Master thesis: The Green Bond Premium: An Extension with Use of Proceeds

Endre Østerud & Anders Rasmussen

Step 1: Data Structuring & Matching

In [2]:

```
%reset
import pandas as pd
import numpy as np
from numpy import *
import datetime as dt
from scipy.interpolate import InterpolatedUnivariateSpline
import matplotlib.pyplot as plt
from tqdm import tqdm
import scipy
```

```
Once deleted, variables cannot be recovered. Proceed (y/[n])? y
```

Importing the ratings from the different agencies to make a average rating of each bond

In [3]:

```
Ratings = pd.read_excel("Ratings.xlsx")
```

In [4]:

```
Ratings.head()
```

Out[4]:

	Moody's	S&P	FITCH	JCR	R&I
0	Aaa	AAA	AAA	AAA	AAA
1	Aa1	AA+	AA+	AA+	AA+
2	Aa2	AA	AA	AA	AA
3	Aa3	AA-	AA-	AA-	AA-
4	A1	A+	A+	A+	A+

In [5]:

```
Ratings = Ratings[["Moody's", "S&P", "FITCH"]] #since we only use Moodys, Fitch or S&P
```

In [6]:

```
Ratings.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20 entries, 0 to 19
Data columns (total 3 columns):
Moody's    20 non-null object
S&P        20 non-null object
FITCH      20 non-null object
dtypes: object(3)
memory usage: 560.0+ bytes
```

Importing a dataset with all the bonds from CBI, but only containing use of proceeds and if it is certified by a third party (1 if certified, 0 if not),

In [7]:

```
cbi = pd.read_excel("cbi_info.xlsx")
```

In [8]:

```
cbi.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3951 entries, 0 to 3950
Data columns (total 3 columns):
ISIN               3844 non-null object
Use of proceeds    3945 non-null object
CBI certified      3951 non-null int64
dtypes: int64(1), object(2)
memory usage: 92.7+ KB
```

In [9]:

```
cbi.nunique() ## Appears to be duplicates in the database, we will remove this
```

Out[9]:

```
ISIN               3761
Use of proceeds     103
CBI certified         2
dtype: int64
```

In [10]:

```
cbi.drop_duplicates(subset = ["ISIN"], inplace = True)
cbi.reset_index(drop=True, inplace = True) ## need to reset the index after re
moving rows.
```

In [11]:

```
cbi.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3762 entries, 0 to 3761
Data columns (total 3 columns):
ISIN                3761 non-null object
Use of proceeds     3758 non-null object
CBI certified       3762 non-null int64
dtypes: int64(1), object(2)
memory usage: 88.2+ KB
```

All the green bonds from Datastream

In [12]:

```
gb_all = pd.read_excel("Green_bond_all.xlsx")
```

In [13]:

```
gb_all.head()
```

Out[13]:

	Issuer Name	Ticker	Coupon	Maturity	Issue Date	ISIN	Preferred
0	City Developments Ltd	CTDM	1.98	2019-04-18 00:00:00	2017-04-18 00:00:00	SG7AG5000005	SGCTDM0
1	European Investment Bank	EIB	3	2019-04-23 00:00:00	2012-04-23 00:00:00	XS0773059042	XS07730
2	Kommuninvesti Sverige AB	KOMEFS	1.5	2019-04-23 00:00:00	2016-03-22 00:00:00	XS1383831648	SE138383
3	Kommuninvesti Sverige AB	KOMEFS	1.5	2019-04-23 00:00:00	2016-03-22 00:00:00	US50046PAU93	50046PAU
4	European Bank for Reconstruction and Development	EBRD	6.88	2019-04-24 00:00:00	2015-04-23 00:00:00	XS1208591880	XS120859

5 rows × 38 columns

In [14]:

```
gb_all.info() # Some bonds lack ISIN from Datastream, we remove these.
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1332 entries, 0 to 1331
Data columns (total 38 columns):
Issuer Name                          1332 non-null object
Ticker                               1332 non-null object
Coupon                               1332 non-null object
Maturity                             1315 non-null object
Issue Date                           1332 non-null object
ISIN                                 1310 non-null object
Preferred RIC                        1297 non-null object
Details                              1329 non-null object
Amount Issued                        1332 non-null object
Principal Currency                   1332 non-null object
Country of Issue                     1327 non-null object
Issuer Type                          1332 non-null object
Instrument Type                      1332 non-null object
Bond Grade                           626 non-null object
Call Type                            241 non-null object
Coupon Type                          1330 non-null object
Market of Issue                      1332 non-null object
Seniority                            1332 non-null object
Current Coupon Class                 1332 non-null object
Coupon Frequency                     1293 non-null object
Green Bond                           1332 non-null object
Use of Proceeds                      1161 non-null object
Issuer Country                       1332 non-null object
TRPS / Composite Price               1332 non-null object
Yield Spread (OTR) to Maturity       1332 non-null object
Callable                             1332 non-null object
Amount Outstanding (USD)             1332 non-null object
Amount Issued (USD)                  1332 non-null object
Amount Outstanding                   1332 non-null object
Price Date                           1168 non-null datetime64[ns]
Yield to Maturity                    1332 non-null object
Next Coupon Payment Date             1225 non-null object
Dual Currency                        1332 non-null object
Sector                               1332 non-null object
Tenor (Beta)                         1315 non-null object
Moodys Rating                        581 non-null object
Fitchs Rating                        327 non-null object
SP Rating                            473 non-null object
dtypes: datetime64[ns](1), object(37)
memory usage: 395.5+ KB
```

In [15]:

```
gb_all.dropna(subset=['ISIN'], inplace = True)
gb_all.reset_index(drop = True, inplace = True)
```

In [16]:

```
gb_all_in_cbi = pd.merge(cbi,gb_all, on=['ISIN'])
```

In [17]:

```
gb_all_in_cbi.info() ## Of the 1332 bonds labeled as green in Datastream, 912 was in the Cbi Database.
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 890 entries, 0 to 889
Data columns (total 40 columns):
ISIN                                890 non-null object
Use of proceeds                     887 non-null object
CBI certified                       890 non-null int64
Issuer Name                         890 non-null object
Ticker                              890 non-null object
Coupon                              890 non-null object
Maturity                            883 non-null object
Issue Date                          890 non-null object
Preferred RIC                       868 non-null object
Details                             889 non-null object
Amount Issued                       890 non-null object
Principal Currency                  890 non-null object
Country of Issue                    886 non-null object
Issuer Type                         890 non-null object
Instrument Type                     890 non-null object
Bond Grade                          493 non-null object
Call Type                           158 non-null object
Coupon Type                         890 non-null object
Market of Issue                     890 non-null object
Seniority                           890 non-null object
Current Coupon Class                890 non-null object
Coupon Frequency                    861 non-null object
Green Bond                          890 non-null object
Use of Proceeds                     818 non-null object
Issuer Country                      890 non-null object
TRPS / Composite Price              890 non-null object
Yield Spread (OTR) to Maturity      890 non-null object
Callable                            890 non-null object
Amount Outstanding (USD)            890 non-null object
Amount Issued (USD)                 890 non-null object
Amount Outstanding                  890 non-null object
Price Date                          776 non-null datetime64[ns]
Yield to Maturity                   890 non-null object
Next Coupon Payment Date            810 non-null object
Dual Currency                       890 non-null object
Sector                              890 non-null object
Tenor (Beta)                        883 non-null object
Moodys Rating                       472 non-null object
Fitchs Rating                       250 non-null object
SP Rating                           368 non-null object
dtypes: datetime64[ns](1), int64(1), object(38)
memory usage: 285.1+ KB
```

Here we transform the ratings into integers where AAA/AAA/Aaa = 0, AA+/AA+/Aa1 = 1 etc. and take the average rounded to the nearest integer. We will use the use the S&P standard for the averaged rating.

In [18]:

```
sample = gb_all_in_cbi[["SP Rating", "Fitchs Rating", "Moodys Rating"]]
sample.head()
```

Out[18]:

	SP Rating	Fitchs Rating	Moodys Rating
0	AAA	NaN	NR
1	AAA	AAA	Aaa
2	AAA	AAA	Aaa
3	NaN	NaN	Aaa
4	AAA	NaN	Aaa

In [19]:

```
a = []
b=[]
c=[]
for i in range(len(sample)):
        if Ratings[Ratings["Moody's"] == sample.iloc[i,2]].describe().iloc[1,1
] != 0:
            a.append(Ratings[Ratings["Moody's"] == sample.iloc[i,2]].index.ast
ype(int)[0])
        else:
            a.append(np.nan)

        if Ratings[Ratings["S&P"] == sample.iloc[i,0]].describe().iloc[1,1] !=
0:
            b.append(Ratings[Ratings["S&P"] == sample.iloc[i,0]].index.astype(
int)[0])
        else:
            b.append(np.nan)

        if Ratings[Ratings["FITCH"] == sample.iloc[i,1]].describe().iloc[1,1]
!= 0:
            c.append(Ratings[Ratings["FITCH"] == sample.iloc[i,1]].index.astyp
e(int)[0])
        else:
            c.append(np.nan)


sample['S&P'] = b
sample['Fitch'] = c
sample['Moodys'] = a
```

```
/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:22: S
ettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pan
das-docs/stable/indexing.html#indexing-view-versus-copy
/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:23: S
ettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pan
das-docs/stable/indexing.html#indexing-view-versus-copy
/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:24: S
ettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pan
das-docs/stable/indexing.html#indexing-view-versus-copy
```

In [20]:

```
sample = sample[["S&P", "Fitch", "Moodys"]]
sample.head()
```

Out[20]:

	S&P	Fitch	Moodys
0	0.0	NaN	NaN
1	0.0	0.0	0.0
2	0.0	0.0	0.0
3	NaN	NaN	0.0
4	0.0	NaN	0.0

In [21]:

```
average = sample.mean(axis = 1).round()
average.head()
```

Out[21]:

```
0    0.0
1    0.0
2    0.0
3    0.0
4    0.0
dtype: float64
```

In [22]:

```
Rdic = Ratings["S&P"].to_dict()
```

In [23]:

```
sample["Ratings"] = average.map(Rdic)
```

In [24]:

```
gb_all_in_cbi["Rating"] = sample["Ratings"]
gb_all_in_cbi.drop(columns = ["SP Rating" ,"Fitchs Rating", "Moodys Rating"],
inplace = True)
```

In [25]:

```
gb_all_in_cbi.head(5)
```

Out[25]:

	ISIN	Use of proceeds	CBI certified	Issuer Name	Ticker	Coupon	Maturity	
0	XS0490636791	Energy, Buildings, Transport, Water, Land Use,...	0	International Bank for Reconstruction and Deve...	IBRD	8	2020-03-02 00:00:00	201 03- 00:
1	XS0490347415	Energy, Buildings, Transport, Water, Land Use,...	0	International Bank for Reconstruction and Deve...	IBRD	7.5	2020-03-05 00:00:00	201 03- 00:
2	XS0491921937	Energy, Buildings, Transport, Water, Land Use,...	0	International Bank for Reconstruction and Deve...	IBRD	0.875	2020-03-09 00:00:00	201 03- 00:
3	XS0536850216	Energy, Buildings, Water,	0	African Development Bank	AFDB	0.5	2020-09-29 00:00:00	201 09- 00:
4	XS0554265032	Energy, Buildings, Transport, Water, Land Use,...	0	International Bank for Reconstruction and Deve...	IBRD	3.5	2020-11-02 00:00:00	201 11- 00:

5 rows × 38 columns

In [26]:

```
gb_all_in_cbi.to_excel('gb_all_cbi.xlsx', index=False )
```

Green bond sample (Investment grade with our restrictions)

In [27]:

```
gb_invest = pd.read_excel("Green_investmentgrade.xlsx")
```

In [28]:

```
gb_invest.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 481 entries, 0 to 480
Data columns (total 30 columns):
Issuer Name                 481 non-null object
Ticker                      481 non-null object
Coupon                      481 non-null float64
Maturity                    481 non-null datetime64[ns]
Issue Date                  481 non-null datetime64[ns]
ISIN                        481 non-null object
Details                     480 non-null object
Amount Issued               481 non-null int64
Principal Currency          481 non-null object
Country of Issue            477 non-null object
Issuer Type                 481 non-null object
Instrument Type             481 non-null object
Bond Grade                  480 non-null object
Call Type                   105 non-null object
Coupon Type                 481 non-null object
Market of Issue             481 non-null object
Seniority                   481 non-null object
Current Coupon Class        481 non-null object
Coupon Frequency            479 non-null object
Issuer Country              481 non-null object
Callable                    481 non-null object
Amount Outstanding (USD)    481 non-null int64
Amount Issued (USD)         481 non-null int64
Amount Outstanding          481 non-null int64
Green Bond                  481 non-null object
Dual Currency               481 non-null object
Sector                      481 non-null object
SP Rating                   371 non-null object
Fitch Rating                248 non-null object
Moodys Rating               414 non-null object
dtypes: datetime64[ns](2), float64(1), int64(4), object(23)
memory usage: 112.8+ KB
```

In [29]:

```
gb_invest.rename(columns={'Green Bond ': 'Green Bond'}, inplace=True)
```

In [30]:

```
gb_invest.head()
```

Out[30]:

	Issuer Name	Ticker	Coupon	Maturity	Issue Date	ISIN	Details
0	Credit Agricole Corporate and Investment Bank SA	CAGRAB	2.960	2019-05-28	2014-11-28	XS1140834455	Sr Note
1	Rodamco Sverige AB	UNBPS	2.250	2019-06-03	2014-06-03	XS1073076991	Sr Note Freq Annual
2	Credit Agricole Corporate and Investment Bank SA	CAGRAB	8.000	2019-06-03	2016-06-03	XS1367225817	Sr Note
3	International Bank for Reconstruction and Deve...	IBRD	1.375	2019-06-23	2014-06-23	XS1078475024	ReOpened
4	Nederlandse Waterschapsbank NV	NDLWR	0.625	2019-07-03	2014-07-03	XS1083955911	Sr Note Freq Annual

5 rows × 30 columns

In [31]:

```
gb_invest = pd.merge(cbi,gb_invest, on=['ISIN'])
```

In [32]:

```
gb_invest.info() ## we lost 75 bonds,
#since they were not registered in the cbi-database, we now have 406 bonds, wh
ere
#one of them dont have use of proceeds
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 406 entries, 0 to 405
Data columns (total 32 columns):
ISIN                        406 non-null object
Use of proceeds             405 non-null object
CBI certified               406 non-null int64
Issuer Name                 406 non-null object
Ticker                      406 non-null object
Coupon                      406 non-null float64
Maturity                    406 non-null datetime64[ns]
Issue Date                  406 non-null datetime64[ns]
Details                     405 non-null object
Amount Issued               406 non-null int64
Principal Currency          406 non-null object
Country of Issue            403 non-null object
Issuer Type                 406 non-null object
Instrument Type             406 non-null object
Bond Grade                  405 non-null object
Call Type                   77 non-null object
Coupon Type                 406 non-null object
Market of Issue             406 non-null object
Seniority                   406 non-null object
Current Coupon Class        406 non-null object
Coupon Frequency            404 non-null object
Issuer Country              406 non-null object
Callable                    406 non-null object
Amount Outstanding (USD)    406 non-null int64
Amount Issued (USD)         406 non-null int64
Amount Outstanding          406 non-null int64
Green Bond                  406 non-null object
Dual Currency               406 non-null object
Sector                      406 non-null object
SP Rating                   310 non-null object
Fitch Rating                206 non-null object
Moodys Rating               352 non-null object
dtypes: datetime64[ns](2), float64(1), int64(5), object(24)
memory usage: 104.7+ KB
```

In [33]:

```
sample = gb_invest[["SP Rating", "Fitch Rating", "Moodys Rating"]]
sample.head()
```

Out[33]:

	SP Rating	Fitch Rating	Moodys Rating
0	AAA	NaN	NR
1	AAA	AAA	Aaa
2	AAA	AAA	Aaa
3	NaN	NaN	Aaa
4	AAA	NaN	Aaa

In [34]:

```
sample.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 406 entries, 0 to 405
Data columns (total 3 columns):
SP Rating        310 non-null object
Fitch Rating     206 non-null object
Moodys Rating    352 non-null object
dtypes: object(3)
memory usage: 12.7+ KB
```

In [35]:

```
a = []
b=[]
c=[]
for i in range(len(sample)):
        if Ratings[Ratings["Moody's"] == sample.iloc[i,2]].describe().iloc[1,1
] != 0:
            a.append(Ratings[Ratings["Moody's"] == sample.iloc[i,2]].index.ast
ype(int)[0])
        else:
            a.append(np.nan)

        if Ratings[Ratings["S&P"] == sample.iloc[i,0]].describe().iloc[1,1] !=
0:
            b.append(Ratings[Ratings["S&P"] == sample.iloc[i,0]].index.astype(
int)[0])
        else:
            b.append(np.nan)

        if Ratings[Ratings["FITCH"] == sample.iloc[i,1]].describe().iloc[1,1]
!= 0:
            c.append(Ratings[Ratings["FITCH"] == sample.iloc[i,1]].index.astyp
e(int)[0])
        else:
            c.append(np.nan)



sample['S&P'] = b
sample['Fitch'] = c
sample['Moodys'] = a
```

```
/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:22: S
ettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pan
das-docs/stable/indexing.html#indexing-view-versus-copy
/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:23: S
ettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pan
das-docs/stable/indexing.html#indexing-view-versus-copy
/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:24: S
ettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pan
das-docs/stable/indexing.html#indexing-view-versus-copy
```

In [36]:

```
sample = sample[["S&P", "Fitch", "Moodys"]]
sample.head()
```

Out[36]:

	S&P	Fitch	Moodys
0	0.0	NaN	NaN
1	0.0	0.0	0.0
2	0.0	0.0	0.0
3	NaN	NaN	0.0
4	0.0	NaN	0.0

In [37]:

```
sample.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 406 entries, 0 to 405
Data columns (total 3 columns):
S&P       307 non-null float64
Fitch     200 non-null float64
Moodys    342 non-null float64
dtypes: float64(3)
memory usage: 32.7 KB
```

In [38]:

```
average = sample.mean(axis = 1).round()
average.head(6)
```

Out[38]:

```
0    0.0
1    0.0
2    0.0
3    0.0
4    0.0
5    2.0
dtype: float64
```

In [39]:

```
Rdic = Ratings["S&P"].to_dict()
```

In [40]:

```
sample["Ratings"] = average.map(Rdic)
```

In [41]:

```
sample.head()
```

Out[41]:

	S&P	Fitch	Moodys	Ratings
0	0.0	NaN	NaN	AAA
1	0.0	0.0	0.0	AAA
2	0.0	0.0	0.0	AAA
3	NaN	NaN	0.0	AAA
4	0.0	NaN	0.0	AAA

In [42]:

```
gb_invest["Rating"] = sample["Ratings"]
gb_invest.drop(columns = ["SP Rating" ,"Fitch Rating", "Moodys Rating"], inpla
ce = True)
```

In [43]:

```
gb_invest.head()
```

Out[43]:

	ISIN	Use of proceeds	CBI certified	Issuer Name	Ticker	Coupon	Maturity	Iss Da
0	XS0490636791	Energy, Buildings, Transport, Water, Land Use,...	0	International Bank for Reconstruction and Deve...	IBRD	8.000	2020-03-02	201 03-
1	XS0490347415	Energy, Buildings, Transport, Water, Land Use,...	0	International Bank for Reconstruction and Deve...	IBRD	7.500	2020-03-05	201 03-
2	XS0491921937	Energy, Buildings, Transport, Water, Land Use,...	0	International Bank for Reconstruction and Deve...	IBRD	0.875	2020-03-09	201 03-
3	XS0536850216	Energy, Buildings, Water,	0	African Development Bank	AFDB	0.500	2020-09-29	201 09-
4	XS0554265032	Energy, Buildings, Transport, Water, Land Use,...	0	International Bank for Reconstruction and Deve...	IBRD	3.500	2020-11-02	201 11-

5 rows × 30 columns

Here we create dummies of the use of proceeds from CBI

In [44]:

```
pd.get_dummies(gb_invest["Use of proceeds "]).head()
```

Out[44]:

	Asia-Pacific	Buildings,	Buildings,	Buildings, Transport,	Buildings, Transport, Water,	Buildings, Transport, Water, Waste,	Buildings, Transport, Water, Waste, Land Use,	Bu Tra Lai Adap
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0

5 rows × 50 columns

Since it counts every combination as a dummy, this is more or less unusable, we create a dummy for each single category

In [45]:

```
dum = gb_invest["Use of proceeds "].str.split(",", expand = True).stack().str.
get_dummies().sum(level=0)

dum.drop(columns = [" "], inplace = True)

dum["Buildings"] = dum["Buildings"] + dum[" Buildings"]
dum.drop(columns = [" Buildings"], inplace = True)

dum["Transport"] = dum["Transport"] + dum[" Transport"]
dum.drop(columns = [" Transport"], inplace = True)

dum["Water"] = dum["Water"] + dum[" Water"]
dum.drop(columns = [" Water"], inplace = True)

dum["Land Use"] = dum["Land Use"] + dum[" Land Use"]
dum.drop(columns = [" Land Use"], inplace = True)
```

In [46]:

```
dum.head()
```

Out[46]:

	Adaptation	Industry	Waste	Asia-Pacific	Buildings	Energy	Land Use	Transport	Water
0	1	0	0	0	1	1	1	1	1
1	1	0	0	0	1	1	1	1	1
2	1	0	0	0	1	1	1	1	1
3	0	0	0	0	1	1	0	0	1
4	1	0	0	0	1	1	1	1	1

In [47]:

```
gb_invest = pd.concat([gb_invest, dum], axis=1)
```

In [48]:

```
gb_invest.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 406 entries, 0 to 405
Data columns (total 39 columns):
ISIN                          406 non-null object
Use of proceeds               405 non-null object
CBI certified                 406 non-null int64
Issuer Name                   406 non-null object
Ticker                        406 non-null object
Coupon                        406 non-null float64
Maturity                      406 non-null datetime64[ns]
Issue Date                    406 non-null datetime64[ns]
Details                       405 non-null object
Amount Issued                 406 non-null int64
Principal Currency            406 non-null object
Country of Issue              403 non-null object
Issuer Type                   406 non-null object
Instrument Type               406 non-null object
Bond Grade                    405 non-null object
Call Type                     77 non-null object
Coupon Type                   406 non-null object
Market of Issue               406 non-null object
Seniority                     406 non-null object
Current Coupon Class          406 non-null object
Coupon Frequency              404 non-null object
Issuer Country                406 non-null object
Callable                      406 non-null object
Amount Outstanding (USD)      406 non-null int64
Amount Issued (USD)           406 non-null int64
Amount Outstanding            406 non-null int64
Green Bond                    406 non-null object
Dual Currency                 406 non-null object
Sector                        406 non-null object
Rating                        406 non-null object
 Adaptation                   405 non-null float64
 Industry                     405 non-null float64
 Waste                        405 non-null float64
Asia-Pacific                  405 non-null float64
Buildings                     405 non-null float64
Energy                        405 non-null float64
Land Use                      405 non-null float64
Transport                     405 non-null float64
Water                         405 non-null float64
dtypes: datetime64[ns](2), float64(10), int64(5), object(22)
memory usage: 126.9+ KB
```

We now have dummies for each category in use of proceeds, we can drop the "use of proceeds" column.

In [49]:

```
gb_invest.drop(["Use of proceeds "], axis = 1, inplace = True)
```

In [50]:

```
gb_invest.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 406 entries, 0 to 405
Data columns (total 38 columns):
ISIN                          406 non-null object
CBI certified                 406 non-null int64
Issuer Name                   406 non-null object
Ticker                        406 non-null object
Coupon                        406 non-null float64
Maturity                      406 non-null datetime64[ns]
Issue Date                    406 non-null datetime64[ns]
Details                       405 non-null object
Amount Issued                 406 non-null int64
Principal Currency            406 non-null object
Country of Issue              403 non-null object
Issuer Type                   406 non-null object
Instrument Type               406 non-null object
Bond Grade                    405 non-null object
Call Type                     77 non-null object
Coupon Type                   406 non-null object
Market of Issue               406 non-null object
Seniority                     406 non-null object
Current Coupon Class          406 non-null object
Coupon Frequency              404 non-null object
Issuer Country                406 non-null object
Callable                      406 non-null object
Amount Outstanding (USD)      406 non-null int64
Amount Issued (USD)           406 non-null int64
Amount Outstanding            406 non-null int64
Green Bond                    406 non-null object
Dual Currency                 406 non-null object
Sector                        406 non-null object
Rating                        406 non-null object
 Adaptation                   405 non-null float64
 Industry                     405 non-null float64
 Waste                        405 non-null float64
Asia-Pacific                  405 non-null float64
Buildings                     405 non-null float64
Energy                        405 non-null float64
Land Use                      405 non-null float64
Transport                     405 non-null float64
Water                         405 non-null float64
dtypes: datetime64[ns](2), float64(10), int64(5), object(21)
memory usage: 123.7+ KB
```

In [51]:

```
gb_invest.to_excel('gbinfo.xlsx', index=False ) ## Exporting this to excel to use later
```

Conventional Bonds

In [52]:

```
cb_invest = pd.read_excel("greybonds_investment_grade.xlsx")
```

In [53]:

```
cb_invest.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 58581 entries, 0 to 58580
Data columns (total 28 columns):
Issuer Name             58581 non-null object
Ticker                  58526 non-null object
Coupon                  58581 non-null object
Maturity                58581 non-null object
Issue Date              58581 non-null object
ISIN                    58521 non-null object
Preferred RIC           56316 non-null object
Details                 58563 non-null object
Amount Issued           58581 non-null object
Principal Currency      58581 non-null object
Country of Issue        58547 non-null object
Issuer Type             58581 non-null object
Instrument Type         58581 non-null object
Bond Grade              58581 non-null object
Call Type               22308 non-null object
Coupon Type             58581 non-null object
Dual Currency           58581 non-null object
Market of Issue         58572 non-null object
Coupon Frequency        54681 non-null object
Seniority               58581 non-null object
Green Bond              58581 non-null object
Yield to Maturity       58581 non-null object
Issuer Country          58581 non-null object
Amount Issued (USD)     58581 non-null object
Callable                58581 non-null object
SP Rating               43041 non-null object
Moodys Rating           50731 non-null object
Fitch Rating            30061 non-null object
dtypes: object(28)
memory usage: 12.5+ MB
```

In [54]:

```
cb_invest.head()
```

Out[54]:

	Issuer Name	Ticker	Coupon	Maturity	Issue Date	ISIN	Preferred R
0	Aichi, Prefecture of	AICHI	1.48	2019-03-27 00:00:00	2009-03-27 00:00:00	JP2230001931	JP20100108=
1	Societe Generale SCF SA	SOGAN	5	2019-03-27 00:00:00	2009-03-27 00:00:00	FR0010742908	FR001074290
2	Federal Home Loan Mortgage Corp	FHLMC	3.75	2019-03-27 00:00:00	2009-03-27 00:00:00	US3137EACA57	US042146684
3	African Development Bank	AFDB	0.5	2019-03-27 00:00:00	2012-03-27 00:00:00	XS0754328796	XS075432879
4	Federal National Mortgage Association	FNMA	1.4	2019-03-27 00:00:00	2013-03-27 00:00:00	US3136G1HS19	3136G1HS1=

5 rows × 28 columns

In [55]:

```
cb_invest.drop_duplicates(subset = ["ISIN"], inplace = True) ## just a precaut
ion since it is a large dataset
```

In [56]:

```
cb_invest.info() ## it was a few duplicates.
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 58522 entries, 0 to 58580
Data columns (total 28 columns):
Issuer Name             58522 non-null object
Ticker                  58467 non-null object
Coupon                  58522 non-null object
Maturity                58522 non-null object
Issue Date              58522 non-null object
ISIN                    58521 non-null object
Preferred RIC           56275 non-null object
Details                 58504 non-null object
Amount Issued           58522 non-null object
Principal Currency      58522 non-null object
Country of Issue        58488 non-null object
Issuer Type             58522 non-null object
Instrument Type         58522 non-null object
Bond Grade              58522 non-null object
Call Type               22301 non-null object
Coupon Type             58522 non-null object
Dual Currency           58522 non-null object
Market of Issue         58513 non-null object
Coupon Frequency        54624 non-null object
Seniority               58522 non-null object
Green Bond              58522 non-null object
Yield to Maturity       58522 non-null object
Issuer Country          58522 non-null object
Amount Issued (USD)     58522 non-null object
Callable                58522 non-null object
SP Rating               42997 non-null object
Moodys Rating           50694 non-null object
Fitch Rating            30053 non-null object
dtypes: object(28)
memory usage: 12.9+ MB
```

We have to convert the ratings here, we will use the same method as we did with the green bonds.

In [57]:

```
sample = cb_invest[["SP Rating", "Fitch Rating", "Moodys Rating"]]
sample.head()
```

Out[57]:

	SP Rating	Fitch Rating	Moodys Rating
0	A+	NaN	NaN
1	AAA	WD	Aaa
2	AA+	AAA	Aaa
3	AAA	AAA	Aaa
4	AA+	AAA	Aaa

In [58]:

```
a = []
b=[]
c=[]
for i in range(len(sample)):
        if Ratings[Ratings["Moody's"] == sample.iloc[i,2]].describe().iloc[1,1
] != 0:
            a.append(Ratings[Ratings["Moody's"] == sample.iloc[i,2]].index.ast
ype(int)[0])
        else:
            a.append(np.nan)

        if Ratings[Ratings["S&P"] == sample.iloc[i,0]].describe().iloc[1,1] !=
0:
            b.append(Ratings[Ratings["S&P"] == sample.iloc[i,0]].index.astype(
int)[0])
        else:
            b.append(np.nan)

        if Ratings[Ratings["FITCH"] == sample.iloc[i,1]].describe().iloc[1,1]
!= 0:
            c.append(Ratings[Ratings["FITCH"] == sample.iloc[i,1]].index.astyp
e(int)[0])
        else:
            c.append(np.nan)


sample['S&P'] = b
sample['Fitch'] = c
sample['Moodys'] = a
```

```
/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:21: S
ettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pan
das-docs/stable/indexing.html#indexing-view-versus-copy
/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:22: S
ettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pan
das-docs/stable/indexing.html#indexing-view-versus-copy
/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:23: S
ettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pan
das-docs/stable/indexing.html#indexing-view-versus-copy
```

In [59]:

```
sample = sample[["S&P", "Fitch", "Moodys"]]
sample.head()
```

Out[59]:

	S&P	Fitch	Moodys
0	4.0	NaN	NaN
1	0.0	NaN	0.0
2	1.0	0.0	0.0
3	0.0	0.0	0.0
4	1.0	0.0	0.0

In [60]:

```
average = sample.mean(axis = 1).round()
average.head(6)
```

Out[60]:

```
0    4.0
1    0.0
2    0.0
3    0.0
4    0.0
5    0.0
dtype: float64
```

In [61]:

```
Rdic = Ratings["S&P"].to_dict()
```

In [62]:

```
sample["Ratings"] = average.map(Rdic)
```

In [63]:

```
cb_invest["Rating"] = sample["Ratings"]
cb_invest.drop(columns = ["SP Rating" ,"Fitch Rating", "Moodys Rating"], inpla
ce = True)
```

In [64]:

```
cb_invest.head(5)
```

Out[64]:

	Issuer Name	Ticker	Coupon	Maturity	Issue Date	ISIN	Preferred R
0	Aichi, Prefecture of	AICHI	1.48	2019-03-27 00:00:00	2009-03-27 00:00:00	JP2230001931	JP20100108=
1	Societe Generale SCF SA	SOGAN	5	2019-03-27 00:00:00	2009-03-27 00:00:00	FR0010742908	FR001074290
2	Federal Home Loan Mortgage Corp	FHLMC	3.75	2019-03-27 00:00:00	2009-03-27 00:00:00	US3137EACA57	US042146684
3	African Development Bank	AFDB	0.5	2019-03-27 00:00:00	2012-03-27 00:00:00	XS0754328796	XS075432879
4	Federal National Mortgage Association	FNMA	1.4	2019-03-27 00:00:00	2013-03-27 00:00:00	US3136G1HS19	3136G1HS1=

5 rows × 26 columns

Matching bonds

We match the respective green bonds with the conventional bonds based on our criteria

In [65]:

```
df_green = gb_invest[["ISIN","Issue Date", "Coupon Type", "Issuer Name", "Prin
cipal Currency", "Amount Issued (USD)", "Maturity", "Rating", "Seniority", "Gr
een Bond", "Call Type"]]
df_grey = cb_invest[["ISIN", "Coupon Type", "Issue Date", "Issuer Name", "Prin
cipal Currency", "Amount Issued (USD)", "Maturity", "Rating", "Seniority", "Gr
een Bond", "Call Type"]]
```

In [66]:

```
df_green.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 406 entries, 0 to 405
Data columns (total 11 columns):
ISIN                  406 non-null object
Issue Date            406 non-null datetime64[ns]
Coupon Type           406 non-null object
Issuer Name           406 non-null object
Principal Currency    406 non-null object
Amount Issued (USD)   406 non-null int64
Maturity              406 non-null datetime64[ns]
Rating                406 non-null object
Seniority             406 non-null object
Green Bond            406 non-null object
Call Type             77 non-null object
dtypes: datetime64[ns](2), int64(1), object(8)
memory usage: 38.1+ KB
```

In [67]:

```
df_grey.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 58522 entries, 0 to 58580
Data columns (total 11 columns):
ISIN                  58521 non-null object
Coupon Type           58522 non-null object
Issue Date            58522 non-null object
Issuer Name           58522 non-null object
Principal Currency    58522 non-null object
Amount Issued (USD)   58522 non-null object
Maturity              58522 non-null object
Rating                58522 non-null object
Seniority             58522 non-null object
Green Bond            58522 non-null object
Call Type             22301 non-null object
dtypes: object(11)
memory usage: 7.9+ MB
```

In [68]:

```
df_grey["Amount Issued (USD)"] = pd.to_numeric(df_grey["Amount Issued (USD)"],
errors="coerce")
```

```
/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:2: Se
ttingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pan
das-docs/stable/indexing.html#indexing-view-versus-copy
```

In [69]:

```
df_green["Call Type"].fillna('No', inplace=True)
df_grey["Call Type"].fillna('No', inplace=True)
```

```
/anaconda3/lib/python3.6/site-packages/pandas/core/generic.py:5430
: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pan
das-docs/stable/indexing.html#indexing-view-versus-copy
  self._update_inplace(new_data)
```

In [70]:

```
def matching(df_green, df_grey):
    df_grey.drop_duplicates(["ISIN"], inplace = True)
    df_green.drop_duplicates(["ISIN"], inplace = True)

    u = dt.timedelta(days = 730)
    df_green['Maturity'] =  pd.to_datetime(df_green['Maturity'])
    df_green['Issue Date'] =  pd.to_datetime(df_green['Issue Date'])

    df_grey['Maturity'] =  pd.to_datetime(df_grey['Maturity'])
    df_grey['Issue Date'] =  pd.to_datetime(df_grey['Issue Date'])

    for z in range(1):
        for i in tqdm(range(len(df_green))):
            globals()['Coll%s' % i] = pd.concat([df_green.iloc[[i]],df_grey[ (
df_grey.Seniority == df_green.Seniority[i])\
                & (df_grey["Coupon Type"] == df_green["Coupon Type"][i]) \
                & (df_grey["Issuer Name"] == df_green["Issuer Name"][i])\
                & (df_grey["Principal Currency"] == df_green["Principal Curre
ncy"][i]) \
                & (df_green["Amount Issued (USD)"][i] <= df_grey["Amount Issu
ed (USD)"] * 4) & (df_grey["Amount Issued (USD)"] / 4 <= df_green["Amount Issu
ed (USD)"][i]) \
                & (df_green["Maturity"][i] - u <= df_grey["Maturity"]) & (df_
grey["Maturity"] <= df_green["Maturity"].iloc[i]+ u) \
                & (df_grey["Rating"] == df_green["Rating"][i])\
                & (df_grey["Call Type"] == df_green["Call Type"][i])]], sort=
True)


    Collgath = Coll0.iloc[0:0]
    for i in range(len(df_green)):
        if (len(globals()['Coll%s' %i]) <= 3):
            next
        else:
            Collgath = pd.concat([Collgath,globals()['Coll%s' % i]] )
    Collgath.to_excel('matched_bonds.xlsx', index=False)
```

In [71]:

```
matching(df_green,df_grey)
```

```
/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:2: Se
ttingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pan
das-docs/stable/indexing.html#indexing-view-versus-copy

/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:3: Se
ttingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pan
das-docs/stable/indexing.html#indexing-view-versus-copy
  This is separate from the ipykernel package so we can avoid doin
g imports until
/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:6: Se
ttingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pan
das-docs/stable/indexing.html#indexing-view-versus-copy

/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:7: Se
ttingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pan
das-docs/stable/indexing.html#indexing-view-versus-copy
  import sys
/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:9: Se
ttingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pan
das-docs/stable/indexing.html#indexing-view-versus-copy
  if __name__ == '__main__':
/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:10: S
ettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pan
das-docs/stable/indexing.html#indexing-view-versus-copy
  # Remove the CWD from sys.path while we load stuff.
100%|██████████| 406/406 [00:15<00:00, 25.40it/s]


In [72]:
```

```
matched_bonds = pd.read_excel("matched_bonds.xlsx")
```

In [73]:

```
matched_bonds.groupby(["Green Bond"])["Green Bond"].count().sort_values(ascend
ing=False)
```

Out[73]:

```
Green Bond
No      903
Yes     143
Name: Green Bond, dtype: int64
```

Twin Creation and structuring of data for analysis

In [2]:

```
%reset
import pandas as pd
import numpy as np
from numpy import *
import datetime as dt
from scipy.interpolate import InterpolatedUnivariateSpline
import matplotlib.pyplot as plt
from tqdm import tqdm
import scipy
```

```
Once deleted, variables cannot be recovered. Proceed (y/[n])? y
```

In [3]:

```
ask = pd.read_excel("ASK_yields_2015_2019.xlsx")
bid = pd.read_excel("BID_yields_2015_2019.xlsx")
bond_info = pd.read_excel("Matched_Bond_info.xlsx")
```

In [4]:

```
ask = pd.merge(bond_info, ask, on = ['ISIN'])
bid = pd.merge(bond_info, bid, on = ['ISIN'])
```

Creating a list for the columns with yields, they are equal for both sets, so we only need to make one (only the yields have datetime as column header)

In [5]:

```
date_list = []
for i in range(len(bid.columns)):
    if isinstance(bid.columns[i], dt.datetime):
        date_list.append(bid.columns[i])
```

Creating a dataframe everything except the yield.

In [6]:

```
bond_info = bid.drop(columns= date_list)
```

In [7]:

```
bond_info.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1305 entries, 0 to 1304
Data columns (total 11 columns):
Amount Issued (USD)    1305 non-null int64
Call Type              1305 non-null object
Coupon Type            1305 non-null object
Green_y                1305 non-null object
ISIN                   1305 non-null object
Issue Date             1305 non-null datetime64[ns]
Issuer Name            1305 non-null object
Maturity               1305 non-null datetime64[ns]
Principal Currency     1305 non-null object
Rating                 1305 non-null object
Seniority              1305 non-null object
dtypes: datetime64[ns](2), int64(1), object(8)
memory usage: 122.3+ KB
```

Calculating the liquidity proxy

Isolating the yields

In [8]:

```
bid_yield = bid[date_list]
ask_yield = ask[date_list]
```

If bid or ask have oppsite + or - sign, the observations are removed

In [9]:

```
bid_yield = bid_yield.mask((bid_yield >0) & (ask_yield <0))
bid_yield = bid_yield.mask((bid_yield <0) & (ask_yield >0))
ask_yield = ask_yield.mask((bid_yield >0) & (ask_yield <0))
ask_yield = ask_yield.mask((bid_yield <0) & (ask_yield >0)) # Note that this d
oesnt account for nan's.
#but this doesnt matter since if we subtract a float with a NaN it will return
NaN.
```

In [10]:

```
ask_yield.head()
```

Out[10]:

	2019-05-01 00:00:00	2019-04-30 00:00:00	2019-04-29 00:00:00	2019-04-26 00:00:00	2019-04-25 00:00:00	2019-04-24 00:00:00	2019-04-23 00:00:00	2019-04-22 00:00:00	20 04 00:00
0	8.426	8.356	8.589	8.548	8.597	8.385	8.310	NaN	NaN
1	8.864	8.272	8.017	7.926	8.141	8.222	7.992	8.944	8.944
2	8.469	8.469	8.467	8.464	8.462	8.457	8.456	8.454	8.454
3	NaN	7.748	7.748	7.707	7.774	7.753	7.697	7.646	7.646
4	NaN	7.719	7.706	7.673	7.719	7.727	7.660	NaN	NaN

5 rows × 1132 columns

Taking the closing percent quoted spread(BA): ask- bid / ((ask+bid)/2)

In [11]:

```
BA = abs(ask_yield - bid_yield ) / (abs(ask_yield + bid_yield)  / 2)
```

In [12]:

```
BA.head()
```

Out[12]:

	2019-05-01 00:00:00	2019-04-30 00:00:00	2019-04-29 00:00:00	2019-04-26 00:00:00	2019-04-25 00:00:00	2019-04-24 00:00:00	2019-04-23 00:00:00	2019-04-22 00:00:00	2
0	0.018694	0.028432	0.018342	0.015439	0.007994	0.020655	0.028002	NaN	I
1	0.012556	0.088945	0.119510	0.124911	0.102552	0.091015	0.115428	0.011892	(
2	0.009402	0.009402	0.009404	0.009290	0.009410	0.009298	0.009299	0.009301	(
3	NaN	0.005406	0.006689	0.008012	0.006667	0.007325	0.007249	0.016602	(
4	NaN	0.056402	0.059187	0.059311	0.058968	0.056223	0.059040	NaN	I

5 rows × 1132 columns

In [13]:

```
BA  = pd.concat([bond_info,BA],axis = 1 )
```

In [14]:

```
BA.head()
```

Out[14]:

	Amount Issued (USD)	Call Type	Coupon Type	Green_y	ISIN	Issue Date	Issuer Name	Maturi
0	98489476	No	Plain Vanilla Fixed Coupon	YES	XS0490347415	2010-03-05	International Bank for Reconstruction and Deve...	2020-03-05
1	120510335	No	Plain Vanilla Fixed Coupon	NO	XS1503155068	2016-10-17	International Bank for Reconstruction and Deve...	2019-10-17
2	26197898	No	Plain Vanilla Fixed Coupon	NO	XS1562153848	2017-02-09	International Bank for Reconstruction and Deve...	2021-02-09
3	52395797	No	Plain Vanilla Fixed Coupon	NO	XS1839767297	2018-06-19	International Bank for Reconstruction and Deve...	2021-07-01
4	26296966	No	Plain Vanilla Fixed Coupon	NO	XS1909091230	2018-11-16	International Bank for Reconstruction and Deve...	2021-11-16

5 rows × 1143 columns

Exporting to excel for a check.

In [15]:

```
BA.to_excel('BA.xlsx', index=False)
```

Due to interest rates close to 0, we get some extreme values on the BA on a few days on some euro denominated bonds, we decide to remove the bonds with a BA > 0.7.

In [16]:

```
BA[date_list] = BA[date_list].mask(BA[date_list] > 0.7, np.nan)
```

In [17]:

```
BA.head()
```

Out[17]:

	Amount Issued (USD)	Call Type	Coupon Type	Green_y	ISIN	Issue Date	Issuer Name	Maturi
0	98489476	No	Plain Vanilla Fixed Coupon	YES	XS0490347415	2010-03-05	International Bank for Reconstruction and Deve...	2020-03-05
1	120510335	No	Plain Vanilla Fixed Coupon	NO	XS1503155068	2016-10-17	International Bank for Reconstruction and Deve...	2019-10-17
2	26197898	No	Plain Vanilla Fixed Coupon	NO	XS1562153848	2017-02-09	International Bank for Reconstruction and Deve...	2021-02-09
3	52395797	No	Plain Vanilla Fixed Coupon	NO	XS1839767297	2018-06-19	International Bank for Reconstruction and Deve...	2021-07-01
4	26296966	No	Plain Vanilla Fixed Coupon	NO	XS1909091230	2018-11-16	International Bank for Reconstruction and Deve...	2021-11-16

5 rows × 1143 columns

Matching and twin creation

The funtion needs one dataframe with the green bonds, and one with the conventional to work. When splitting the dataset we have to reset the indexes since it keeps the index ffrom the original dataset.

In [18]:

```
ask_green = ask[ask.Green_y == "YES"]
ask_grey = ask[ask.Green_y == "NO"]
ask_green.reset_index(drop=True, inplace = True)
ask_grey.reset_index(drop = True, inplace = True)
```

In [19]:

```
ask.head()
```

Out[19]:

	Amount Issued (USD)	Call Type	Coupon Type	Green_y	ISIN	Issue Date	Issuer Name	Maturi
0	98489476	No	Plain Vanilla Fixed Coupon	YES	XS0490347415	2010-03-05	International Bank for Reconstruction and Deve...	2020-03-05
1	120510335	No	Plain Vanilla Fixed Coupon	NO	XS1503155068	2016-10-17	International Bank for Reconstruction and Deve...	2019-10-17
2	26197898	No	Plain Vanilla Fixed Coupon	NO	XS1562153848	2017-02-09	International Bank for Reconstruction and Deve...	2021-02-09
3	52395797	No	Plain Vanilla Fixed Coupon	NO	XS1839767297	2018-06-19	International Bank for Reconstruction and Deve...	2021-07-01
4	26296966	No	Plain Vanilla Fixed Coupon	NO	XS1909091230	2018-11-16	International Bank for Reconstruction and Deve...	2021-11-16

5 rows × 1143 columns

In [20]:

```
BA_green = BA[BA.Green_y == "YES"]
BA_grey = BA[BA.Green_y == "NO"]
BA_green.reset_index(drop=True, inplace = True)
BA_grey.reset_index(drop = True, inplace = True)
```

We have one function for the yield, and one for the liquidity. The matching criteria are the same, the only difference is that one interpolates the yield, while the other takes the distance weighted average of the closing percent quoted spread

In [21]:

```
def twin_y(df_green, df_grey):
    df_grey.drop_duplicates(["ISIN"], inplace = True)
    df_green.drop_duplicates(["ISIN"], inplace = True)
```

```
    u = dt.timedelta(days = 730)
    df_green['Maturity'] =  pd.to_datetime(df_green['Maturity'])
    df_green['Issue Date'] =  pd.to_datetime(df_green['Issue Date'])

    df_grey['Maturity'] =  pd.to_datetime(df_grey['Maturity'])
    df_grey['Issue Date'] =  pd.to_datetime(df_grey['Issue Date'])

    for z in range(1):
        for i in tqdm(range(len(df_green))):
            globals()['Coll%s' % i] = pd.concat([df_green.iloc[[i]],df_grey[ (
df_grey.Seniority == df_green.Seniority[i])\
                & (df_grey["Coupon Type"] == df_green["Coupon Type"][i]) \
                & (df_grey["Issuer Name"] == df_green["Issuer Name"][i])\
                & (df_grey["Principal Currency"] == df_green["Principal Curre
ncy"][i]) \
                & (df_green["Amount Issued (USD)"][i] <= df_grey["Amount Issu
ed (USD)"] * 4) & (df_grey["Amount Issued (USD)"] / 4 <= df_green["Amount Issu
ed (USD)"][i]) \
                & (df_green["Maturity"][i] - u <= df_grey["Maturity"]) & (df_
grey["Maturity"] <= df_green["Maturity"].iloc[i]+ u) \
                & (df_grey["Rating"] == df_green["Rating"][i])\
                & (df_grey["Call Type"] == df_green["Call Type"][i])], sort=
True)

            globals()['Coll%s' % i] = pd.concat([globals()['Coll%s' % i].iloc[
0:1], globals()['Coll%s' % i].iloc[1:len(globals()['Coll%s' % i])].sort_values
(by = ['Maturity'])])
            globals()['Coll%s' % i]['Maturity'] = pd.to_datetime(globals()['Co
ll%s' % i]['Maturity']).apply(lambda x: x.date()) #Converting maturity to date
time
            globals()['Coll%s' % i]['Maturity'] = (globals()['Coll%s' % i]['Ma
turity'].apply(lambda x: (x- dt.date(2019,4,15)).days/365)) #Converting matuir
ty(datetime) to year-ratio
            globals()['Coll%s' % i].iloc[1:len(globals()['Coll%s' % i])] = glo
bals()['Coll%s' % i].iloc[1:len(globals()['Coll%s' % i])].drop_duplicates(subs
et=['Maturity'], keep='first') #Drops duplicated maturities because of lin.exp
iteration needs x-axis to increase
            globals()['Coll%s' % i].dropna(subset=['Maturity'], inplace=True)
#drops the above nan line
            globals()['Coll%s' % i].loc['Twin'] = globals()['Coll%s' % i][1:le
n(globals()['Coll%s' % i])-1].mean(numeric_only=True) #Ness. to generate Twin.
Creates mean of every number if necc.



            if len(globals()['Coll%s' % i])<=4: #CanNOT choose lower because o
f lin.exp if neccessary
                globals()['Coll%s' % i] = np.nan
            else:
                for j in range(len(globals()['Coll%s' % i].columns)):
                    if isinstance(globals()['Coll%s' % i].columns[j], dt.date)
:
                        globals()['fp%s' % j] = globals()['Coll%s' % i][global
s()['Coll%s' % i].columns[j]].iloc[1:len(globals()['Coll%s' % i])-1]
                        globals()['xp%s' % j] = globals()['Coll%s' % i]["Matur
ity"].iloc[1:len(globals()['Coll%s' % i])-1]
```

```
                        if globals()['Coll%s' % i]['Maturity'].iloc[0] < np.mi
n(globals()['Coll%s' % i]["Maturity"].iloc[1:len(globals()['Coll%s' % i])-1])
or globals()['Coll%s' % i]['Maturity'].iloc[0] > np.max(globals()['Coll%s' % i
]['Maturity'].iloc[1:len(globals()['Coll%s' % i])-1]):
                            globals()['s%s' % j] = scipy.interpolate.Univariat
eSpline(globals()['xp%s' % j], globals()['fp%s' % j], k=1)
                        else:
                            globals()['s%s' % j] = scipy.interpolate.Univariat
eSpline(globals()['xp%s' % j], globals()['fp%s' % j], k=2)

                        globals()['Coll%s' % i].at['Twin', globals()['Coll%s'
% i].columns[j]]= globals()['s%s' % j](globals()['Coll%s' % i]["Maturity"].ilo
c[0]) #Lin.exp of YTM for spesific maturity date
                    else:
                        next
                #for j in range(1, len(globals()['Coll%s' % i].columns)):
                    #if isinstance(globals()['Coll%s' % i].columns[j], dt.date
):
                        #globals()['fp%s' % j] = globals()['Coll%s' % i][globa
ls()['Coll%s' % i].columns[j]].iloc[1:len(globals()['Coll%s' % i])-1]
                        #globals()['xp%s' % j] = globals()['Coll%s' % i]["Matu
rity"].iloc[1:len(globals()['Coll%s' % i])-1]
                        #globals()['s%s' % j] = InterpolatedUnivariateSpline(g
lobals()['xp%s' % j], globals()['fp%s' % j], k=1)
                        #globals()['Coll%s' % i].loc['Twin'] = globals()['Coll
%s' % i][1:len(globals()['Coll%s' % i])-1].mean(numeric_only=True) #Ness. to g
enerate Twin. Creates mean of every number if necc.
                        #globals()['Coll%s' % i].at['Twin', globals()['Coll%s'
% i].columns[j]]= globals()['s%s' % j](globals()['Coll%s' % i]["Maturity"].iloc
[0]) #Lin.exp of YTM for spesific maturity date
                    #else:
                        #next
                globals()['Coll%s' % i].at['Twin', 'Maturity']= globals()['Col
l%s' % i]["Maturity"].iloc[0]
                globals()['Coll%s' % i].at['Twin', 'Coupon Type'] = globals()[
'Coll%s' % i].at[float(i), 'Coupon Type']
                globals()['Coll%s' % i].at['Twin', 'Rating'] = globals()['Coll
%s' % i].at[float(i), 'Rating']
                globals()['Coll%s' % i].at['Twin', 'Principal Currency'] = glo
bals()['Coll%s' % i].at[float(i), 'Principal Currency']
                globals()['Coll%s' % i].at['Twin', 'Seniority'] = globals()['C
oll%s' % i].at[float(i), 'Seniority']
                globals()['Coll%s' % i].at['Twin', 'Green_y'] = "NO"
                globals()['Coll%s' % i]=globals()['Coll%s' % i].iloc[[0, len(g
lobals()['Coll%s' % i])-1]] #removes the CBs and only keep GB and Synthetic tw
in.
    Collgath = Coll0.iloc[0:0]
    for i in range(len(df_green)):
        if isinstance(globals()['Coll%s' % i], float):
            next
        else:
            Collgath = pd.concat([Collgath,globals()['Coll%s' % i]] )
    Collgath.to_excel('Y_output.xlsx', index=False)



def twin_x(df_green, df_grey):
```

```
    df_grey.drop_duplicates(["ISIN"], inplace = True)
    df_green.drop_duplicates(["ISIN"], inplace = True)

    u = dt.timedelta(days = 730)
    df_green['Maturity'] =  pd.to_datetime(df_green['Maturity'])
    df_green['Issue Date'] =  pd.to_datetime(df_green['Issue Date'])

    df_grey['Maturity'] =  pd.to_datetime(df_grey['Maturity'])
    df_grey['Issue Date'] =  pd.to_datetime(df_grey['Issue Date'])

    #for i in range(len(df_green)):
    for z in range(1):
        for i in tqdm(range(len(df_green))):
            globals()['Coll%s' % i] = pd.concat([df_green.iloc[[i]],df_grey[ (
df_grey.Seniority == df_green.Seniority[i]) \
                & (df_grey["Coupon Type"] == df_green["Coupon Type"][i])\
                & (df_grey["Issuer Name"] == df_green["Issuer Name"][i])\
                & (df_grey["Principal Currency"] == df_green["Principal Curre
ncy"][i]) \
                & (df_green["Amount Issued (USD)"][i] <= df_grey["Amount Issu
ed (USD)"] * 4) & (df_grey["Amount Issued (USD)"] / 4 <= df_green["Amount Issu
ed (USD)"][i]) \
                & (df_green["Maturity"][i] - u <= df_grey["Maturity"]) & (df_
grey["Maturity"] <= df_green["Maturity"].iloc[i]+ u) \
                & (df_grey["Rating"] == df_green["Rating"][i])
                & (df_grey["Call Type"] == df_green["Call Type"][i])], sort=
True)

            globals()['Coll%s' % i] = pd.concat([globals()['Coll%s' % i].iloc[
0:1], globals()['Coll%s' % i].iloc[1:len(globals()['Coll%s' % i])].sort_values
(by = ['Maturity'])])
            globals()['Coll%s' % i]['Maturity'] = pd.to_datetime(globals()['Co
ll%s' % i]['Maturity']).apply(lambda x: x.date()) #Converting maturity to date
time
            globals()['Coll%s' % i]['Maturity'] = (globals()['Coll%s' % i]['Ma
turity'].apply(lambda x: (x- dt.date(2019,4,15)).days/365)) #Converting matuir
ty(datetime) to year-ratio
            globals()['Coll%s' % i].iloc[1:len(globals()['Coll%s' % i])] = glo
bals()['Coll%s' % i].iloc[1:len(globals()['Coll%s' % i])].drop_duplicates(subs
et=['Maturity'], keep='first') #Drops duplicated maturities because of lin.exp
iteration needs x-axis to increase
            globals()['Coll%s' % i].dropna(subset=['Maturity'], inplace=True)
#drops the above nan line
            globals()['Coll%s' % i].loc['Twin'] = globals()['Coll%s' % i][1:le
n(globals()['Coll%s' % i])-1].mean(numeric_only=True) #Ness. to generate Twin.
Creates mean of every number if necc.

            if len(globals()['Coll%s' % i])<=4: #Can choose lower because of l
in.exp if neccessary
                globals()['Coll%s' % i] = np.nan
            else:
                for j in range(len(globals()['Coll%s' % i].columns)):
                    if isinstance(globals()['Coll%s' % i].columns[j], dt.date)
:
                        globals()['Coll%s' % i].at['Twin', globals()['Coll%s'
% i].columns[j]]= 0
                        nevneren = 0
```

```
                    for k in range(1, len(globals()['Coll%s'%i])-1):
                        if globals()['Coll%s' % i][globals()['Coll%s' % i]
.columns[j]].isna()[k]:
                            next
                        elif len(globals()['Coll%s' % i]['Maturity'][np.is
nan(globals()['Coll%s' % i][globals()['Coll%s' % i].columns[j]])==False])<2:
                            next
                        else:
                            nevneren = nevneren + 1/(abs(globals()['Coll%s
' % i]['Maturity'].iloc[0] -  globals()['Coll%s' % i]['Maturity'].iloc[k]))

                    for k in range(1, len(globals()['Coll%s'%i])-1):
                        if globals()['Coll%s' % i][globals()['Coll%s' % i]
.columns[j]].isna()[k]:
                            next
                        elif len(globals()['Coll%s' % i]['Maturity'][np.is
nan(globals()['Coll%s' % i][globals()['Coll%s' % i].columns[j]])==False])<2:
                            next
                        else:
                            globals()['Coll%s' % i].at['Twin',globals()['C
oll%s' % i].columns[j]] = globals()['Coll%s' % i][globals()['Coll%s' % i].colu
mns[j]].loc['Twin'] + 1/(abs((globals()['Coll%s' % i]['Maturity'].iloc[0] - gl
obals()['Coll%s' % i]['Maturity'].iloc[k]))*globals()['Coll%s' % i][globals()
['Coll%s' % i].columns[j]].iloc[k]/nevneren
                globals()['Coll%s' % i].at['Twin', 'Coupon Type'] = globals()[
'Coll%s' % i].at[float(i), 'Coupon Type']
                globals()['Coll%s' % i].at['Twin', 'Rating'] = globals()['Coll
%s' % i].at[float(i), 'Rating']
                globals()['Coll%s' % i].at['Twin', 'Principal Currency'] = glo
bals()['Coll%s' % i].at[float(i), 'Principal Currency']
                globals()['Coll%s' % i].at['Twin', 'Seniority'] = globals()['C
oll%s' % i].at[float(i), 'Seniority']
                globals()['Coll%s' % i].at['Twin', 'Green_y'] = "NO"
                globals()['Coll%s' % i]=globals()['Coll%s' % i].iloc[[0, len(g
lobals()['Coll%s' % i])-1]] #removes the CBs and only keep GB and Synthetic tw
in.
    Collgath = Coll0.iloc[0:0]
    for i in range(len(df_green)):
        if isinstance(globals()['Coll%s' % i], float):
            next
        else:
            Collgath = pd.concat([Collgath,globals()['Coll%s' % i]] )
    Collgath.to_excel('X_output2.xlsx', index=False)
```

Creating twin for yield with the function: twin_y

In [22]:

```
ask_green.to_excel('ask_green.xlsx', index=False)
ask_grey.to_excel('ask_grey.xlsx', index=False )
```

In [23]:

```
df_green = pd.read_excel("ask_green.xlsx")
df_grey = pd.read_excel("ask_grey.xlsx")
```

In [24]:

```
df_green.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 131 entries, 0 to 130
Columns: 1143 entries, Amount Issued (USD) to 2015-01-01 00:00:00
dtypes: datetime64[ns](2), float64(1132), int64(1), object(8)
memory usage: 1.1+ MB
```

In [25]:

```
df_grey.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1174 entries, 0 to 1173
Columns: 1143 entries, Amount Issued (USD) to 2015-01-01 00:00:00
dtypes: datetime64[ns](2), float64(1132), int64(1), object(8)
memory usage: 10.2+ MB
```

In [26]:

```
twin_y(df_green,df_grey)
```

```
100%|██████████| 131/131 [03:49<00:00,  1.83s/it]
```

Creating a twin for liquidity, we overwrite df_green and df_grey since they are defined in both functions.

In [27]:

```
BA_green.to_excel('BA_green.xlsx', index=False)
BA_grey.to_excel('BA_grey.xlsx', index=False )
```

In [28]:

```
df_green = pd.read_excel("BA_green.xlsx")
df_grey = pd.read_excel("BA_grey.xlsx")
```

In [29]:

```
twin_x(df_green,df_grey)
```

```
 68%|██████▊   | 89/131 [19:36<04:13,  6.03s/it]/anaconda3/lib/py
thon3.6/site-packages/ipykernel_launcher.py:103: RuntimeWarning: d
ivide by zero encountered in double_scalars
/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:111:
RuntimeWarning: divide by zero encountered in double_scalars
/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:111:
RuntimeWarning: invalid value encountered in double_scalars
100%|██████████| 131/131 [22:57<00:00,  6.11s/it]
```

Creating the the datasets with difference in yield and liquidity

Importing the output from the functions, Y_output.xlsx is the yield, while X_output2.xlsx is the liquidity premium.

We start with taking the difference in yield

In [30]:

```
twin_yield= pd.read_excel("Y_output.xlsx")
```

In [31]:

```
twin_yield.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 234 entries, 0 to 233
Columns: 1143 entries, Amount Issued (USD) to 2015-01-01 00:00:00
dtypes: datetime64[ns](1), float64(1134), object(8)
memory usage: 2.0+ MB
```

In [32]:

```
twin_yield.head()
```

Out[32]:

	Amount Issued (USD)	Call Type	Coupon Type	Green_y	ISIN	Issue Date	Issuer Name	Ma
0	9.848948e+07	No	Plain Vanilla Fixed Coupon	YES	XS0490347415	2010-03-05	International Bank for Reconstruction and Deve...	0.8
1	6.636801e+07	NaN	['Plain Vanilla Fixed Coupon' 'Plain Vanilla F...	NO	NaN	NaT	NaN	0.8
2	3.345601e+09	No	Plain Vanilla Fixed Coupon	YES	LU0953782009	2013-07-18	European Investment Bank	0.5
3	5.021708e+09	NaN	Plain Vanilla Fixed Coupon	NO	NaN	NaT	NaN	0.5
4	1.561280e+09	No	Plain Vanilla Fixed Coupon	YES	FR0011637586	2013-11-27	Electricite de France SA	2.0

5 rows × 1143 columns

In [33]:

```
yield_green = twin_yield[twin_yield.Green_y == "YES"]
yield_grey = twin_yield[twin_yield.Green_y =="NO"]
```

In [34]:

```
yield_green.reset_index(drop=True, inplace = True)
yield_grey.reset_index(drop = True, inplace = True)
```

Storing the info on the green bonds

In [35]:

```
yield_info = yield_green.drop(columns = date_list)
```

In [36]:

```
yield_diff = yield_green[date_list] - yield_grey[date_list]
```

In [37]:

```
yield_diff  = pd.concat([yield_info,yield_diff],axis = 1 )
```

In [38]:

```
yield_diff.head()
```

Out[38]:

	Amount Issued (USD)	Call Type	Coupon Type	Green_y	ISIN	Issue Date	Issuer Name	Ma
0	9.848948e+07	No	Plain Vanilla Fixed Coupon	YES	XS0490347415	2010-03-05	International Bank for Reconstruction and Deve...	0.8
1	3.345601e+09	No	Plain Vanilla Fixed Coupon	YES	LU0953782009	2013-07-18	European Investment Bank	0.5
2	1.561280e+09	No	Plain Vanilla Fixed Coupon	YES	FR0011637586	2013-11-27	Electricite de France SA	2.0
3	3.467063e+08	No	Plain Vanilla Fixed Coupon	YES	CH0233004172	2014-02-04	European Investment Bank	5.8
4	2.278971e+09	No	Plain Vanilla Fixed Coupon	YES	XS1051861851	2014-04-08	European Investment Bank	0.8

5 rows × 1143 columns

Then we take the difference in liquidity

In [39]:

```
twin_liquidity= pd.read_excel("X_output2.xlsx")
```

In [40]:

```
twin_liquidity.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 234 entries, 0 to 233
Columns: 1143 entries, Amount Issued (USD) to 2015-01-01 00:00:00
dtypes: datetime64[ns](1), float64(1134), object(8)
memory usage: 2.0+ MB
```

In [41]:

```
twin_liquidity.head()
```

Out[41]:

	Amount Issued (USD)	Call Type	Coupon Type	Green_y	ISIN	Issue Date	Issuer Name	Ma
0	9.848948e+07	No	Plain Vanilla Fixed Coupon	YES	XS0490347415	2010-03-05	International Bank for Reconstruction and Deve...	0.8
1	6.636801e+07	NaN	['Plain Vanilla Fixed Coupon' 'Plain Vanilla F...	NO	NaN	NaT	NaN	1.5
2	3.345601e+09	No	Plain Vanilla Fixed Coupon	YES	LU0953782009	2013-07-18	European Investment Bank	0.5
3	5.021708e+09	NaN	Plain Vanilla Fixed Coupon	NO	NaN	NaT	NaN	1.1
4	1.561280e+09	No	Plain Vanilla Fixed Coupon	YES	FR0011637586	2013-11-27	Electricite de France SA	2.0

5 rows × 1143 columns

In [42]:

```
liquidity_green = twin_liquidity[twin_liquidity.Green_y == "YES"]
liquidity_grey = twin_liquidity[twin_liquidity.Green_y =="NO"]
```

In [43]:

```
liquidity_green.reset_index(drop=True, inplace = True)
liquidity_grey.reset_index(drop = True, inplace = True)
```

In [44]:

```
liquidity_info = liquidity_green.drop(columns = date_list)
```

In [45]:

```
liquidity_diff = liquidity_green[date_list] - liquidity_grey[date_list]
```

In [46]:

```
liquidity_diff  = pd.concat([liquidity_info,liquidity_diff],axis = 1 )
```

In [47]:

```
liquidity_diff.head()
```

Out[47]:

	Amount Issued (USD)	Call Type	Coupon Type	Green_y	ISIN	Issue Date	Issuer Name	Ma
0	9.848948e+07	No	Plain Vanilla Fixed Coupon	YES	XS0490347415	2010-03-05	International Bank for Reconstruction and Deve...	0.8
1	3.345601e+09	No	Plain Vanilla Fixed Coupon	YES	LU0953782009	2013-07-18	European Investment Bank	0.5
2	1.561280e+09	No	Plain Vanilla Fixed Coupon	YES	FR0011637586	2013-11-27	Electricite de France SA	2.0
3	3.467063e+08	No	Plain Vanilla Fixed Coupon	YES	CH0233004172	2014-02-04	European Investment Bank	5.8
4	2.278971e+09	No	Plain Vanilla Fixed Coupon	YES	XS1051861851	2014-04-08	European Investment Bank	0.8

5 rows × 1143 columns

In [48]:

```
time_to_maturity = liquidity_diff[["ISIN", "Maturity"]]
```

In [49]:

```
time_to_maturity.rename(columns = {'Maturity':'Time to maturity'}, inplace = True)
```

```
/anaconda3/lib/python3.6/site-packages/pandas/core/frame.py:3778:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pan
das-docs/stable/indexing.html#indexing-view-versus-copy
  return super(DataFrame, self).rename(**kwargs)
```

In [50]:

```
time_to_maturity.head()
```

Out[50]:

	ISIN	Time to maturity
0	XS0490347415	0.890411
1	LU0953782009	0.586301
2	FR0011637586	2.035616
3	CH0233004172	5.813699
4	XS1051861851	0.895890

Transform our datasets into a panel dataset

In [51]:

```
yield_diff = pd.melt(yield_diff, id_vars=["ISIN"], value_vars=date_list)
```

In [52]:

```
yield_diff.rename(columns = {'variable':'date','value':'y'}, inplace = True )
```

In [53]:

```
yield_diff.head()
```

Out[53]:

	ISIN	date	y
0	XS0490347415	2019-05-01	NaN
1	LU0953782009	2019-05-01	NaN
2	FR0011637586	2019-05-01	NaN
3	CH0233004172	2019-05-01	NaN
4	XS1051861851	2019-05-01	-0.025509

In [54]:

```
liquidity_diff = pd.melt(liquidity_diff, id_vars=["ISIN"],value_vars = date_li
st)
```

In [55]:

```
liquidity_diff.rename(columns = {'variable':'date','value':'x'}, inplace = True )
```

In [56]:

```
liquidity_diff.head()
```

Out[56]:

	ISIN	date	x
0	XS0490347415	2019-05-01	0.006138
1	LU0953782009	2019-05-01	NaN
2	FR0011637586	2019-05-01	-0.023673
3	CH0233004172	2019-05-01	NaN
4	XS1051861851	2019-05-01	-0.000214

Connecting yield and liquidity

In [57]:

```
df = pd.concat([yield_diff,liquidity_diff["x"]], axis = 1)
```

In [58]:

```
df.head()
```

Out[58]:

	ISIN	date	y	x
0	XS0490347415	2019-05-01	NaN	0.006138
1	LU0953782009	2019-05-01	NaN	NaN
2	FR0011637586	2019-05-01	NaN	-0.023673
3	CH0233004172	2019-05-01	NaN	NaN
4	XS1051861851	2019-05-01	-0.025509	-0.000214

If a spesific ISIN on a spesific date have only yield(y), or only liquidity(x), we replace the value with NaN.

In [59]:

```
for i in range(len(df)):
    if np.isnan(df.y.iloc[i]):
        df.x.iat[i,] = np.nan
    elif np.isnan(df.x.iloc[i]):
        df.y.iat[i,] = np.nan
```

In [60]:

```
df
```

Out[60]:

	ISIN	date	y	x
0	XS0490347415	2019-05-01	NaN	NaN
1	LU0953782009	2019-05-01	NaN	NaN
2	FR0011637586	2019-05-01	NaN	NaN
3	CH0233004172	2019-05-01	NaN	NaN
4	XS1051861851	2019-05-01	-0.025509	-0.000214
5	XS1057055060	2019-05-01	NaN	NaN
6	XS1083955911	2019-05-01	0.044308	-0.000298
7	XS1087815483	2019-05-01	NaN	NaN
8	XS1111084718	2019-05-01	NaN	NaN
9	XS1107718279	2019-05-01	NaN	NaN
10	US65562QAW50	2019-05-01	0.000021	-0.016330
11	US298785GQ39	2019-05-01	-0.004210	-0.002467
12	AU3CB0226090	2019-05-01	-0.039832	-0.000486
13	US45905URL07	2019-05-01	-0.030575	0.011773
14	US045167CY77	2019-05-01	0.020964	-0.000099
15	XS1209864229	2019-05-01	-0.059917	0.000720
16	AU000KFWHAC9	2019-05-01	-0.088430	0.001314
17	XS1218319702	2019-05-01	0.059901	-0.013983
18	DE000BHY0GP5	2019-05-01	NaN	NaN
19	AU3CB0230100	2019-05-01	-0.131561	0.000569
20	XS1244060486	2019-05-01	-0.000358	0.100527
21	US25389JAL08	2019-05-01	-0.292184	0.001392
22	XS1253847815	2019-05-01	-0.014261	0.138690
23	XS1268337844	2019-05-01	NaN	NaN

24	XS1280834992	2019-05-01	NaN	NaN
25	XS1311459694	2019-05-01	NaN	NaN
26	DE000NWB0AC0	2019-05-01	NaN	NaN
27	US500769GU24	2019-05-01	NaN	NaN
28	XS1324217733	2019-05-01	NaN	NaN
29	XS1324923520	2019-05-01	NaN	NaN
...	...	...	...	...
132414	DE000LB1M214	2015-01-01	NaN	NaN
132415	AU3CB0249787	2015-01-01	NaN	NaN
132416	XS1760129608	2015-01-01	NaN	NaN
132417	XS1766612672	2015-01-01	NaN	NaN
132418	XS1697651468	2015-01-01	NaN	NaN
132419	XS1684812255	2015-01-01	NaN	NaN
132420	XS1684811794	2015-01-01	NaN	NaN
132421	XS1796211933	2015-01-01	NaN	NaN
132422	US63254ABA51	2015-01-01	NaN	NaN
132423	XS1839888754	2015-01-01	NaN	NaN
132424	US015271AM12	2015-01-01	NaN	NaN
132425	XS1848875172	2015-01-01	NaN	NaN
132426	DE000LB1P9C8	2015-01-01	NaN	NaN
132427	XS1814390099	2015-01-01	NaN	NaN
132428	XS1872032369	2015-01-01	NaN	NaN
132429	AU3CB0256162	2015-01-01	NaN	NaN
132430	DE000DHY4994	2015-01-01	NaN	NaN
132431	IT0005346579	2015-01-01	NaN	NaN
132432	XS1856795510	2015-01-01	NaN	NaN
132433	US045167EJ82	2015-01-01	NaN	NaN
132434	US45905UX338	2015-01-01	NaN	NaN
132435	XS1893621026	2015-01-01	NaN	NaN
132436	XS1897258098	2015-01-01	NaN	NaN
132437	DE000BHY0GC3	2015-01-01	NaN	NaN
132438	DE000MHB21J0	2015-01-01	NaN	NaN
132439	AU3SG0001878	2015-01-01	NaN	NaN
132440	HK0000375300	2015-01-01	NaN	NaN

132441	AU3CB0258739	2015-01-01	NaN	NaN
132442	US00828EDF34	2015-01-01	NaN	NaN
132443	XS1917719319	2015-01-01	NaN	NaN

132444 rows × 4 columns

The maturity in this set is constructed in the function as time to maturity, we keep time to maturity, y, x, and date from the dataset df. The rest of the information will be merged from the dataset created in "Data structering" containing all the information about the green bonds, including dummies for use of proceeds and review of third party

In [61]:

```
df = pd.merge(time_to_maturity, df, on=["ISIN"], how = "left")
```

In [62]:

```
gbinfo = pd.read_excel("gbinfo.xlsx")
```

In [63]:

```
df = pd.merge(df, gbinfo, on = (["ISIN"]), how = "left")
```

In [64]:

```
df.sort_values(by=['date', "ISIN"], ascending = False, inplace = True)
```

In [65]:

```
df.head()
```

Out[65]:

	ISIN	Time to maturity	date	y	x	CBI certified	Issuer
131312	XS1917719319	3.643836	2019-05-01	0.264479	0.000039	0	Nederlandse Waterschap NV
123388	XS1897258098	4.131507	2019-05-01	-0.045287	-0.025632	0	Kommuninv Sverige AB
122256	XS1893621026	6.501370	2019-05-01	0.016882	-0.060647	0	EDP Financ
114332	XS1872032369	4.378082	2019-05-01	NaN	NaN	1	National Aus Bank Ltd
118860	XS1856795510	4.457534	2019-05-01	0.002315	0.000068	1	State Bank India (Londo Branch)

5 rows × 42 columns

In [66]:

```
df.reset_index(drop=True, inplace = True) ## lets reset the index
```

In [67]:

```
df.head()
```

Out[67]:

	ISIN	Time to maturity	date	y	x	CBI certified	Issuer Name
0	XS1917719319	3.643836	2019-05-01	0.264479	0.000039	0	Nederlandse Waterschapsbank NV
1	XS1897258098	4.131507	2019-05-01	-0.045287	-0.025632	0	Kommuninvest i Sverige AB
2	XS1893621026	6.501370	2019-05-01	0.016882	-0.060647	0	EDP Finance BV
3	XS1872032369	4.378082	2019-05-01	NaN	NaN	1	National Australia Bank Ltd
4	XS1856795510	4.457534	2019-05-01	0.002315	0.000068	1	State Bank of India (London Branch)

5 rows × 42 columns

In [68]:

```
df["ISIN"].nunique() ## 117 Green bonds
```

Out[68]:

```
117
```

Exporting to excel, this will be imported in the analysis notebook.

In [69]:

```
df.to_excel('Analysis.xlsx', index=False )
```

In [1]:

```
%reset
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib
from linearmodels import PanelOLS
import linearmodels
import linearmodels as lm
import statsmodels.api as sm
from dateutil import rrule
from datetime import datetime, timedelta
import statsmodels
import scipy
import plotly
import plotly.plotly as py
import plotly.graph_objs as go
from astropy.table import Table, Column
```

```
Once deleted, variables cannot be recovered. Proceed (y/[n])? y
```

In [2]:

```
df = pd.read_excel("Analysis.xlsx")
```

In [13]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
MultiIndex: 71019 entries, (XS1917719319, 2019-04-30 00:00:00) to
(AU000KFWHAC9, 2017-01-02 00:00:00)
Data columns (total 40 columns):
Time to maturity            71019 non-null float64
y                           28482 non-null float64
x                           28482 non-null float64
CBI certified               71019 non-null int64
Issuer Name                 71019 non-null object
Ticker                      71019 non-null object
Coupon                      71019 non-null float64
Maturity                    71019 non-null datetime64[ns]
Issue Date                  71019 non-null datetime64[ns]
Details                     70412 non-null object
Amount Issued               71019 non-null int64
Principal Currency          71019 non-null object
Country of Issue            71019 non-null object
Issuer Type                 71019 non-null object
Instrument Type             71019 non-null object
Bond Grade                  70412 non-null object
Call Type                   4856 non-null object
Coupon Type                 71019 non-null object
Market of Issue             71019 non-null object
Seniority                   71019 non-null object
Current Coupon Class        71019 non-null object
Coupon Frequency            71019 non-null object
Issuer Country              71019 non-null object
Callable                    71019 non-null object
Amount Outstanding (USD)    71019 non-null int64
Amount Issued (USD)         71019 non-null int64
Amount Outstanding          71019 non-null int64
Green Bond                  71019 non-null object
Dual Currency               71019 non-null object
Sector                      71019 non-null object
Rating                      71019 non-null object
Adaptation                  71019 non-null int64
Industry                    71019 non-null int64
Waste                       71019 non-null int64
Asia-Pacific                71019 non-null int64
Buildings                   71019 non-null int64
Energy                      71019 non-null int64
Land Use                    71019 non-null int64
Transport                   71019 non-null int64
Water                       71019 non-null int64
dtypes: datetime64[ns](2), float64(4), int64(14), object(20)
memory usage: 21.9+ MB
```

In [4]:

```
df.columns
```

Out[4]:

```
Index(['ISIN', 'Time to maturity', 'date', 'y', 'x', 'CBI certifie
d',
       'Issuer Name', 'Ticker', 'Coupon', 'Maturity', 'Issue Date'
, 'Details',
       'Amount Issued', 'Principal Currency', 'Country of Issue',
       'Issuer Type', 'Instrument Type', 'Bond Grade', 'Call Type'
,
       'Coupon Type', 'Market of Issue', 'Seniority', 'Current Cou
pon Class',
       'Coupon Frequency', 'Issuer Country', 'Callable',
       'Amount Outstanding (USD)', 'Amount Issued (USD)', 'Amount
Outstanding',
       'Green Bond', 'Dual Currency', 'Sector', 'Rating', ' Adapta
tion',
       ' Industry', ' Waste', 'Asia-Pacific', 'Buildings', 'Energy
',
       'Land Use', 'Transport', 'Water'],
      dtype='object')
```

In [5]:

```
df.rename(columns={' Adaptation': 'Adaptation', ' Industry': 'Industry', " Was
te": "Waste"}, inplace=True)
```

Removing Lira and NZD, only one bond in each, lack of obs

Time cleansing

In [8]:

```
df.set_index(['date'], inplace=True)
```

In [9]:

```
df = df['2019-04-30':'2016-04-30']
```

In [10]:

```
df = df[df.index.get_level_values('date') >= datetime(2017,1,1)]
```

Setting panel data index

In [11]:

```
df.reset_index(inplace=True)
df.set_index(['ISIN', 'date'], inplace=True)
```

Creating Corp and Gov

In [354]:

```
corp = df[df["Issuer Type"] == "Corporate"]
gov = df[(df['Issuer Type'] != "Corporate")]
```

In [355]:

```
df.index.get_level_values("ISIN").unique()
```

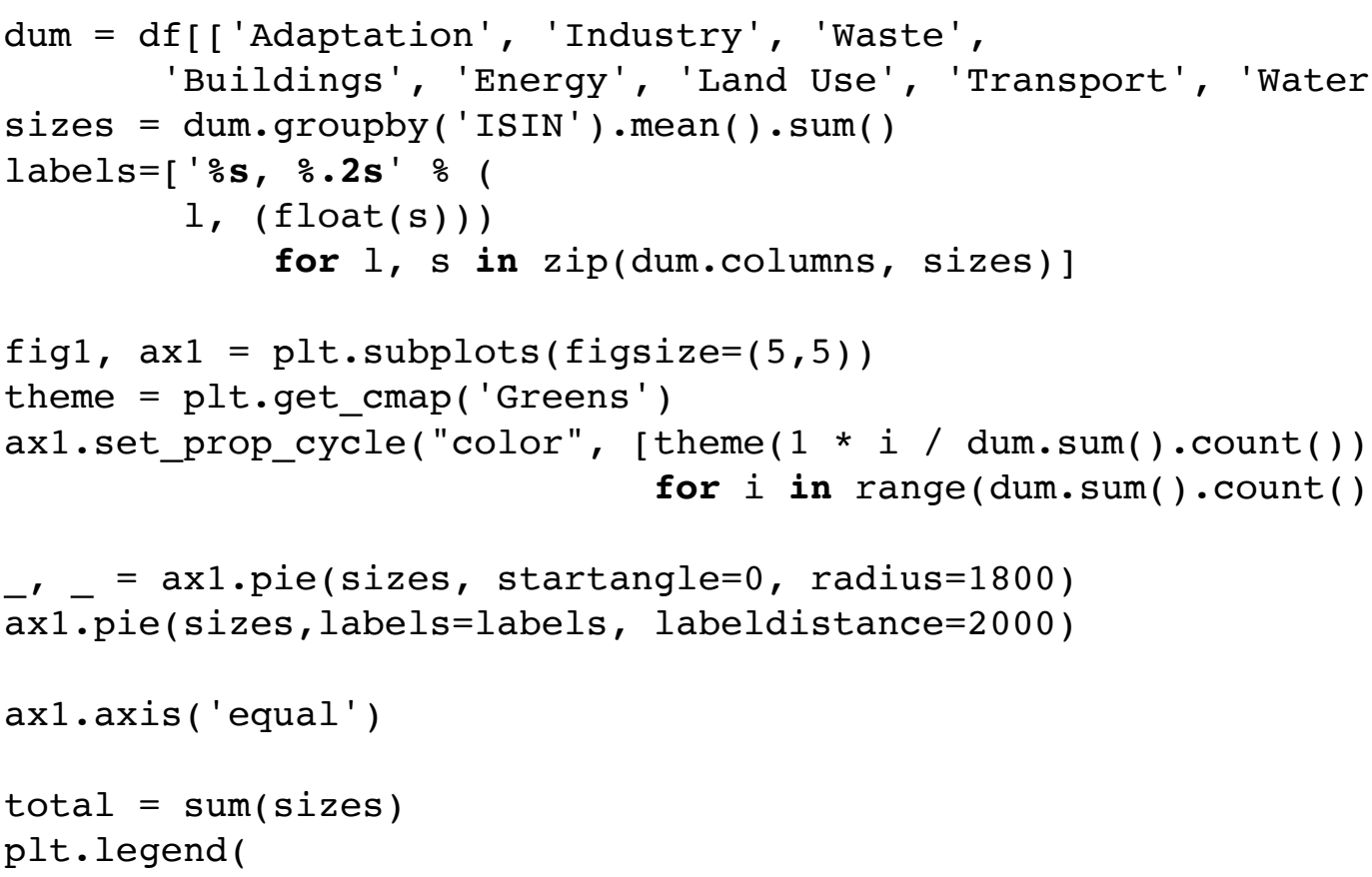
Out[355]:

```
Index(['XS1917719319', 'XS1897258098', 'XS1893621026', 'XS18720323
69',
       'XS1856795510', 'XS1848875172', 'XS1839888754', 'XS18143900
99',
       'XS1796211933', 'XS1766612672',
       ...
       'CA68323ADL58', 'AU3SG0001878', 'AU3CB0258739', 'AU3CB02561
62',
       'AU3CB0249787', 'AU3CB0243657', 'AU3CB0237683', 'AU3CB02301
00',
       'AU3CB0226090', 'AU000KFWHAC9'],
      dtype='object', name='ISIN', length=117)
```

UoP distribution

In [356]:

```
dum = df[['Adaptation', 'Industry', 'Waste',
       'Buildings', 'Energy', 'Land Use', 'Transport', 'Water']]
sizes = dum.groupby('ISIN').mean().sum()
labels=['%s, %.2s' % (
        l, (float(s)))
            for l, s in zip(dum.columns, sizes)]

fig1, ax1 = plt.subplots(figsize=(5,5))
theme = plt.get_cmap('Greens')
ax1.set_prop_cycle("color", [theme(1 * i / dum.sum().count())
                             for i in range(dum.sum().count())])

_, _ = ax1.pie(sizes, startangle=0, radius=1800)
ax1.pie(sizes,labels=labels, labeldistance=2000)

ax1.axis('equal')

total = sum(sizes)
plt.legend(
    loc='upper right',
    labels=dum.columns,
    prop={'size': 11},
    bbox_to_anchor=(0.0, 1),
    bbox_transform=fig1.transFigure
);

fig1.savefig('UoP distribution.png', )
```



In [16]:

```
mod = PanelOLS.from_formula('y ~1 + x+ EntityEffects', df)
mod.fit(cov_type = 'kernel')
```

```
/anaconda3/lib/python3.6/site-packages/linearmodels/utility.py:476
: MissingValueWarning:


Inputs contain missing values. Dropping rows with missing observat
ions.
```

Out[16]:

PanelOLS Estimation Summary

Dep. Variable:	y	**R-squared:**	0.0618
Estimator:	PanelOLS	**R-squared (Between):**	0.0765
No. Observations:	28482	**R-squared (Within):**	0.0618
Date:	Tue, Jun 18 2019	**R-squared (Overall):**	0.0959
Time:	18:01:19	**Log-likelihood**	3.739e+04
Cov. Estimator:	Driscoll-Kraay		
		F-statistic:	1867.6
Entities:	117	**P-value**	0.0000
Avg Obs:	243.44	**Distribution:**	F(1,28364)
Min Obs:	3.0000		
Max Obs:	607.00	**F-statistic (robust):**	258.10
		P-value	0.0000
Time periods:	607	**Distribution:**	F(1,28364)
Avg Obs:	46.923		
Min Obs:	12.000		
Max Obs:	101.00		

Parameter Estimates

	Parameter	Std. Err.	T-stat	P-value	Lower CI	Upper CI
Intercept	-0.0174	0.0007	-23.329	0.0000	-0.0189	-0.0159
x	-0.1866	0.0116	-16.065	0.0000	-0.2093	-0.1638

F-test for Poolability: 256.05
P-value: 0.0000
Distribution: F(116,28364)

Included effects: Entity
id: 0x1c16f22198

Part one regression:

In [359]:

```
mod_no_cons = PanelOLS.from_formula('y ~x + EntityEffects', df)
res = mod_no_cons.fit(cov_type = 'kernel')
```

In [360]:

```
res.estimated_effects.mean()
```

Out[360]:

```
estimated_effects   -0.017408
dtype: float64
```

In [361]:

```
res
```

Out[361]:

PanelOLS Estimation Summary

Dep. Variable:	y	**R-squared:**	0.0618
Estimator:	PanelOLS	**R-squared (Between):**	0.0836
No. Observations:	28482	**R-squared (Within):**	0.0618
Date:	Sun, Jun 16 2019	**R-squared (Overall):**	0.0964
Time:	19:40:36	**Log-likelihood**	3.739e+04
Cov. Estimator:	Driscoll-Kraay		
		F-statistic:	1867.6
Entities:	117	**P-value**	0.0000
Avg Obs:	243.44	**Distribution:**	F(1,28364)
Min Obs:	3.0000		
Max Obs:	607.00	**F-statistic (robust):**	258.10
		P-value	0.0000
Time periods:	607	**Distribution:**	F(1,28364)
Avg Obs:	46.923		
Min Obs:	12.000		
Max Obs:	101.00		

Parameter Estimates

	Parameter	Std. Err.	T-stat	P-value	Lower CI	Upper CI
x	-0.1866	0.0116	-16.065	0.0000	-0.2093	-0.1638

F-test for Poolability: 256.05
P-value: 0.0000
Distribution: F(116,28364)

Included effects: Entity
id: 0x2a90818d630

Output table

In [362]:

```
open('regression1.tex', 'w').write(linearmodels.panel.compare((res, res), prec
ision='std_errors').summary.as_latex())
```

Out[362]:

1342

Graph Alpha over time

In [827]:

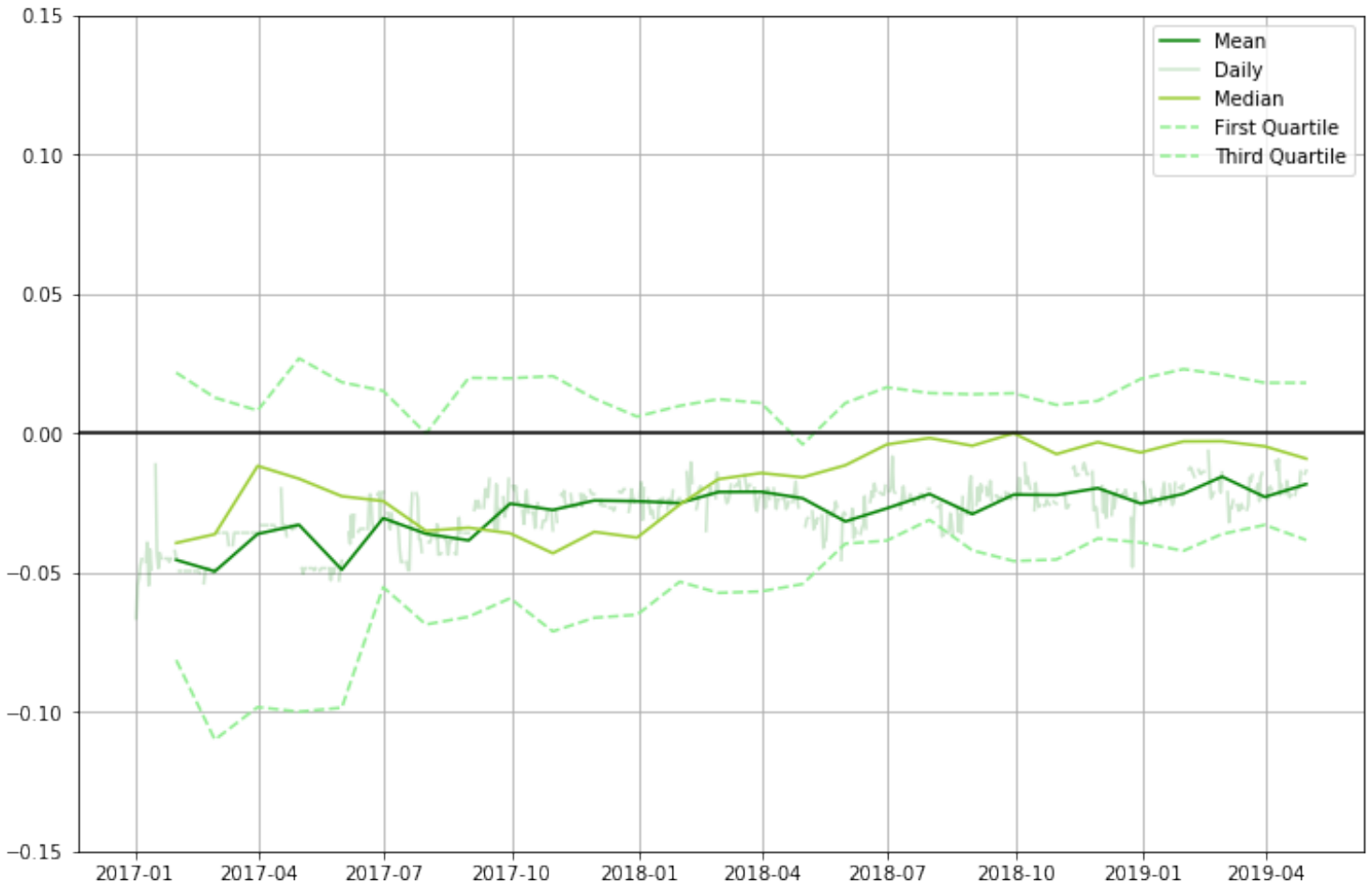
```
def graffunk(dataset, Category=None, times =datetime(2017,1,1)):
    if Category != None:
        nest = pd.DataFrame()
        for dt in rrule.rrule(rrule.MONTHLY, dtstart=times, until=datetime(201
9,4,30)):
            if dt.month < 12:
                dfdate = dataset[dataset[Category] == 1][(dataset[dataset[Cate
gory] == 1].index.get_level_values('date')<datetime(dt.year,dt.month+1,dt.day)
) & (dataset[dataset[Category] == 1].index.get_level_values('date') >dt)]
            else:
                dfdate = dataset[dataset[Category] == 1][(dataset[dataset[Cate
gory] == 1].index.get_level_values('date')<datetime(dt.year+1,1,dt.day)) & (da
taset[dataset[Category] == 1].index.get_level_values('date') >dt)]
            mod_no_cons = PanelOLS.from_formula('y ~x + EntityEffects', dfdate
)
            nest = pd.concat([nest, mod_no_cons.fit(cov_type = 'kernel').estim
ated_effects])

        mod = PanelOLS.from_formula('y ~x + EntityEffects', dataset[dataset[Ca
tegory] == 1])
        res = mod.fit(cov_type = 'kernel')
        mod_category = PanelOLS(dataset.y[dataset[Category] == 1], dataset.x[d
ataset[Category] == 1],entity_effects=True)
        alfa = (dataset.y[dataset.y.notnull() & dataset[Category] == 1] - data
set.x[dataset.y.notnull() & dataset[Category] == 1]*mod.fit().params[0]-mod_ca
tegory.fit().resids).groupby('ISIN').mean()
        beta = res.estimated_effects.groupby('ISIN').mean()
        beta_mean = res.estimated_effects.mean()
        print(beta_mean)
        plt.figure(figsize=(12,8))
        plt.plot(nest.estimated_effects.groupby('date').mean().resample('M').m
ean(), color="green")
        plt.plot(nest.estimated_effects.groupby('date').mean().resample('d').m
ean(), color="green", alpha=0.2)
        plt.plot(nest.estimated_effects.groupby('date').median().resample('M')
.mean(), color="yellowgreen")
        plt.plot(nest.estimated_effects.groupby('date').quantile(0.25).resampl
e('M').mean(), linestyle='dashed', color="lightgreen")
        plt.plot(nest.estimated_effects.groupby('date').quantile(0.75).resampl
e('M').mean(), linestyle='dashed', color="lightgreen")
        plt.ylim(-0.15, 0.15)
        plt.grid(b=True)
        plt.legend(["Mean","Daily", "Median", "First Quartile", "Third Quartil
```

```
e"])

        plt.axhline(y=0, color='black', linestyle='-')

        plt.savefig(str(Category) + str("Time.png"))

    if Category == None:
        nest = pd.DataFrame()
        for dt in rrule.rrule(rrule.MONTHLY, dtstart=times, until=datetime(201
9,4,30)):
            if dt.month < 12:
                dfdate = dataset[(dataset.index.get_level_values('date')<datet
ime(dt.year,dt.month+1,dt.day)) & (dataset.index.get_level_values('date') >dt)
]
            else:
                dfdate = dataset[(dataset.index.get_level_values('date')<datet
ime(dt.year+1,1,dt.day)) & (dataset.index.get_level_values('date') >dt)]
            mod_no_cons = PanelOLS.from_formula('y ~x + EntityEffects', dfdate
)
            nest = pd.concat([nest, mod_no_cons.fit(cov_type = 'kernel').estim
ated_effects])

        mod = PanelOLS.from_formula('y ~x + EntityEffects', dataset)
        res = mod.fit(cov_type = 'kernel')
        alfa = (dataset.y[dataset.y.notnull()]-dataset.x[dataset.y.notnull()]*
res.params.x-res.resids).groupby('ISIN').mean()
        plt.figure(figsize=(12,8))
        plt.plot(nest.groupby('date').mean().resample('M').mean(), color="gree
n")
        plt.plot(nest.groupby('date').mean().resample('d').mean(), color="gree
n", alpha=0.2)
        plt.plot(nest.groupby('date').median().resample('M').mean(), color="ye
llowgreen")
        plt.plot(nest.groupby('date').quantile(0.25).resample('M').mean(), lin
estyle='dashed', color="lightgreen")
        plt.plot(nest.groupby('date').quantile(0.75).resample('M').mean(), lin
estyle='dashed', color="lightgreen")
        plt.ylim(-0.15, 0.15)
        plt.xlim(df.index[len(df)-1][1], df.index[0][1])
        plt.grid(b=True)
        plt.axhline(y=0, color='black', linestyle='-')
        plt.legend(["Mean","Daily", "Median", "Third Quartile", "First Quartil
e"])
        beta = res.estimated_effects.groupby('ISIN').mean()
        beta_mean = res.estimated_effects.mean()
        print(beta_mean)
        if dataset.iloc[1][1] == corp.iloc[1][1]:
            dataset1 = "corp"
        elif dataset.iloc[1][1] == gov.iloc[1][1]:
            dataset1 ="gov"
        else:
            dataset1 == "Full"
        plt.savefig(str(dataset1) + str('Time.png'))
```

In [828]:

```
graffunk(df, Category = "Energy", times= datetime(2017,1,1))
```

```
estimated_effects   -0.024167
dtype: float64
```



In [836]:

```
df[df.Waste == 1].alpha.mean()
```

Out[836]:

```
-0.021209839971070444
```

Creating alpha

In [365]:

```
df["alpha"] = res.estimated_effects
```

Dummies

In [366]:

```
df = pd.concat([df, pd.get_dummies(df.Rating)], axis=1)
df = pd.concat([df, pd.get_dummies(df['Seniority'])], axis=1)
df = pd.concat([df, pd.get_dummies(df['Principal Currency'])], axis=1)
df = pd.concat([df, pd.get_dummies(df['Country of Issue'])], axis=1)
```

Creating Ordinal Rating:

In [367]:

```
df.Rating.replace({'AAA': 1, 'AA+': 2, 'AA': 3, 'AA-':4, 'A+':5, 'A':6, 'A-':7
, 'BBB+':8, 'BBB':9, 'BBB-':10}, inplace=True)
```

Wilcoxon signed-rank test

In [368]:

```
df['A-nest'] = df['A']+df['A+']+df['A-']
df['AA-nest'] = df['AA']+df['AA+']+df['AA-']
df['BBB-nest'] = df['BBB']+df['BBB+']+df['BBB-']
df['Log_AmountIssued'] = np.log(df['Amount Issued'])


wilc_isin0 = scipy.stats.wilcoxon(df["alpha"].groupby('ISIN').mean())
wilc_Energy = scipy.stats.wilcoxon(df["alpha"][df['Energy']==1].groupby('ISIN'
).mean())
wilc_Buildings = scipy.stats.wilcoxon(df["alpha"][df['Buildings']==1].groupby(
'ISIN').mean())
wilc_Waste = scipy.stats.wilcoxon(df["alpha"][df['Waste']==1].groupby('ISIN').
mean())
wilc_Transport = scipy.stats.wilcoxon(df["alpha"][df['Transport']==1].groupby(
'ISIN').mean())
wilc_Adaption = scipy.stats.wilcoxon(df["alpha"][df['Adaptation']==1].groupby(
'ISIN').mean())
wilc_Water = scipy.stats.wilcoxon(df["alpha"][df['Water']==1].groupby('ISIN').
mean())
wilc_Industry = scipy.stats.wilcoxon(df["alpha"][df['Industry']==1].groupby('I
SIN').mean())
wilc_LandUse = scipy.stats.wilcoxon(df["alpha"][df['Land Use']==1].groupby('IS
IN').mean())
wilc_AAA = scipy.stats.wilcoxon(df["alpha"][df['AAA']==1].groupby('ISIN').mean
())
wilc_AA = scipy.stats.wilcoxon(df["alpha"][df['AA-nest']==1].groupby('ISIN').m
ean())
wilc_A = scipy.stats.wilcoxon(df["alpha"][df['A-nest']==1].groupby('ISIN').mea
n())
wilc_BBB = scipy.stats.wilcoxon(df["alpha"][df['BBB-nest']==1].groupby('ISIN')
.mean())
wilc_CBI = scipy.stats.wilcoxon(df["alpha"][df['CBI certified']==1].groupby('I
SIN').mean())

wilc_isin0c = df["alpha"].groupby('ISIN').mean().count()
wilc_Energyc = df["alpha"][df['Energy']==1].groupby('ISIN').mean().count()
wilc_Buildingsc = df["alpha"][df['Buildings']==1].groupby('ISIN').mean().count
()
wilc_Wastec = df["alpha"][df['Waste']==1].groupby('ISIN').mean().count()
wilc_Transportc = df["alpha"][df['Transport']==1].groupby('ISIN').mean().count
()
wilc_Adaptionc = df["alpha"][df['Adaptation']==1].groupby('ISIN').mean().count
()
wilc_Waterc = df["alpha"][df['Water']==1].groupby('ISIN').mean().count()
wilc_Industryc =df["alpha"][df['Industry']==1].groupby('ISIN').mean().count()
wilc_LandUsec = df["alpha"][df['Land Use']==1].groupby('ISIN').mean().count()
wilc_AAAc = df["alpha"][df['AAA']==1].groupby('ISIN').mean().count()
```

```
wilc_AAc = df["alpha"][df['AA-nest']==1].groupby('ISIN').mean().count()
wilc_Ac = df["alpha"][df['A-nest']==1].groupby('ISIN').mean().count()
wilc_BBBc = df["alpha"][df['BBB-nest']==1].groupby('ISIN').mean().count()
wilc_CBIc = df["alpha"][df['CBI certified']==1].groupby('ISIN').mean().count()

t = Table([['Full sample', 'Adaption', 'Buildings', 'Energy', 'Industry', 'Lan
d Use', 'Transport', 'Waste', 'Water', 'AAA', 'AA', 'A', 'BBB', 'CBI certified
'],[wilc_isin0.pvalue, wilc_Adaption.pvalue, wilc_Buildings.pvalue, wilc_Energ
y.pvalue, wilc_Industry.pvalue, wilc_LandUse.pvalue, wilc_Transport.pvalue, wi
lc_Waste.pvalue, wilc_Water.pvalue, wilc_AAA.pvalue, wilc_AA.pvalue, wilc_A.pv
alue, wilc_BBB.pvalue, wilc_CBI.pvalue], [wilc_isin0c, wilc_Adaptionc, wilc_Bu
ildingsc, wilc_Energyc, wilc_Industryc, wilc_LandUsec, wilc_Transportc, wilc_W
astec, wilc_Waterc, wilc_AAAc, wilc_AAc, wilc_Ac, wilc_BBBc, wilc_CBIc]], name
s = ('Description', 'p-value', "No.bonds"))
t['p-value'].format = '.4f'

import sys
t.write(sys.stdout, format='latex')
```

```
\begin{table}
\begin{tabular}{ccc}
Description & p-value & No.bonds \\
Full sample & 0.0500 & 117 \\
Adaption & 0.6496 & 15 \\
Buildings & 0.3184 & 73 \\
Energy & 0.0321 & 90 \\
Industry & 0.5002 & 5 \\
Land Use & 0.7103 & 24 \\
Transport & 0.9471 & 53 \\
Waste & 0.4842 & 23 \\
Water & 0.9615 & 43 \\
AAA & 0.1332 & 60 \\
AA & 0.6309 & 27 \\
A & 0.0552 & 17 \\
BBB & 0.5067 & 13 \\
CBI certified & 0.5701 & 15 \\
\end{tabular}
\end{table}
```

```
C:\Users\eoest\Anaconda3\lib\site-packages\scipy\stats\morestats.p
y:2388: UserWarning:

Warning: sample size too small for normal approximation.
```

Alpha statistic table

In [369]:

```
mod_no_cons = PanelOLS.from_formula('y ~x + EntityEffects', df)
res = mod_no_cons.fit(cov_type = 'kernel')
df["alpha"] = res.estimated_effects
```

In [370]:

```
df["alpha"]
t = Table([[' '],[df["alpha"].min()],[df["alpha"].describe()['25%']],[df["alph
a"].median()],[df["alpha"].mean()],[df["alpha"].describe()['75%']],[df["alpha"
].max()],[df["alpha"].std()]], names = ('Alpha','Min', '1Q', 'Median', 'Mean',
'3Q', 'Max', 'Std.dev'))
t['Min'].format  = '.4f'
t['1Q'].format  = '.4f'
t['Median'].format  = '.4f'
t['Mean'].format  = '.4f'
t['3Q'].format  = '.4f'
t['Max'].format  = '.4f'
t['Std.dev'].format  = '.4f'
import sys
t.write(sys.stdout, format='latex')
```

```
\begin{table}
\begin{tabular}{cccccccc}
Alpha & Min & 1Q & Median & Mean & 3Q & Max & Std.dev \\
 & -0.3503 & -0.0408 & -0.0047 & -0.0174 & 0.0115 & 0.4212 & 0.067
8 \\
\end{tabular}
\end{table}
```

BA statistic table

In [371]:

```
df["x"]
t = Table([[' '],[df["x"].min()],[df["x"].describe()['25%']],[df["x"].median()
],[df["x"].mean()],[df["x"].describe()['75%']],[df["x"].max()],[df["x"].std()]
], names = ('BA','Min', '1Q', 'Median', 'Mean', '3Q', 'Max', 'Std.dev'))
t['Min'].format  = '.4f'
t['1Q'].format  = '.4f'
t['Median'].format  = '.4f'
t['Mean'].format  = '.4f'
t['3Q'].format  = '.4f'
t['Max'].format  = '.4f'
t['Std.dev'].format  = '.4f'
import sys
t.write(sys.stdout, format='latex')
```

```
\begin{table}
\begin{tabular}{cccccccc}
BA & Min & 1Q & Median & Mean & 3Q & Max & Std.dev \\
 & -0.5843 & -0.0204 & -0.0004 & 0.0056 & 0.0090 & 0.6458 & 0.1096
\\
\end{tabular}
\end{table}
```

Part two-regression:

In [733]:

```
df1 = pd.concat([df.groupby('ISIN').mean(), df.groupby('ISIN').head(1).reset_i
ndex().set_index('ISIN')], axis=1, sort=False, ignore_index=False)
df1 = df1.loc[:,~df1.columns.duplicated()]
df1['cons'] = 1
df1['A-nest'] = df1['A']+df1['A+']+df1['A-']
df1['AA-nest'] = df1['AA']+df1['AA+']+df1['AA-']
df1['BBB-nest'] = df1['BBB']+df1['BBB+']+df1['BBB-']
df1['Log_AmountIssued'] = np.log(df1['Amount Issued (USD)']/1000000000)
df1 = pd.concat([df1, pd.get_dummies(df1['Principal Currency'])], axis=1)
df1 = pd.concat([df1, pd.get_dummies(df1['Sector'])], axis=1)
df1.columns
df2 = pd.concat([df.groupby('ISIN').mean(), df.groupby('ISIN').head(1).reset_i
ndex().set_index('ISIN')], axis=1, sort=False, ignore_index=False)
df2 = df2.loc[:,~df2.columns.duplicated()]
```

In [734]:

```
df.y.describe()
```

Out[734]:

```
count    28482.000000
mean        -0.018451
std          0.098886
min         -0.999546
25%         -0.051160
50%         -0.007037
75%          0.022468
max          0.883858
Name: y, dtype: float64
```

Alpha on rating:

In [735]:

```
reg_rating = sm.OLS(df1.alpha, exog=df1[['cons','Time to maturity','Log_Amount
Issued','A', 'A+', 'A-', 'AA', 'AA+', 'AA-', 'BBB', 'BBB+', 'BBB-']], missing=
'drop').fit().summary()
reg_rating
```

Out[735]:

OLS Regression Results

Dep. Variable:	alpha	**R-squared:**	0.159
Model:	OLS	**Adj. R-squared:**	0.071
Method:	Least Squares	**F-statistic:**	1.806
Date:	Mon, 17 Jun 2019	**Prob (F-statistic):**	0.0618
Time:	17:26:01	**Log-Likelihood:**	135.07
No. Observations:	117	**AIC:**	-246.1
Df Residuals:	105	**BIC:**	-213.0
Df Model:	11		
Covariance Type:	nonrobust		

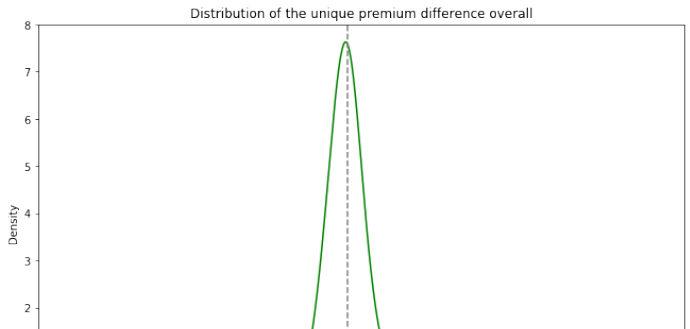
	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
cons	-0.0288	0.017	-1.658	0.100	-0.063	0.006
Time to maturity	-0.0006	0.003	-0.179	0.859	-0.007	0.006
Log_AmountIssued	-0.0202	0.009	-2.369	0.020	-0.037	-0.003
A	0.0364	0.058	0.626	0.533	-0.079	0.152
A+	-0.0569	0.026	-2.153	0.034	-0.109	-0.004
A-	0.0480	0.042	1.139	0.257	-0.036	0.132
AA	0.0873	0.038	2.327	0.022	0.013	0.162
AA+	0.0185	0.033	0.561	0.576	-0.047	0.084
AA-	0.0280	0.023	1.205	0.231	-0.018	0.074
BBB	0.0077	0.042	0.184	0.855	-0.075	0.090
BBB+	-0.0009	0.035	-0.025	0.980	-0.070	0.068
BBB-	0.0253	0.048	0.531	0.597	-0.069	0.120

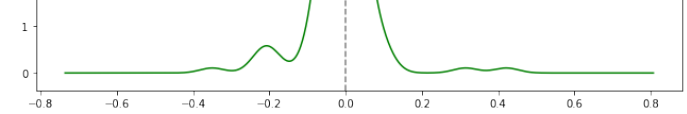
Omnibus:	23.819	**Durbin-Watson:**	2.149
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	165.675
Skew:	-0.176	**Prob(JB):**	1.06e-36
Kurtosis:	8.819	**Cond. No.**	35.6

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

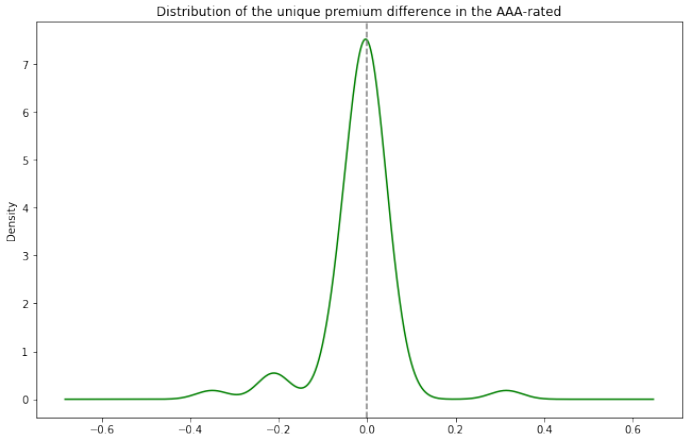
In [736]:

```
plt.figure(figsize=(24,48))
plt.subplot(621)
df1.alpha.plot.kde(color='g')
plt.axvline(x=0, color='grey', linestyle='--');
plt.title("Distribution of the unique premium difference overall")

plt.subplot(623)
df1.alpha[df1.AAA==1].plot.kde(color='g')
plt.title("Distribution of the unique premium difference in the AAA-rated")
plt.axvline(x=0, color='grey', linestyle='--');
plt.subplot(624)
df1.alpha[df1['AA+']==1].plot.kde(color='g')
plt.title("Distribution of the unique premium difference in the AA+-rated")
plt.axvline(x=0, color='grey', linestyle='--');
plt.subplot(625)
df1.alpha[df1['AA']==1].plot.kde(color='g')
plt.title("Distribution of the unique premium difference in the AA-rated")
plt.axvline(x=0, color='grey', linestyle='--');
plt.subplot(626)
df1.alpha[df1['AA-']==1].plot.kde(color='g')
plt.title("Distribution of the unique premium difference in the AA--rated")
plt.axvline(x=0, color='grey', linestyle='--');
plt.subplot(627)
df1.alpha[df1['A+']==1].plot.kde(color='g')
plt.title("Distribution of the unique premium difference in the A+-rated")
plt.axvline(x=0, color='grey', linestyle='--');
plt.subplot(628)
df1.alpha[df1['A']==1].plot.kde(color='g')
plt.title("Distribution of the unique premium difference in the A-rated")
plt.axvline(x=0, color='grey', linestyle='--');
plt.subplot(629)
df1.alpha[df1['A-']==1].plot.kde(color='g')
plt.title("Distribution of the unique premium difference in the A--rated")
plt.axvline(x=0, color='grey', linestyle='--');
plt.subplot(6,2,10)
df1.alpha[df1['BBB+']==1].plot.kde(color='g')
plt.title("Distribution of the unique premium difference in the BBB+-rated")
plt.axvline(x=0, color='grey', linestyle='--');
plt.subplot(6,2,11)
df1.alpha[df1['BBB']==1].plot.kde(color='g')
plt.title("Distribution of the unique premium difference in the BBB-rated")
plt.axvline(x=0, color='grey', linestyle='--');
plt.subplot(6,2,12)
df1.alpha[df1['BBB-']==1].plot.kde(color='g')
plt.title("Distribution of the unique premium difference in the BBB--rated")
plt.axvline(x=0, color='grey', linestyle='--');
plt.savefig("Alpha distribution Appendix.PNG")
```

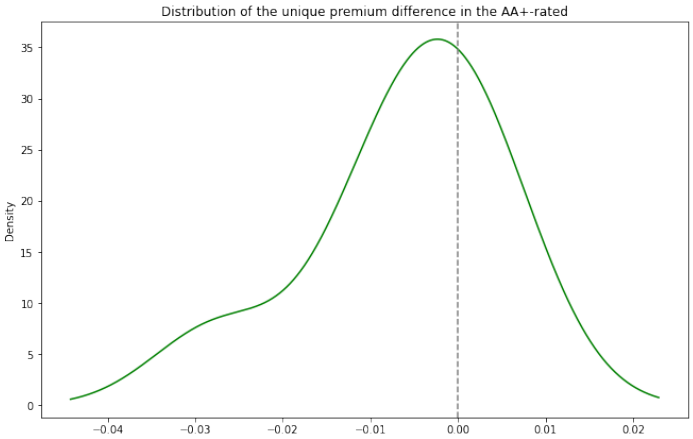




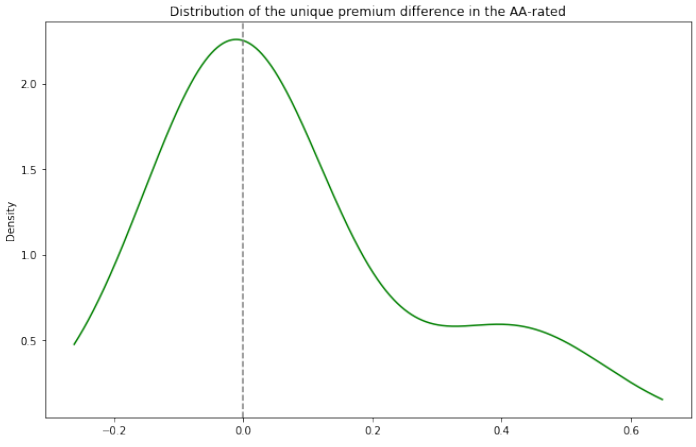
Distribution of the unique premium difference in the AAA-rated



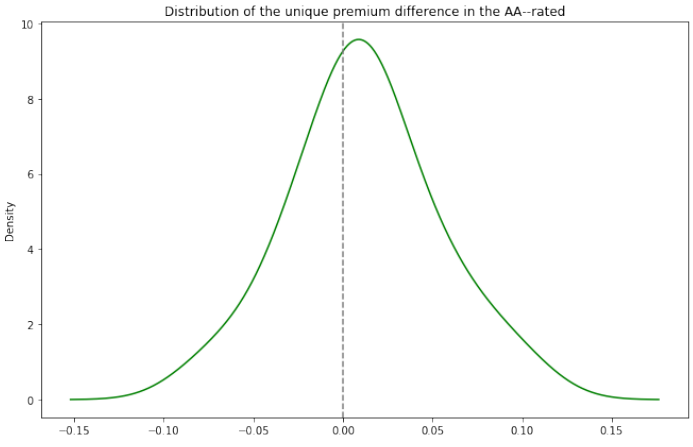
Distribution of the unique premium difference in the AA+-rated



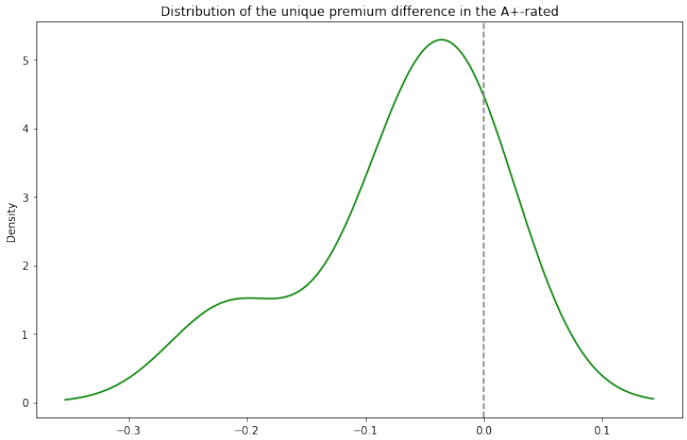
Distribution of the unique premium difference in the AA-rated



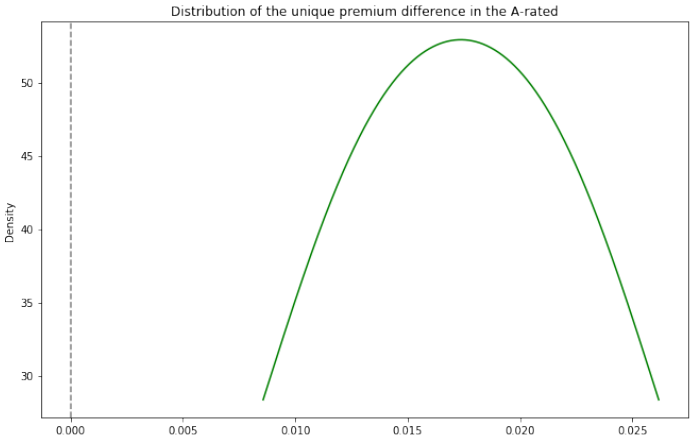
Distribution of the unique premium difference in the AA--rated



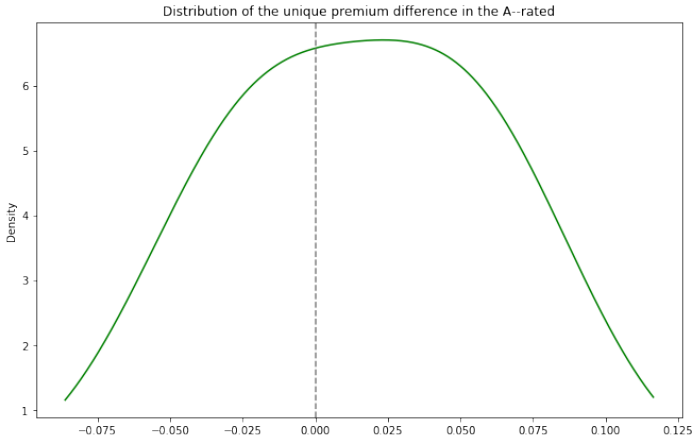
Distribution of the unique premium difference in the A+-rated



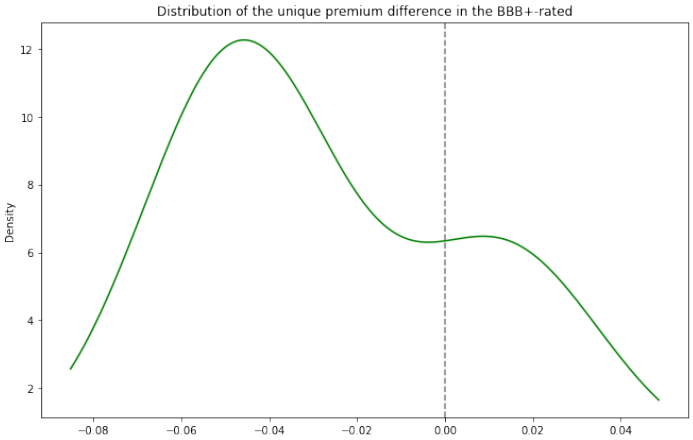
Distribution of the unique premium difference in the A-rated



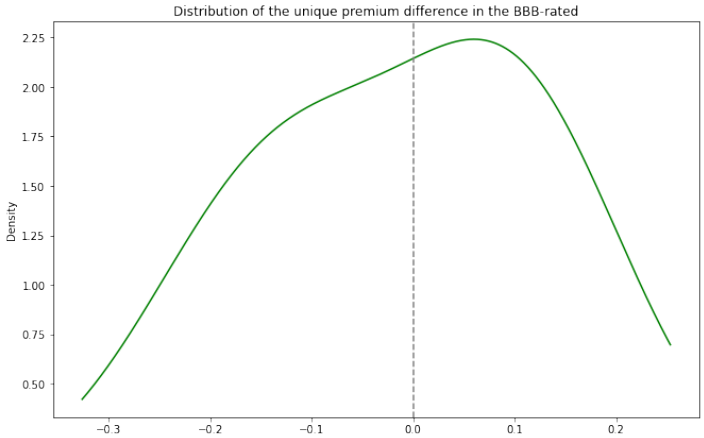
Distribution of the unique premium difference in the A--rated



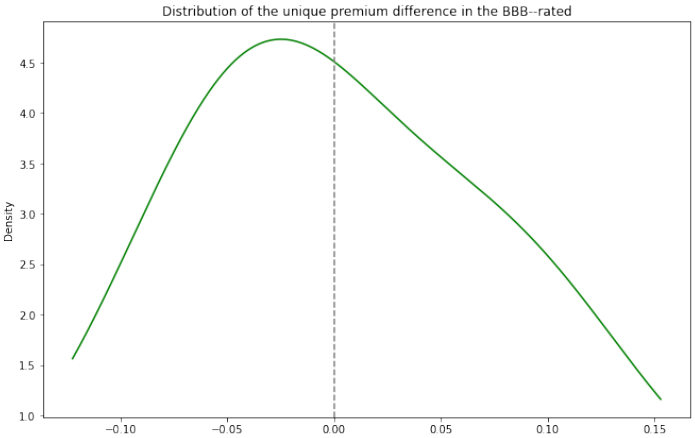
Distribution of the unique premium difference in the BBB+-rated



Distribution of the unique premium difference in the BBB-rated

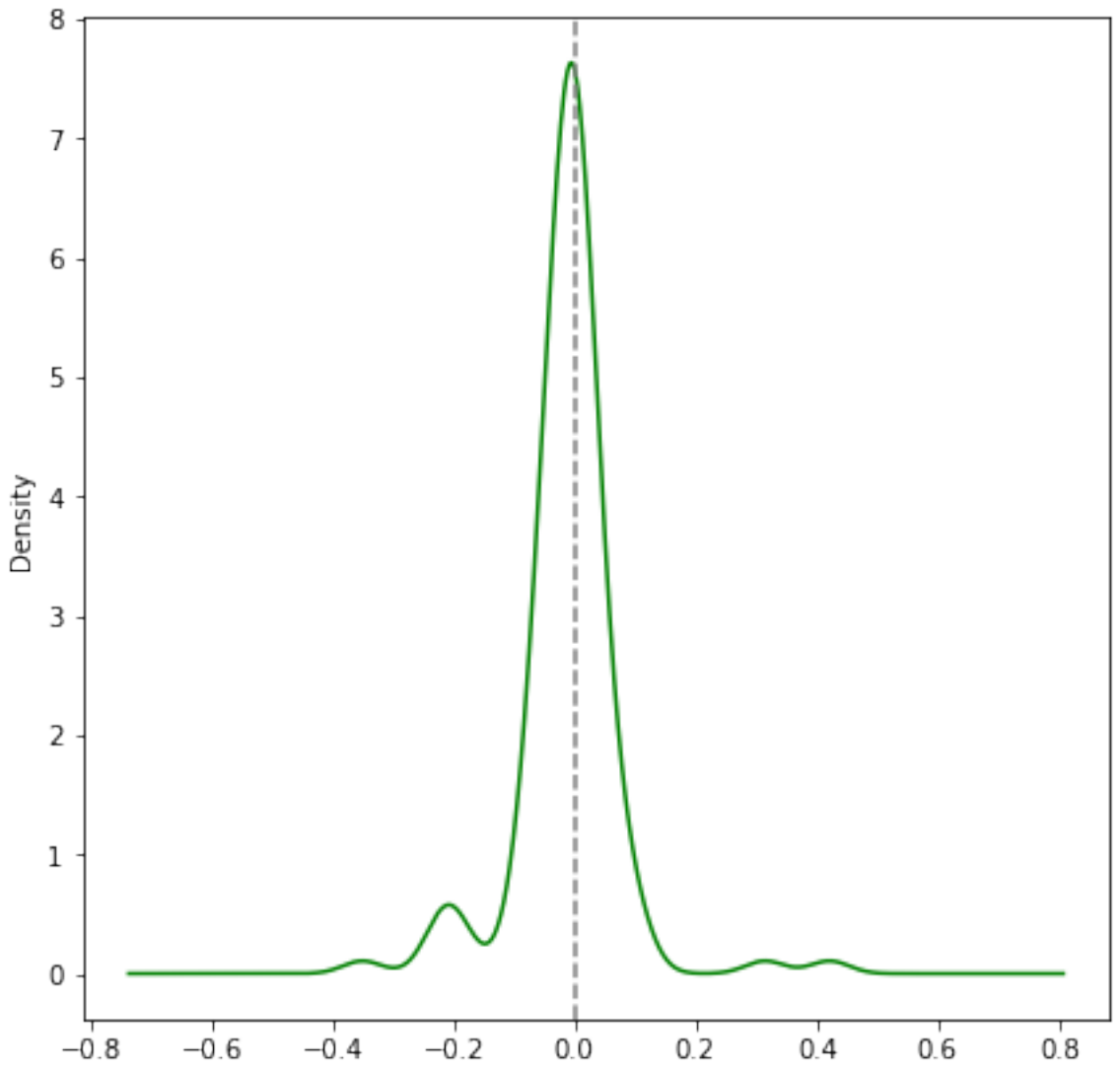


Distribution of the unique premium difference in the BBB--rated



In [737]:

```
plt.figure(figsize=(7,7))
df1.alpha.plot.kde(color='g')
plt.axvline(x=0, color='grey', linestyle='--');
```



Alpha on Use of Proceeds:

In [738]:

```
reg_uop = sm.OLS(df1.alpha, exog=df1[['cons', 'Adaptation', 'Industry', 'Waste
', 'Buildings',
       'Energy', 'Land Use', 'Transport', 'Water',]], missing='drop').fit().su
mmary2()
reg_uop
```

Out[738]:

Model:	OLS	Adj. R-squared:	0.034
Dependent Variable:	alpha	AIC:	-244.2911
Date:	2019-06-17 17:26	BIC:	-219.4316
No. Observations:	117	Log-Likelihood:	131.15
Df Model:	8	F-statistic:	1.513
Df Residuals:	108	Prob (F-statistic):	0.161
R-squared:	0.101	Scale:	0.0067404

	Coef.	**Std.Err.**	**t**	**P>|t|**	**[0.025**	**0.975]**
cons	-0.0024	0.0203	-0.1205	0.9043	-0.0427	0.0378
Adaptation	-0.0273	0.0321	-0.8521	0.3961	-0.0909	0.0363
Industry	0.1097	0.0455	2.4087	0.0177	0.0194	0.2000
Waste	0.0123	0.0266	0.4617	0.6452	-0.0404	0.0649
Buildings	0.0081	0.0173	0.4661	0.6421	-0.0262	0.0424
Energy	-0.0312	0.0197	-1.5833	0.1163	-0.0704	0.0079
Land Use	0.0228	0.0278	0.8181	0.4151	-0.0324	0.0779
Transport	-0.0120	0.0213	-0.5621	0.5752	-0.0542	0.0302
Water	0.0201	0.0254	0.7917	0.4303	-0.0302	0.0705

Omnibus:	26.004	Durbin-Watson:	2.097
Prob(Omnibus):	0.000	Jarque-Bera (JB):	226.200
Skew:	0.077	Prob(JB):	0.000
Kurtosis:	9.810	Condition No.:	11

In [837]:

```
df1 = df1.loc[:,~df1.columns.duplicated()] # remove duplicates
```

In [838]:

```
df['Principal Currency'].unique()
```

Out[838]:

```
array(['US Dollar', 'Swedish Krona', 'Euro', 'Hong Kong Dollar',
       'Indian Rupee', 'British Pound', 'Mexican Peso', 'Canadian
Dollar',
       'Norwegian Krone', 'Swiss Franc', 'Australian Dollar'],
      dtype=object)
```

In [839]:

```
df1["Financial Sector"] = df1["Banking"] + df1["Financial - Other"] + df1["Mor
tgage Banking"]
df1["Goverment Sector"] = df1["Supranational"] + df1["Official and Muni"] + df
1["Agency"] + df1["Sovereign"]
df1["Real Estate Sector"] = df1["Real Estate Investment Trust"] + df1["Home Bu
ilders"]
df1["Technology"] = df1["Electronics"]
df1["Utility"] = df1["Utility - Other"]
```

In [840]:

```
df2["Energy"].sum()
```

Out[840]:

```
90
```

In [743]:

```
df1 = df1[(df1["Euro"] == 1) | (df1["Australian Dollar"] == 1) | (df1["US Doll
ar"] == 1) | (df1["Swedish Krona"] == 1)]
df1['cons'] = 1
df1 = df1[(df1["Financial Sector"] == 1) | (df1["Goverment Sector"] == 1) |(df
1["Real Estate Sector"] == 1) | (df1["Industry"] == 1)]
```

In [813]:

```
df1['EnergyBuildings'] = df1['Energy']*df1['Buildings']
df1['EnergyAdaptation'] = df1['Energy']*df1['Adaptation']
df1['EnergyLanduse'] = df1['Energy']*df1['Land Use']
df1['EnergyIndustry'] = df1['Energy']*df1['Industry']
df1['EnergyTransport'] = df1['Energy']*df1['Transport']
df1['EnergyWaste'] = df1['Energy']*df1['Waste']
df1['EnergyWater'] = df1['Energy']*df1['Water']
print('Buildings: '+ str(df1['EnergyBuildings'].sum()) + "/"+ str(df1['Buildin
gs'].sum()))
print('Adaptation: '+ str(df1['EnergyAdaptation'].sum())+ "/"+ str(df1['Adapta
tion'].sum()))
print('Land Use: '+ str(df1['EnergyLanduse'].sum())+ "/"+ str(df1['Land Use'].
sum()))
print('Industry: '+ str(df1['EnergyIndustry'].sum())+ "/"+ str(df1['Industry']
.sum()))
print('Transport: '+ str(df1['EnergyTransport'].sum())+ "/"+ str(df1['Transpor
t'].sum()))
print('Waste: '+ str(df1['EnergyWaste'].sum())+ "/"+ str(df1['Waste'].sum()))
print('Water: '+ str(df1['EnergyWater'].sum())+ "/"+ str(df1['Water'].sum()))
```

```
Buildings: 50/66
Adaptation: 12/14
Land Use: 23/23
Industry: 2/2
Transport: 44/48
Waste: 21/21
Water: 31/36
```

In [814]:

```
df1['cons'] = 1
df2['cons'] = 1
# Create lists of variables to be used in each regression
X1 = ['cons', 'Adaptation', 'Industry', 'Waste', 'Buildings','Energy', 'Land U
se', 'Transport', 'Water', 'CBI certified']
X2 = ['cons', 'Time to maturity', 'Log_AmountIssued','AA-nest', 'A-nest','BBB-
nest', 'Financial Sector',  'Real Estate Sector','Australian Dollar','Euro','S
wedish Krona','Adaptation', 'Waste', 'Buildings','Energy', 'Land Use', 'Transp
ort', 'Water', 'CBI certified']
X3 = ['cons', 'Time to maturity', 'Log_AmountIssued', 'Rating', 'Financial Sec
tor', 'Real Estate Sector','Australian Dollar','Euro','Swedish Krona','Adaptat
ion', 'Waste', 'Buildings','Energy', 'Land Use', 'Transport', 'Water', 'CBI ce
rtified']
X4 = ['cons', "AA-nest", 'A-nest', 'BBB-nest']
X5 = ['cons', 'EnergyBuildings', 'EnergyAdaptation', 'EnergyLanduse', 'EnergyI
ndustry', 'EnergyTransport', 'EnergyWaste', 'EnergyWater']
# Estimate an OLS regression for each set of variables
reg1 = sm.OLS(df2['alpha'], df2[X1], missing='drop').fit(cov_type='HC3')
reg2 = sm.OLS(df1['alpha'], df1[X2], missing='drop').fit(cov_type='HC3')
reg3 = sm.OLS(df1['alpha'], df1[X3], missing='drop').fit(cov_type='HC3')
reg4 = sm.OLS(df1['alpha'], df1[X4], missing='drop').fit(cov_type='HC3')
reg5 = sm.OLS(df1['alpha'], df1[X5], missing='drop').fit(cov_type='HC3')
```

In [816]:

```
from statsmodels.iolib.summary2 import summary_col

info_dict={'R-squared' : lambda x: f"{x.rsquared:.2f}",
           'No. observations' : lambda x: f"{int(x.nobs):d}"}

results_table = summary_col(results=[reg1, reg2, reg5],
                            float_format='%0.4f',
                            stars = True,

                            info_dict=info_dict,
                            regressor_order=['cons', 'Time to maturity', 'Log_
AmountIssued', 'Rating', 'AA-nest','A-nest','BBB-nest', 'Financial Sector',  '
Real Estate Sector','Technology','Utility', 'Australian Dollar','Euro','Swedis
h Krona','Adaptation','Industry', 'Waste', 'Buildings','Energy', 'Land Use', '
Transport', 'Water', 'CBI certified'])

results_table.add_title('Dependent variable: alpha')

print(results_table)


beginningtex = """\\documentclass{report}
\\usepackage{booktabs}
\\begin{document}"""
endtex = "\end{document}"

f = open('Part2regression.tex', 'w')

f.write(results_table.as_latex())

f.close()
```

```
            Dependent variable: alpha
================================================
                    alpha I   alpha II  alpha III
------------------------------------------------
cons                -0.0033   0.0190    -0.0206**
                    (0.0161)  (0.0279)  (0.0091)
Time to maturity              0.0022
                              (0.0043)
Log_AmountIssued              -0.0157
                              (0.0157)
AA-nest                       0.0303*
                              (0.0182)
A-nest                        -0.0280
                              (0.0232)
BBB-nest                      -0.0092
                              (0.0309)
Financial Sector              -0.0291
                              (0.0258)
Real Estate Sector            -0.0107
                              (0.0504)
Australian Dollar             -0.0326
                              (0.0351)
Euro                          -0.0054
```

```
                          (0.0207)
Swedish Krona             -0.0272
                          (0.0332)
Adaptation        -0.0275  -0.0493
                 (0.0380) (0.0310)
Industry          0.1120
                 (0.1044)
Waste             0.0150   0.0441
                 (0.0358) (0.0370)
Buildings         0.0062   -0.0116
                 (0.0146) (0.0188)
Energy           -0.0323** -0.0495***
                 (0.0149) (0.0189)
Land Use          0.0260   0.0067
                 (0.0257) (0.0225)
Transport        -0.0136   -0.0016
                 (0.0299) (0.0263)
Water             0.0210   0.0000
                 (0.0251) (0.0252)
CBI certified     0.0149   0.0349
                 (0.0177) (0.0270)
EnergyAdaptation                    -0.0496
                                    (0.0397)
EnergyBuildings                     -0.0051
                                    (0.0195)
EnergyIndustry                      0.0258
                                    (0.0385)
EnergyLanduse                       0.0039
                                    (0.0251)
EnergyTransport                     0.0038
                                    (0.0364)
EnergyWaste                         0.0358
                                    (0.0312)
EnergyWater                         -0.0019
                                    (0.0300)
R-squared          0.10      0.19       0.06
No. observations   117       101        101
================================================
Standard errors in parentheses.
* p<.1, ** p<.05, ***p<.01
```

In [810]:

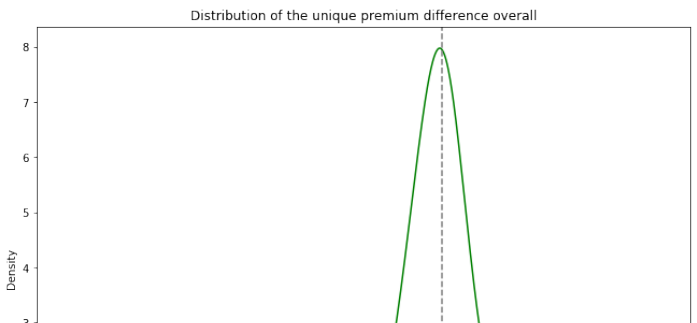
```
reg2.fvalue, reg2.f_pvalue
```

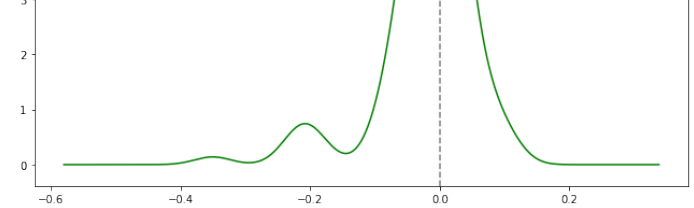
Out[810]:

```
(array([[1.11848296]]), array(0.35024366))
```

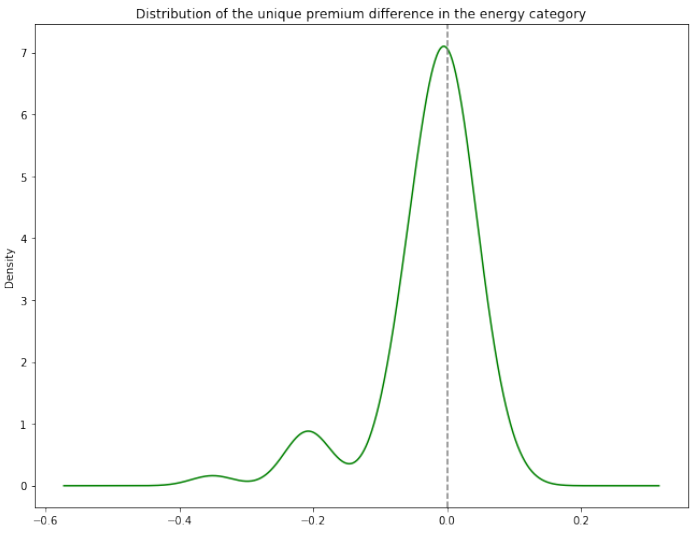
In [748]:

```
plt.figure(figsize=(24,48))
plt.subplot(521)
df1.alpha.plot.kde(color='g')
plt.axvline(x=0, color='grey', linestyle='--');
plt.title("Distribution of the unique premium difference overall")

plt.subplot(523)
df1.alpha[df1.Energy==1].plot.kde(color='g')
plt.title("Distribution of the unique premium difference in the energy categor
y")
plt.axvline(x=0, color='grey', linestyle='--');
plt.subplot(524)
df1.alpha[df1.Buildings==1].plot.kde(color='g')
plt.title("Distribution of the unique premium difference in the building categ
ory")
plt.axvline(x=0, color='grey', linestyle='--');
plt.subplot(525)
df1.alpha[df1.Adaptation==1].plot.kde(color='g')
plt.title("Distribution of the unique premium difference in the adaptation cat
egory")
plt.axvline(x=0, color='grey', linestyle='--');
plt.subplot(526)
df1.alpha[df1.Industry==1].plot.kde(color='g')
plt.title("Distribution of the unique premium difference in the industry categ
ory")
plt.axvline(x=0, color='grey', linestyle='--');
plt.subplot(527)
df1.alpha[df1.Waste==1].plot.kde(color='g')
plt.title("Distribution of the unique premium difference in the waste category
")
plt.axvline(x=0, color='grey', linestyle='--');
plt.subplot(528)
df1.alpha[df1["Land Use"]==1].plot.kde(color='g')
plt.title("Distribution of the unique premium difference in the land use categ
ory")
plt.axvline(x=0, color='grey', linestyle='--');
plt.subplot(529)
df1.alpha[df1.Transport==1].plot.kde(color='g')
plt.title("Distribution of the unique premium difference in the Transport cate
gory")
plt.axvline(x=0, color='grey', linestyle='--');
plt.subplot(5,2,10)
df1.alpha[df1.Water==1].plot.kde(color='g')
plt.title("Distribution of the unique premium difference in the Water category
")
plt.axvline(x=0, color='grey', linestyle='--');
plt.savefig("Alpha distribution UoP Appendix.PNG")
```

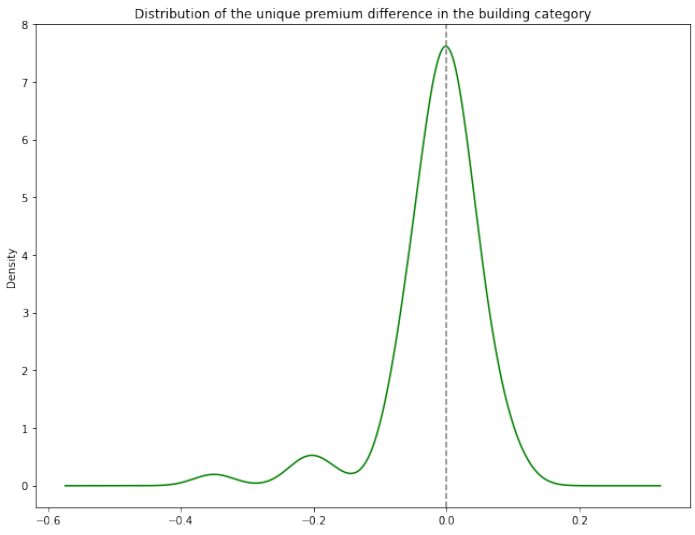




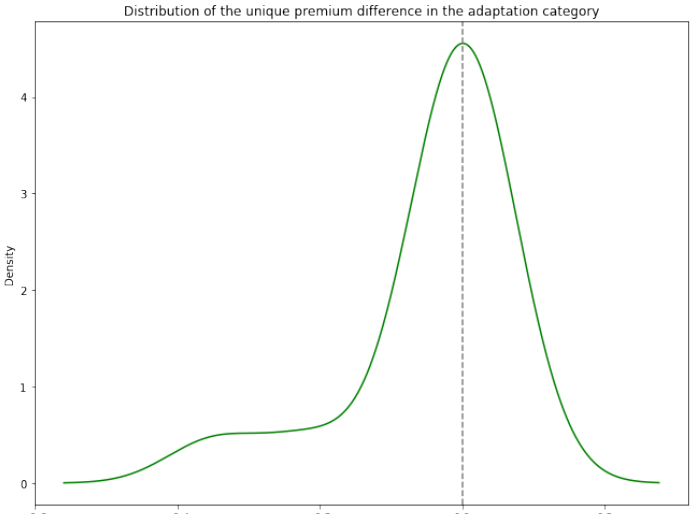
Distribution of the unique premium difference in the energy category



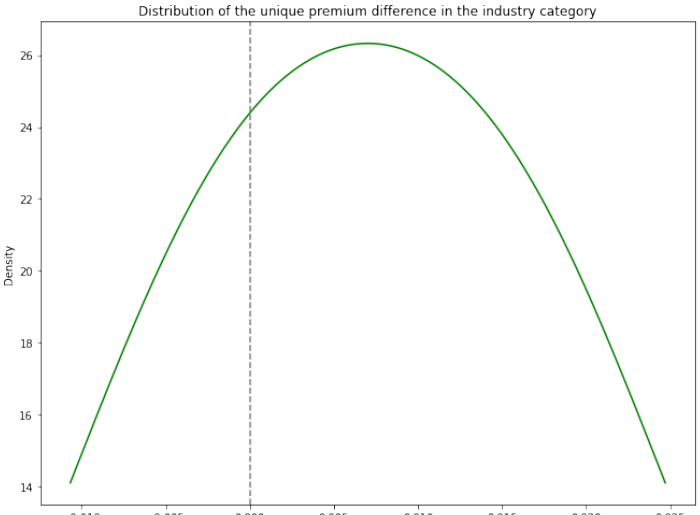
Distribution of the unique premium difference in the building category



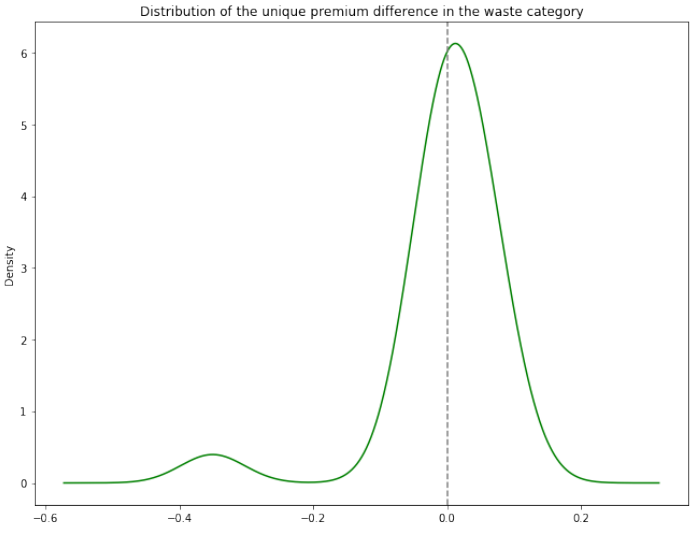
Distribution of the unique premium difference in the adaptation category



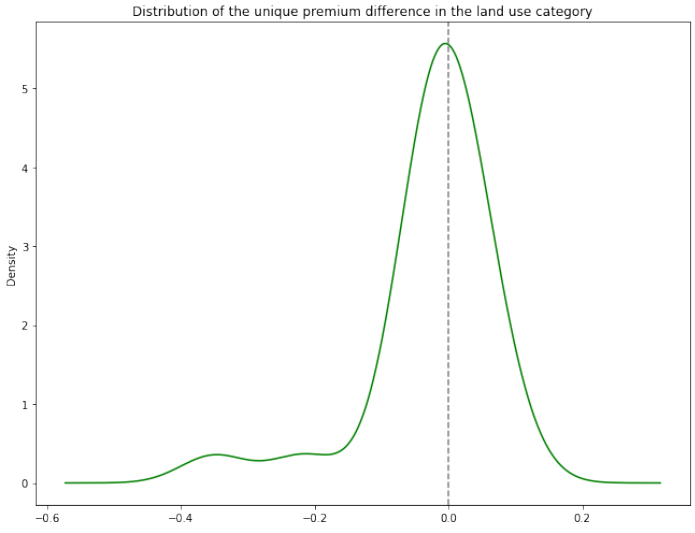
Distribution of the unique premium difference in the industry category



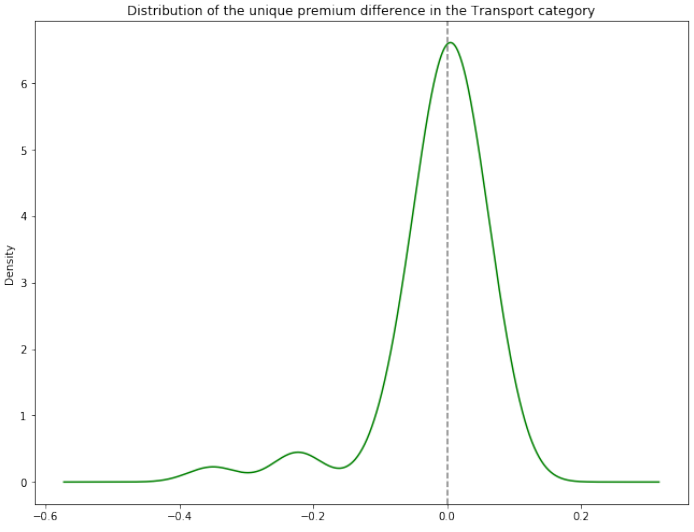
Distribution of the unique premium difference in the waste category



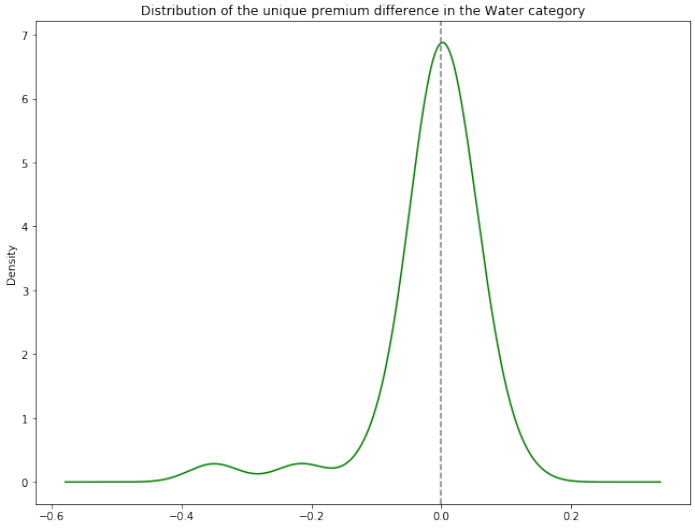
Distribution of the unique premium difference in the land use category



Distribution of the unique premium difference in the Transport category



Distribution of the unique premium difference in the Water category



In [609]:

```
print(str(np.sum(df1.Transport*df1.Water)))
print(str(np.sum(df1.Transport)))
print(str(np.sum(df1.Water)))
print(str(np.sum(df1.Industry)))
```

```
34
53
43
5
```

In [610]:

```
df1[df1.Industry==1]
```

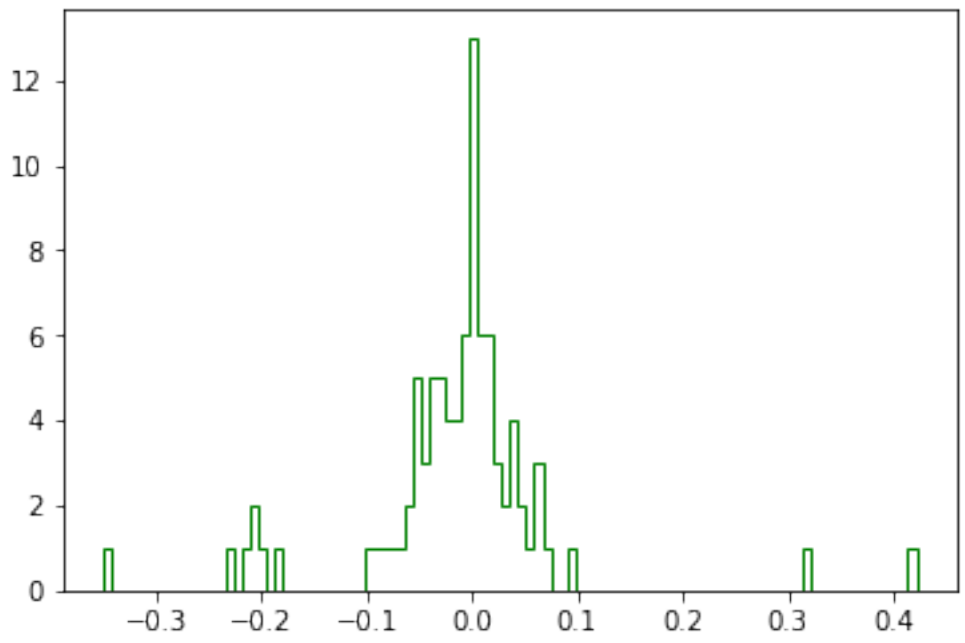
Out[610]:

	Time to maturity	y	x	CBI certified	Coupon	Amount Issued	Out
US045167CY77	5.931507	0.015854	0.000048	0	2.125	500000000	500
US045167EJ82	9.457534	-0.001793	-0.000234	0	3.125	750000000	750
XS1655322953	2.315068	0.420933	0.001224	0	6.200	3200000000	459
XS1796211933	0.936986	0.000140	0.000012	0	1.950	100000000	127
XS1814390099	1.043836	-0.002711	0.000007	0	2.250	100000000	127

5 rows × 96 columns

In [611]:

```
mod_energy = PanelOLS(df.y[df.Energy == 1], df.x[df.Energy == 1],entity_effect
s=True)
alfae = (df.y[df.y.notnull() & df.Energy == 1] - df.x[df.y.notnull() & df.Ener
gy == 1]*mod_energy.fit().params[0]-mod_energy.fit().resids).groupby('ISIN').m
ean()
(plt.hist(alfae, bins=100, histtype="step", color ="g"));
```



In [612]:

```
res = PanelOLS.from_formula('y ~x +EntityEffects', df).fit()
res
```

Out[612]:

PanelOLS Estimation Summary

Dep. Variable:	y	**R-squared:**	0.0618
Estimator:	PanelOLS	**R-squared (Between):**	0.0836
No. Observations:	28482	**R-squared (Within):**	0.0618
Date:	Mon, Jun 17 2019	**R-squared (Overall):**	0.0964
Time:	14:50:09	**Log-likelihood**	3.739e+04
Cov. Estimator:	Unadjusted		
		F-statistic:	1867.6
Entities:	117	**P-value**	0.0000
Avg Obs:	243.44	**Distribution:**	F(1,28364)
Min Obs:	3.0000		
Max Obs:	607.00	**F-statistic (robust):**	1867.6
		P-value	0.0000
Time periods:	607	**Distribution:**	F(1,28364)
Avg Obs:	46.923		
Min Obs:	12.000		
Max Obs:	101.00		

Parameter Estimates

	Parameter	Std. Err.	T-stat	P-value	Lower CI	Upper CI
x	-0.1866	0.0043	-43.216	0.0000	-0.1950	-0.1781

F-test for Poolability: 256.05
P-value: 0.0000
Distribution: F(116,28364)

Included effects: Entity
id: 0x2a90adfc470

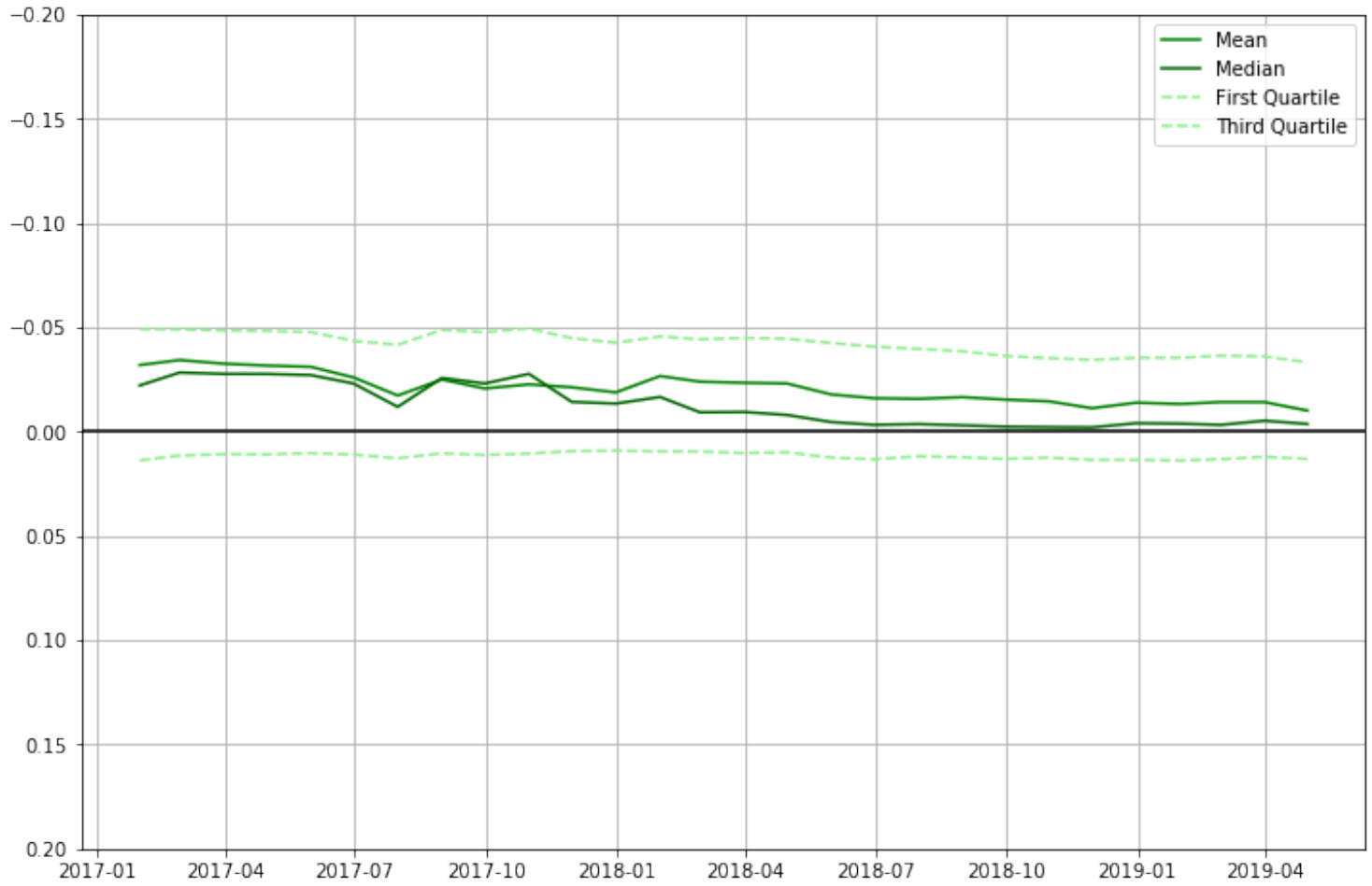
In [613]:

```
#ploti = plt.Line(res.estimated_effects.groupby('date').mean()['estimated_effe
cts'], res.estimated_effects.groupby('date').mean().index, color='g')
plt.figure(figsize=(12,8))
plt.tick_params(top='off', bottom='on', left='on', right='off', labelleft='on'
, labelbottom='on')
plt.plot(res.estimated_effects.groupby('date').mean().resample('M').mean(), co
lor="green")
plt.plot(res.estimated_effects.groupby('date').median().resample('M').mean(),
color="darkgreen")
plt.plot(res.estimated_effects.groupby('date').quantile(0.25).resample('M').me
an(), linestyle='dashed', color="lightgreen")
plt.plot(res.estimated_effects.groupby('date').quantile(0.75).resample('M').me
an(), linestyle='dashed', color="lightgreen")
plt.ylim(0.2, -0.2)
plt.grid(b=True)
plt.axhline(y=0, color='black', linestyle='-')
plt.legend(["Mean", "Median", "First Quartile", "Third Quartile"])
```

```
C:\Users\eoest\Anaconda3\lib\site-packages\matplotlib\cbook\deprec
ation.py:107: MatplotlibDeprecationWarning:

Passing one of 'on', 'true', 'off', 'false' as a boolean is deprec
ated; use an actual boolean (True/False) instead.

C:\Users\eoest\Anaconda3\lib\site-packages\matplotlib\cbook\deprec
ation.py:107: MatplotlibDeprecationWarning:

Passing one of 'on', 'true', 'off', 'false' as a boolean is deprec
ated; use an actual boolean (True/False) instead.

C:\Users\eoest\Anaconda3\lib\site-packages\matplotlib\cbook\deprec
ation.py:107: MatplotlibDeprecationWarning:

Passing one of 'on', 'true', 'off', 'false' as a boolean is deprec
ated; use an actual boolean (True/False) instead.

C:\Users\eoest\Anaconda3\lib\site-packages\matplotlib\cbook\deprec
ation.py:107: MatplotlibDeprecationWarning:

Passing one of 'on', 'true', 'off', 'false' as a boolean is deprec
ated; use an actual boolean (True/False) instead.

C:\Users\eoest\Anaconda3\lib\site-packages\matplotlib\cbook\deprec
ation.py:107: MatplotlibDeprecationWarning:

Passing one of 'on', 'true', 'off', 'false' as a boolean is deprec
ated; use an actual boolean (True/False) instead.

C:\Users\eoest\Anaconda3\lib\site-packages\matplotlib\cbook\deprec
ation.py:107: MatplotlibDeprecationWarning:

Passing one of 'on', 'true', 'off', 'false' as a boolean is deprec
ated; use an actual boolean (True/False) instead.
```

Out[613]:

<matplotlib.legend.Legend at 0x2a9023b5b38>



In [614]:

```
res.estimated_effects.groupby('date').quantile(0.25).resample('M').mean()
```

Out[614]:

0.25	estimated_effects
date	
2017-01-31	-0.049131
2017-02-28	-0.049058
2017-03-31	-0.048494
2017-04-30	-0.048313
2017-05-31	-0.047716
2017-06-30	-0.043511
2017-07-31	-0.041613
2017-08-31	-0.048761
2017-09-30	-0.047735
2017-10-31	-0.049495
2017-11-30	-0.044808
2017-12-31	-0.042631
2018-01-31	-0.045627
2018-02-28	-0.044244
2018-03-31	-0.044899
2018-04-30	-0.044533
2018-05-31	-0.042494
2018-06-30	-0.040750
2018-07-31	-0.039668
2018-08-31	-0.038525
2018-09-30	-0.036278
2018-10-31	-0.035220
2018-11-30	-0.034407
2018-12-31	-0.035405
2019-01-31	-0.035447
2019-02-28	-0.036424
2019-03-31	-0.036057
2019-04-30	-0.033228

In [615]:

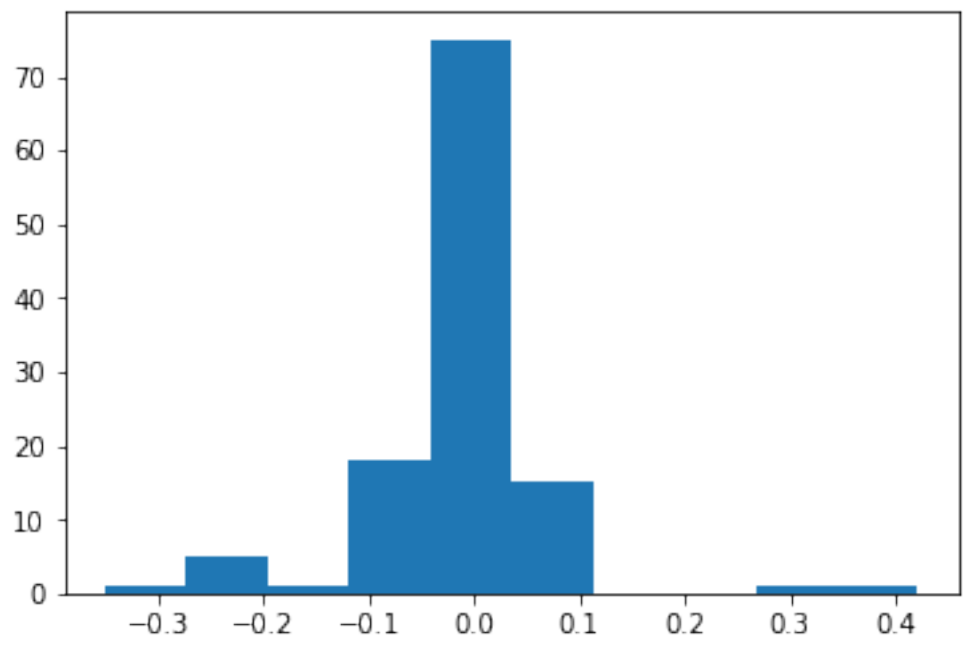
```
tulleset = [0, 1, 2, 3, 4, 5, 6]
tulleset = pd.DataFrame(tulleset)
tulleset.quantile(0.25)
```

Out[615]:

```
0    1.5
Name: 0.25, dtype: float64
```

In [432]:

```
hist1 = (res.estimated_effects.groupby('ISIN').mean())
plt.hist(hist1['estimated_effects'], bins = 10);
```



In [433]:

```
res.resids.groupby("date").mean().resample('M').mean()
```

Out[433]:

```
date
2017-01-31   -0.003406
2017-02-28   -0.009420
2017-03-31    0.001549
2017-04-30    0.006588
2017-05-31   -0.005174
2017-06-30    0.007023
2017-07-31   -0.005214
2017-08-31   -0.002957
2017-09-30   -0.000107
2017-10-31    0.002771
2017-11-30    0.005549
2017-12-31    0.004492
2018-01-31    0.007929
2018-02-28    0.013299
2018-03-31    0.015573
2018-04-30    0.012989
2018-05-31   -0.001235
2018-06-30   -0.000713
2018-07-31    0.001374
2018-08-31   -0.004526
2018-09-30   -0.002802
2018-10-31   -0.000056
2018-11-30    0.002854
2018-12-31   -0.004355
2019-01-31   -0.006423
2019-02-28    0.001288
2019-03-31   -0.003996
2019-04-30   -0.006475
Freq: M, Name: residual, dtype: float64
```

In [434]:

```
res.resids.groupby('date').mean();
```

In [435]:

```
res.predict().groupby("ISIN").mean().mean()
```

Out[435]:

```
fitted_values   -0.001201
dtype: float64
```

In [436]:

```
print(res)
```

PanelOLS Estimation Summary

Dep. Variable:	y	R-squared:	0.0618
Estimator:	PanelOLS	R-squared (Between):	0.0836
No. Observations:	28482	R-squared (Within):	0.0618
Date:	Mon, Jun 17 2019	R-squared (Overall):	0.0964
Time:	14:14:05	Log-likelihood	3.739e+04
Cov. Estimator:	Unadjusted		
		F-statistic:	1867.6
Entities:	117	P-value	0.0000
Avg Obs:	243.44	Distribution:	F(1,28364)
Min Obs:	3.0000		
Max Obs:	607.00	F-statistic (robust):	1867.6
		P-value	0.0000
Time periods:	607	Distribution:	F(1,28364)
Avg Obs:	46.923		
Min Obs:	12.000		
Max Obs:	101.00		

Parameter Estimates

	Parameter	Std. Err.	T-stat	P-value	Lower CI	Upper CI
x	-0.1866	0.0043	-43.216	0.0000	-0.1950	-0.1781

F-test for Poolability: 256.05
P-value: 0.0000
Distribution: F(116,28364)

Included effects: Entity

Split by instituitons:

In [437]:

```
mod_gov = PanelOLS.from_formula('y ~1 + x + EntityEffects', gov)
res_gov = mod_gov.fit()
```

```
C:\Users\eoest\Anaconda3\lib\site-packages\linearmodels\utility.py
:476: MissingValueWarning:


Inputs contain missing values. Dropping rows with missing observat
ions.
```

In [438]:

```
print(mod_gov.fit())
```

PanelOLS Estimation Summary

Dep. Variable:	y	R-squared:	0.0700
Estimator:	PanelOLS	R-squared (Between):	0.0891
No. Observations:	15160	R-squared (Within):	0.0700
Date:	Mon, Jun 17 2019	R-squared (Overall):	0.1133
Time:	14:14:08	Log-likelihood	1.923e+04
Cov. Estimator:	Unadjusted		
		F-statistic:	1136.0
Entities:	69	P-value	0.0000
Avg Obs:	219.71	Distribution:	F(1,15090)
Min Obs:	3.0000		
Max Obs:	594.00	F-statistic (robust):	1136.0
		P-value	0.0000
Time periods:	607	Distribution:	F(1,15090)
Avg Obs:	24.975		
Min Obs:	3.0000		
Max Obs:	61.000		

Parameter Estimates

	Parameter	Std. Err.	T-stat	P-value	Lower CI	Upper CI
Intercept	-0.0086	0.0006	-15.380	0.0000	-0.0097	-0.0075
x	-0.2443	0.0072	-33.705	0.0000	-0.2586	-0.2301

F-test for Poolability: 187.02
P-value: 0.0000
Distribution: F(68,15090)

Included effects: Entity

In [439]:

```
mod_corp = PanelOLS.from_formula('y ~1 + x + EntityEffects', corp)
res_corp = mod_corp.fit()
```

```
C:\Users\eoest\Anaconda3\lib\site-packages\linearmodels\utility.py
:476: MissingValueWarning:


Inputs contain missing values. Dropping rows with missing observat
ions.
```

In [440]:

```
print(mod_corp.fit())
```

PanelOLS Estimation Summary

Dep. Variable:	y	R-squared:	0.0592
Estimator:	PanelOLS	R-squared (Between):	0.0714
No. Observations:	13322	R-squared (Within):	0.0592
Date:	Mon, Jun 17 2019	R-squared (Overall):	0.0878
Time:	14:14:08	Log-likelihood	1.83e+04
Cov. Estimator:	Unadjusted		
		F-statistic:	834.60
Entities:	48	P-value	0.0000
Avg Obs:	277.54	Distribution:	F(1,13273)
Min Obs:	23.000		
Max Obs:	607.00	F-statistic (robust):	834.60
		P-value	0.0000
Time periods:	607	Distribution:	F(1,13273)
Avg Obs:	21.947		
Min Obs:	4.0000		
Max Obs:	44.000		

Parameter Estimates

	Parameter	Std. Err.	T-stat	P-value	Lower CI	Upper CI
Intercept	-0.0269	0.0005	-50.528	0.0000	-0.0279	-0.0258
x	-0.1500	0.0052	-28.890	0.0000	-0.1602	-0.1398

F-test for Poolability: 364.30
P-value: 0.0000
Distribution: F(47,13273)

Included effects: Entity