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]:

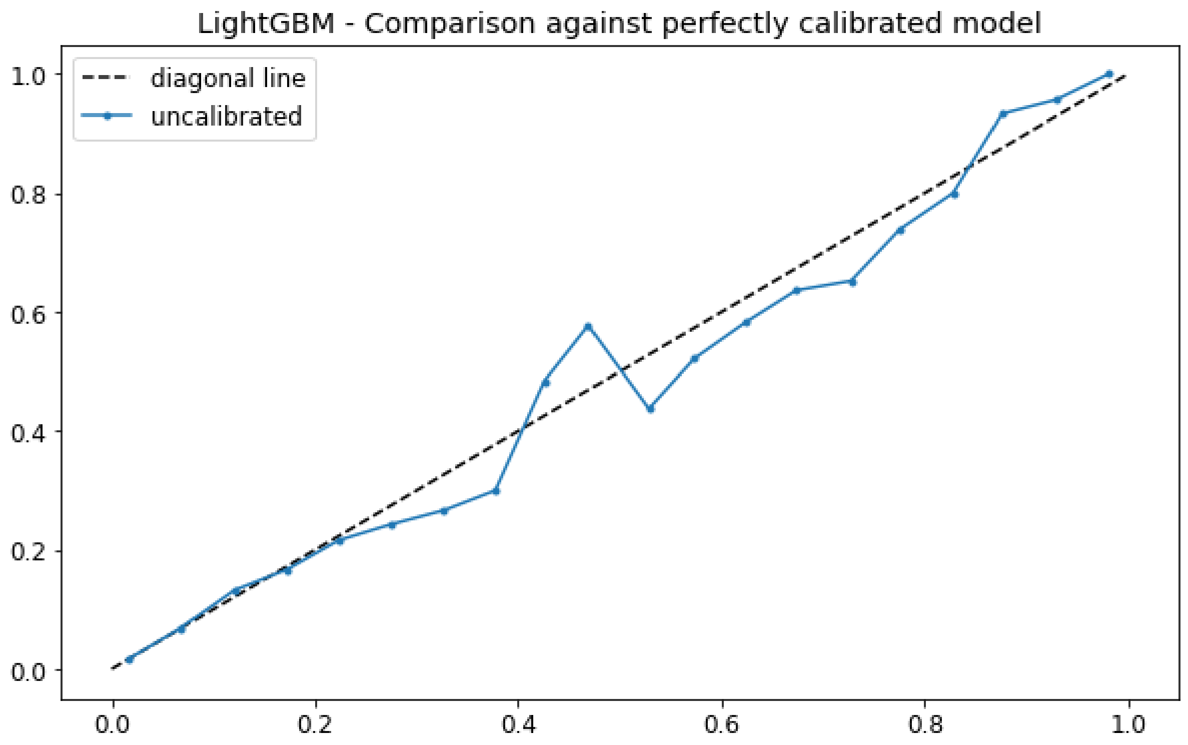
```
df = pd.read_csv(r'C:\Users\esp1_\OneDrive\Skrivebord\thesis/data_v1.9.csv', low_memory = False)

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 [13]:

```
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=
```

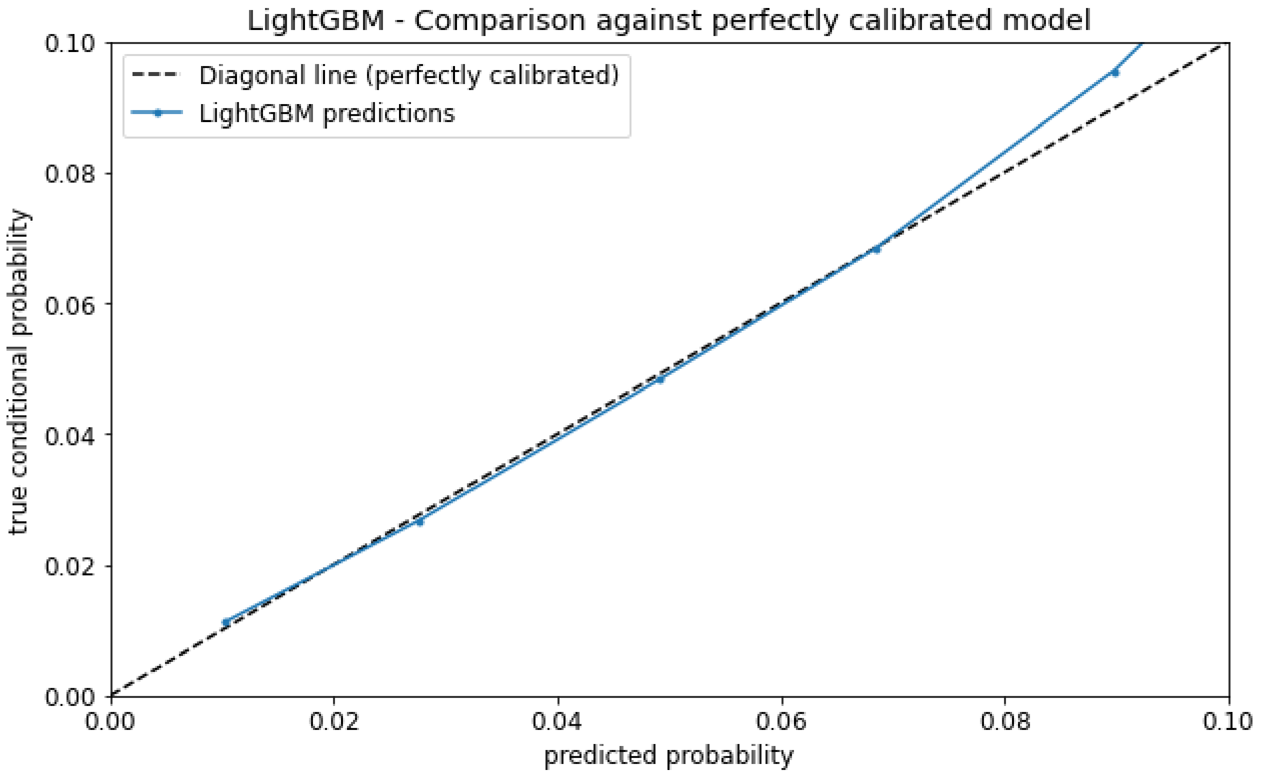
In [14]:

```
X1 = df1.drop('enrolled', axis=1)
X = csr_matrix(X1.values)
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):

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

    lightgbm = LGBMClassifier(random_state=0) # , scale_pos_weight=9
    calibrated = CalibratedClassifierCV(lightgbm, 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('LightGBM - 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()
```

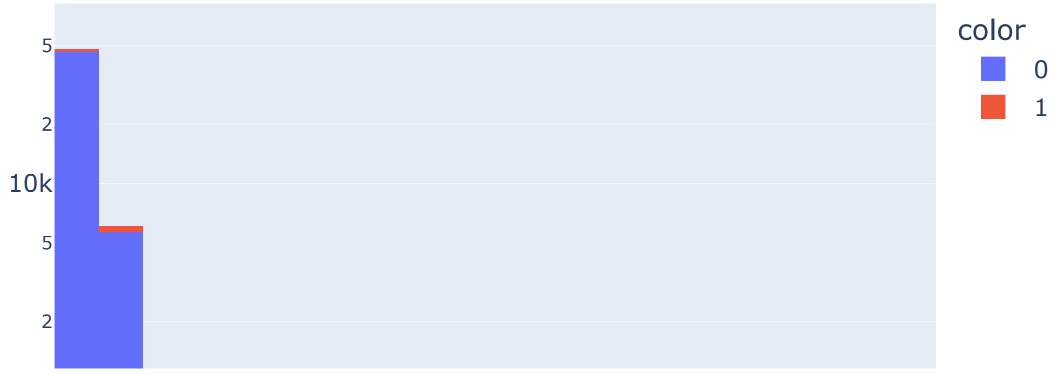


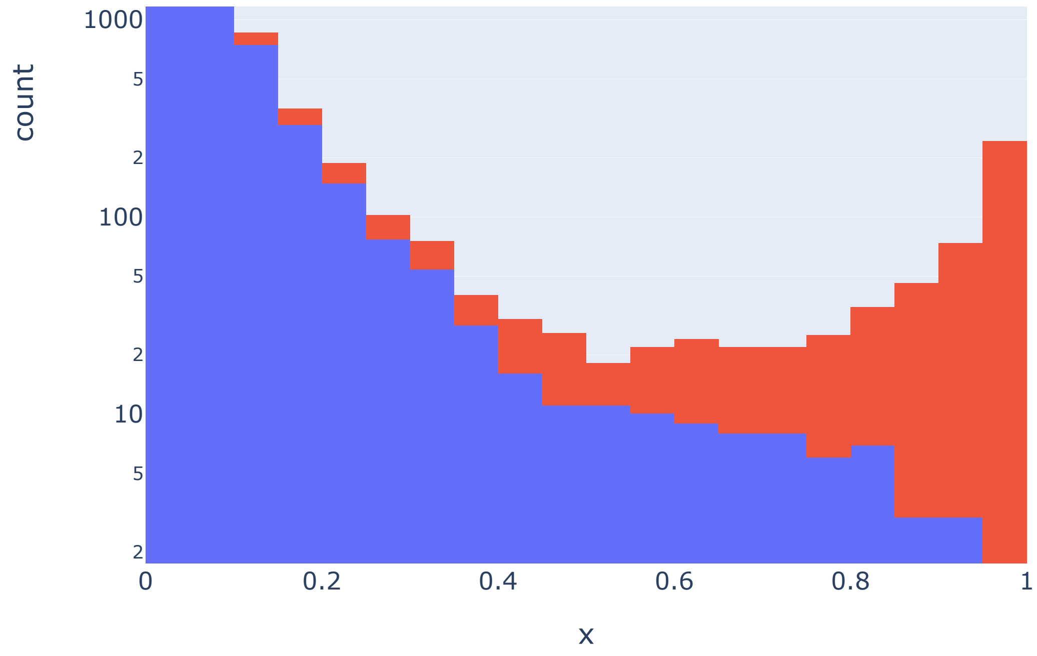
In [10]:

```
# 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('LightGBM - 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)','LightGBM predictions'])
plt.show()
```



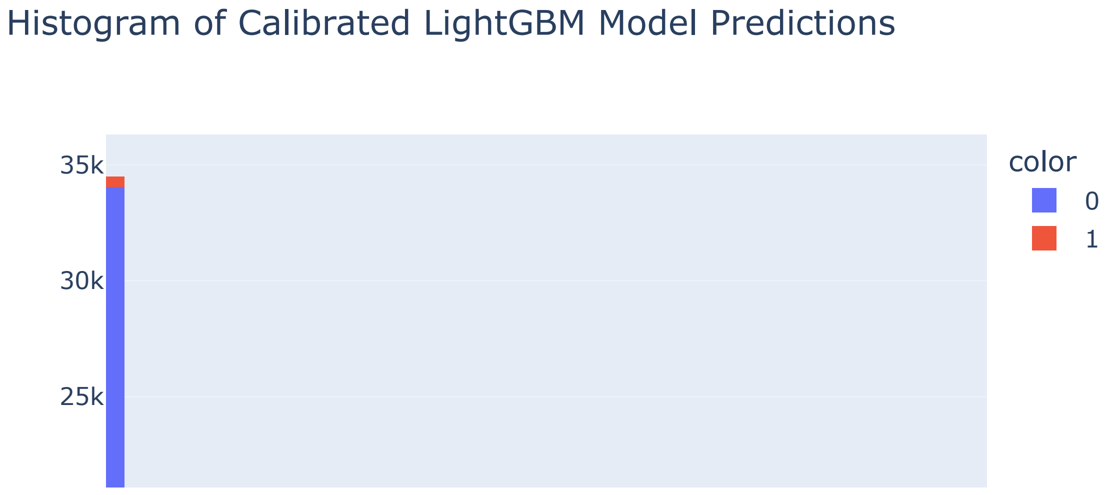
In [60]:
```
fig = px.histogram(df, x=yhat_uncalibrated, color=y_test,width=600, height=600, nbins=20, log_y = True)
fig.show()
```

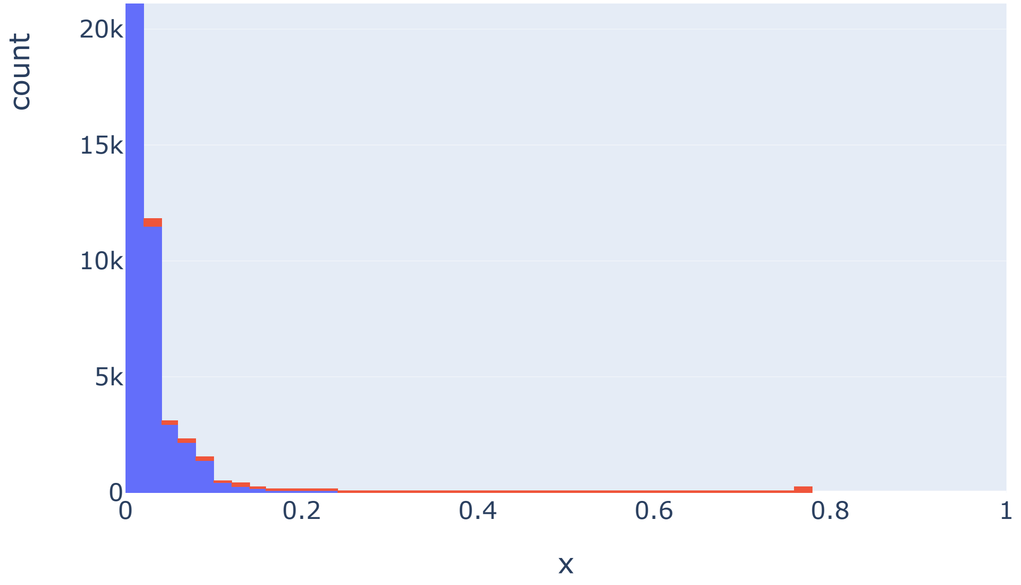




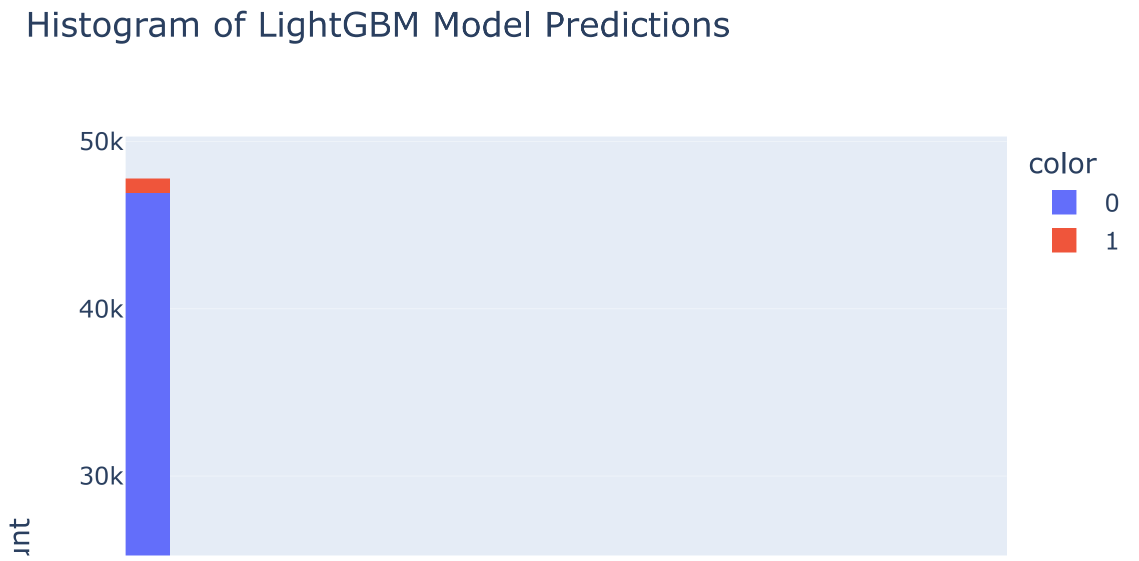
In [10]:

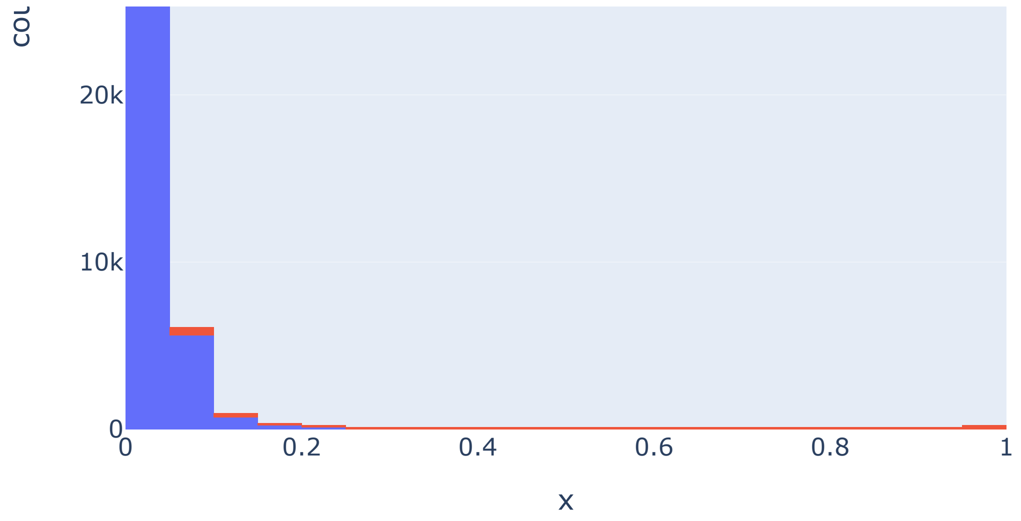
```
'''fig = px.histogram(df, x=yhat_calibrated, color=y_test,width=600, height=600, nbins=40, range_x=[0,1], title = 'Histogram of Ca
fig.show()'''
```





In [62]:
```
fig = px.histogram(df, x=yhat_uncalibrated, color=y_test,width=600, height=600, nbins=40, range_x=[0,1], title = 'Histogram of Lig
fig.show()
```





In [5]:
```
accuracy= []
recall =[]
roc_auc= []
precision = []
model_names =[]

lightgbm = LGBMClassifier(random_state=0) # , scale_pos_weight=9
# calibrated = CalibratedClassifierCV(lightgbm, method='sigmoid', cv=5)
lightgbm.fit(X_train, y_train)

y_pred1 = lightgbm.predict(X_test)
y_pred = lightgbm.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 = ['LightGBM']
result_lgbm = pd.DataFrame({'Accuracy':accuracy,'Recall':recall, 'Roc_Auc':roc_auc, 'Precision':precision}, index=model_names)
result_lgbm
```

Out[5]:

| | Accuracy | Recall | Roc_Auc | Precision |
|---|---|---|---|---|

| | Accuracy | Recall | Roc_Auc | Precision |
|---|---|---|---|---|
| **LightGBM** | 0.9716 | 0.2324 | 0.8128 | 0.8769 |

In [65]:

```
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 [66]:

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

for i in range(1000):

    y_pred_opt_1 = (lightgbm.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
```

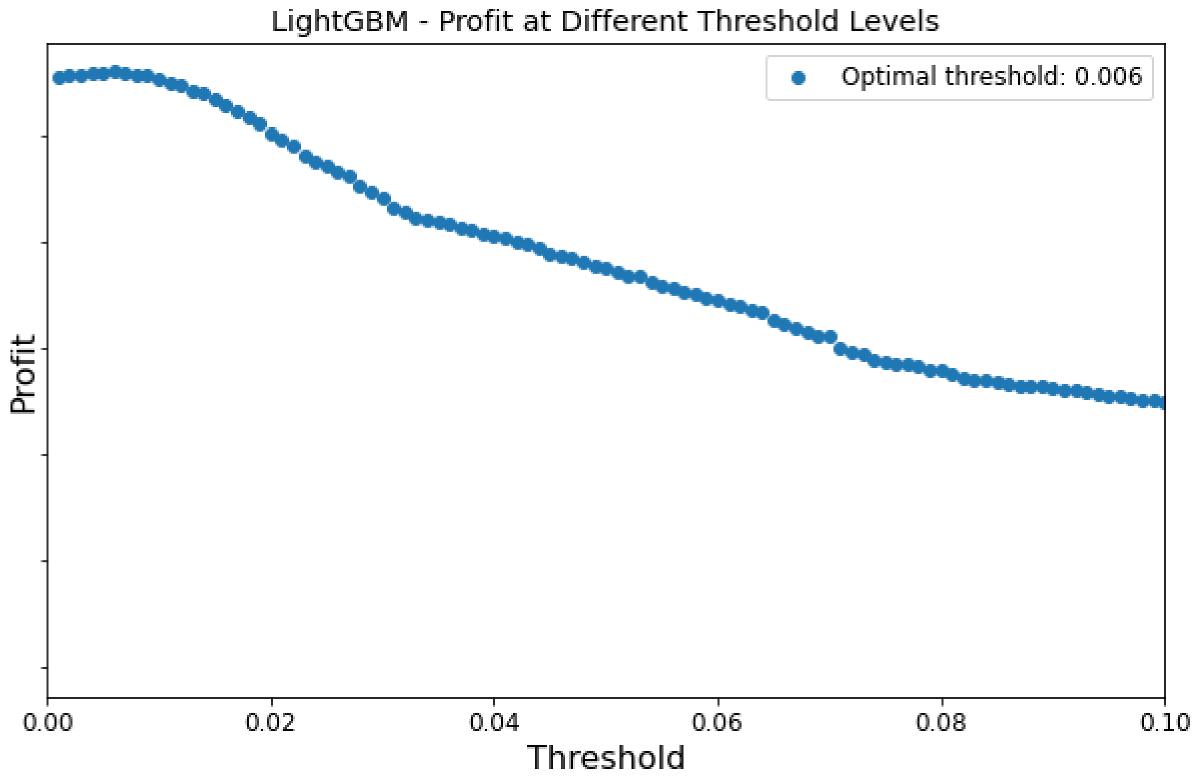
In [67]:

```
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    5
dtype: int64
profit    55886897.97
dtype: float64
```

In [68]:

```
plt.subplots(1, figsize=(10,6))
plt.rcParams['font.size'] = '12'
plt.title('LightGBM - 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[5,:] # 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 [6]:

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

In [70]:
```
# Theoretical threshold
threshold = 0.0058
y_pred = (lightgbm.predict_proba(X_ho)[:, 1] > threshold).astype('float')
cf_matrix = confusion_matrix(y_ho, y_pred)
print(cf_matrix)
```

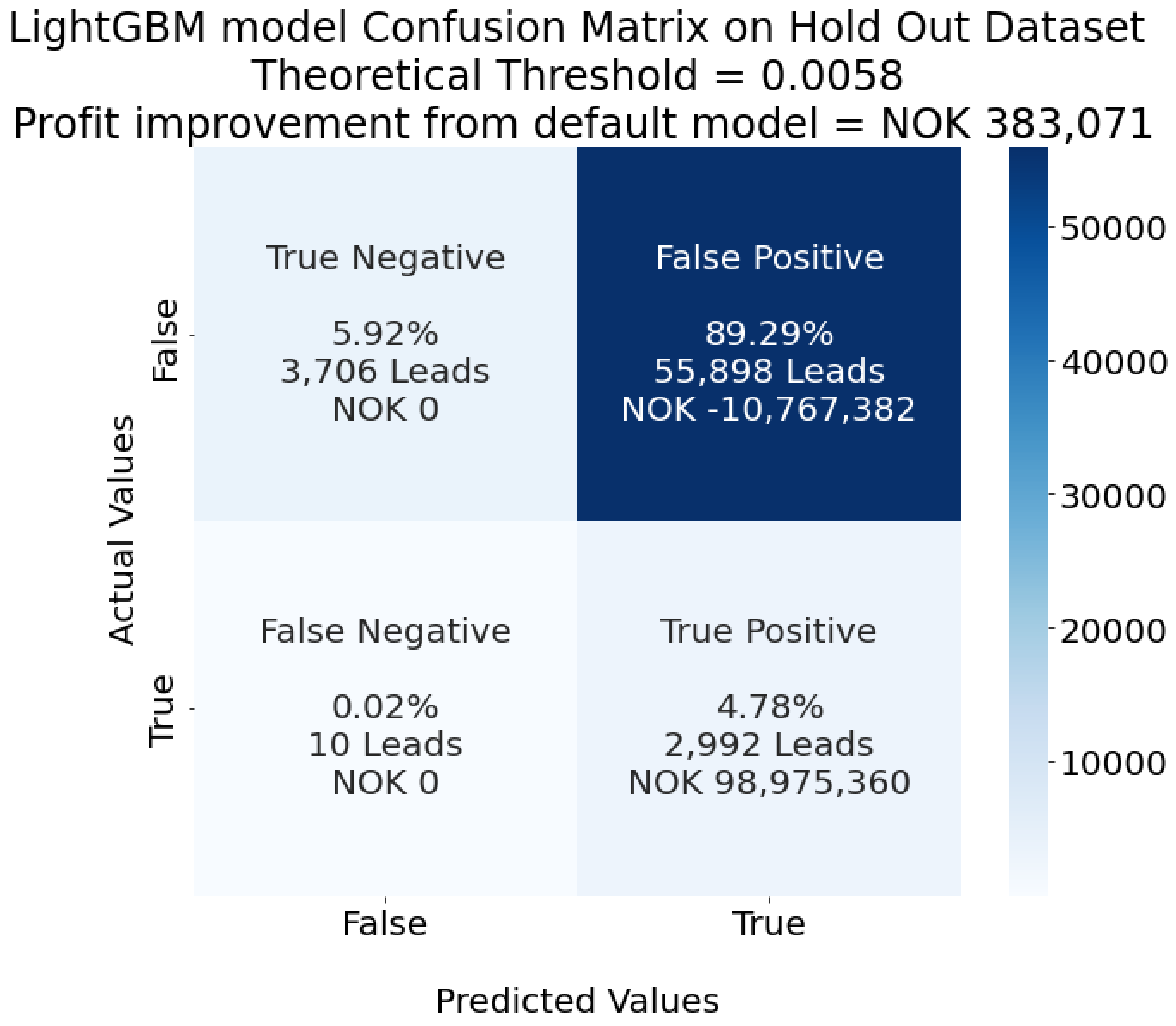
```
[[ 3706 55898]
 [   10  2992]]
```

In [71]:
```

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, -10767382.35, 0, 98975360]

FP cost = NOK -10767382
TP revenue NOK 98975360
TN revenue NOK 0
FN cost NOK 0
The total profit is NOK 383,071
```

In [72]:

```
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('LightGBM model Confusion Matrix on Hold Out Dataset\n Theoretical Threshold = 0.0058 \n Profit improvement from defa
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 [73]:

```
# Optimal Threshold
threshold = 0.006
y_pred = (lightgbm.predict_proba(X_ho)[:, 1] > threshold).astype('float')
cf_matrix = confusion_matrix(y_ho, y_pred)
print(cf_matrix)
```

```
[[ 3977 55627]
 [   10  2992]]
```

In [74]:

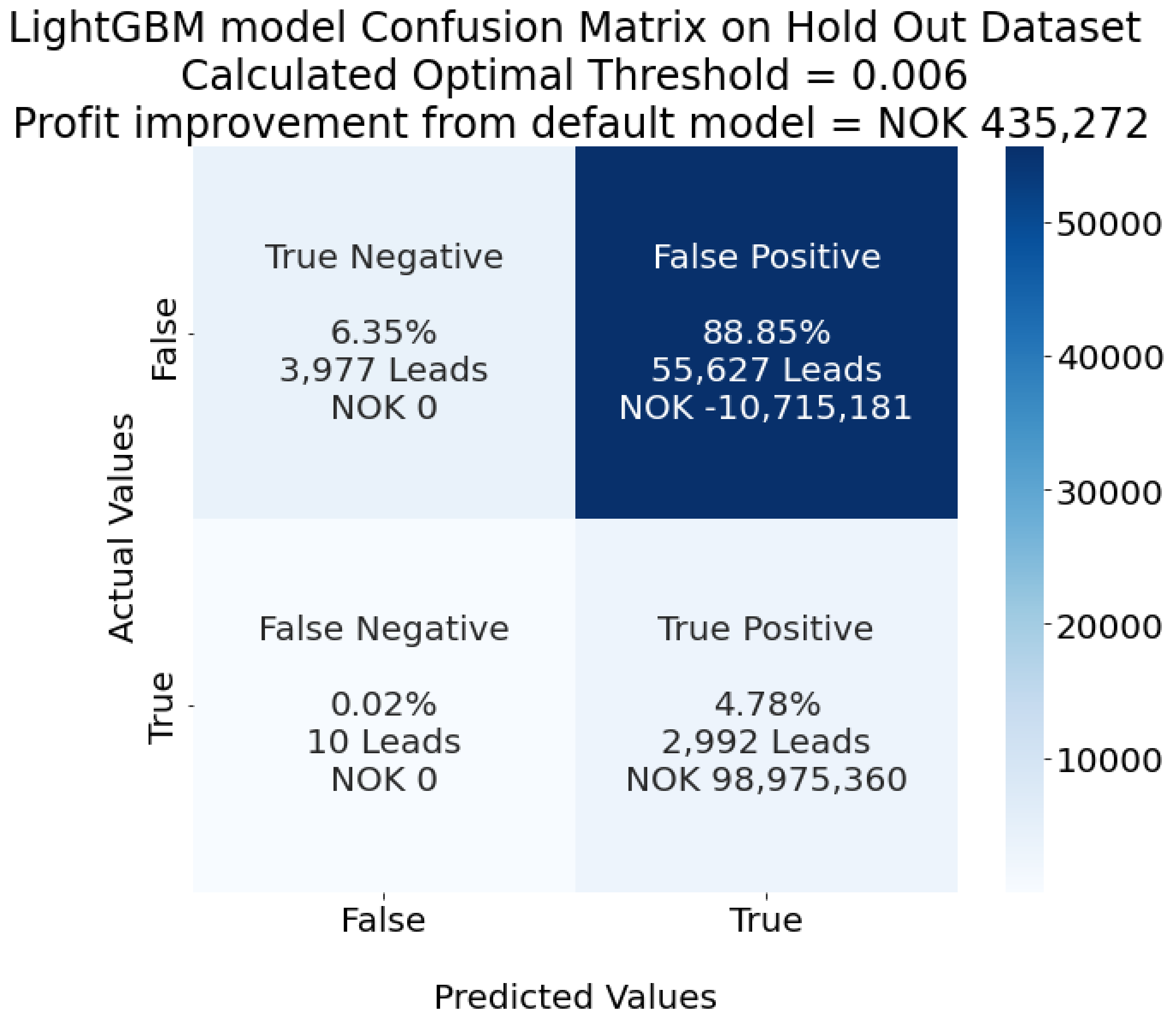
```
FP_Cost = round(cf_matrix[0,1] * Cost_p_lead, 2)
TP_Rev  = round(cf_matrix[1,1] * Revenue, 2)
TN_Rev  = round(cf_matrix[0,0] * 0,2)
FN_Cost = round(cf_matrix[1,0] * 0,2)
profit_matrix = [TN_Rev,FP_Cost, FN_Cost,TP_Rev]

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 total default profit is NOK {:,.0f}".format(Default))
print("The total profit is NOK {:,.0f}".format(Profit))
```

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

The profit_matrix contains:
[0, -10715180.83, 0, 98975360]

FP cost = NOK -10715181
TP revenue NOK 98975360
TN revenue NOK 0
FN cost NOK 0
The total default profit is NOK 87,824,907
The total profit is NOK 435,272
```

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('LightGBM model Confusion Matrix on Hold Out Dataset\n Calculated Optimal Threshold = 0.006 \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()
```



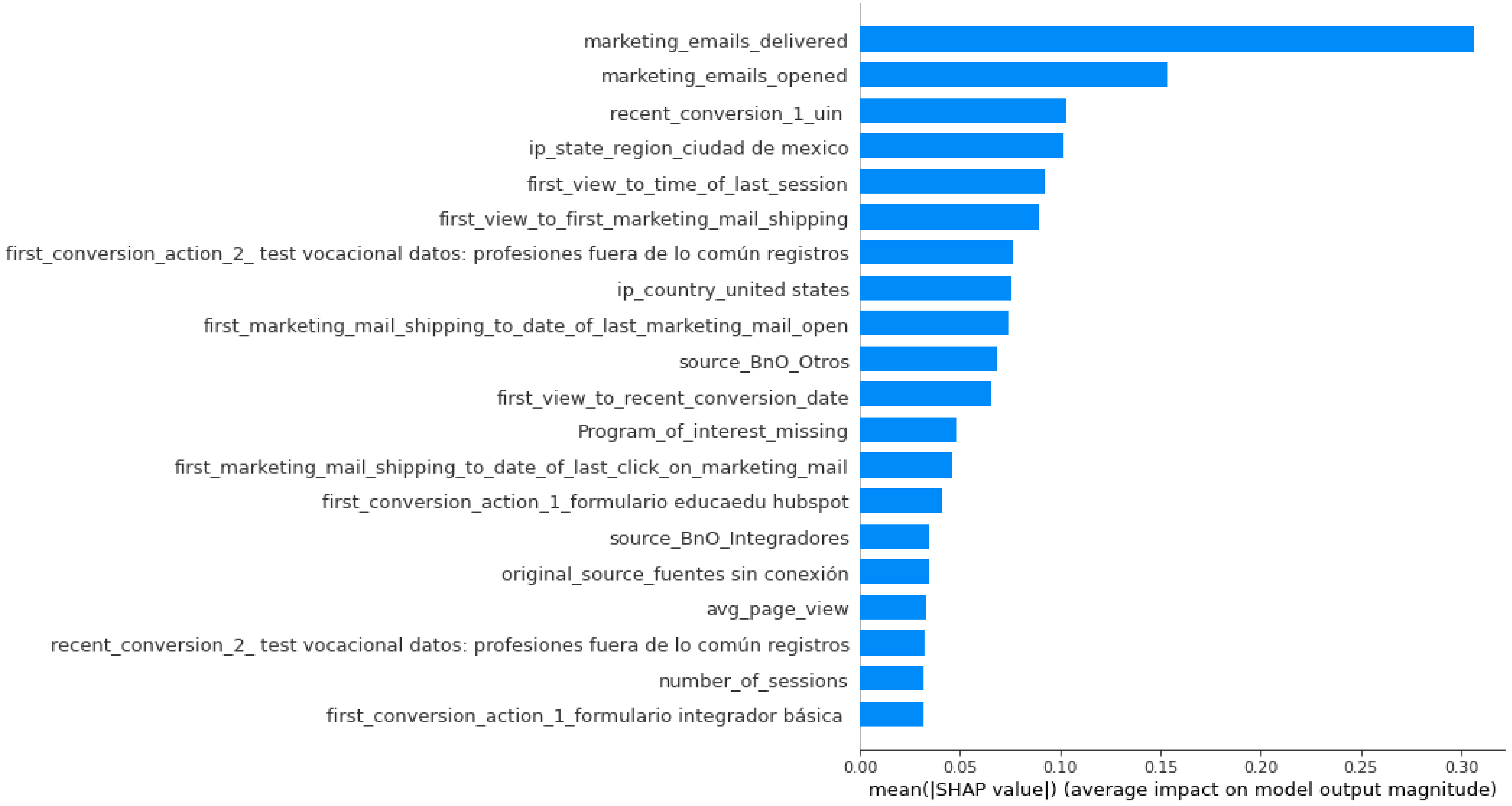
## SHAP features

In [19]:
```
X = df1.drop('enrolled', axis=1)
#X = csr_matrix(X1.values)
```

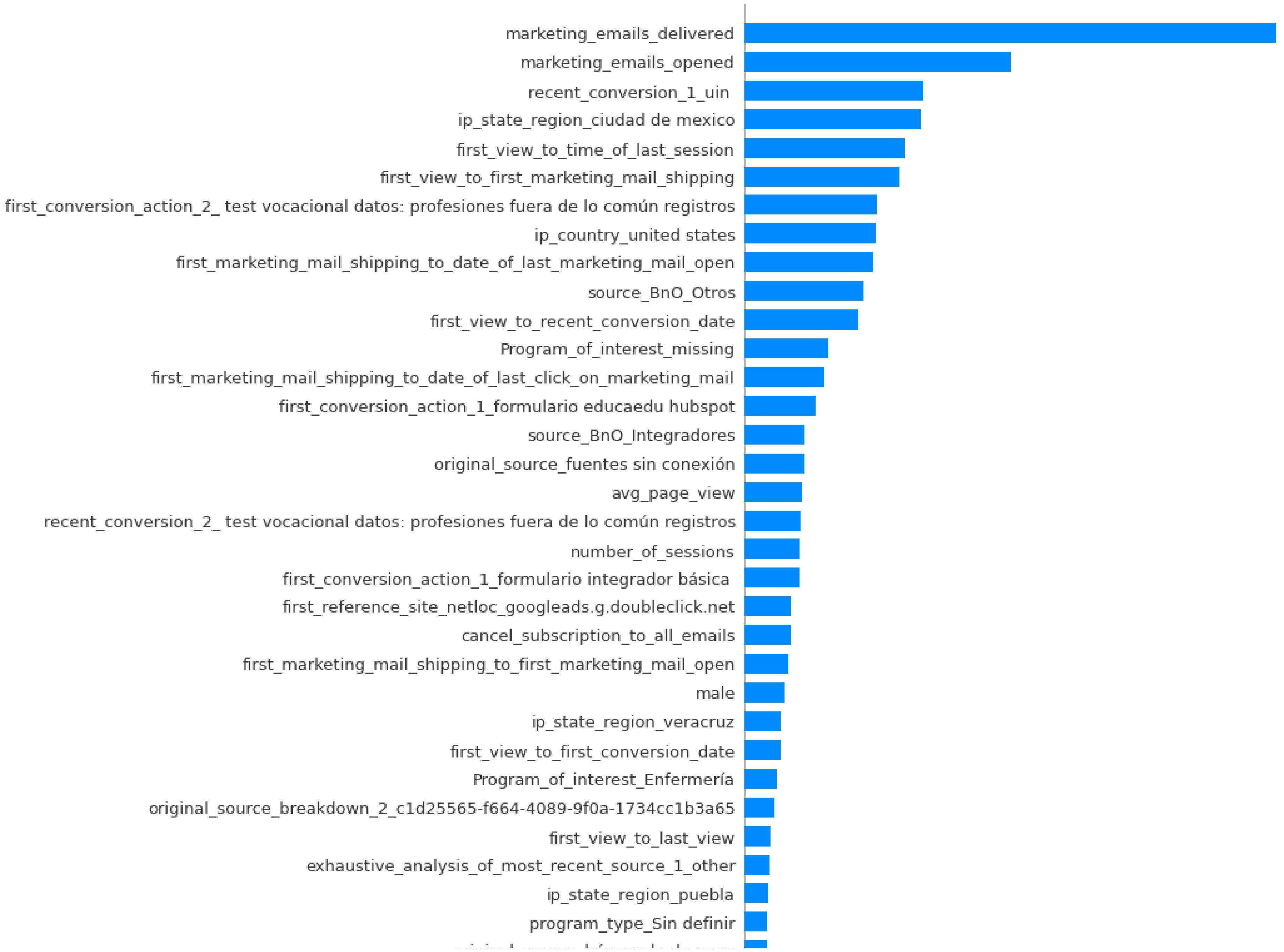
```
y = df1['enrolled']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

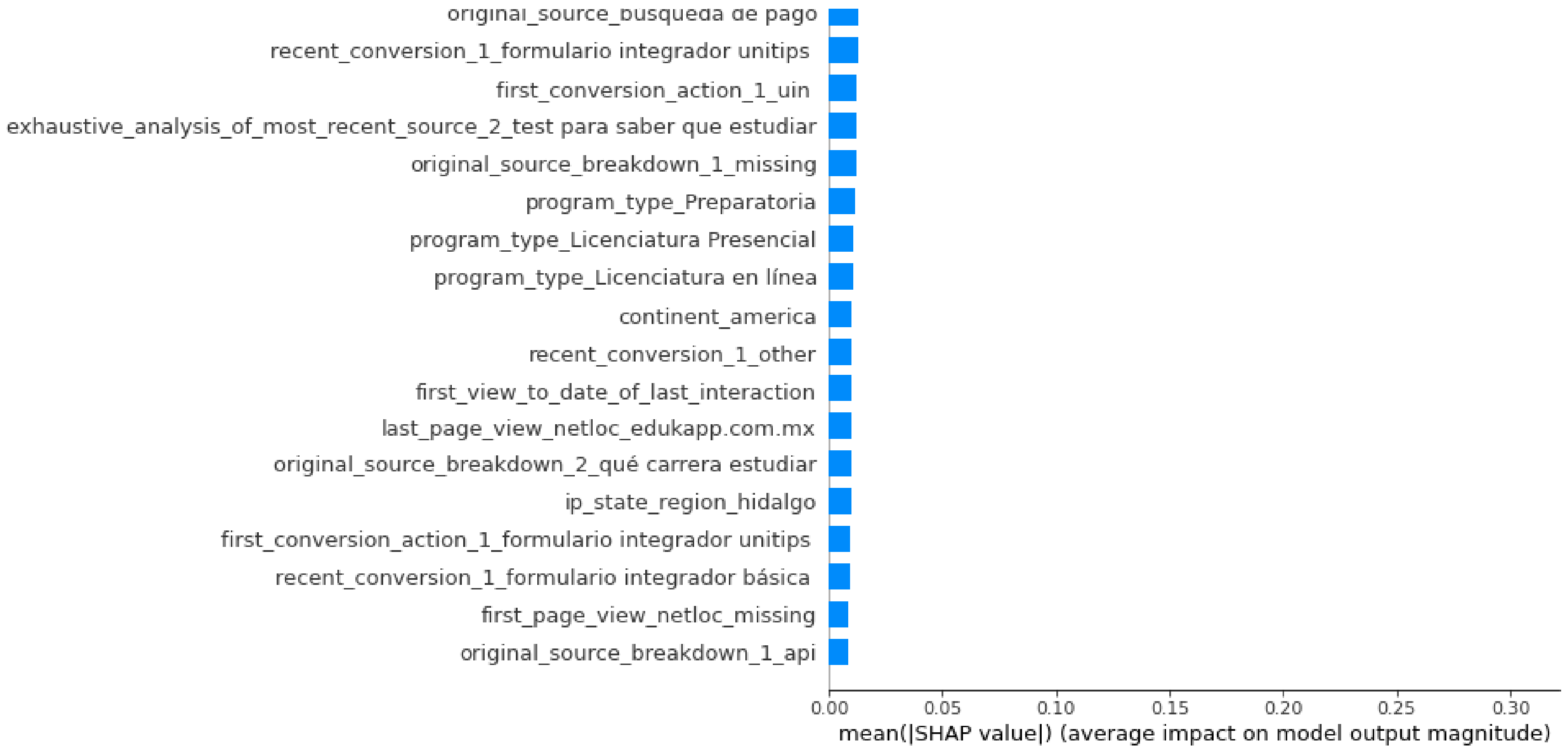
In [20]:
```
explainerlgbmc = shap.TreeExplainer(lightgbm)
shap_values_LightGBM_test = explainerlgbmc.shap_values(X_test)
shap_values_LightGBM_train = explainerlgbmc.shap_values(X_train)
```

In [24]:
```
shap.summary_plot(shap_values_LightGBM_train[1], X_train, plot_type="bar")
```



In [23]:
```
shap.summary_plot(shap_values_LightGBM_train[1], X_train, plot_type="bar", max_display=50)
```





## Explaining ROC AUC

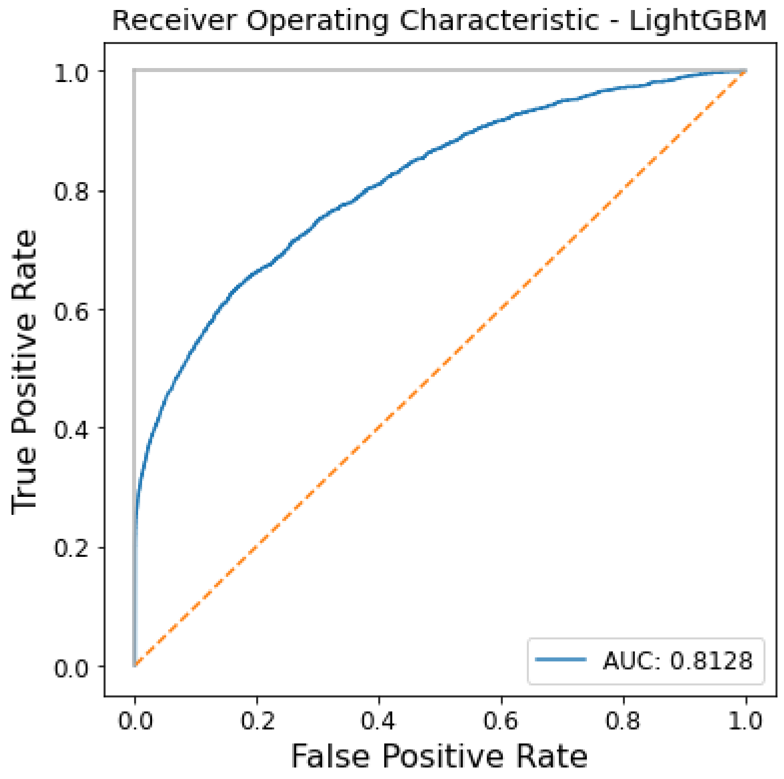
In [117…

```
y_pred_test = lightgbm.predict_proba(X_test)
false_positive_rate, true_positive_rate, threshold = roc_curve(y_test, y_pred_test[:,1])
print('roc_auc_score for LightGBM: ', roc_auc_score(y_test, y_pred_test[:,1]))
```

```
roc_auc_score for LightGBM:  0.8128306593863942
```

In [118…

```
import matplotlib.pyplot as plt
plt.subplots(1, figsize=(6,6))
plt.title('Receiver Operating Characteristic - LightGBM')
plt.rcParams['font.size'] = '20'
plt.plot(false_positive_rate, true_positive_rate)
plt.plot([0, 1], ls="--")
plt.rcParams['font.size'] = '12'
plt.plot([0, 0], [1, 0] , c=".7"), plt.plot([1, 1] , c=".7")
plt.ylabel('True Positive Rate', fontsize=16)
plt.xlabel('False Positive Rate', fontsize=16)
auc_score = 'AUC: ' + str(round(roc_auc_score(y_test, y_pred_test[:,1]),4))
plt.legend([auc_score])
plt.show()
```



## Heuristic approach

In [38]:

```
threshold = 0.006
y_pred = (lightgbm.predict_proba(X_test)[:, 1] > threshold).astype('float')
```

In [18]:

```
# min prediction value for enroller
y_pred = lightgbm.predict_proba(X_test)
y_pred
```

Out[18]:
```
array([[0.97217471, 0.02782529],
       [0.98635274, 0.01364726],
       [0.96437732, 0.03562268],
       ...,
       [0.97981787, 0.02018213],
       [0.98852211, 0.01147789],
       [0.84805768, 0.15194232]])
```

In [23]:
```
y_test_df = pd.DataFrame(y_test)
y_pred_test_df = pd.DataFrame(y_pred)
y_test_df['C'] = np.arange(len(y_test_df))
y_pred_test_df['C'] = np.arange(len(y_pred_test_df))
```

In [24]:
```
df5 = pd.merge(y_test_df, y_pred_test_df, on='C')
```

In [25]:
```
df3 =  df5[df5['enrolled'] != 0]
df4 =  df5[df5['enrolled'] != 1]
print(min(df4[1]))
print(min(df3[1]))
print(max(df4[1]))
print(max(df3[1]))
```

```
0.0007492975877396471
0.0024277130448915837
0.9429000211593515
0.9955287899822818
```

In [51]:
```
threshold = 0.08
y_pred = (lightgbm.predict_proba(X_ho)[:, 1] > threshold).astype('float')
cf_matrix = confusion_matrix(y_ho, y_pred)
print(cf_matrix)
```

```
[[55466  4138]
 [ 2002  1000]]
```