```
!pip install catboost feature_engine
import numpy as np
import pandas as pd
import statsmodels.api as sm
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots
from plotly.offline import init_notebook_mode
from sklearn.preprocessing import LabelEncoder
import warnings, gc
warnings.filterwarnings("ignore")
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, confusion_matrix,
ConfusionMatrixDisplay, accuracy_score, precision_score, recall_score,
f1_score, roc_auc_score, PrecisionRecallDisplay, RocCurveDisplay
from scipy.stats import probplot
from feature_engine.outliers import Winsorizer
from feature_engine.selection import DropConstantFeatures,
DropCorrelatedFeatures, DropDuplicateFeatures
from sklearn.pipeline import Pipeline
from xgboost import XGBClassifier, XGBRFClassifier
from catboost import CatBoostClassifier
from lightgbm import LGBMClassifier
from imblearn.over_sampling import BorderlineSMOTE
from collections import Counter
from yellowbrick.classifier import ClassPredictionError

Requirement already satisfied: catboost in
/usr/local/lib/python3.12/dist-packages (1.2.8)
Requirement already satisfied: feature_engine in
/usr/local/lib/python3.12/dist-packages (1.9.3)
Requirement already satisfied: graphviz in
/usr/local/lib/python3.12/dist-packages (from catboost) (0.21)
Requirement already satisfied: matplotlib in
/usr/local/lib/python3.12/dist-packages (from catboost) (3.10.0)
Requirement already satisfied: numpy<3.0,>=1.16.0 in
/usr/local/lib/python3.12/dist-packages (from catboost) (2.0.2)
Requirement already satisfied: pandas>=0.24 in
/usr/local/lib/python3.12/dist-packages (from catboost) (2.2.2)
Requirement already satisfied: scipy in
/usr/local/lib/python3.12/dist-packages (from catboost) (1.16.2)
Requirement already satisfied: plotly in
/usr/local/lib/python3.12/dist-packages (from catboost) (5.24.1)
Requirement already satisfied: six in /usr/local/lib/python3.12/dist-
packages (from catboost) (1.17.0)
Requirement already satisfied: scikit-learn>=1.4.0 in
/usr/local/lib/python3.12/dist-packages (from feature_engine) (1.6.1)
Requirement already satisfied: statsmodels>=0.11.1 in
```

```
/usr/local/lib/python3.12/dist-packages (from feature_engine) (0.14.5)
Requirement already satisfied: python-dateutil>=2.8.2 in
/usr/local/lib/python3.12/dist-packages (from pandas>=0.24->catboost)
(2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in
/usr/local/lib/python3.12/dist-packages (from pandas>=0.24->catboost)
(2025.2)
Requirement already satisfied: tzdata>=2022.7 in
/usr/local/lib/python3.12/dist-packages (from pandas>=0.24->catboost)
(2025.2)
Requirement already satisfied: joblib>=1.2.0 in
/usr/local/lib/python3.12/dist-packages (from scikit-learn>=1.4.0-
>feature_engine) (1.5.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in
/usr/local/lib/python3.12/dist-packages (from scikit-learn>=1.4.0-
>feature_engine) (3.6.0)
Requirement already satisfied: patsy>=0.5.6 in
/usr/local/lib/python3.12/dist-packages (from statsmodels>=0.11.1-
>feature_engine) (1.0.2)
Requirement already satisfied: packaging>=21.3 in
/usr/local/lib/python3.12/dist-packages (from statsmodels>=0.11.1-
>feature_engine) (25.0)
Requirement already satisfied: contourpy>=1.0.1 in
/usr/local/lib/python3.12/dist-packages (from matplotlib->catboost)
(1.3.3)
Requirement already satisfied: cycler>=0.10 in
/usr/local/lib/python3.12/dist-packages (from matplotlib->catboost)
(0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/usr/local/lib/python3.12/dist-packages (from matplotlib->catboost)
(4.60.1)
Requirement already satisfied: kiwisolver>=1.3.1 in
/usr/local/lib/python3.12/dist-packages (from matplotlib->catboost)
(1.4.9)
Requirement already satisfied: pillow>=8 in
/usr/local/lib/python3.12/dist-packages (from matplotlib->catboost)
(11.3.0)
Requirement already satisfied: pyparsing>=2.3.1 in
/usr/local/lib/python3.12/dist-packages (from matplotlib->catboost)
(3.2.5)
Requirement already satisfied: tenacity>=6.2.0 in
/usr/local/lib/python3.12/dist-packages (from plotly->catboost)
(8.5.0)

data = pd.read_csv("credit_risk_dataset.csv")
df=data.copy()
df.head()

{"summary":"{\n  \"name\": \"df\",\n  \"rows\": 32581,\n  \"fields\":
[\n    {\n      \"column\": \"person_age\",\n      \"properties\": {\n
```

\"dtype\": \"number\",\n   \"std\": 6,\n   \"min\": 20,\n
\"max\": 144,\n   \"num_unique_values\": 58,\n
\"samples\": [\n   22,\n   26,\n   65\
n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"person_income\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 61983,\n   \"min\": 4000,\n
\"max\": 6000000,\n   \"num_unique_values\": 4295,\n
\"samples\": [\n   20800,\n   54417,\n   144000\n
],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   },\n   {\n   \"column\": \"person_home_ownership\",\n
\"properties\": {\n   \"dtype\": \"category\",\n
\"num_unique_values\": 4,\n   \"samples\": [\n   \"OWN\",\
n   \"OTHER\",\n   \"RENT\"\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"person_emp_length\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
4.142630148887972,\n   \"min\": 0.0,\n   \"max\": 123.0,\n
\"num_unique_values\": 36,\n   \"samples\": [\n   30.0,\n
18.0,\n   26.0\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"loan_intent\",\n   \"properties\": {\n   \"dtype\":
\"category\",\n   \"num_unique_values\": 6,\n   \"samples\":
[\n   \"PERSONAL\",\n   \"EDUCATION\",\n
\"DEBTCONSOLIDATION\"\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"loan_grade\",\n   \"properties\": {\n   \"dtype\":
\"category\",\n   \"num_unique_values\": 7,\n   \"samples\":
[\n   \"D\",\n   \"B\",\n   \"F\"\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"loan_amnt\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
6322,\n   \"min\": 500,\n   \"max\": 35000,\n
\"num_unique_values\": 753,\n   \"samples\": [\n   13125,\
n   3675,\n   13800\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"loan_int_rate\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
3.2404594649559195,\n   \"min\": 5.42,\n   \"max\": 23.22,\n
\"num_unique_values\": 348,\n   \"samples\": [\n   10.28,\
n   13.35,\n   18.07\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"loan_status\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
0,\n   \"min\": 0,\n   \"max\": 1,\n
\"num_unique_values\": 2,\n   \"samples\": [\n   0,\n
1\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"loan_percent_income\",\n   \"properties\": {\n   \"dtype\":

```
\"number\",\n        \"std\": 0.10678175634492885,\n        \"min\":
0.0,\n        \"max\": 0.83,\n        \"num_unique_values\": 77,\n
\"samples\": [\n          0.55,\n          0.34\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"cb_person_default_on_file\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 2,\n        \"samples\": [\n          \"N\",\n
\"Y\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"cb_person_cred_hist_length\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 4,\n        \"min\": 2,\n
\"max\": 30,\n        \"num_unique_values\": 29,\n        \"samples\":
[\n          24,\n          25\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"df"}
```

```
df = df.drop_duplicates()
```

```
df = df.dropna()
df.isnull().sum()
```

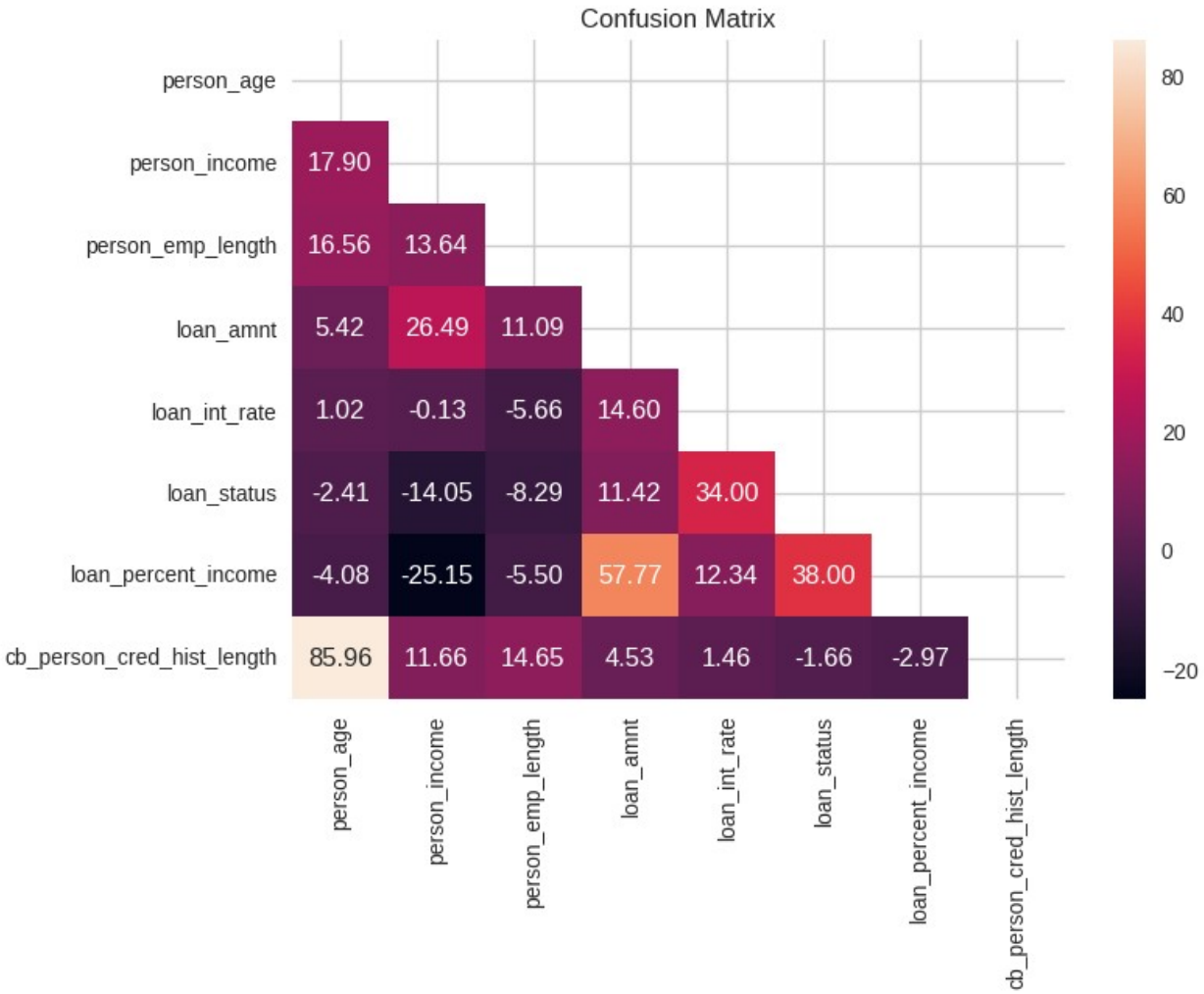
```
person_age                    0
person_income                 0
person_home_ownership         0
person_emp_length             0
loan_intent                   0
loan_grade                    0
loan_amnt                     0
loan_int_rate                 0
loan_status                   0
loan_percent_income           0
cb_person_default_on_file     0
cb_person_cred_hist_length    0
dtype: int64
```

```
def grab_col_names(dataframe, cat_th=10, car_th=20):
    """
    grab_col_names for given dataframe

    :param dataframe:
    :param cat_th:
    :param car_th:
    :return:
    """

    cat_cols = [col for col in dataframe.columns if
dataframe[col].dtypes == "O"]

    num_but_cat = [col for col in dataframe.columns if
dataframe[col].nunique() < cat_th and
```

```
                    dataframe[col].dtypes != "O"]

    cat_but_car = [col for col in dataframe.columns if
dataframe[col].nunique() > car_th and
                    dataframe[col].dtypes == "O"]

    cat_cols = cat_cols + num_but_cat
    cat_cols = [col for col in cat_cols if col not in cat_but_car]

    num_cols = [col for col in dataframe.columns if
dataframe[col].dtypes != "O"]
    num_cols = [col for col in num_cols if col not in num_but_cat]

    print(f"Observations: {dataframe.shape[0]}")
    print(f"Variables: {dataframe.shape[1]}")
    print(f'cat_cols: {len(cat_cols)}')
    print(f'num_cols: {len(num_cols)}')
    print(f'cat_but_car: {len(cat_but_car)}')
    print(f'num_but_cat: {len(num_but_cat)}')
    return cat_cols, cat_but_car, num_cols

cat_cols, cat_but_car, num_cols = grab_col_names(df)
```

```
Observations: 28501
Variables: 12
cat_cols: 5
num_cols: 7
cat_but_car: 0
num_but_cat: 1
```

```
def high_correlated_cols(dataframe, plot=False, corr_th=0.70):
    numeric_dataframe = dataframe.select_dtypes(include=['number'])

    corr = numeric_dataframe.corr()
    cor_matrix = corr.abs()
    upper_triangle_matrix =
cor_matrix.where(np.triu(np.ones(cor_matrix.shape), k=1).astype(bool))
    drop_list = [col for col in upper_triangle_matrix.columns if
any(upper_triangle_matrix[col] > corr_th)]

    if plot:
        mask = np.zeros_like(corr)
        mask[np.triu_indices_from(mask)] = True
        sns.heatmap(corr * 100, annot= True, fmt='.2f', mask=mask)
        plt.title('Confusion Matrix')
        plt.show()

    return drop_list

high_correlated_cols(df, plot=True)
```




```
['cb_person_cred_hist_length']

temp=dict(layout=go.Layout(font=dict(family="Franklin Gothic",
size=12),
                           height=500, width=1000))
target=df.loan_status.value_counts(normalize=True)
target.rename(index={1:'Default',0:'non default'},inplace=True)
pal, color=['#016CC9','#DEB078'], ['#8DBAE2','#EDD3B3']
fig=go.Figure()
fig.add_trace(go.Pie(labels=target.index, values=target*100, hole=.45,
                     showlegend=True,sort=False,

marker=dict(colors=color,line=dict(color=pal,width=2.5)),
                     hovertemplate = "%{label} Accounts: %{value:.2f}
%<extra></extra>"))
fig.update_layout(template=temp, title='Target Distribution',
                  legend=dict(traceorder='reversed',y=1.05,x=0),
                  uniformtext_minsize=15,
```

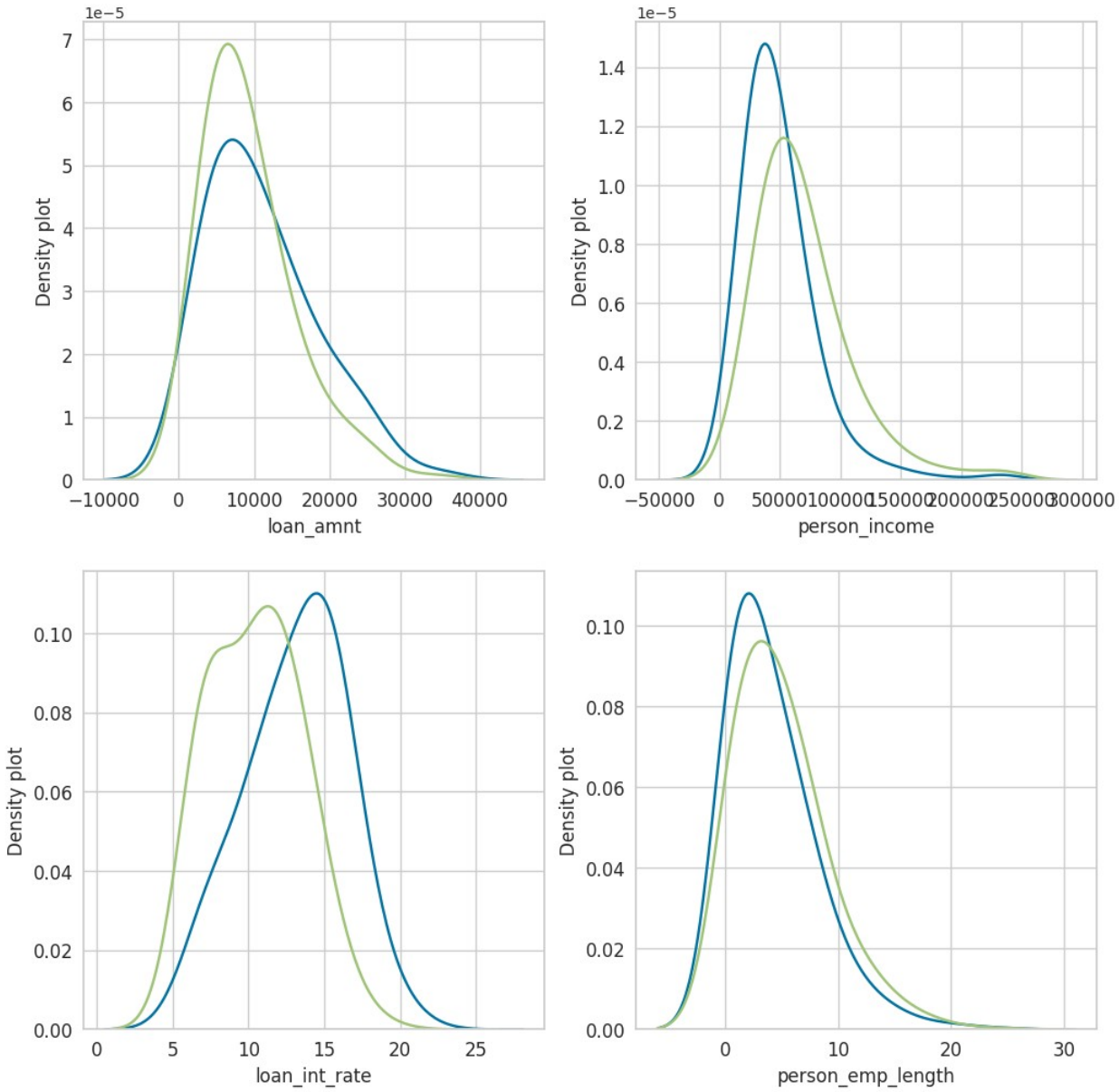
```
uniformtext_mode='hide',width=700)
fig.show()

def outlier_thresholds(dataframe, variable, low_quantile=0.10,
up_quantile=0.90):
    quantile_one = dataframe[variable].quantile(low_quantile)
    quantile_three = dataframe[variable].quantile(up_quantile)
    interquantile_range = quantile_three - quantile_one
    up_limit = quantile_three + 1.5 * interquantile_range
    low_limit = quantile_one - 1.5 * interquantile_range
    return low_limit, up_limit

def check_outlier(dataframe, col_name):
    low_limit, up_limit = outlier_thresholds(dataframe, col_name)
    if dataframe[(dataframe[col_name] > up_limit) |
(dataframe[col_name] < low_limit)].any(axis=None):
        return True
    else:
        return False
for col in num_cols:
    if col != "loan_status":
      print(col, check_outlier(df, col))

person_age True
person_income True
person_emp_length True
loan_amnt False
loan_int_rate False
loan_percent_income True
cb_person_cred_hist_length True

def replace_with_thresholds(dataframe, variable):
    low_limit, up_limit = outlier_thresholds(dataframe, variable)
    dataframe.loc[(dataframe[variable] < low_limit), variable] =
low_limit
    dataframe.loc[(dataframe[variable] > up_limit), variable] =
up_limit


for col in num_cols:
    if col != "loan_status":
        replace_with_thresholds(df,col)

import plotly.tools as tls
import plotly.offline as py
df_good = df.loc[df["loan_status"] == 1]['person_age'].values.tolist()
df_bad = df.loc[df["loan_status"] == 0]['person_age'].values.tolist()
df_age = df['person_age'].values.tolist()

#First plot
trace0 = go.Histogram(
```

```
    x=df_good,
    histnorm='probability',
    name="Loan status = 1"
)
#Second plot
trace1 = go.Histogram(
    x=df_bad,
    histnorm='probability',
    name="Loan status = 0"
)
#Third plot
trace2 = go.Histogram(
    x=df_age,
    histnorm='probability',
    name="Overall Age"
)

#the grid
fig = tls.make_subplots(rows=2, cols=2, specs=[[{}, {}], [{'colspan':
2}, None]],
                          subplot_titles=('Good','Bad', 'General
Distribuition'))

fig.append_trace(trace0, 1, 1)
fig.append_trace(trace1, 1, 2)
fig.append_trace(trace2, 2, 1)

fig['layout'].update(showlegend=True, title='Age Distribuition',
bargap=0.05)
py.iplot(fig, filename='custom-sized-subplot-with-subplot-titles')

trace0 = go.Bar(
    x = df[df["loan_status"]== 1]
["person_home_ownership"].value_counts().index.values,
    y = df[df["loan_status"]== 1]
["person_home_ownership"].value_counts().values,
    name='Loan status = 1'
)

#Second plot
trace1 = go.Bar(
    x = df[df["loan_status"]== 0]
["person_home_ownership"].value_counts().index.values,
    y = df[df["loan_status"]== 0]
["person_home_ownership"].value_counts().values,
    name="Loan status = 0"
)

data = [trace0, trace1]
```

```
layout = go.Layout(
    title='Housing Distribuition'
)


fig = go.Figure(data=data, layout=layout)

py.iplot(fig, filename='Housing-Grouped')

trace0 = go.Bar(
    x = df[df["loan_status"]== 1]
["loan_grade"].value_counts().index.values,
    y = df[df["loan_status"]== 1]["loan_grade"].value_counts().values,
    name='Loan status = 1'
)

#Second plot
trace1 = go.Bar(
    x = df[df["loan_status"]== 0]
["loan_grade"].value_counts().index.values,
    y = df[df["loan_status"]== 0]["loan_grade"].value_counts().values,
    name="Loan status = 0"
)

data = [trace0, trace1]

layout = go.Layout(
    title='Loan grade'
)


fig = go.Figure(data=data, layout=layout)

py.iplot(fig, filename='Loan grade')

trace0 = go.Bar(
    x = df[df["loan_status"]== 1]
["loan_intent"].value_counts().index.values,
    y = df[df["loan_status"]== 1]
["loan_intent"].value_counts().values,
    name='Loan status = 1'
)

#Second plot
trace1 = go.Bar(
    x = df[df["loan_status"]== 0]
["loan_intent"].value_counts().index.values,
    y = df[df["loan_status"]== 0]
["loan_intent"].value_counts().values,
    name="Loan status = 0"
)
```

```
data = [trace0, trace1]

layout = go.Layout(
    title='Loan intent'
)


fig = go.Figure(data=data, layout=layout)

py.iplot(fig, filename='Loan intent')

def plot_distribution_comp(var,nrow=2):

    i = 0
    t1 = df.loc[df['loan_status'] != 0]
    t0 = df.loc[df['loan_status'] == 0]

    sns.set_style('whitegrid')
    plt.figure()
    fig, ax = plt.subplots(nrow,2,figsize=(12,6*nrow))

    for feature in var:
        i += 1
        plt.subplot(nrow,2,i)
        sns.kdeplot(t1[feature], bw=0.5,label="TARGET = 1")
        sns.kdeplot(t0[feature], bw=0.5,label="TARGET = 0")
        plt.ylabel('Density plot', fontsize=12)
        plt.xlabel(feature, fontsize=12)
        locs, labels = plt.xticks()
        plt.tick_params(axis='both', which='major', labelsize=12)
    plt.show();
var = ['loan_amnt','person_income','loan_int_rate',
'person_emp_length']
plot_distribution_comp(var,nrow=2)
```

```
<Figure size 800x550 with 0 Axes>
```




```
df['income_group'] = pd.cut(df['person_income'],
                            bins=[0, 25000, 50000, 75000, 100000,
float('inf')],
                            labels=['low', 'low-middle', 'middle',
'high-middle', 'high'])

dfx=df.copy()
cat_cols, cat_but_car, num_cols = grab_col_names(dfx)
cat_cols.remove("loan_status")


def one_hot_encoder(dataframe, categorical_cols, drop_first=False):
    dataframe = pd.get_dummies(dataframe, columns=categorical_cols,
```

```
drop_first=drop_first,dtype=int)
    return dataframe

dfx = one_hot_encoder(dfx, cat_cols, drop_first=True)
```

```
Observations: 28501
Variables: 13
cat_cols: 6
num_cols: 7
cat_but_car: 0
num_but_cat: 2
```

```
X = dfx.drop(['loan_status',"person_age","person_income"], axis=1)
y = dfx['loan_status']

pipeline = Pipeline(steps=[
    ('constant',DropConstantFeatures()),
    ('correlated',DropCorrelatedFeatures()),
    ('duplicate',DropDuplicateFeatures())
])

X = pipeline.fit_transform(X)
X.shape
```

```
(28501, 24)
```

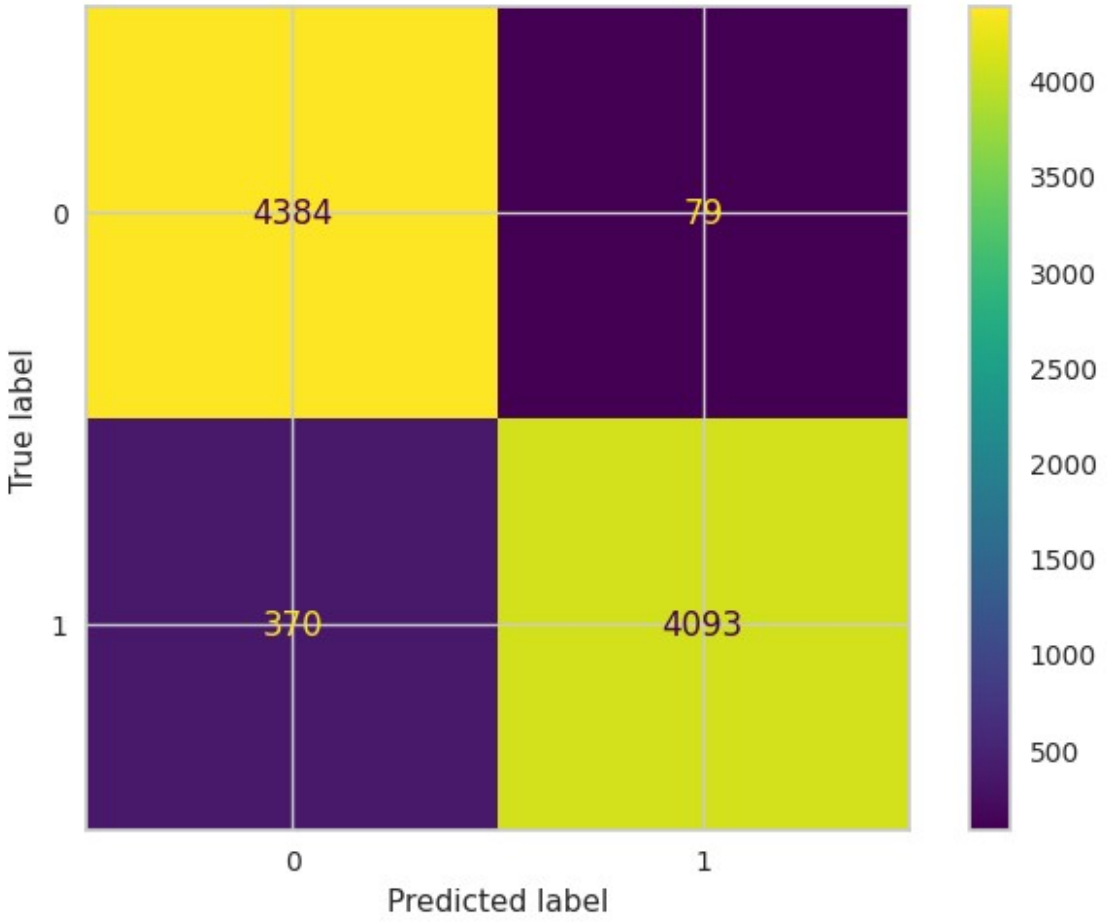
```
smote = BorderlineSMOTE()
X, y = smote.fit_resample(X, y)
print("Final dimensions of target label classes:", Counter(y))
```

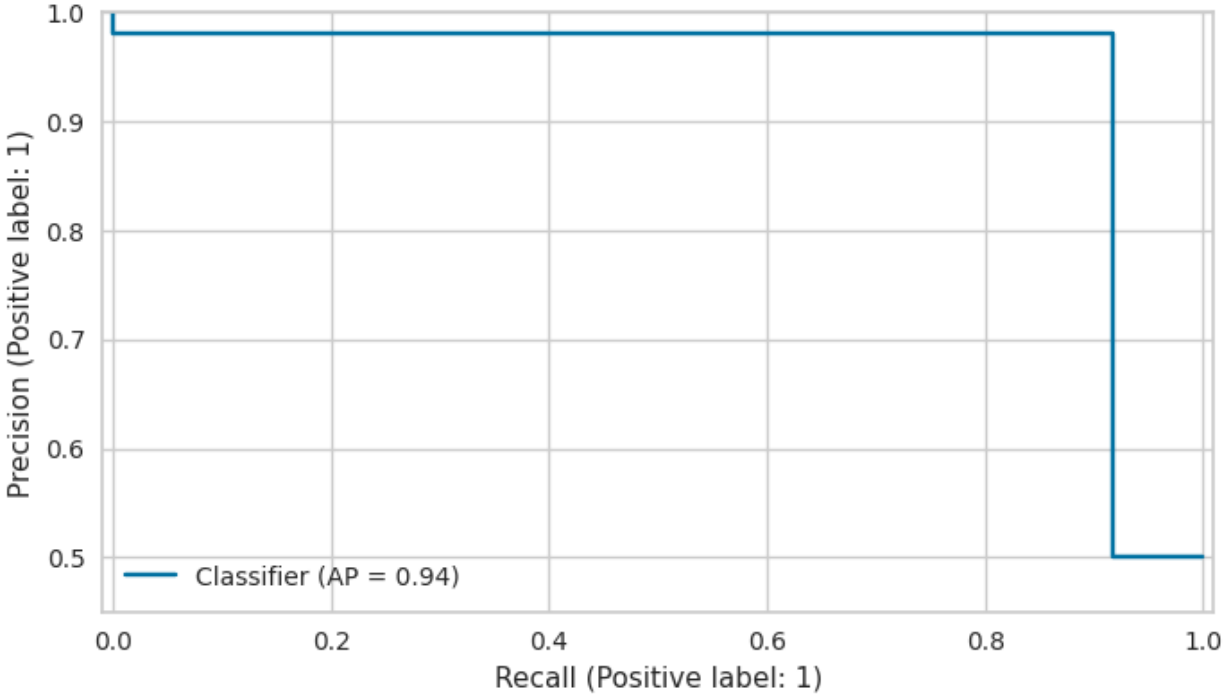
```
Final dimensions of target label classes: Counter({1: 22313, 0:
22313})
```

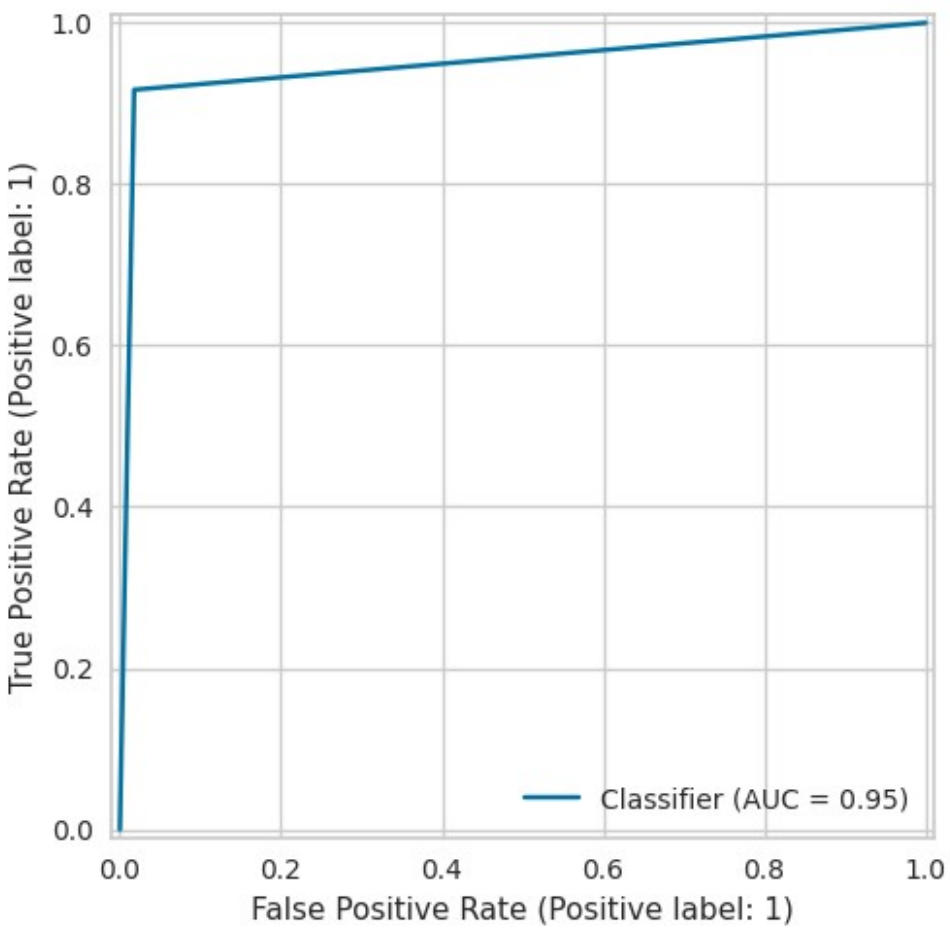
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.20, random_state=42,stratify=y)
#scaling variables
scaler = StandardScaler()
#RobustScaler()
scaled_train_X = scaler.fit_transform(X_train)
scaled_test_X = scaler.transform(X_test)

import re
models = []
accuracy_scores = []
precision_scores = []
recall_scores = []
f1_scores = []
roc_auc_scores = []
```

```
def train_and_evaluate_model(model):
    model.fit(scaled_train_X,y_train)
    y_pred = model.predict(scaled_test_X)
    print("Classification Report:")
    print(classification_report(y_test,y_pred))
    print('-'*50)
    ConfusionMatrixDisplay.from_predictions(y_test,y_pred)
    pr_display =
PrecisionRecallDisplay.from_predictions(y_test,y_pred)
    pr_display.ax_.set_ylim([0.45, 1.0])
    plt.show()
    RocCurveDisplay.from_predictions(y_test,y_pred)
    acc = accuracy_score(y_test,y_pred)
    precision = precision_score(y_test,y_pred,average='macro')
    recall = recall_score(y_test,y_pred,average='macro')
    f1 = f1_score(y_test,y_pred,average='macro')
    roc_auc = roc_auc_score(y_test,y_pred,average='macro')

    if re.search('catboost', str(model)) == None:
        plt.figure(figsize=(6, 4))
        visualizer = ClassPredictionError(model,
orientation='vertical')
        visualizer.score(scaled_test_X, y_test)
        visualizer.show(outpath=None, clear_figure=True)
        del visualizer

    accuracy_scores.append(acc)
    precision_scores.append(precision)
    recall_scores.append(recall)
    f1_scores.append(f1)
    roc_auc_scores.append(roc_auc)
    models.append(model)
    del acc, precision, recall, f1, roc_auc
    gc.collect()
train_and_evaluate_model(CatBoostClassifier(silent=True))
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.98      0.95      4463
           1       0.98      0.92      0.95      4463

    accuracy                           0.95      8926
   macro avg       0.95      0.95      0.95      8926
weighted avg       0.95      0.95      0.95      8926

--------------------------------------------------
```

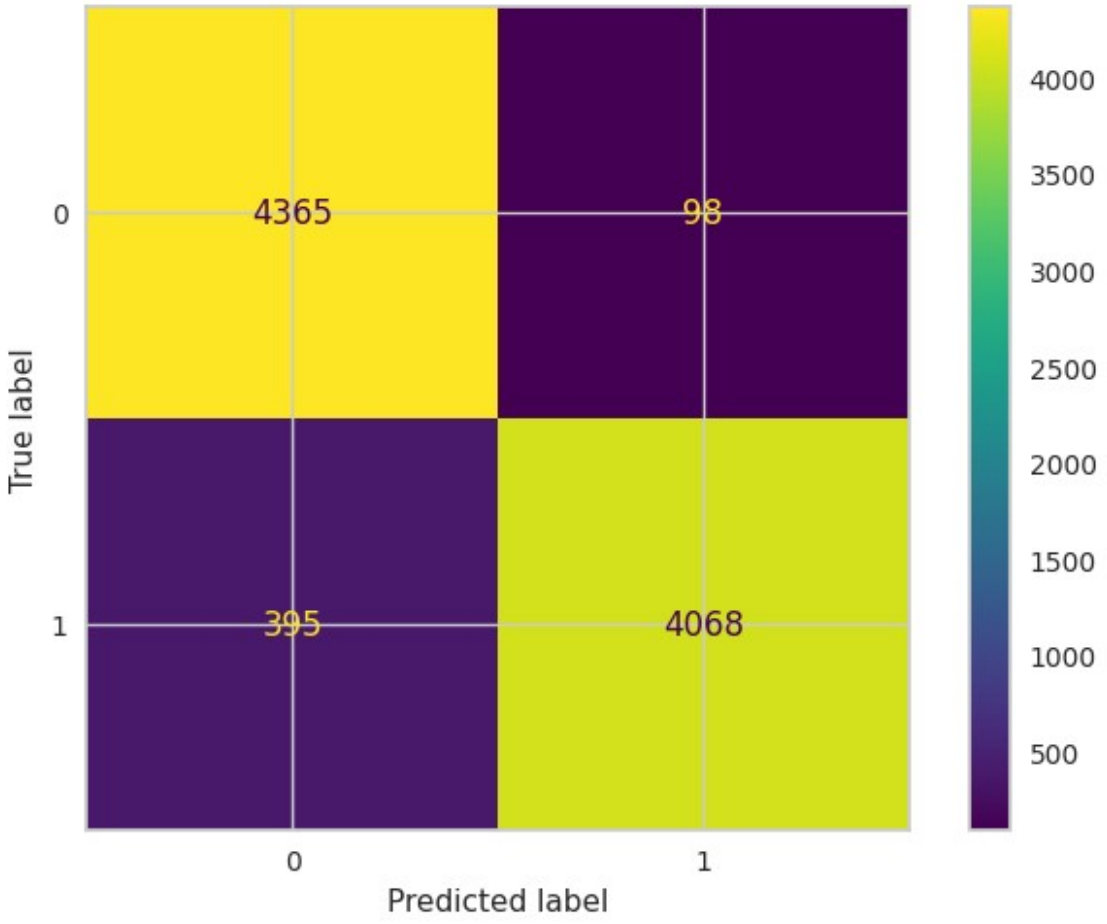

4384
79
370
4093
0
1
True label
0
1
Predicted label
4000
3500
3000
2500
2000
1500
1000
500

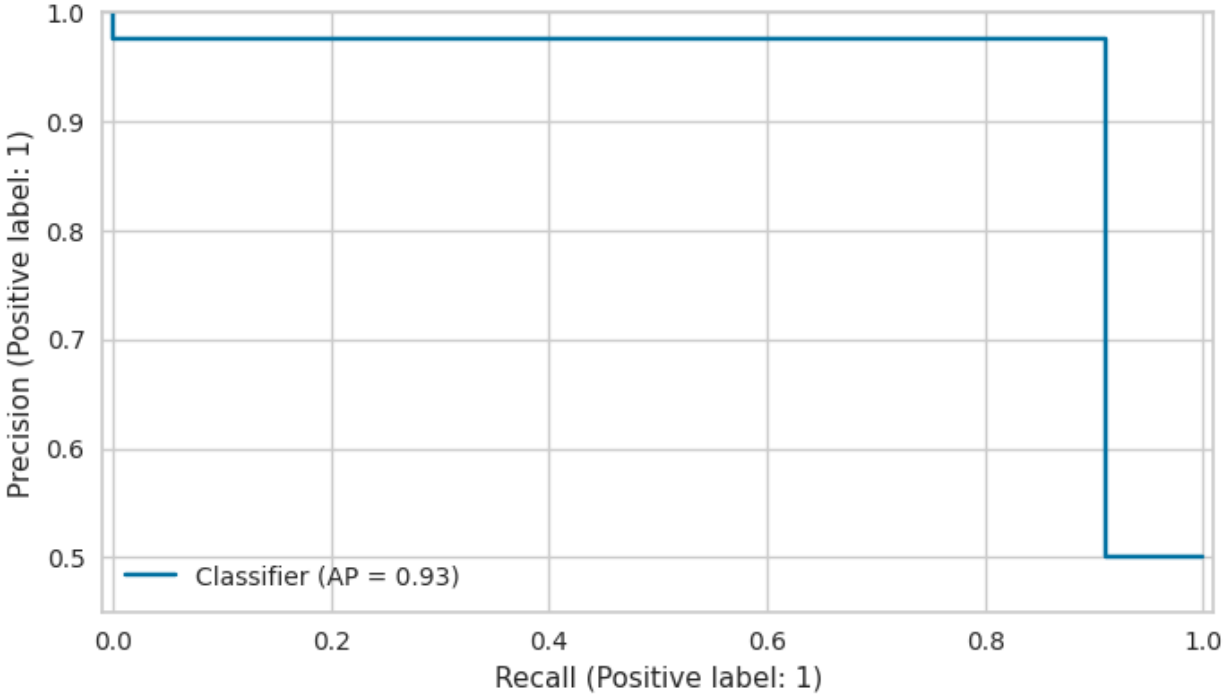

1.0
0.9
0.8
0.7
0.6
0.5
Precision (Positive label: 1)
Classifier (AP = 0.94)
0.0
0.2
0.4
0.6
0.8
1.0
Recall (Positive label: 1)

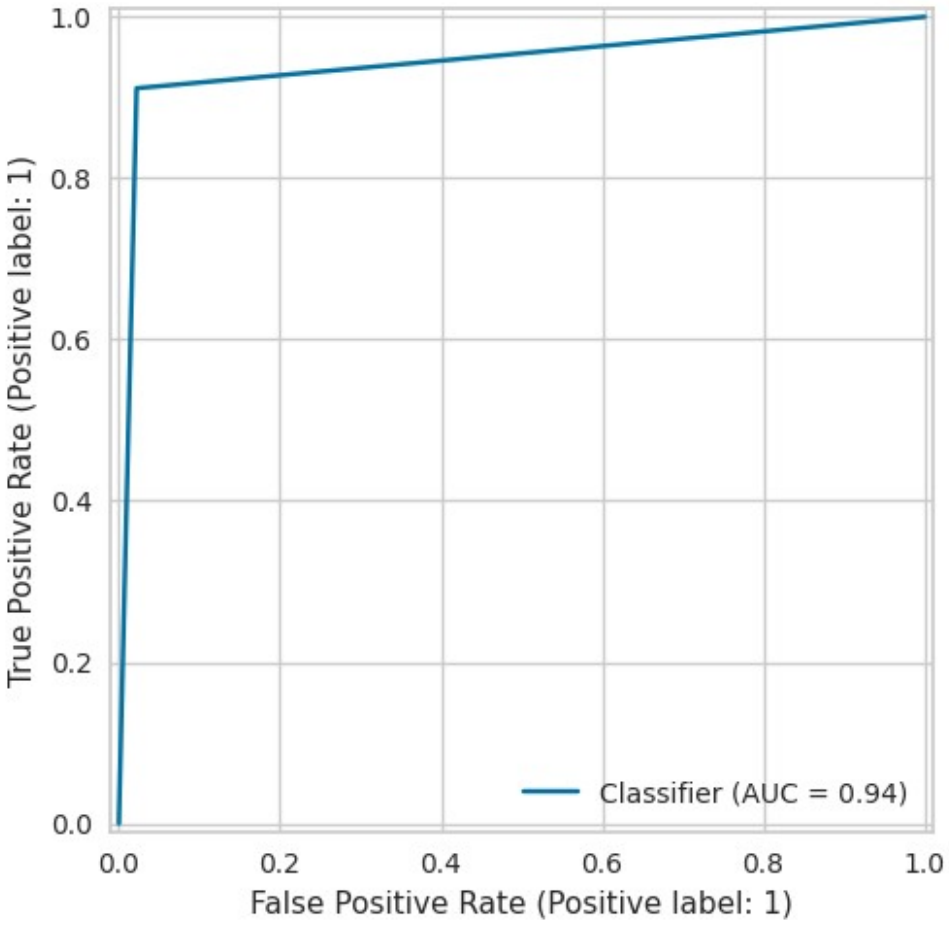


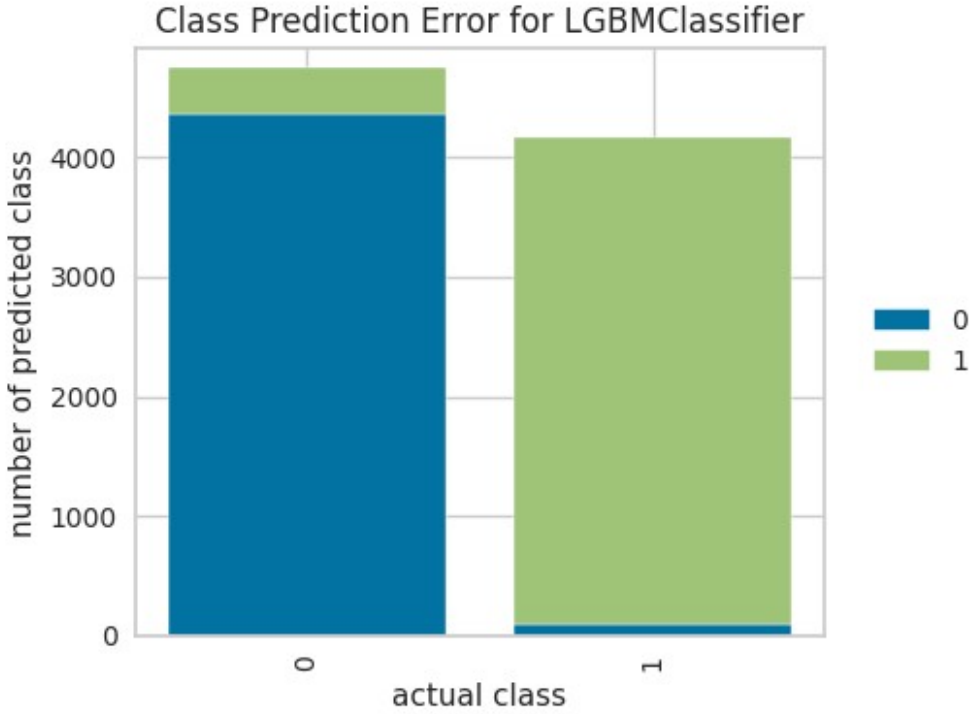

```
train_and_evaluate_model(LGBMClassifier(verbose=-1))
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.98      0.95      4463
           1       0.98      0.91      0.94      4463

    accuracy                           0.94      8926
   macro avg       0.95      0.94      0.94      8926
weighted avg       0.95      0.94      0.94      8926

--------------------------------------------------
```

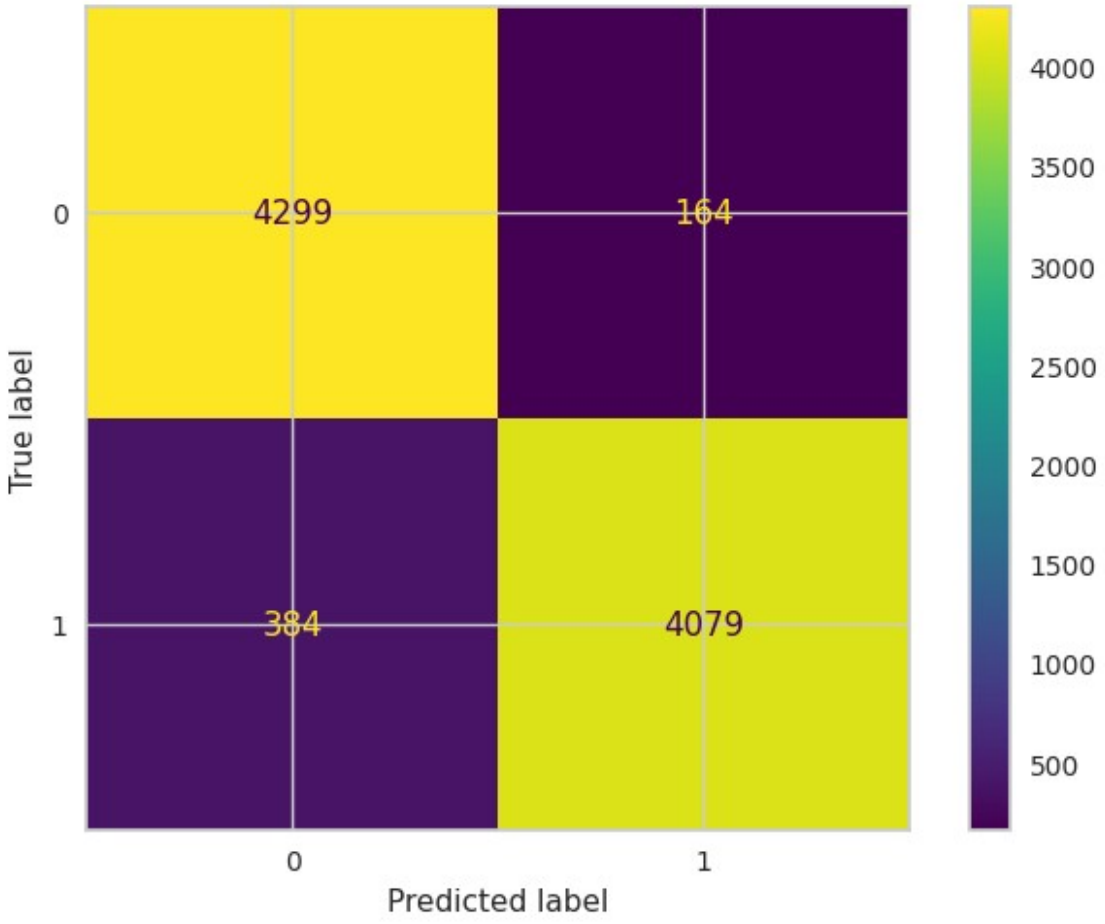

4365
98
395
4068
0
1
True label
0
1
Predicted label
4000
3500
3000
2500
2000
1500
1000
500

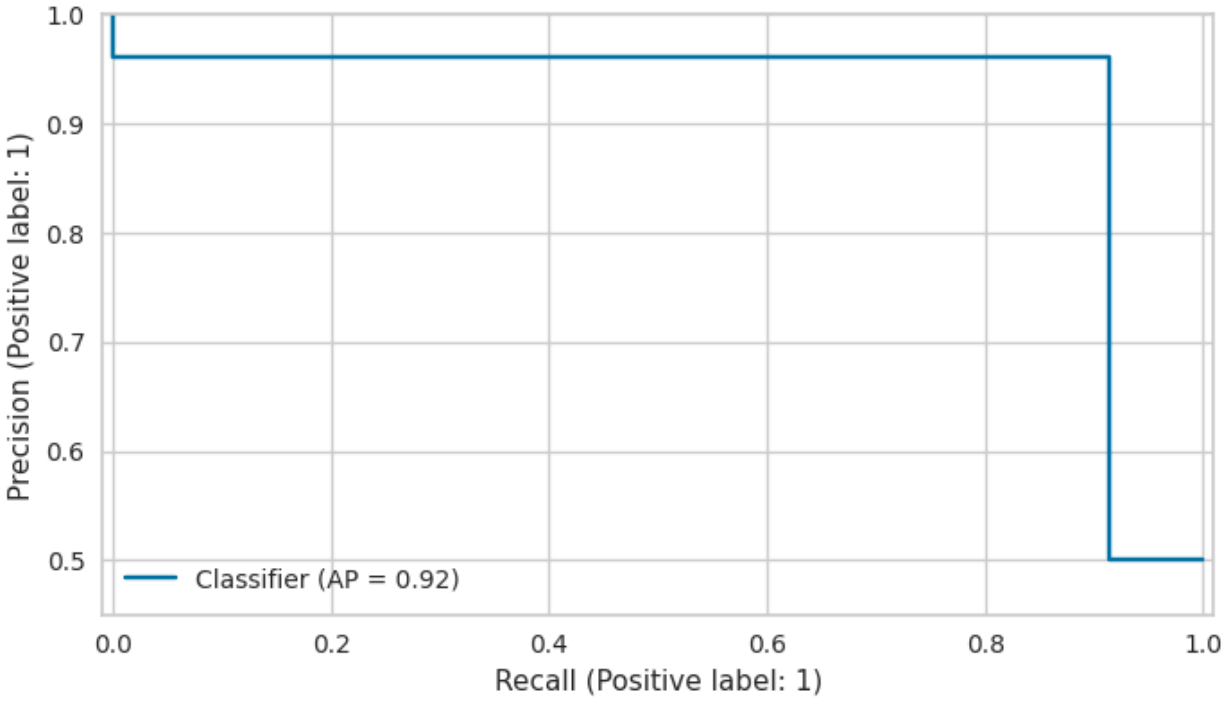

1.0
0.9
0.8
0.7
0.6
0.5
Precision (Positive label: 1)
Classifier (AP = 0.93)
0.0
0.2
0.4
0.6
0.8
1.0
Recall (Positive label: 1)

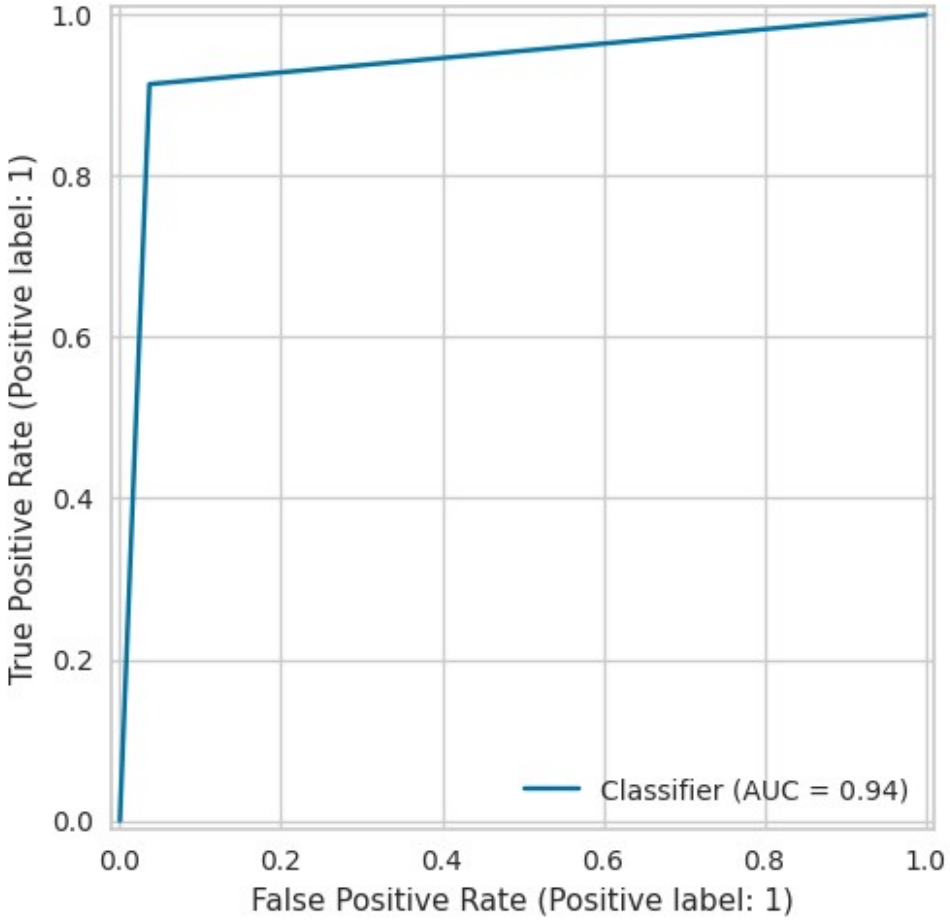

True Positive Rate (Positive label: 1)
False Positive Rate (Positive label: 1)
Classifier (AUC = 0.94)
0.0
0.2
0.4
0.6
0.8
1.0


Class Prediction Error for LGBMClassifier
number of predicted class
actual class
0
1
0
1000
2000
3000
4000

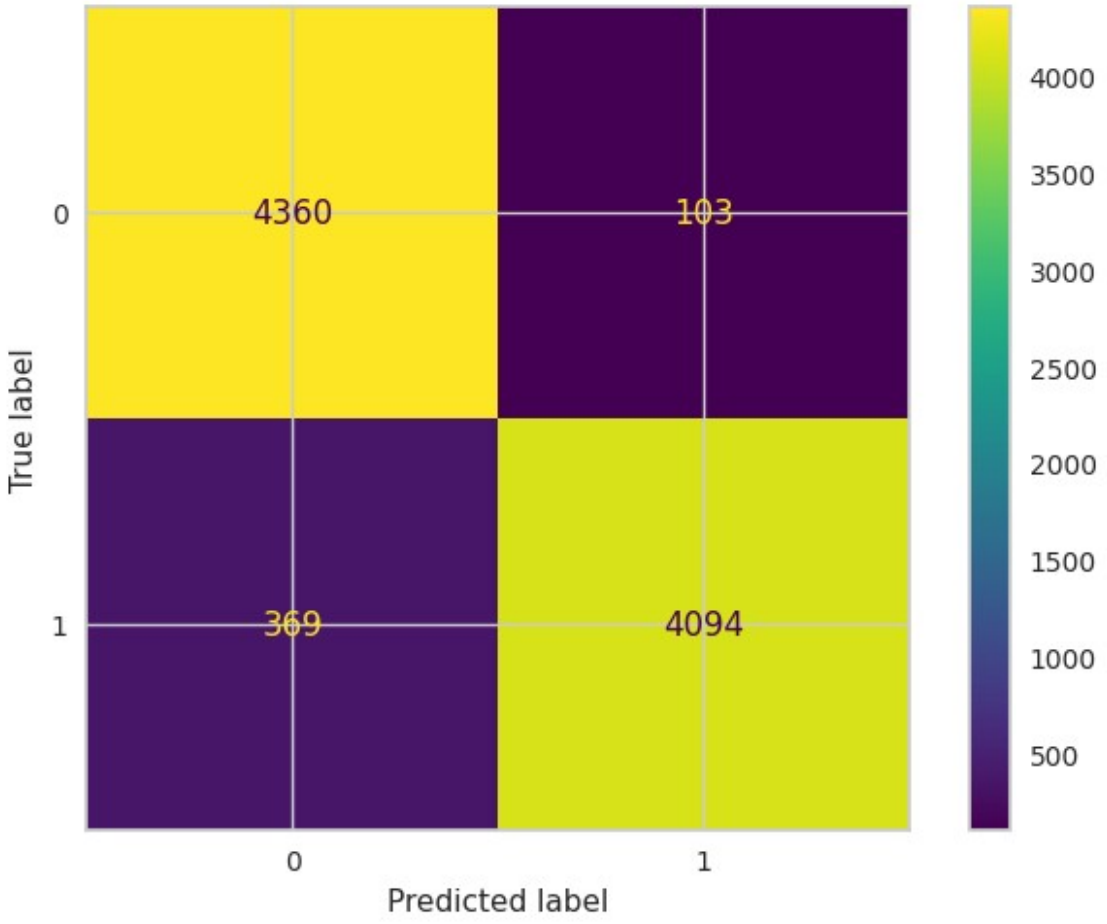
```
from sklearn.ensemble import RandomForestClassifier
train_and_evaluate_model(RandomForestClassifier())

Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.96      0.94      4463
           1       0.96      0.91      0.94      4463

    accuracy                           0.94      8926
   macro avg       0.94      0.94      0.94      8926
weighted avg       0.94      0.94      0.94      8926

--------------------------------------------------
```

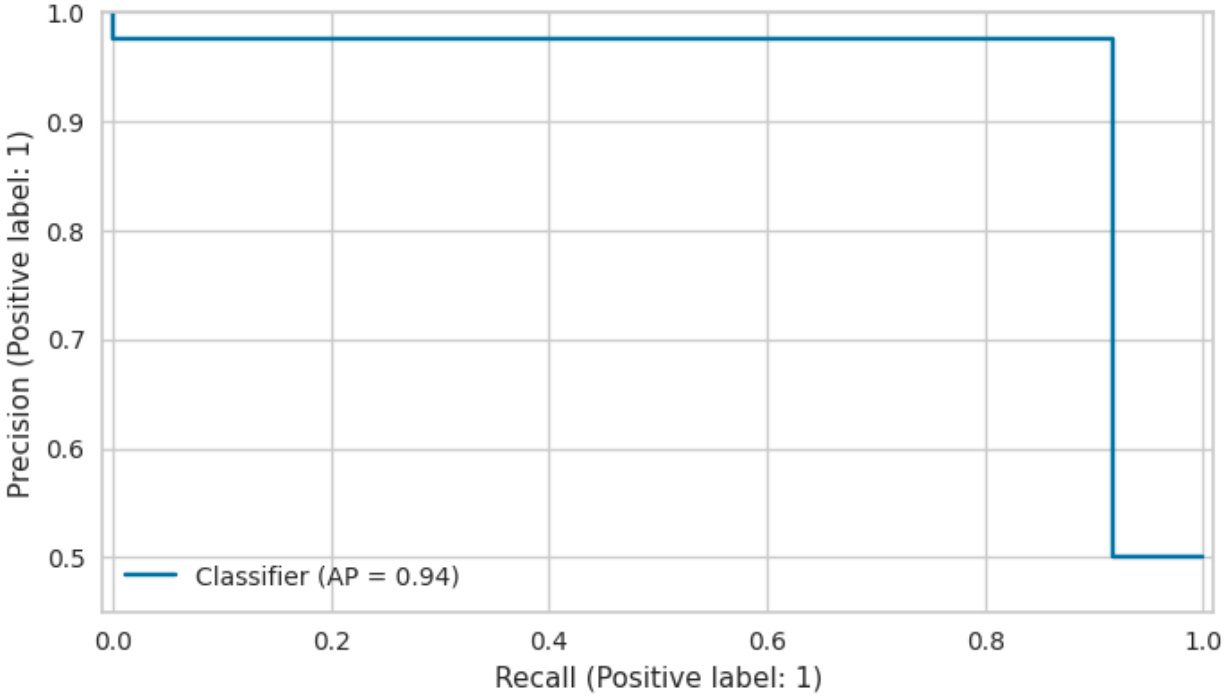


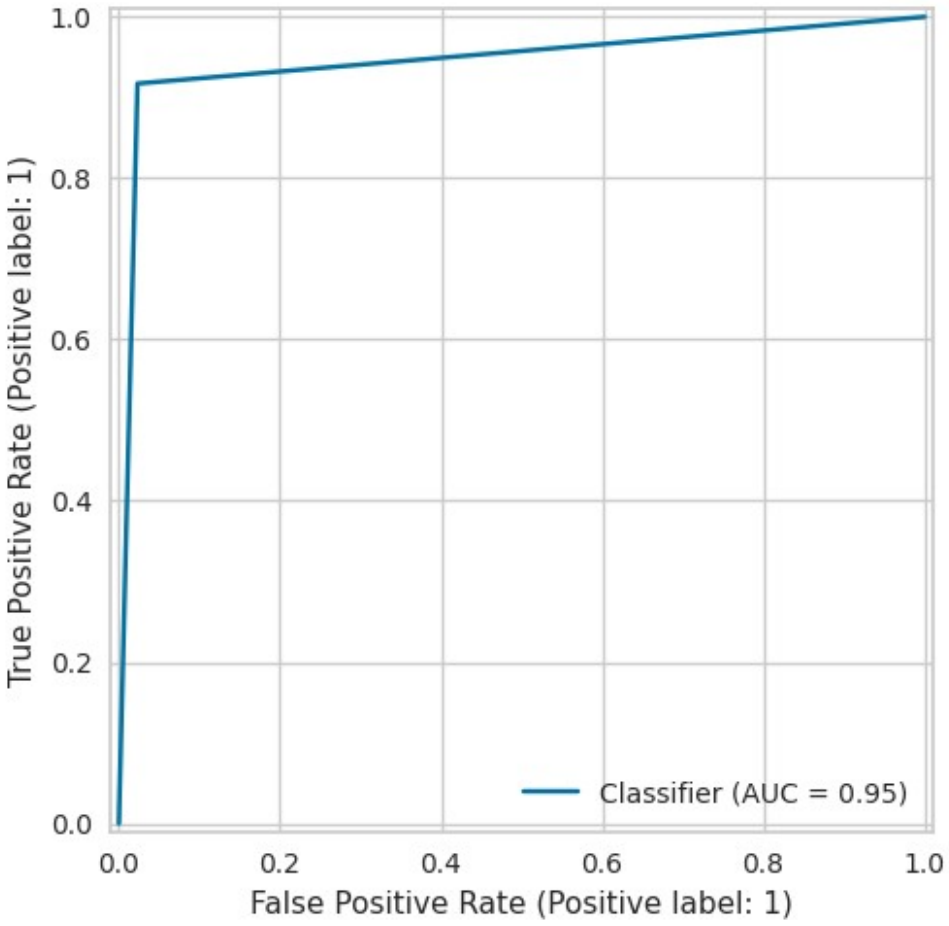

Precision (Positive label: 1)
Recall (Positive label: 1)
Classifier (AP = 0.92)
1.0
0.9
0.8
0.7
0.6
0.5
0.0
0.2
0.4
0.6
0.8
1.0

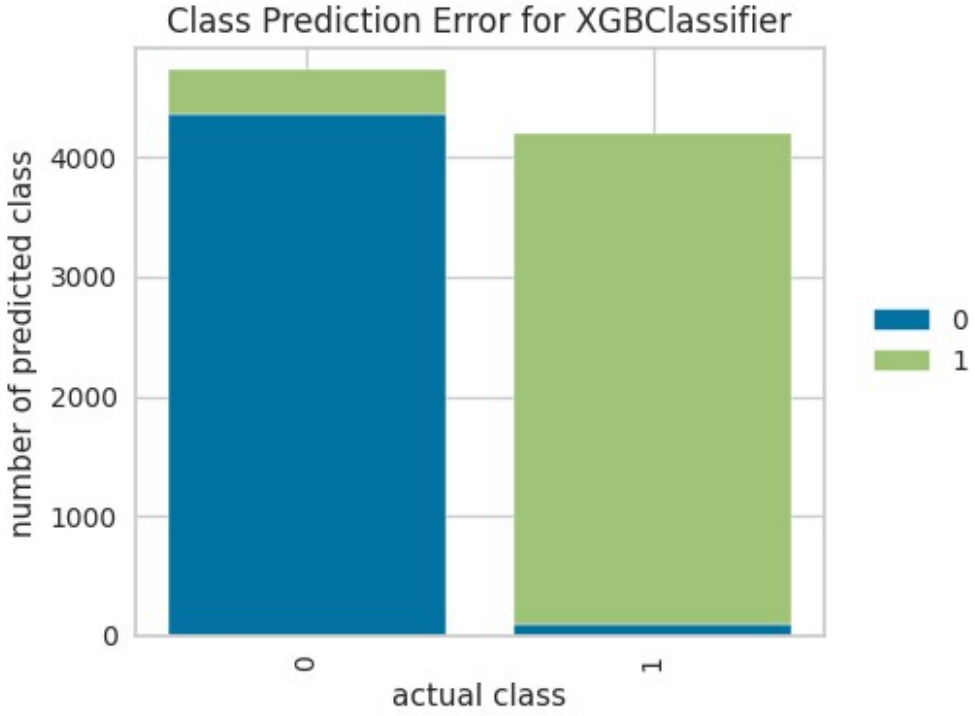

True Positive Rate (Positive label: 1)
False Positive Rate (Positive label: 1)
Classifier (AUC = 0.94)
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0


```
train_and_evaluate_model(XGBClassifier())

Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.98      0.95      4463
           1       0.98      0.92      0.95      4463

    accuracy                           0.95      8926
   macro avg       0.95      0.95      0.95      8926
weighted avg       0.95      0.95      0.95      8926

--------------------------------------------------
```


4360
103
369
4094
0
1
True label
0
1
Predicted label
4000
3500
3000
2500
2000
1500
1000
500

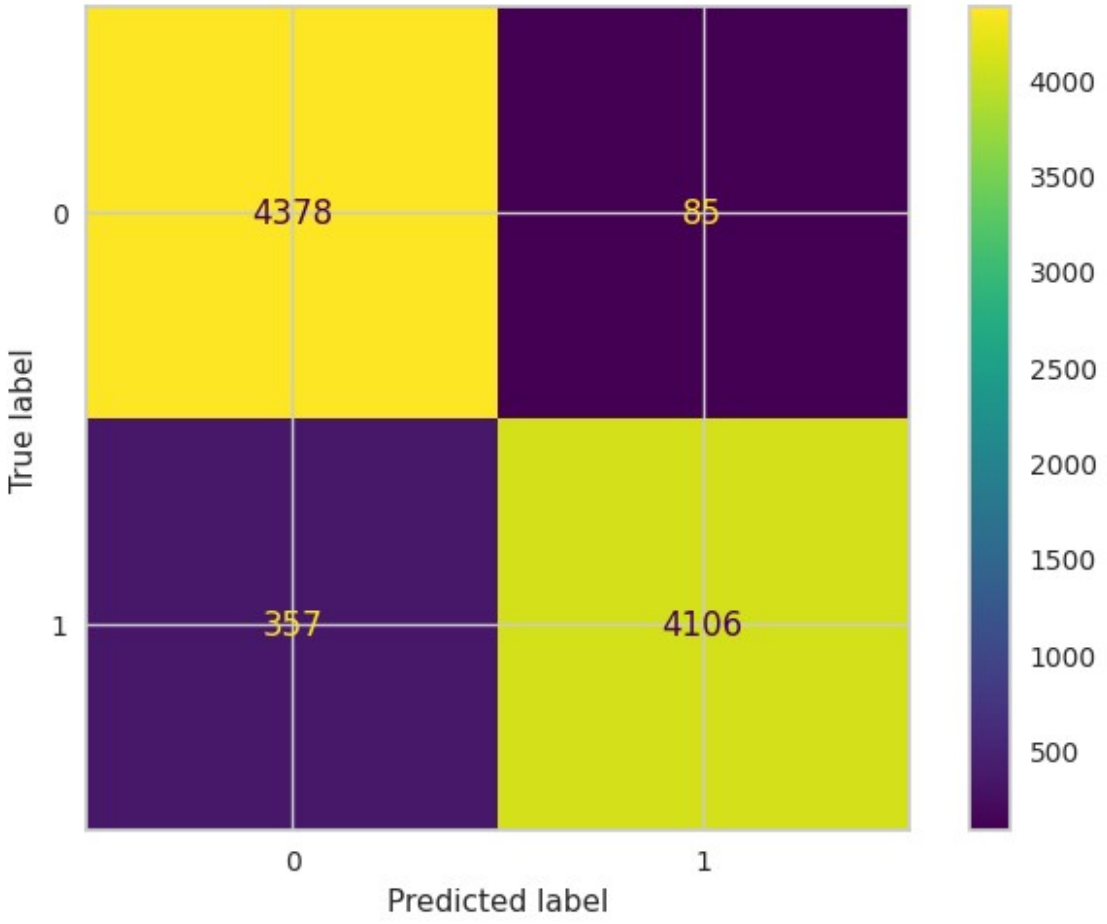

1.0
0.9
0.8
0.7
0.6
0.5
Precision (Positive label: 1)
Classifier (AP = 0.94)
0.0
0.2
0.4
0.6
0.8
1.0
Recall (Positive label: 1)

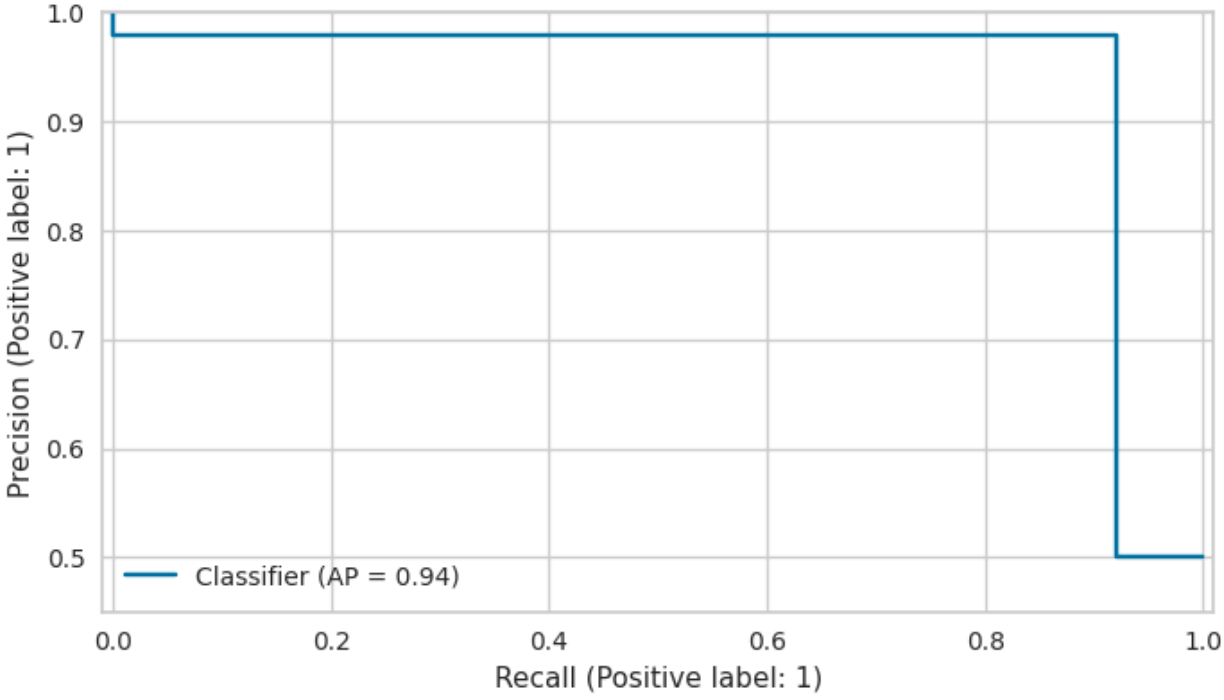

1.0
0.8
0.6
0.4
0.2
0.0
True Positive Rate (Positive label: 1)
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate (Positive label: 1)
Classifier (AUC = 0.95)

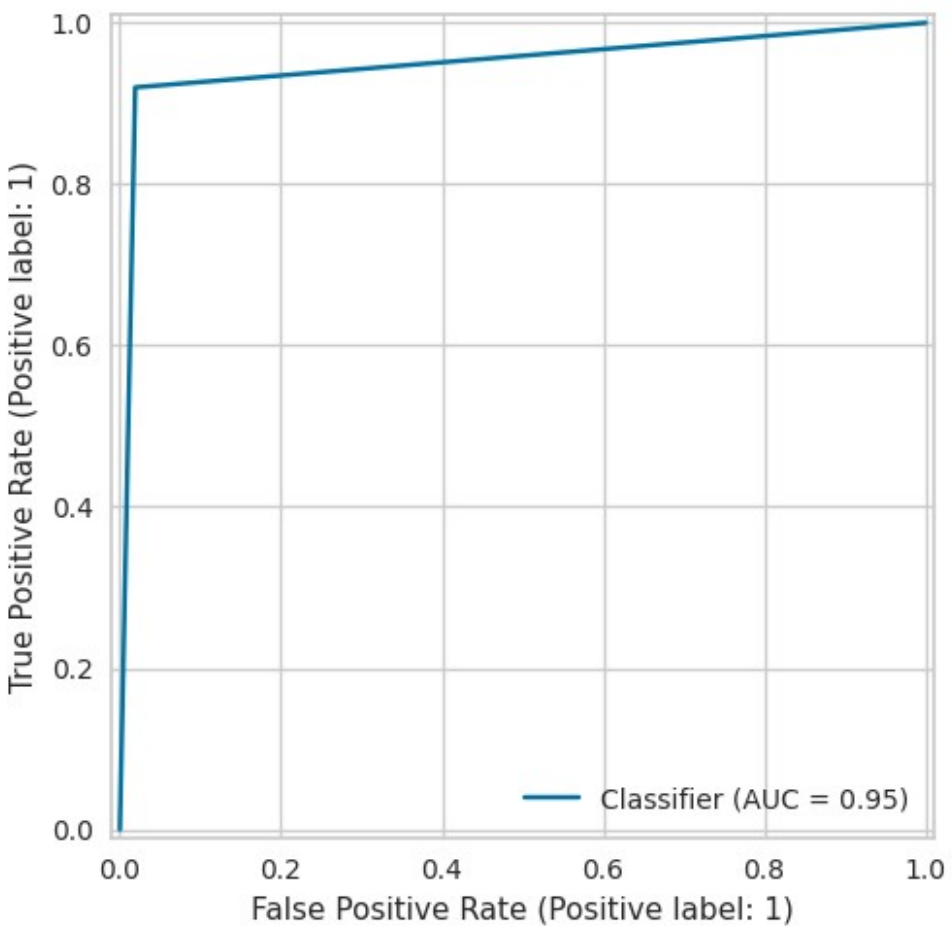

Class Prediction Error for XGBClassifier
4000
3000
2000
1000
0
number of predicted class
0
1
actual class
0
1

```
from sklearn.ensemble import ExtraTreesClassifier, StackingClassifier

train_and_evaluate_model(StackingClassifier(estimators=[
    ('ET',ExtraTreesClassifier()),
    ('XGB',XGBClassifier()),
    ('CAT',CatBoostClassifier(silent=True))
],final_estimator=RandomForestClassifier(),verbose=2))
```

```
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   16.0s finished
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    2.1s finished
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   60.0s finished

Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.98      0.95      4463
           1       0.98      0.92      0.95      4463

    accuracy                           0.95      8926
   macro avg       0.95      0.95      0.95      8926
weighted avg       0.95      0.95      0.95      8926

--------------------------------------------------
```


4378
85
357
4106
0
1
True label
0
1
Predicted label
4000
3500
3000
2500
2000
1500
1000
500


1.0
0.9
0.8
0.7
0.6
0.5
Precision (Positive label: 1)
Classifier (AP = 0.94)
0.0
0.2
0.4
0.6
0.8
1.0
Recall (Positive label: 1)



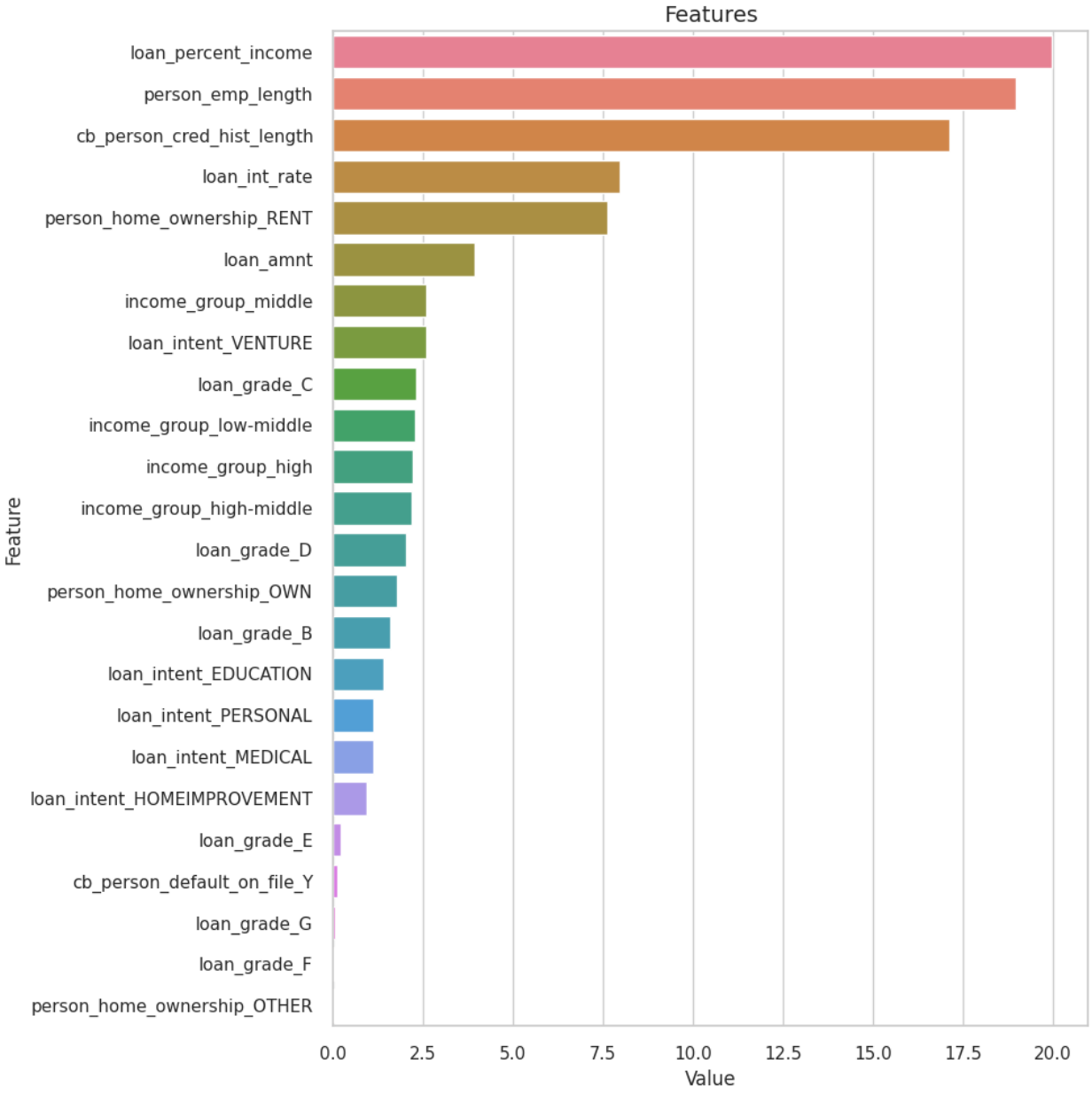

```
model_perfs = pd.DataFrame({'Model': models,
                            'Accuracy': accuracy_scores,
                            'Precision': precision_scores,
                            'Recall': recall_scores,
                            'F1': f1_scores,
                            'ROC-AUC':
roc_auc_scores}).sort_values('Accuracy',ascending=False).reset_index(d
rop=True)
model_perfs
```


```
{"summary":"{\n  \"name\": \"model_perfs\",\n  \"rows\": 5,\n
\"fields\": [\n    {\n      \"column\": \"Model\",\n
\"properties\": {\n        \"dtype\": \"object\",\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Accuracy\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
0.004772495604842811,\n        \"min\": 0.9386063186197625,\n
\"max\": 0.9504817387407574,\n        \"num_unique_values\": 5,\n
\"samples\": [\n          0.9496975128837105,\n
0.9386063186197625,\n          0.9471207707819852\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
```

```
n    },\n    {\n      \"column\": \"Precision\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.00503883713048017,\n        \"min\": 0.9396746919937319,\n
\"max\": 0.952161228478484,\n        \"num_unique_values\": 5,\n
\"samples\": [\n          0.9516175206547568,\n
0.9396746919937319,\n          0.948714739651564\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Recall\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
0.004772495604842862,\n        \"min\": 0.9386063186197624,\n
\"max\": 0.9504817387407574,\n        \"num_unique_values\": 5,\n
\"samples\": [\n          0.9496975128837105,\n
0.9386063186197624,\n          0.9471207707819852\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"F1\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0.004768707004805346,\n
\"min\": 0.9385690005495254,\n        \"max\": 0.9504357138359627,\n
\"num_unique_values\": 5,\n        \"samples\": [\n
0.949643991971483,\n          0.9385690005495254,\n
0.9470737683335444\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"ROC-AUC\",\n      \"properties\": {\n        \"dtype\": \"number\",\
n        \"std\": 0.004772495604842807,\n        \"min\":
0.9386063186197625,\n        \"max\": 0.9504817387407573,\n
\"num_unique_values\": 5,\n        \"samples\": [\n
0.9496975128837106,\n          0.9386063186197625,\n
0.9471207707819852\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"model_perfs"}
```


```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
from catboost import CatBoostClassifier

def plot_importance(model, features, num=None, save=False):
    if num is None:
        num = len(features.columns)

    feature_imp = pd.DataFrame({
        "Value": model.feature_importances_,
        "Feature": features.columns
    })

    feature_imp = feature_imp.sort_values(by="Value", ascending=False)
[:num]

    plt.figure(figsize=(10, 10))
    sns.set(font_scale=1)
    sns.set_style("whitegrid")
```

```
    colors = sns.color_palette("husl", n_colors=num)

    # Plot bar chart
    ax = sns.barplot(
        x="Value",
        y="Feature",
        data=feature_imp,
        palette=colors
    )

    plt.title("Features", fontsize=14)
    plt.xlabel("Value")
    plt.ylabel("Feature")
    plt.tight_layout()

    plt.show(block=True)

    # Save
    if save:
        plt.savefig("importances.png", bbox_inches="tight")

model = CatBoostClassifier(silent=True)
model.fit(X, y)

plot_importance(model, X)
```


Features
loan_percent_income
person_emp_length
cb_person_cred_hist_length
loan_int_rate
person_home_ownership_RENT
loan_amnt
income_group_middle
loan_intent_VENTURE
loan_grade_C
income_group_low-middle
income_group_high
income_group_high-middle
loan_grade_D
person_home_ownership_OWN
loan_grade_B
loan_intent_EDUCATION
loan_intent_PERSONAL
loan_intent_MEDICAL
loan_intent_HOMEIMPROVEMENT
loan_grade_E
cb_person_default_on_file_Y
loan_grade_G
loan_grade_F
person_home_ownership_OTHER
Feature
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Value