BUSINESS PROBLEM

Description

As the data scientist in your team, you must train a classification model to predict the label (0 or 1) for the customers. This involves:

- doing EDA, feature engineering and model selection.
- creating a model Python class to represent your model, following the framework provided by the machine learning engineer.
- creating a few Docker images to:
  - train and store your model
  - serve your EDA as a document that can be viewed by others

Implementation Requirements

EDA

You should explore the data found in the `companydata` database and prepare a few visualizations using the tool of your choice. You should also provide the exploration code you used for the initial model architecture selection, where you can also include some thoughts or commentary about your decisions if relevant.

These should go in the `eda` folder. A good solution that meets these requirements is a PDF export of a Jupyter notebook.

0.0. IMPORTS

In [4]:
```
xgb.__version__
```

Out[4]:
```
'1.5.1'
```

In [3]:
```
#data manipulation packages
import numpy as np
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```python
#data visualization
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib import gridspec
from scikitplot import metrics as mt

from IPython.display import Image
from IPython.core.display import HTML

#balanced imports
from imblearn import under_sampling as us
from imblearn import over_sampling as oversamp
from imblearn import combine as c

#sklearn
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler, RobustScaler
from sklearn.ensemble import ExtraTreesClassifier, RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split, RepeatedStratifiedKFold, cross_val_score, cross_val_predict, RandomizedSearc
from sklearn.metrics import accuracy_score, classification_report, cohen_kappa_score, recall_score, f1_score, roc_auc_score, plot_
from sklearn import neighbors as nh
import xgboost as xgb
from lightgbm import LGBMClassifier

from sqlalchemy import create_engine
import psycopg2
from boruta import BorutaPy
import joblib

import warnings
```

In [5]:

```python
#method to display all columns and rows
pd.set_option("display.max_columns", None)
pd.set_option("display.max_rows", 200)
pd.set_option('display.float_format', lambda x: '%.3f' % x)
warnings.filterwarnings('ignore')
```

In [ ]:
```
# Create an engine instance

alchemyEngine = create_engine(
    "postgresql+psycopg2://ds_user:ds_user@localhost/companydata")

# Connect to PostgreSQL server

dbConnection = alchemyEngine.connect()

# Read data from PostgreSQL database table and load into a DataFrame instance

query = """
        select fs2.* from feature_store fs2

"""

dataFrame = pd.read_sql(query, dbConnection)
```

In [474...
```
dataFrame.head()
```

Out[474...

	idx	attr_a	attr_b	scd_a	scd_b	label
0	4	1	c	0.662	4	0
1	6	1	d	0.730	2	0
2	7	3	c	0.350	3	1
3	8	3	c	0.097	3	0
4	11	5	b	0.041	3	0

WE HAVE NO KNOWLEDGE OF THE INFORMATION FOR EACH VARIABLE, THEREFORE, WE WILL ANALYZE THE DATA ONLY.

0.2. Helper Functions

In [44]:
```
def jupyter_settings():
    %matplotlib inline
    %pylab inline

    plt.style.use( 'bmh' )
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
    plt.rcParams['figure.figsize'] = [25, 12]
    plt.rcParams['font.size'] = 24

    display( HTML( '') )
    pd.options.display.max_columns = None
    pd.options.display.max_rows = None
    pd.set_option( 'display.expand_frame_repr', False )

    sns.set()
jupyter_settings()

def score_metrics(y_true,yhat):
    #accuracy
    acc = m.accuracy_score(y_true,yhat)
    #precision
    pre = m.precision_score(y_true,yhat)
    #recall
    recall = m.recall_score(y_true,yhat)
    #f1-score
    f1 = m.f1_score(y_true,yhat)

    score_dict = {
                  "Accuracy": acc,
                  "Precision":pre,
                  "Recall":recall,
                  "F1-Score":f1}


    final_score = pd.DataFrame.from_dict(score_dict, orient = 'index').T

    return final_score
```

```
Populating the interactive namespace from numpy and matplotlib
```

1.0. DATA DESCRIPTION

In [20]:

```
#raw data copy
df1 = dataFrame.copy()
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

1.1 DATA DIMENSION

Let's understand how big our dataset is. This will be important because a robust machine learning model needs a considerable amount of data to train our algorithm. As you can see below, we have 150k rows and 6 columns.

In [21]:
```
#Check number of columns and number of rows
print(f"Number of Columns: {df1.shape[1]}")
print(f"Number of Rows: {df1.shape[0]}")
```

```
Number of Columns: 6
Number of Rows: 150000
```

1.2. DATA TYPES

This step is important for us to understand what is the type of our features. Therefore, we need to treat each variable according to its specific type ... for example, we cannot treat a date as a number or as a text, we cannot treat a number as a text. It is necessary for us to understand the context of our business for the changes to be made.

In [22]:
```
#check data types
df1.dtypes
```

Out[22]:
```
idx          int64
attr_a       int64
attr_b      object
scd_a      float64
scd_b        int64
label        int64
dtype: object
```

1.3. CHECK MISSSING VALUES

This step is extremely important because our machine learning algorithms are not able of handling null values. To solve this problem there is no right answer, it will depend on your business context, we have to be aware of it and test what will be best suited to that situation. There are three ways to fix this.

Exclude lines with null values. Replace with the average or median, for example. Change according to the business context. This choice is very

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js the rows from our data set, depending on the amount of null values, we will eliminate a considerable amount of data so that our model could train.

In [23]:
```
#check NA
df1.isnull().sum().sort_values(ascending = False).head(30)
```

Out[23]:
```
idx        0
attr_a     0
attr_b     0
scd_a      0
scd_b      0
label      0
dtype: int64
```

1.4. DESCRIPTIVE STATISTICAL

1.4.1. NUMERICAL ATTRIBUTES

For the numerical variables we use two types of analysis.

Central tendency

- average
- median

Dispersion

- std - standard deviation
- min
- max
- range
- skew
- kurtosis

In [24]:
```
num_attributes = df1.select_dtypes(include = ['int64','float64'])
```



```
#central tendency - mean, median
```

```
ct1 = pd.DataFrame(num_attributes.apply(np.mean)).T
ct2 = pd.DataFrame(num_attributes.apply(np.median)).T

#Dispersion - Std, Min, Max, Range, Skew, Kurtosis
d1 = pd.DataFrame(num_attributes.apply(np.std)).T
d2 = pd.DataFrame(num_attributes.apply(min)).T
d3 = pd.DataFrame(num_attributes.apply(max)).T
d4 = pd.DataFrame(num_attributes.apply(lambda x: x.max() - x.min())).T
d5 = pd.DataFrame(num_attributes.apply(lambda x: x.skew())).T
d6 = pd.DataFrame(num_attributes.apply(lambda x: x.kurtosis())).T

m = pd.concat([d2, d3, d4, ct1, ct2, d1, d5, d6]).T.reset_index()
m.columns = [
    'attributes', 'min', 'max', 'range', 'mean', 'median', 'std', 'skew',
    'kurtosis'
]
m
```

Out[27]:

	attributes	min	max	range	mean	median	std	skew	kurtosis
0	idx	0.000	149999.000	149999.000	74999.500	74999.500	43301.270	0.000	-1.200
1	attr_a	1.000	5.000	4.000	2.491	1.000	1.656	0.438	-1.520
2	scd_a	0.000	1.000	1.000	0.500	0.500	0.288	-0.001	-1.198
3	scd_b	1.000	5.000	4.000	3.001	3.000	1.414	0.001	-1.300
4	label	0.000	1.000	1.000	0.091	0.000	0.288	2.834	6.031

1.4.2. DISCRETE VARIABLES

For categorical variables we will use the boxplot, which provides a visual analysis of the position, dispersion, symmetry, tails and outliers of the data set.

In [49]:

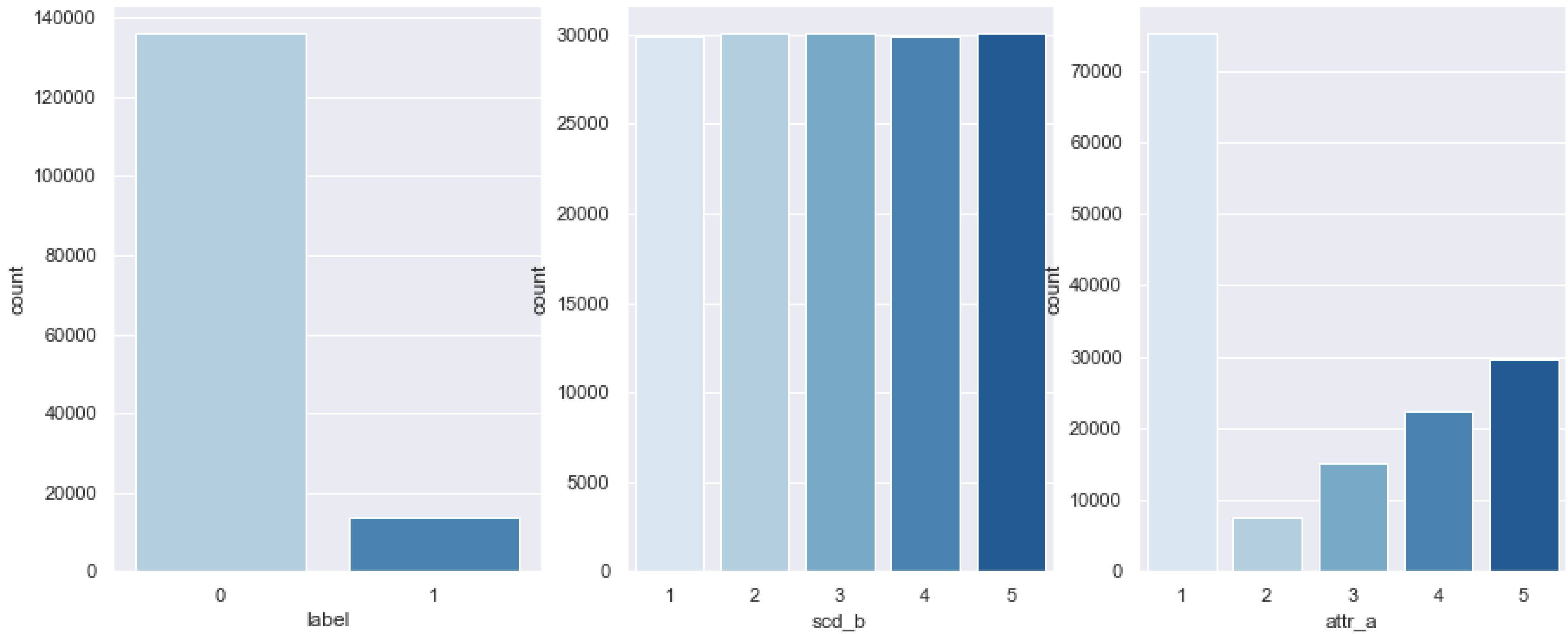
```
cat_attributes = df1[['label','scd_b','attr_a']]

plt.figure(figsize = (15,6))
plt.subplot(1,3,1)
sns.countplot(cat_attributes["label"], palette = 'Blues')

plt.subplot(1,3,2)
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
sns.countplot(cat_attributes["scd_b"], palette = 'Blues')

plt.subplot(1,3,3)
sns.countplot(cat_attributes["attr_a"], palette = 'Blues')
```

Out[49]:

```
<AxesSubplot:xlabel='attr_a', ylabel='count'>
```



2.0. FEATURE ENGINEERING

In [29]:

```
df2 = df1.copy()
```

3.0. EXPLORATORY DATA ANALYSIS

Exploratory Data Analysis (EDA) is the process of visualizing and analyzing data to extract insights from it. In other words, EDA is the process of summarizing important characteristics of data in order to gain better understanding of the dataset. Therefore, this part will be done in the following three steps:

- Univariate Analysis
- Bivariate Analysis
- Multivariate Analysis

References:

- https://medium.com/code-heroku/introduction-to-exploratory-data-analysis-eda-c0257f888676#:~:text=Exploratory%20Data%20Analysis%20(EDA)%20is,better%20understanding%20of%20the%20dataset.

- https://hotcubator.com.au/research/what-is-univariate-bivariate-and-multivariate-analysis/

In [30]:
```
df3 = df2.copy()
```

3.1. UNIVARIATE ANALYSIS

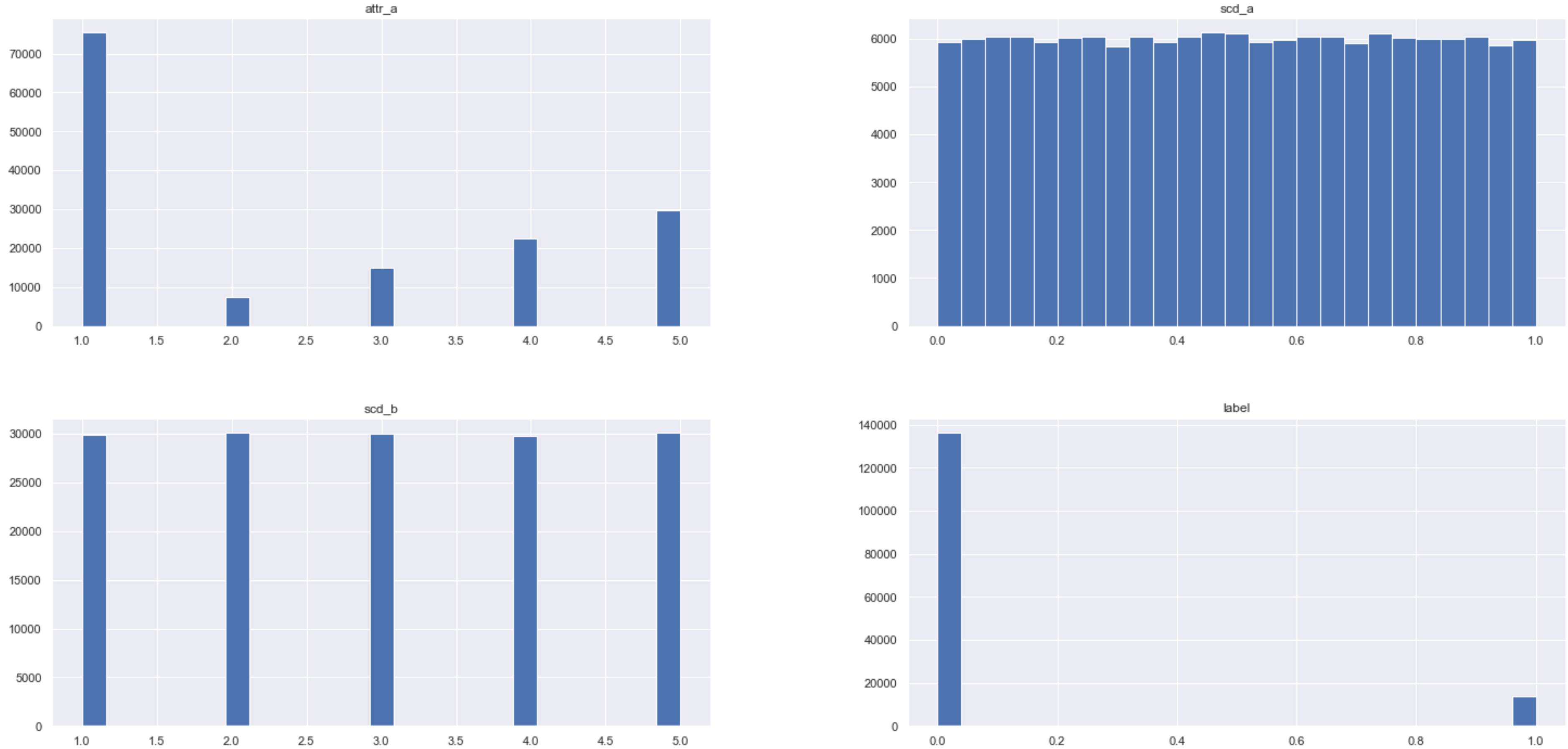
3.1.1. DISTRIBUTION

Univariate analysis is the most basic form of statistical data analysis technique. When the data contains only one variable and doesn't deal with a causes or effect relationships then a Univariate analysis technique is used. To visualize the distribution of our numeric variables, we will construct a histogram to see how each feature behaves.

In [52]:
```
aux = df3.drop(columns = 'idx', axis = 1)

plt.figure(figsize= (20,20))
aux.hist(bins = 25);
```

```
<Figure size 1440x1440 with 0 Axes>
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js



- There know high variance
- scd_a semms constant
- Imbalanced data

3.1.2. RESPONSE VARIABLE

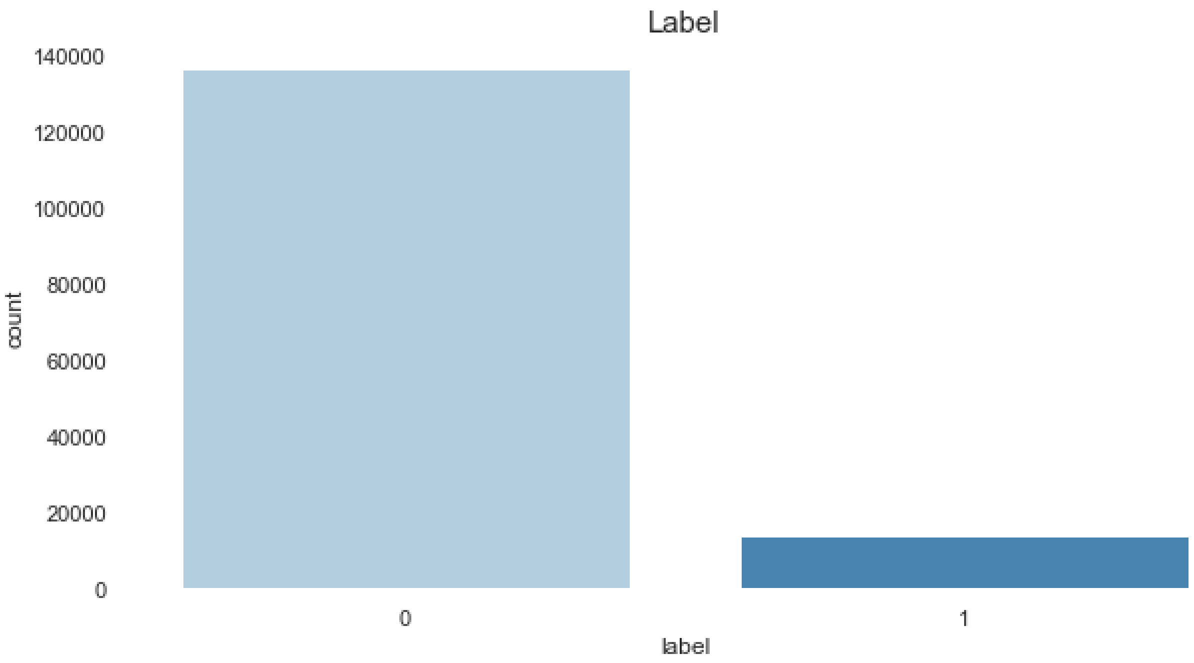
In [53]:

```
plt.figure(figsize = (10,5 ))
sns.countplot(x = 'label',data = df3, palette = 'Blues')
plt.box(False)
plt.title("Label", fontsize = 15)
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
Text(0.5, 1.0, 'Label')
```

Out[53]:



In [54]:

```
Label = df3['label'].value_counts() / len(df3)
Label
```

Out[54]:

```
0    0.909
1    0.091
Name: label, dtype: float64
```

3.2. BIVARIATE ANALYSIS

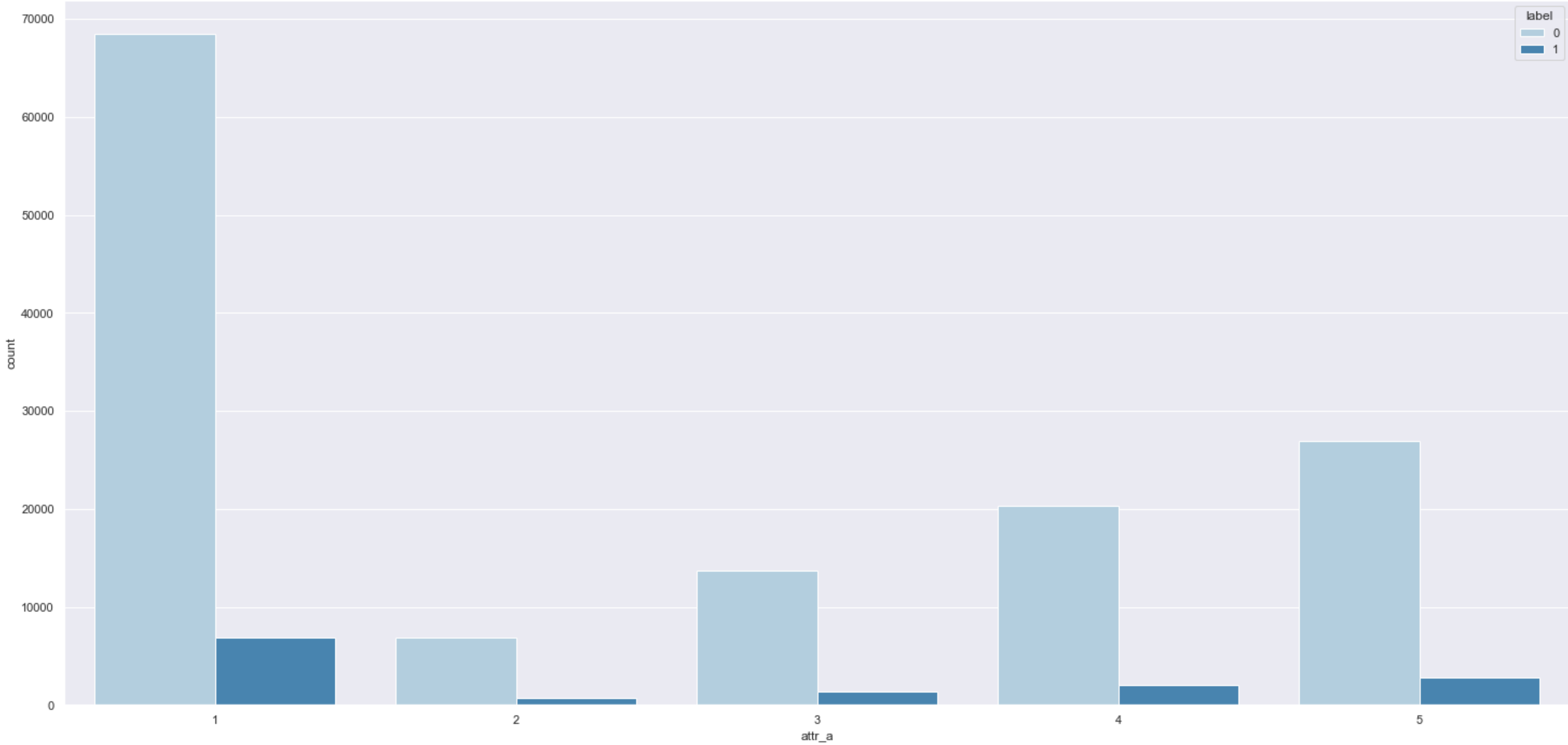
3.2.1. ATTR_A

In [447…

```
aux = df2[['attr_a','label']].groupby(["attr_a",'label']).size().reset_index()
aux = aux.rename(columns ={0:"count"})
sns.barplot(x = 'attr_a', y = 'count', hue = 'label', data = aux, palette = 'Blues')
```

<AxesSubplot:xlabel='attr_a', ylabel='count'>

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js



3.2.2. ATTR_B

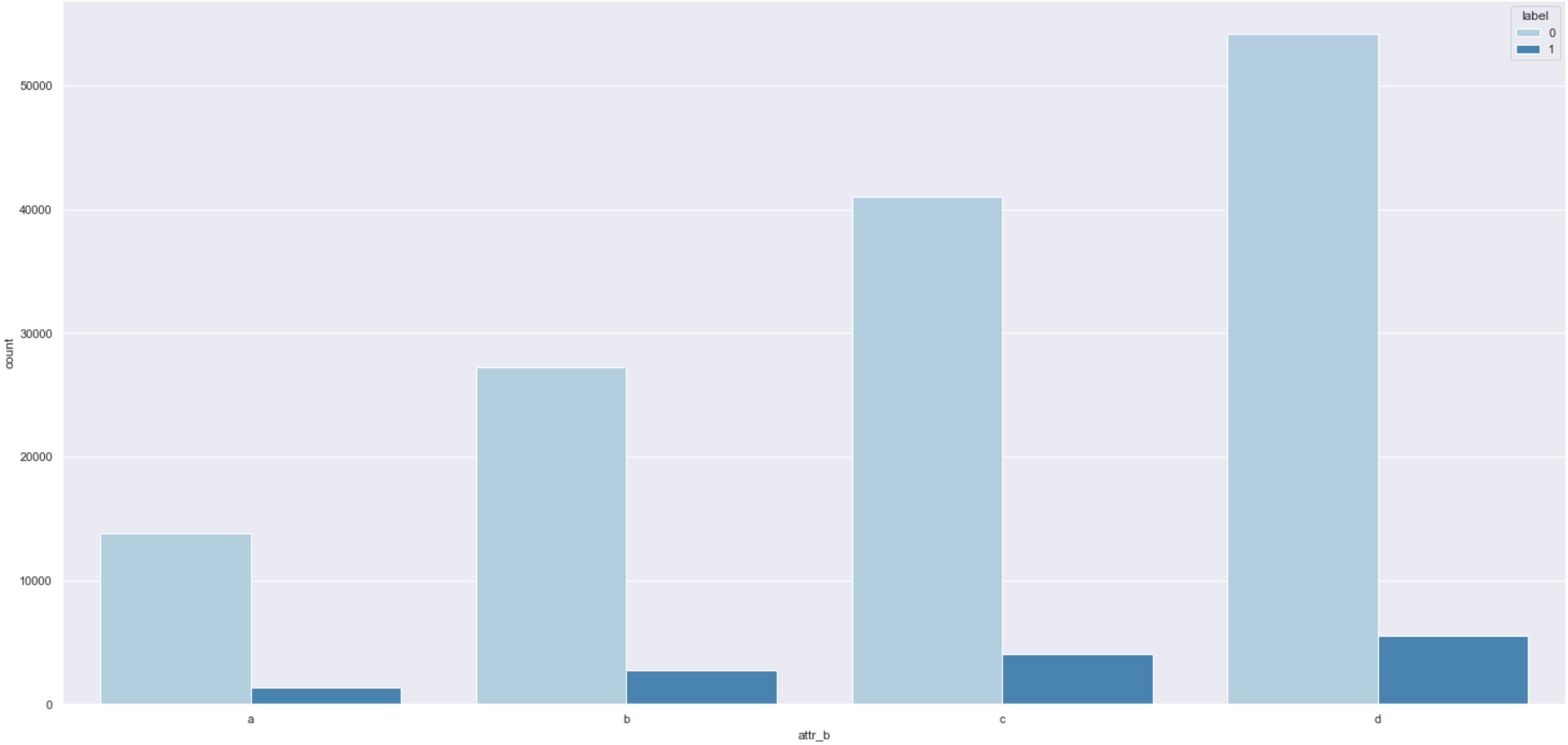
In [448…

```
aux = df2[['attr_b','label']].groupby(["attr_b",'label']).size().reset_index()
aux = aux.rename(columns ={0:"count"})
sns.barplot(x = 'attr_b', y = 'count', hue = 'label', data = aux, palette = 'Blues')
```

Out[448…

```
<AxesSubplot:xlabel='attr_b', ylabel='count'>
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js



3.2.3. SCD_A

In [452...

```
aux = df2[['scd_a','label']].groupby(['label']).mean().reset_index()
aux
```

Out[452...

	label	scd_a
0	0	0.500
1	1	0.500

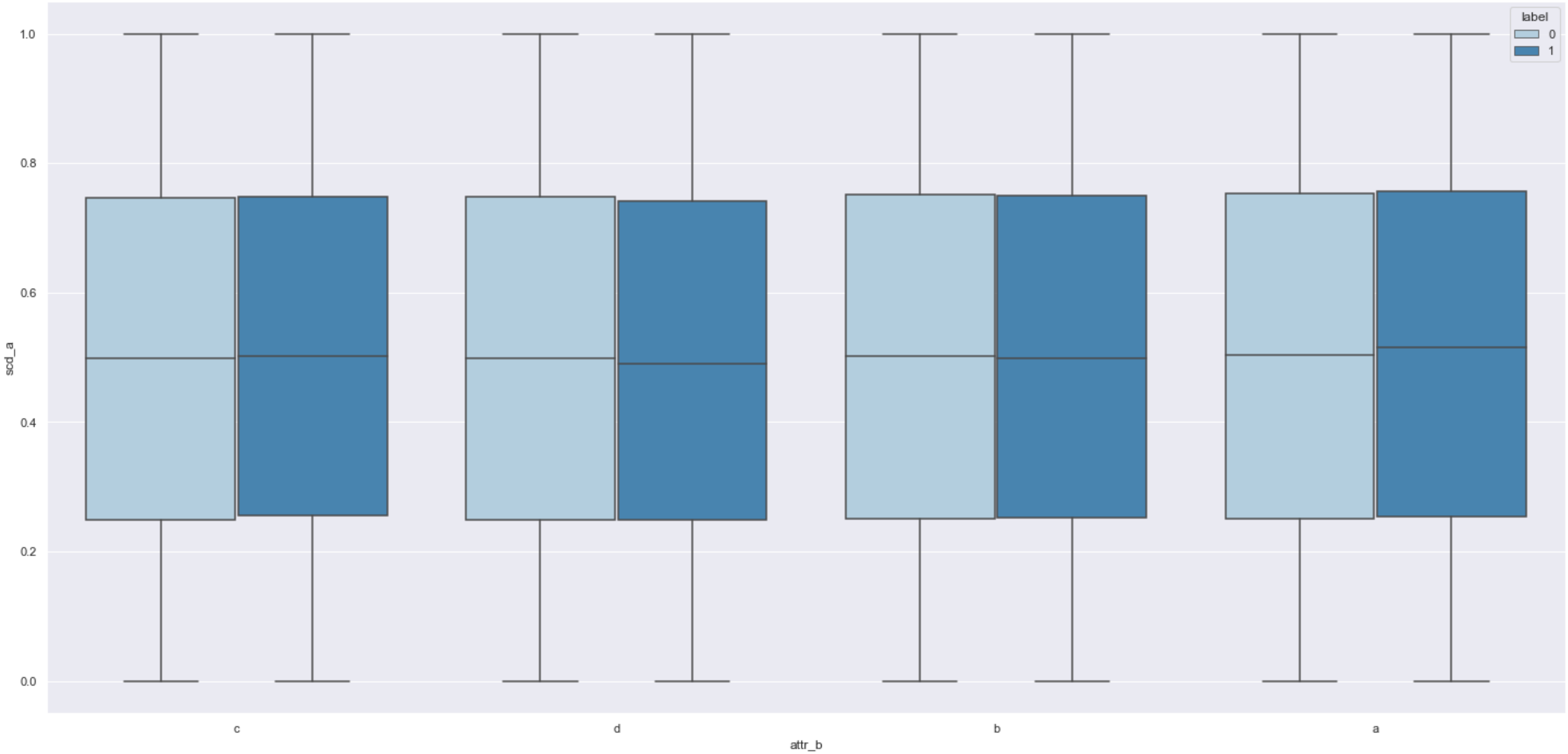
Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [457…

```
sns.boxplot(data = df2, x = 'attr_b',y = 'scd_a', hue = 'label', palette = 'Blues')
```

Out[457…

```
<AxesSubplot:xlabel='attr_b', ylabel='scd_a'>
```



3.3.4. SCD_B

In [459…

```
aux = df2[['scd_b','label']].groupby(['label']).mean().reset_index()
aux
```

Out[459…

	label	scd_b

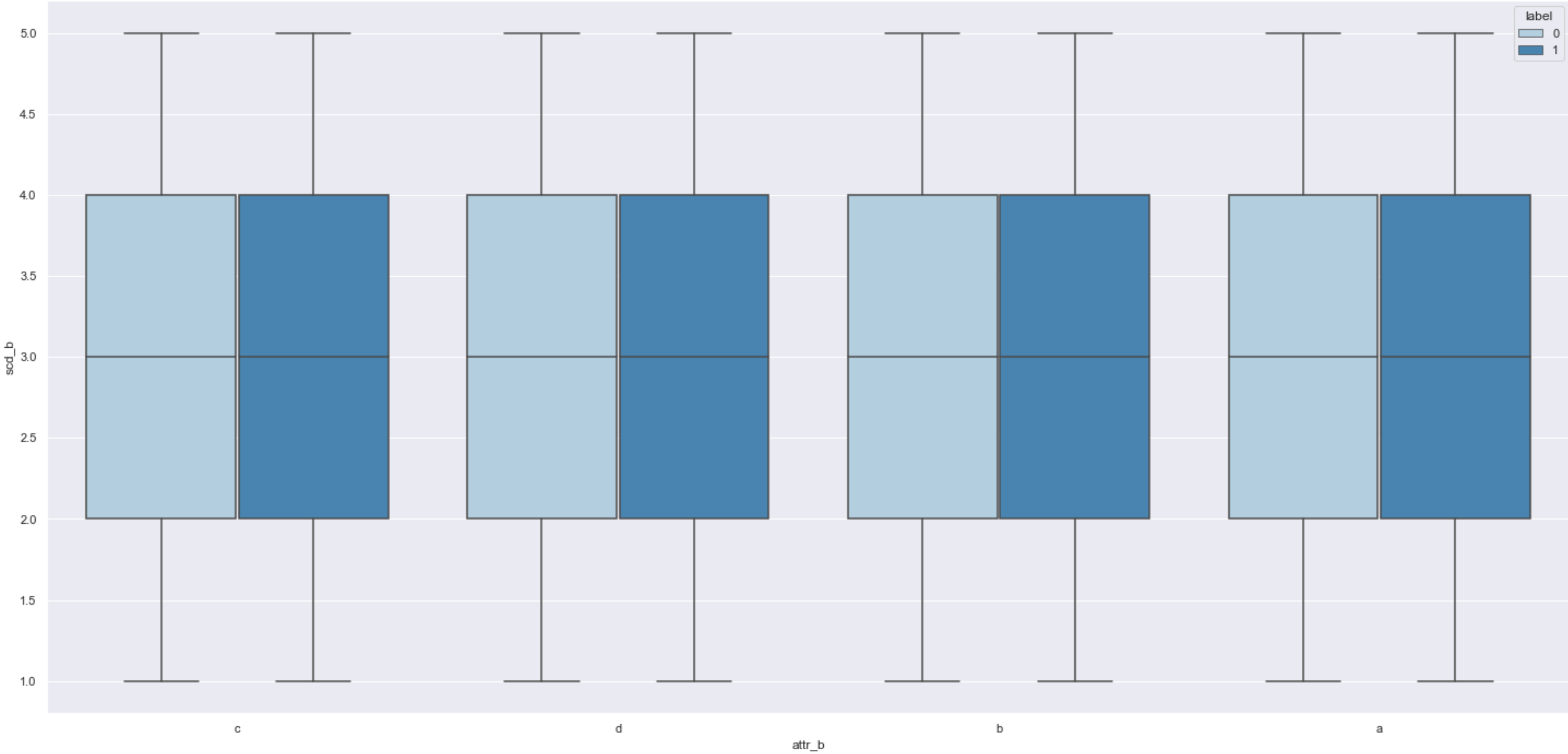
Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

	label	scd_b
1	1	3.003

There's no difference between label

In [460…
```
sns.boxplot(data = df2, x = 'attr_b',y = 'scd_b', hue = 'label', palette = 'Blues')
```

Out[460…
```
<AxesSubplot:xlabel='attr_b', ylabel='scd_b'>
```



3.3 MULTIVARIATE ANALYSIS

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

Multivariate analysis (MVA) is based on the principles of multivariate statistics, which involves observation and analysis of more than one statistical outcome variable at a time. Typically, MVA is used to address the situations where multiple measurements are made on each experimental unit and the relations among these measurements and their structures are important.

This step will be done because the machine learning algorithms assume some premises, among them, the Occam's razor principle. Ockham's rule is associated with the requirement to recognize, for each object analyzed, only one sufficient explanation.

We use the concepts of linear dependency, that is, let's assume that we have two columns that are linearly dependent, that is, whose influence on the problem is similar, we can take one that the effect will be maintained. To find this, we can look at the correlation between variables as an alternative so that we can reduce the dimensionality of our dataset.

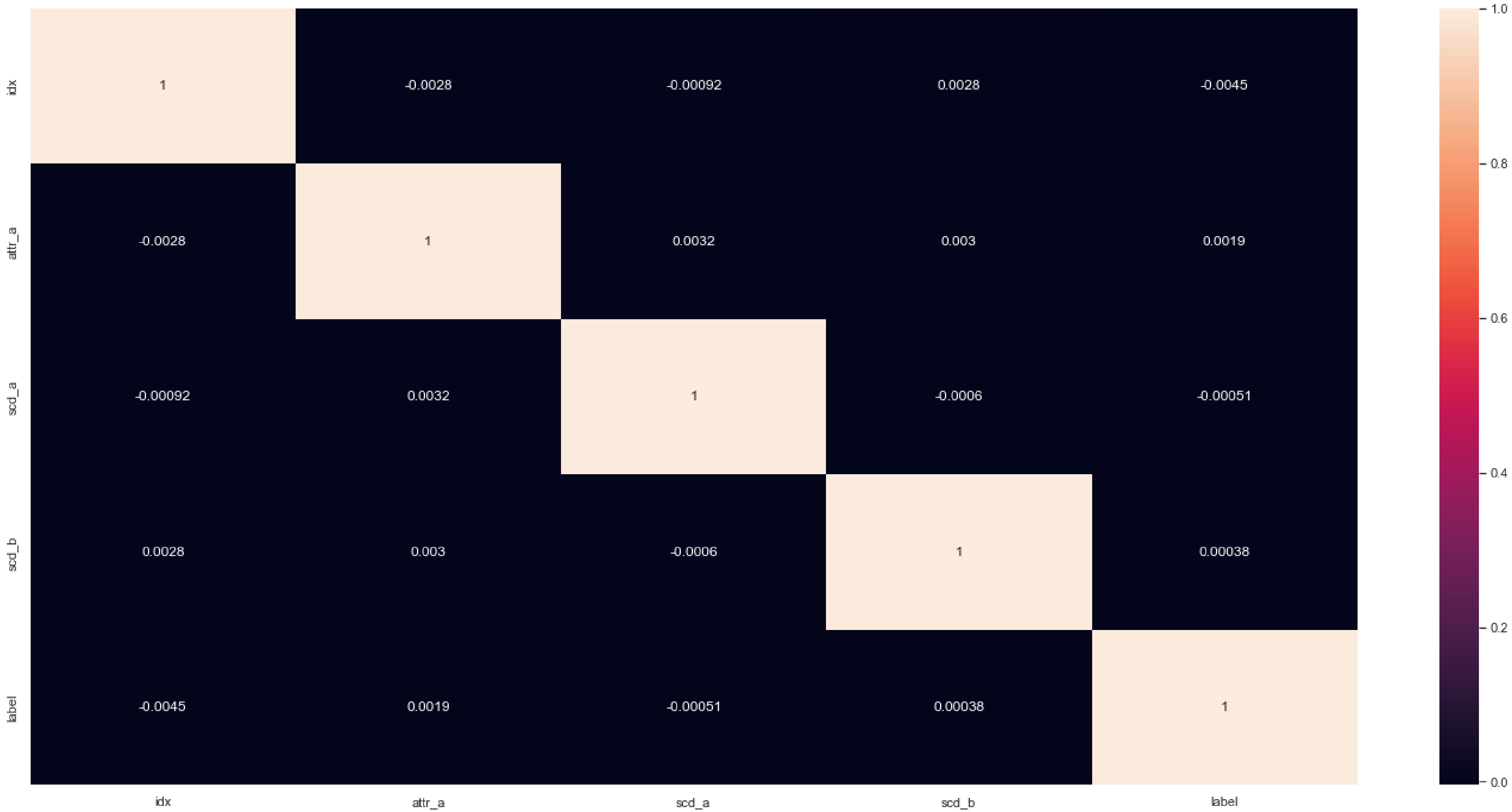
In [441…

```
correlation = num_attributes.corr(method = 'pearson')
sns.heatmap(correlation, annot = True)
```

Out[441…

```
<AxesSubplot:>
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js



4.0. DATA PREPARATION

In [55]:
```
df4 = df3.copy()
```

4.1. ENCODING

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

- Label Enconder

LabelEncoder is a utility class to help normalize labels such that they contain only values between 0 and n_classes-1. This is sometimes useful for writing efficient Cython routines.It can also be used to transform non-numerical labels (as long as they are hashable and comparable) to numerical labels.

- One Hot Encoder

This type of encoding can be obtained with the OneHotEncoder, which transforms each categorical feature with n_categories possible values into n*categories binary features, with one of them 1, and all others 0. By default, the values each feature can take is inferred automatically from the dataset and can be found in the categories* attribute.

- Ordinal Encoder

To convert categorical features to such integer codes, we can use the OrdinalEncoder. This estimator transforms each categorical feature to one new feature of integers (0 to n_categories - 1) as being ordered.

In [57]:

```
df4['attr_b'].value_counts()
```

Out[57]:

```
d    59758
c    45083
b    30017
a    15142
Name: attr_b, dtype: int64
```

In [58]:

```
label_encoder = {"a": 0, "b": 1, "c": 2, "d": 3}

df4['attr_b'] = df4['attr_b'].map(label_encoder)
```

4.2. BALANCE DATA

In [225…

```
#define sampler
smt = c.SMOTETomek(sampling_strategy = 'auto', random_state = 32, n_jobs = -1)

#apply sampler
x_smt, y_smt = smt.fit_resample(df4, df4['label'])
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [226…

```
x_smt.shape, y_smt.shape
```

Out[226…

```
((254962, 6), (254962,))
```

4.3. SPLIT DATA INTO TRAINING AND TEST

We will separate the training and test data that will be used in the application of the models. As the objective of the project is to make a sales forecast for the next six weeks. The test data set will contain random data.

In [252…

```
#predict variables
#X = df4.drop(columns = ['label'], axis = 1)
#target variable
#y = df4['label']

#Split data
X_train, X_val, y_train, y_val = train_test_split(x_smt, y_smt , test_size = 0.3, random_state = 42)
```

In [253…

```
X_train = X_train.drop(columns = ['label'], axis = 1)
X_val = X_val.drop(columns = ['label'], axis = 1)
```

4.4. RESCALING

In [229…

```
mms_attr_a = MinMaxScaler()
X_train['attr_a'] = mms_attr_a.fit_transform(X_train[['attr_a']])

mms_attr_b = MinMaxScaler()
X_train['attr_b'] = mms_attr_b.fit_transform(X_train[['attr_b']])

mms_scd_a = MinMaxScaler()
X_train['scd_a'] = mms_scd_a.fit_transform(X_train[['scd_a']])

mms_scd_b = MinMaxScaler()
X_train['scd_b'] = mms_scd_a.fit_transform(X_train[['scd_b']])
```

In [230…

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
filename = 'models/mms_attr_a.sav'
#joblib.dump(mms_attr_a, filename)
pickle.dump(mms_attr_a, open(filename, 'wb'))

filename = 'models/mms_attr_b.sav'
#joblib.dump(mms_attr_b, filename)
pickle.dump(mms_attr_b, open(filename, 'wb'))

filename = 'models/mms_scd_a.sav'
#joblib.dump(mms_scd_a, filename)
pickle.dump(mms_scd_a, open(filename, 'wb'))

filename = 'models/mms_scd_b.sav'
#joblib.dump(mms_scd_b, filename)
pickle.dump(mms_scd_b, open(filename, 'wb'))
```

In [231…

```
X_val['attr_a'] = mms_attr_a.transform(X_val[['attr_a']])
X_val['attr_b'] = mms_attr_b.transform(X_val[['attr_b']])
X_val['scd_a'] = mms_scd_a.transform(X_val[['scd_a']])
X_val['scd_b'] = mms_scd_a.transform(X_val[['scd_b']])
```

5.0. FEATURE SELECTION

In [254…

```
X_train_n = X_train.copy()
X_val_n = X_val.copy()
```

In [255…

```
X_train_n = X_train_n.drop(columns = 'idx', axis = 1)
X_val_n = X_val_n.drop(columns = 'idx', axis = 1)
```

5.1. RANDOM FOREST - IMPORTANCE FEATURES

In [256…

```
# model definition
forest = ExtraTreesClassifier( n_estimators=250, random_state=0, n_jobs=-1 )
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
x_train_n = X_train_n
```

```
y_train_n = y_train.values
forest.fit( x_train_n, y_train_n )
```

Out[256…

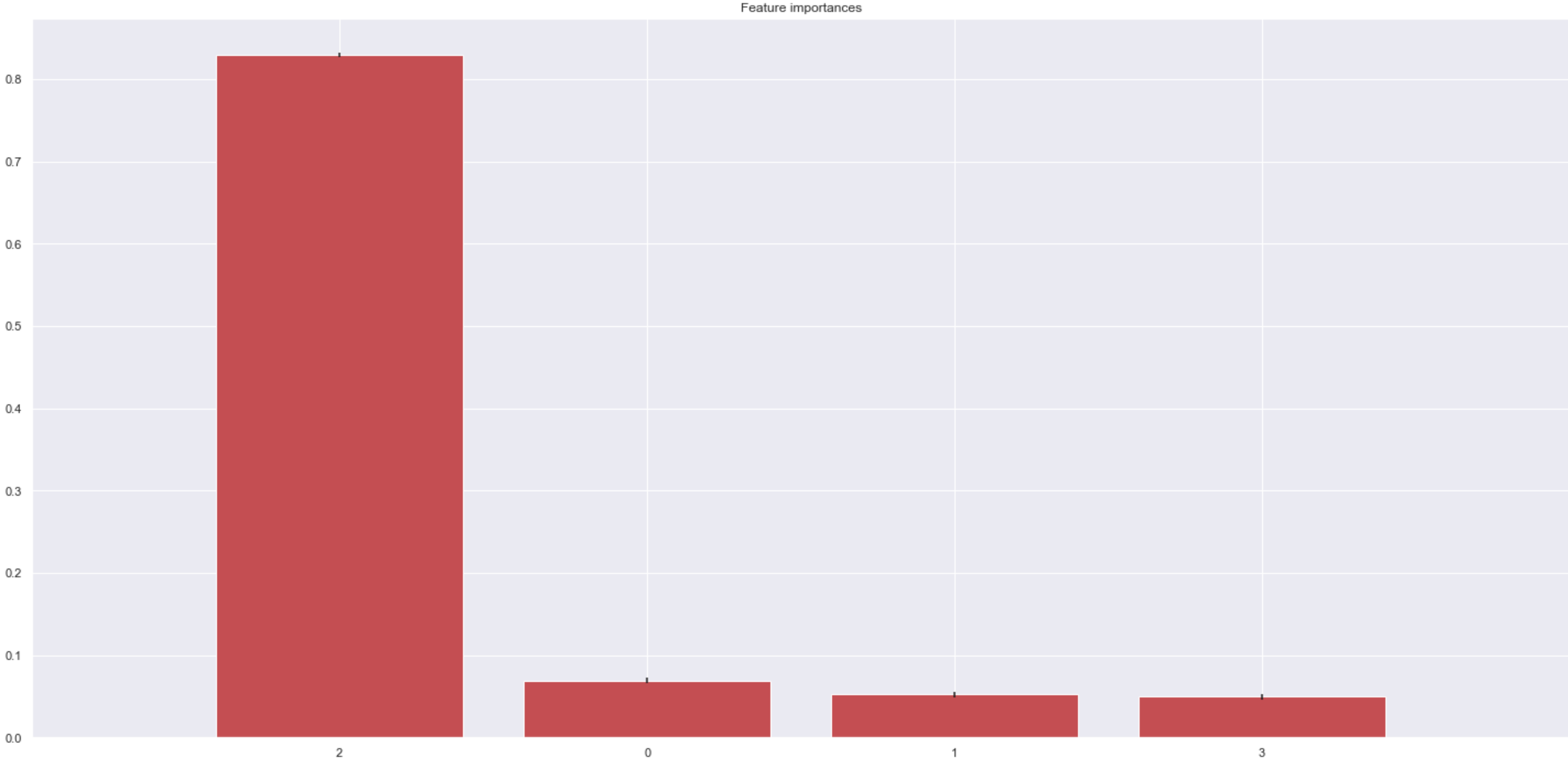
```
ExtraTreesClassifier(n_estimators=250, n_jobs=-1, random_state=0)
```

In [257…

```
importances = forest.feature_importances_
std = np.std([tree.feature_importances_ for tree in forest.estimators_], axis=0)
indices = np.argsort(importances)[::-1]

# Print the feature ranking
print("Feature ranking:")
df = pd.DataFrame()
for i, j in zip( x_train_n, forest.feature_importances_ ):
    aux = pd.DataFrame( {'feature': i, 'importance': j}, index=[0] )
    df = pd.concat( [df, aux], axis=0 )

print( df.sort_values( 'importance', ascending=False ) )

# Plot the impurity-based feature importances of the forest
plt.figure()
plt.title("Feature importances")
plt.bar(range(x_train_n.shape[1]), importances[indices], color="r", yerr=std[indices], align="center")
plt.xticks(range(x_train_n.shape[1]), indices)
plt.xlim([-1, x_train_n.shape[1]])
plt.show()
```

```
Feature ranking:
  feature  importance
0   scd_a       0.829
0  attr_a       0.069
0  attr_b       0.052
0   scd_b       0.049
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js



In [258...

```
df1.head()
```

Out[258...

	idx	attr_a	attr_b	scd_a	scd_b	label
0	4	1	c	0.662	4	0
1	6	1	d	0.730	2	0
2	7	3	c	0.350	3	1
3	8	3	c	0.097	3	0
4	11	5	b	0.041	3	0

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

5.2. WE COULD APPLY: RFE OR BORUTA

6.0. MACHINE LEARNING MODELLING

In this section, we will finally build our predictive models. Therefore, we will use 4 machine learning algorithms, which will be:

- Linear Regression
- Random Forest Regressor
- XGBoost
- LGBM

For each algorithm, we will build what we call the cross-validation technique. Cross-validation is primarily used in applied machine learning to estimate the skill of a machine learning model on unseen data. That is, to use a limited sample in order to estimate how the model is expected to perform in general when used to make predictions on data not used during the training of the model.

It is a popular method because it is simple to understand and because it generally results in a less biased or less optimistic estimate of the model skill than other methods, such as a simple train / test split.

For this, we will separate our predictor variables from our target variable and then separate them in training and testing.

6.1. LOGISTIC REGRESSION

In [259…

```
y_test = y_val
```

In [260…

```
X_val_n.head()
```

Out[260…

	attr_a	attr_b	scd_a	scd_b
14983	1	3	0.101	2
9919	4	1	0.801	3
136636	1	1	0.722	2

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

	attr_a	attr_b	scd_a	scd_b
144277	1	1	0.226	2

In [261…

```
#define model
logreg = LogisticRegression()

#training model
logreg.fit(X_train_n.values,y_train.values.ravel())

#predict
yhat_log = logreg.predict(X_val_n)

# AUC-ROC
logreg_cv = cross_val_predict(logreg, X_train_n, y_train, cv=5, method='decision_function')
logreg_roc = roc_auc_score(y_train, logreg_cv)

#performance
result_log = pd.DataFrame(["Logistic Regression",accuracy_score(y_test,yhat_log),cohen_kappa_score(y_test,yhat_log),
                          recall_score(y_test,yhat_log), f1_score(y_test,yhat_log),
                           precision_score(y_test,yhat_log), logreg_roc]).T
result_log.columns = ["Model",'Accuracy',"Kappa Score","Recall","F1-Score","Precision_score","roc_auc_score"]

#classification_report
print(classification_report(y_test, yhat_log))
```

```
              precision    recall  f1-score   support

           0       0.60      0.59      0.60     38282
           1       0.60      0.60      0.60     38207

    accuracy                           0.60     76489
   macro avg       0.60      0.60      0.60     76489
weighted avg       0.60      0.60      0.60     76489
```

In [262…

```
#performance logistic regression
result_log
```

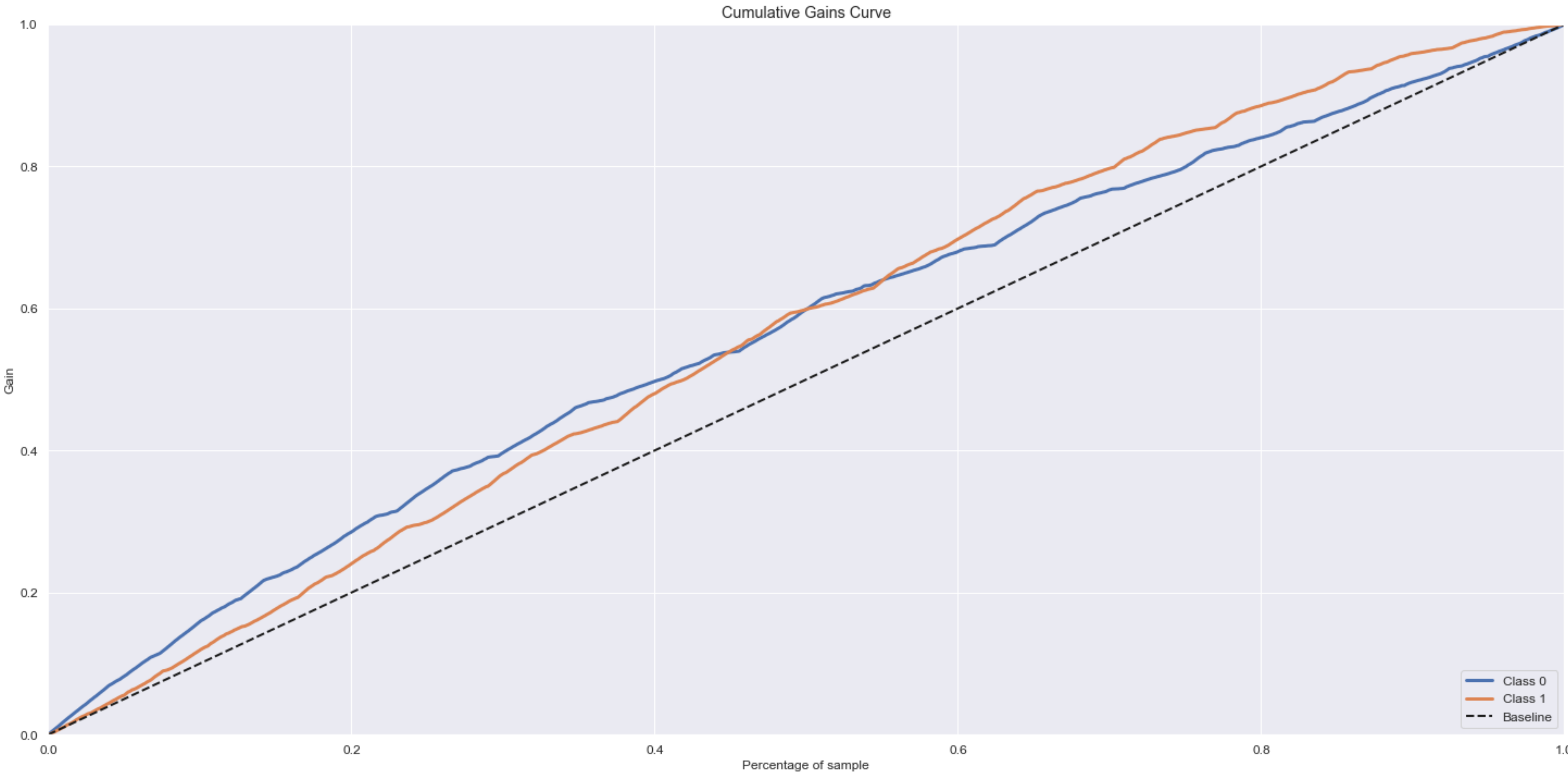
Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

Recall	F1-Score	Precision_score	roc_auc_score

	Model	Accuracy	Kappa Score	Recall	F1-Score	Precision_score	roc_auc_score
0	Logistic Regression	0.598	0.195	0.601	0.599	0.597	0.626

In [263…

```
## Accumulative Gain
y_hat_lr = logreg.predict_proba(X_val_n)

skplt.metrics.plot_cumulative_gain( y_val, y_hat_lr );
```



In [264…

```
#Scikitplot library is there to help
l, yhat_lr )
```

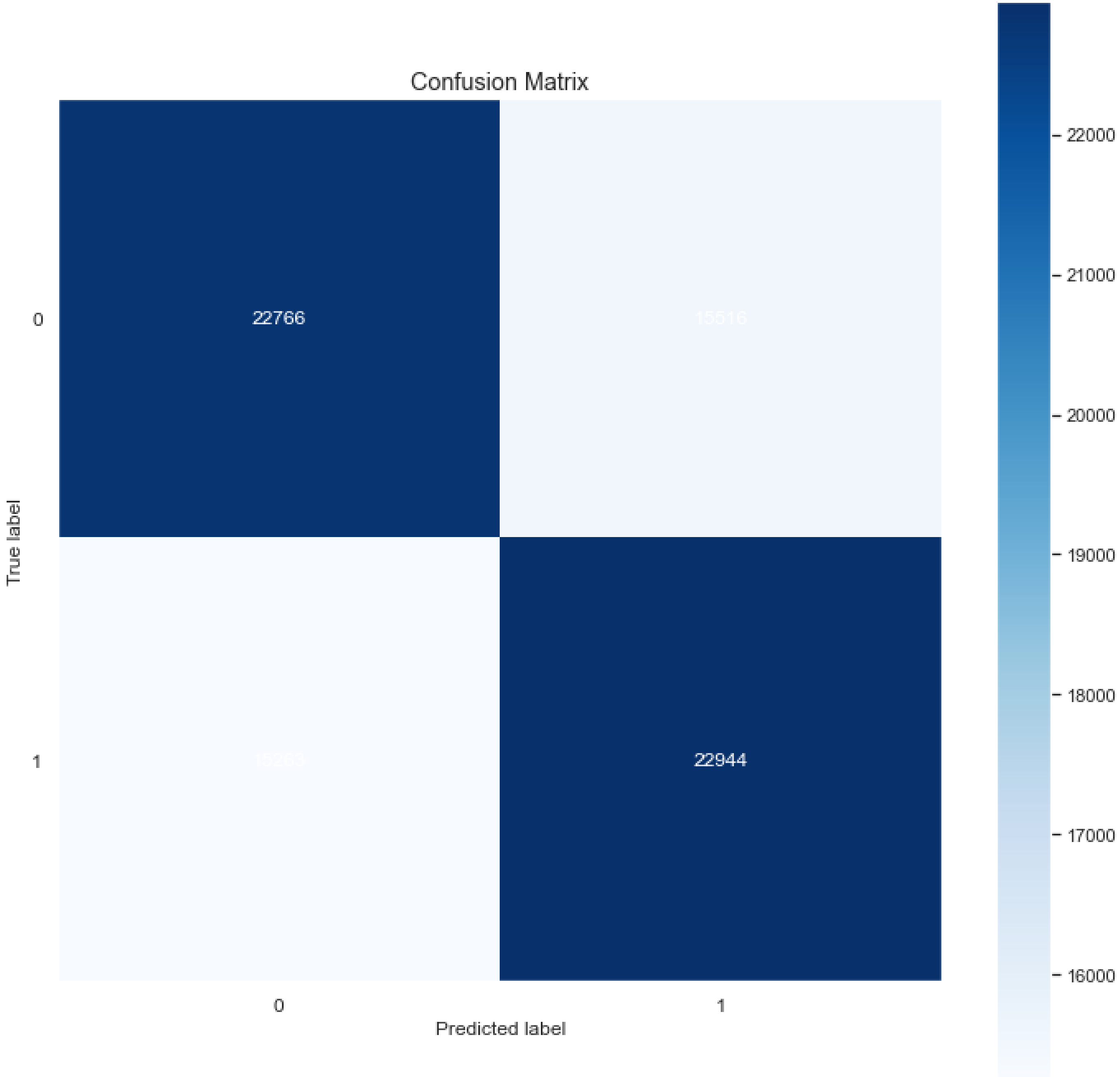
Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [265…

```
#confusion_matrix
mt.plot_confusion_matrix(y_test,yhat_log, normalize = False, figsize = (12,12))
```

Out[265…

```
<AxesSubplot:title={'center':'Confusion Matrix'}, xlabel='Predicted label', ylabel='True label'>
```



Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
X_train.head()
```

Out[274...

	idx	attr_a	attr_b	scd_a	scd_b
172339	82615	2	2	0.233	1
114760	39147	1	3	0.286	5
80571	42843	1	2	0.135	2
248207	2064	3	1	0.513	1
6653	28345	1	2	0.330	5

In [273...

```
filename = 'models/model.sav'
pickle.dump(lr, open(filename, 'wb'))
```

6.2. RANDOM FOREST CLASSIFIER

In [266...

```
#define model
rf = RandomForestClassifier(n_jobs = -1)

#training model
rf.fit(X_train_n, y_train)

#predict model
yhat_rf = rf.predict(X_val_n)

# AUC-ROC
rf_cv = cross_val_predict(rf, X_train, y_train, cv=5)
rf_roc = roc_auc_score(y_train, rf_cv)

#performance
result_rf = pd.DataFrame(["Random Forest Classifier",accuracy_score(y_test,yhat_rf),cohen_kappa_score(y_test,yhat_rf),
                          recall_score(y_test,yhat_rf), f1_score(y_test,yhat_rf),rf_roc,
                           precision_score(y_test,yhat_rf)]).T
result_rf.columns = ["Model",'Accuracy',"Kappa Score","Recall","F1-Score","roc_auc_score","Precision_score"]

#classification_report
print(classification_report(y_test, yhat_rf))
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
              precision    recall  f1-score   support
```

```
           0       0.60      0.60      0.60     38282
           1       0.60      0.60      0.60     38207

    accuracy                           0.60     76489
   macro avg       0.60      0.60      0.60     76489
weighted avg       0.60      0.60      0.60     76489
```

In [267…

```
X_train_n.head()
```

Out[267…

	attr_a	attr_b	scd_a	scd_b
172339	2	2	0.233	1
114760	1	3	0.286	5
80571	1	2	0.135	2
248207	3	1	0.513	1
6653	1	2	0.330	5

In [269…

```
result_rf
```

Out[269…

	Model	Accuracy	Kappa Score	Recall	F1-Score	roc_auc_score	Precision_score
0	Random Forest Classifier	0.601	0.202	0.601	0.601	0.711	0.600

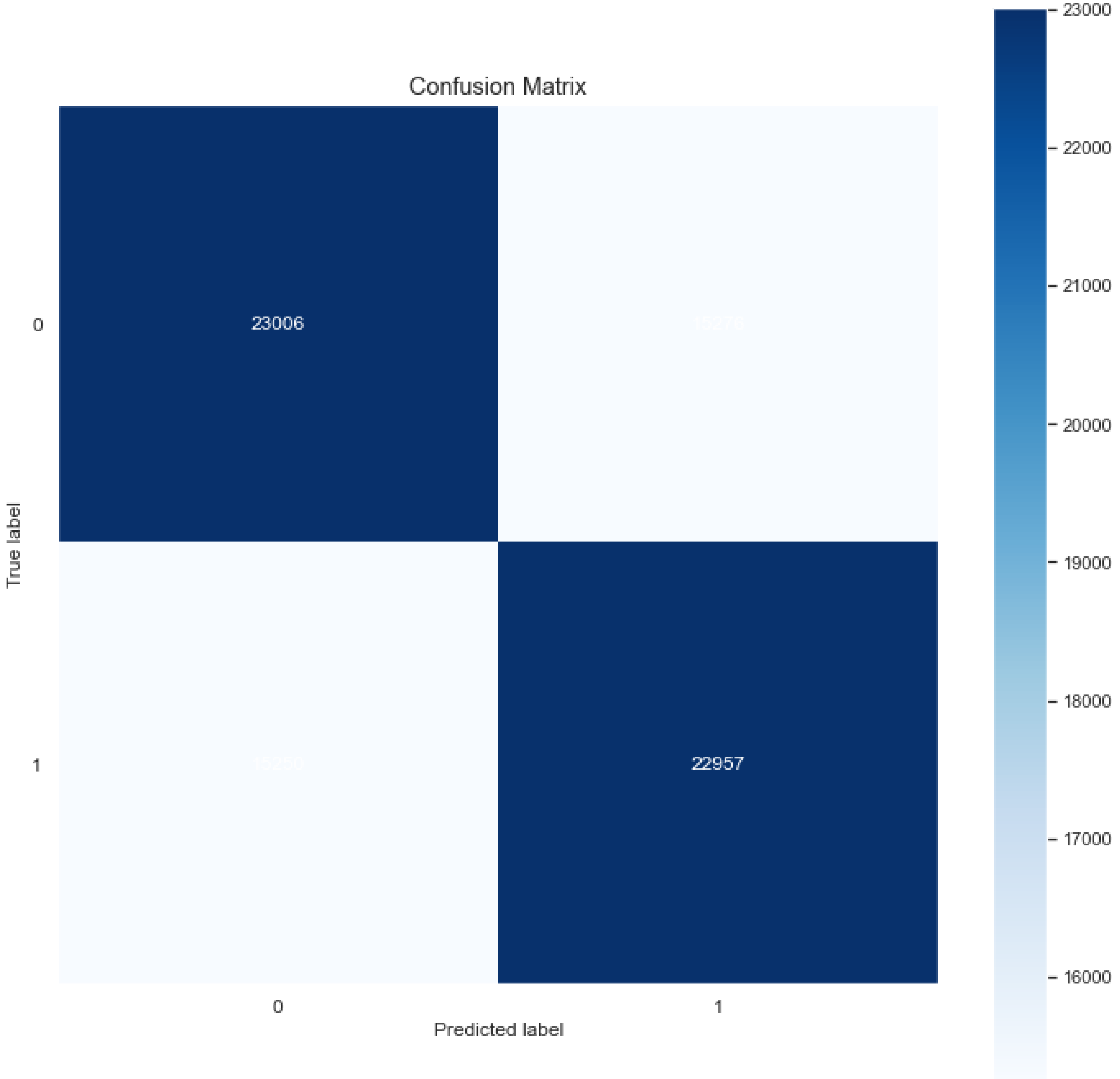
In [270…

```
#confusion_matrix
mt.plot_confusion_matrix(y_test,yhat_rf, normalize = False, figsize = (12,12))
```

Out[270…

```
<AxesSubplot:title={'center':'Confusion Matrix'}, xlabel='Predicted label', ylabel='True label'>
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

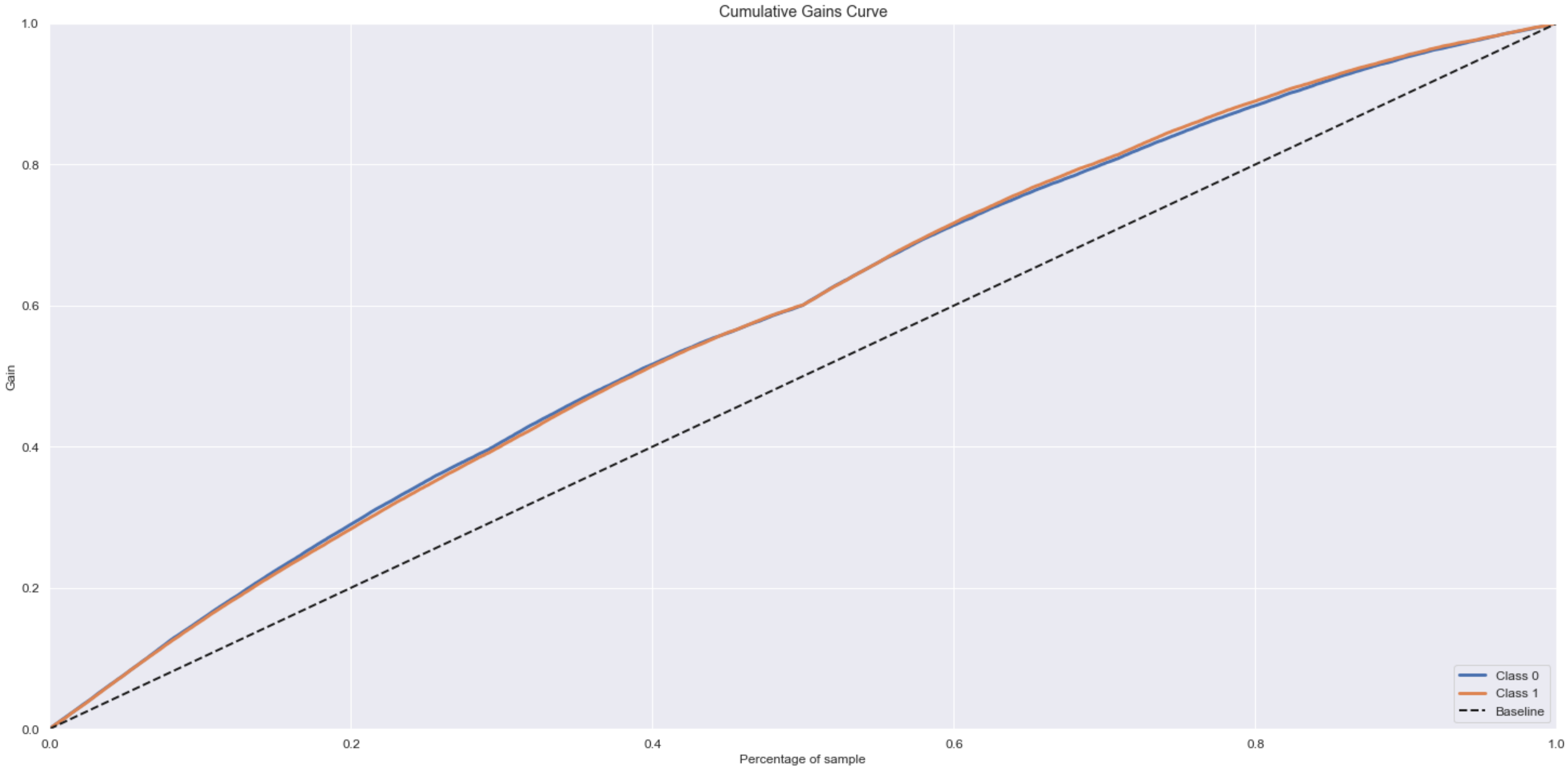


Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [271…

```
#Accumaltive Gain
```

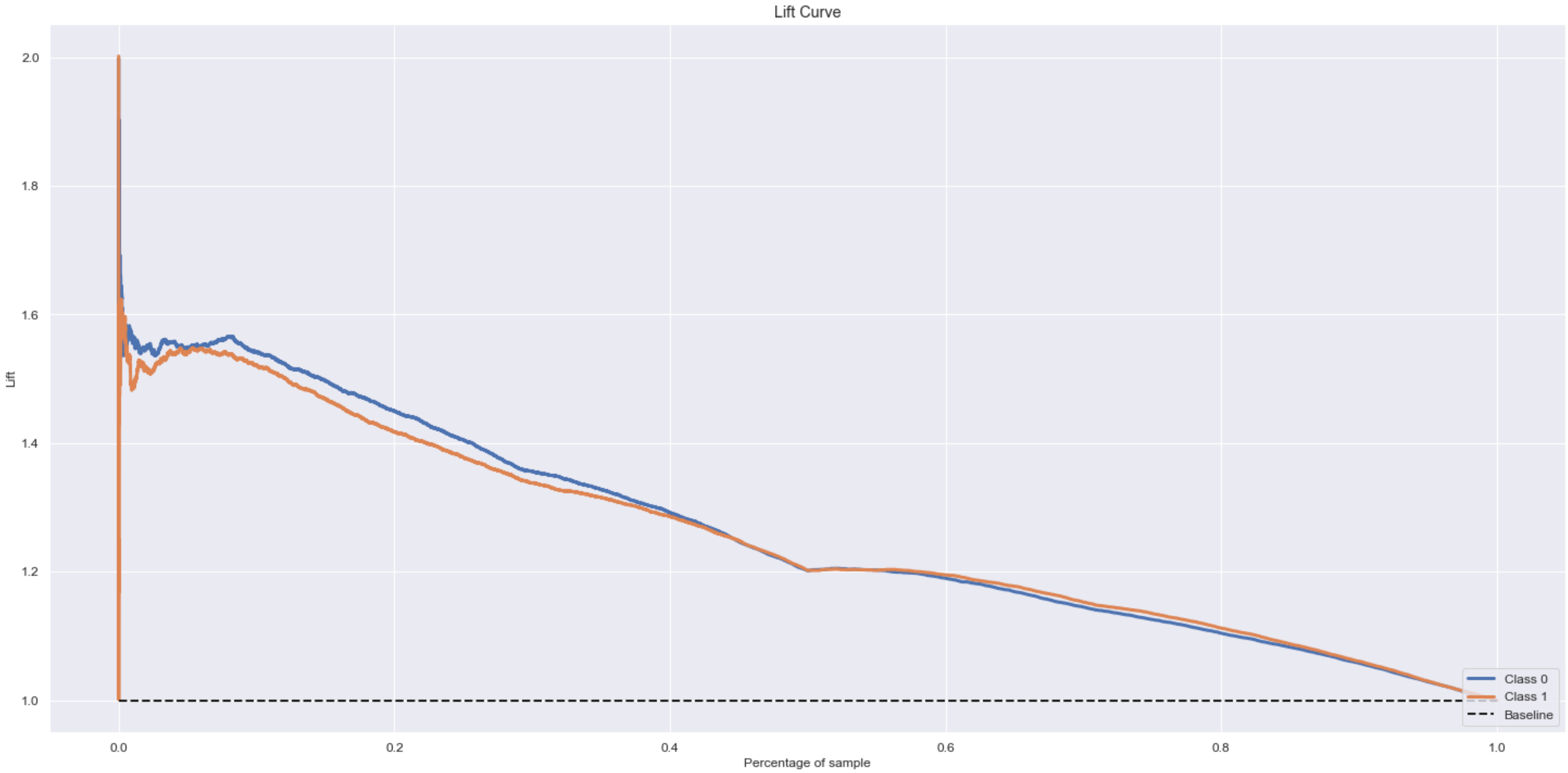
```
## Accumulative Gain
y_hat_rf = rf.predict_proba(X_val_n)

skplt.metrics.plot_cumulative_gain(y_val,y_hat_rf);
```



In [272…

```
#lifc curve
skplt.metrics.plot_lift_curve(y_val,y_hat_rf);
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js



6.3. XGBOOST

In [463…

```
#model definition
xgb_model = xgb.XGBClassifier(n_jobs = -1)

#fit model
xgb_model.fit(X_train_n,y_train)

#prediction
yhat_xgb = xgb_model.predict(X_val_n)
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```python
xgb_cv = cross_val_predict(xgb_model, X_train_n, y_train, cv=5)
xgb_roc = roc_auc_score(y_train, xgb_cv)


#performance
result_xgb = pd.DataFrame(["XGBoost Classifier",accuracy_score(y_test,yhat_xgb),cohen_kappa_score(y_test,yhat_xgb),
                           recall_score(y_test,yhat_xgb), f1_score(y_test,yhat_xgb),xgb_roc,
                           precision_score(y_test,yhat_xgb)]).T
result_xgb.columns = ["Model",'Accuracy',"Kappa Score","Recall","F1-Score","roc_auc_score","Precision_score"]

#classification_report
print(classification_report(y_test, yhat_xgb))
```

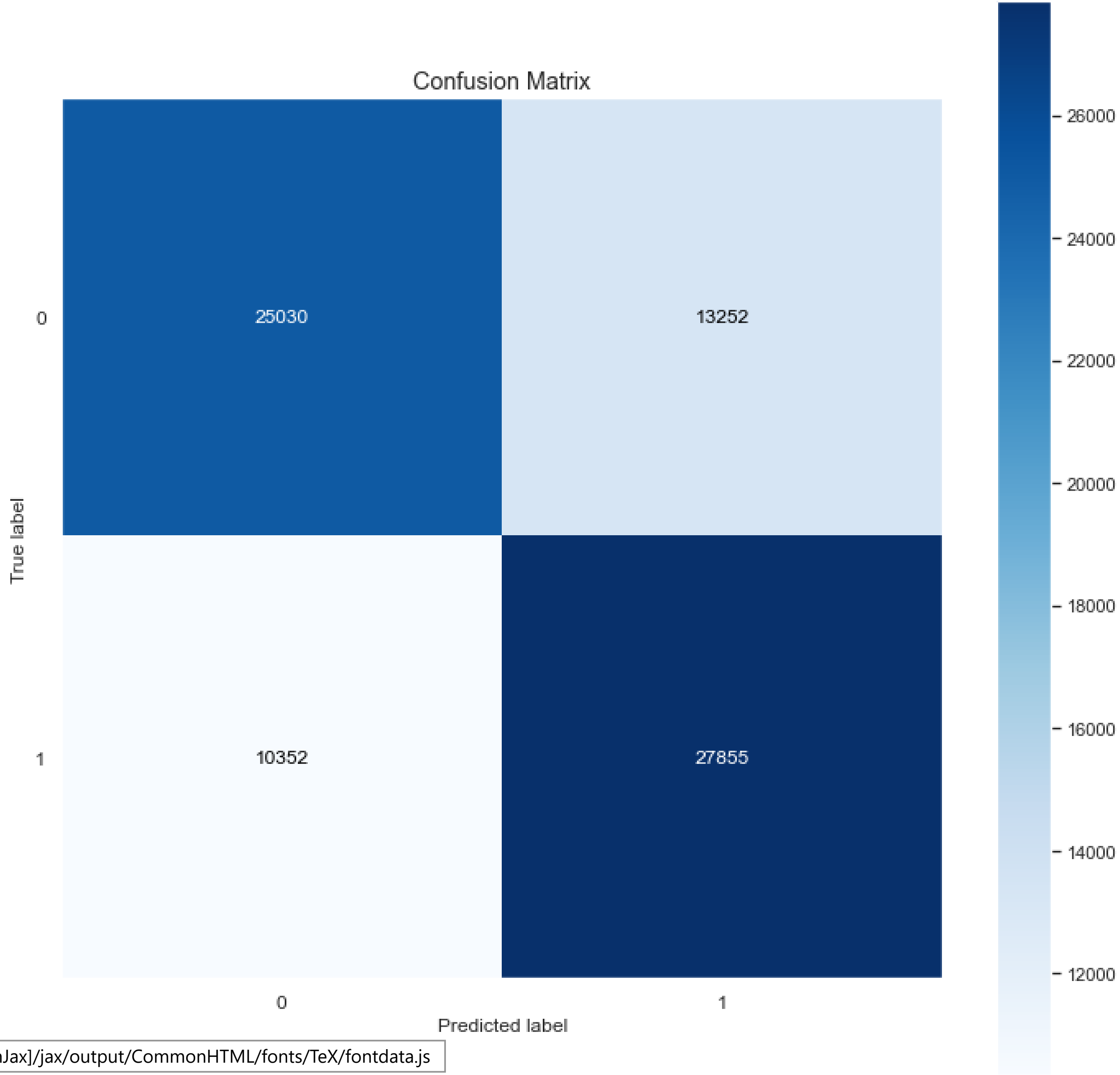
```
[00:14:19] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/learner.cc:1115: Starting in XGBoost 1.3.0, t
he default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_
metric if you'd like to restore the old behavior.
[00:14:25] 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.
[00:14:30] 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.
[00:14:36] 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.
[00:14:42] 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.
[00:14:48] 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.
              precision    recall  f1-score   support

           0       0.71      0.65      0.68     38282
           1       0.68      0.73      0.70     38207

    accuracy                           0.69     76489
   macro avg       0.69      0.69      0.69     76489
weighted avg       0.69      0.69      0.69     76489
```

In [464


```python
mt.plot_confusion_matrix(y_test,yhat_xgb, normalize = False, figsize = (12,12))
```

Out[464…

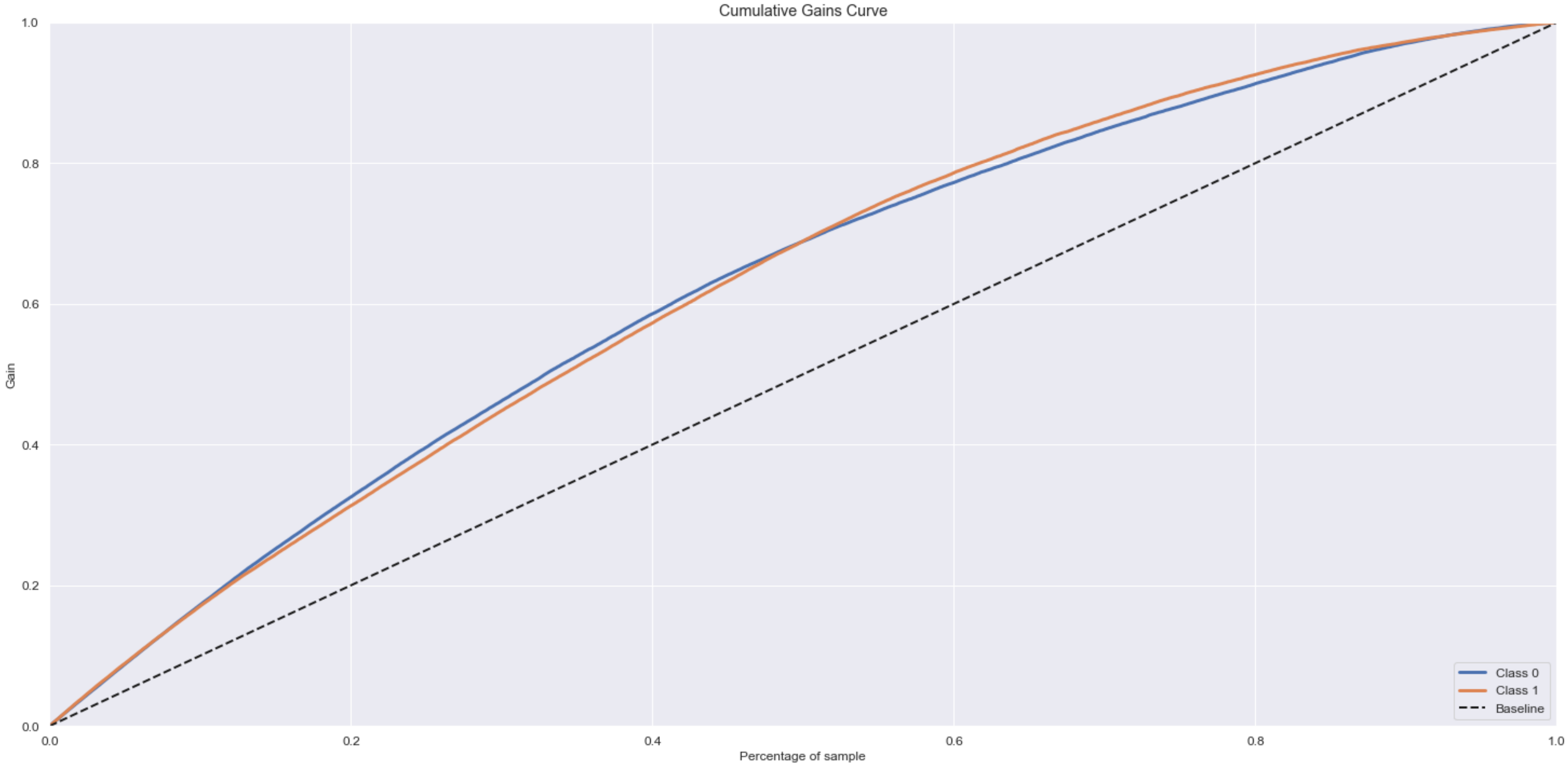
```
<AxesSubplot:title={'center':'Confusion Matrix'}, xlabel='Predicted label', ylabel='True label'>
```



Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [465…

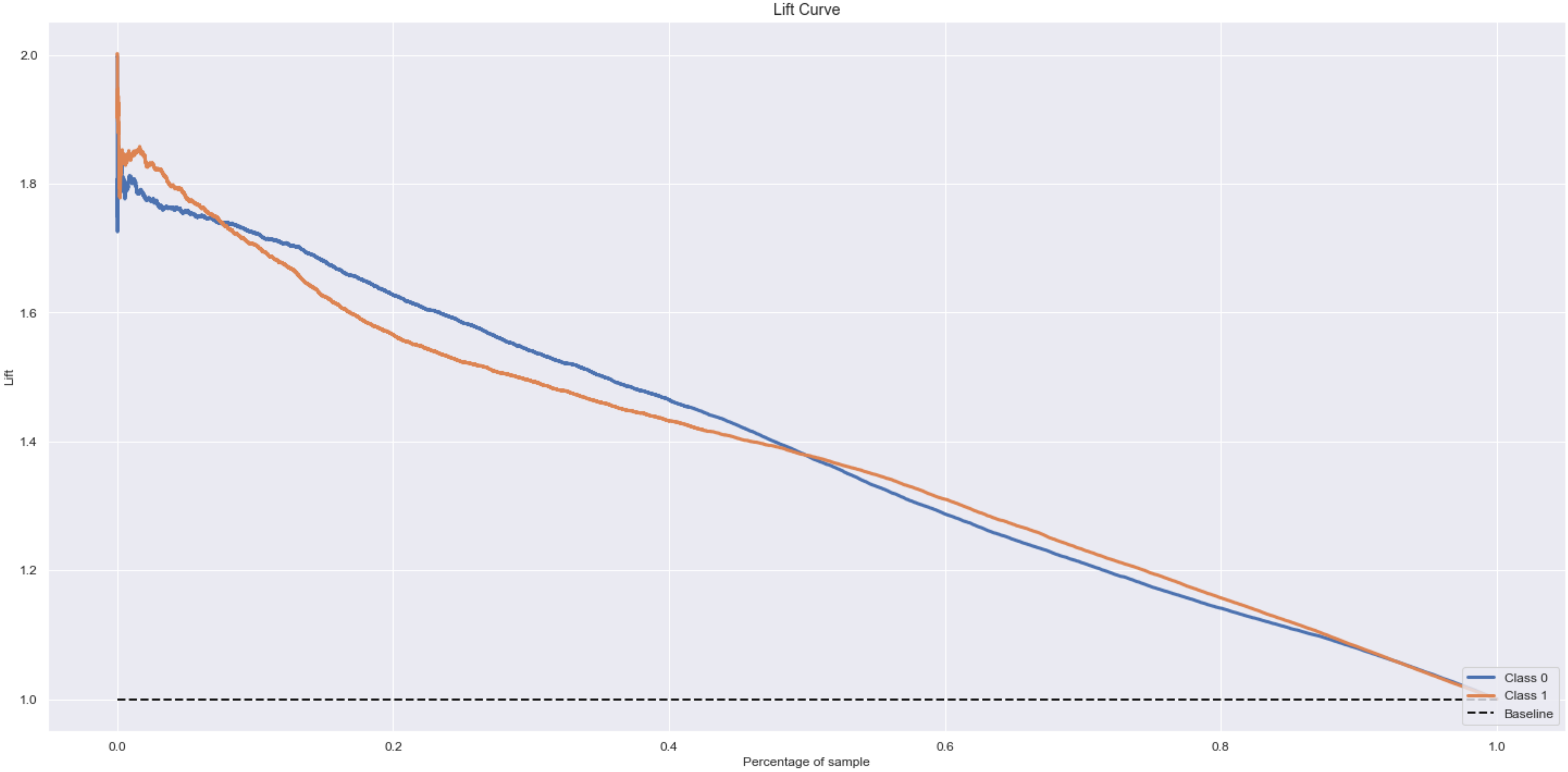
```
#Accumaltive Gain
## Accumulative Gain
y_hat_xgb = xgb_model.predict_proba(X_val_n)

skplt.metrics.plot_cumulative_gain(y_val,y_hat_xgb);
```



In [466…

```
#lifc curve
skplt.metrics.plot_lift_curve(y_val,y_hat_xgb);
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js



6.4. LGBM Classifier

In [467…

```
#define model
model_lgbm = LGBMClassifier(random_state=42, n_jobs=-1)

#training model
model_lgbm.fit(X_train_n, y_train)

#predict model
yhat_lgbm = model_lgbm.predict(X_val_n)
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
lgbm_cv = cross_val_predict(model_lgbm, X_train_n, y_train, cv=5)
lgbm_roc = roc_auc_score(y_train, lgbm_cv)


#performance
result_lgbm = pd.DataFrame(["LGBM Classifier",accuracy_score(y_test,yhat_lgbm),cohen_kappa_score(y_test,yhat_lgbm),
                          recall_score(y_test,yhat_lgbm), f1_score(y_test,yhat_lgbm),lgbm_roc,
                           precision_score(y_test,yhat_lgbm)]).T
result_lgbm.columns = ["Model",'Accuracy',"Kappa Score","Recall","F1-Score","roc_auc_score","Precision_score"]

#classification_report
print(classification_report(y_test, yhat_lgbm))
```

```
              precision    recall  f1-score   support

           0       0.71      0.65      0.68     38282
           1       0.68      0.73      0.70     38207

    accuracy                           0.69     76489
   macro avg       0.69      0.69      0.69     76489
weighted avg       0.69      0.69      0.69     76489
```

In [468…

```
#result lgbg
result_lgbm
```

Out[468…

	Model	Accuracy	Kappa Score	Recall	F1-Score	roc_auc_score	Precision_score
0	LGBM Classifier	0.692	0.384	0.730	0.703	0.691	0.678

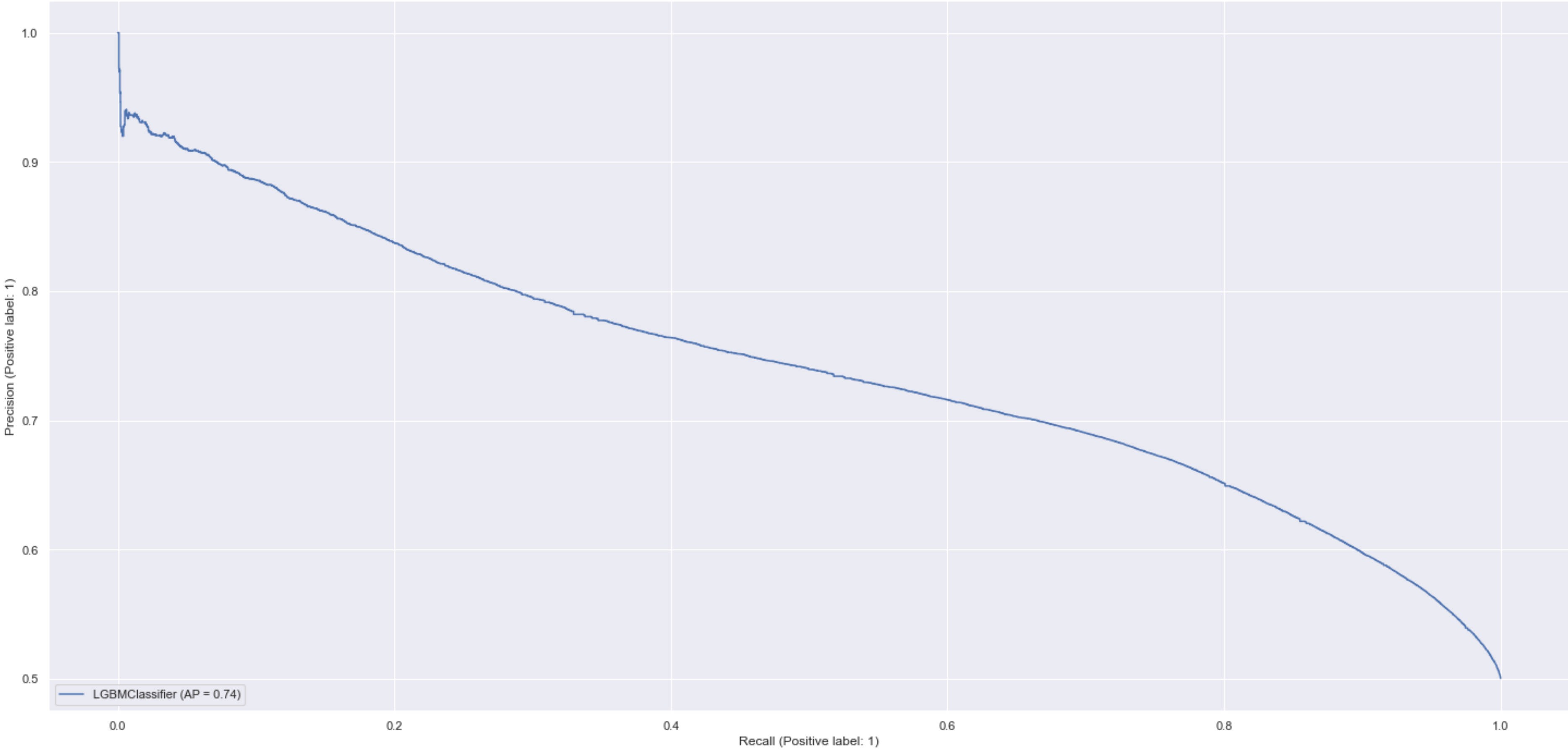
In [471…

```
#precision_recall_curve
plot_precision_recall_curve(model_lgbm, X_train_n, y_train)

mt.plot_confusion_matrix(y_test,yhat_lgbm, normalize = False, figsize = (12,12))
```

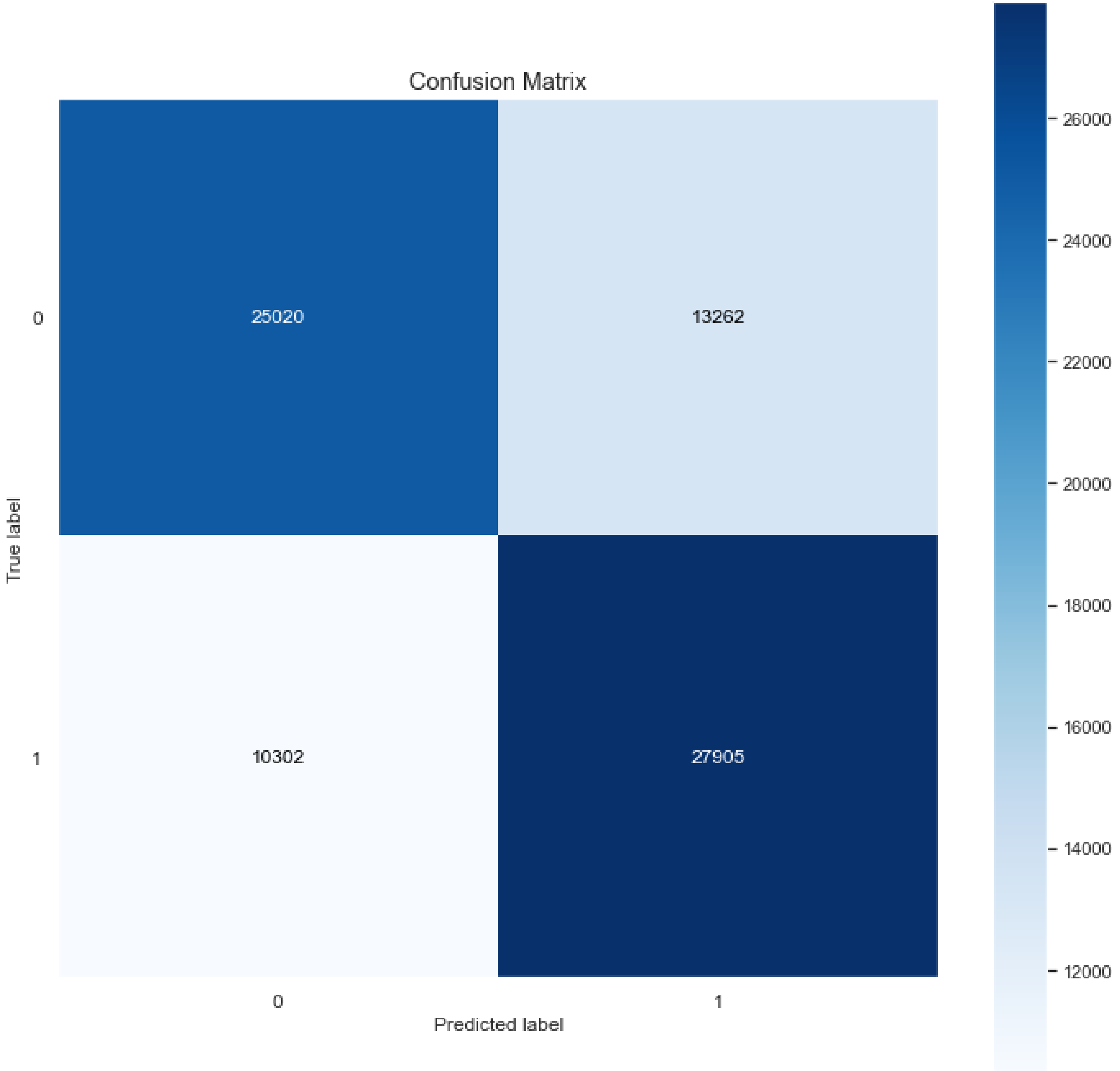
Out[471…

```
<AxesSubplot:title={'center':'Confusion Matrix'}, xlabel='Predicted label', ylabel='True label'>
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js



Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js



Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

6.5. PERFORMANCE METRICS

In [473…

```
#concat all the models as a dataframe
result_model = pd.concat([result_log,result_xgb,result_lgbm,result_rf])
result_model.sort_values("F1-Score", ascending = False)
```

Out[473…

	Model	Accuracy	Kappa Score	Recall	F1-Score	Precision_score	roc_auc_score
0	LGBM Classifier	0.692	0.384	0.730	0.703	0.678	0.691
0	XGBoost Classifier	0.691	0.383	0.729	0.702	0.678	0.690
0	Random Forest Classifier	0.601	0.202	0.601	0.601	0.600	0.711
0	Logistic Regression	0.598	0.195	0.601	0.599	0.597	0.626

The metric we selected was the F1-Score. It is the harmonic mean between recall and precision. Therefore, we will continue with XGBoost

In [485…

```
filename = 'models/model.sav'
pickle.dump(xgb_model, open(filename, 'wb'))
```

7.0. TEST API

In [483…

```
# convert Dataframe to json

data = json.dumps( X_val.drop(columns = ['idx'], axis = 1).sample(3).to_dict( orient='records' ) )
```

In [484…

```
url = 'http://127.0.0.1:5000/'
header = {'Content-Type': 'application/json' }

r = requests.post( url, data=data, headers=header )
print( 'Status Code {}'.format( r.status_code ) )
```

```
Status Code 200
```

In [ ]:

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js