In [4]:

```
import numpy as np
import pandas as pd
import seaborn as sns

from sklearn.calibration import calibration_curve
from sklearn.model_selection import cross_val_score,cross_val_predict, train_test_split
from sklearn.preprocessing import OneHotEncoder,StandardScaler,PowerTransformer,LabelEncoder
from sklearn.compose import make_column_transformer
from sklearn.pipeline import make_pipeline
from sklearn.metrics import accuracy_score,classification_report, recall_score,confusion_matrix, roc_auc_score, precision_score, f
from sklearn.calibration import CalibratedClassifierCV
from sklearn.linear_model import LogisticRegression

import matplotlib.pyplot as plt

from scipy.sparse import csr_matrix

import optuna
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
from catboost import CatBoostClassifier


#importing plotly and cufflinks in offline mode
import cufflinks as cf
import plotly.offline
cf.go_offline()
cf.set_config_file(offline=False, world_readable=True)


import plotly
import plotly.express as px
import plotly.graph_objs as go
import plotly.offline as py
from plotly.offline import iplot
from plotly.subplots import make_subplots
import plotly.figure_factory as ff

import shap

import missingno as msno
```

```
import warnings
warnings.filterwarnings("ignore")
```

In [5]:

```
eval_df = pd.read_csv(r'C:\Users\esp1_\OneDrive\Skrivebord\thesis\df_last_two_cycles.csv', low_memory = False)
df = pd.read_csv(r'C:\Users\esp1_\OneDrive\Skrivebord\thesis\df_first_two_cycles.csv', low_memory = False)

categorical = df.select_dtypes('object').columns
categorical = categorical.tolist()
df[categorical] = df[categorical].fillna('missing')

df1 = df.drop(['contact_id', 'Inscrito', 'Admitido', 'Useless', 'Contacto', 'Asistencia', 'Cita', 'no contactado', 'original_sourc

categorical = eval_df.select_dtypes('object').columns
categorical = categorical.tolist()
eval_df[categorical] = eval_df[categorical].fillna('missing')

df2 = eval_df.drop(['contact_id', 'Inscrito', 'Admitido', 'Useless', 'Contacto', 'Asistencia', 'Cita', 'no contactado', 'original_
```

In [6]:

```
columns = ['male',
       'first_view_to_first_conversion_date', 'first_view_to_last_view',
       'first_view_to_time_of_first_session',
       'first_view_to_time_of_last_session',
       'first_view_to_date_of_last_interaction', 'first_view_to_last_contact',
       'first_view_to_date_of_last_activity',
       'first_view_to_recent_conversion_date',
       'first_view_to_first_marketing_mail_shipping',
       'first_marketing_mail_shipping_to_first_marketing_mail_open',
       'first_marketing_mail_shipping_to_click_on_marketing_mail',
       'first_marketing_mail_shipping_to_date_of_last_click_on_marketing_mail',
       'first_marketing_mail_shipping_to_date_of_last_marketing_mail_open']
for col in df1[columns]:
    df1[col+"_missing"] = df1[col].isnull()
for col in df2[columns]:
    df2[col+"_missing"] = df2[col].isnull()

df1 = df1.fillna(df1.median())
df2 = df2.fillna(df2.median())
```

In [7]:

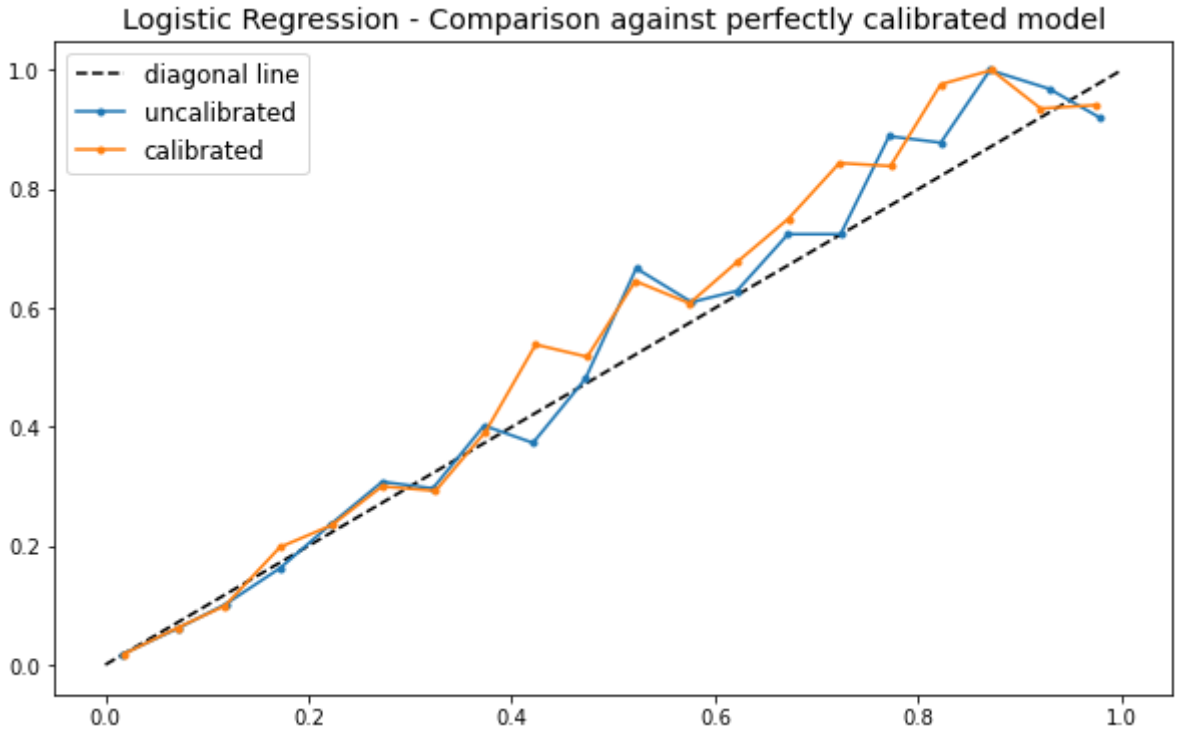
```
df = pd.concat([df1, df2], axis=0)
```

```
df_dummies = pd.get_dummies(df)
df1_dummies_1 = df_dummies.loc[df_dummies['Ciclo Generación_2021-3'] == 1]
df1_dummies_2 = df_dummies.loc[df_dummies['Ciclo Generación_2022-1'] == 1]
df1 = pd.concat([df1_dummies_1,df1_dummies_2])
df2_dummies_1 = df_dummies.loc[df_dummies['Ciclo Generación_2022-2'] == 1]
df2_dummies_2 = df_dummies.loc[df_dummies['Ciclo Generación_2022-3'] == 1]
df2 = pd.concat([df2_dummies_1,df2_dummies_2])

df1 = df1.drop(['Ciclo Generación_2021-3', 'Ciclo Generación_2022-1', 'Ciclo Generación_2022-2', 'Ciclo Generación_2022-3'], axis=
df2 = df2.drop(['Ciclo Generación_2021-3', 'Ciclo Generación_2022-1', 'Ciclo Generación_2022-2', 'Ciclo Generación_2022-3'], axis=
```

In [8]:

```
X = df1.drop('enrolled', axis=1)
y = df1['enrolled']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)


# predict uncalibrated probabilities
def uncalibrated(X_train, X_test, y_train):

    lg = LogisticRegression(random_state=0)
    lg.fit(X_train, y_train)
    return lg.predict_proba(X_test)[:, 1]

# predict calibrated probabilities
def calibrated(X_train, X_test, y_train):

    lg = LogisticRegression(random_state=0) # , scale_pos_weight=9
    calibrated = CalibratedClassifierCV(lg, method='sigmoid', cv=5)
    calibrated.fit(X_train, y_train)
    return calibrated.predict_proba(X_test)[:, 1]


# uncalibrated predictions
yhat_uncalibrated = uncalibrated(X_train, X_test, y_train)
# calibrated predictions
yhat_calibrated = calibrated(X_train, X_test, y_train)
# reliability diagrams
fop_uncalibrated, mpv_uncalibrated = calibration_curve(y_test, yhat_uncalibrated, n_bins=20, normalize=True)
fop_calibrated, mpv_calibrated = calibration_curve(y_test, yhat_calibrated, n_bins=20)
# plot perfectly calibrated
plt.subplots(1, figsize=(10,6))
plt.rcParams['font.size'] = '12'
```

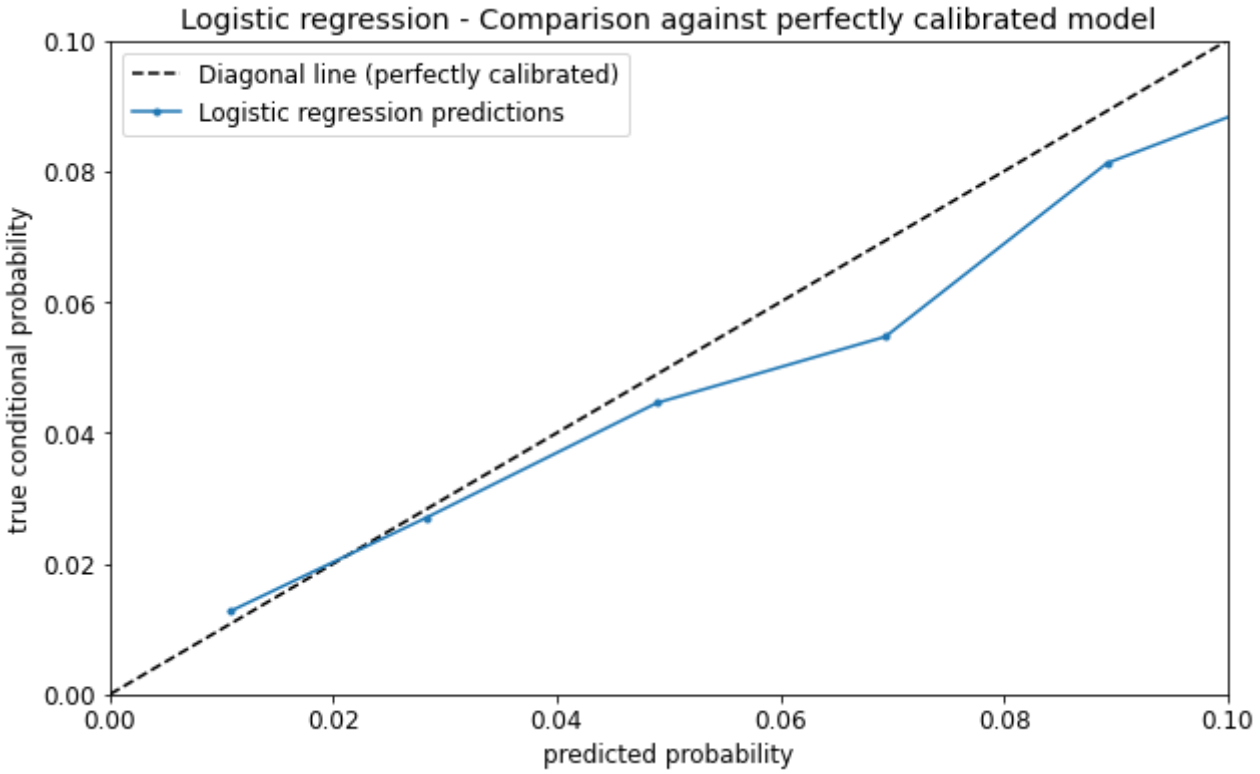
```
plt.title('Logistic Regression - Comparison against perfectly calibrated model')
plt.plot([0, 1], [0, 1], linestyle='--', color='black')
# plot model reliabilities
plt.plot(mpv_uncalibrated, fop_uncalibrated, marker='.')
plt.plot(mpv_calibrated, fop_calibrated, marker='.')
plt.legend(['diagonal line','uncalibrated', 'calibrated'])
plt.show()
```



In [9]:

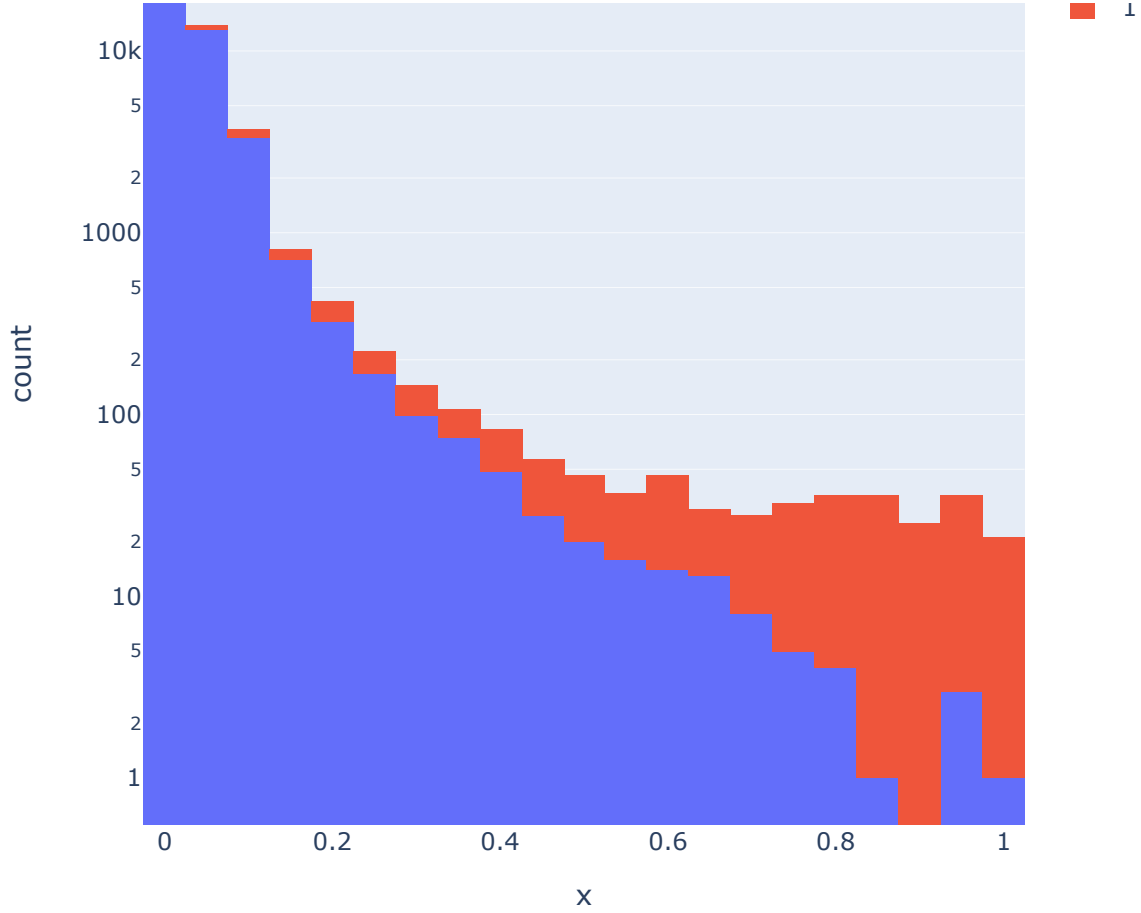
```
# reliability diagrams
fop_uncalibrated, mpv_uncalibrated = calibration_curve(y_test, yhat_uncalibrated, n_bins=50, normalize=True)
#fop_calibrated, mpv_calibrated = calibration_curve(y_test, yhat_calibrated, n_bins=50)
# plot perfectly calibrated
plt.subplots(1, figsize=(10,6))
plt.rcParams['font.size'] = '14'
plt.xlim([0, 0.1])
plt.ylim([0, 0.1])
plt.xlabel("predicted probability")
plt.ylabel("true conditional probability")
plt.rcParams['font.size'] = '12'
```

```
plt.title('Logistic regression - Comparison against perfectly calibrated model')
plt.plot([0, 1], [0, 1], linestyle='--', color='black')
# plot model reliabilities
plt.plot(mpv_uncalibrated, fop_uncalibrated, marker='.')
#plt.plot(mpv_calibrated, fop_calibrated, marker='.')
plt.legend(['Diagonal line (perfectly calibrated)','Logistic regression predictions', ''])
plt.show()
```

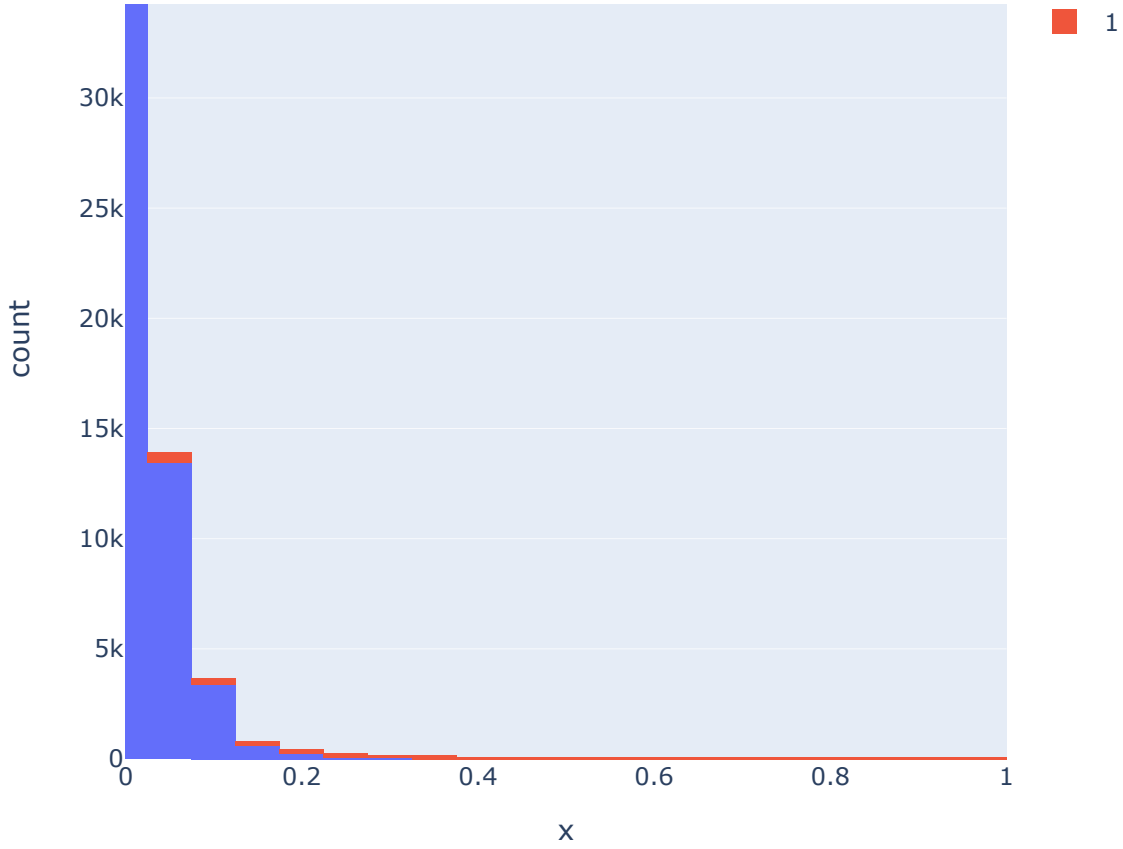


In [10]:

```
fig = px.histogram(df, x=yhat_uncalibrated, color=y_test,width=600, height=600, nbins=20, log_y = True)
fig.show()
```



In [11]:
```
fig = px.histogram(df, x=yhat_uncalibrated, color=y_test,width=600, height=600, nbins=40, range_x=[0,1], title = 'Histogram of Log
fig.show()
```



In [12]:

```
accuracy= []
recall =[]
roc_auc= []
precision = []
model_names =[]

# categorical_features_indices = np.where(X_train.dtypes != np.float)[0]

lg = LogisticRegression(random_state=0)
lg.fit(X_train, y_train)

y_pred1_shap = lg.predict(X_test)
y_pred_shap = lg.predict_proba(X_test)
```

```
accuracy.append(round(accuracy_score(y_test, y_pred1_shap),4))
recall.append(round(recall_score(y_test, y_pred1_shap),4))
roc_auc.append(round(roc_auc_score(y_test, y_pred_shap[:,1]),4))
precision.append(round(precision_score(y_test, y_pred1_shap),4))

model_names = ['Logistic Regression']
result_LG = pd.DataFrame({'Accuracy':accuracy,'Recall':recall, 'Roc_Auc':roc_auc, 'Precision':precision}, index=model_names)
result_LG
```

Out[12]:

	Accuracy	Recall	Roc_Auc	Precision
Logistic Regression	0.9682	0.1401	0.7894	0.7949

In [58]:

```
Leads_costing = 13000

Talent    = 800000
Telco     = 44132
Software  = 35000
Marketing = 1550000
Hardware  = 75000
Total     = Talent + Telco + Software + Marketing + Hardware

Cost_p_lead = (Total/Leads_costing)*-1

Revenue   = 33080
```

In [60]:

```
profit_df
```

Out[60]:

	profit	threshold
0	55485442.17	0.001
1	55544579.58	0.002
2	55498084.65	0.003
3	55489397.56	0.004
4	55399052.16	0.005
...	...	...

	profit	threshold
849	3274149.48	0.85
850	3241069.48	0.851
851	3207989.48	0.852
852	3207989.48	0.853
853	3174909.48	0.854

854 rows × 2 columns

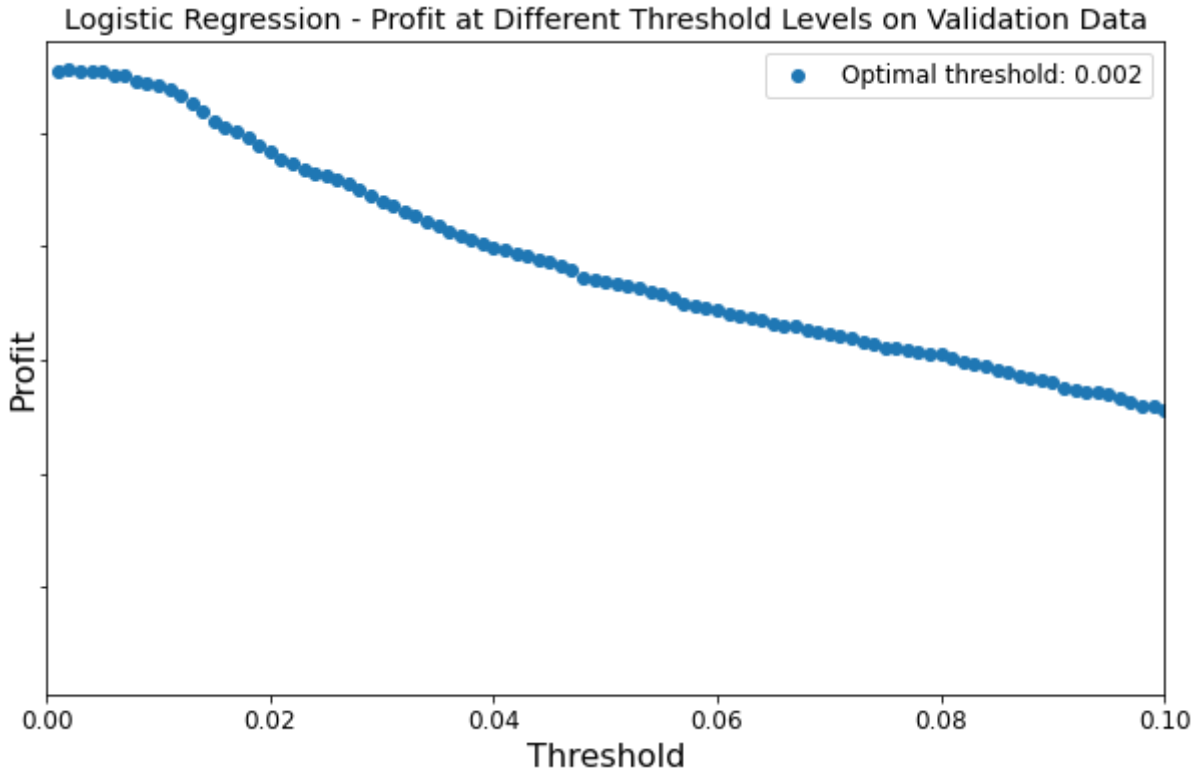
In [ ]:
```
column_names = ['profit', 'threshold']
profit_df = pd.DataFrame(columns = column_names)

for i in range(1000):

    y_pred_opt_1 = (lg.predict_proba(X_test)[:, 1] > (i+1)/1000).astype('float')
    x = confusion_matrix(y_test, y_pred_opt_1)
    FP = x[0,1]*-192.63
    TP = x[1,1]*33080
    profit = TP+FP
    profit_df.loc[i, 'profit'] = profit
    profit_df.loc[i, 'threshold'] = (i+1)/1000
```

In [61]:
```
profit_df['profit'] = pd.to_numeric(profit_df['profit'])
profit_df['threshold'] = pd.to_numeric(profit_df['threshold'])
print(profit_df[['profit']].idxmax())
print(profit_df[['profit']].max())
```

```
profit    1
dtype: int64
profit    55544579.58
dtype: float64
```

In [62]:
```
plt.subplots(1, figsize=(10,6))
plt.rcParams['font.size'] = '12'
plt.title('Logistic Regression - Profit at Different Threshold Levels on Validation Data')
plt.scatter(profit_df['threshold'], profit_df['profit'])
```

```
plt.xlabel("Threshold", fontsize=16)
plt.tick_params(labelleft=False)
plt.ylabel("Profit", fontsize=16)
profit_threshold = profit_df.loc[1,:] # set threshold location found in prev cell
plt.xlim([0.0, .1])
p1 = 'Optimal threshold: ' + str(profit_threshold[1])
plt.legend([p1])
plt.show()
```



In [63]:

```
X_ho = df2.drop('enrolled', axis=1)
y_ho = df2['enrolled']
```

In [73]:

```
# Theoretical threshold
threshold = 0.0058
y_pred = (lg.predict_proba(X_ho)[:, 1] > threshold).astype('float')
```

```
cf_matrix = confusion_matrix(y_ho, y_pred)
print(cf_matrix)
```

```
[[ 1789 57815]
 [   13  2989]]
```

In [74]:

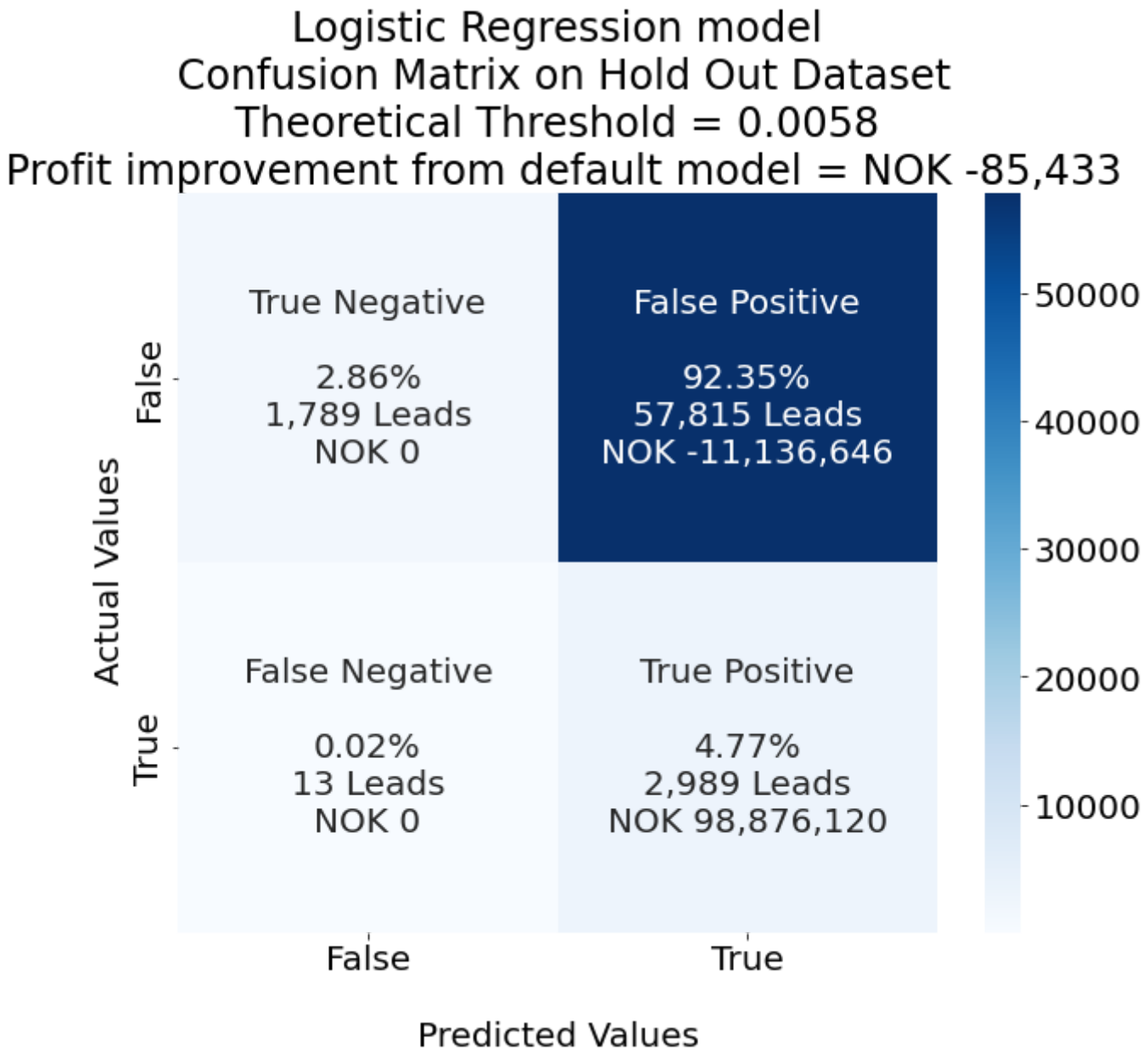
```
FP_Cost = round(cf_matrix[0,1] * Cost_p_lead, 2)
TP_Rev  = round(cf_matrix[1,1] * Revenue, 2)
TN_Rev  = round(cf_matrix[0,0] * 0,2)
FN_Cost = round(cf_matrix[1,0] * 0,2)
profit_matrix = [TN_Rev,FP_Cost, FN_Cost,TP_Rev]

print ("Total costs   = " , Total)
print ("Cost per lead = ", "{:.2f}".format(Cost_p_lead))
print("")
print("The profit_matrix contains: ")
print(profit_matrix)
print("")
print("FP cost = NOK {:.0f}".format(FP_Cost))
print("TP revenue NOK {:.0f}".format(TP_Rev))
print("TN revenue NOK {:.0f}".format(TN_Rev))
print("FN cost NOK {:.0f}" .format(FN_Cost))

Default = 87824907
Profit = FP_Cost+TP_Rev+TN_Rev+FN_Cost -Default
print("The default profit is NOK {:,.0f}".format(Default))
print("The profit over default is NOK {:,.0f}".format(Profit))
```

```
Total costs   =  2504132
Cost per lead =  -192.63

The profit_matrix contains:
[0, -11136645.51, 0, 98876120]

FP cost = NOK -11136646
TP revenue NOK 98876120
TN revenue NOK 0
FN cost NOK 0
The default profit is NOK 87,824,907
The profit over default is NOK -85,433
```

In [75]:

```
group_names = ['True Negative','False Positive','False Negative','True Positive']
```

```
group_percentages = ["{0:.2%}".format(value) for value in
                     cf_matrix.flatten()/np.sum(cf_matrix)]

group_counts = ["{0:,.0f} Leads".format(value) for value in
                cf_matrix.flatten()]

profit_each = ["NOK {0:,.0f}".format(value) for value in
                profit_matrix]

labels = [f"{v1}\n\n{v2}\n{v3}\n{v4}" for v1, v2, v3, v4 in
          zip(group_names,group_percentages,group_counts,profit_each)]

labels = np.asarray(labels).reshape(2,2)

fig, ax = plt.subplots(figsize=(10, 8))
plt.rcParams['font.size'] = '20'

ax = sns.heatmap(cf_matrix, annot=labels, fmt='', cmap='Blues')

#plt.title("Histograms for {0:.2f}".format(df.columns[i]))

ax.set_title('Logistic Regression model\n Confusion Matrix on Hold Out Dataset\n Theoretical Threshold = 0.0058 \n Profit improvem
ax.set_xlabel('\nPredicted Values', fontsize=20)
ax.set_ylabel('Actual Values ', fontsize=20);

## Ticket labels - List must be in alphabetical order
ax.xaxis.set_ticklabels(['False','True'], fontsize=20)
ax.yaxis.set_ticklabels(['False','True'], fontsize=20)

## Display the visualization of the Confusion Matrix.
plt.show()
```



In [76]:

```
# Optimal Threshold
threshold = 0.002
y_pred = (lg.predict_proba(X_ho)[:, 1] > threshold).astype('float')
```

```
cf_matrix = confusion_matrix(y_ho, y_pred)
print(cf_matrix)
```

```
[[   53 59551]
 [    1  3001]]
```

In [77]:

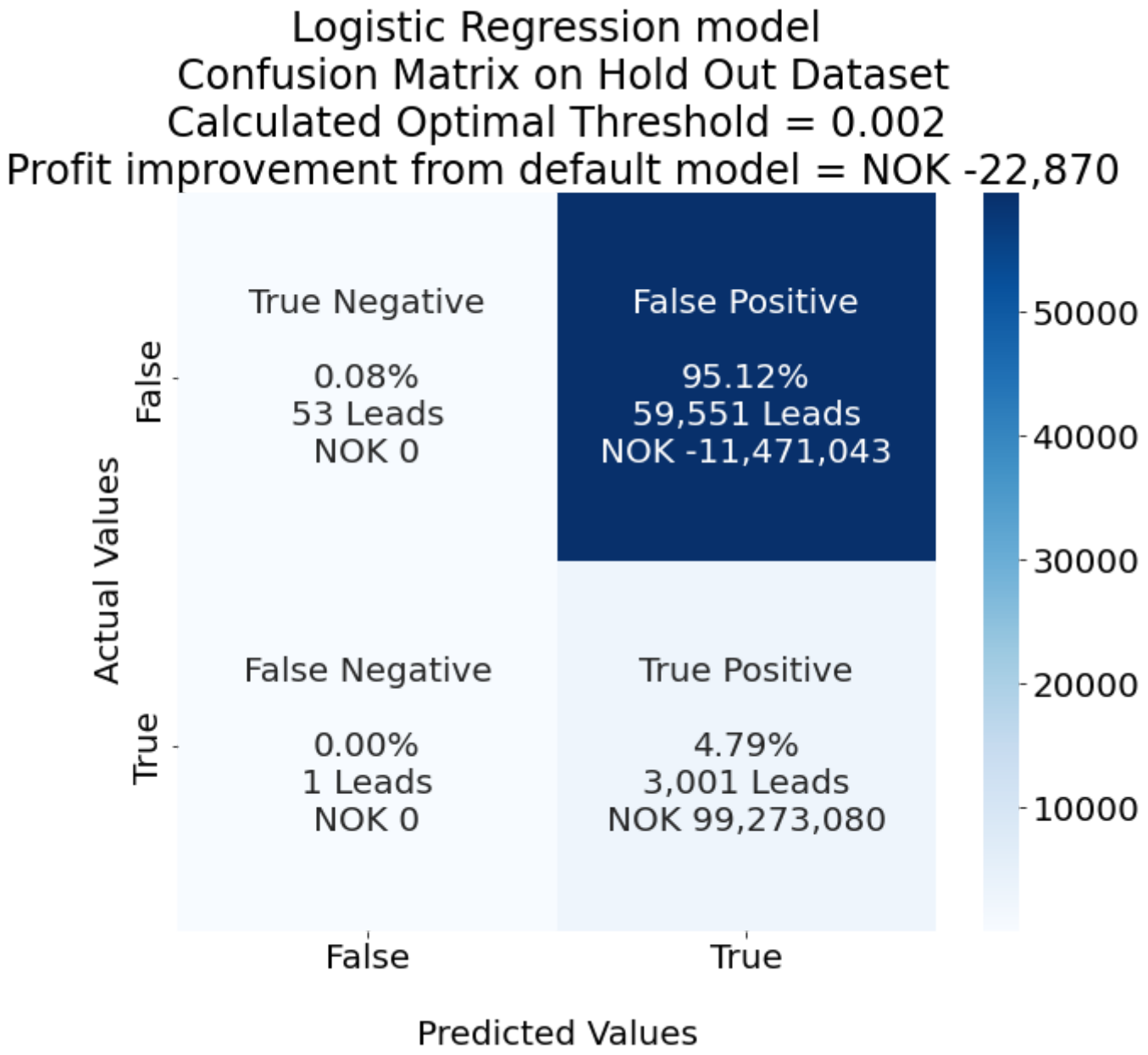
```
FP_Cost = round(cf_matrix[0,1] * Cost_p_lead, 2)
TP_Rev  = round(cf_matrix[1,1] * Revenue, 2)
TN_Rev  = round(cf_matrix[0,0] * 0,2)
FN_Cost = round(cf_matrix[1,0] * 0,2)
profit_matrix = [TN_Rev,FP_Cost, FN_Cost,TP_Rev]

print ("Total costs   = " , Total)
print ("Cost per lead = ", "{:.2f}".format(Cost_p_lead))
print("")
print("The profit_matrix contains: ")
print(profit_matrix)
print("")
print("FP cost = NOK {:.0f}".format(FP_Cost))
print("TP revenue NOK {:.0f}".format(TP_Rev))
print("TN revenue NOK {:.0f}".format(TN_Rev))
print("FN cost NOK {:.0f}" .format(FN_Cost))

Default = 87824907
Profit = FP_Cost+TP_Rev+TN_Rev+FN_Cost - Default
print("The total profit is NOK {:,.0f}".format(Profit))
```

```
Total costs   =  2504132
Cost per lead =  -192.63

The profit_matrix contains:
[0, -11471043.44, 0, 99273080]

FP cost = NOK -11471043
TP revenue NOK 99273080
TN revenue NOK 0
FN cost NOK 0
The total profit is NOK -22,870
```

In [78]:

```
group_names = ['True Negative','False Positive','False Negative','True Positive']

group_percentages = ["{0:.2%}".format(value) for value in
```

```
                     cf_matrix.flatten()/np.sum(cf_matrix)]

group_counts = ["{0:,.0f} Leads".format(value) for value in
                cf_matrix.flatten()]

profit_each = ["NOK {0:,.0f}".format(value) for value in
                profit_matrix]

labels = [f"{v1}\n\n{v2}\n{v3}\n{v4}" for v1, v2, v3, v4 in
          zip(group_names,group_percentages,group_counts,profit_each)]

labels = np.asarray(labels).reshape(2,2)

fig, ax = plt.subplots(figsize=(10, 8))
plt.rcParams['font.size'] = '20'

ax = sns.heatmap(cf_matrix, annot=labels, fmt='', cmap='Blues')

#plt.title("Histograms for {0:.2f}".format(df.columns[i]))

ax.set_title('Logistic Regression model\n Confusion Matrix on Hold Out Dataset\n Calculated Optimal Threshold = 0.002 \n Profit im
ax.set_xlabel('\nPredicted Values', fontsize=20)
ax.set_ylabel('Actual Values ', fontsize=20);

## Ticket Labels - List must be in alphabetical order
ax.xaxis.set_ticklabels(['False','True'], fontsize=20)
ax.yaxis.set_ticklabels(['False','True'], fontsize=20)

## Display the visualization of the Confusion Matrix.
plt.show()
```


Logistic Regression model
Confusion Matrix on Hold Out Dataset
Calculated Optimal Threshold = 0.002
Profit improvement from default model = NOK -22,870
True Negative
0.08%
53 Leads
NOK 0
False Positive
95.12%
59,551 Leads
NOK -11,471,043
False Negative
0.00%
1 Leads
NOK 0
True Positive
4.79%
3,001 Leads
NOK 99,273,080
Actual Values
False
True
False
True
Predicted Values
50000
40000
30000
20000
10000

In [1]:

```
import numpy as np
import pandas as pd
import seaborn as sns

from sklearn.calibration import calibration_curve
from sklearn.model_selection import cross_val_score,cross_val_predict, train_test_split
from sklearn.preprocessing import OneHotEncoder,StandardScaler,PowerTransformer,LabelEncoder
from sklearn.compose import make_column_transformer
from sklearn.pipeline import make_pipeline
from sklearn.metrics import accuracy_score,classification_report, recall_score,confusion_matrix, roc_auc_score, precision_score, f
from sklearn.calibration import CalibratedClassifierCV

import matplotlib.pyplot as plt

from scipy.sparse import csr_matrix

import optuna
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
from catboost import CatBoostClassifier


#importing plotly and cufflinks in offline mode
import cufflinks as cf
import plotly.offline
cf.go_offline()
cf.set_config_file(offline=False, world_readable=True)


import plotly
import plotly.express as px
import plotly.graph_objs as go
import plotly.offline as py
from plotly.offline import iplot
from plotly.subplots import make_subplots
import plotly.figure_factory as ff

import shap

import missingno as msno
```

```
import warnings
warnings.filterwarnings("ignore")
```

In [2]:

```
df = pd.read_csv(r'C:\Users\esp1_\OneDrive\Skrivebord\thesis/data_v1.9.csv', low_memory = False)

categorical = df.select_dtypes('object').columns
categorical = categorical.tolist()
df[categorical] = df[categorical].fillna('missing')

df = df.drop(['contact_id', 'Inscrito', 'Admitido', 'Useless', 'Contacto', 'Asistencia', 'Cita', 'no contactado', 'original_source
```

In [3]:

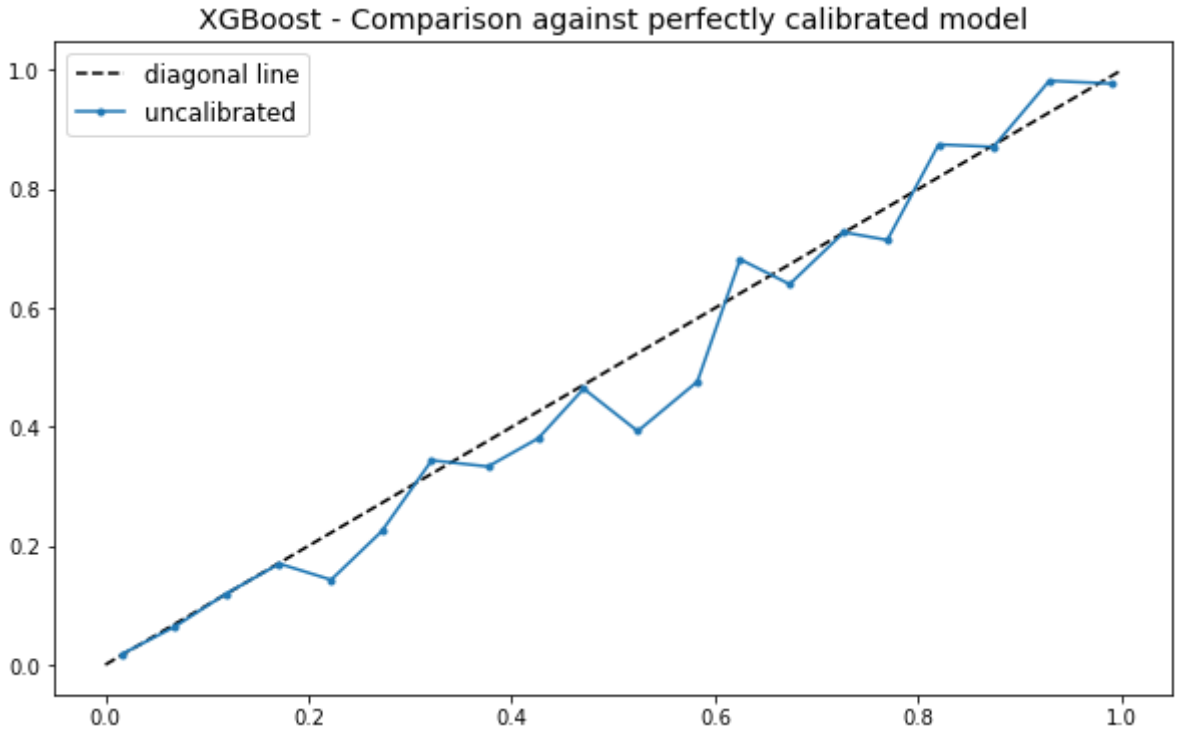
```
df_dummies = pd.get_dummies(df)
df1_dummies_1 = df_dummies.loc[df_dummies['Ciclo Generación_2021-3'] == 1]
df1_dummies_2 = df_dummies.loc[df_dummies['Ciclo Generación_2022-1'] == 1]
df1 = pd.concat([df1_dummies_1,df1_dummies_2])
df2_dummies_1 = df_dummies.loc[df_dummies['Ciclo Generación_2022-2'] == 1]
df2_dummies_2 = df_dummies.loc[df_dummies['Ciclo Generación_2022-3'] == 1]
df2 = pd.concat([df2_dummies_1,df2_dummies_2])

df1 = df1.drop(['Ciclo Generación_2021-3', 'Ciclo Generación_2022-1', 'Ciclo Generación_2022-2', 'Ciclo Generación_2022-3'], axis=
df2 = df2.drop(['Ciclo Generación_2021-3', 'Ciclo Generación_2022-1', 'Ciclo Generación_2022-2', 'Ciclo Generación_2022-3'], axis=
```

In [4]:

```
X = df1.drop('enrolled', axis=1)
y = df1['enrolled']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)


# predict uncalibrated probabilities
def uncalibrated(X_train, X_test, y_train):

    xgboost = XGBClassifier(random_state=0) # , scale_pos_weight=9
    xgboost.fit(X_train, y_train)
    return xgboost.predict_proba(X_test)[:, 1]
'''
# predict calibrated probabilities
def calibrated(X_train, X_test, y_train):

    xgboost = XGBClassifier(random_state=0) # , scale_pos_weight=9
    calibrated = CalibratedClassifierCV(xgboost, method='sigmoid', cv=5)
```

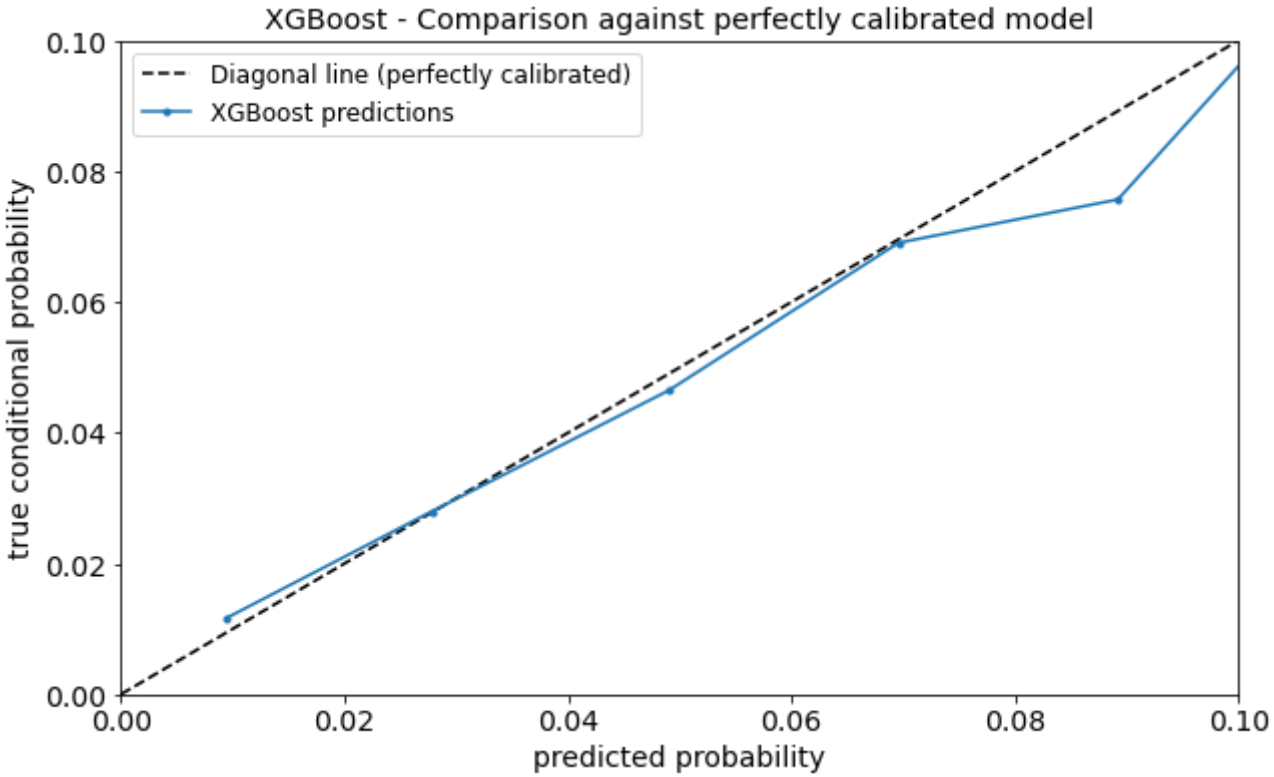
```
    calibrated.fit(X_train, y_train)
    return calibrated.predict_proba(X_test)[:, 1]
'''

# uncalibrated predictions
yhat_uncalibrated = uncalibrated(X_train, X_test, y_train)
# calibrated predictions
#yhat_calibrated = calibrated(X_train, X_test, y_train)
# reliability diagrams
fop_uncalibrated, mpv_uncalibrated = calibration_curve(y_test, yhat_uncalibrated, n_bins=20, normalize=True)
#fop_calibrated, mpv_calibrated = calibration_curve(y_test, yhat_calibrated, n_bins=20)
# plot perfectly calibrated
plt.subplots(1, figsize=(10,6))
plt.rcParams['font.size'] = '12'
plt.title('XGBoost - Comparison against perfectly calibrated model')
plt.plot([0, 1], [0, 1], linestyle='--', color='black')
# plot model reliabilities
plt.plot(mpv_uncalibrated, fop_uncalibrated, marker='.', )
#plt.plot(mpv_calibrated, fop_calibrated, marker='.')
plt.legend(['diagonal line','uncalibrated', 'calibrated'])
plt.show()
```

```
[14:16:40] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/learner.cc:1115: Starting in XGBoost 1.3.0, t
he default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_
metric if you'd like to restore the old behavior.
```



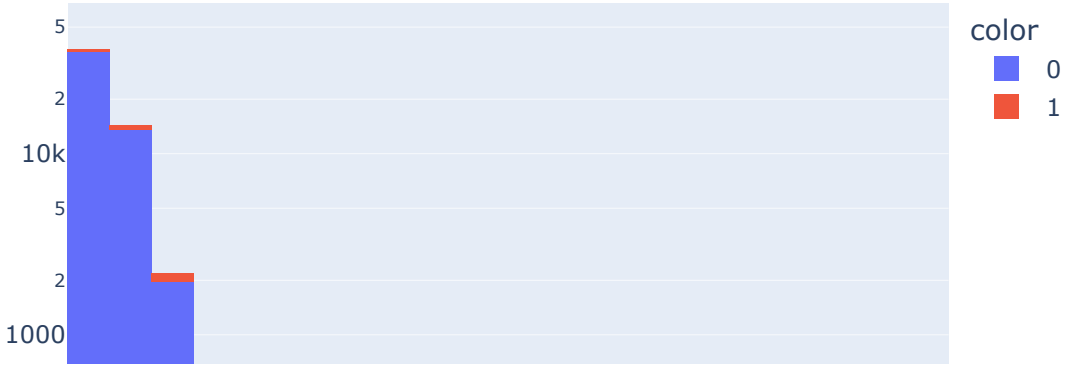
In [7]:

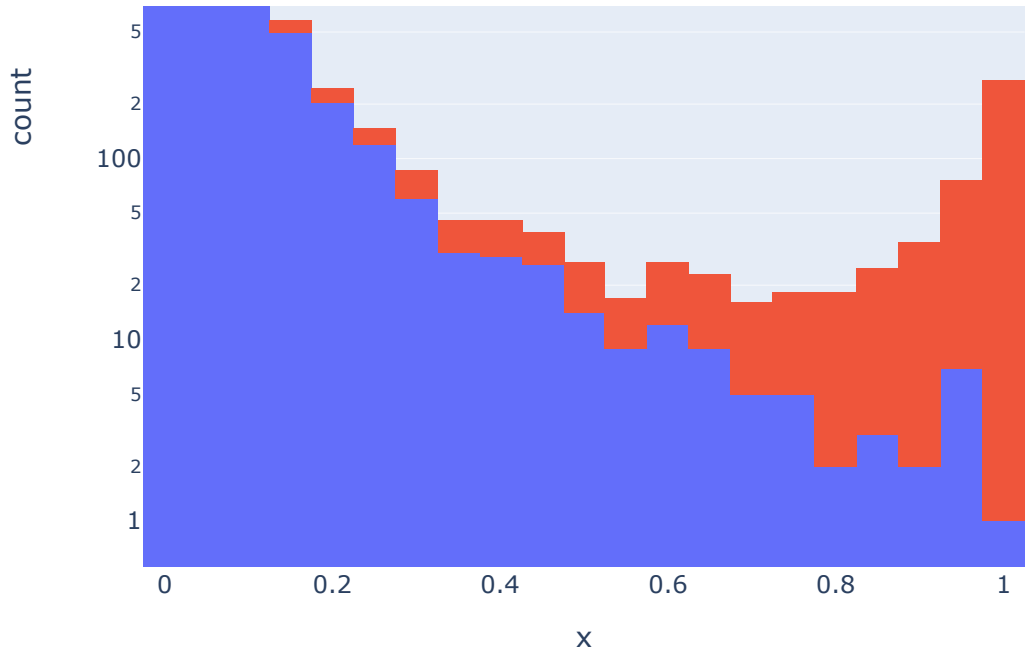
```
# reliability diagrams
fop_uncalibrated, mpv_uncalibrated = calibration_curve(y_test, yhat_uncalibrated, n_bins=50, normalize=True)
#fop_calibrated, mpv_calibrated = calibration_curve(y_test, yhat_calibrated, n_bins=50)
# plot perfectly calibrated
plt.subplots(1, figsize=(10,6))
plt.rcParams['font.size'] = '14'
plt.xlim([0, 0.1])
plt.ylim([0, 0.1])
plt.xlabel("predicted probability")
plt.ylabel("true conditional probability")
plt.rcParams['font.size'] = '12'
plt.title('XGBoost - Comparison against perfectly calibrated model')
plt.plot([0, 1], [0, 1], linestyle='--', color='black')
# plot model reliabilities
plt.plot(mpv_uncalibrated, fop_uncalibrated, marker='.')
#plt.plot(mpv_calibrated, fop_calibrated, marker='.')
plt.legend(['Diagonal line (perfectly calibrated)','XGBoost predictions'])
plt.show()
```



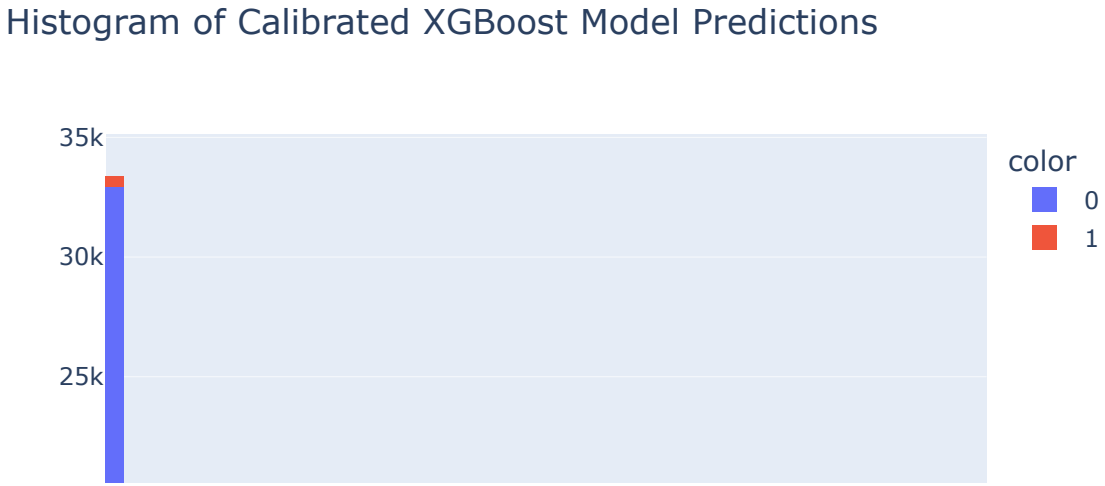
In [32]:

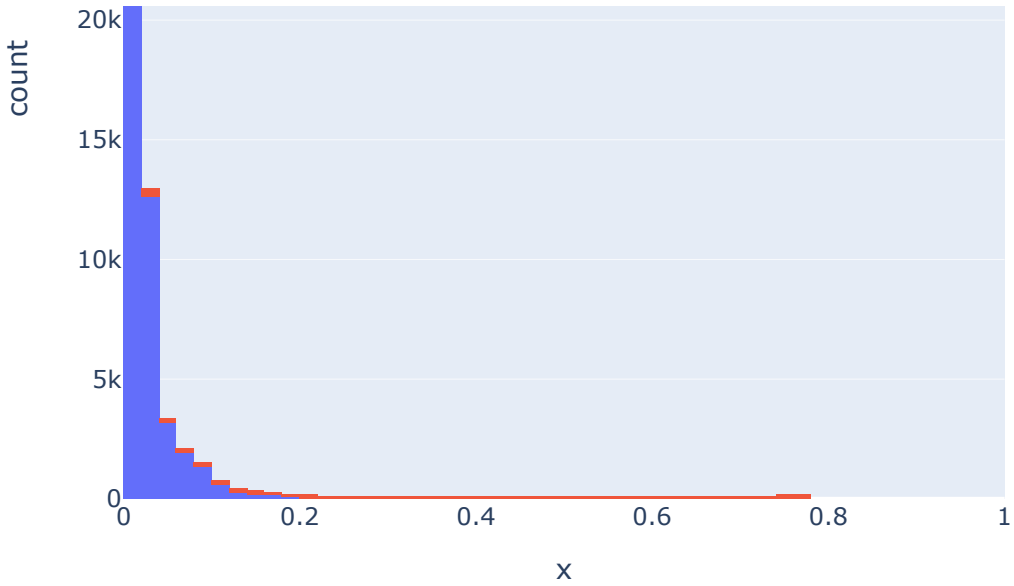
```
fig = px.histogram(df, x=yhat_uncalibrated, color=y_test,width=600, height=600, nbins=20, log_y = True)
fig.show()
```



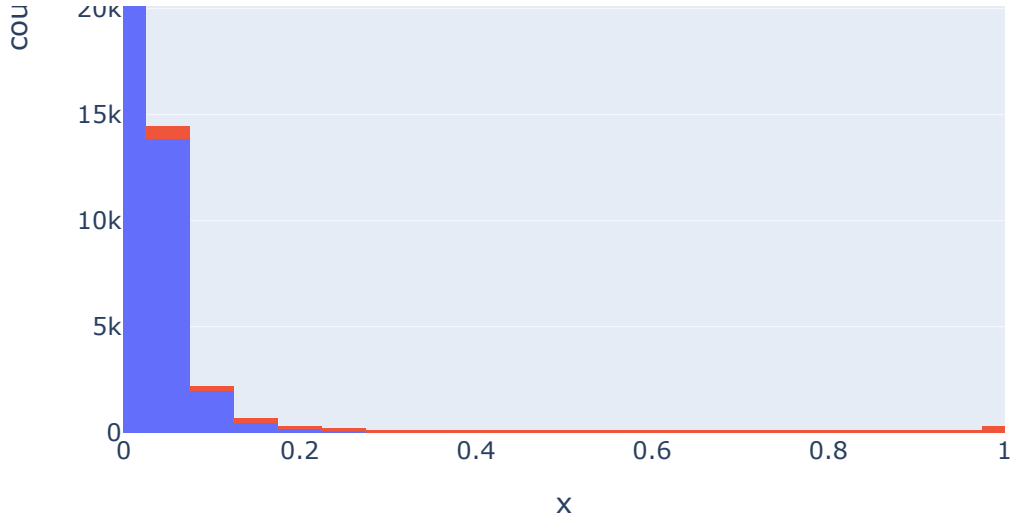


In [13]:
```
'''fig = px.histogram(df, x=yhat_calibrated, color=y_test,width=600, height=600, nbins=40, range_x=[0,1], title = 'Histogram of Ca
fig.show()
```





In [34]:
```
fig = px.histogram(df, x=yhat_uncalibrated, color=y_test,width=600, height=600, nbins=40, range_x=[0,1], title = 'Histogram of XGB
fig.show()
```



In [5]:

```
accuracy= []
recall =[]
roc_auc= []
precision = []
model_names =[]

xgboost = XGBClassifier(random_state=0, eval_metric='logloss') # , scale_pos_weight=9
# calibrated = CalibratedClassifierCV(xgboost, method='sigmoid', cv=5)
xgboost.fit(X_train, y_train)

y_pred1 = xgboost.predict(X_test)
y_pred = xgboost.predict_proba(X_test)

accuracy.append(round(accuracy_score(y_test, y_pred1),4))
recall.append(round(recall_score(y_test, y_pred1),4))
roc_auc.append(round(roc_auc_score(y_test, y_pred[:,1]),4))
precision.append(round(precision_score(y_test, y_pred1),4))

model_names = ['XGBoost']
result_XGBoost = pd.DataFrame({'Accuracy':accuracy,'Recall':recall, 'Roc_Auc':roc_auc, 'Precision':precision}, index=model_names)
result_XGBoost
```

Out[5]:

	Accuracy	Recall	Roc_Auc	Precision

	Accuracy	Recall	Roc_Auc	Precision
XGBoost	0.9718	0.239	0.8097	0.8782

In [37]:
```
Leads_costing = 13000

Talent    = 800000
Telco     = 44132
Software  = 35000
Marketing = 1550000
Hardware  = 75000
Total     = Talent + Telco + Software + Marketing + Hardware

Cost_p_lead = (Total/Leads_costing)*-1

Revenue   = 33080
```

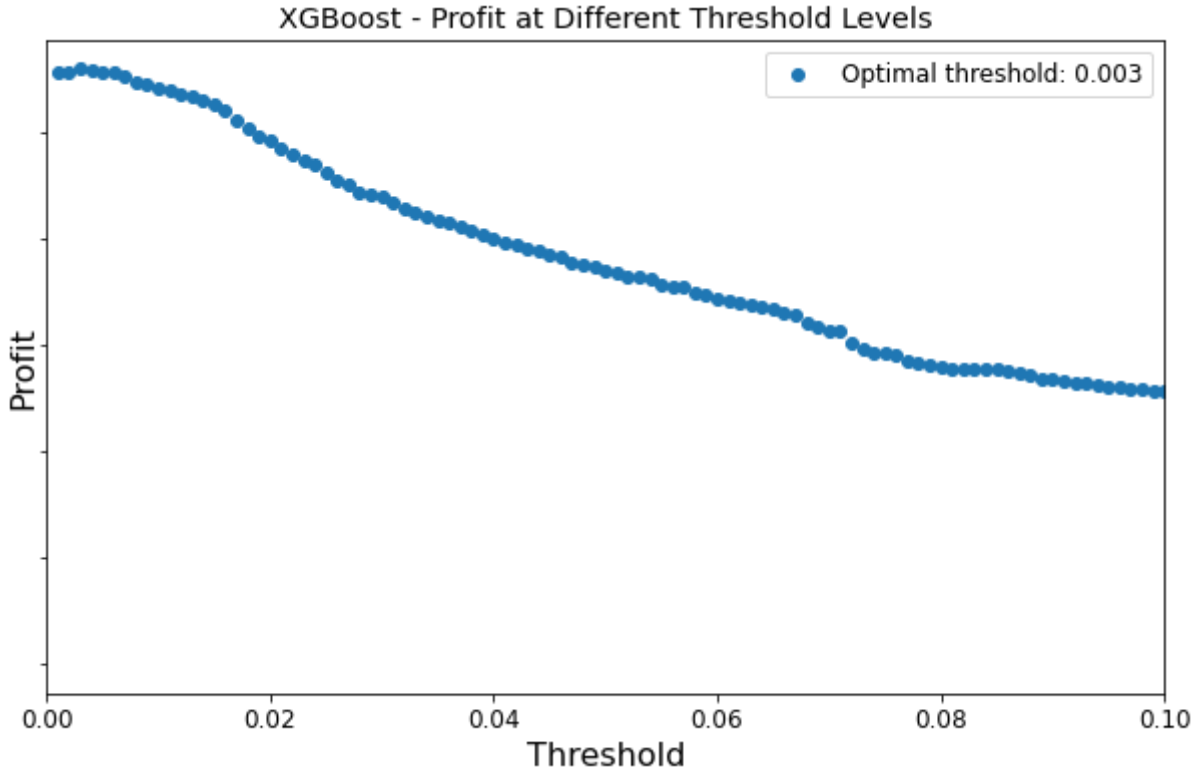
In [38]:
```
column_names = ['profit', 'threshold']
profit_df = pd.DataFrame(columns = column_names)

for i in range(1000):

    y_pred_opt_1 = (xgboost.predict_proba(X_test)[:, 1] > (i+1)/1000).astype('float')
    x = confusion_matrix(y_test, y_pred_opt_1)
    FP = x[0,1]*-192.63
    TP = x[1,1]*33080
    profit = TP+FP
    profit_df.loc[i, 'profit'] = profit
    profit_df.loc[i, 'threshold'] = (i+1)/1000
```

In [39]:
```
profit_df['profit'] = pd.to_numeric(profit_df['profit'])
profit_df['threshold'] = pd.to_numeric(profit_df['threshold'])
print(profit_df[['profit']].idxmax())
print(profit_df[['profit']].max())
```

```
profit    2
dtype: int64
profit    55910453.12
dtype: float64
```

In [40]:

```
plt.subplots(1, figsize=(10,6))
plt.rcParams['font.size'] = '12'
plt.title('XGBoost - Profit at Different Threshold Levels')
plt.scatter(profit_df['threshold'], profit_df['profit'])
plt.xlabel("Threshold", fontsize=16)
plt.tick_params(labelleft=False)
plt.ylabel("Profit", fontsize=16)
profit_threshold = profit_df.loc[2,:] # set threshold location found in prev cell
plt.xlim([0.0, .1])
p1 = 'Optimal threshold: ' + str(profit_threshold[1])
plt.legend([p1])
plt.show()
```



In [41]:

```
X_ho = df2.drop('enrolled', axis=1)
y_ho = df2['enrolled']
```

In [42]:
```
# Theoretical threshold
threshold = 0.0058
y_pred = (xgboost.predict_proba(X_ho)[:, 1] > threshold).astype('float')
cf_matrix = confusion_matrix(y_ho, y_pred)
print(cf_matrix)
```

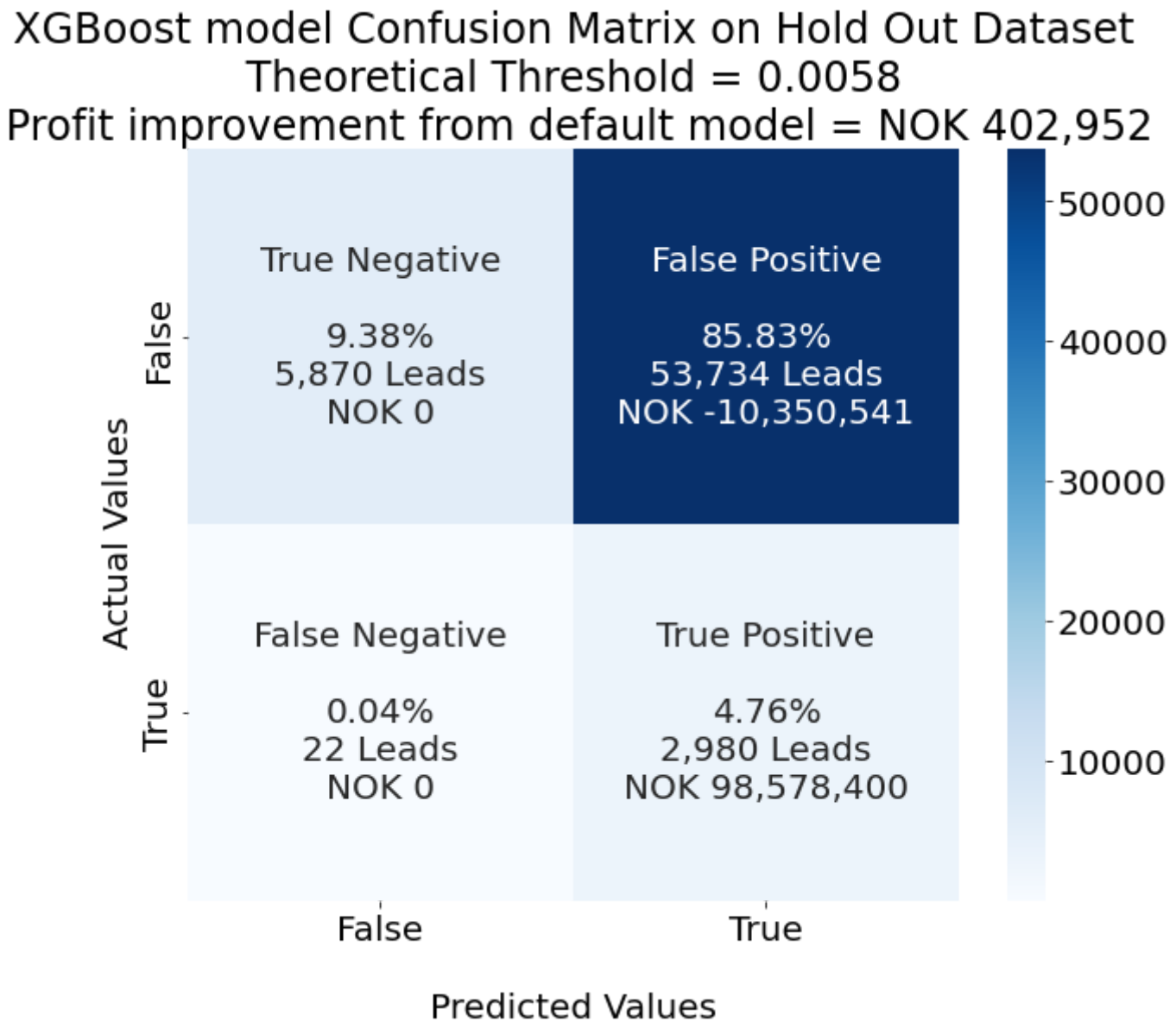
```
[[ 5870 53734]
 [   22  2980]]
```

In [43]:
```
FP_Cost = round(cf_matrix[0,1] * Cost_p_lead, 2)
TP_Rev  = round(cf_matrix[1,1] * Revenue, 2)
TN_Rev  = round(cf_matrix[0,0] * 0,2)
FN_Cost = round(cf_matrix[1,0] * 0,2)
profit_matrix = [TN_Rev,FP_Cost, FN_Cost,TP_Rev]

print ("Total costs   = " , Total)
print ("Cost per lead = ", "{:.2f}".format(Cost_p_lead))
print("")
print("The profit_matrix contains: ")
print(profit_matrix)
print("")
print("FP cost = NOK {:.0f}".format(FP_Cost))
print("TP revenue NOK {:.0f}".format(TP_Rev))
print("TN revenue NOK {:.0f}".format(TN_Rev))
print("FN cost NOK {:.0f}" .format(FN_Cost))

Default = 87824907
Profit = FP_Cost+TP_Rev+TN_Rev+FN_Cost-Default
print("The total profit is NOK {:,.0f}".format(Profit))
```

```
Total costs   =  2504132
Cost per lead =  -192.63

The profit_matrix contains:
[0, -10350540.68, 0, 98578400]

FP cost = NOK -10350541
TP revenue NOK 98578400
TN revenue NOK 0
FN cost NOK 0
The total profit is NOK 402,952
```

In [44]:

```
group_names = ['True Negative','False Positive','False Negative','True Positive']

group_percentages = ["{0:.2%}".format(value) for value in
                     cf_matrix.flatten()/np.sum(cf_matrix)]

group_counts = ["{0:,.0f} Leads".format(value) for value in
                cf_matrix.flatten()]

profit_each = ["NOK {0:,.0f}".format(value) for value in
                profit_matrix]

labels = [f"{v1}\n\n{v2}\n{v3}\n{v4}" for v1, v2, v3, v4 in
          zip(group_names,group_percentages,group_counts,profit_each)]

labels = np.asarray(labels).reshape(2,2)

fig, ax = plt.subplots(figsize=(10, 8))
plt.rcParams['font.size'] = '20'

ax = sns.heatmap(cf_matrix, annot=labels, fmt='', cmap='Blues')

#plt.title("Histograms for {0:.2f}".format(df.columns[i]))

ax.set_title('XGBoost model Confusion Matrix on Hold Out Dataset\n Theoretical Threshold = 0.0058 \n Profit improvement from defau
ax.set_xlabel('\nPredicted Values', fontsize=20)
ax.set_ylabel('Actual Values ', fontsize=20);

## Ticket labels - List must be in alphabetical order
ax.xaxis.set_ticklabels(['False','True'], fontsize=20)
ax.yaxis.set_ticklabels(['False','True'], fontsize=20)

## Display the visualization of the Confusion Matrix.
plt.show()
```



In [45]:

```
# Optimal Threshold
threshold = 0.003
y_pred = (xgboost.predict_proba(X_ho)[:, 1] > threshold).astype('float')
cf_matrix = confusion_matrix(y_ho, y_pred)
print(cf_matrix)
```

```
[[ 2028 57576]
 [    9  2993]]
```

In [46]:

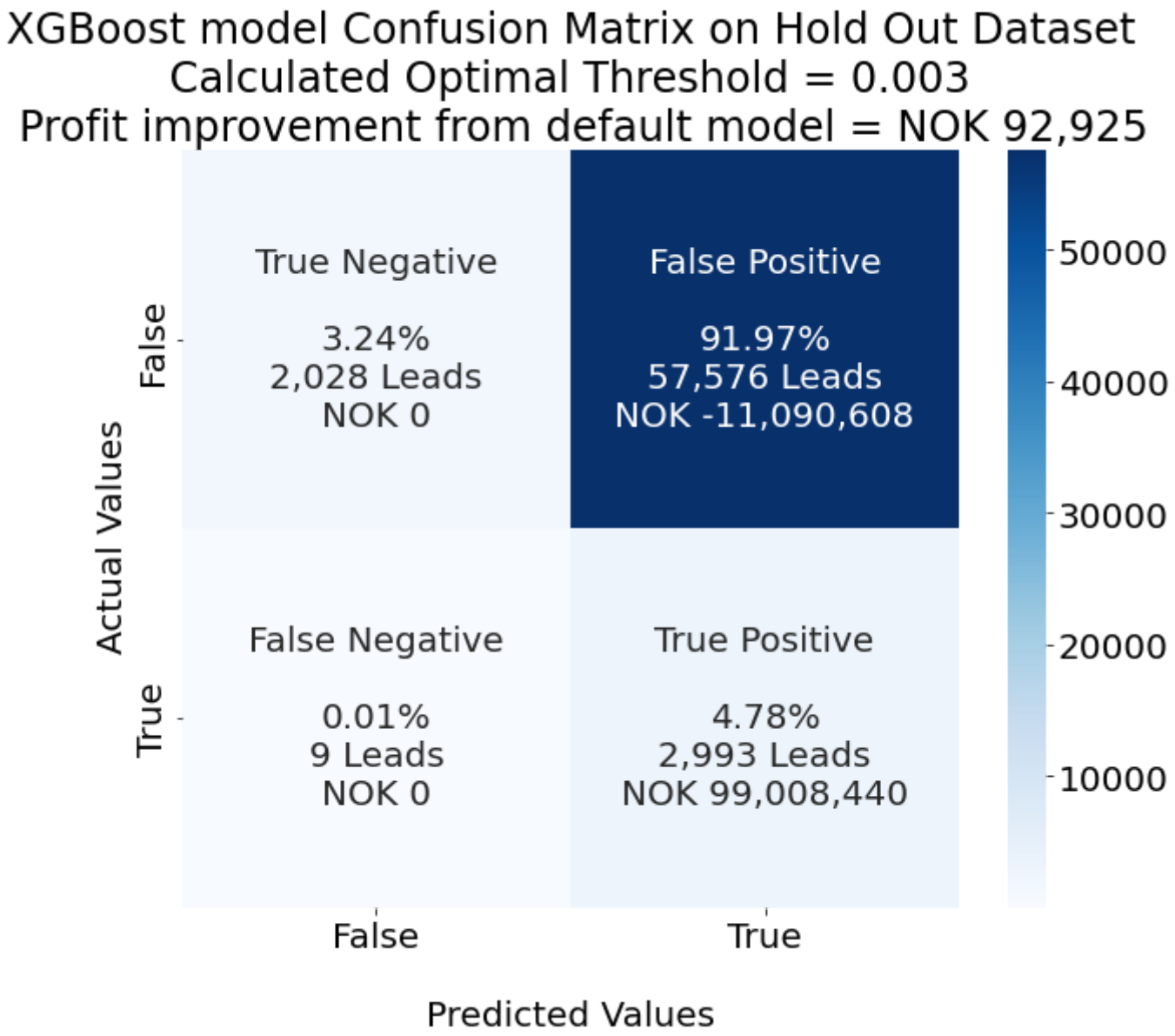
```
FP_Cost = round(cf_matrix[0,1] * Cost_p_lead, 2)
TP_Rev  = round(cf_matrix[1,1] * Revenue, 2)
TN_Rev  = round(cf_matrix[0,0] * 0,2)
FN_Cost = round(cf_matrix[1,0] * 0,2)
profit_matrix = [TN_Rev,FP_Cost, FN_Cost,TP_Rev]

print ("Total costs   = " , Total)
print ("Cost per lead = ", "{:.2f}".format(Cost_p_lead))
print("")
print("The profit_matrix contains: ")
print(profit_matrix)
print("")
print("FP cost = NOK {:.0f}".format(FP_Cost))
print("TP revenue NOK {:.0f}".format(TP_Rev))
print("TN revenue NOK {:.0f}".format(TN_Rev))
print("FN cost NOK {:.0f}" .format(FN_Cost))

Default = 87824907
Profit = FP_Cost+TP_Rev+TN_Rev+FN_Cost-Default
print("The total profit is NOK {:,.0f}".format(Profit))
```

```
Total costs   =  2504132
Cost per lead =  -192.63

The profit_matrix contains:
[0, -11090608.0, 0, 99008440]

FP cost = NOK -11090608
TP revenue NOK 99008440
TN revenue NOK 0
FN cost NOK 0
The total profit is NOK 92,925
```

In [47]:

```
group_names = ['True Negative','False Positive','False Negative','True Positive']

group_percentages = ["{0:.2%}".format(value) for value in
                     cf_matrix.flatten()/np.sum(cf_matrix)]

group_counts = ["{0:,.0f} Leads".format(value) for value in
```

```
                cf_matrix.flatten()]

profit_each = ["NOK {0:,.0f}".format(value) for value in
                profit_matrix]

labels = [f"{v1}\n\n{v2}\n{v3}\n{v4}" for v1, v2, v3, v4 in
          zip(group_names,group_percentages,group_counts,profit_each)]

labels = np.asarray(labels).reshape(2,2)

fig, ax = plt.subplots(figsize=(10, 8))
plt.rcParams['font.size'] = '20'

ax = sns.heatmap(cf_matrix, annot=labels, fmt='', cmap='Blues')

#plt.title("Histograms for {0:.2f}".format(df.columns[i]))

ax.set_title('XGBoost model Confusion Matrix on Hold Out Dataset\n Calculated Optimal Threshold = 0.003 \n  Profit improvement fro
ax.set_xlabel('\nPredicted Values', fontsize=20)
ax.set_ylabel('Actual Values ', fontsize=20);

## Ticket labels - List must be in alphabetical order
ax.xaxis.set_ticklabels(['False','True'], fontsize=20)
ax.yaxis.set_ticklabels(['False','True'], fontsize=20)

## Display the visualization of the Confusion Matrix.
plt.show()
```


XGBoost model Confusion Matrix on Hold Out Dataset
Calculated Optimal Threshold = 0.003
Profit improvement from default model = NOK 92,925
True Negative
3.24%
2,028 Leads
NOK 0
False Positive
91.97%
57,576 Leads
NOK -11,090,608
False Negative
0.01%
9 Leads
NOK 0
True Positive
4.78%
2,993 Leads
NOK 99,008,440
False
True
Actual Values
False
True
Predicted Values
50000
40000
30000
20000
10000

In [ ]:

In [ ]: