.68, 0.70, 0.69])';
n = 4;
ltv_ratio_order =
[repmat(l(1),n,1);repmat(l(2),n,1);repmat(l(3),n,1);repmat(l(4),n,1);repmat(l(5),n,1);
repmat(l(6),n,1);repmat(l(7),n,1);repmat(l(8),n,1);repmat(l(9),n,1);repmat(l(10),n,1);
repmat(l(11),n,1)];

% Allocate correct ltv-rates to each quarter in price_houses set
price_inner_western.ltvr        = repmat(ltv_ratio_order,(length(C)),1);
price_inner_western.lagged_ltvr = lagmatrix(price_inner_western.ltvr,1);

% Allocate ltv ratio as of 2010Q1 to each quarter in price_houses set
price_inner_western.ltvr_2010Q1(:) = price_inner_western.ltvr(1);

% 10. WE DONT USE THE RETURN SERIES COMPUTED IN SECTION 10
% Compute quarterly equity returns for each unique ID (no costs)
price_inner_western.Q_eq_ret_no_costs = ((price_inner_western.estimated_price -
price_inner_western.lagged_estimated_price)./(price_inner_western.lagged_estimated_pri
ce.*(1-price_inner_western.lagged_ltvr)));
price_inner_western.Q_eq_ret_no_costs(1:44:end) = NaN;

% Compute mean quarterly equity returns for houses (mean of all IDs per quarter) (no
costs)
priceindex_inner_western.Q_eq_ret_no_costs = zeros(height(priceindex_inner_western),
end);

for i = 1:height(priceindex_inner_western)
    priceindex_inner_western.Q_eq_ret_no_costs(i) =
mean(price_inner_western.Q_eq_ret_no_costs(find(price_inner_western.Q ==
priceindex_inner_western.Q_order(i))));
end

% Compute mean of mean quarterly equity returns for houses (no costs)
mean_Q_eq_ret_no_costs = nanmean(priceindex_inner_western.Q_eq_ret_no_costs);

% Compute excess quarterly equity returns for houses (no costs)
priceindex_inner_western.excess_Q_eq_ret_no_costs =
priceindex_inner_western.Q_eq_ret_no_costs - priceindex_inner_western.rf;

% Compute mean excess quarterly equity returns for houses (no costs)
mean_excess_Q_eq_ret_no_costs =
nanmean(priceindex_inner_western.excess_Q_eq_ret_no_costs);

% Compute standard deviation for each unique IDs returns
st_Q_eq_ret_no_costs= zeros(length(C),1);
time(1,:) = clock;
for i = 1:length(C)
    if b(i)<b(length(C))
```

```
        st_Q_eq_ret_no_costs(i) =
std(price_inner_western.Q_eq_ret_no_costs(((b(i)+1):(b(i+1)-1))));
    else
        st_Q_eq_ret_no_costs(i) =
std(price_inner_western.Q_eq_ret_no_costs((b(length(C))+1):height(price_inner_western)
));
    end
time(i+1,:) = clock;
fprintf('\nTo complete %.f/%.f iterations (unique IDs) it took %.2f seconds\n', i,
length(C), etime(time(i+1,:), time(1,:)));
end

% Standard deviation of all quarterly equity returns
std_Q_eq_ret_no_costs = mean(st_Q_eq_ret_no_costs);

% Import stats for 'Q_eq_ret_no_costs' in summary table
sum_stats_inner_western.House_Indices(4) = 'Q_eq_ret_no_costs';
sum_stats_inner_western.Return(4)        = mean_Q_eq_ret_no_costs;
sum_stats_inner_western.std_dev(4)       = std_Q_eq_ret_no_costs;
sum_stats_inner_western.Excess_return(4) = mean_excess_Q_eq_ret_no_costs;
sum_stats_inner_western.Sharpe_ratio(4) =
sum_stats_inner_western.Excess_return(4)/sum_stats_inner_western.std_dev(4);
sum_stats_inner_western.Skewness(4)      =
skewness(priceindex_inner_western.Q_eq_ret_no_costs(2:44));
sum_stats_inner_western.Kurtosis(4)      =
kurtosis(priceindex_inner_western.Q_eq_ret_no_costs(2:44));

% COMPUTE CUMULATIVE ROE (NC)
price_inner_western.cum_Q_eq_ret_no_costs = ((price_inner_western.estimated_price -
price_inner_western.first_obs_price_repeated)./(price_inner_western.first_obs_price_re
peated.*(1-price_inner_western.ltvr_2010Q1)));
price_inner_western.cum_Q_eq_ret_no_costs(1:44:end) = NaN;

% Compute mean cumulative returns for houses (mean of all IDs per quarter) (no costs)
priceindex_inner_western.cum_Q_eq_ret_no_costs =
zeros(height(priceindex_inner_western),end);
for i = 1:height(priceindex_inner_western)
    priceindex_inner_western.cum_Q_eq_ret_no_costs(i) =
nanmean(price_inner_western.cum_Q_eq_ret_no_costs(find(price_inner_western.Q ==
priceindex_inner_western.Q_order(i))));
end

% 11. WE DONT USE THE RETURN SERIES COMPUTED IN SECTION 11
% Compute quarterly equity returns for each unique ID (inc tot costs)
price_inner_western.Q_eq_ret_tot_costs = (((price_inner_western.estimated_price -
price_inner_western.t_cost_s - price_inner_western.lagged_t_cost -
price_inner_western.lagged_estimated_price).*(1-price_inner_western.tax_rates)) -
price_inner_western.living_exp +
price_inner_western.implicit_rent)./(price_inner_western.lagged_estimated_price.*(1-
price_inner_western.lagged_ltvr));
price_inner_western.Q_eq_ret_tot_costs(1:44:end) = NaN;

% Compute mean quarterly equity returns for houses (mean of all IDs per quarter) (inc
tot costs)
priceindex_inner_western.Q_eq_ret_tot_costs = zeros(height(priceindex_inner_western),
end);

for i = 1:length(priceindex_inner_western.mean_prices)
    priceindex_inner_western.Q_eq_ret_tot_costs(i) =
mean(price_inner_western.Q_eq_ret_tot_costs(find(price_inner_western.Q ==
priceindex_inner_western.Q_order(i))));
end

% Compute mean of mean quarterly equity returns for houses (inc tot costs)
mean_Q_eq_ret_tot_costs = nanmean(priceindex_inner_western.Q_eq_ret_tot_costs);

% Compute excess quarterly equity returns for houses (inc tot costs)
priceindex_inner_western.excess_Q_eq_ret_tot_costs =
priceindex_inner_western.Q_eq_ret_tot_costs - priceindex_inner_western.rf;

% Compute mean excess quarterly equity returns for houses (inc tot costs)
mean_excess_Q_eq_ret_tot_costs =
nanmean(priceindex_inner_western.excess_Q_eq_ret_tot_costs);
```

```
% Compute standard deviation for each unique IDs returns
st_Q_eq_ret_tot_costs= zeros(length(C),1);
time(1,:) = clock;
for i = 1:length(C)
    if b(i)<b(length(C))
       st_Q_eq_ret_tot_costs(i) =
std(price_inner_western.Q_eq_ret_tot_costs(((b(i)+1):(b(i+1)-1))));
    else
       st_Q_eq_ret_tot_costs(i) =
std(price_inner_western.Q_eq_ret_tot_costs((b(length(C))+1):height(price_inner_western
)));
    end
time(i+1,:) = clock;
fprintf('\nTo complete %.f/%.f iterations (unique IDs) it took %.2f seconds\n', i,
length(C), etime(time(i+1,:), time(1,:)));
end

% Standard deviation of all quarterly equity returns
std_Q_eq_ret_tot_costs = mean(st_Q_eq_ret_tot_costs);

% Import stats for 'Q_eq_ret_tot_costs' in summary table
sum_stats_inner_western.House_Indices(5) = 'Q_eq_ret_tot_costs';
sum_stats_inner_western.Return(5)        = mean_Q_eq_ret_tot_costs;
sum_stats_inner_western.std_dev(5)       = std_Q_eq_ret_tot_costs;
sum_stats_inner_western.Excess_return(5) = mean_excess_Q_eq_ret_tot_costs;
sum_stats_inner_western.Sharpe_ratio(5) =
sum_stats_inner_western.Excess_return(5)/sum_stats_inner_western.std_dev(5);
sum_stats_inner_western.Skewness(5)      =
skewness(priceindex_inner_western.Q_eq_ret_tot_costs(2:44));
sum_stats_inner_western.Kurtosis(5)      =
kurtosis(priceindex_inner_western.Q_eq_ret_tot_costs(2:44));

% COMPUTE CUMULATIVE RETURNS BASED ON QUARTERLY EQUITY RETURNS FROM PRICE INDEX (INC
TOT COSTS)
priceindex_inner_western.cum_Q_eq_ret_tot_costs =
zeros(height(priceindex_inner_western),end);
priceindex_inner_western.cum_Q_eq_ret_tot_costs =
cumprod((1+priceindex_inner_western.Q_eq_ret_tot_costs),'omitnan') -1;

% 12. COMPUTATION OF QUARTERLY EQUITY RETURNS ON ASSET/HOUSE FROM PERIOD TO PERIOD ,
INCL SMOOTHED COSTS
% Compute quarterly EQUITY returns given that you buy in 2010Q1 and sell in 2020Q4
% and spread the relevant costs on the sample period.
price_inner_western.Q_eq_ret_smooth_costs = (price_inner_western.estimated_price -
price_inner_western.cost_2010Q1 - price_inner_western.cost_2020Q4 -
price_inner_western.living_exp - price_inner_western.lagged_estimated_price +
price_inner_western.implicit_rent)./(price_inner_western.lagged_estimated_price.*(1-
price_inner_western.ltvr_2010Q1));
price_inner_western.Q_eq_ret_smooth_costs(1:44:end) = NaN;

% Compute mean quarterly equity returns for houses (mean of all IDs per quarter) (inc
smooth costs)
priceindex_inner_western.Q_eq_ret_smooth_costs =
zeros(height(priceindex_inner_western), end);

for i = 1:length(priceindex_inner_western.mean_prices)
    priceindex_inner_western.Q_eq_ret_smooth_costs(i) =
mean(price_inner_western.Q_eq_ret_smooth_costs(find(price_inner_western.Q ==
priceindex_inner_western.Q_order(i))));
end

% Compute mean of mean quarterly equity returns for houses (inc smooth costs)
mean_Q_eq_ret_smooth_costs = nanmean(priceindex_inner_western.Q_eq_ret_smooth_costs);

% Compute excess quarterly equity returns for houses (inc smooth costs)
priceindex_inner_western.excess_Q_eq_ret_smooth_costs =
priceindex_inner_western.Q_eq_ret_smooth_costs - priceindex_inner_western.rf;

% Compute mean excess quarterly equity returns for houses (inc smooth costs)
mean_excess_Q_eq_ret_smooth_costs =
nanmean(priceindex_inner_western.excess_Q_eq_ret_smooth_costs);

% Compute standard deviation for each unique IDs returns
st_Q_eq_ret_smooth_costs= zeros(length(C),1);
```

```
time(1,:) = clock;
for i = 1:length(C)
    if b(i)<b(length(C))
       st_Q_eq_ret_smooth_costs(i) =
std(price_inner_western.Q_eq_ret_smooth_costs(((b(i)+1):(b(i+1)-1))));
    else
       st_Q_eq_ret_smooth_costs(i) =
std(price_inner_western.Q_eq_ret_smooth_costs((b(length(C))+1):height(price_inner_west
ern)));
    end
time(i+1,:) = clock;
fprintf('\nTo complete %.f/%.f iterations (unique IDs) it took %.2f seconds\n', i,
length(C), etime(time(i+1,:), time(1,:)));
end

% Standard deviation of all quarterly equity returns
std_Q_eq_ret_smooth_costs = mean(st_Q_eq_ret_smooth_costs);

% Import stats for 'Q_eq_ret_smooth_costs' in summary table
sum_stats_inner_western.House_Indices(6) = 'Q_eq_ret_smooth_costs';
sum_stats_inner_western.Return(6)        = mean_Q_eq_ret_smooth_costs;
sum_stats_inner_western.std_dev(6)       = std_Q_eq_ret_smooth_costs;
sum_stats_inner_western.Excess_return(6) = mean_excess_Q_eq_ret_smooth_costs;
sum_stats_inner_western.Sharpe_ratio(6)  =
sum_stats_inner_western.Excess_return(6)/sum_stats_inner_western.std_dev(6);
sum_stats_inner_western.Skewness(6)      =
skewness(priceindex_inner_western.Q_eq_ret_smooth_costs(2:44));
sum_stats_inner_western.Kurtosis(6)      =
kurtosis(priceindex_inner_western.Q_eq_ret_smooth_costs(2:44));

% COMPUTE CUMULATIVE ROE(SC)
price_inner_western.cum_Q_eq_ret_smooth_costs = (price_inner_western.estimated_price -
cumcosts_SC -
price_inner_western.first_obs_price_repeated)./(price_inner_western.first_obs_price_re
peated.*(1-price_inner_western.ltvr_2010Q1));
price_inner_western.cum_Q_eq_ret_smooth_costs(1:44:end) = NaN;

% Compute mean cumulative returns for houses (mean of all IDs per quarter) (smooth
costs)
priceindex_inner_western.cum_Q_eq_ret_smooth_costs =
zeros(height(priceindex_inner_western),end);
for i = 1:height(priceindex_inner_western)
    priceindex_inner_western.cum_Q_eq_ret_smooth_costs(i) =
nanmean(price_inner_western.cum_Q_eq_ret_smooth_costs(find(price_inner_western.Q ==
priceindex_inner_western.Q_order(i))));
end
% MAKE PLOT OF THE RELEVANT INDICES TOGHETER, MAKE MULTIPLE PLOTS FOR COMPARISON
REASONS
inner_western_price_changes = priceindex_inner_western(:,[1 5 8 11 14 17 20]);

inner_western_price_indices = priceindex_inner_western(:,[1 7 10 13 16 19 22]);
inner_western_price_indices.cum_Q_ret_no_costs(1) = 0;
inner_western_price_indices.cum_Q_ret_tot_costs(1) = 0;
inner_western_price_indices.cum_Q_ret_smooth_costs(1) = 0;
inner_western_price_indices.cum_Q_eq_ret_no_costs(1) = 0;
inner_western_price_indices.cum_Q_eq_ret_tot_costs(1) = 0;
inner_western_price_indices.cum_Q_eq_ret_smooth_costs(1) = 0;

inner_western_price_indices.cum_Q_ret_no_costs        =
inner_western_price_indices.cum_Q_ret_no_costs + 1;
inner_western_price_indices.cum_Q_ret_tot_costs       =
inner_western_price_indices.cum_Q_ret_tot_costs + 1;
inner_western_price_indices.cum_Q_ret_smooth_costs    =
inner_western_price_indices.cum_Q_ret_smooth_costs + 1;
inner_western_price_indices.cum_Q_eq_ret_no_costs     =
inner_western_price_indices.cum_Q_eq_ret_no_costs + 1;
inner_western_price_indices.cum_Q_eq_ret_tot_costs    =
inner_western_price_indices.cum_Q_eq_ret_tot_costs + 1;
inner_western_price_indices.cum_Q_eq_ret_smooth_costs =
inner_western_price_indices.cum_Q_eq_ret_smooth_costs + 1;

clearvars -except Property_Data summary_table Rf Rent_Data...
                  house_set hedonic_houses housing_model...
                  price_houses uniq_house_set avg_cost_rates_houses...
```

```
                priceindex_houses sum_stats_houses houseprice_changes...
                houseprice_indices...
                smallhouse_set hedonic_smallhouses smallhouse_model...
                price_smallhouses uniq_smallhouse_set avg_cost_rates_smallhouses...
                priceindex_smallhouses sum_stats_smallhouses...
                smallhouse_price_changes smallhouse_price_indices...
                apartment_set hedonic_apartments apartment_model price_apartments...
                uniq_apartment_set avg_cost_rates_apartments
priceindex_apartments...
                sum_stats_apartments apartment_price_changes...
                apartment_price_indices...
                inner_eastern_set hedonic_inner_eastern inner_eastern_model...
                price_inner_eastern uniq_inner_eastern_set
avg_cost_rates_inner_eastern...
                priceindex_inner_eastern sum_stats_inner_eastern
inner_eastern_price_changes...
                inner_eastern_price_indices...
                inner_western_set hedonic_inner_western inner_western_model...
                price_inner_western uniq_inner_western_set
avg_cost_rates_inner_western...
                priceindex_inner_western sum_stats_inner_western
inner_western_price_changes...
                inner_western_price_indices...
                outer_eastern_set hedonic_outer_eastern outer_eastern_model...
                price_outer_eastern uniq_outer_eastern_set
avg_cost_rates_outer_eastern...
                priceindex_outer_eastern sum_stats_outer_eastern
outer_eastern_price_changes...
                outer_eastern_price_indices...
                outer_western_set hedonic_outer_western outer_western_model...
                price_outer_western uniq_outer_western_set
avg_cost_rates_outer_western...
                priceindex_outer_western sum_stats_outer_western
outer_western_price_changes...
                outer_western_price_indices...
                outer_southern_set hedonic_outer_southern outer_southern_model...
                price_outer_southern uniq_outer_southern_set
avg_cost_rates_outer_southern...
                priceindex_outer_southern sum_stats_outer_southern
outer_southern_price_changes...
                outer_southern_price_indices...
                property_price_changes property_price_indices hedonic_total...
                ROA_NC_stats ROE_NC_stats ROA_TC_stats ROE_TC_stats...
                ROA_SC_stats ROE_SC_stats sum_stats_OSEFX OSEFX...
                ROA_NC_sharpe_test ROA_TC_sharpe_test ROA_SC_sharpe_test...
                ROE_SC_sharpe_test

%% Regression for Outer Western
% 1. PREPERATION OF DATASET
outer_western      = (apartment_set.Bydel == 'Ullern') + (apartment_set.Bydel ==
'Vestre Aker') + (apartment_set.Bydel == 'Nordre Aker');
outer_western      = logical(outer_western);
outer_western_set  = apartment_set(outer_western,:);

% Make categorical for dummy-purposes
outer_western_set.Alder    = categorical(outer_western_set.Alder);
outer_western_set.Q        = categorical(outer_western_set.Q);
outer_western_set.Eierform = categorical(outer_western_set.Eierform);
outer_western_set.Etasje   = categorical(outer_western_set.Etasje);

% Reorder categories/dummies to find reference variables
outer_western_set.Alder    = reordercats(outer_western_set.Alder,    {'Age1', 'Age2',
'Age3', 'Age4'});
outer_western_set.Eierform = reordercats(outer_western_set.Eierform, {'Selveier',
'Borettslag'});
outer_western_set.Etasje   = reordercats(outer_western_set.Etasje,   {'1', '2', '3',
'4', '5', '6', '7', '8', '9', '10', '11', '12', '13', '14', '15', '-1', '-2', '-3', '-
4'});

% Remove unused categories
outer_western_set.Eierform = removecats(outer_western_set.Eierform);

% Check that we in fact removed unused categories
categories(outer_western_set.Alder)
```

```
categories(outer_western_set.Eierform)

% Make log prices and sizes
outer_western_set.log_pris = log(outer_western_set.Pris);
outer_western_set.log_prom = log(outer_western_set.prom);

% 2. Run regression - Log model with log(prom), Q, Bydel, Soverom, Alder on log(pris)
% Explanatory variables log(prom), soverom - continous
% Explanatory variables Q, Bydel, Eierform, Alder and Etasje - dummies
outer_western_model =
fitlm(outer_western_set,'log_pris~log_prom+Q+Eierform+Soverom+Alder+Etasje');

% 3. ALLOCATE TABLE FOR ESTIMATED PRICES (BASED ON MODEL FROM PT. 2) FOR EVERY QUARTER
FOR EACH ID
outer_western_set   = sortrows(outer_western_set,'BoligID','ascend');
[C,ia,ic]           = unique(outer_western_set.BoligID,'first');
price_outer_western = zeros(length(C)*44, width(outer_western_set));
price_outer_western = array2table(price_outer_western);

price_outer_western.Properties.VariableNames = {'BoligID', 'Salgsdato', 'Bydel',
'Pris', 'Fellesgjeld', 'Prisantydning', 'Eierform', 'Byggeaar', 'prom', 'Etasje',
'Soverom', 'Boligtype', 'year', 'Q', 'Alder', 'kvmpris', 'log_pris', 'log_prom' };

Q_rep   = repmat(["Q1"; "Q2"; "Q3";"Q4"],(length(C)*44)/4,1);
x       = (2010:2020)';
n       = 4;
newx    =
[repmat(x(1),n,1);repmat(x(2),n,1);repmat(x(3),n,1);repmat(x(4),n,1);repmat(x(5),n,1);
repmat(x(6),n,1);repmat(x(7),n,1);repmat(x(8),n,1);repmat(x(9),n,1);repmat(x(10),n,1);
repmat(x(11),n,1)];
Y_order = repmat(newx,(length(C)),1);

price_outer_western.Q_order = string(Y_order(:,1)) + Q_rep(:,1);
price_outer_western.Q       = price_outer_western.Q_order;
price_outer_western.Q_order = [];

price_outer_western.BoligID = categorical(price_outer_western.BoligID);
price_outer_western.BoligID = repelem(C, 44);

price_outer_western           = convertvars(price_outer_western,{'BoligID', 'Bydel',
'Eierform', 'Boligtype', 'Etasje', 'Alder', 'Q'},'categorical');
price_outer_western.Salgsdato =
datetime(price_outer_western.Salgsdato,'ConvertFrom','yyyymmdd');

b = 1:length(newx):height(price_outer_western);

% Create set containing only the unique observations of house_set. When
% multiple unique values it stores the first observation
uniq_outer_western_set = outer_western_set(ia,:);

% Allocate variables into price_houses to use for regression model
price_outer_western(:,[3 5 7 9 10 11 15 18]) = repelem(uniq_outer_western_set(:,[3 5 7
9 10 11 15 18]), 44, 1);

% Store relevant variables for regression model in new table
x3 = price_outer_western(:,[18 14 7 11 15 10]);

% GET COLUMN OF ESTIMATED PRICES BASED ON REGRESSION MODEL
price_outer_western.log_estimated_price    = predict(outer_western_model, x3);
price_outer_western.estimated_price        =
exp(price_outer_western.log_estimated_price);
price_outer_western.lagged_estimated_price =
lagmatrix(price_outer_western.estimated_price,1);

% 4. IMPLEMENT RELEVANT COSTS RELATED TO HOUSING INVESTMENTS
% Transaction costs buyer, b
gebyr_b             = zeros(height(price_outer_western),1);
dokavg_b            = zeros(height(price_outer_western),1);
insurance_b         = zeros(height(price_outer_western),1);
price_outer_western.t_cost = zeros(height(price_outer_western),1);

% If type of ownership = 'Selveier'
idx_selv             = price_outer_western.Eierform == "Selveier";
gebyr_b(idx_selv)    = 540 + 540 + 172;
```

```
dokavg_b(idx_selv)    =
0.025.*(price_outer_western.estimated_price(idx_selv)+price_outer_western.Fellesgjeld(
idx_selv));
insurance_b(idx_selv) = 7400;

% If type of ownership = 'Borettslag'
idx_andel             = price_outer_western.Eierform == "Borettslag";
gebyr_b(idx_andel)    = 440 + 440 + 172;
dokavg_b(idx_andel)   = 0;
insurance_b(idx_andel) = 4100;

% Compute total transaction cost for buyer, b, for each observation/quarter
price_outer_western.t_cost        = gebyr_b+dokavg_b+insurance_b;
price_outer_western.lagged_t_cost = lagmatrix(price_outer_western.t_cost,1);

% Transaction costs seller, s
eierskiftegebyr_s     = zeros(height(price_outer_western),1);
public_fees_s         = zeros(height(price_outer_western),1);
insurance_s           = zeros(height(price_outer_western),1);
commission_s          = zeros(height(price_outer_western),1);
price_outer_western.t_cost_s = zeros(height(price_outer_western),1);
indices_selv          = zeros(height(price_outer_western),1);
indices_andel         = zeros(height(price_outer_western),1);

% If type of ownership = 'Selveier'
idx_selv                   = price_outer_western.Eierform == "Selveier";
public_fees_s(idx_selv)    = 8430;
eierskiftegebyr_s(idx_selv) = 0;
insurance_s(idx_selv)      =
max(0.0033*(price_outer_western.estimated_price(idx_selv)+price_outer_western.Fellesgj
eld(idx_selv)), 4000);
indices_selv               = insurance_s > 24000;
indices_selv               = logical(indices_selv);
insurance_s(indices_selv)  = 24000;

% If type of ownership = 'Borettslag'
idx_andel                  = price_outer_western.Eierform == "Borettslag";
public_fees_s(idx_andel)   = 14319;
eierskiftegebyr_s(idx_andel) = 4858;
insurance_s(idx_andel)     =
max(0.002*(price_outer_western.estimated_price(idx_andel)+price_outer_western.Fellesgj
eld(idx_andel)), 2500);
indices_andel              = zeros(height(price_outer_western),1);
indices_andel(idx_andel)   = insurance_s(idx_andel) > 24000;
indices_andel              = logical(indices_andel);
insurance_s(indices_andel) = 24000;

% Commission to broker independent of type of ownership
commission_s =
max(0.02*(price_outer_western.estimated_price+price_outer_western.Fellesgjeld),40000);

% Compute total transaction cost for seller, s, for each observation/quarter
price_outer_western.t_cost_s =
public_fees_s+eierskiftegebyr_s+insurance_s+commission_s;

% Compute transaction cost rates for buyer and seller
% For buyer
t_cost_rate   = mean(price_outer_western.t_cost./price_outer_western.estimated_price);
t_cost_rate_Q = t_cost_rate/4;

% For seller
t_cost_s_rate   =
mean(price_outer_western.t_cost_s./price_outer_western.estimated_price);
t_cost_s_rate_Q = t_cost_s_rate/4;

% Define holding period for each dwelling
holding_period = 11;

% Define living expenses per quarter - reference apartment is 70 sqm
liv_exp_a_70_Q = ([76734/4, 82530/4, 83430/4, 89095/4, 88857/4, 83084/4, 83769/4,
86976/4, 91419/4, 103712/4, 87905/4])';

% Allocate correct liwing expenses to each quarter in price_houses set
n = 4;
```

```
liv_exp_apartments =
[repmat(liv_exp_a_70_Q(1),n,1);repmat(liv_exp_a_70_Q(2),n,1);repmat(liv_exp_a_70_Q(3),
n,1);repmat(liv_exp_a_70_Q(4),n,1);repmat(liv_exp_a_70_Q(5),n,1);repmat(liv_exp_a_70_Q
(6),n,1);repmat(liv_exp_a_70_Q(7),n,1);repmat(liv_exp_a_70_Q(8),n,1);repmat(liv_exp_a_
70_Q(9),n,1);repmat(liv_exp_a_70_Q(10),n,1);repmat(liv_exp_a_70_Q(11),n,1)];
price_outer_western.ordered_liv_exp_70 = repmat(liv_exp_apartments,(length(C)),1);

% Compute living expenses based on a reference apartment of 70 sqm
price_outer_western.living_exp =
(price_outer_western.prom./70).*price_outer_western.ordered_liv_exp_70;
avg_living_exp_Q =
mean(price_outer_western.living_exp./price_outer_western.estimated_price);

% Allocate correct implicit rents
implicit_rent     = zeros(height(price_outer_western),1);
rental_allocation = repmat(Rent_Data, length(C), 1);

implicit_rent(find(price_outer_western.Soverom == 0)) =
rental_allocation.room1(find(price_outer_western.Soverom == 0));
implicit_rent(find(price_outer_western.Soverom == 1)) =
rental_allocation.room1(find(price_outer_western.Soverom == 1));
implicit_rent(find(price_outer_western.Soverom == 2)) =
rental_allocation.room2(find(price_outer_western.Soverom == 2));
implicit_rent(find(price_outer_western.Soverom == 3)) =
rental_allocation.room3(find(price_outer_western.Soverom == 3));
implicit_rent(find(price_outer_western.Soverom == 4)) =
rental_allocation.room4(find(price_outer_western.Soverom == 4));
implicit_rent(find(price_outer_western.Soverom >= 5)) =
rental_allocation.room5_(find(price_outer_western.Soverom >= 5));

price_outer_western.implicit_rent = implicit_rent;
avg_implicit_rent_Q =
mean(price_outer_western.implicit_rent./price_outer_western.estimated_price);

% Implement tax rates for sample period
tax_rates = ([ 0.28, 0.28, 0.28, 0.28, 0.27, 0.27, 0.25, 0.24, 0.23, 0.22, 0.22])';
n = 4;
tax_rates_order =
[repmat(tax_rates(1),n,1);repmat(tax_rates(2),n,1);repmat(tax_rates(3),n,1);repmat(tax
_rates(4),n,1);repmat(tax_rates(5),n,1);repmat(tax_rates(6),n,1);repmat(tax_rates(7),n
,1);repmat(tax_rates(8),n,1);repmat(tax_rates(9),n,1);repmat(tax_rates(10),n,1);repmat
(tax_rates(11),n,1)];

% Allocate correct tax rates to each quarter in price_houses set
price_outer_western.tax_rates = repmat(tax_rates_order,(length(C)),1);

% Create a table to store different cost rates
avg_cost_rates_outer_western                          = zeros(4,3); %JUSTERE DENNE
ETTERSOM HVORDAN ST√òRRELSEN VIL BLI
avg_cost_rates_outer_western                          =
array2table(avg_cost_rates_outer_western);
avg_cost_rates_outer_western.Properties.VariableNames = {'Average_cost_rates',
'Yearly', 'Quarterly'};
avg_cost_rates_outer_western.Average_cost_rates       =
categorical(avg_cost_rates_outer_western.Average_cost_rates);

% Import stats for cost rates
avg_cost_rates_outer_western.Average_cost_rates(1) = 't_cost_buyer';
avg_cost_rates_outer_western.Yearly(1)             = t_cost_rate;
avg_cost_rates_outer_western.Quarterly(1)          = t_cost_rate_Q;

avg_cost_rates_outer_western.Average_cost_rates(2) = 't_cost_seller';
avg_cost_rates_outer_western.Yearly(2)             = t_cost_s_rate;
avg_cost_rates_outer_western.Quarterly(2)          = t_cost_s_rate_Q;

avg_cost_rates_outer_western.Average_cost_rates(3) = 'living_expenses';
avg_cost_rates_outer_western.Yearly(3)             = avg_living_exp_Q * 4;
avg_cost_rates_outer_western.Quarterly(3)          = avg_living_exp_Q;

avg_cost_rates_outer_western.Average_cost_rates(4) = 'implicit_rent';
avg_cost_rates_outer_western.Yearly(4)             = avg_implicit_rent_Q * 4;
avg_cost_rates_outer_western.Quarterly(4)          = avg_implicit_rent_Q;

% 5. ALLOCATE TABLE FOR PRICE INDICES
```

```
priceindex_outer_western          = table();
Y_index                           = Y_order(1:44);
Q_index                           = Q_rep(1:44);
priceindex_outer_western.Q_order  = string(Y_index(:,1)) + Q_index(:,1);
priceindex_outer_western.Q_order  = categorical(priceindex_outer_western.Q_order);

% Create a table to store returns, std, risk free, sharpe
sum_stats_outer_western                          = zeros(10,7);
sum_stats_outer_western                          =
array2table(sum_stats_outer_western);
sum_stats_outer_western.Properties.VariableNames = {'House_Indices', 'Return',
'std_dev','Excess_return','Sharpe_ratio', 'Skewness', 'Kurtosis'};
sum_stats_outer_western.House_Indices            =
categorical(sum_stats_outer_western.House_Indices);

% Import risk-free rates (innskuddsrente)
n = 4;
priceindex_outer_western.rf = zeros(44,1);
priceindex_outer_western.rf =
[repmat(Rf.rf(1),n,1);repmat(Rf.rf(2),n,1);repmat(Rf.rf(3),n,1);repmat(Rf.rf(4),n,1);r
epmat(Rf.rf(5),n,1);repmat(Rf.rf(6),n,1);repmat(Rf.rf(7),n,1);repmat(Rf.rf(8),n,1);rep
mat(Rf.rf(9),n,1);repmat(Rf.rf(10),n,1);repmat(Rf.rf(11),n,1)];
priceindex_outer_western.rf = priceindex_outer_western.rf./4;

% 6. COMPUTATION OF MEAN PRICES FOR HOUSES AND PRICE INDEX BASED ON MEAN PRICES
% Calculate mean prices for houses
priceindex_outer_western.mean_prices = zeros(height(priceindex_outer_western), end);
for i = 1:length(priceindex_outer_western.Q_order)
    priceindex_outer_western.mean_prices(i) =
mean(price_outer_western.estimated_price(find(price_outer_western.Q ==
priceindex_outer_western.Q_order(i))));
end

% Calculate price index for houses based on mean prices
priceindex_outer_western.index_values = zeros(height(priceindex_outer_western), end);
for i = 1:length(priceindex_outer_western.index_values)
    priceindex_outer_western.index_values(i) =
priceindex_outer_western.mean_prices(i)/priceindex_outer_western.mean_prices(1);
end

% 7. COMPUTATION OF QUARTERLY RETURNS ON ASSET/HOUSE FROM PERIOD TO PERIOD, NO COSTS
%     Costs not included

% Compute quarterly returns for each unique ID
price_outer_western.Q_ret_no_costs =
(price_outer_western.estimated_price./price_outer_western.lagged_estimated_price)-1;
price_outer_western.Q_ret_no_costs(1:44:end) = NaN;

% Compute mean quarterly returns for houses (mean of all IDs per quarter) (no costs)
priceindex_outer_western.Q_ret_no_costs = zeros(height(priceindex_outer_western),
end);

for i = 1:height(priceindex_outer_western)
    priceindex_outer_western.Q_ret_no_costs(i) =
nanmean(price_outer_western.Q_ret_no_costs(find(price_outer_western.Q ==
priceindex_outer_western.Q_order(i))));
end

% Compute mean of mean quarterly returns for houses (no costs)
mean_Q_ret_no_costs = nanmean(priceindex_outer_western.Q_ret_no_costs);

% Compute excess quarterly returns for houses (no costs)
priceindex_outer_western.excess_Q_ret_no_costs =
priceindex_outer_western.Q_ret_no_costs-priceindex_outer_western.rf;

% Compute mean excess quarterly returns for houses (no costs)
mean_excess_Q_ret_no_costs = nanmean(priceindex_outer_western.excess_Q_ret_no_costs);

% Compute standard deviation for each unique ID
st_Q_ret_no_costs= zeros(length(C),1);
time(1,:) = clock;
for i = 1:length(C)
    if b(i)<b(length(C))
```

```
        st_Q_ret_no_costs(i) =
std(price_outer_western.Q_ret_no_costs(((b(i)+1):(b(i+1)-1))));
    else
        st_Q_ret_no_costs(i) =
std(price_outer_western.Q_ret_no_costs((b(length(C))+1):height(price_outer_western)));
    end
time(i+1,:) = clock;
fprintf('\nTo complete %.f/%.f iterations (unique IDs) it took %.2f seconds\n', i,
length(C), etime(time(i+1,:), time(1,:)));
end

% Standard deviation of all quarterly returns
std_Q_ret_no_costs = mean(st_Q_ret_no_costs);

% Import stats for 'Q_ret_no_costs' in summary table
sum_stats_outer_western.House_Indices(1) = 'Q_ret_no_costs';
sum_stats_outer_western.Return(1)        = mean_Q_ret_no_costs;
sum_stats_outer_western.std_dev(1)       = std_Q_ret_no_costs;
sum_stats_outer_western.Excess_return(1) = mean_excess_Q_ret_no_costs;
sum_stats_outer_western.Sharpe_ratio(1)  =
sum_stats_outer_western.Excess_return(1)/sum_stats_outer_western.std_dev(1);
sum_stats_outer_western.Skewness(1)      =
skewness(priceindex_outer_western.Q_ret_no_costs(2:44));
sum_stats_outer_western.Kurtosis(1)      =
kurtosis(priceindex_outer_western.Q_ret_no_costs(2:44));

% COMPUTE CUMULATIVE ROA(NC)
first_obs_price = price_outer_western.estimated_price(1:44:end);
price_outer_western.first_obs_price_repeated = repelem(first_obs_price, 44);

price_outer_western.cum_Q_ret_no_costs =
(price_outer_western.estimated_price./price_outer_western.first_obs_price_repeated)-1;
price_outer_western.cum_Q_ret_no_costs(1:44:end) = NaN;

priceindex_outer_western.cum_Q_ret_no_costs =
zeros(height(priceindex_outer_western),end);
% Compute mean cumulative ROA(NC)
for i = 1:height(priceindex_outer_western)
    priceindex_outer_western.cum_Q_ret_no_costs(i) =
nanmean(price_outer_western.cum_Q_ret_no_costs(find(price_outer_western.Q ==
priceindex_outer_western.Q_order(i))));
end

% 8. COMPUTATION OF QUARTERLY RETURNS ON ASSET/HOUSE FROM PERIOD TO PERIOD (BUY/SELL),
INCL TOTAL COSTS
% Compute quarterly returns for each unique ID
price_outer_western.Q_ret_tot_costs = (((price_outer_western.estimated_price -
price_outer_western.t_cost_s - price_outer_western.lagged_t_cost -
price_outer_western.lagged_estimated_price).*(1-price_outer_western.tax_rates)) -
price_outer_western.living_exp +
price_outer_western.implicit_rent)./(price_outer_western.lagged_estimated_price);
price_outer_western.Q_ret_tot_costs(1:44:end) = NaN;

% Compute mean quarterly returns for houses (mean of all IDs per quarter) (inc tot
costs)
priceindex_outer_western.Q_ret_tot_costs = zeros(height(priceindex_outer_western),
end);

for i = 1:length(priceindex_outer_western.mean_prices)
    priceindex_outer_western.Q_ret_tot_costs(i) =
mean(price_outer_western.Q_ret_tot_costs(find(price_outer_western.Q ==
priceindex_outer_western.Q_order(i))));
end

% Compute mean of mean quarterly returns for houses (inc tot costs)
mean_Q_ret_tot_costs = nanmean(priceindex_outer_western.Q_ret_tot_costs);

% Compute excess quarterly returns for houses (inc tot costs)
priceindex_outer_western.excess_Q_ret_tot_costs =
priceindex_outer_western.Q_ret_tot_costs-priceindex_outer_western.rf;

% Compute mean excess quarterly returns for houses (inc tot costs)
mean_excess_Q_ret_tot_costs =
nanmean(priceindex_outer_western.excess_Q_ret_tot_costs);
```

```
% Compute standard deviation for each unique IDs returns
st_Q_ret_tot_costs= zeros(length(C),1);
time(1,:) = clock;
for i = 1:length(C)
    if b(i)<b(length(C))
       st_Q_ret_tot_costs(i) =
std(price_outer_western.Q_ret_tot_costs(((b(i)+1):(b(i+1)-1))));
    else
       st_Q_ret_tot_costs(i) =
std(price_outer_western.Q_ret_tot_costs((b(length(C))+1):height(price_outer_western)))
;
    end
time(i+1,:) = clock;
fprintf('\nTo complete %.f/%.f iterations (unique IDs) it took %.2f seconds\n', i,
length(C), etime(time(i+1,:), time(1,:)));
end

% Standard deviation of all quarterly returns
std_Q_ret_tot_costs = mean(st_Q_ret_tot_costs);

% Import stats for 'Q_ret_tot_costs' in summary table
sum_stats_outer_western.House_Indices(2) = 'Q_ret_tot_costs';
sum_stats_outer_western.Return(2)        = mean_Q_ret_tot_costs;
sum_stats_outer_western.std_dev(2)       = std_Q_ret_tot_costs;
sum_stats_outer_western.Excess_return(2) = mean_excess_Q_ret_tot_costs;
sum_stats_outer_western.Sharpe_ratio(2) =
sum_stats_outer_western.Excess_return(2)/sum_stats_outer_western.std_dev(2);
sum_stats_outer_western.Skewness(2)      =
skewness(priceindex_outer_western.Q_ret_tot_costs(2:44));
sum_stats_outer_western.Kurtosis(2)      =
kurtosis(priceindex_outer_western.Q_ret_tot_costs(2:44));

% COMPUTE CUMULATIVE ROA (TC)
taxes_TC = (price_outer_western.estimated_price - price_outer_western.t_cost_s -
price_outer_western.lagged_t_cost -
price_outer_western.lagged_estimated_price).*price_outer_western.tax_rates;
taxes_TC(1:44:end) = NaN;

net_costs_TC = (price_outer_western.living_exp + price_outer_western.lagged_t_cost +
price_outer_western.t_cost_s + taxes_TC - price_outer_western.implicit_rent);
net_costs_TC(1:44:end) = NaN;

% Compute cumulative costs
cumcosts_TC = zeros(height(price_outer_western), end);
time(1,:) = clock;
for i = 1:length(C)
    if b(i)<b(length(C))
       cumcosts_TC((b(i)):(b(i+1)-1)) = cumsum(net_costs_TC((b(i)):(b(i+1)-
1)),'omitnan');
    else
       cumcosts_TC(b(length(C)):height(price_outer_western)) =
cumsum(net_costs_TC(b(length(C)):height(price_outer_western)),'omitnan');
    end
time(i+1,:) = clock;
fprintf('\nTo complete %.f/%.f iterations (unique IDs) it took %.2f seconds\n', i,
length(C), etime(time(i+1,:), time(1,:)));
end

price_outer_western.cum_Q_ret_tot_costs = (price_outer_western.estimated_price -
cumcosts_TC -
price_outer_western.first_obs_price_repeated)./price_outer_western.first_obs_price_rep
eated;
price_outer_western.cum_Q_ret_tot_costs(1:44:end) = NaN;

priceindex_outer_western.cum_Q_ret_tot_costs =
zeros(height(priceindex_outer_western),end);
% Compute mean cumulative returns for houses (mean of all IDs per quarter) (no costs)
for i = 1:height(priceindex_outer_western)
    priceindex_outer_western.cum_Q_ret_tot_costs(i) =
nanmean(price_outer_western.cum_Q_ret_tot_costs(find(price_outer_western.Q ==
priceindex_outer_western.Q_order(i))));
end
```

```
all_lost = priceindex_outer_western.cum_Q_ret_tot_costs < -1;
priceindex_outer_western.cum_Q_ret_tot_costs(all_lost) = -1;

% 9. COMPUTATION OF QUARTERLY RETURNS ON ASSET/HOUSE FROM PERIOD TO PERIOD , INCL
SMOOTHED COSTS
% Locate buyer and sellers cost when buying in 2010Q1 and selling in 2020Q4
% Find buyer and seller cost per quarter for sample period
cost_2010Q1 = price_outer_western.t_cost(1:44:end);
cost_2010Q1 = cost_2010Q1/44;
price_outer_western.cost_2010Q1 = repelem(cost_2010Q1, 44);

cost_2020Q4 = price_outer_western.t_cost_s(44:44:end);
cost_2020Q4 = cost_2020Q4/44;
price_outer_western.cost_2020Q4 = repelem(cost_2020Q4, 44);

% Compute quarterly returns given that you buy in 2010Q1 and sell in 2020Q4
% and spred the relevant costs on the sample period.
price_outer_western.Q_ret_smooth_costs = (price_outer_western.estimated_price -
price_outer_western.cost_2010Q1 - price_outer_western.cost_2020Q4 -
price_outer_western.living_exp - price_outer_western.lagged_estimated_price +
price_outer_western.implicit_rent)./price_outer_western.lagged_estimated_price;
price_outer_western.Q_ret_smooth_costs(1:44:end) = NaN;

% Compute mean quarterly returns for houses (mean of all IDs per quarter) (inc smooth
costs)
priceindex_outer_western.Q_ret_smooth_costs = zeros(height(priceindex_outer_western),
end);

for i = 1:length(priceindex_outer_western.mean_prices)
    priceindex_outer_western.Q_ret_smooth_costs(i) =
mean(price_outer_western.Q_ret_smooth_costs(find(price_outer_western.Q ==
priceindex_outer_western.Q_order(i))));
end

% Compute mean of mean quarterly returns for houses (inc smooth costs)
mean_Q_ret_smooth_costs = nanmean(priceindex_outer_western.Q_ret_smooth_costs);

% Compute excess quarterly returns for houses (inc smooth costs)
priceindex_outer_western.excess_Q_ret_smooth_costs =
priceindex_outer_western.Q_ret_smooth_costs - priceindex_outer_western.rf;

% Compute mean excess quarterly returns for houses (inc smooth costs)
mean_excess_Q_ret_smooth_costs =
nanmean(priceindex_outer_western.excess_Q_ret_smooth_costs);

% Compute standard deviation for each unique IDs returns
st_Q_ret_smooth_costs= zeros(length(C),1);
time(1,:) = clock;
for i = 1:length(C)
    if b(i)<b(length(C))
       st_Q_ret_smooth_costs(i) =
std(price_outer_western.Q_ret_smooth_costs(((b(i)+1):(b(i+1)-1))));
    else
       st_Q_ret_smooth_costs(i) =
std(price_outer_western.Q_ret_smooth_costs((b(length(C))+1):height(price_outer_western
)));
    end
time(i+1,:) = clock;
fprintf('\nTo complete %.f/%.f iterations (unique IDs) it took %.2f seconds\n', i,
length(C), etime(time(i+1,:), time(1,:)));
end

% Standard deviation of all quarterly returns
std_Q_ret_smooth_costs = mean(st_Q_ret_smooth_costs);

% Import stats for 'Q_ret_smooth_costs' in summary table
sum_stats_outer_western.House_Indices(3) = 'Q_ret_smooth_costs';
sum_stats_outer_western.Return(3)        = mean_Q_ret_smooth_costs;
sum_stats_outer_western.std_dev(3)       = std_Q_ret_smooth_costs;
sum_stats_outer_western.Excess_return(3) = mean_excess_Q_ret_smooth_costs;
sum_stats_outer_western.Sharpe_ratio(3) =
sum_stats_outer_western.Excess_return(3)/sum_stats_outer_western.std_dev(3);
sum_stats_outer_western.Skewness(3)      =
skewness(priceindex_outer_western.Q_ret_smooth_costs(2:44));
```

```
sum_stats_outer_western.Kurtosis(3)      =
kurtosis(priceindex_outer_western.Q_ret_smooth_costs(2:44));

% COMPUTE CUMULATIVE ROA(SC)
net_costs_SC = (price_outer_western.living_exp + price_outer_western.cost_2010Q1 +
price_outer_western.cost_2020Q4 - price_outer_western.implicit_rent);
net_costs_SC(1:44:end) = NaN;

cumcosts_SC = zeros(height(price_outer_western), end);
time(1,:) = clock;
for i = 1:length(C)
    if b(i)<b(length(C))
       cumcosts_SC((b(i)):(b(i+1)-1)) = cumsum(net_costs_SC((b(i)):(b(i+1)-
1)),'omitnan');
    else
       cumcosts_SC(b(length(C)):height(price_outer_western)) =
cumsum(net_costs_SC(b(length(C)):height(price_outer_western)),'omitnan');
    end
time(i+1,:) = clock;
fprintf('\nTo complete %.f/%.f iterations (unique IDs) it took %.2f seconds\n', i,
length(C), etime(time(i+1,:), time(1,:)));
end

price_outer_western.cum_Q_ret_smooth_costs = (price_outer_western.estimated_price -
cumcosts_SC -
price_outer_western.first_obs_price_repeated)./price_outer_western.first_obs_price_rep
eated;
price_outer_western.cum_Q_ret_smooth_costs(1:44:end) = NaN;

% Compute mean cumulative returns for houses (mean of all IDs per quarter) (no costs)
priceindex_outer_western.cum_Q_ret_smooth_costs =
zeros(height(priceindex_outer_western),end);
for i = 1:height(priceindex_outer_western)
    priceindex_outer_western.cum_Q_ret_smooth_costs(i) =
nanmean(price_outer_western.cum_Q_ret_smooth_costs(find(price_outer_western.Q ==
priceindex_outer_western.Q_order(i))));
end

% Identify average loan-to-value ratio for housing investments
l = ([0.6957, 0.6957, 0.6957, 0.6957, 0.71, 0.76, 0.67, 0.66, 0.68, 0.70, 0.69])';
n = 4;
ltv_ratio_order =
[repmat(l(1),n,1);repmat(l(2),n,1);repmat(l(3),n,1);repmat(l(4),n,1);repmat(l(5),n,1);
repmat(l(6),n,1);repmat(l(7),n,1);repmat(l(8),n,1);repmat(l(9),n,1);repmat(l(10),n,1);
repmat(l(11),n,1)];

% Allocate correct ltv-rates to each quarter in price_houses set
price_outer_western.ltvr        = repmat(ltv_ratio_order,(length(C)),1);
price_outer_western.lagged_ltvr = lagmatrix(price_outer_western.ltvr,1);

% Allocate ltv ratio as of 2010Q1 to each quarter in price_houses set
price_outer_western.ltvr_2010Q1(:) = price_outer_western.ltvr(1);

% 10. WE DONT USE THE RETURN SERIES COMPUTED IN SECTION 10
% Compute quarterly equity returns for each unique ID (no costs)
price_outer_western.Q_eq_ret_no_costs = ((price_outer_western.estimated_price -
price_outer_western.lagged_estimated_price)./(price_outer_western.lagged_estimated_pri
ce.*(1-price_outer_western.lagged_ltvr)));
price_outer_western.Q_eq_ret_no_costs(1:44:end) = NaN;

% Compute mean quarterly equity returns for houses (mean of all IDs per quarter) (no
costs)
priceindex_outer_western.Q_eq_ret_no_costs = zeros(height(priceindex_outer_western),
end);

for i = 1:height(priceindex_outer_western)
    priceindex_outer_western.Q_eq_ret_no_costs(i) =
mean(price_outer_western.Q_eq_ret_no_costs(find(price_outer_western.Q ==
priceindex_outer_western.Q_order(i))));
end

% Compute mean of mean quarterly equity returns for houses (no costs)
mean_Q_eq_ret_no_costs = nanmean(priceindex_outer_western.Q_eq_ret_no_costs);
```

```
% Compute excess quarterly equity returns for houses (no costs)
priceindex_outer_western.excess_Q_eq_ret_no_costs =
priceindex_outer_western.Q_eq_ret_no_costs - priceindex_outer_western.rf;

% Compute mean excess quarterly equity returns for houses (no costs)
mean_excess_Q_eq_ret_no_costs =
nanmean(priceindex_outer_western.excess_Q_eq_ret_no_costs);

% Compute standard deviation for each unique IDs returns
st_Q_eq_ret_no_costs= zeros(length(C),1);
time(1,:) = clock;
for i = 1:length(C)
    if b(i)<b(length(C))
        st_Q_eq_ret_no_costs(i) =
std(price_outer_western.Q_eq_ret_no_costs(((b(i)+1):(b(i+1)-1))));
    else
        st_Q_eq_ret_no_costs(i) =
std(price_outer_western.Q_eq_ret_no_costs((b(length(C))+1):height(price_outer_western)
));
    end
time(i+1,:) = clock;
fprintf('\nTo complete %.f/%.f iterations (unique IDs) it took %.2f seconds\n', i,
length(C), etime(time(i+1,:), time(1,:)));
end

% Standard deviation of all quarterly equity returns
std_Q_eq_ret_no_costs = mean(st_Q_eq_ret_no_costs);

% Import stats for 'Q_eq_ret_no_costs' in summary table
sum_stats_outer_western.House_Indices(4) = 'Q_eq_ret_no_costs';
sum_stats_outer_western.Return(4)        = mean_Q_eq_ret_no_costs;
sum_stats_outer_western.std_dev(4)       = std_Q_eq_ret_no_costs;
sum_stats_outer_western.Excess_return(4) = mean_excess_Q_eq_ret_no_costs;
sum_stats_outer_western.Sharpe_ratio(4) =
sum_stats_outer_western.Excess_return(4)/sum_stats_outer_western.std_dev(4);
sum_stats_outer_western.Skewness(4)      =
skewness(priceindex_outer_western.Q_eq_ret_no_costs(2:44));
sum_stats_outer_western.Kurtosis(4)      =
kurtosis(priceindex_outer_western.Q_eq_ret_no_costs(2:44));

% COMPUTE CUMULATIVE ROE (NC)
price_outer_western.cum_Q_eq_ret_no_costs = ((price_outer_western.estimated_price -
price_outer_western.first_obs_price_repeated)./(price_outer_western.first_obs_price_re
peated.*(1-price_outer_western.ltvr_2010Q1)));
price_outer_western.cum_Q_eq_ret_no_costs(1:44:end) = NaN;

% Compute mean cumulative returns for houses (mean of all IDs per quarter) (no costs)
priceindex_outer_western.cum_Q_eq_ret_no_costs =
zeros(height(priceindex_outer_western),end);
for i = 1:height(priceindex_outer_western)
    priceindex_outer_western.cum_Q_eq_ret_no_costs(i) =
nanmean(price_outer_western.cum_Q_eq_ret_no_costs(find(price_outer_western.Q ==
priceindex_outer_western.Q_order(i))));
end

% 11. WE DONT USE THE RETURN SERIES COMPUTED IN SECTION 11
% Compute quarterly equity returns for each unique ID (inc tot costs)
price_outer_western.Q_eq_ret_tot_costs = (((price_outer_western.estimated_price -
price_outer_western.t_cost_s - price_outer_western.lagged_t_cost -
price_outer_western.lagged_estimated_price).*(1-price_outer_western.tax_rates)) -
price_outer_western.living_exp +
price_outer_western.implicit_rent)./(price_outer_western.lagged_estimated_price.*(1-
price_outer_western.lagged_ltvr));
price_outer_western.Q_eq_ret_tot_costs(1:44:end) = NaN;

% Compute mean quarterly equity returns for houses (mean of all IDs per quarter) (inc
tot costs)
priceindex_outer_western.Q_eq_ret_tot_costs = zeros(height(priceindex_outer_western),
end);

for i = 1:length(priceindex_outer_western.mean_prices)
    priceindex_outer_western.Q_eq_ret_tot_costs(i) =
mean(price_outer_western.Q_eq_ret_tot_costs(find(price_outer_western.Q ==
priceindex_outer_western.Q_order(i))));
```

```
end

% Compute mean of mean quarterly equity returns for houses (inc tot costs)
mean_Q_eq_ret_tot_costs = nanmean(priceindex_outer_western.Q_eq_ret_tot_costs);

% Compute excess quarterly equity returns for houses (inc tot costs)
priceindex_outer_western.excess_Q_eq_ret_tot_costs =
priceindex_outer_western.Q_eq_ret_tot_costs - priceindex_outer_western.rf;

% Compute mean excess quarterly equity returns for houses (inc tot costs)
mean_excess_Q_eq_ret_tot_costs =
nanmean(priceindex_outer_western.excess_Q_eq_ret_tot_costs);

% Compute standard deviation for each unique IDs returns
st_Q_eq_ret_tot_costs= zeros(length(C),1);
time(1,:) = clock;
for i = 1:length(C)
    if b(i)<b(length(C))
       st_Q_eq_ret_tot_costs(i) =
std(price_outer_western.Q_eq_ret_tot_costs(((b(i)+1):(b(i+1)-1))));
    else
       st_Q_eq_ret_tot_costs(i) =
std(price_outer_western.Q_eq_ret_tot_costs((b(length(C))+1):height(price_outer_western
)));
    end
time(i+1,:) = clock;
fprintf('\nTo complete %.f/%.f iterations (unique IDs) it took %.2f seconds\n', i,
length(C), etime(time(i+1,:), time(1,:)));
end

% Standard deviation of all quarterly equity returns
std_Q_eq_ret_tot_costs = mean(st_Q_eq_ret_tot_costs);

% Import stats for 'Q_eq_ret_tot_costs' in summary table
sum_stats_outer_western.House_Indices(5) = 'Q_eq_ret_tot_costs';
sum_stats_outer_western.Return(5)        = mean_Q_eq_ret_tot_costs;
sum_stats_outer_western.std_dev(5)       = std_Q_eq_ret_tot_costs;
sum_stats_outer_western.Excess_return(5) = mean_excess_Q_eq_ret_tot_costs;
sum_stats_outer_western.Sharpe_ratio(5) =
sum_stats_outer_western.Excess_return(5)/sum_stats_outer_western.std_dev(5);
sum_stats_outer_western.Skewness(5)      =
skewness(priceindex_outer_western.Q_eq_ret_tot_costs(2:44));
sum_stats_outer_western.Kurtosis(5)      =
kurtosis(priceindex_outer_western.Q_eq_ret_tot_costs(2:44));

% COMPUTE CUMULATIVE RETURNS BASED ON QUARTERLY EQUITY RETURNS FROM PRICE INDEX (INC
TOT COSTS)
priceindex_outer_western.cum_Q_eq_ret_tot_costs =
zeros(height(priceindex_outer_western),end);
priceindex_outer_western.cum_Q_eq_ret_tot_costs =
cumprod((1+priceindex_outer_western.Q_eq_ret_tot_costs),'omitnan') -1;

% 12. COMPUTATION OF QUARTERLY EQUITY RETURNS ON ASSET/HOUSE FROM PERIOD TO PERIOD ,
INCL SMOOTHED COSTS
% Compute quarterly EQUITY returns given that you buy in 2010Q1 and sell in 2020Q4
% and spread the relevant costs on the sample period.
price_outer_western.Q_eq_ret_smooth_costs = (price_outer_western.estimated_price -
price_outer_western.cost_2010Q1 - price_outer_western.cost_2020Q4 -
price_outer_western.living_exp - price_outer_western.lagged_estimated_price +
price_outer_western.implicit_rent)./(price_outer_western.lagged_estimated_price.*(1-
price_outer_western.ltvr_2010Q1));
price_outer_western.Q_eq_ret_smooth_costs(1:44:end) = NaN;

% Compute mean quarterly equity returns for houses (mean of all IDs per quarter) (inc
smooth costs)
priceindex_outer_western.Q_eq_ret_smooth_costs =
zeros(height(priceindex_outer_western), end);

for i = 1:length(priceindex_outer_western.mean_prices)
    priceindex_outer_western.Q_eq_ret_smooth_costs(i) =
mean(price_outer_western.Q_eq_ret_smooth_costs(find(price_outer_western.Q ==
priceindex_outer_western.Q_order(i))));
end
```

```
% Compute mean of mean quarterly equity returns for houses (inc smooth costs)
mean_Q_eq_ret_smooth_costs = nanmean(priceindex_outer_western.Q_eq_ret_smooth_costs);

% Compute excess quarterly equity returns for houses (inc smooth costs)
priceindex_outer_western.excess_Q_eq_ret_smooth_costs =
priceindex_outer_western.Q_eq_ret_smooth_costs - priceindex_outer_western.rf;

% Compute mean excess quarterly equity returns for houses (inc smooth costs)
mean_excess_Q_eq_ret_smooth_costs =
nanmean(priceindex_outer_western.excess_Q_eq_ret_smooth_costs);

% Compute standard deviation for each unique IDs returns
st_Q_eq_ret_smooth_costs= zeros(length(C),1);
time(1,:) = clock;
for i = 1:length(C)
    if b(i)<b(length(C))
       st_Q_eq_ret_smooth_costs(i) =
std(price_outer_western.Q_eq_ret_smooth_costs(((b(i)+1):(b(i+1)-1))));
    else
       st_Q_eq_ret_smooth_costs(i) =
std(price_outer_western.Q_eq_ret_smooth_costs((b(length(C))+1):height(price_outer_west
ern)));
    end
time(i+1,:) = clock;
fprintf('\nTo complete %.f/%.f iterations (unique IDs) it took %.2f seconds\n', i,
length(C), etime(time(i+1,:), time(1,:)));
end

% Standard deviation of all quarterly equity returns
std_Q_eq_ret_smooth_costs = mean(st_Q_eq_ret_smooth_costs);

% Import stats for 'Q_eq_ret_smooth_costs' in summary table
sum_stats_outer_western.House_Indices(6) = 'Q_eq_ret_smooth_costs';
sum_stats_outer_western.Return(6)        = mean_Q_eq_ret_smooth_costs;
sum_stats_outer_western.std_dev(6)       = std_Q_eq_ret_smooth_costs;
sum_stats_outer_western.Excess_return(6) = mean_excess_Q_eq_ret_smooth_costs;
sum_stats_outer_western.Sharpe_ratio(6) =
sum_stats_outer_western.Excess_return(6)/sum_stats_outer_western.std_dev(6);
sum_stats_outer_western.Skewness(6)      =
skewness(priceindex_outer_western.Q_eq_ret_smooth_costs(2:44));
sum_stats_outer_western.Kurtosis(6)      =
kurtosis(priceindex_outer_western.Q_eq_ret_smooth_costs(2:44));

% COMPUTE CUMULATIVE ROE(SC)
price_outer_western.cum_Q_eq_ret_smooth_costs = (price_outer_western.estimated_price -
cumcosts_SC -
price_outer_western.first_obs_price_repeated)./(price_outer_western.first_obs_price_re
peated.*(1-price_outer_western.ltvr_2010Q1));
price_outer_western.cum_Q_eq_ret_smooth_costs(1:44:end) = NaN;

% Compute mean cumulative returns for houses (mean of all IDs per quarter) (smooth
costs)
priceindex_outer_western.cum_Q_eq_ret_smooth_costs =
zeros(height(priceindex_outer_western),end);
for i = 1:height(priceindex_outer_western)
    priceindex_outer_western.cum_Q_eq_ret_smooth_costs(i) =
nanmean(price_outer_western.cum_Q_eq_ret_smooth_costs(find(price_outer_western.Q ==
priceindex_outer_western.Q_order(i))));
end

% MAKE PLOT OF THE RELEVANT INDICES TOGHETER, MAKE MULTIPLE PLOTS FOR COMPARISON
REASONS

% Remember to adjust according to model
outer_western_price_changes = priceindex_outer_western(:,[1 5 8 11 14 17 20]);

outer_western_price_indices = priceindex_outer_western(:,[1 7 10 13 16 19 22]);
outer_western_price_indices.cum_Q_ret_no_costs(1) = 0;
outer_western_price_indices.cum_Q_ret_tot_costs(1) = 0;
outer_western_price_indices.cum_Q_ret_smooth_costs(1) = 0;
outer_western_price_indices.cum_Q_eq_ret_no_costs(1) = 0;
outer_western_price_indices.cum_Q_eq_ret_tot_costs(1) = 0;
outer_western_price_indices.cum_Q_eq_ret_smooth_costs(1) = 0;
```

```
outer_western_price_indices.cum_Q_ret_no_costs        =
outer_western_price_indices.cum_Q_ret_no_costs + 1;
outer_western_price_indices.cum_Q_ret_tot_costs       =
outer_western_price_indices.cum_Q_ret_tot_costs + 1;
outer_western_price_indices.cum_Q_ret_smooth_costs    =
outer_western_price_indices.cum_Q_ret_smooth_costs + 1;
outer_western_price_indices.cum_Q_eq_ret_no_costs     =
outer_western_price_indices.cum_Q_eq_ret_no_costs + 1;
outer_western_price_indices.cum_Q_eq_ret_tot_costs    =
outer_western_price_indices.cum_Q_eq_ret_tot_costs + 1;
outer_western_price_indices.cum_Q_eq_ret_smooth_costs =
outer_western_price_indices.cum_Q_eq_ret_smooth_costs + 1;

clearvars -except Property_Data summary_table Rf Rent_Data...
                  house_set hedonic_houses housing_model...
                  price_houses uniq_house_set avg_cost_rates_houses...
                  priceindex_houses sum_stats_houses houseprice_changes...
                  houseprice_indices...
                  smallhouse_set hedonic_smallhouses smallhouse_model...
                  price_smallhouses uniq_smallhouse_set avg_cost_rates_smallhouses...
                  priceindex_smallhouses sum_stats_smallhouses...
                  smallhouse_price_changes smallhouse_price_indices...
                  apartment_set hedonic_apartments apartment_model price_apartments...
                  uniq_apartment_set avg_cost_rates_apartments
priceindex_apartments...
                  sum_stats_apartments apartment_price_changes...
                  apartment_price_indices...
                  inner_eastern_set hedonic_inner_eastern inner_eastern_model...
                  price_inner_eastern uniq_inner_eastern_set
avg_cost_rates_inner_eastern...
                  priceindex_inner_eastern sum_stats_inner_eastern
inner_eastern_price_changes...
                  inner_eastern_price_indices...
                  inner_western_set hedonic_inner_western inner_western_model...
                  price_inner_western uniq_inner_western_set
avg_cost_rates_inner_western...
                  priceindex_inner_western sum_stats_inner_western
inner_western_price_changes...
                  inner_western_price_indices...
                  outer_eastern_set hedonic_outer_eastern outer_eastern_model...
                  price_outer_eastern uniq_outer_eastern_set
avg_cost_rates_outer_eastern...
                  priceindex_outer_eastern sum_stats_outer_eastern
outer_eastern_price_changes...
                  outer_eastern_price_indices...
                  outer_western_set hedonic_outer_western outer_western_model...
                  price_outer_western uniq_outer_western_set
avg_cost_rates_outer_western...
                  priceindex_outer_western sum_stats_outer_western
outer_western_price_changes...
                  outer_western_price_indices...
                  outer_southern_set hedonic_outer_southern outer_southern_model...
                  price_outer_southern uniq_outer_southern_set
avg_cost_rates_outer_southern...
                  priceindex_outer_southern sum_stats_outer_southern
outer_southern_price_changes...
                  outer_southern_price_indices...
                  property_price_changes property_price_indices hedonic_total...
                  ROA_NC_stats ROE_NC_stats ROA_TC_stats ROE_TC_stats...
                  ROA_SC_stats ROE_SC_stats sum_stats_OSEFX OSEFX...
                  ROA_NC_sharpe_test ROA_TC_sharpe_test ROA_SC_sharpe_test...
                  ROE_SC_sharpe_test

%% Regression for Outer Eastern
% 1. PREPERATION OF DATASET
outer_eastern      = (apartment_set.Bydel == 'Bjerke') + (apartment_set.Bydel ==
'Grorud') + (apartment_set.Bydel == 'Stovner') + (apartment_set.Bydel == 'Alna');
outer_eastern      = logical(outer_eastern);
outer_eastern_set  = apartment_set(outer_eastern,:);

% Make categorical for dummy-purposes
outer_eastern_set.Alder    = categorical(outer_eastern_set.Alder);
outer_eastern_set.Q        = categorical(outer_eastern_set.Q);
outer_eastern_set.Eierform = categorical(outer_eastern_set.Eierform);
```

```
outer_eastern_set.Etasje   = categorical(outer_eastern_set.Etasje);

% Reorder categories/dummies to find reference variables
outer_eastern_set.Alder    = reordercats(outer_eastern_set.Alder,    {'Age1', 'Age2',
'Age3', 'Age4'});
outer_eastern_set.Eierform = reordercats(outer_eastern_set.Eierform, {'Selveier',
'Borettslag'});
outer_eastern_set.Etasje   = reordercats(outer_eastern_set.Etasje,   {'1', '2', '3',
'4', '5', '6', '7', '8', '9', '10', '11', '12', '13', '14', '15', '-1', '-2', '-3', '-
4'});

% Remove unused categories
outer_eastern_set.Eierform = removecats(outer_eastern_set.Eierform);

% Check that we in fact removed unused categories
categories(outer_eastern_set.Alder)
categories(outer_eastern_set.Eierform)

% Make log prices and sizes
outer_eastern_set.log_pris = log(outer_eastern_set.Pris);
outer_eastern_set.log_prom = log(outer_eastern_set.prom);

% 2. Run regression - Log model with log(prom), Q, Bydel, Soverom, Alder on log(pris)
% Explanatory variables log(prom), soverom - continous
% Explanatory variables Q, Bydel, Eierform, Alder and Etasje - dummies
outer_eastern_model =
fitlm(outer_eastern_set,'log_pris~log_prom+Q+Eierform+Soverom+Alder+Etasje');

% 3. ALLOCATE TABLE FOR ESTIMATED PRICES (BASED ON MODEL FROM PT. 2) FOR EVERY QUARTER
FOR EACH ID
outer_eastern_set   = sortrows(outer_eastern_set,'BoligID','ascend');
[C,ia,ic]        = unique(outer_eastern_set.BoligID,'first');
price_outer_eastern = zeros(length(C)*44, width(outer_eastern_set));
price_outer_eastern = array2table(price_outer_eastern);

price_outer_eastern.Properties.VariableNames = {'BoligID', 'Salgsdato', 'Bydel',
'Pris', 'Fellesgjeld', 'Prisantydning', 'Eierform', 'Byggeaar', 'prom', 'Etasje',
'Soverom', 'Boligtype', 'year', 'Q', 'Alder', 'kvmpris', 'log_pris', 'log_prom' };

Q_rep   = repmat(["Q1"; "Q2"; "Q3";"Q4"],(length(C)*44)/4,1);
x       = (2010:2020)';
n       = 4;
newx    =
[repmat(x(1),n,1);repmat(x(2),n,1);repmat(x(3),n,1);repmat(x(4),n,1);repmat(x(5),n,1);
repmat(x(6),n,1);repmat(x(7),n,1);repmat(x(8),n,1);repmat(x(9),n,1);repmat(x(10),n,1);
repmat(x(11),n,1)];
Y_order = repmat(newx,(length(C)),1);

price_outer_eastern.Q_order = string(Y_order(:,1)) + Q_rep(:,1);
price_outer_eastern.Q       = price_outer_eastern.Q_order;
price_outer_eastern.Q_order = [];

price_outer_eastern.BoligID = categorical(price_outer_eastern.BoligID);
price_outer_eastern.BoligID = repelem(C, 44);

price_outer_eastern           = convertvars(price_outer_eastern,{'BoligID', 'Bydel',
'Eierform', 'Boligtype', 'Etasje', 'Alder', 'Q'},'categorical');
price_outer_eastern.Salgsdato =
datetime(price_outer_eastern.Salgsdato,'ConvertFrom','yyyymmdd');

b = 1:length(newx):height(price_outer_eastern);

% Create set containing only the unique observations of house_set. When
% multiple unique values it stores the first observation
uniq_outer_eastern_set = outer_eastern_set(ia,:);

% Allocate variables into price_houses to use for regression model
price_outer_eastern(:,[3 5 7 9 10 11 15 18]) = repelem(uniq_outer_eastern_set(:,[3 5 7
9 10 11 15 18]), 44, 1);

% Store relevant variables for regression model in new table
x3 = price_outer_eastern(:,[18 14 7 11 15 10]);

% GET COLUMN OF ESTIMATED PRICES BASED ON REGRESSION MODEL
```

```
price_outer_eastern.log_estimated_price    = predict(outer_eastern_model, x3);
price_outer_eastern.estimated_price        =
exp(price_outer_eastern.log_estimated_price);
price_outer_eastern.lagged_estimated_price =
lagmatrix(price_outer_eastern.estimated_price,1);

% 4. IMPLEMENT RELEVANT COSTS RELATED TO HOUSING INVESTMENTS
% Transaction costs buyer, b
gebyr_b             = zeros(height(price_outer_eastern),1);
dokavg_b            = zeros(height(price_outer_eastern),1);
insurance_b         = zeros(height(price_outer_eastern),1);
price_outer_eastern.t_cost = zeros(height(price_outer_eastern),1);

% If type of ownership = 'Selveier'
idx_selv              = price_outer_eastern.Eierform == "Selveier";
gebyr_b(idx_selv)     = 540 + 540 + 172;
dokavg_b(idx_selv)    =
0.025.*(price_outer_eastern.estimated_price(idx_selv)+price_outer_eastern.Fellesgjeld(
idx_selv));
insurance_b(idx_selv) = 7400;

% If type of ownership = 'Borettslag'
idx_andel             = price_outer_eastern.Eierform == "Borettslag";
gebyr_b(idx_andel)    = 440 + 440 + 172;
dokavg_b(idx_andel)   = 0;
insurance_b(idx_andel) = 4100;

% Compute total transaction cost for buyer, b, for each observation/quarter
price_outer_eastern.t_cost        = gebyr_b+dokavg_b+insurance_b;
price_outer_eastern.lagged_t_cost = lagmatrix(price_outer_eastern.t_cost,1);

% Transaction costs seller, s
eierskiftegebyr_s     = zeros(height(price_outer_eastern),1);
public_fees_s         = zeros(height(price_outer_eastern),1);
insurance_s           = zeros(height(price_outer_eastern),1);
commission_s          = zeros(height(price_outer_eastern),1);
price_outer_eastern.t_cost_s = zeros(height(price_outer_eastern),1);
indices_selv          = zeros(height(price_outer_eastern),1);
indices_andel         = zeros(height(price_outer_eastern),1);

% If type of ownership = 'Selveier'
idx_selv                    = price_outer_eastern.Eierform == "Selveier";
public_fees_s(idx_selv)     = 8430;
eierskiftegebyr_s(idx_selv) = 0;
insurance_s(idx_selv)       =
max(0.0033*(price_outer_eastern.estimated_price(idx_selv)+price_outer_eastern.Fellesgj
eld(idx_selv)), 4000);
indices_selv                = insurance_s > 24000;
indices_selv                = logical(indices_selv);
insurance_s(indices_selv)   = 24000;

% If type of ownership = 'Borettslag'
idx_andel                    = price_outer_eastern.Eierform == "Borettslag";
public_fees_s(idx_andel)     = 14319;
eierskiftegebyr_s(idx_andel) = 4858;
insurance_s(idx_andel)       =
max(0.002*(price_outer_eastern.estimated_price(idx_andel)+price_outer_eastern.Fellesgj
eld(idx_andel)), 2500);
indices_andel                = zeros(height(price_outer_eastern),1);
indices_andel(idx_andel)     = insurance_s(idx_andel) > 24000;
indices_andel                = logical(indices_andel);
insurance_s(indices_andel)   = 24000;

% Commission to broker independent of type of ownership
commission_s =
max(0.02*(price_outer_eastern.estimated_price+price_outer_eastern.Fellesgjeld),40000);

% Compute total transaction cost for seller, s, for each observation/quarter
price_outer_eastern.t_cost_s =
public_fees_s+eierskiftegebyr_s+insurance_s+commission_s;

% Compute transaction cost rates for buyer and seller
% For buyer
t_cost_rate   = mean(price_outer_eastern.t_cost./price_outer_eastern.estimated_price);
```

```
t_cost_rate_Q = t_cost_rate/4;

% For seller
t_cost_s_rate   =
mean(price_outer_eastern.t_cost_s./price_outer_eastern.estimated_price);
t_cost_s_rate_Q = t_cost_s_rate/4;

% Define holding period for each dwelling
holding_period = 11;

% Define living expenses per quarter - reference apartment is 70 sqm
liv_exp_a_70_Q = ([76734/4, 82530/4, 83430/4, 89095/4, 88857/4, 83084/4, 83769/4,
86976/4, 91419/4, 103712/4, 87905/4])';

% Allocate correct liwing expenses to each quarter in price_houses set
n = 4;
liv_exp_apartments =
[repmat(liv_exp_a_70_Q(1),n,1);repmat(liv_exp_a_70_Q(2),n,1);repmat(liv_exp_a_70_Q(3),
n,1);repmat(liv_exp_a_70_Q(4),n,1);repmat(liv_exp_a_70_Q(5),n,1);repmat(liv_exp_a_70_Q
(6),n,1);repmat(liv_exp_a_70_Q(7),n,1);repmat(liv_exp_a_70_Q(8),n,1);repmat(liv_exp_a_
70_Q(9),n,1);repmat(liv_exp_a_70_Q(10),n,1);repmat(liv_exp_a_70_Q(11),n,1)];
price_outer_eastern.ordered_liv_exp_70 = repmat(liv_exp_apartments,(length(C)),1);

% Compute living expenses based on a reference apartment of 70 sqm
price_outer_eastern.living_exp =
(price_outer_eastern.prom./70).*price_outer_eastern.ordered_liv_exp_70;
avg_living_exp_Q =
mean(price_outer_eastern.living_exp./price_outer_eastern.estimated_price);

% Allocate correct implicit rents
implicit_rent     = zeros(height(price_outer_eastern),1);
rental_allocation = repmat(Rent_Data, length(C), 1);

implicit_rent(find(price_outer_eastern.Soverom == 0)) =
rental_allocation.room1(find(price_outer_eastern.Soverom == 0));
implicit_rent(find(price_outer_eastern.Soverom == 1)) =
rental_allocation.room1(find(price_outer_eastern.Soverom == 1));
implicit_rent(find(price_outer_eastern.Soverom == 2)) =
rental_allocation.room2(find(price_outer_eastern.Soverom == 2));
implicit_rent(find(price_outer_eastern.Soverom == 3)) =
rental_allocation.room3(find(price_outer_eastern.Soverom == 3));
implicit_rent(find(price_outer_eastern.Soverom == 4)) =
rental_allocation.room4(find(price_outer_eastern.Soverom == 4));
implicit_rent(find(price_outer_eastern.Soverom >= 5)) =
rental_allocation.room5_(find(price_outer_eastern.Soverom >= 5));

price_outer_eastern.implicit_rent = implicit_rent;
avg_implicit_rent_Q =
mean(price_outer_eastern.implicit_rent./price_outer_eastern.estimated_price);

% Implement tax rates for sample period
tax_rates = ([ 0.28, 0.28, 0.28, 0.28, 0.27, 0.27, 0.25, 0.24, 0.23, 0.22, 0.22])';
n = 4;
tax_rates_order =
[repmat(tax_rates(1),n,1);repmat(tax_rates(2),n,1);repmat(tax_rates(3),n,1);repmat(tax
_rates(4),n,1);repmat(tax_rates(5),n,1);repmat(tax_rates(6),n,1);repmat(tax_rates(7),n
,1);repmat(tax_rates(8),n,1);repmat(tax_rates(9),n,1);repmat(tax_rates(10),n,1);repmat
(tax_rates(11),n,1)];

% Allocate correct tax rates to each quarter in price_houses set
price_outer_eastern.tax_rates = repmat(tax_rates_order,(length(C)),1);

% Create a table to store different cost rates
avg_cost_rates_outer_eastern                          = zeros(4,3); %JUSTERE DENNE
ETTERSOM HVORDAN STØRRELSEN VIL BLI
avg_cost_rates_outer_eastern                          =
array2table(avg_cost_rates_outer_eastern);
avg_cost_rates_outer_eastern.Properties.VariableNames = {'Average_cost_rates',
'Yearly', 'Quarterly'};
avg_cost_rates_outer_eastern.Average_cost_rates       =
categorical(avg_cost_rates_outer_eastern.Average_cost_rates);

% Import stats for cost rates
avg_cost_rates_outer_eastern.Average_cost_rates(1) = 't_cost_buyer';
```

```
avg_cost_rates_outer_eastern.Yearly(1)             = t_cost_rate;
avg_cost_rates_outer_eastern.Quarterly(1)          = t_cost_rate_Q;

avg_cost_rates_outer_eastern.Average_cost_rates(2) = 't_cost_seller';
avg_cost_rates_outer_eastern.Yearly(2)             = t_cost_s_rate;
avg_cost_rates_outer_eastern.Quarterly(2)          = t_cost_s_rate_Q;

avg_cost_rates_outer_eastern.Average_cost_rates(3) = 'living_expenses';
avg_cost_rates_outer_eastern.Yearly(3)             = avg_living_exp_Q * 4;
avg_cost_rates_outer_eastern.Quarterly(3)          = avg_living_exp_Q;

avg_cost_rates_outer_eastern.Average_cost_rates(4) = 'implicit_rent';
avg_cost_rates_outer_eastern.Yearly(4)             = avg_implicit_rent_Q * 4;
avg_cost_rates_outer_eastern.Quarterly(4)          = avg_implicit_rent_Q;

% 5. ALLOCATE TABLE FOR PRICE INDICES
priceindex_outer_eastern          = table();
Y_index                           = Y_order(1:44);
Q_index                           = Q_rep(1:44);
priceindex_outer_eastern.Q_order  = string(Y_index(:,1)) + Q_index(:,1);
priceindex_outer_eastern.Q_order  = categorical(priceindex_outer_eastern.Q_order);

% Create a table to store returns, std, risk free, sharpe
sum_stats_outer_eastern                          = zeros(10,7);
sum_stats_outer_eastern                          =
array2table(sum_stats_outer_eastern);
sum_stats_outer_eastern.Properties.VariableNames = {'House_Indices', 'Return',
'std_dev','Excess_return','Sharpe_ratio', 'Skewness', 'Kurtosis'};
sum_stats_outer_eastern.House_Indices            =
categorical(sum_stats_outer_eastern.House_Indices);

% Import risk-free rates (innskuddsrente)
n = 4;
priceindex_outer_eastern.rf = zeros(44,1);
priceindex_outer_eastern.rf =
[repmat(Rf.rf(1),n,1);repmat(Rf.rf(2),n,1);repmat(Rf.rf(3),n,1);repmat(Rf.rf(4),n,1);r
epmat(Rf.rf(5),n,1);repmat(Rf.rf(6),n,1);repmat(Rf.rf(7),n,1);repmat(Rf.rf(8),n,1);rep
mat(Rf.rf(9),n,1);repmat(Rf.rf(10),n,1);repmat(Rf.rf(11),n,1)];
priceindex_outer_eastern.rf = priceindex_outer_eastern.rf./4;

% 6. COMPUTATION OF MEAN PRICES FOR HOUSES AND PRICE INDEX BASED ON MEAN PRICES
% Calculate mean prices for houses
priceindex_outer_eastern.mean_prices = zeros(height(priceindex_outer_eastern), end);
for i = 1:length(priceindex_outer_eastern.Q_order)
    priceindex_outer_eastern.mean_prices(i) =
mean(price_outer_eastern.estimated_price(find(price_outer_eastern.Q ==
priceindex_outer_eastern.Q_order(i))));
end

% Calculate price index for houses based on mean prices
priceindex_outer_eastern.index_values = zeros(height(priceindex_outer_eastern), end);
for i = 1:length(priceindex_outer_eastern.index_values)
    priceindex_outer_eastern.index_values(i) =
priceindex_outer_eastern.mean_prices(i)/priceindex_outer_eastern.mean_prices(1);
end

% 7. COMPUTATION OF QUARTERLY RETURNS ON ASSET/HOUSE FROM PERIOD TO PERIOD, NO COSTS
% Compute quarterly returns for each unique ID
price_outer_eastern.Q_ret_no_costs =
(price_outer_eastern.estimated_price./price_outer_eastern.lagged_estimated_price)-1;
price_outer_eastern.Q_ret_no_costs(1:44:end) = NaN;

% Compute mean quarterly returns for houses (mean of all IDs per quarter) (no costs)
priceindex_outer_eastern.Q_ret_no_costs = zeros(height(priceindex_outer_eastern),
end);

for i = 1:height(priceindex_outer_eastern)
    priceindex_outer_eastern.Q_ret_no_costs(i) =
nanmean(price_outer_eastern.Q_ret_no_costs(find(price_outer_eastern.Q ==
priceindex_outer_eastern.Q_order(i))));
end

% Compute mean of mean quarterly returns for houses (no costs)
mean_Q_ret_no_costs = nanmean(priceindex_outer_eastern.Q_ret_no_costs);
```

```
% Compute excess quarterly returns for houses (no costs)
priceindex_outer_eastern.excess_Q_ret_no_costs =
priceindex_outer_eastern.Q_ret_no_costs-priceindex_outer_eastern.rf;

% Compute mean excess quarterly returns for houses (no costs)
mean_excess_Q_ret_no_costs = nanmean(priceindex_outer_eastern.excess_Q_ret_no_costs);

% Compute standard deviation for each unique ID
st_Q_ret_no_costs= zeros(length(C),1);
time(1,:) = clock;
for i = 1:length(C)
    if b(i)<b(length(C))
       st_Q_ret_no_costs(i) =
std(price_outer_eastern.Q_ret_no_costs(((b(i)+1):(b(i+1)-1))));
    else
       st_Q_ret_no_costs(i) =
std(price_outer_eastern.Q_ret_no_costs((b(length(C))+1):height(price_outer_eastern)));
    end
time(i+1,:) = clock;
fprintf('\nTo complete %.f/%.f iterations (unique IDs) it took %.2f seconds\n', i,
length(C), etime(time(i+1,:), time(1,:)));
end

% Standard deviation of all quarterly returns
std_Q_ret_no_costs = mean(st_Q_ret_no_costs);

% Import stats for 'Q_ret_no_costs' in summary table
sum_stats_outer_eastern.House_Indices(1) = 'Q_ret_no_costs';
sum_stats_outer_eastern.Return(1)        = mean_Q_ret_no_costs;
sum_stats_outer_eastern.std_dev(1)       = std_Q_ret_no_costs;
sum_stats_outer_eastern.Excess_return(1) = mean_excess_Q_ret_no_costs;
sum_stats_outer_eastern.Sharpe_ratio(1)  =
sum_stats_outer_eastern.Excess_return(1)/sum_stats_outer_eastern.std_dev(1);
sum_stats_outer_eastern.Skewness(1)      =
skewness(priceindex_outer_eastern.Q_ret_no_costs(2:44));
sum_stats_outer_eastern.Kurtosis(1)      =
kurtosis(priceindex_outer_eastern.Q_ret_no_costs(2:44));

% COMPUTE CUMULATIVE ROA(NC)
first_obs_price = price_outer_eastern.estimated_price(1:44:end);
price_outer_eastern.first_obs_price_repeated = repelem(first_obs_price, 44);

price_outer_eastern.cum_Q_ret_no_costs =
(price_outer_eastern.estimated_price./price_outer_eastern.first_obs_price_repeated)-1;
price_outer_eastern.cum_Q_ret_no_costs(1:44:end) = NaN;

priceindex_outer_eastern.cum_Q_ret_no_costs =
zeros(height(priceindex_outer_eastern),end);
% Compute mean cumulative ROA(NC)
for i = 1:height(priceindex_outer_eastern)
    priceindex_outer_eastern.cum_Q_ret_no_costs(i) =
nanmean(price_outer_eastern.cum_Q_ret_no_costs(find(price_outer_eastern.Q ==
priceindex_outer_eastern.Q_order(i))));
end

% 8. COMPUTATION OF QUARTERLY RETURNS ON ASSET/HOUSE FROM PERIOD TO PERIOD (BUY/SELL),
INCL TOTAL COSTS
% Compute quarterly returns for each unique ID
price_outer_eastern.Q_ret_tot_costs = (((price_outer_eastern.estimated_price -
price_outer_eastern.t_cost_s - price_outer_eastern.lagged_t_cost -
price_outer_eastern.lagged_estimated_price).*(1-price_outer_eastern.tax_rates)) -
price_outer_eastern.living_exp +
price_outer_eastern.implicit_rent)./(price_outer_eastern.lagged_estimated_price);
price_outer_eastern.Q_ret_tot_costs(1:44:end) = NaN;

% Compute mean quarterly returns for houses (mean of all IDs per quarter) (inc tot
costs)
priceindex_outer_eastern.Q_ret_tot_costs = zeros(height(priceindex_outer_eastern),
end);

for i = 1:length(priceindex_outer_eastern.mean_prices)
```

```
    priceindex_outer_eastern.Q_ret_tot_costs(i) =
mean(price_outer_eastern.Q_ret_tot_costs(find(price_outer_eastern.Q ==
priceindex_outer_eastern.Q_order(i))));
end

% Compute mean of mean quarterly returns for houses (inc tot costs)
mean_Q_ret_tot_costs = nanmean(priceindex_outer_eastern.Q_ret_tot_costs);

% Compute excess quarterly returns for houses (inc tot costs)
priceindex_outer_eastern.excess_Q_ret_tot_costs =
priceindex_outer_eastern.Q_ret_tot_costs-priceindex_outer_eastern.rf;

% Compute mean excess quarterly returns for houses (inc tot costs)
mean_excess_Q_ret_tot_costs =
nanmean(priceindex_outer_eastern.excess_Q_ret_tot_costs);

% Compute standard deviation for each unique IDs returns
st_Q_ret_tot_costs= zeros(length(C),1);
time(1,:) = clock;
for i = 1:length(C)
    if b(i)<b(length(C))
       st_Q_ret_tot_costs(i) =
std(price_outer_eastern.Q_ret_tot_costs(((b(i)+1):(b(i+1)-1))));
    else
       st_Q_ret_tot_costs(i) =
std(price_outer_eastern.Q_ret_tot_costs((b(length(C))+1):height(price_outer_eastern)))
;
    end
time(i+1,:) = clock;
fprintf('\nTo complete %.f/%.f iterations (unique IDs) it took %.2f seconds\n', i,
length(C), etime(time(i+1,:), time(1,:)));
end

% Standard deviation of all quarterly returns
std_Q_ret_tot_costs = mean(st_Q_ret_tot_costs);

% Import stats for 'Q_ret_tot_costs' in summary table
sum_stats_outer_eastern.House_Indices(2) = 'Q_ret_tot_costs';
sum_stats_outer_eastern.Return(2)        = mean_Q_ret_tot_costs;
sum_stats_outer_eastern.std_dev(2)       = std_Q_ret_tot_costs;
sum_stats_outer_eastern.Excess_return(2) = mean_excess_Q_ret_tot_costs;
sum_stats_outer_eastern.Sharpe_ratio(2) =
sum_stats_outer_eastern.Excess_return(2)/sum_stats_outer_eastern.std_dev(2);
sum_stats_outer_eastern.Skewness(2)      =
skewness(priceindex_outer_eastern.Q_ret_tot_costs(2:44));
sum_stats_outer_eastern.Kurtosis(2)      =
kurtosis(priceindex_outer_eastern.Q_ret_tot_costs(2:44));

% COMPUTE CUMULATIVE ROA (TC)
taxes_TC = (price_outer_eastern.estimated_price - price_outer_eastern.t_cost_s -
price_outer_eastern.lagged_t_cost -
price_outer_eastern.lagged_estimated_price).*price_outer_eastern.tax_rates;
taxes_TC(1:44:end) = NaN;

net_costs_TC = (price_outer_eastern.living_exp + price_outer_eastern.lagged_t_cost +
price_outer_eastern.t_cost_s + taxes_TC - price_outer_eastern.implicit_rent);
net_costs_TC(1:44:end) = NaN;

% Compute cumulative costs
cumcosts_TC = zeros(height(price_outer_eastern), end);
time(1,:) = clock;
for i = 1:length(C)
    if b(i)<b(length(C))
       cumcosts_TC((b(i)):(b(i+1)-1)) = cumsum(net_costs_TC((b(i)):(b(i+1)-
1)),'omitnan');
    else
       cumcosts_TC(b(length(C)):height(price_outer_eastern)) =
cumsum(net_costs_TC(b(length(C)):height(price_outer_eastern)),'omitnan');
    end
time(i+1,:) = clock;
fprintf('\nTo complete %.f/%.f iterations (unique IDs) it took %.2f seconds\n', i,
length(C), etime(time(i+1,:), time(1,:)));
end
```

```
price_outer_eastern.cum_Q_ret_tot_costs = (price_outer_eastern.estimated_price -
cumcosts_TC -
price_outer_eastern.first_obs_price_repeated)./price_outer_eastern.first_obs_price_rep
eated;
price_outer_eastern.cum_Q_ret_tot_costs(1:44:end) = NaN;

priceindex_outer_eastern.cum_Q_ret_tot_costs =
zeros(height(priceindex_outer_eastern),end);
% Compute mean cumulative returns for houses (mean of all IDs per quarter) (no costs)
for i = 1:height(priceindex_outer_eastern)
    priceindex_outer_eastern.cum_Q_ret_tot_costs(i) =
nanmean(price_outer_eastern.cum_Q_ret_tot_costs(find(price_outer_eastern.Q ==
priceindex_outer_eastern.Q_order(i))));
end

all_lost = priceindex_outer_eastern.cum_Q_ret_tot_costs < -1;
priceindex_outer_eastern.cum_Q_ret_tot_costs(all_lost) = -1;

% 9. COMPUTATION OF QUARTERLY RETURNS ON ASSET/HOUSE FROM PERIOD TO PERIOD , INCL
SMOOTHED COSTS
% Locate buyer and sellers cost when buying in 2010Q1 and selling in 2020Q4
% Find buyer and seller cost per quarter for sample period
cost_2010Q1 = price_outer_eastern.t_cost(1:44:end);
cost_2010Q1 = cost_2010Q1/44;
price_outer_eastern.cost_2010Q1 = repelem(cost_2010Q1, 44);

cost_2020Q4 = price_outer_eastern.t_cost_s(44:44:end);
cost_2020Q4 = cost_2020Q4/44;
price_outer_eastern.cost_2020Q4 = repelem(cost_2020Q4, 44);

% Compute quarterly returns given that you buy in 2010Q1 and sell in 2020Q4
% and spred the relevant costs on the sample period.
price_outer_eastern.Q_ret_smooth_costs = (price_outer_eastern.estimated_price -
price_outer_eastern.cost_2010Q1 - price_outer_eastern.cost_2020Q4 -
price_outer_eastern.living_exp - price_outer_eastern.lagged_estimated_price +
price_outer_eastern.implicit_rent)./price_outer_eastern.lagged_estimated_price;
price_outer_eastern.Q_ret_smooth_costs(1:44:end) = NaN;

% Compute mean quarterly returns for houses (mean of all IDs per quarter) (inc smooth
costs)
priceindex_outer_eastern.Q_ret_smooth_costs = zeros(height(priceindex_outer_eastern),
end);

for i = 1:length(priceindex_outer_eastern.mean_prices)
    priceindex_outer_eastern.Q_ret_smooth_costs(i) =
mean(price_outer_eastern.Q_ret_smooth_costs(find(price_outer_eastern.Q ==
priceindex_outer_eastern.Q_order(i))));
end

% Compute mean of mean quarterly returns for houses (inc smooth costs)
mean_Q_ret_smooth_costs = nanmean(priceindex_outer_eastern.Q_ret_smooth_costs);

% Compute excess quarterly returns for houses (inc smooth costs)
priceindex_outer_eastern.excess_Q_ret_smooth_costs =
priceindex_outer_eastern.Q_ret_smooth_costs - priceindex_outer_eastern.rf;

% Compute mean excess quarterly returns for houses (inc smooth costs)
mean_excess_Q_ret_smooth_costs =
nanmean(priceindex_outer_eastern.excess_Q_ret_smooth_costs);

% Compute standard deviation for each unique IDs returns
st_Q_ret_smooth_costs= zeros(length(C),1);
time(1,:) = clock;
for i = 1:length(C)
    if b(i)<b(length(C))
       st_Q_ret_smooth_costs(i) =
std(price_outer_eastern.Q_ret_smooth_costs(((b(i)+1):(b(i+1)-1))));
    else
       st_Q_ret_smooth_costs(i) =
std(price_outer_eastern.Q_ret_smooth_costs((b(length(C))+1):height(price_outer_eastern
)));
    end
time(i+1,:) = clock;
```

```
fprintf('\nTo complete %.f/%.f iterations (unique IDs) it took %.2f seconds\n', i,
length(C), etime(time(i+1,:), time(1,:)));
end

% Standard deviation of all quarterly returns
std_Q_ret_smooth_costs = mean(st_Q_ret_smooth_costs);

% Import stats for 'Q_ret_smooth_costs' in summary table
sum_stats_outer_eastern.House_Indices(3) = 'Q_ret_smooth_costs';
sum_stats_outer_eastern.Return(3)        = mean_Q_ret_smooth_costs;
sum_stats_outer_eastern.std_dev(3)       = std_Q_ret_smooth_costs;
sum_stats_outer_eastern.Excess_return(3) = mean_excess_Q_ret_smooth_costs;
sum_stats_outer_eastern.Sharpe_ratio(3) =
sum_stats_outer_eastern.Excess_return(3)/sum_stats_outer_eastern.std_dev(3);
sum_stats_outer_eastern.Skewness(3)      =
skewness(priceindex_outer_eastern.Q_ret_smooth_costs(2:44));
sum_stats_outer_eastern.Kurtosis(3)      =
kurtosis(priceindex_outer_eastern.Q_ret_smooth_costs(2:44));

% COMPUTE CUMULATIVE ROA(SC)
net_costs_SC = (price_outer_eastern.living_exp + price_outer_eastern.cost_2010Q1 +
price_outer_eastern.cost_2020Q4 - price_outer_eastern.implicit_rent);
net_costs_SC(1:44:end) = NaN;

cumcosts_SC = zeros(height(price_outer_eastern), end);
time(1,:) = clock;
for i = 1:length(C)
    if b(i)<b(length(C))
       cumcosts_SC((b(i)):(b(i+1)-1)) = cumsum(net_costs_SC((b(i)):(b(i+1)-
1)),'omitnan');
    else
       cumcosts_SC(b(length(C)):height(price_outer_eastern)) =
cumsum(net_costs_SC(b(length(C)):height(price_outer_eastern)),'omitnan');
    end
time(i+1,:) = clock;
fprintf('\nTo complete %.f/%.f iterations (unique IDs) it took %.2f seconds\n', i,
length(C), etime(time(i+1,:), time(1,:)));
end

price_outer_eastern.cum_Q_ret_smooth_costs = (price_outer_eastern.estimated_price -
cumcosts_SC -
price_outer_eastern.first_obs_price_repeated)./price_outer_eastern.first_obs_price_rep
eated;
price_outer_eastern.cum_Q_ret_smooth_costs(1:44:end) = NaN;

% Compute mean cumulative returns for houses (mean of all IDs per quarter) (no costs)
priceindex_outer_eastern.cum_Q_ret_smooth_costs =
zeros(height(priceindex_outer_eastern),end);
for i = 1:height(priceindex_outer_eastern)
    priceindex_outer_eastern.cum_Q_ret_smooth_costs(i) =
nanmean(price_outer_eastern.cum_Q_ret_smooth_costs(find(price_outer_eastern.Q ==
priceindex_outer_eastern.Q_order(i))));
end

% Identify average loan-to-value ratio for housing investments
l = ([0.6957, 0.6957, 0.6957, 0.6957, 0.71, 0.76, 0.67, 0.66, 0.68, 0.70, 0.69])';
n = 4;
ltv_ratio_order =
[repmat(l(1),n,1);repmat(l(2),n,1);repmat(l(3),n,1);repmat(l(4),n,1);repmat(l(5),n,1);
repmat(l(6),n,1);repmat(l(7),n,1);repmat(l(8),n,1);repmat(l(9),n,1);repmat(l(10),n,1);
repmat(l(11),n,1)];

% Allocate correct ltv-rates to each quarter in price_houses set
price_outer_eastern.ltvr        = repmat(ltv_ratio_order,(length(C)),1);
price_outer_eastern.lagged_ltvr = lagmatrix(price_outer_eastern.ltvr,1);

% Allocate ltv ratio as of 2010Q1 to each quarter in price_houses set
price_outer_eastern.ltvr_2010Q1(:) = price_outer_eastern.ltvr(1);

% 10. WE DONT USE THE RETURN SERIES IN SECTION 10
% Compute quarterly equity returns for each unique ID (no costs)
price_outer_eastern.Q_eq_ret_no_costs = ((price_outer_eastern.estimated_price -
price_outer_eastern.lagged_estimated_price)./(price_outer_eastern.lagged_estimated_pri
ce.*(1-price_outer_eastern.lagged_ltvr)));
```

```
price_outer_eastern.Q_eq_ret_no_costs(1:44:end) = NaN;

% Compute mean quarterly equity returns for houses (mean of all IDs per quarter) (no
costs)
priceindex_outer_eastern.Q_eq_ret_no_costs = zeros(height(priceindex_outer_eastern),
end);

for i = 1:height(priceindex_outer_eastern)
    priceindex_outer_eastern.Q_eq_ret_no_costs(i) =
mean(price_outer_eastern.Q_eq_ret_no_costs(find(price_outer_eastern.Q ==
priceindex_outer_eastern.Q_order(i))));
end

% Compute mean of mean quarterly equity returns for houses (no costs)
mean_Q_eq_ret_no_costs = nanmean(priceindex_outer_eastern.Q_eq_ret_no_costs);

% Compute excess quarterly equity returns for houses (no costs)
priceindex_outer_eastern.excess_Q_eq_ret_no_costs =
priceindex_outer_eastern.Q_eq_ret_no_costs - priceindex_outer_eastern.rf;

% Compute mean excess quarterly equity returns for houses (no costs)
mean_excess_Q_eq_ret_no_costs =
nanmean(priceindex_outer_eastern.excess_Q_eq_ret_no_costs);

% Compute standard deviation for each unique IDs returns
st_Q_eq_ret_no_costs= zeros(length(C),1);
time(1,:) = clock;
for i = 1:length(C)
    if b(i)<b(length(C))
       st_Q_eq_ret_no_costs(i) =
std(price_outer_eastern.Q_eq_ret_no_costs(((b(i)+1):(b(i+1)-1))));
    else
       st_Q_eq_ret_no_costs(i) =
std(price_outer_eastern.Q_eq_ret_no_costs((b(length(C))+1):height(price_outer_eastern)
));
    end
time(i+1,:) = clock;
fprintf('\nTo complete %.f/%.f iterations (unique IDs) it took %.2f seconds\n', i,
length(C), etime(time(i+1,:), time(1,:)));
end

% Standard deviation of all quarterly equity returns
std_Q_eq_ret_no_costs = mean(st_Q_eq_ret_no_costs);

% Import stats for 'Q_eq_ret_no_costs' in summary table
sum_stats_outer_eastern.House_Indices(4) = 'Q_eq_ret_no_costs';
sum_stats_outer_eastern.Return(4)        = mean_Q_eq_ret_no_costs;
sum_stats_outer_eastern.std_dev(4)       = std_Q_eq_ret_no_costs;
sum_stats_outer_eastern.Excess_return(4) = mean_excess_Q_eq_ret_no_costs;
sum_stats_outer_eastern.Sharpe_ratio(4) =
sum_stats_outer_eastern.Excess_return(4)/sum_stats_outer_eastern.std_dev(4);
sum_stats_outer_eastern.Skewness(4)      =
skewness(priceindex_outer_eastern.Q_eq_ret_no_costs(2:44));
sum_stats_outer_eastern.Kurtosis(4)      =
kurtosis(priceindex_outer_eastern.Q_eq_ret_no_costs(2:44));

% COMPUTE CUMULATIVE ROE (NC)
price_outer_eastern.cum_Q_eq_ret_no_costs = ((price_outer_eastern.estimated_price -
price_outer_eastern.first_obs_price_repeated)./(price_outer_eastern.first_obs_price_re
peated.*(1-price_outer_eastern.ltvr_2010Q1)));
price_outer_eastern.cum_Q_eq_ret_no_costs(1:44:end) = NaN;

% Compute mean cumulative returns for houses (mean of all IDs per quarter) (no costs)
priceindex_outer_eastern.cum_Q_eq_ret_no_costs =
zeros(height(priceindex_outer_eastern),end);
for i = 1:height(priceindex_outer_eastern)
    priceindex_outer_eastern.cum_Q_eq_ret_no_costs(i) =
nanmean(price_outer_eastern.cum_Q_eq_ret_no_costs(find(price_outer_eastern.Q ==
priceindex_outer_eastern.Q_order(i))));
end

% 11. WE DONT USE THE RETURN SERIES IN SECTION 11
% Compute quarterly equity returns for each unique ID (inc tot costs)
```

```
price_outer_eastern.Q_eq_ret_tot_costs = (((price_outer_eastern.estimated_price -
price_outer_eastern.t_cost_s - price_outer_eastern.lagged_t_cost -
price_outer_eastern.lagged_estimated_price).*(1-price_outer_eastern.tax_rates)) -
price_outer_eastern.living_exp +
price_outer_eastern.implicit_rent)./(price_outer_eastern.lagged_estimated_price.*(1-
price_outer_eastern.lagged_ltvr));
price_outer_eastern.Q_eq_ret_tot_costs(1:44:end) = NaN;

% Compute mean quarterly equity returns for houses (mean of all IDs per quarter) (inc
tot costs)
priceindex_outer_eastern.Q_eq_ret_tot_costs = zeros(height(priceindex_outer_eastern),
end);

for i = 1:length(priceindex_outer_eastern.mean_prices)
    priceindex_outer_eastern.Q_eq_ret_tot_costs(i) =
mean(price_outer_eastern.Q_eq_ret_tot_costs(find(price_outer_eastern.Q ==
priceindex_outer_eastern.Q_order(i))));
end

% Compute mean of mean quarterly equity returns for houses (inc tot costs)
mean_Q_eq_ret_tot_costs = nanmean(priceindex_outer_eastern.Q_eq_ret_tot_costs);

% Compute excess quarterly equity returns for houses (inc tot costs)
priceindex_outer_eastern.excess_Q_eq_ret_tot_costs =
priceindex_outer_eastern.Q_eq_ret_tot_costs - priceindex_outer_eastern.rf;

% Compute mean excess quarterly equity returns for houses (inc tot costs)
mean_excess_Q_eq_ret_tot_costs =
nanmean(priceindex_outer_eastern.excess_Q_eq_ret_tot_costs);

% Compute standard deviation for each unique IDs returns
st_Q_eq_ret_tot_costs= zeros(length(C),1);
time(1,:) = clock;
for i = 1:length(C)
    if b(i)<b(length(C))
       st_Q_eq_ret_tot_costs(i) =
std(price_outer_eastern.Q_eq_ret_tot_costs(((b(i)+1):(b(i+1)-1))));
    else
       st_Q_eq_ret_tot_costs(i) =
std(price_outer_eastern.Q_eq_ret_tot_costs((b(length(C))+1):height(price_outer_eastern
)));
    end
time(i+1,:) = clock;
fprintf('\nTo complete %.f/%.f iterations (unique IDs) it took %.2f seconds\n', i,
length(C), etime(time(i+1,:), time(1,:)));
end

% Standard deviation of all quarterly equity returns
std_Q_eq_ret_tot_costs = mean(st_Q_eq_ret_tot_costs);

% Import stats for 'Q_eq_ret_tot_costs' in summary table
sum_stats_outer_eastern.House_Indices(5) = 'Q_eq_ret_tot_costs';
sum_stats_outer_eastern.Return(5)        = mean_Q_eq_ret_tot_costs;
sum_stats_outer_eastern.std_dev(5)       = std_Q_eq_ret_tot_costs;
sum_stats_outer_eastern.Excess_return(5) = mean_excess_Q_eq_ret_tot_costs;
sum_stats_outer_eastern.Sharpe_ratio(5) =
sum_stats_outer_eastern.Excess_return(5)/sum_stats_outer_eastern.std_dev(5);
sum_stats_outer_eastern.Skewness(5)      =
skewness(priceindex_outer_eastern.Q_eq_ret_tot_costs(2:44));
sum_stats_outer_eastern.Kurtosis(5)      =
kurtosis(priceindex_outer_eastern.Q_eq_ret_tot_costs(2:44));

% COMPUTE CUMULATIVE RETURNS BASED ON QUARTERLY EQUITY RETURNS FROM PRICE INDEX (INC
TOT COSTS)
priceindex_outer_eastern.cum_Q_eq_ret_tot_costs =
zeros(height(priceindex_outer_eastern),end);
priceindex_outer_eastern.cum_Q_eq_ret_tot_costs =
cumprod((1+priceindex_outer_eastern.Q_eq_ret_tot_costs),'omitnan') -1;

% 12. COMPUTATION OF QUARTERLY EQUITY RETURNS ON ASSET/HOUSE FROM PERIOD TO PERIOD ,
INCL SMOOTHED COSTS
% Compute quarterly EQUITY returns given that you buy in 2010Q1 and sell in 2020Q4
% and spread the relevant costs on the sample period.
```

```
price_outer_eastern.Q_eq_ret_smooth_costs = (price_outer_eastern.estimated_price -
price_outer_eastern.cost_2010Q1 - price_outer_eastern.cost_2020Q4 -
price_outer_eastern.living_exp - price_outer_eastern.lagged_estimated_price +
price_outer_eastern.implicit_rent)./(price_outer_eastern.lagged_estimated_price.*(1-
price_outer_eastern.ltvr_2010Q1));
price_outer_eastern.Q_eq_ret_smooth_costs(1:44:end) = NaN;

% Compute mean quarterly equity returns for houses (mean of all IDs per quarter) (inc
smooth costs)
priceindex_outer_eastern.Q_eq_ret_smooth_costs =
zeros(height(priceindex_outer_eastern), end);

for i = 1:length(priceindex_outer_eastern.mean_prices)
    priceindex_outer_eastern.Q_eq_ret_smooth_costs(i) =
mean(price_outer_eastern.Q_eq_ret_smooth_costs(find(price_outer_eastern.Q ==
priceindex_outer_eastern.Q_order(i))));
end

% Compute mean of mean quarterly equity returns for houses (inc smooth costs)
mean_Q_eq_ret_smooth_costs = nanmean(priceindex_outer_eastern.Q_eq_ret_smooth_costs);

% Compute excess quarterly equity returns for houses (inc smooth costs)
priceindex_outer_eastern.excess_Q_eq_ret_smooth_costs =
priceindex_outer_eastern.Q_eq_ret_smooth_costs - priceindex_outer_eastern.rf;

% Compute mean excess quarterly equity returns for houses (inc smooth costs)
mean_excess_Q_eq_ret_smooth_costs =
nanmean(priceindex_outer_eastern.excess_Q_eq_ret_smooth_costs);

% Compute standard deviation for each unique IDs returns
st_Q_eq_ret_smooth_costs= zeros(length(C),1);
time(1,:) = clock;
for i = 1:length(C)
    if b(i)<b(length(C))
       st_Q_eq_ret_smooth_costs(i) =
std(price_outer_eastern.Q_eq_ret_smooth_costs(((b(i)+1):(b(i+1)-1))));
    else
       st_Q_eq_ret_smooth_costs(i) =
std(price_outer_eastern.Q_eq_ret_smooth_costs((b(length(C))+1):height(price_outer_east
ern)));
    end
time(i+1,:) = clock;
fprintf('\nTo complete %.f/%.f iterations (unique IDs) it took %.2f seconds\n', i,
length(C), etime(time(i+1,:), time(1,:)));
end

% Standard deviation of all quarterly equity returns
std_Q_eq_ret_smooth_costs = mean(st_Q_eq_ret_smooth_costs);

% Import stats for 'Q_eq_ret_smooth_costs' in summary table
sum_stats_outer_eastern.House_Indices(6) = 'Q_eq_ret_smooth_costs';
sum_stats_outer_eastern.Return(6)        = mean_Q_eq_ret_smooth_costs;
sum_stats_outer_eastern.std_dev(6)       = std_Q_eq_ret_smooth_costs;
sum_stats_outer_eastern.Excess_return(6) = mean_excess_Q_eq_ret_smooth_costs;
sum_stats_outer_eastern.Sharpe_ratio(6) =
sum_stats_outer_eastern.Excess_return(6)/sum_stats_outer_eastern.std_dev(6);
sum_stats_outer_eastern.Skewness(6)      =
skewness(priceindex_outer_eastern.Q_eq_ret_smooth_costs(2:44));
sum_stats_outer_eastern.Kurtosis(6)      =
kurtosis(priceindex_outer_eastern.Q_eq_ret_smooth_costs(2:44));

% COMPUTE CUMULATIVE ROE(SC)
price_outer_eastern.cum_Q_eq_ret_smooth_costs = (price_outer_eastern.estimated_price -
cumcosts_SC -
price_outer_eastern.first_obs_price_repeated)./(price_outer_eastern.first_obs_price_re
peated.*(1-price_outer_eastern.ltvr_2010Q1));
price_outer_eastern.cum_Q_eq_ret_smooth_costs(1:44:end) = NaN;

% Compute mean cumulative returns for houses (mean of all IDs per quarter) (smooth
costs)
priceindex_outer_eastern.cum_Q_eq_ret_smooth_costs =
zeros(height(priceindex_outer_eastern),end);
for i = 1:height(priceindex_outer_eastern)
```

```
    priceindex_outer_eastern.cum_Q_eq_ret_smooth_costs(i) =
nanmean(price_outer_eastern.cum_Q_eq_ret_smooth_costs(find(price_outer_eastern.Q ==
priceindex_outer_eastern.Q_order(i))));
end

% MAKE PLOT OF THE RELEVANT INDICES TOGHETER, MAKE MULTIPLE PLOTS FOR COMPARISON
REASONS
outer_eastern_price_changes = priceindex_outer_eastern(:,[1 5 8 11 14 17 20]);

outer_eastern_price_indices = priceindex_outer_eastern(:,[1 7 10 13 16 19 22]);
outer_eastern_price_indices.cum_Q_ret_no_costs(1) = 0;
outer_eastern_price_indices.cum_Q_ret_tot_costs(1) = 0;
outer_eastern_price_indices.cum_Q_ret_smooth_costs(1) = 0;
outer_eastern_price_indices.cum_Q_eq_ret_no_costs(1) = 0;
outer_eastern_price_indices.cum_Q_eq_ret_tot_costs(1) = 0;
outer_eastern_price_indices.cum_Q_eq_ret_smooth_costs(1) = 0;

outer_eastern_price_indices.cum_Q_ret_no_costs        =
outer_eastern_price_indices.cum_Q_ret_no_costs + 1;
outer_eastern_price_indices.cum_Q_ret_tot_costs       =
outer_eastern_price_indices.cum_Q_ret_tot_costs + 1;
outer_eastern_price_indices.cum_Q_ret_smooth_costs    =
outer_eastern_price_indices.cum_Q_ret_smooth_costs + 1;
outer_eastern_price_indices.cum_Q_eq_ret_no_costs     =
outer_eastern_price_indices.cum_Q_eq_ret_no_costs + 1;
outer_eastern_price_indices.cum_Q_eq_ret_tot_costs    =
outer_eastern_price_indices.cum_Q_eq_ret_tot_costs + 1;
outer_eastern_price_indices.cum_Q_eq_ret_smooth_costs =
outer_eastern_price_indices.cum_Q_eq_ret_smooth_costs + 1;

clearvars -except Property_Data summary_table Rf Rent_Data...
                  house_set hedonic_houses housing_model...
                  price_houses uniq_house_set avg_cost_rates_houses...
                  priceindex_houses sum_stats_houses houseprice_changes...
                  houseprice_indices...
                  smallhouse_set hedonic_smallhouses smallhouse_model...
                  price_smallhouses uniq_smallhouse_set avg_cost_rates_smallhouses...
                  priceindex_smallhouses sum_stats_smallhouses...
                  smallhouse_price_changes smallhouse_price_indices...
                  apartment_set hedonic_apartments apartment_model price_apartments...
                  uniq_apartment_set avg_cost_rates_apartments
priceindex_apartments...
                  sum_stats_apartments apartment_price_changes...
                  apartment_price_indices...
                  inner_eastern_set hedonic_inner_eastern inner_eastern_model...
                  price_inner_eastern uniq_inner_eastern_set
avg_cost_rates_inner_eastern...
                  priceindex_inner_eastern sum_stats_inner_eastern
inner_eastern_price_changes...
                  inner_eastern_price_indices...
                  inner_western_set hedonic_inner_western inner_western_model...
                  price_inner_western uniq_inner_western_set
avg_cost_rates_inner_western...
                  priceindex_inner_western sum_stats_inner_western
inner_western_price_changes...
                  inner_western_price_indices...
                  outer_eastern_set hedonic_outer_eastern outer_eastern_model...
                  price_outer_eastern uniq_outer_eastern_set
avg_cost_rates_outer_eastern...
                  priceindex_outer_eastern sum_stats_outer_eastern
outer_eastern_price_changes...
                  outer_eastern_price_indices...
                  outer_western_set hedonic_outer_western outer_western_model...
                  price_outer_western uniq_outer_western_set
avg_cost_rates_outer_western...
                  priceindex_outer_western sum_stats_outer_western
outer_western_price_changes...
                  outer_western_price_indices...
                  outer_southern_set hedonic_outer_southern outer_southern_model...
                  price_outer_southern uniq_outer_southern_set
avg_cost_rates_outer_southern...
                  priceindex_outer_southern sum_stats_outer_southern
outer_southern_price_changes...
                  outer_southern_price_indices...
```

```
                property_price_changes property_price_indices hedonic_total...
                ROA_NC_stats ROE_NC_stats ROA_TC_stats ROE_TC_stats...
                ROA_SC_stats ROE_SC_stats sum_stats_OSEFX OSEFX...
                ROA_NC_sharpe_test ROA_TC_sharpe_test ROA_SC_sharpe_test...
                ROE_SC_sharpe_test

%% Regression for Outer Southern (Apartments)
% 1. PREPERATION OF DATASET
outer_southern      = (apartment_set.Bydel == 'Østensjø') + (apartment_set.Bydel ==
'Nordstrand') + (apartment_set.Bydel == 'Søndre Nordstrand');
outer_southern      = logical(outer_southern);
outer_southern_set  = apartment_set(outer_southern,:);

% Make categorical for dummy-purposes
outer_southern_set.Alder    = categorical(outer_southern_set.Alder);
outer_southern_set.Q        = categorical(outer_southern_set.Q);
outer_southern_set.Eierform = categorical(outer_southern_set.Eierform);
outer_southern_set.Etasje   = categorical(outer_southern_set.Etasje);

% Reorder categories/dummies to find reference variables
outer_southern_set.Alder    = reordercats(outer_southern_set.Alder,    {'Age1',
'Age2', 'Age3', 'Age4'});
outer_southern_set.Eierform = reordercats(outer_southern_set.Eierform, {'Selveier',
'Borettslag'});
outer_southern_set.Etasje   = reordercats(outer_southern_set.Etasje,   {'1', '2', '3',
'4', '5', '6', '7', '8', '9', '10', '11', '12', '13', '14', '15', '-1', '-2', '-3', '-
4'});

% Remove unused categories
outer_southern_set.Eierform = removecats(outer_southern_set.Eierform);

% Check that we in fact removed unused categories
categories(outer_southern_set.Alder)
categories(outer_southern_set.Eierform)

% Make log prices and sizes
outer_southern_set.log_pris = log(outer_southern_set.Pris);
outer_southern_set.log_prom = log(outer_southern_set.prom);


% 2. Run regression - Log model with log(prom), Q, Bydel, Soverom, Alder on log(pris)
% Explanatory variables log(prom), soverom - continous
% Explanatory variables Q, Bydel, Eierform, Alder and Etasje - dummies
outer_southern_model =
fitlm(outer_southern_set,'log_pris~log_prom+Q+Eierform+Soverom+Alder+Etasje');

% 3. ALLOCATE TABLE FOR ESTIMATED PRICES (BASED ON MODEL FROM PT. 2) FOR EVERY QUARTER
FOR EACH ID
outer_southern_set   = sortrows(outer_southern_set,'BoligID','ascend');
[C,ia,ic]        = unique(outer_southern_set.BoligID,'first');
price_outer_southern = zeros(length(C)*44, width(outer_southern_set));
price_outer_southern = array2table(price_outer_southern);

price_outer_southern.Properties.VariableNames = {'BoligID', 'Salgsdato', 'Bydel',
'Pris', 'Fellesgjeld', 'Prisantydning', 'Eierform', 'Byggeaar', 'prom', 'Etasje',
'Soverom', 'Boligtype', 'year', 'Q', 'Alder', 'kvmpris', 'log_pris', 'log_prom' };

Q_rep   = repmat(["Q1"; "Q2"; "Q3";"Q4"],(length(C)*44)/4,1);
x       = (2010:2020)';
n       = 4;
newx    =
[repmat(x(1),n,1);repmat(x(2),n,1);repmat(x(3),n,1);repmat(x(4),n,1);repmat(x(5),n,1);
repmat(x(6),n,1);repmat(x(7),n,1);repmat(x(8),n,1);repmat(x(9),n,1);repmat(x(10),n,1);
repmat(x(11),n,1)];
Y_order = repmat(newx,(length(C)),1);

price_outer_southern.Q_order = string(Y_order(:,1)) + Q_rep(:,1);
price_outer_southern.Q       = price_outer_southern.Q_order;
price_outer_southern.Q_order = [];

price_outer_southern.BoligID = categorical(price_outer_southern.BoligID);
price_outer_southern.BoligID = repelem(C, 44);
```

```
price_outer_southern           = convertvars(price_outer_southern,{'BoligID', 'Bydel',
'Eierform', 'Boligtype', 'Etasje', 'Alder', 'Q'},'categorical');
price_outer_southern.Salgsdato =
datetime(price_outer_southern.Salgsdato,'ConvertFrom','yyyymmdd');

b = 1:length(newx):height(price_outer_southern);

% Create set containing only the unique observations of house_set. When
% multiple unique values it stores the first observation
uniq_outer_southern_set = outer_southern_set(ia,:);

% Allocate variables into price_houses to use for regression model
price_outer_southern(:,[3 5 7 9 10 11 15 18]) = repelem(uniq_outer_southern_set(:,[3 5
7 9 10 11 15 18]), 44, 1);

% Store relevant variables for regression model in new table
x3 = price_outer_southern(:,[18 14 7 11 15 10]);

% GET COLUMN OF ESTIMATED PRICES BASED ON REGRESSION MODEL
price_outer_southern.log_estimated_price    = predict(outer_southern_model, x3);
price_outer_southern.estimated_price        =
exp(price_outer_southern.log_estimated_price);
price_outer_southern.lagged_estimated_price =
lagmatrix(price_outer_southern.estimated_price,1);

% 4. IMPLEMENT RELEVANT COSTS RELATED TO HOUSING INVESTMENTS
% Transaction costs buyer, b
gebyr_b             = zeros(height(price_outer_southern),1);
dokavg_b            = zeros(height(price_outer_southern),1);
insurance_b         = zeros(height(price_outer_southern),1);
price_outer_southern.t_cost = zeros(height(price_outer_southern),1);

% If type of ownership = 'Selveier'
idx_selv              = price_outer_southern.Eierform == "Selveier";
gebyr_b(idx_selv)     = 540 + 540 + 172;
dokavg_b(idx_selv)    =
0.025.*(price_outer_southern.estimated_price(idx_selv)+price_outer_southern.Fellesgjel
d(idx_selv));
insurance_b(idx_selv) = 7400;

% If type of ownership = 'Borettslag'
idx_andel              = price_outer_southern.Eierform == "Borettslag";
gebyr_b(idx_andel)     = 440 + 440 + 172;
dokavg_b(idx_andel)    = 0;
insurance_b(idx_andel) = 4100;

% Compute total transaction cost for buyer, b, for each observation/quarter
price_outer_southern.t_cost        = gebyr_b+dokavg_b+insurance_b;
price_outer_southern.lagged_t_cost = lagmatrix(price_outer_southern.t_cost,1);

% Transaction costs seller, s
eierskiftegebyr_s     = zeros(height(price_outer_southern),1);
public_fees_s         = zeros(height(price_outer_southern),1);
insurance_s           = zeros(height(price_outer_southern),1);
commission_s          = zeros(height(price_outer_southern),1);
price_outer_southern.t_cost_s = zeros(height(price_outer_southern),1);
indices_selv          = zeros(height(price_outer_southern),1);
indices_andel         = zeros(height(price_outer_southern),1);

% If type of ownership = 'Selveier'
idx_selv                   = price_outer_southern.Eierform == "Selveier";
public_fees_s(idx_selv)    = 8430;
eierskiftegebyr_s(idx_selv) = 0;
insurance_s(idx_selv)      =
max(0.0033*(price_outer_southern.estimated_price(idx_selv)+price_outer_southern.Felles
gjeld(idx_selv)), 4000);
indices_selv               = insurance_s > 24000;
indices_selv               = logical(indices_selv);
insurance_s(indices_selv)  = 24000;

% If type of ownership = 'Borettslag'
idx_andel                  = price_outer_southern.Eierform == "Borettslag";
public_fees_s(idx_andel)   = 14319;
eierskiftegebyr_s(idx_andel) = 4858;
```

```
insurance_s(idx_andel)       =
max(0.002*(price_outer_southern.estimated_price(idx_andel)+price_outer_southern.Felles
gjeld(idx_andel)), 2500);
indices_andel                = zeros(height(price_outer_southern),1);
indices_andel(idx_andel)     = insurance_s(idx_andel) > 24000;
indices_andel                = logical(indices_andel);
insurance_s(indices_andel)   = 24000;

% Commission to broker independent of type of ownership
commission_s =
max(0.02*(price_outer_southern.estimated_price+price_outer_southern.Fellesgjeld),40000
);

% Compute total transaction cost for seller, s, for each observation/quarter
price_outer_southern.t_cost_s =
public_fees_s+eierskiftegebyr_s+insurance_s+commission_s;

% Compute transaction cost rates for buyer and seller
% For buyer
t_cost_rate   =
mean(price_outer_southern.t_cost./price_outer_southern.estimated_price);
t_cost_rate_Q = t_cost_rate/4;

% For seller
t_cost_s_rate   =
mean(price_outer_southern.t_cost_s./price_outer_southern.estimated_price);
t_cost_s_rate_Q = t_cost_s_rate/4;

% Define holding period for each dwelling
holding_period = 11;

% Define living expenses per quarter - reference apartment is 70 sqm
liv_exp_a_70_Q = ([76734/4, 82530/4, 83430/4, 89095/4, 88857/4, 83084/4, 83769/4,
86976/4, 91419/4, 103712/4, 87905/4])';

% Allocate correct liwing expenses to each quarter in price_houses set
n = 4;
liv_exp_apartments =
[repmat(liv_exp_a_70_Q(1),n,1);repmat(liv_exp_a_70_Q(2),n,1);repmat(liv_exp_a_70_Q(3),
n,1);repmat(liv_exp_a_70_Q(4),n,1);repmat(liv_exp_a_70_Q(5),n,1);repmat(liv_exp_a_70_Q
(6),n,1);repmat(liv_exp_a_70_Q(7),n,1);repmat(liv_exp_a_70_Q(8),n,1);repmat(liv_exp_a_
70_Q(9),n,1);repmat(liv_exp_a_70_Q(10),n,1);repmat(liv_exp_a_70_Q(11),n,1)];
price_outer_southern.ordered_liv_exp_70 = repmat(liv_exp_apartments,(length(C)),1);

% Compute living expenses based on a reference apartment of 70 sqm
price_outer_southern.living_exp =
(price_outer_southern.prom./70).*price_outer_southern.ordered_liv_exp_70;
avg_living_exp_Q =
mean(price_outer_southern.living_exp./price_outer_southern.estimated_price);

% Allocate correct implicit rents
implicit_rent     = zeros(height(price_outer_southern),1);
rental_allocation = repmat(Rent_Data, length(C), 1);

implicit_rent(find(price_outer_southern.Soverom == 0)) =
rental_allocation.room1(find(price_outer_southern.Soverom == 0));
implicit_rent(find(price_outer_southern.Soverom == 1)) =
rental_allocation.room1(find(price_outer_southern.Soverom == 1));
implicit_rent(find(price_outer_southern.Soverom == 2)) =
rental_allocation.room2(find(price_outer_southern.Soverom == 2));
implicit_rent(find(price_outer_southern.Soverom == 3)) =
rental_allocation.room3(find(price_outer_southern.Soverom == 3));
implicit_rent(find(price_outer_southern.Soverom == 4)) =
rental_allocation.room4(find(price_outer_southern.Soverom == 4));
implicit_rent(find(price_outer_southern.Soverom >= 5)) =
rental_allocation.room5_(find(price_outer_southern.Soverom >= 5));

price_outer_southern.implicit_rent = implicit_rent;
avg_implicit_rent_Q =
mean(price_outer_southern.implicit_rent./price_outer_southern.estimated_price);

% Implement tax rates for sample period
tax_rates = ([ 0.28, 0.28, 0.28, 0.28, 0.27, 0.27, 0.25, 0.24, 0.23, 0.22, 0.22])';
n = 4;
```

```
tax_rates_order =
[repmat(tax_rates(1),n,1);repmat(tax_rates(2),n,1);repmat(tax_rates(3),n,1);repmat(tax
_rates(4),n,1);repmat(tax_rates(5),n,1);repmat(tax_rates(6),n,1);repmat(tax_rates(7),n
,1);repmat(tax_rates(8),n,1);repmat(tax_rates(9),n,1);repmat(tax_rates(10),n,1);repmat
(tax_rates(11),n,1)];

% Allocate correct tax rates to each quarter in price_houses set
price_outer_southern.tax_rates = repmat(tax_rates_order,(length(C)),1);

% Create a table to store different cost rates
avg_cost_rates_outer_southern                          = zeros(4,3); %JUSTERE DENNE
ETTERSOM HVORDAN ST√òRRELSEN VIL BLI
avg_cost_rates_outer_southern                          =
array2table(avg_cost_rates_outer_southern);
avg_cost_rates_outer_southern.Properties.VariableNames = {'Average_cost_rates',
'Yearly', 'Quarterly'};
avg_cost_rates_outer_southern.Average_cost_rates       =
categorical(avg_cost_rates_outer_southern.Average_cost_rates);

% Import stats for cost rates
avg_cost_rates_outer_southern.Average_cost_rates(1) = 't_cost_buyer';
avg_cost_rates_outer_southern.Yearly(1)             = t_cost_rate;
avg_cost_rates_outer_southern.Quarterly(1)          = t_cost_rate_Q;

avg_cost_rates_outer_southern.Average_cost_rates(2) = 't_cost_seller';
avg_cost_rates_outer_southern.Yearly(2)             = t_cost_s_rate;
avg_cost_rates_outer_southern.Quarterly(2)          = t_cost_s_rate_Q;

avg_cost_rates_outer_southern.Average_cost_rates(3) = 'living_expenses';
avg_cost_rates_outer_southern.Yearly(3)             = avg_living_exp_Q * 4;
avg_cost_rates_outer_southern.Quarterly(3)          = avg_living_exp_Q;

% 5. ALLOCATE TABLE FOR PRICE INDICES
priceindex_outer_southern          = table();
Y_index                           = Y_order(1:44);
Q_index                           = Q_rep(1:44);
priceindex_outer_southern.Q_order  = string(Y_index(:,1)) + Q_index(:,1);
priceindex_outer_southern.Q_order  = categorical(priceindex_outer_southern.Q_order);

% Create a table to store returns, std, risk free, sharpe
sum_stats_outer_southern                           = zeros(10,7);
sum_stats_outer_southern                           =
array2table(sum_stats_outer_southern);
sum_stats_outer_southern.Properties.VariableNames = {'House_Indices', 'Return',
'std_dev','Excess_return','Sharpe_ratio', 'Skewness', 'Kurtosis'};
sum_stats_outer_southern.House_Indices             =
categorical(sum_stats_outer_southern.House_Indices);

% Import risk-free rates (innskuddsrente)
n = 4;
priceindex_outer_southern.rf = zeros(44,1);
priceindex_outer_southern.rf =
[repmat(Rf.rf(1),n,1);repmat(Rf.rf(2),n,1);repmat(Rf.rf(3),n,1);repmat(Rf.rf(4),n,1);r
epmat(Rf.rf(5),n,1);repmat(Rf.rf(6),n,1);repmat(Rf.rf(7),n,1);repmat(Rf.rf(8),n,1);rep
mat(Rf.rf(9),n,1);repmat(Rf.rf(10),n,1);repmat(Rf.rf(11),n,1)];
priceindex_outer_southern.rf = priceindex_outer_southern.rf./4;

% 6. COMPUTATION OF MEAN PRICES FOR HOUSES AND PRICE INDEX BASED ON MEAN PRICES
% Calculate mean prices for houses
priceindex_outer_southern.mean_prices = zeros(height(priceindex_outer_southern), end);
for i = 1:length(priceindex_outer_southern.Q_order)
    priceindex_outer_southern.mean_prices(i) =
mean(price_outer_southern.estimated_price(find(price_outer_southern.Q ==
priceindex_outer_southern.Q_order(i))));
end

% Calculate price index for houses based on mean prices
priceindex_outer_southern.index_values = zeros(height(priceindex_outer_southern),
end);
for i = 1:length(priceindex_outer_southern.index_values)
    priceindex_outer_southern.index_values(i) =
priceindex_outer_southern.mean_prices(i)/priceindex_outer_southern.mean_prices(1);
end
```

```
% 7. COMPUTATION OF QUARTERLY RETURNS ON ASSET/HOUSE FROM PERIOD TO PERIOD, NO COSTS
% Compute quarterly returns for each unique ID
price_outer_southern.Q_ret_no_costs =
(price_outer_southern.estimated_price./price_outer_southern.lagged_estimated_price)-1;
price_outer_southern.Q_ret_no_costs(1:44:end) = NaN;

% Compute mean quarterly returns for houses (mean of all IDs per quarter) (no costs)
priceindex_outer_southern.Q_ret_no_costs = zeros(height(priceindex_outer_southern),
end);

for i = 1:height(priceindex_outer_southern)
    priceindex_outer_southern.Q_ret_no_costs(i) =
nanmean(price_outer_southern.Q_ret_no_costs(find(price_outer_southern.Q ==
priceindex_outer_southern.Q_order(i))));
end

% Compute mean of mean quarterly returns for houses (no costs)
mean_Q_ret_no_costs = nanmean(priceindex_outer_southern.Q_ret_no_costs);

% Compute excess quarterly returns for houses (no costs)
priceindex_outer_southern.excess_Q_ret_no_costs =
priceindex_outer_southern.Q_ret_no_costs-priceindex_outer_southern.rf;

% Compute mean excess quarterly returns for houses (no costs)
mean_excess_Q_ret_no_costs = nanmean(priceindex_outer_southern.excess_Q_ret_no_costs);

% Compute standard deviation for each unique ID
st_Q_ret_no_costs= zeros(length(C),1);
time(1,:) = clock;
for i = 1:length(C)
    if b(i)<b(length(C))
       st_Q_ret_no_costs(i) =
std(price_outer_southern.Q_ret_no_costs(((b(i)+1):(b(i+1)-1))));
    else
       st_Q_ret_no_costs(i) =
std(price_outer_southern.Q_ret_no_costs((b(length(C))+1):height(price_outer_southern))
);
    end
time(i+1,:) = clock;
fprintf('\nTo complete %.f/%.f iterations (unique IDs) it took %.2f seconds\n', i,
length(C), etime(time(i+1,:), time(1,:)));
end

% Standard deviation of all quarterly returns
std_Q_ret_no_costs = mean(st_Q_ret_no_costs);

% Import stats for 'Q_ret_no_costs' in summary table
sum_stats_outer_southern.House_Indices(1) = 'Q_ret_no_costs';
sum_stats_outer_southern.Return(1)        = mean_Q_ret_no_costs;
sum_stats_outer_southern.std_dev(1)       = std_Q_ret_no_costs;
sum_stats_outer_southern.Excess_return(1) = mean_excess_Q_ret_no_costs;
sum_stats_outer_southern.Sharpe_ratio(1) =
sum_stats_outer_southern.Excess_return(1)/sum_stats_outer_southern.std_dev(1);
sum_stats_outer_southern.Skewness(1)      =
skewness(priceindex_outer_southern.Q_ret_no_costs(2:44));
sum_stats_outer_southern.Kurtosis(1)      =
kurtosis(priceindex_outer_southern.Q_ret_no_costs(2:44));

% COMPUTE CUMULATIVE ROA(NC)
first_obs_price = price_outer_southern.estimated_price(1:44:end);
price_outer_southern.first_obs_price_repeated = repelem(first_obs_price, 44);

price_outer_southern.cum_Q_ret_no_costs =
(price_outer_southern.estimated_price./price_outer_southern.first_obs_price_repeated)-
1;
price_outer_southern.cum_Q_ret_no_costs(1:44:end) = NaN;

priceindex_outer_southern.cum_Q_ret_no_costs =
zeros(height(priceindex_outer_southern),end);
% Compute mean cumulative ROA(NC)
for i = 1:height(priceindex_outer_southern)
    priceindex_outer_southern.cum_Q_ret_no_costs(i) =
nanmean(price_outer_southern.cum_Q_ret_no_costs(find(price_outer_southern.Q ==
priceindex_outer_southern.Q_order(i))));
```

```
end

% 8. COMPUTATION OF QUARTERLY RETURNS ON ASSET/HOUSE FROM PERIOD TO PERIOD (BUY/SELL),
INCL TOTAL COSTS
% Compute quarterly returns for each unique ID
price_outer_southern.Q_ret_tot_costs = (((price_outer_southern.estimated_price -
price_outer_southern.t_cost_s - price_outer_southern.lagged_t_cost -
price_outer_southern.lagged_estimated_price).*(1-price_outer_southern.tax_rates)) -
price_outer_southern.living_exp +
price_outer_southern.implicit_rent)./(price_outer_southern.lagged_estimated_price);
price_outer_southern.Q_ret_tot_costs(1:44:end) = NaN;

% Compute mean quarterly returns for houses (mean of all IDs per quarter) (inc tot
costs)
priceindex_outer_southern.Q_ret_tot_costs = zeros(height(priceindex_outer_southern),
end);

for i = 1:length(priceindex_outer_southern.mean_prices)
    priceindex_outer_southern.Q_ret_tot_costs(i) =
mean(price_outer_southern.Q_ret_tot_costs(find(price_outer_southern.Q ==
priceindex_outer_southern.Q_order(i))));
end

% Compute mean of mean quarterly returns for houses (inc tot costs)
mean_Q_ret_tot_costs = nanmean(priceindex_outer_southern.Q_ret_tot_costs);

% Compute excess quarterly returns for houses (inc tot costs)
priceindex_outer_southern.excess_Q_ret_tot_costs =
priceindex_outer_southern.Q_ret_tot_costs-priceindex_outer_southern.rf;

% Compute mean excess quarterly returns for houses (inc tot costs)
mean_excess_Q_ret_tot_costs =
nanmean(priceindex_outer_southern.excess_Q_ret_tot_costs);

% Compute standard deviation for each unique IDs returns
st_Q_ret_tot_costs= zeros(length(C),1);
time(1,:) = clock;
for i = 1:length(C)
    if b(i)<b(length(C))
       st_Q_ret_tot_costs(i) =
std(price_outer_southern.Q_ret_tot_costs(((b(i)+1):(b(i+1)-1))));
    else
       st_Q_ret_tot_costs(i) =
std(price_outer_southern.Q_ret_tot_costs((b(length(C))+1):height(price_outer_southern)
));
    end
time(i+1,:) = clock;
fprintf('\nTo complete %.f/%.f iterations (unique IDs) it took %.2f seconds\n', i,
length(C), etime(time(i+1,:), time(1,:)));
end

% Standard deviation of all quarterly returns
std_Q_ret_tot_costs = mean(st_Q_ret_tot_costs);

% Import stats for 'Q_ret_tot_costs' in summary table
sum_stats_outer_southern.House_Indices(2) = 'Q_ret_tot_costs';
sum_stats_outer_southern.Return(2)        = mean_Q_ret_tot_costs;
sum_stats_outer_southern.std_dev(2)       = std_Q_ret_tot_costs;
sum_stats_outer_southern.Excess_return(2) = mean_excess_Q_ret_tot_costs;
sum_stats_outer_southern.Sharpe_ratio(2) =
sum_stats_outer_southern.Excess_return(2)/sum_stats_outer_southern.std_dev(2);
sum_stats_outer_southern.Skewness(2)      =
skewness(priceindex_outer_southern.Q_ret_tot_costs(2:44));
sum_stats_outer_southern.Kurtosis(2)      =
kurtosis(priceindex_outer_southern.Q_ret_tot_costs(2:44));

% COMPUTE CUMULATIVE ROA (TC)
taxes_TC = (price_outer_southern.estimated_price - price_outer_southern.t_cost_s -
price_outer_southern.lagged_t_cost -
price_outer_southern.lagged_estimated_price).*price_outer_southern.tax_rates;
taxes_TC(1:44:end) = NaN;

net_costs_TC = (price_outer_southern.living_exp + price_outer_southern.lagged_t_cost +
price_outer_southern.t_cost_s + taxes_TC - price_outer_southern.implicit_rent);
```

```
net_costs_TC(1:44:end) = NaN;

% Compute cumulative costs
cumcosts_TC = zeros(height(price_outer_southern), end);
time(1,:) = clock;
for i = 1:length(C)
    if b(i)<b(length(C))
       cumcosts_TC((b(i)):(b(i+1)-1)) = cumsum(net_costs_TC((b(i)):(b(i+1)-
1)),'omitnan');
    else
       cumcosts_TC(b(length(C)):height(price_outer_southern)) =
cumsum(net_costs_TC(b(length(C)):height(price_outer_southern)),'omitnan');
    end
time(i+1,:) = clock;
fprintf('\nTo complete %.f/%.f iterations (unique IDs) it took %.2f seconds\n', i,
length(C), etime(time(i+1,:), time(1,:)));
end

price_outer_southern.cum_Q_ret_tot_costs = (price_outer_southern.estimated_price -
cumcosts_TC -
price_outer_southern.first_obs_price_repeated)./price_outer_southern.first_obs_price_r
epeated;
price_outer_southern.cum_Q_ret_tot_costs(1:44:end) = NaN;

priceindex_outer_southern.cum_Q_ret_tot_costs =
zeros(height(priceindex_outer_southern),end);
% Compute mean cumulative returns for houses (mean of all IDs per quarter) (no costs)
for i = 1:height(priceindex_outer_southern)
    priceindex_outer_southern.cum_Q_ret_tot_costs(i) =
nanmean(price_outer_southern.cum_Q_ret_tot_costs(find(price_outer_southern.Q ==
priceindex_outer_southern.Q_order(i))));
end

all_lost = priceindex_outer_southern.cum_Q_ret_tot_costs < -1;
priceindex_outer_southern.cum_Q_ret_tot_costs(all_lost) = -1;

% 9. COMPUTATION OF QUARTERLY RETURNS ON ASSET/HOUSE FROM PERIOD TO PERIOD , INCL
SMOOTHED COSTS
% Locate buyer and sellers cost when buying in 2010Q1 and selling in 2020Q4
% Find buyer and seller cost per quarter for sample period
cost_2010Q1 = price_outer_southern.t_cost(1:44:end);
cost_2010Q1 = cost_2010Q1/44;
price_outer_southern.cost_2010Q1 = repelem(cost_2010Q1, 44);

cost_2020Q4 = price_outer_southern.t_cost_s(44:44:end);
cost_2020Q4 = cost_2020Q4/44;
price_outer_southern.cost_2020Q4 = repelem(cost_2020Q4, 44);

% Compute quarterly returns given that you buy in 2010Q1 and sell in 2020Q4
% and spred the relevant costs on the sample period.
price_outer_southern.Q_ret_smooth_costs = (price_outer_southern.estimated_price -
price_outer_southern.cost_2010Q1 - price_outer_southern.cost_2020Q4 -
price_outer_southern.living_exp - price_outer_southern.lagged_estimated_price +
price_outer_southern.implicit_rent)./price_outer_southern.lagged_estimated_price;
price_outer_southern.Q_ret_smooth_costs(1:44:end) = NaN;

% Compute mean quarterly returns for houses (mean of all IDs per quarter) (inc smooth
costs)
priceindex_outer_southern.Q_ret_smooth_costs =
zeros(height(priceindex_outer_southern), end);

for i = 1:length(priceindex_outer_southern.mean_prices)
    priceindex_outer_southern.Q_ret_smooth_costs(i) =
mean(price_outer_southern.Q_ret_smooth_costs(find(price_outer_southern.Q ==
priceindex_outer_southern.Q_order(i))));
end

% Compute mean of mean quarterly returns for houses (inc smooth costs)
mean_Q_ret_smooth_costs = nanmean(priceindex_outer_southern.Q_ret_smooth_costs);

% Compute excess quarterly returns for houses (inc smooth costs)
priceindex_outer_southern.excess_Q_ret_smooth_costs =
priceindex_outer_southern.Q_ret_smooth_costs - priceindex_outer_southern.rf;
```

```
% Compute mean excess quarterly returns for houses (inc smooth costs)
mean_excess_Q_ret_smooth_costs =
nanmean(priceindex_outer_southern.excess_Q_ret_smooth_costs);

% Compute standard deviation for each unique IDs returns
st_Q_ret_smooth_costs= zeros(length(C),1);
time(1,:) = clock;
for i = 1:length(C)
    if b(i)<b(length(C))
       st_Q_ret_smooth_costs(i) =
std(price_outer_southern.Q_ret_smooth_costs(((b(i)+1):(b(i+1)-1))));
    else
       st_Q_ret_smooth_costs(i) =
std(price_outer_southern.Q_ret_smooth_costs((b(length(C))+1):height(price_outer_southe
rn)));
    end
time(i+1,:) = clock;
fprintf('\nTo complete %.f/%.f iterations (unique IDs) it took %.2f seconds\n', i,
length(C), etime(time(i+1,:), time(1,:)));
end

% Standard deviation of all quarterly returns
std_Q_ret_smooth_costs = mean(st_Q_ret_smooth_costs);

% Import stats for 'Q_ret_smooth_costs' in summary table
sum_stats_outer_southern.House_Indices(3) = 'Q_ret_smooth_costs';
sum_stats_outer_southern.Return(3)        = mean_Q_ret_smooth_costs;
sum_stats_outer_southern.std_dev(3)       = std_Q_ret_smooth_costs;
sum_stats_outer_southern.Excess_return(3) = mean_excess_Q_ret_smooth_costs;
sum_stats_outer_southern.Sharpe_ratio(3) =
sum_stats_outer_southern.Excess_return(3)/sum_stats_outer_southern.std_dev(3);
sum_stats_outer_southern.Skewness(3)      =
skewness(priceindex_outer_southern.Q_ret_smooth_costs(2:44));
sum_stats_outer_southern.Kurtosis(3)      =
kurtosis(priceindex_outer_southern.Q_ret_smooth_costs(2:44));

% COMPUTE CUMULATIVE ROA(SC)
net_costs_SC = (price_outer_southern.living_exp + price_outer_southern.cost_2010Q1 +
price_outer_southern.cost_2020Q4 - price_outer_southern.implicit_rent);
net_costs_SC(1:44:end) = NaN;

cumcosts_SC = zeros(height(price_outer_southern), end);
time(1,:) = clock;
for i = 1:length(C)
    if b(i)<b(length(C))
       cumcosts_SC((b(i)):(b(i+1)-1)) = cumsum(net_costs_SC((b(i)):(b(i+1)-
1)),'omitnan');
    else
       cumcosts_SC(b(length(C)):height(price_outer_southern)) =
cumsum(net_costs_SC(b(length(C)):height(price_outer_southern)),'omitnan');
    end
time(i+1,:) = clock;
fprintf('\nTo complete %.f/%.f iterations (unique IDs) it took %.2f seconds\n', i,
length(C), etime(time(i+1,:), time(1,:)));
end

price_outer_southern.cum_Q_ret_smooth_costs = (price_outer_southern.estimated_price -
cumcosts_SC -
price_outer_southern.first_obs_price_repeated)./price_outer_southern.first_obs_price_r
epeated;
price_outer_southern.cum_Q_ret_smooth_costs(1:44:end) = NaN;

% Compute mean cumulative returns for houses (mean of all IDs per quarter) (no costs)
priceindex_outer_southern.cum_Q_ret_smooth_costs =
zeros(height(priceindex_outer_southern),end);
for i = 1:height(priceindex_outer_southern)
    priceindex_outer_southern.cum_Q_ret_smooth_costs(i) =
nanmean(price_outer_southern.cum_Q_ret_smooth_costs(find(price_outer_southern.Q ==
priceindex_outer_southern.Q_order(i))));
end

% Identify average loan-to-value ratio for housing investments
l = ([0.6957, 0.6957, 0.6957, 0.6957, 0.71, 0.76, 0.67, 0.66, 0.68, 0.70, 0.69])';
n = 4;
```

```
ltv_ratio_order =
[repmat(l(1),n,1);repmat(l(2),n,1);repmat(l(3),n,1);repmat(l(4),n,1);repmat(l(5),n,1);
repmat(l(6),n,1);repmat(l(7),n,1);repmat(l(8),n,1);repmat(l(9),n,1);repmat(l(10),n,1);
repmat(l(11),n,1)];

% Allocate correct ltv-rates to each quarter in price_houses set
price_outer_southern.ltvr        = repmat(ltv_ratio_order,(length(C)),1);
price_outer_southern.lagged_ltvr = lagmatrix(price_outer_southern.ltvr,1);

% Allocate ltv ratio as of 2010Q1 to each quarter in price_houses set
price_outer_southern.ltvr_2010Q1(:) = price_outer_southern.ltvr(1);

% 10. WE DONT USE THE RETURN SERIES COMPUTED IN SECTION 10
% Compute quarterly equity returns for each unique ID (no costs)
price_outer_southern.Q_eq_ret_no_costs = ((price_outer_southern.estimated_price -
price_outer_southern.lagged_estimated_price)./(price_outer_southern.lagged_estimated_p
rice.*(1-price_outer_southern.lagged_ltvr)));
price_outer_southern.Q_eq_ret_no_costs(1:44:end) = NaN;

% Compute mean quarterly equity returns for houses (mean of all IDs per quarter) (no
costs)
priceindex_outer_southern.Q_eq_ret_no_costs = zeros(height(priceindex_outer_southern),
end);

for i = 1:height(priceindex_outer_southern)
    priceindex_outer_southern.Q_eq_ret_no_costs(i) =
mean(price_outer_southern.Q_eq_ret_no_costs(find(price_outer_southern.Q ==
priceindex_outer_southern.Q_order(i))));
end

% Compute mean of mean quarterly equity returns for houses (no costs)
mean_Q_eq_ret_no_costs = nanmean(priceindex_outer_southern.Q_eq_ret_no_costs);

% Compute excess quarterly equity returns for houses (no costs)
priceindex_outer_southern.excess_Q_eq_ret_no_costs =
priceindex_outer_southern.Q_eq_ret_no_costs - priceindex_outer_southern.rf;

% Compute mean excess quarterly equity returns for houses (no costs)
mean_excess_Q_eq_ret_no_costs =
nanmean(priceindex_outer_southern.excess_Q_eq_ret_no_costs);

% Compute standard deviation for each unique IDs returns
st_Q_eq_ret_no_costs= zeros(length(C),1);
time(1,:) = clock;
for i = 1:length(C)
    if b(i)<b(length(C))
       st_Q_eq_ret_no_costs(i) =
std(price_outer_southern.Q_eq_ret_no_costs(((b(i)+1):(b(i+1)-1))));
    else
       st_Q_eq_ret_no_costs(i) =
std(price_outer_southern.Q_eq_ret_no_costs((b(length(C))+1):height(price_outer_souther
n)));
    end
time(i+1,:) = clock;
fprintf('\nTo complete %.f/%.f iterations (unique IDs) it took %.2f seconds\n', i,
length(C), etime(time(i+1,:), time(1,:)));
end

% Standard deviation of all quarterly equity returns
std_Q_eq_ret_no_costs = mean(st_Q_eq_ret_no_costs);

% Import stats for 'Q_eq_ret_no_costs' in summary table
sum_stats_outer_southern.House_Indices(4) = 'Q_eq_ret_no_costs';
sum_stats_outer_southern.Return(4)        = mean_Q_eq_ret_no_costs;
sum_stats_outer_southern.std_dev(4)       = std_Q_eq_ret_no_costs;
sum_stats_outer_southern.Excess_return(4) = mean_excess_Q_eq_ret_no_costs;
sum_stats_outer_southern.Sharpe_ratio(4) =
sum_stats_outer_southern.Excess_return(4)/sum_stats_outer_southern.std_dev(4);
sum_stats_outer_southern.Skewness(4)      =
skewness(priceindex_outer_southern.Q_eq_ret_no_costs(2:44));
sum_stats_outer_southern.Kurtosis(4)      =
kurtosis(priceindex_outer_southern.Q_eq_ret_no_costs(2:44));

% COMPUTE CUMULATIVE ROE (NC)
```

```
price_outer_southern.cum_Q_eq_ret_no_costs = ((price_outer_southern.estimated_price -
price_outer_southern.first_obs_price_repeated)./(price_outer_southern.first_obs_price_
repeated.*(1-price_outer_southern.ltvr_2010Q1)));
price_outer_southern.cum_Q_eq_ret_no_costs(1:44:end) = NaN;

% Compute mean cumulative returns for houses (mean of all IDs per quarter) (no costs)
priceindex_outer_southern.cum_Q_eq_ret_no_costs =
zeros(height(priceindex_outer_southern),end);
for i = 1:height(priceindex_outer_southern)
    priceindex_outer_southern.cum_Q_eq_ret_no_costs(i) =
nanmean(price_outer_southern.cum_Q_eq_ret_no_costs(find(price_outer_southern.Q ==
priceindex_outer_southern.Q_order(i))));
end

% 11. WE DONT USE THE RETURN SERIES COMPUTED IN SECTION 11
% Compute quarterly equity returns for each unique ID (inc tot costs)
price_outer_southern.Q_eq_ret_tot_costs = (((price_outer_southern.estimated_price -
price_outer_southern.t_cost_s - price_outer_southern.lagged_t_cost -
price_outer_southern.lagged_estimated_price).*(1-price_outer_southern.tax_rates)) -
price_outer_southern.living_exp +
price_outer_southern.implicit_rent)./(price_outer_southern.lagged_estimated_price.*(1-
price_outer_southern.lagged_ltvr));
price_outer_southern.Q_eq_ret_tot_costs(1:44:end) = NaN;

% Compute mean quarterly equity returns for houses (mean of all IDs per quarter) (inc
tot costs)
priceindex_outer_southern.Q_eq_ret_tot_costs =
zeros(height(priceindex_outer_southern), end);

for i = 1:length(priceindex_outer_southern.mean_prices)
    priceindex_outer_southern.Q_eq_ret_tot_costs(i) =
mean(price_outer_southern.Q_eq_ret_tot_costs(find(price_outer_southern.Q ==
priceindex_outer_southern.Q_order(i))));
end

% Compute mean of mean quarterly equity returns for houses (inc tot costs)
mean_Q_eq_ret_tot_costs = nanmean(priceindex_outer_southern.Q_eq_ret_tot_costs);

% Compute excess quarterly equity returns for houses (inc tot costs)
priceindex_outer_southern.excess_Q_eq_ret_tot_costs =
priceindex_outer_southern.Q_eq_ret_tot_costs - priceindex_outer_southern.rf;

% Compute mean excess quarterly equity returns for houses (inc tot costs)
mean_excess_Q_eq_ret_tot_costs =
nanmean(priceindex_outer_southern.excess_Q_eq_ret_tot_costs);

% Compute standard deviation for each unique IDs returns
st_Q_eq_ret_tot_costs= zeros(length(C),1);
time(1,:) = clock;
for i = 1:length(C)
    if b(i)<b(length(C))
       st_Q_eq_ret_tot_costs(i) =
std(price_outer_southern.Q_eq_ret_tot_costs(((b(i)+1):(b(i+1)-1))));
    else
       st_Q_eq_ret_tot_costs(i) =
std(price_outer_southern.Q_eq_ret_tot_costs((b(length(C))+1):height(price_outer_southe
rn)));
    end
time(i+1,:) = clock;
fprintf('\nTo complete %.f/%.f iterations (unique IDs) it took %.2f seconds\n', i,
length(C), etime(time(i+1,:), time(1,:)));
end

% Standard deviation of all quarterly equity returns
std_Q_eq_ret_tot_costs = mean(st_Q_eq_ret_tot_costs);

% Import stats for 'Q_eq_ret_tot_costs' in summary table
sum_stats_outer_southern.House_Indices(5) = 'Q_eq_ret_tot_costs';
sum_stats_outer_southern.Return(5)        = mean_Q_eq_ret_tot_costs;
sum_stats_outer_southern.std_dev(5)       = std_Q_eq_ret_tot_costs;
sum_stats_outer_southern.Excess_return(5) = mean_excess_Q_eq_ret_tot_costs;
sum_stats_outer_southern.Sharpe_ratio(5) =
sum_stats_outer_southern.Excess_return(5)/sum_stats_outer_southern.std_dev(5);
```

```
sum_stats_outer_southern.Skewness(5)      =
skewness(priceindex_outer_southern.Q_eq_ret_tot_costs(2:44));
sum_stats_outer_southern.Kurtosis(5)      =
kurtosis(priceindex_outer_southern.Q_eq_ret_tot_costs(2:44));

% COMPUTE CUMULATIVE RETURNS BASED ON QUARTERLY EQUITY RETURNS FROM PRICE INDEX (INC
TOT COSTS)
priceindex_outer_southern.cum_Q_eq_ret_tot_costs =
zeros(height(priceindex_outer_southern),end);
priceindex_outer_southern.cum_Q_eq_ret_tot_costs =
cumprod((1+priceindex_outer_southern.Q_eq_ret_tot_costs),'omitnan') -1;

% 12. COMPUTATION OF QUARTERLY EQUITY RETURNS ON ASSET/HOUSE FROM PERIOD TO PERIOD ,
INCL SMOOTHED COSTS
% Compute quarterly EQUITY returns given that you buy in 2010Q1 and sell in 2020Q4
% and spread the relevant costs on the sample period.
price_outer_southern.Q_eq_ret_smooth_costs = (price_outer_southern.estimated_price -
price_outer_southern.cost_2010Q1 - price_outer_southern.cost_2020Q4 -
price_outer_southern.living_exp - price_outer_southern.lagged_estimated_price +
price_outer_southern.implicit_rent)./(price_outer_southern.lagged_estimated_price.*(1-
price_outer_southern.ltvr_2010Q1));
price_outer_southern.Q_eq_ret_smooth_costs(1:44:end) = NaN;

% Compute mean quarterly equity returns for houses (mean of all IDs per quarter) (inc
smooth costs)
priceindex_outer_southern.Q_eq_ret_smooth_costs =
zeros(height(priceindex_outer_southern), end);

for i = 1:length(priceindex_outer_southern.mean_prices)
    priceindex_outer_southern.Q_eq_ret_smooth_costs(i) =
mean(price_outer_southern.Q_eq_ret_smooth_costs(find(price_outer_southern.Q ==
priceindex_outer_southern.Q_order(i))));
end

% Compute mean of mean quarterly equity returns for houses (inc smooth costs)
mean_Q_eq_ret_smooth_costs = nanmean(priceindex_outer_southern.Q_eq_ret_smooth_costs);

% Compute excess quarterly equity returns for houses (inc smooth costs)
priceindex_outer_southern.excess_Q_eq_ret_smooth_costs =
priceindex_outer_southern.Q_eq_ret_smooth_costs - priceindex_outer_southern.rf;

% Compute mean excess quarterly equity returns for houses (inc smooth costs)
mean_excess_Q_eq_ret_smooth_costs =
nanmean(priceindex_outer_southern.excess_Q_eq_ret_smooth_costs);

% Compute standard deviation for each unique IDs returns
st_Q_eq_ret_smooth_costs= zeros(length(C),1);
time(1,:) = clock;
for i = 1:length(C)
    if b(i)<b(length(C))
       st_Q_eq_ret_smooth_costs(i) =
std(price_outer_southern.Q_eq_ret_smooth_costs(((b(i)+1):(b(i+1)-1))));
    else
       st_Q_eq_ret_smooth_costs(i) =
std(price_outer_southern.Q_eq_ret_smooth_costs((b(length(C))+1):height(price_outer_sou
thern)));
    end
time(i+1,:) = clock;
fprintf('\nTo complete %.f/%.f iterations (unique IDs) it took %.2f seconds\n', i,
length(C), etime(time(i+1,:), time(1,:)));
end

% Standard deviation of all quarterly equity returns
std_Q_eq_ret_smooth_costs = mean(st_Q_eq_ret_smooth_costs);

% Import stats for 'Q_eq_ret_smooth_costs' in summary table
sum_stats_outer_southern.House_Indices(6) = 'Q_eq_ret_smooth_costs';
sum_stats_outer_southern.Return(6)        = mean_Q_eq_ret_smooth_costs;
sum_stats_outer_southern.std_dev(6)       = std_Q_eq_ret_smooth_costs;
sum_stats_outer_southern.Excess_return(6) = mean_excess_Q_eq_ret_smooth_costs;
sum_stats_outer_southern.Sharpe_ratio(6) =
sum_stats_outer_southern.Excess_return(6)/sum_stats_outer_southern.std_dev(6);
sum_stats_outer_southern.Skewness(6)      =
skewness(priceindex_outer_southern.Q_eq_ret_smooth_costs(2:44));
```

```
sum_stats_outer_southern.Kurtosis(6)      =
kurtosis(priceindex_outer_southern.Q_eq_ret_smooth_costs(2:44));

% COMPUTE CUMULATIVE ROE(SC)
price_outer_southern.cum_Q_eq_ret_smooth_costs = (price_outer_southern.estimated_price
- cumcosts_SC -
price_outer_southern.first_obs_price_repeated)./(price_outer_southern.first_obs_price_
repeated.*(1-price_outer_southern.ltvr_2010Q1));
price_outer_southern.cum_Q_eq_ret_smooth_costs(1:44:end) = NaN;

% Compute mean cumulative returns for houses (mean of all IDs per quarter) (smooth
costs)
priceindex_outer_southern.cum_Q_eq_ret_smooth_costs =
zeros(height(priceindex_outer_southern),end);
for i = 1:height(priceindex_outer_southern)
    priceindex_outer_southern.cum_Q_eq_ret_smooth_costs(i) =
nanmean(price_outer_southern.cum_Q_eq_ret_smooth_costs(find(price_outer_southern.Q ==
priceindex_outer_southern.Q_order(i))));
end

% MAKE PLOT OF THE RELEVANT INDICES TOGHETER, MAKE MULTIPLE PLOTS FOR COMPARISON
REASONS
outer_southern_price_changes = priceindex_outer_southern(:,[1 5 8 11 14 17 20]);

outer_southern_price_indices = priceindex_outer_southern(:,[1 7 10 13 16 19 22]);
outer_southern_price_indices.cum_Q_ret_no_costs(1) = 0;
outer_southern_price_indices.cum_Q_ret_tot_costs(1) = 0;
outer_southern_price_indices.cum_Q_ret_smooth_costs(1) = 0;
outer_southern_price_indices.cum_Q_eq_ret_no_costs(1) = 0;
outer_southern_price_indices.cum_Q_eq_ret_tot_costs(1) = 0;
outer_southern_price_indices.cum_Q_eq_ret_smooth_costs(1) = 0;

outer_southern_price_indices.cum_Q_ret_no_costs        =
outer_southern_price_indices.cum_Q_ret_no_costs + 1;
outer_southern_price_indices.cum_Q_ret_tot_costs       =
outer_southern_price_indices.cum_Q_ret_tot_costs + 1;
outer_southern_price_indices.cum_Q_ret_smooth_costs    =
outer_southern_price_indices.cum_Q_ret_smooth_costs + 1;
outer_southern_price_indices.cum_Q_eq_ret_no_costs     =
outer_southern_price_indices.cum_Q_eq_ret_no_costs + 1;
outer_southern_price_indices.cum_Q_eq_ret_tot_costs    =
outer_southern_price_indices.cum_Q_eq_ret_tot_costs + 1;
outer_southern_price_indices.cum_Q_eq_ret_smooth_costs =
outer_southern_price_indices.cum_Q_eq_ret_smooth_costs + 1;

clearvars -except Property_Data summary_table Rf Rent_Data...
                  house_set hedonic_houses housing_model...
                  price_houses uniq_house_set avg_cost_rates_houses...
                  priceindex_houses sum_stats_houses houseprice_changes...
                  houseprice_indices...
                  smallhouse_set hedonic_smallhouses smallhouse_model...
                  price_smallhouses uniq_smallhouse_set avg_cost_rates_smallhouses...
                  priceindex_smallhouses sum_stats_smallhouses...
                  smallhouse_price_changes smallhouse_price_indices...
                  apartment_set hedonic_apartments apartment_model price_apartments...
                  uniq_apartment_set avg_cost_rates_apartments
priceindex_apartments...
                  sum_stats_apartments apartment_price_changes...
                  apartment_price_indices...
                  inner_eastern_set hedonic_inner_eastern inner_eastern_model...
                  price_inner_eastern uniq_inner_eastern_set
avg_cost_rates_inner_eastern...
                  priceindex_inner_eastern sum_stats_inner_eastern
inner_eastern_price_changes...
                  inner_eastern_price_indices...
                  inner_western_set hedonic_inner_western inner_western_model...
                  price_inner_western uniq_inner_western_set
avg_cost_rates_inner_western...
                  priceindex_inner_western sum_stats_inner_western
inner_western_price_changes...
                  inner_western_price_indices...
                  outer_eastern_set hedonic_outer_eastern outer_eastern_model...
                  price_outer_eastern uniq_outer_eastern_set
avg_cost_rates_outer_eastern...
```

```
                priceindex_outer_eastern sum_stats_outer_eastern
outer_eastern_price_changes...
                outer_eastern_price_indices...
                outer_western_set hedonic_outer_western outer_western_model...
                price_outer_western uniq_outer_western_set
avg_cost_rates_outer_western...
                priceindex_outer_western sum_stats_outer_western
outer_western_price_changes...
                outer_western_price_indices...
                outer_southern_set hedonic_outer_southern outer_southern_model...
                price_outer_southern uniq_outer_southern_set
avg_cost_rates_outer_southern...
                priceindex_outer_southern sum_stats_outer_southern
outer_southern_price_changes...
                outer_southern_price_indices...
                property_price_changes property_price_indices hedonic_total...
                ROA_NC_stats ROE_NC_stats ROA_TC_stats ROE_TC_stats...
                ROA_SC_stats ROE_SC_stats sum_stats_OSEFX OSEFX...
                ROA_NC_sharpe_test ROA_TC_sharpe_test ROA_SC_sharpe_test...
                ROE_SC_sharpe_test

%% Returns and indices on whole dataset (Oslo properties) based on previous models
% Get ratios of how many dwellings of each type
house_ratio      = height(house_set)/height(Property_Data);
smallhouse_ratio = height(smallhouse_set)/height(Property_Data);
apartment_ratio  = height(apartment_set)/height(Property_Data);

% How much each dwelling type contributes to overall property price changes
house_contribution_changes      =
house_ratio.*table2array(houseprice_changes(:,2:end));
smallhouse_contribution_changes =
smallhouse_ratio.*table2array(smallhouse_price_changes(:,2:end));
apartment_contribution_changes  =
apartment_ratio.*table2array(apartment_price_changes(:,2:end));

% Compute quarterly price change for dwellings overall
price_change_properties                          = zeros(44,7);
price_change_properties                          =
array2table(price_change_properties);
price_change_properties.Properties.VariableNames =
apartment_price_changes.Properties.VariableNames;
price_change_properties.Q_order                  =
categorical(price_change_properties.Q_order);
price_change_properties(:,1)                     = apartment_price_changes(:,1);

price_change_properties(:,2:end) = array2table(house_contribution_changes +
smallhouse_contribution_changes + apartment_contribution_changes);
property_price_changes = price_change_properties;

% How much each dwelling type contributes to overall property price indices
house_contribution_indices      =
house_ratio.*table2array(houseprice_indices(:,2:end));
smallhouse_contribution_indices =
smallhouse_ratio.*table2array(smallhouse_price_indices(:,2:end));
apartment_contribution_indices  =
apartment_ratio.*table2array(apartment_price_indices(:,2:end));

% Compute indices based on quarterly price changes for dwellings overall
price_indices_properties                          = zeros(44,7);
price_indices_properties                          =
array2table(price_indices_properties);
price_indices_properties.Properties.VariableNames =
apartment_price_indices.Properties.VariableNames;
price_indices_properties.Q_order                  =
categorical(price_indices_properties.Q_order);
price_indices_properties(:,1)                     = apartment_price_indices(:,1);

price_indices_properties(:,2:end) = array2table(house_contribution_indices +
smallhouse_contribution_indices + apartment_contribution_indices);
property_price_indices = price_indices_properties;

%% Get stats from hedonic regression models:
writetable(apartment_model.Coefficients,      'apartment_model.xlsx','Sheet',1)
writetable(housing_model.Coefficients,        'housing_model.xlsx','Sheet',1)
```

```
writetable(smallhouse_model.Coefficients,     'smallhouse_model.xlsx','Sheet',1)
writetable(inner_eastern_model.Coefficients,  'inner_eastern_model.xlsx','Sheet',1)
writetable(inner_western_model.Coefficients,  'inner_western_model.xlsx','Sheet',1)
writetable(outer_eastern_model.Coefficients,  'outer_eastern_model.xlsx','Sheet',1)
writetable(outer_western_model.Coefficients,  'outer_western_model.xlsx','Sheet',1)
writetable(outer_southern_model.Coefficients, 'outer_southern_model.xlsx','Sheet',1)

%% Summary stats from all hedonic models we use
hedonic_total = zeros(8,9);
hedonic_total = array2table(hedonic_total);
hedonic_total.Properties.VariableNames = {'key_stats', 'Apartments', 'Houses',
'Small_Houses', 'Inner_Eastern', 'Inner_Western', 'Outer_Eastern', 'Outer_Western',
'Outer_Southern'};
hedonic_total.key_stats = categorical(hedonic_total.key_stats);
hedonic_total.key_stats = hedonic_apartments.key_stats;

hedonic_total.Apartments(1) = apartment_model.NumObservations;
hedonic_total.Apartments(2) = apartment_model.NumCoefficients;
hedonic_total.Apartments(3) = apartment_model.Rsquared.Ordinary;
hedonic_total.Apartments(4) = apartment_model.Rsquared.Adjusted;
hedonic_total.Apartments(5) = apartment_model.RMSE;

hedonic_total.Houses(1) = housing_model.NumObservations;
hedonic_total.Houses(2) = housing_model.NumCoefficients;
hedonic_total.Houses(3) = housing_model.Rsquared.Ordinary;
hedonic_total.Houses(4) = housing_model.Rsquared.Adjusted;
hedonic_total.Houses(5) = housing_model.RMSE;

hedonic_total.Small_Houses(1) = smallhouse_model.NumObservations;
hedonic_total.Small_Houses(2) = smallhouse_model.NumCoefficients;
hedonic_total.Small_Houses(3) = smallhouse_model.Rsquared.Ordinary;
hedonic_total.Small_Houses(4) = smallhouse_model.Rsquared.Adjusted;
hedonic_total.Small_Houses(5) = smallhouse_model.RMSE;

hedonic_total.Inner_Eastern(1) = inner_eastern_model.NumObservations;
hedonic_total.Inner_Eastern(2) = inner_eastern_model.NumCoefficients;
hedonic_total.Inner_Eastern(3) = inner_eastern_model.Rsquared.Ordinary;
hedonic_total.Inner_Eastern(4) = inner_eastern_model.Rsquared.Adjusted;
hedonic_total.Inner_Eastern(5) = inner_eastern_model.RMSE;

hedonic_total.Inner_Western(1) = inner_western_model.NumObservations;
hedonic_total.Inner_Western(2) = inner_western_model.NumCoefficients;
hedonic_total.Inner_Western(3) = inner_western_model.Rsquared.Ordinary;
hedonic_total.Inner_Western(4) = inner_western_model.Rsquared.Adjusted;
hedonic_total.Inner_Western(5) = inner_western_model.RMSE;

hedonic_total.Outer_Eastern(1) = outer_eastern_model.NumObservations;
hedonic_total.Outer_Eastern(2) = outer_eastern_model.NumCoefficients;
hedonic_total.Outer_Eastern(3) = outer_eastern_model.Rsquared.Ordinary;
hedonic_total.Outer_Eastern(4) = outer_eastern_model.Rsquared.Adjusted;
hedonic_total.Outer_Eastern(5) = outer_eastern_model.RMSE;

hedonic_total.Outer_Western(1) = outer_western_model.NumObservations;
hedonic_total.Outer_Western(2) = outer_western_model.NumCoefficients;
hedonic_total.Outer_Western(3) = outer_western_model.Rsquared.Ordinary;
hedonic_total.Outer_Western(4) = outer_western_model.Rsquared.Adjusted;
hedonic_total.Outer_Western(5) = outer_western_model.RMSE;

hedonic_total.Outer_Southern(1) = outer_southern_model.NumObservations;
hedonic_total.Outer_Southern(2) = outer_southern_model.NumCoefficients;
hedonic_total.Outer_Southern(3) = outer_southern_model.Rsquared.Ordinary;
hedonic_total.Outer_Southern(4) = outer_southern_model.Rsquared.Adjusted;
hedonic_total.Outer_Southern(5) = outer_southern_model.RMSE;

clearvars -except Property_Data summary_table Rf Rent_Data...
                  house_set hedonic_houses housing_model...
                  price_houses uniq_house_set avg_cost_rates_houses...
                  priceindex_houses sum_stats_houses houseprice_changes...
                  houseprice_indices...
                  smallhouse_set hedonic_smallhouses smallhouse_model...
                  price_smallhouses uniq_smallhouse_set avg_cost_rates_smallhouses...
                  priceindex_smallhouses sum_stats_smallhouses...
                  smallhouse_price_changes smallhouse_price_indices...
                  apartment_set hedonic_apartments apartment_model price_apartments...
```

```
                  uniq_apartment_set avg_cost_rates_apartments
priceindex_apartments...
                  sum_stats_apartments apartment_price_changes...
                  apartment_price_indices...
                  inner_eastern_set hedonic_inner_eastern inner_eastern_model...
                  price_inner_eastern uniq_inner_eastern_set
avg_cost_rates_inner_eastern...
                  priceindex_inner_eastern sum_stats_inner_eastern
inner_eastern_price_changes...
                  inner_eastern_price_indices...
                  inner_western_set hedonic_inner_western inner_western_model...
                  price_inner_western uniq_inner_western_set
avg_cost_rates_inner_western...
                  priceindex_inner_western sum_stats_inner_western
inner_western_price_changes...
                  inner_western_price_indices...
                  outer_eastern_set hedonic_outer_eastern outer_eastern_model...
                  price_outer_eastern uniq_outer_eastern_set
avg_cost_rates_outer_eastern...
                  priceindex_outer_eastern sum_stats_outer_eastern
outer_eastern_price_changes...
                  outer_eastern_price_indices...
                  outer_western_set hedonic_outer_western outer_western_model...
                  price_outer_western uniq_outer_western_set
avg_cost_rates_outer_western...
                  priceindex_outer_western sum_stats_outer_western
outer_western_price_changes...
                  outer_western_price_indices...
                  outer_southern_set hedonic_outer_southern outer_southern_model...
                  price_outer_southern uniq_outer_southern_set
avg_cost_rates_outer_southern...
                  priceindex_outer_southern sum_stats_outer_southern
outer_southern_price_changes...
                  outer_southern_price_indices...
                  property_price_changes property_price_indices hedonic_total...
                  ROA_NC_stats ROE_NC_stats ROA_TC_stats ROE_TC_stats...
                  ROA_SC_stats ROE_SC_stats sum_stats_OSEFX OSEFX...
                  ROA_NC_sharpe_test ROA_TC_sharpe_test ROA_SC_sharpe_test...
                  ROE_SC_sharpe_test


%% OSEFX data
% 1. Import data
opts = delimitedTextImportOptions("NumVariables", 3);
% Specify range and delimiter
opts.DataLines = [2511, 5270];
opts.Delimiter = ",";

% Specify column names and types
opts.VariableNames = ["time", "OSEFX_price", "Var3"];
opts.SelectedVariableNames = ["time", "OSEFX_price"];
opts.VariableTypes = ["datetime", "double", "string"];
opts = setvaropts(opts, 1, "InputFormat", "yyyy-MM-dd HH:mm");
opts = setvaropts(opts, 3, "WhitespaceRule", "preserve");
opts = setvaropts(opts, 3, "EmptyFieldRule", "auto");
opts.ExtraColumnsRule = "ignore";
opts.EmptyLineRule = "read";

% Import the data
OSEFX = readtimetable("OSEFX_Historical_Data_2010_2020.csv", opts);

OSEFX = retime(OSEFX, 'quarterly', 'lastvalue');
OSEFX = timetable2table(OSEFX);
OSEFX.time = categorical(price_houses.Q(1:44));
clear opts

% Import risk-free rates (innskuddsrente)
OSEFX.rf = priceindex_houses.rf;

% Implement tax rates for sample period
tax_rates = ([ 0.28, 0.28, 0.28, 0.28, 0.27, 0.27, 0.25, 0.24, 0.23, 0.22, 0.22])';
n = 4;
tax_rates_order =
[repmat(tax_rates(1),n,1);repmat(tax_rates(2),n,1);repmat(tax_rates(3),n,1);repmat(tax
```

```
_rates(4),n,1);repmat(tax_rates(5),n,1);repmat(tax_rates(6),n,1);repmat(tax_rates(7),n
,1);repmat(tax_rates(8),n,1);repmat(tax_rates(9),n,1);repmat(tax_rates(10),n,1);repmat
(tax_rates(11),n,1)];
OSEFX.tax_rates = tax_rates_order;

% 2. Compute index values, OSEFX
OSEFX.index_values = zeros(height(OSEFX), end);
for i = 1:length(OSEFX.index_values)
    OSEFX.index_values(i) = OSEFX.OSEFX_price(i)/OSEFX.OSEFX_price(1);
end

% 3. Compute quarterly returns, (NC)
OSEFX.returns = zeros(height(OSEFX), end);
for i = 2:length(OSEFX.index_values)
    OSEFX.returns(i) = (OSEFX.OSEFX_price(i)/OSEFX.OSEFX_price(i-1))-1;
end

% Compute mean of mean quarterly returns for OSEFX (NC)
mean_Q_ret_OSEFX = nanmean(OSEFX.returns);

% Compute excess quarterly returns for OSEFX (NC)
OSEFX.excess_Q_ret    = OSEFX.returns-OSEFX.rf;
OSEFX.excess_Q_ret(1) = 0;

% Compute mean excess quarterly returns for OSEFX (NC)
mean_excess_Q_ret_OSEFX = nanmean(OSEFX.excess_Q_ret);

% Standard deviation of all quarterly returns
std_Q_ret_OSEFX = std(OSEFX.returns(2:end));

% Compute cumulative returns, no costs
OSEFX.cum_Q_ret = zeros(height(OSEFX),end);
OSEFX.cum_Q_ret = cumprod((1+OSEFX.returns),'omitnan');

% Create summary stats table for OSEFX returns
sum_stats_OSEFX                          = zeros(2,7);
sum_stats_OSEFX                          = array2table(sum_stats_OSEFX);
sum_stats_OSEFX.Properties.VariableNames = {'OSEFX', 'Return',
'std_dev','Excess_return','Sharpe_ratio', 'Skewness', 'Kurtosis'};
sum_stats_OSEFX.OSEFX                    = categorical(sum_stats_OSEFX.OSEFX);

% Import stats for 'Q_ret_OSEFX' in summary table
sum_stats_OSEFX.OSEFX(1)         = 'Q_ret_OSEFX';
sum_stats_OSEFX.Return(1)        = mean_Q_ret_OSEFX;
sum_stats_OSEFX.std_dev(1)       = std_Q_ret_OSEFX;
sum_stats_OSEFX.Excess_return(1) = mean_excess_Q_ret_OSEFX;
sum_stats_OSEFX.Sharpe_ratio(1)  =
sum_stats_OSEFX.Excess_return(1)/sum_stats_OSEFX.std_dev(1);
sum_stats_OSEFX.Skewness(1)      = skewness(OSEFX.returns(2:44));
sum_stats_OSEFX.Kurtosis(1)      = kurtosis(OSEFX.returns(2:44));

% 4. Compute cumulative returns (SC)
OSEFX.cum_Q_ret_SC = OSEFX.cum_Q_ret;
OSEFX.cum_Q_ret_SC(end) = ((OSEFX.cum_Q_ret(end)-1)*(1-OSEFX.tax_rates(end)))+1;

% 5. Compute quarterly returns (TC)
OSEFX.returns_TC = OSEFX.returns.*(1-OSEFX.tax_rates);

% Compute cumulative returns (TC)
OSEFX.cum_Q_ret_TC = zeros(height(OSEFX),end);
OSEFX.cum_Q_ret_TC = cumprod((1+OSEFX.returns_TC),'omitnan');

% Compute mean of mean quarterly returns for OSEFX (TC)
mean_Q_ret_OSEFX_TC = nanmean(OSEFX.returns_TC);

% Compute excess quarterly returns for OSEFX (TC)
OSEFX.excess_Q_ret_TC    = OSEFX.returns_TC-OSEFX.rf;
OSEFX.excess_Q_ret_TC(1) = 0;

% Compute mean excess quarterly returns for OSEFX (TC)
mean_excess_Q_ret_OSEFX_TC = nanmean(OSEFX.excess_Q_ret_TC);

% Standard deviation of all quarterly returns (TC)
std_Q_ret_OSEFX_TC = std(OSEFX.returns_TC(2:end));
```

```
% Import stats for 'Q_ret_OSEFX_TC' in summary table
sum_stats_OSEFX.OSEFX(2)          = 'Q_ret_OSEFX_TC';
sum_stats_OSEFX.Return(2)         = mean_Q_ret_OSEFX_TC;
sum_stats_OSEFX.std_dev(2)        = std_Q_ret_OSEFX_TC;
sum_stats_OSEFX.Excess_return(2) = mean_excess_Q_ret_OSEFX_TC;
sum_stats_OSEFX.Sharpe_ratio(2)   =
sum_stats_OSEFX.Excess_return(2)/sum_stats_OSEFX.std_dev(2);
sum_stats_OSEFX.Skewness(2)       = skewness(OSEFX.returns_TC(2:44));
sum_stats_OSEFX.Kurtosis(2)       = kurtosis(OSEFX.returns_TC(2:44));

clearvars -except Property_Data summary_table Rf Rent_Data...
                  house_set hedonic_houses housing_model...
                  price_houses uniq_house_set avg_cost_rates_houses...
                  priceindex_houses sum_stats_houses houseprice_changes...
                  houseprice_indices...
                  smallhouse_set hedonic_smallhouses smallhouse_model...
                  price_smallhouses uniq_smallhouse_set avg_cost_rates_smallhouses...
                  priceindex_smallhouses sum_stats_smallhouses...
                  smallhouse_price_changes smallhouse_price_indices...
                  apartment_set hedonic_apartments apartment_model price_apartments...
                  uniq_apartment_set avg_cost_rates_apartments
priceindex_apartments...
                  sum_stats_apartments apartment_price_changes...
                  apartment_price_indices...
                  inner_eastern_set hedonic_inner_eastern inner_eastern_model...
                  price_inner_eastern uniq_inner_eastern_set
avg_cost_rates_inner_eastern...
                  priceindex_inner_eastern sum_stats_inner_eastern
inner_eastern_price_changes...
                  inner_eastern_price_indices...
                  inner_western_set hedonic_inner_western inner_western_model...
                  price_inner_western uniq_inner_western_set
avg_cost_rates_inner_western...
                  priceindex_inner_western sum_stats_inner_western
inner_western_price_changes...
                  inner_western_price_indices...
                  outer_eastern_set hedonic_outer_eastern outer_eastern_model...
                  price_outer_eastern uniq_outer_eastern_set
avg_cost_rates_outer_eastern...
                  priceindex_outer_eastern sum_stats_outer_eastern
outer_eastern_price_changes...
                  outer_eastern_price_indices...
                  outer_western_set hedonic_outer_western outer_western_model...
                  price_outer_western uniq_outer_western_set
avg_cost_rates_outer_western...
                  priceindex_outer_western sum_stats_outer_western
outer_western_price_changes...
                  outer_western_price_indices...
                  outer_southern_set hedonic_outer_southern outer_southern_model...
                  price_outer_southern uniq_outer_southern_set
avg_cost_rates_outer_southern...
                  priceindex_outer_southern sum_stats_outer_southern
outer_southern_price_changes...
                  outer_southern_price_indices...
                  property_price_changes property_price_indices hedonic_total...
                  ROA_NC_stats ROE_NC_stats ROA_TC_stats ROE_TC_stats...
                  ROA_SC_stats ROE_SC_stats sum_stats_OSEFX OSEFX...
                  ROA_NC_sharpe_test ROA_TC_sharpe_test ROA_SC_sharpe_test...
                  ROE_SC_sharpe_test

%% PRESENT SUMARY PLOTS, STATS AND TEST RESULTS
% Plot quarterly asset returns/changes (no costs) from 2010Q1
plot(apartment_price_indices.Q_order, [property_price_changes.Q_ret_no_costs,
OSEFX.returns],'-x', 'MarkerSize', 5, 'MarkerIndices', 1:44, 'LineWidth',1.5)
yline(0,'--')
xticks({'2010Q1' '2011Q1' '2012Q1' '2013Q1' '2014Q1' '2015Q1' '2016Q1' '2017Q1'
'2018Q1' '2019Q1' '2020Q1' '2020Q4'})
xticklabels({'2010','2011','2012','2013','2014','2015','2016','2017','2018','2019','20
20','2021'})
xlabel('(a) Changes')
ylabel('Quarterly returns')
a = [cellstr(num2str(get(gca,'ytick')'*100))];
pct = char(ones(size(a,1),1)*'%');
```

```
new_yticks = [char(a),pct];
set(gca,'yticklabel',new_yticks)
legend({'Oslo properties (NC)','OSEFX (NC)'},'Location','best')

% Plot cumulative asset returns (no costs) from 2010Q1
plot(apartment_price_indices.Q_order, [property_price_indices.cum_Q_ret_no_costs,
OSEFX.cum_Q_ret], 'LineWidth',1.5)
yline(1,'--')
xticks({'2010Q1' '2011Q1' '2012Q1' '2013Q1' '2014Q1' '2015Q1' '2016Q1' '2017Q1'
'2018Q1' '2019Q1' '2020Q1' '2020Q4'})
xticklabels({'2010','2011','2012','2013','2014','2015','2016','2017','2018','2019','20
20','2021'})
xlabel('(b) Indexes')
ylabel('Index (2010Q1 = 1)')
legend({'Oslo properties (NC)','OSEFX (NC)'},'Location','northwest')

% Plot quarterly asset returns/changes (no costs) from 2010Q1, for dwelling
% types
plot(apartment_price_indices.Q_order, [apartment_price_changes.Q_ret_no_costs,
OSEFX.returns, houseprice_changes.Q_ret_no_costs,
smallhouse_price_changes.Q_ret_no_costs],'-x', 'MarkerSize', 5, 'MarkerIndices', 1:44,
'LineWidth',1.5)
yline(0,'--')
xticks({'2010Q1' '2011Q1' '2012Q1' '2013Q1' '2014Q1' '2015Q1' '2016Q1' '2017Q1'
'2018Q1' '2019Q1' '2020Q1' '2020Q4'})
xticklabels({'2010','2011','2012','2013','2014','2015','2016','2017','2018','2019','20
20','2021'})
xlabel('(a) Changes')
ylabel('Quarterly returns')
a = [cellstr(num2str(get(gca,'ytick')'*100))];
pct = char(ones(size(a,1),1)*'%');
new_yticks = [char(a),pct];
set(gca,'yticklabel',new_yticks)
legend({'Apartments (NC)', 'OSEFX (NC)', 'Houses (NC)','Small Houses
(NC)'},'Location','best')

% Plot cummulative asset returns (no costs) from 2010Q1 for dwelling types
plot(apartment_price_indices.Q_order, [apartment_price_indices.cum_Q_ret_no_costs,
OSEFX.cum_Q_ret, houseprice_indices.cum_Q_ret_no_costs,
smallhouse_price_indices.cum_Q_ret_no_costs], 'LineWidth',1.5)
yline(1,'--')
xticks({'2010Q1' '2011Q1' '2012Q1' '2013Q1' '2014Q1' '2015Q1' '2016Q1' '2017Q1'
'2018Q1' '2019Q1' '2020Q1' '2020Q4'})
xticklabels({'2010','2011','2012','2013','2014','2015','2016','2017','2018','2019','20
20','2021'})
xlabel('(b) Indexes')
ylabel('Index (2010Q1 = 1)')
legend({'Apartments (NC)', 'OSEFX (NC)', 'Houses (NC)','Small Houses
(NC)'},'Location','northwest')

% Plot quarterly asset returns/changes (no costs) from 2010Q1, for
% districts
plot(apartment_price_indices.Q_order, [inner_eastern_price_changes.Q_ret_no_costs,
OSEFX.returns, inner_western_price_changes.Q_ret_no_costs,
outer_eastern_price_changes.Q_ret_no_costs,
outer_western_price_changes.Q_ret_no_costs,
outer_southern_price_changes.Q_ret_no_costs],'-x', 'MarkerSize', 5, 'MarkerIndices',
1:44, 'LineWidth',1.5)
yline(0,'--')
xticks({'2010Q1' '2011Q1' '2012Q1' '2013Q1' '2014Q1' '2015Q1' '2016Q1' '2017Q1'
'2018Q1' '2019Q1' '2020Q1' '2020Q4'})
xticklabels({'2010','2011','2012','2013','2014','2015','2016','2017','2018','2019','20
20','2021'})
xlabel('(a) Changes')
ylabel('Quarterly returns')
a = [cellstr(num2str(get(gca,'ytick')'*100))];
pct = char(ones(size(a,1),1)*'%');
new_yticks = [char(a),pct];
set(gca,'yticklabel',new_yticks)
legend({'Inner east (NC)', 'OSEFX (NC)', 'Inner west (NC)', 'Outer east (NC)','Outer
west (NC)','Outer south (NC)'},'Location','best')

% Plot cummulative asset returns (no costs) from 2010Q1 for districts
```

```
plot(apartment_price_indices.Q_order, [inner_eastern_price_indices.cum_Q_ret_no_costs,
OSEFX.cum_Q_ret, inner_western_price_indices.cum_Q_ret_no_costs,
outer_eastern_price_indices.cum_Q_ret_no_costs,
outer_western_price_indices.cum_Q_ret_no_costs,
outer_southern_price_indices.cum_Q_ret_no_costs], 'LineWidth',1.5)
yline(1,'--')
xticks({'2010Q1' '2011Q1' '2012Q1' '2013Q1' '2014Q1' '2015Q1' '2016Q1' '2017Q1'
'2018Q1' '2019Q1' '2020Q1' '2020Q4'})
xticklabels({'2010','2011','2012','2013','2014','2015','2016','2017','2018','2019','20
20','2021'})
xlabel('(b) Indexes')
ylabel('Index (2010Q1 = 1)')
legend({'Inner east (NC)', 'OSEFX (NC)', 'Inner west (NC)', 'Outer east (NC)','Outer
west (NC)','Outer south (NC)'},'Location','northwest')

%ROA(NC) TABLE
% Create a table to store returns, std, risk free, sharpe
ROA_NC_stats                          = zeros(9,7);
ROA_NC_stats                          = array2table(ROA_NC_stats);
ROA_NC_stats.Properties.VariableNames = {'Stratum', 'Return',
'std_dev','Excess_return','Sharpe_ratio', 'Skewness', 'Kurtosis'};
ROA_NC_stats.Stratum                  = categorical(ROA_NC_stats.Stratum);

ROA_NC_stats.Stratum(1) = 'Apartments';
ROA_NC_stats.Stratum(2) = 'Houses';
ROA_NC_stats.Stratum(3) = 'Small houses';
ROA_NC_stats.Stratum(4) = 'Inner Eastern';
ROA_NC_stats.Stratum(5) = 'Inner Western';
ROA_NC_stats.Stratum(6) = 'Outer Eastern';
ROA_NC_stats.Stratum(7) = 'Outer Western';
ROA_NC_stats.Stratum(8) = 'Outer Southern';
ROA_NC_stats.Stratum(9) = 'OSEFX';
ROA_NC_stats(1,2:end)   = sum_stats_apartments(1,2:end);
ROA_NC_stats(2,2:end)   = sum_stats_houses(1,2:end);
ROA_NC_stats(3,2:end)   = sum_stats_smallhouses(1,2:end);
ROA_NC_stats(4,2:end)   = sum_stats_inner_eastern(1,2:end);
ROA_NC_stats(5,2:end)   = sum_stats_inner_western(1,2:end);
ROA_NC_stats(6,2:end)   = sum_stats_outer_eastern(1,2:end);
ROA_NC_stats(7,2:end)   = sum_stats_outer_western(1,2:end);
ROA_NC_stats(8,2:end)   = sum_stats_outer_southern(1,2:end);
ROA_NC_stats(9,2:end)   = sum_stats_OSEFX(1,2:end);

% Plot volatility and excess returns - Sharpe evaluation
h = gscatter(ROA_NC_stats.std_dev, ROA_NC_stats.Excess_return, ROA_NC_stats.Stratum);
for i = 2:10
    jgroup = h(i);
    jgroup.MarkerSize = 30;
end
xlabel('Volatility');
ylabel('Excess quarterly returns');
a = [cellstr(num2str(get(gca,'ytick')'*100))];
pct = char(ones(size(a,1),1)*'%');
new_yticks = [char(a),pct];
set(gca,'yticklabel',new_yticks);
b = [cellstr(num2str(get(gca,'xtick')'*100))];
pct = char(ones(size(b,1),1)*'%');
new_xticks = [char(b),pct];
set(gca,'xticklabel',new_xticks);

% Hypothesis tesing - testing difference in means between returns to real
% estate (ROA(NC)) and OSEFX
[h,p,ci,stats] = ttest2(apartment_price_changes.Q_ret_no_costs(2:44),
OSEFX.returns(2:44), 'Tail', 'right', 'Vartype', 'unequal')
[h,p,ci,stats] = ttest2(houseprice_changes.Q_ret_no_costs(2:44),
OSEFX.returns(2:44), 'Tail', 'right', 'Vartype', 'unequal')
[h,p,ci,stats] = ttest2(smallhouse_price_changes.Q_ret_no_costs(2:44),
OSEFX.returns(2:44), 'Tail', 'right', 'Vartype', 'unequal')
[h,p,ci,stats] = ttest2(inner_eastern_price_changes.Q_ret_no_costs(2:44),
OSEFX.returns(2:44), 'Tail', 'right', 'Vartype', 'unequal')
[h,p,ci,stats] = ttest2(inner_western_price_changes.Q_ret_no_costs(2:44),
OSEFX.returns(2:44), 'Tail', 'right', 'Vartype', 'unequal')
[h,p,ci,stats] = ttest2(outer_eastern_price_changes.Q_ret_no_costs(2:44),
OSEFX.returns(2:44), 'Tail', 'right', 'Vartype', 'unequal')
```

```
[h,p,ci,stats] = ttest2(outer_western_price_changes.Q_ret_no_costs(2:44),
OSEFX.returns(2:44), 'Tail', 'right', 'Vartype', 'unequal')
[h,p,ci,stats] = ttest2(outer_southern_price_changes.Q_ret_no_costs(2:44),
OSEFX.returns(2:44), 'Tail', 'right', 'Vartype', 'unequal')

% Hypothesis tesing - testing for equal variances (F-test) between returns to real
% estate (ROA(NC)) and OSEFX
[h,p,ci,stats] = vartest2(apartment_price_changes.Q_ret_no_costs(2:44),
OSEFX.returns(2:44), 'Tail', 'left')
[h,p,ci,stats] = vartest2(houseprice_changes.Q_ret_no_costs(2:44),
OSEFX.returns(2:44), 'Tail', 'left')
[h,p,ci,stats] = vartest2(smallhouse_price_changes.Q_ret_no_costs(2:44),
OSEFX.returns(2:44), 'Tail', 'left')
[h,p,ci,stats] = vartest2(inner_eastern_price_changes.Q_ret_no_costs(2:44),
OSEFX.returns(2:44), 'Tail', 'left')
[h,p,ci,stats] = vartest2(inner_western_price_changes.Q_ret_no_costs(2:44),
OSEFX.returns(2:44), 'Tail', 'left')
[h,p,ci,stats] = vartest2(outer_eastern_price_changes.Q_ret_no_costs(2:44),
OSEFX.returns(2:44), 'Tail', 'left')
[h,p,ci,stats] = vartest2(outer_western_price_changes.Q_ret_no_costs(2:44),
OSEFX.returns(2:44), 'Tail', 'left')
[h,p,ci,stats] = vartest2(outer_southern_price_changes.Q_ret_no_costs(2:44),
OSEFX.returns(2:44), 'Tail', 'left')

% Hypothesis tesing - testing for equal Sharpe ratios between returns to real
% estate (ROA(NC)) and OSEFX
ROA_NC_sharpe_test                          = zeros(9,8);
ROA_NC_sharpe_test                          = array2table(ROA_NC_sharpe_test);
ROA_NC_sharpe_test.Properties.VariableNames = {'Stratum', 'Sharpe_ratio', 'Skewness',
'Kurtosis', 'Sharpe_stDev', 'z_stat' 'PSR', 'pValue'};
ROA_NC_sharpe_test.Stratum                  = categorical(ROA_NC_sharpe_test.Stratum);
ROA_NC_sharpe_test.Stratum                  = ROA_NC_stats.Stratum;
ROA_NC_sharpe_test.Sharpe_ratio             = ROA_NC_stats.Sharpe_ratio;

ROA_NC_sharpe_test(1,3:4)   = sum_stats_apartments(1,6:7);
ROA_NC_sharpe_test(2,3:4)   = sum_stats_houses(1,6:7);
ROA_NC_sharpe_test(3,3:4)   = sum_stats_smallhouses(1,6:7);
ROA_NC_sharpe_test(4,3:4)   = sum_stats_inner_eastern(1,6:7);
ROA_NC_sharpe_test(5,3:4)   = sum_stats_inner_western(1,6:7);
ROA_NC_sharpe_test(6,3:4)   = sum_stats_outer_eastern(1,6:7);
ROA_NC_sharpe_test(7,3:4)   = sum_stats_outer_western(1,6:7);
ROA_NC_sharpe_test(8,3:4)   = sum_stats_outer_southern(1,6:7);
ROA_NC_sharpe_test(9,3:4)   = sum_stats_OSEFX(1,6:7);

for i = 1:9
    ROA_NC_sharpe_test.Sharpe_stDev(i) =
sqrt((1/43)*(1+(0.5*((ROA_NC_sharpe_test.Sharpe_ratio(i))^2))-
(ROA_NC_sharpe_test.Skewness(i)*ROA_NC_sharpe_test.Sharpe_ratio(i))+(((ROA_NC_sharpe_t
est.Kurtosis(i)-3)/4)*((ROA_NC_sharpe_test.Sharpe_ratio(i))^2))));
    ROA_NC_sharpe_test.z_stat(i) = (ROA_NC_sharpe_test.Sharpe_ratio(i) -
ROA_NC_sharpe_test.Sharpe_ratio(9))/ROA_NC_sharpe_test.Sharpe_stDev(i);
    ROA_NC_sharpe_test.PSR(i) = normcdf(ROA_NC_sharpe_test.z_stat(i));
    ROA_NC_sharpe_test.pValue(i) = 1 - ROA_NC_sharpe_test.PSR(i);
end
% 5% significance level, crit_val = 1.645;

%% ROA (TC)
% Plot quarterly asset returns/changes (tot costs) from 2010Q1
plot(apartment_price_indices.Q_order, [property_price_changes.Q_ret_tot_costs,
OSEFX.returns_TC],'-x', 'MarkerSize', 5, 'MarkerIndices', 1:44, 'LineWidth',1.5)
yline(0,'--')
xticks({'2010Q1' '2011Q1' '2012Q1' '2013Q1' '2014Q1' '2015Q1' '2016Q1' '2017Q1'
'2018Q1' '2019Q1' '2020Q1' '2020Q4'})
xticklabels({'2010','2011','2012','2013','2014','2015','2016','2017','2018','2019','20
20','2021'})
xlabel('(a) Changes')
ylabel('Quarterly returns')
a = [cellstr(num2str(get(gca,'ytick')'*100))];
pct = char(ones(size(a,1),1)*'%');
new_yticks = [char(a),pct];
set(gca,'yticklabel',new_yticks)
legend({'Oslo properties (TC)','OSEFX (TC)'},'Location','northwest')

% Plot cumulative asset returns (tot costs) from 2010Q1
```

```
plot(apartment_price_indices.Q_order, [property_price_indices.cum_Q_ret_tot_costs,
OSEFX.cum_Q_ret_TC], 'LineWidth',1.5)
yline(1,'--')
xticks({'2010Q1' '2011Q1' '2012Q1' '2013Q1' '2014Q1' '2015Q1' '2016Q1' '2017Q1'
'2018Q1' '2019Q1' '2020Q1' '2020Q4'})
xticklabels({'2010','2011','2012','2013','2014','2015','2016','2017','2018','2019','20
20','2021'})
xlabel('(b) Indexes')
ylabel('Index (2010Q1 = 1)')
legend({'Oslo properties (TC)','OSEFX (TC)'},'Location','northwest')

% Plot quarterly asset returns/changes (tot costs) from 2010Q1, for dwelling
% types
plot(apartment_price_indices.Q_order, [apartment_price_changes.Q_ret_tot_costs,
OSEFX.returns, houseprice_changes.Q_ret_tot_costs,
smallhouse_price_changes.Q_ret_tot_costs],'-x', 'MarkerSize', 5, 'MarkerIndices',
1:44, 'LineWidth',1.5)
yline(0,'--')
xticks({'2010Q1' '2011Q1' '2012Q1' '2013Q1' '2014Q1' '2015Q1' '2016Q1' '2017Q1'
'2018Q1' '2019Q1' '2020Q1' '2020Q4'})
xticklabels({'2010','2011','2012','2013','2014','2015','2016','2017','2018','2019','20
20','2021'})
xlabel('(a) Changes')
ylabel('Quarterly returns')
a = [cellstr(num2str(get(gca,'ytick')'*100))];
pct = char(ones(size(a,1),1)*'%');
new_yticks = [char(a),pct];
set(gca,'yticklabel',new_yticks)
legend({'Apartments', 'OSEFX', 'Houses','Small Houses'},'Location','northwest')

% Plot cummulative asset returns (tot costs) from 2010Q1 for dwelling types
plot(apartment_price_indices.Q_order, [apartment_price_indices.cum_Q_ret_tot_costs,
OSEFX.cum_Q_ret, houseprice_indices.cum_Q_ret_tot_costs,
smallhouse_price_indices.cum_Q_ret_tot_costs], 'LineWidth',1.5)
yline(1,'--')
xticks({'2010Q1' '2011Q1' '2012Q1' '2013Q1' '2014Q1' '2015Q1' '2016Q1' '2017Q1'
'2018Q1' '2019Q1' '2020Q1' '2020Q4'})
xticklabels({'2010','2011','2012','2013','2014','2015','2016','2017','2018','2019','20
20','2021'})
xlabel('(a) Indexes')
ylabel('Index (2010Q1 = 1)')
legend({'Apartments', 'OSEFX', 'Houses','Small Houses'},'Location','northwest')

% Plot quarterly asset returns/changes (tot costs) from 2010Q1, for
% districts
plot(apartment_price_indices.Q_order, [inner_eastern_price_changes.Q_ret_tot_costs,
OSEFX.returns, inner_western_price_changes.Q_ret_tot_costs,
outer_eastern_price_changes.Q_ret_tot_costs,
outer_western_price_changes.Q_ret_tot_costs,
outer_southern_price_changes.Q_ret_tot_costs],'-x', 'MarkerSize', 5, 'MarkerIndices',
1:44, 'LineWidth',1.5)
yline(0,'--')
xticks({'2010Q1' '2011Q1' '2012Q1' '2013Q1' '2014Q1' '2015Q1' '2016Q1' '2017Q1'
'2018Q1' '2019Q1' '2020Q1' '2020Q4'})
xticklabels({'2010','2011','2012','2013','2014','2015','2016','2017','2018','2019','20
20','2021'})
xlabel('(a) Changes')
ylabel('Quarterly returns')
a = [cellstr(num2str(get(gca,'ytick')'*100))];
pct = char(ones(size(a,1),1)*'%');
new_yticks = [char(a),pct];
set(gca,'yticklabel',new_yticks)
legend({'Inner east', 'OSEFX', 'Inner west', 'Outer east','Outer west','Outer
south'},'Location','northwest')

% Plot cummulative asset returns (tot costs) from 2010Q1 for districts
plot(apartment_price_indices.Q_order,
[inner_eastern_price_indices.cum_Q_ret_tot_costs, OSEFX.cum_Q_ret,
inner_western_price_indices.cum_Q_ret_tot_costs,
outer_eastern_price_indices.cum_Q_ret_tot_costs,
outer_western_price_indices.cum_Q_ret_tot_costs,
outer_southern_price_indices.cum_Q_ret_tot_costs], 'LineWidth',1.5)
yline(1,'--')
```

```
xticks({'2010Q1' '2011Q1' '2012Q1' '2013Q1' '2014Q1' '2015Q1' '2016Q1' '2017Q1'
'2018Q1' '2019Q1' '2020Q1' '2020Q4'})
xticklabels({'2010','2011','2012','2013','2014','2015','2016','2017','2018','2019','20
20','2021'})
xlabel('(a) Indexes')
ylabel('Index (2010Q1 = 1)')
legend({'Inner east', 'OSEFX', 'Inner west', 'Outer east','Outer west','Outer
south'},'Location','northwest')

%ROA(TC) TABLE
% Create a table to store returns, std, risk free, sharpe
ROA_TC_stats                          = zeros(9,7);
ROA_TC_stats                          = array2table(ROA_TC_stats);
ROA_TC_stats.Properties.VariableNames = {'Stratum', 'Return',
'std_dev','Excess_return','Sharpe_ratio', 'Skewness', 'Kurtosis'};
ROA_TC_stats.Stratum                  = categorical(ROA_TC_stats.Stratum);

ROA_TC_stats.Stratum  = ROA_NC_stats.Stratum;
ROA_TC_stats(1,2:end) = sum_stats_apartments(2,2:end);
ROA_TC_stats(2,2:end) = sum_stats_houses(2,2:end);
ROA_TC_stats(3,2:end) = sum_stats_smallhouses(2,2:end);
ROA_TC_stats(4,2:end) = sum_stats_inner_eastern(2,2:end);
ROA_TC_stats(5,2:end) = sum_stats_inner_western(2,2:end);
ROA_TC_stats(6,2:end) = sum_stats_outer_eastern(2,2:end);
ROA_TC_stats(7,2:end) = sum_stats_outer_western(2,2:end);
ROA_TC_stats(8,2:end) = sum_stats_outer_southern(2,2:end);
ROA_TC_stats(9,2:end) = sum_stats_OSEFX(2,2:end);

% Plot volatility and excess returns - Sharpe evaluation
h = gscatter(ROA_TC_stats.std_dev, ROA_TC_stats.Excess_return, ROA_TC_stats.Stratum);
for i = 2:10
    jgroup = h(i);
    jgroup.MarkerSize = 30;
end
xlabel('Volatility');
ylabel('Excess quarterly returns');
a = [cellstr(num2str(get(gca,'ytick')'*100))];
pct = char(ones(size(a,1),1)*'%');
new_yticks = [char(a),pct];
set(gca,'yticklabel',new_yticks);
b = [cellstr(num2str(get(gca,'xtick')'*100))];
pct = char(ones(size(b,1),1)*'%');
new_xticks = [char(b),pct];
set(gca,'xticklabel',new_xticks);

% Hypothesis tesing - testing difference in means between returns to real
% estate (ROA(TC)) and OSEFX
[h,p,ci,stats] = ttest2(apartment_price_changes.Q_ret_tot_costs(2:44),
OSEFX.returns_TC(2:44), 'Tail', 'left', 'Vartype', 'unequal')
[h,p,ci,stats] = ttest2(houseprice_changes.Q_ret_tot_costs(2:44),
OSEFX.returns_TC(2:44), 'Tail', 'left', 'Vartype', 'unequal')
[h,p,ci,stats] = ttest2(smallhouse_price_changes.Q_ret_tot_costs(2:44),
OSEFX.returns_TC(2:44), 'Tail', 'left', 'Vartype', 'unequal')
[h,p,ci,stats] = ttest2(inner_eastern_price_changes.Q_ret_tot_costs(2:44),
OSEFX.returns_TC(2:44), 'Tail', 'left', 'Vartype', 'unequal')
[h,p,ci,stats] = ttest2(inner_western_price_changes.Q_ret_tot_costs(2:44),
OSEFX.returns_TC(2:44), 'Tail', 'left', 'Vartype', 'unequal')
[h,p,ci,stats] = ttest2(outer_eastern_price_changes.Q_ret_tot_costs(2:44),
OSEFX.returns_TC(2:44), 'Tail', 'left', 'Vartype', 'unequal')
[h,p,ci,stats] = ttest2(outer_western_price_changes.Q_ret_tot_costs(2:44),
OSEFX.returns_TC(2:44), 'Tail', 'left', 'Vartype', 'unequal')
[h,p,ci,stats] = ttest2(outer_southern_price_changes.Q_ret_tot_costs(2:44),
OSEFX.returns_TC(2:44), 'Tail', 'left', 'Vartype', 'unequal')

% Hypothesis tesing - testing for equal variances (F-test) between returns to real
% estate (ROA(TC)) and OSEFX
[h,p,ci,stats] = vartest2(apartment_price_changes.Q_ret_tot_costs(2:44),
OSEFX.returns_TC(2:44), 'Tail', 'left')
[h,p,ci,stats] = vartest2(houseprice_changes.Q_ret_tot_costs(2:44),
OSEFX.returns_TC(2:44), 'Tail', 'left')
[h,p,ci,stats] = vartest2(smallhouse_price_changes.Q_ret_tot_costs(2:44),
OSEFX.returns_TC(2:44), 'Tail', 'left')
[h,p,ci,stats] = vartest2(inner_eastern_price_changes.Q_ret_tot_costs(2:44),
OSEFX.returns_TC(2:44), 'Tail', 'left')
```

```
[h,p,ci,stats] = vartest2(inner_western_price_changes.Q_ret_tot_costs(2:44),
OSEFX.returns_TC(2:44), 'Tail', 'left')
[h,p,ci,stats] = vartest2(outer_eastern_price_changes.Q_ret_tot_costs(2:44),
OSEFX.returns_TC(2:44), 'Tail', 'left')
[h,p,ci,stats] = vartest2(outer_western_price_changes.Q_ret_tot_costs(2:44),
OSEFX.returns_TC(2:44), 'Tail', 'left')
[h,p,ci,stats] = vartest2(outer_southern_price_changes.Q_ret_tot_costs(2:44),
OSEFX.returns_TC(2:44), 'Tail', 'left')

% Hypothesis tesing - testing for equal Sharpe ratios between returns to real
% estate (ROA(TC)) and OSEFX
ROA_TC_sharpe_test                          = zeros(9,8);
ROA_TC_sharpe_test                          = array2table(ROA_TC_sharpe_test);
ROA_TC_sharpe_test.Properties.VariableNames = {'Stratum', 'Sharpe_ratio', 'Skewness',
'Kurtosis', 'Sharpe_stDev', 'z_stat' 'PSR', 'pValue'};
ROA_TC_sharpe_test.Stratum                  = categorical(ROA_TC_sharpe_test.Stratum);
ROA_TC_sharpe_test.Stratum                  = ROA_TC_stats.Stratum;
ROA_TC_sharpe_test.Sharpe_ratio             = ROA_TC_stats.Sharpe_ratio;

ROA_TC_sharpe_test(1,3:4)   = sum_stats_apartments(1,6:7);
ROA_TC_sharpe_test(2,3:4)   = sum_stats_houses(1,6:7);
ROA_TC_sharpe_test(3,3:4)   = sum_stats_smallhouses(1,6:7);
ROA_TC_sharpe_test(4,3:4)   = sum_stats_inner_eastern(1,6:7);
ROA_TC_sharpe_test(5,3:4)   = sum_stats_inner_western(1,6:7);
ROA_TC_sharpe_test(6,3:4)   = sum_stats_outer_eastern(1,6:7);
ROA_TC_sharpe_test(7,3:4)   = sum_stats_outer_western(1,6:7);
ROA_TC_sharpe_test(8,3:4)   = sum_stats_outer_southern(1,6:7);
ROA_TC_sharpe_test(9,3:4)   = sum_stats_OSEFX(2,6:7);

for i = 1:9
    ROA_TC_sharpe_test.Sharpe_stDev(i) =
sqrt((1/43)*(1+(0.5*((ROA_TC_sharpe_test.Sharpe_ratio(i))^2))-
(ROA_TC_sharpe_test.Skewness(i)*ROA_TC_sharpe_test.Sharpe_ratio(i))+(((ROA_TC_sharpe_t
est.Kurtosis(i)-3)/4)*((ROA_TC_sharpe_test.Sharpe_ratio(i))^2))));
    ROA_TC_sharpe_test.z_stat(i) = (ROA_TC_sharpe_test.Sharpe_ratio(i) -
ROA_TC_sharpe_test.Sharpe_ratio(9))/ROA_TC_sharpe_test.Sharpe_stDev(i);
    ROA_TC_sharpe_test.PSR(i) = 1 - normcdf(ROA_TC_sharpe_test.z_stat(i));
    ROA_TC_sharpe_test.pValue(i) = 1 - ROA_TC_sharpe_test.PSR(i);
end
% 5% significance level, crit_val = 1.645;

%% ROA (SC)
% Plot quarterly asset returns/changes (smooth costs) from 2010Q1
plot(apartment_price_indices.Q_order, [property_price_changes.Q_ret_no_costs,
OSEFX.returns, property_price_changes.Q_ret_smooth_costs],'-x', 'MarkerSize', 5,
'MarkerIndices', 1:44, 'LineWidth',1.5)
yline(0,'--')
xticks({'2010Q1' '2011Q1' '2012Q1' '2013Q1' '2014Q1' '2015Q1' '2016Q1' '2017Q1'
'2018Q1' '2019Q1' '2020Q1' '2020Q4'})
xticklabels({'2010','2011','2012','2013','2014','2015','2016','2017','2018','2019','20
20','2021'})
xlabel('(a) Changes')
ylabel('Quarterly returns')
a = [cellstr(num2str(get(gca,'ytick')'*100))];
pct = char(ones(size(a,1),1)*'%');
new_yticks = [char(a),pct];
set(gca,'yticklabel',new_yticks)
legend({'Oslo properties (NC)', 'OSEFX (SC)', 'Oslo properties
(SC)'},'Location','northwest')

% Plot cumulative asset returns (smooth costs) from 2010Q1
plot(apartment_price_indices.Q_order, [property_price_indices.cum_Q_ret_no_costs,
OSEFX.cum_Q_ret_SC, property_price_indices.cum_Q_ret_smooth_costs], 'LineWidth',1.5)
yline(1,'--')
xticks({'2010Q1' '2011Q1' '2012Q1' '2013Q1' '2014Q1' '2015Q1' '2016Q1' '2017Q1'
'2018Q1' '2019Q1' '2020Q1' '2020Q4'})
xticklabels({'2010','2011','2012','2013','2014','2015','2016','2017','2018','2019','20
20','2021'})
xlabel('(b) Indexes')
ylabel('Index (2010Q1 = 1)')
legend({'Oslo properties (NC)', 'OSEFX (SC)', 'Oslo properties
(SC)'},'Location','northwest')

% Plot quarterly asset returns/changes (smooth costs) from 2010Q1, for dwelling
```

```
% types
plot(apartment_price_indices.Q_order, [apartment_price_changes.Q_ret_smooth_costs,
OSEFX.returns, houseprice_changes.Q_ret_smooth_costs,
smallhouse_price_changes.Q_ret_smooth_costs],'-x', 'MarkerSize', 5, 'MarkerIndices',
1:44, 'LineWidth',1.5)
yline(0,'--')
xticks({'2010Q1' '2011Q1' '2012Q1' '2013Q1' '2014Q1' '2015Q1' '2016Q1' '2017Q1'
'2018Q1' '2019Q1' '2020Q1' '2020Q4'})
xticklabels({'2010','2011','2012','2013','2014','2015','2016','2017','2018','2019','20
20','2021'})
xlabel('(a) Changes')
ylabel('Quarterly returns')
a = [cellstr(num2str(get(gca,'ytick')'*100))];
pct = char(ones(size(a,1),1)*'%');
new_yticks = [char(a),pct];
set(gca,'yticklabel',new_yticks)
legend({'Apartments (SC)', 'OSEFX (SC)', 'Houses (SC)','Small Houses
(SC)'},'Location','northwest')

% Plot cummulative asset returns (smooth costs) from 2010Q1 for dwelling types
plot(apartment_price_indices.Q_order, [apartment_price_indices.cum_Q_ret_smooth_costs,
OSEFX.cum_Q_ret_SC, houseprice_indices.cum_Q_ret_smooth_costs,
smallhouse_price_indices.cum_Q_ret_smooth_costs], 'LineWidth',1.5)
yline(1,'--')
xticks({'2010Q1' '2011Q1' '2012Q1' '2013Q1' '2014Q1' '2015Q1' '2016Q1' '2017Q1'
'2018Q1' '2019Q1' '2020Q1' '2020Q4'})
xticklabels({'2010','2011','2012','2013','2014','2015','2016','2017','2018','2019','20
20','2021'})
xlabel('(a) Indexes')
ylabel('Index (2010Q1 = 1)')
legend({'Apartments (SC)', 'OSEFX (SC)', 'Houses (SC)','Small Houses
(SC)'},'Location','northwest')

% Plot quarterly asset returns/changes (smooth costs) from 2010Q1, for
% districts
plot(apartment_price_indices.Q_order, [inner_eastern_price_changes.Q_ret_smooth_costs,
OSEFX.returns, inner_western_price_changes.Q_ret_smooth_costs,
outer_eastern_price_changes.Q_ret_smooth_costs,
outer_western_price_changes.Q_ret_smooth_costs,
outer_southern_price_changes.Q_ret_smooth_costs],'-x', 'MarkerSize', 5,
'MarkerIndices', 1:44, 'LineWidth',1.5)
yline(0,'--')
xticks({'2010Q1' '2011Q1' '2012Q1' '2013Q1' '2014Q1' '2015Q1' '2016Q1' '2017Q1'
'2018Q1' '2019Q1' '2020Q1' '2020Q4'})
xticklabels({'2010','2011','2012','2013','2014','2015','2016','2017','2018','2019','20
20','2021'})
xlabel('(a) Changes')
ylabel('Quarterly returns')
a = [cellstr(num2str(get(gca,'ytick')'*100))];
pct = char(ones(size(a,1),1)*'%');
new_yticks = [char(a),pct];
set(gca,'yticklabel',new_yticks)
legend({'Inner east (SC)', 'OSEFX (SC)', 'Inner west (SC)', 'Outer east (SC)','Outer
west (SC)','Outer south (SC)'},'Location','northwest')

% Plot cummulative asset returns (smooth costs) from 2010Q1 for districts
plot(apartment_price_indices.Q_order,
[inner_eastern_price_indices.cum_Q_ret_smooth_costs, OSEFX.cum_Q_ret_SC,
inner_western_price_indices.cum_Q_ret_smooth_costs,
outer_eastern_price_indices.cum_Q_ret_smooth_costs,
outer_western_price_indices.cum_Q_ret_smooth_costs,
outer_southern_price_indices.cum_Q_ret_smooth_costs], 'LineWidth',1.5)
yline(1,'--')
xticks({'2010Q1' '2011Q1' '2012Q1' '2013Q1' '2014Q1' '2015Q1' '2016Q1' '2017Q1'
'2018Q1' '2019Q1' '2020Q1' '2020Q4'})
xticklabels({'2010','2011','2012','2013','2014','2015','2016','2017','2018','2019','20
20','2021'})
xlabel('(a) Indexes')
ylabel('Index (2010Q1 = 1)')
legend({'Inner east (SC)', 'OSEFX (SC)', 'Inner west (SC)', 'Outer east (SC)','Outer
west (SC)','Outer south (SC)'},'Location','northwest')

%ROA(SC) TABLE
% Create a table to store returns, std, risk free, sharpe
```

```
ROA_SC_stats                            = zeros(9,7);
ROA_SC_stats                            = array2table(ROA_SC_stats);
ROA_SC_stats.Properties.VariableNames = {'Stratum', 'Return',
'std_dev','Excess_return','Sharpe_ratio', 'Skewness', 'Kurtosis'};
ROA_SC_stats.Stratum                    = categorical(ROA_SC_stats.Stratum);

ROA_SC_stats.Stratum = ROA_NC_stats.Stratum;

ROA_SC_stats(1,2:end)   = sum_stats_apartments(3,2:end);
ROA_SC_stats(2,2:end)   = sum_stats_houses(3,2:end);
ROA_SC_stats(3,2:end)   = sum_stats_smallhouses(3,2:end);
ROA_SC_stats(4,2:end)   = sum_stats_inner_eastern(3,2:end);
ROA_SC_stats(5,2:end)   = sum_stats_inner_western(3,2:end);
ROA_SC_stats(6,2:end)   = sum_stats_outer_eastern(3,2:end);
ROA_SC_stats(7,2:end)   = sum_stats_outer_western(3,2:end);
ROA_SC_stats(8,2:end)   = sum_stats_outer_southern(3,2:end);
ROA_SC_stats(9,2:end)   = sum_stats_OSEFX(1,2:end);

% Plot volatility and excess returns - Sharpe evaluation
h = gscatter(ROA_SC_stats.std_dev, ROA_SC_stats.Excess_return, ROA_SC_stats.Stratum);
for i = 2:10
    jgroup = h(i);
    jgroup.MarkerSize = 30;
end
xlabel('Volatility');
ylabel('Excess quarterly returns');
a = [cellstr(num2str(get(gca,'ytick')'*100))];
pct = char(ones(size(a,1),1)*'%');
new_yticks = [char(a),pct];
set(gca,'yticklabel',new_yticks);
b = [cellstr(num2str(get(gca,'xtick')'*100))];
pct = char(ones(size(b,1),1)*'%');
new_xticks = [char(b),pct];
set(gca,'xticklabel',new_xticks);

% Hypothesis tesing - testing difference in means between returns to real
% estate (ROA(SC)) and OSEFX
[h,p,ci,stats] = ttest2(apartment_price_changes.Q_ret_smooth_costs(2:44),
OSEFX.returns(2:44), 'Tail', 'right', 'Vartype', 'unequal')
[h,p,ci,stats] = ttest2(houseprice_changes.Q_ret_smooth_costs(2:44),
OSEFX.returns(2:44), 'Tail', 'right', 'Vartype', 'unequal')
[h,p,ci,stats] = ttest2(smallhouse_price_changes.Q_ret_smooth_costs(2:44),
OSEFX.returns(2:44), 'Tail', 'right', 'Vartype', 'unequal')
[h,p,ci,stats] = ttest2(inner_eastern_price_changes.Q_ret_smooth_costs(2:44),
OSEFX.returns(2:44), 'Tail', 'right', 'Vartype', 'unequal')
[h,p,ci,stats] = ttest2(inner_western_price_changes.Q_ret_smooth_costs(2:44),
OSEFX.returns(2:44), 'Tail', 'right', 'Vartype', 'unequal')
[h,p,ci,stats] = ttest2(outer_eastern_price_changes.Q_ret_smooth_costs(2:44),
OSEFX.returns(2:44), 'Tail', 'right', 'Vartype', 'unequal')
[h,p,ci,stats] = ttest2(outer_western_price_changes.Q_ret_smooth_costs(2:44),
OSEFX.returns(2:44), 'Tail', 'right', 'Vartype', 'unequal')
[h,p,ci,stats] = ttest2(outer_southern_price_changes.Q_ret_smooth_costs(2:44),
OSEFX.returns(2:44), 'Tail', 'right', 'Vartype', 'unequal')

% Hypothesis tesing - testing for equal variances (F-test) between returns to real
% estate (ROA(SC)) and OSEFX
[h,p,ci,stats] = vartest2(apartment_price_changes.Q_ret_smooth_costs(2:44),
OSEFX.returns(2:44), 'Tail', 'left')
[h,p,ci,stats] = vartest2(houseprice_changes.Q_ret_smooth_costs(2:44),
OSEFX.returns(2:44), 'Tail', 'left')
[h,p,ci,stats] = vartest2(smallhouse_price_changes.Q_ret_smooth_costs(2:44),
OSEFX.returns(2:44), 'Tail', 'left')
[h,p,ci,stats] = vartest2(inner_eastern_price_changes.Q_ret_smooth_costs(2:44),
OSEFX.returns(2:44), 'Tail', 'left')
[h,p,ci,stats] = vartest2(inner_western_price_changes.Q_ret_smooth_costs(2:44),
OSEFX.returns(2:44), 'Tail', 'left')
[h,p,ci,stats] = vartest2(outer_eastern_price_changes.Q_ret_smooth_costs(2:44),
OSEFX.returns(2:44), 'Tail', 'left')
[h,p,ci,stats] = vartest2(outer_western_price_changes.Q_ret_smooth_costs(2:44),
OSEFX.returns(2:44), 'Tail', 'left')
[h,p,ci,stats] = vartest2(outer_southern_price_changes.Q_ret_smooth_costs(2:44),
OSEFX.returns(2:44), 'Tail', 'left')

% Hypothesis tesing - testing for equal Sharpe ratios between returns to real
```

```
% estate (ROA(SC)) and OSEFX
ROA_SC_sharpe_test                          = zeros(9,8);
ROA_SC_sharpe_test                          = array2table(ROA_SC_sharpe_test);
ROA_SC_sharpe_test.Properties.VariableNames = {'Stratum', 'Sharpe_ratio', 'Skewness',
'Kurtosis', 'Sharpe_stDev', 'z_stat' 'PSR', 'pValue'};
ROA_SC_sharpe_test.Stratum                  = categorical(ROA_SC_sharpe_test.Stratum);
ROA_SC_sharpe_test.Stratum                  = ROA_SC_stats.Stratum;
ROA_SC_sharpe_test.Sharpe_ratio             = ROA_SC_stats.Sharpe_ratio;

ROA_SC_sharpe_test(1,3:4)   = sum_stats_apartments(1,6:7);
ROA_SC_sharpe_test(2,3:4)   = sum_stats_houses(1,6:7);
ROA_SC_sharpe_test(3,3:4)   = sum_stats_smallhouses(1,6:7);
ROA_SC_sharpe_test(4,3:4)   = sum_stats_inner_eastern(1,6:7);
ROA_SC_sharpe_test(5,3:4)   = sum_stats_inner_western(1,6:7);
ROA_SC_sharpe_test(6,3:4)   = sum_stats_outer_eastern(1,6:7);
ROA_SC_sharpe_test(7,3:4)   = sum_stats_outer_western(1,6:7);
ROA_SC_sharpe_test(8,3:4)   = sum_stats_outer_southern(1,6:7);
ROA_SC_sharpe_test(9,3:4)   = sum_stats_OSEFX(1,6:7);

for i = 1:9
    ROA_SC_sharpe_test.Sharpe_stDev(i) =
sqrt((1/43)*(1+(0.5*((ROA_SC_sharpe_test.Sharpe_ratio(i))^2))-
(ROA_SC_sharpe_test.Skewness(i)*ROA_SC_sharpe_test.Sharpe_ratio(i))+(((ROA_SC_sharpe_t
est.Kurtosis(i)-3)/4)*((ROA_SC_sharpe_test.Sharpe_ratio(i))^2))));
    ROA_SC_sharpe_test.z_stat(i) = (ROA_SC_sharpe_test.Sharpe_ratio(i) -
ROA_SC_sharpe_test.Sharpe_ratio(9))/ROA_SC_sharpe_test.Sharpe_stDev(i);
    ROA_SC_sharpe_test.PSR(i) = normcdf(ROA_SC_sharpe_test.z_stat(i));
    ROA_SC_sharpe_test.pValue(i) = 1 - ROA_SC_sharpe_test.PSR(i);
end
% 5% significance level, crit_val = 1.645;

%% ROE (SC)
% Plot quarterly equity returns/changes (smooth costs) from 2010Q1
plot(apartment_price_indices.Q_order, [property_price_changes.Q_eq_ret_smooth_costs,
OSEFX.returns, property_price_changes.Q_ret_smooth_costs],'-x', 'MarkerSize', 5,
'MarkerIndices', 1:44, 'LineWidth',1.5)
yline(0,'--')
xticks({'2010Q1' '2011Q1' '2012Q1' '2013Q1' '2014Q1' '2015Q1' '2016Q1' '2017Q1'
'2018Q1' '2019Q1' '2020Q1' '2020Q4'})
xticklabels({'2010','2011','2012','2013','2014','2015','2016','2017','2018','2019','20
20','2021'})
xlabel('(a) Changes')
ylabel('Quarterly returns')
a = [cellstr(num2str(get(gca,'ytick')'*100))];
pct = char(ones(size(a,1),1)*'%');
new_yticks = [char(a),pct];
set(gca,'yticklabel',new_yticks)
legend({'Oslo properties (ROE,SC)','OSEFX (ROE,SC)', 'Oslo properties
(ROA,SC)'},'Location','northwest')

% Plot cumulative equity returns (smooth costs) from 2010Q1
plot(apartment_price_indices.Q_order,
[property_price_indices.cum_Q_eq_ret_smooth_costs, OSEFX.cum_Q_ret_SC,
property_price_indices.cum_Q_ret_smooth_costs], 'LineWidth',1.5)
yline(1,'--')
xticks({'2010Q1' '2011Q1' '2012Q1' '2013Q1' '2014Q1' '2015Q1' '2016Q1' '2017Q1'
'2018Q1' '2019Q1' '2020Q1' '2020Q4'})
xticklabels({'2010','2011','2012','2013','2014','2015','2016','2017','2018','2019','20
20','2021'})
xlabel('(b) Indexes')
ylabel('Index (2010Q1 = 1)')
legend({'Oslo properties (ROE,SC)','OSEFX (ROE,SC)', 'Oslo properties
(ROA,SC)'},'Location','northwest')

% Plot quarterly equity returns/changes (smooth costs) from 2010Q1 for
% dwelling types
plot(apartment_price_indices.Q_order, [apartment_price_changes.Q_eq_ret_smooth_costs,
OSEFX.returns, houseprice_changes.Q_eq_ret_smooth_costs,
smallhouse_price_changes.Q_eq_ret_smooth_costs],'-x', 'MarkerSize', 5,
'MarkerIndices', 1:44, 'LineWidth',1.5)
yline(0,'--')
xticks({'2010Q1' '2011Q1' '2012Q1' '2013Q1' '2014Q1' '2015Q1' '2016Q1' '2017Q1'
'2018Q1' '2019Q1' '2020Q1' '2020Q4'})
```

```
xticklabels({'2010','2011','2012','2013','2014','2015','2016','2017','2018','2019','20
20','2021'})
xlabel('(a) Changes')
ylabel('Quarterly returns')
a = [cellstr(num2str(get(gca,'ytick')'*100))];
pct = char(ones(size(a,1),1)*'%');
new_yticks = [char(a),pct];
set(gca,'yticklabel',new_yticks)
legend({'Apartments (ROE,SC)', 'OSEFX (ROE,SC)', 'Houses (ROE,SC)','Small Houses
(ROE,SC)'},'Location','northwest')

% Plot cummulative equity returns (smooth costs) from 2010Q1 for dwelling types
plot(apartment_price_indices.Q_order,
[apartment_price_indices.cum_Q_eq_ret_smooth_costs, OSEFX.cum_Q_ret_SC,
houseprice_indices.cum_Q_eq_ret_smooth_costs,
smallhouse_price_indices.cum_Q_eq_ret_smooth_costs], 'LineWidth',1.5)
yline(1,'--')
xticks({'2010Q1' '2011Q1' '2012Q1' '2013Q1' '2014Q1' '2015Q1' '2016Q1' '2017Q1'
'2018Q1' '2019Q1' '2020Q1' '2020Q4'})
xticklabels({'2010','2011','2012','2013','2014','2015','2016','2017','2018','2019','20
20','2021'})
xlabel('(a) Indexes')
ylabel('Index (2010Q1 = 1)')
legend({'Apartments (ROE,SC)', 'OSEFX (ROE,SC)', 'Houses (ROE,SC)','Small Houses
(ROE,SC)'},'Location','northwest')

% Plot quarterly equity returns/changes (smooth costs) from 2010Q1, for
% districts
plot(apartment_price_indices.Q_order,
[inner_eastern_price_changes.Q_eq_ret_smooth_costs, OSEFX.returns,
inner_western_price_changes.Q_eq_ret_smooth_costs,
outer_eastern_price_changes.Q_eq_ret_smooth_costs,
outer_western_price_changes.Q_eq_ret_smooth_costs,
outer_southern_price_changes.Q_eq_ret_smooth_costs],'-x', 'MarkerSize', 5,
'MarkerIndices', 1:44, 'LineWidth',1.5)
yline(0,'--')
xticks({'2010Q1' '2011Q1' '2012Q1' '2013Q1' '2014Q1' '2015Q1' '2016Q1' '2017Q1'
'2018Q1' '2019Q1' '2020Q1' '2020Q4'})
xticklabels({'2010','2011','2012','2013','2014','2015','2016','2017','2018','2019','20
20','2021'})
xlabel('(a) Changes')
ylabel('Quarterly returns')
a = [cellstr(num2str(get(gca,'ytick')'*100))];
pct = char(ones(size(a,1),1)*'%');
new_yticks = [char(a),pct];
set(gca,'yticklabel',new_yticks)
legend({'Inner east (ROE,SC)', 'OSEFX (ROE,SC)', 'Inner west (ROE,SC)', 'Outer east
(ROE,SC)','Outer west (ROE,SC)','Outer south (ROE,SC)'},'Location','northwest')

% Plot cummulative equity returns (smooth costs) from 2010Q1 for districts
plot(apartment_price_indices.Q_order,
[inner_eastern_price_indices.cum_Q_eq_ret_smooth_costs, OSEFX.cum_Q_ret_SC,
inner_western_price_indices.cum_Q_eq_ret_smooth_costs,
outer_eastern_price_indices.cum_Q_eq_ret_smooth_costs,
outer_western_price_indices.cum_Q_eq_ret_smooth_costs,
outer_southern_price_indices.cum_Q_eq_ret_smooth_costs], 'LineWidth',1.5)
yline(1,'--')
xticks({'2010Q1' '2011Q1' '2012Q1' '2013Q1' '2014Q1' '2015Q1' '2016Q1' '2017Q1'
'2018Q1' '2019Q1' '2020Q1' '2020Q4'})
xticklabels({'2010','2011','2012','2013','2014','2015','2016','2017','2018','2019','20
20','2021'})
xlabel('(a) Indexes')
ylabel('Index (2010Q1 = 1)')
legend({'Inner east (ROE,SC)', 'OSEFX (ROE,SC)', 'Inner west (ROE,SC)', 'Outer east
(ROE,SC)','Outer west (ROE,SC)','Outer south (ROE,SC)'},'Location','northwest')

%ROE(SC) TABLE
% Create a table to store returns, std, risk free, sharpe
ROE_SC_stats                          = zeros(9,7);
ROE_SC_stats                          = array2table(ROE_SC_stats);
ROE_SC_stats.Properties.VariableNames = {'Stratum', 'Return',
'std_dev','Excess_return','Sharpe_ratio', 'Skewness', 'Kurtosis'};
ROE_SC_stats.Stratum                  = categorical(ROE_SC_stats.Stratum);
```

```
ROE_SC_stats.Stratum    = ROA_NC_stats.Stratum;
ROE_SC_stats(1,2:end)   = sum_stats_apartments(6,2:end);
ROE_SC_stats(2,2:end)   = sum_stats_houses(6,2:end);
ROE_SC_stats(3,2:end)   = sum_stats_smallhouses(6,2:end);
ROE_SC_stats(4,2:end)   = sum_stats_inner_eastern(6,2:end);
ROE_SC_stats(5,2:end)   = sum_stats_inner_western(6,2:end);
ROE_SC_stats(6,2:end)   = sum_stats_outer_eastern(6,2:end);
ROE_SC_stats(7,2:end)   = sum_stats_outer_western(6,2:end);
ROE_SC_stats(8,2:end)   = sum_stats_outer_southern(6,2:end);
ROE_SC_stats(9,2:end)   = sum_stats_OSEFX(1,2:end);

% Plot volatility and excess returns - Sharpe evaluation
h = gscatter(ROE_SC_stats.std_dev, ROE_SC_stats.Excess_return, ROE_SC_stats.Stratum);
for i = 2:10
    jgroup = h(i);
    jgroup.MarkerSize = 30;
end
xlabel('Volatility');
ylabel('Excess quarterly returns');
a = [cellstr(num2str(get(gca,'ytick')'*100))];
pct = char(ones(size(a,1),1)*'%');
new_yticks = [char(a),pct];
set(gca,'yticklabel',new_yticks);
b = [cellstr(num2str(get(gca,'xtick')'*100))];
pct = char(ones(size(b,1),1)*'%');
new_xticks = [char(b),pct];
set(gca,'xticklabel',new_xticks);

% Hypothesis tesing - testing difference in means between returns to real
% estate (ROE(SC)) and OSEFX
[h,p,ci,stats] = ttest2(apartment_price_changes.Q_eq_ret_smooth_costs(2:44),
OSEFX.returns(2:44), 'Tail', 'right', 'Vartype', 'unequal')
[h,p,ci,stats] = ttest2(houseprice_changes.Q_eq_ret_smooth_costs(2:44),
OSEFX.returns(2:44), 'Tail', 'right', 'Vartype', 'unequal')
[h,p,ci,stats] = ttest2(smallhouse_price_changes.Q_eq_ret_smooth_costs(2:44),
OSEFX.returns(2:44), 'Tail', 'right', 'Vartype', 'unequal')
[h,p,ci,stats] = ttest2(inner_eastern_price_changes.Q_eq_ret_smooth_costs(2:44),
OSEFX.returns(2:44), 'Tail', 'right', 'Vartype', 'unequal')
[h,p,ci,stats] = ttest2(inner_western_price_changes.Q_eq_ret_smooth_costs(2:44),
OSEFX.returns(2:44), 'Tail', 'right', 'Vartype', 'unequal')
[h,p,ci,stats] = ttest2(outer_eastern_price_changes.Q_eq_ret_smooth_costs(2:44),
OSEFX.returns(2:44), 'Tail', 'right', 'Vartype', 'unequal')
[h,p,ci,stats] = ttest2(outer_western_price_changes.Q_eq_ret_smooth_costs(2:44),
OSEFX.returns(2:44), 'Tail', 'right', 'Vartype', 'unequal')
[h,p,ci,stats] = ttest2(outer_southern_price_changes.Q_eq_ret_smooth_costs(2:44),
OSEFX.returns(2:44), 'Tail', 'right', 'Vartype', 'unequal')

% Hypothesis tesing - testing for equal variances (F-test) between returns to real
% estate (ROE(NC)) and OSEFX
[h,p,ci,stats] = vartest2(apartment_price_changes.Q_eq_ret_smooth_costs(2:44),
OSEFX.returns(2:44), 'Tail', 'left')
[h,p,ci,stats] = vartest2(houseprice_changes.Q_eq_ret_smooth_costs(2:44),
OSEFX.returns(2:44), 'Tail', 'left')
[h,p,ci,stats] = vartest2(smallhouse_price_changes.Q_eq_ret_smooth_costs(2:44),
OSEFX.returns(2:44), 'Tail', 'left')
[h,p,ci,stats] = vartest2(inner_eastern_price_changes.Q_eq_ret_smooth_costs(2:44),
OSEFX.returns(2:44), 'Tail', 'left')
[h,p,ci,stats] = vartest2(inner_western_price_changes.Q_eq_ret_smooth_costs(2:44),
OSEFX.returns(2:44), 'Tail', 'left')
[h,p,ci,stats] = vartest2(outer_eastern_price_changes.Q_eq_ret_smooth_costs(2:44),
OSEFX.returns(2:44), 'Tail', 'left')
[h,p,ci,stats] = vartest2(outer_western_price_changes.Q_eq_ret_smooth_costs(2:44),
OSEFX.returns(2:44), 'Tail', 'left')
[h,p,ci,stats] = vartest2(outer_southern_price_changes.Q_eq_ret_smooth_costs(2:44),
OSEFX.returns(2:44), 'Tail', 'left')

% Hypothesis tesing - testing for equal Sharpe ratios between returns to real
% estate (ROE(SC)) and OSEFX
ROE_SC_sharpe_test                          = zeros(9,8);
ROE_SC_sharpe_test                          = array2table(ROE_SC_sharpe_test);
ROE_SC_sharpe_test.Properties.VariableNames = {'Stratum', 'Sharpe_ratio', 'Skewness',
'Kurtosis', 'Sharpe_stDev', 'z_stat' 'PSR', 'pValue'};
ROE_SC_sharpe_test.Stratum                  = categorical(ROE_SC_sharpe_test.Stratum);
ROE_SC_sharpe_test.Stratum                  = ROE_SC_stats.Stratum;
```

```
ROE_SC_sharpe_test.Sharpe_ratio             = ROE_SC_stats.Sharpe_ratio;

ROE_SC_sharpe_test(1,3:4)    = sum_stats_apartments(1,6:7);
ROE_SC_sharpe_test(2,3:4)    = sum_stats_houses(1,6:7);
ROE_SC_sharpe_test(3,3:4)    = sum_stats_smallhouses(1,6:7);
ROE_SC_sharpe_test(4,3:4)    = sum_stats_inner_eastern(1,6:7);
ROE_SC_sharpe_test(5,3:4)    = sum_stats_inner_western(1,6:7);
ROE_SC_sharpe_test(6,3:4)    = sum_stats_outer_eastern(1,6:7);
ROE_SC_sharpe_test(7,3:4)    = sum_stats_outer_western(1,6:7);
ROE_SC_sharpe_test(8,3:4)    = sum_stats_outer_southern(1,6:7);
ROE_SC_sharpe_test(9,3:4)    = sum_stats_OSEFX(1,6:7);

for i = 1:9
    ROE_SC_sharpe_test.Sharpe_stDev(i) =
sqrt((1/43)*(1+(0.5*((ROE_SC_sharpe_test.Sharpe_ratio(i))^2))-
(ROE_SC_sharpe_test.Skewness(i)*ROE_SC_sharpe_test.Sharpe_ratio(i))+(((ROE_SC_sharpe_t
est.Kurtosis(i)-3)/4)*((ROE_SC_sharpe_test.Sharpe_ratio(i))^2))));
    ROE_SC_sharpe_test.z_stat(i) = (ROE_SC_sharpe_test.Sharpe_ratio(i) -
ROE_SC_sharpe_test.Sharpe_ratio(9))/ROE_SC_sharpe_test.Sharpe_stDev(i);
    ROE_SC_sharpe_test.PSR(i) = normcdf(ROE_SC_sharpe_test.z_stat(i));
    ROE_SC_sharpe_test.pValue(i) = 1 - ROE_SC_sharpe_test.PSR(i);
end
% 5% significance level, crit_val = 1.645;

%% CLEAR VARIABLES FROM WORKSPACE
clearvars -except Property_Data summary_table Rf Rent_Data...
                  house_set hedonic_houses housing_model...
                  price_houses uniq_house_set avg_cost_rates_houses...
                  priceindex_houses sum_stats_houses houseprice_changes...
                  houseprice_indices...
                  smallhouse_set hedonic_smallhouses smallhouse_model...
                  price_smallhouses uniq_smallhouse_set avg_cost_rates_smallhouses...
                  priceindex_smallhouses sum_stats_smallhouses...
                  smallhouse_price_changes smallhouse_price_indices...
                  apartment_set hedonic_apartments apartment_model price_apartments...
                  uniq_apartment_set avg_cost_rates_apartments
priceindex_apartments...
                  sum_stats_apartments apartment_price_changes...
                  apartment_price_indices...
                  inner_eastern_set hedonic_inner_eastern inner_eastern_model...
                  price_inner_eastern uniq_inner_eastern_set
avg_cost_rates_inner_eastern...
                  priceindex_inner_eastern sum_stats_inner_eastern
inner_eastern_price_changes...
                  inner_eastern_price_indices...
                  inner_western_set hedonic_inner_western inner_western_model...
                  price_inner_western uniq_inner_western_set
avg_cost_rates_inner_western...
                  priceindex_inner_western sum_stats_inner_western
inner_western_price_changes...
                  inner_western_price_indices...
                  outer_eastern_set hedonic_outer_eastern outer_eastern_model...
                  price_outer_eastern uniq_outer_eastern_set
avg_cost_rates_outer_eastern...
                  priceindex_outer_eastern sum_stats_outer_eastern
outer_eastern_price_changes...
                  outer_eastern_price_indices...
                  outer_western_set hedonic_outer_western outer_western_model...
                  price_outer_western uniq_outer_western_set
avg_cost_rates_outer_western...
                  priceindex_outer_western sum_stats_outer_western
outer_western_price_changes...
                  outer_western_price_indices...
                  outer_southern_set hedonic_outer_southern outer_southern_model...
                  price_outer_southern uniq_outer_southern_set
avg_cost_rates_outer_southern...
                  priceindex_outer_southern sum_stats_outer_southern
outer_southern_price_changes...
                  outer_southern_price_indices...
                  property_price_changes property_price_indices hedonic_total...
                  ROA_NC_stats ROE_NC_stats ROA_TC_stats ROE_TC_stats...
                  ROA_SC_stats ROE_SC_stats sum_stats_OSEFX OSEFX...
                  ROA_NC_sharpe_test ROA_TC_sharpe_test ROA_SC_sharpe_test...
                  ROE_SC_sharpe_test
```

toc