Recsify Technologies

Machine Learning internship Based on the given financial data create a ML model to predict if the client is high risk or low risk if we were to provide them loan. We need to predict the column Risk_Flag and it contains value 1 if the client is high risk else it will be 0.

Perform all the various steps of machine learning like data exploration, feature engineering and model building.

Submitted By: Anuroop Arya

```
!pip install catboost
```

```
Requirement already satisfied: catboost in
/usr/local/lib/python3.10/dist-packages (1.2.5)
Requirement already satisfied: graphviz in
/usr/local/lib/python3.10/dist-packages (from catboost) (0.20.3)
Requirement already satisfied: matplotlib in
/usr/local/lib/python3.10/dist-packages (from catboost) (3.7.1)
Requirement already satisfied: numpy>=1.16.0 in
/usr/local/lib/python3.10/dist-packages (from catboost) (1.25.2)
Requirement already satisfied: pandas>=0.24 in
/usr/local/lib/python3.10/dist-packages (from catboost) (2.0.3)
Requirement already satisfied: scipy in
/usr/local/lib/python3.10/dist-packages (from catboost) (1.11.4)
Requirement already satisfied: plotly in
/usr/local/lib/python3.10/dist-packages (from catboost) (5.15.0)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-
packages (from catboost) (1.16.0)
Requirement already satisfied: python-dateutil>=2.8.2 in
/usr/local/lib/python3.10/dist-packages (from pandas>=0.24->catboost)
(2.8.2)
Requirement already satisfied: pytz>=2020.1 in
/usr/local/lib/python3.10/dist-packages (from pandas>=0.24->catboost)
(2023.4)
Requirement already satisfied: tzdata>=2022.1 in
/usr/local/lib/python3.10/dist-packages (from pandas>=0.24->catboost)
(2024.1)
Requirement already satisfied: contourpy>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->catboost)
(1.2.1)
Requirement already satisfied: cycler>=0.10 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->catboost)
(0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->catboost)
(4.53.0)
Requirement already satisfied: kiwisolver>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->catboost)
```

```
(1.4.5)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->catboost)
(24.1)
Requirement already satisfied: pillow>=6.2.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->catboost)
(9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->catboost)
(3.1.2)
Requirement already satisfied: tenacity>=6.2.0 in
/usr/local/lib/python3.10/dist-packages (from plotly->catboost)
(8.3.0)
```

Load Data:

```
# This code assumes the data file is named
'loan_approval_dataset.json' and located in the 'input' directory.
# Adjust the file path if necessary.
# dataset =
pd.read_json('/kaggle/input/loan-approval-dataset/loan_approval_datase
t.json')
# df = dataset.copy()
# df.head()

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import json
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from imblearn.over_sampling import SMOTE
from xgboost import XGBClassifier
from sklearn.linear_model import LogisticRegression
from catboost import CatBoostClassifier
from sklearn.decomposition import PCA


# Step 2: Load JSON File
import pandas as pd
df = pd.read_json('/content/loan_approval_dataset.json')
df.head()
```

{"type":"dataframe","variable_name":"df"}

Data Exploration:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 252000 entries, 0 to 251999
Data columns (total 13 columns):
 #   Column              Non-Null Count   Dtype 
---  ------              --------------   ----- 
 0   Id                  252000 non-null  int64 
 1   Income              252000 non-null  int64 
 2   Age                 252000 non-null  int64 
 3   Experience          252000 non-null  int64 
 4   Married/Single      252000 non-null  object
 5   House_Ownership     252000 non-null  object
 6   Car_Ownership       252000 non-null  object
 7   Profession          252000 non-null  object
 8   CITY                252000 non-null  object
 9   STATE               252000 non-null  object
 10  CURRENT_JOB_YRS     252000 non-null  int64 
 11  CURRENT_HOUSE_YRS   252000 non-null  int64 
 12  Risk_Flag           252000 non-null  int64 
dtypes: int64(7), object(6)
memory usage: 26.9+ MB
```

```
df.describe(include='all')
```


```
{"summary":"{\n  \"name\": \"df\",\n  \"rows\": 11,\n  \"fields\": [\n
{\n      \"column\": \"Id\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 90821.71443937485,\n
\"min\": 1.0,\n        \"max\": 252000.0,\n
\"num_unique_values\": 6,\n        \"samples\": [\n
252000.0,\n          126000.5,\n          189000.25\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Income\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
3451618.6532876026,\n        \"min\": 10310.0,\n        \"max\":
9999938.0,\n        \"num_unique_values\": 8,\n        \"samples\": [\
n          4997116.665325397,\n          5000694.5,\n
252000.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Age\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 89079.4450610442,\n        \"min\": 17.063854818338424,\n
\"max\": 252000.0,\n        \"num_unique_values\": 8,\n
\"samples\": [\n          49.95407142857143,\n          50.0,\n
252000.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Experience\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 89092.11674211107,\n        \"min\":
0.0,\n        \"max\": 252000.0,\n        \"num_unique_values\": 8,\n
\"samples\": [\n          10.084436507936507,\n          10.0,\n
252000.0\n        ],\n        \"semantic_type\": \"\",\n
```

```
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Married/Single\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 4,\n        \"samples\":
[\n          2,\n          \"226272\",\n          \"252000\"\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"House_Ownership\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 4,\n        \"samples\":
[\n          3,\n          \"231898\",\n          \"252000\"\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Car_Ownership\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 4,\n        \"samples\":
[\n          2,\n          \"176000\",\n          \"252000\"\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Profession\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 4,\n        \"samples\":
[\n          51,\n          \"5957\",\n          \"252000\"\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"CITY\",\n      \"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 4,\n        \"samples\": [\n          317,\n
\"1259\",\n          \"252000\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"STATE\",\n      \"properties\": {\
n        \"dtype\": \"category\",\n        \"num_unique_values\": 4,\n
\"samples\": [\n          29,\n          \"28400\",\n
\"252000\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"CURRENT_JOB_YRS\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 89093.33417147434,\n        \"min\":
0.0,\n        \"max\": 252000.0,\n        \"num_unique_values\": 8,\n
\"samples\": [\n          6.333876984126984,\n          6.0,\n
252000.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"CURRENT_HOUSE_YRS\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 89091.74741498423,\n        \"min\":
1.3990369853603093,\n        \"max\": 252000.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n
11.997793650793652,\n          12.0,\n          252000.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Risk_Flag\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
89095.38112148164,\n        \"min\": 0.0,\n        \"max\": 252000.0,\
n        \"num_unique_values\": 5,\n        \"samples\": [\n
0.123,\n          1.0,\n          0.3284378602737852\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe"}
```

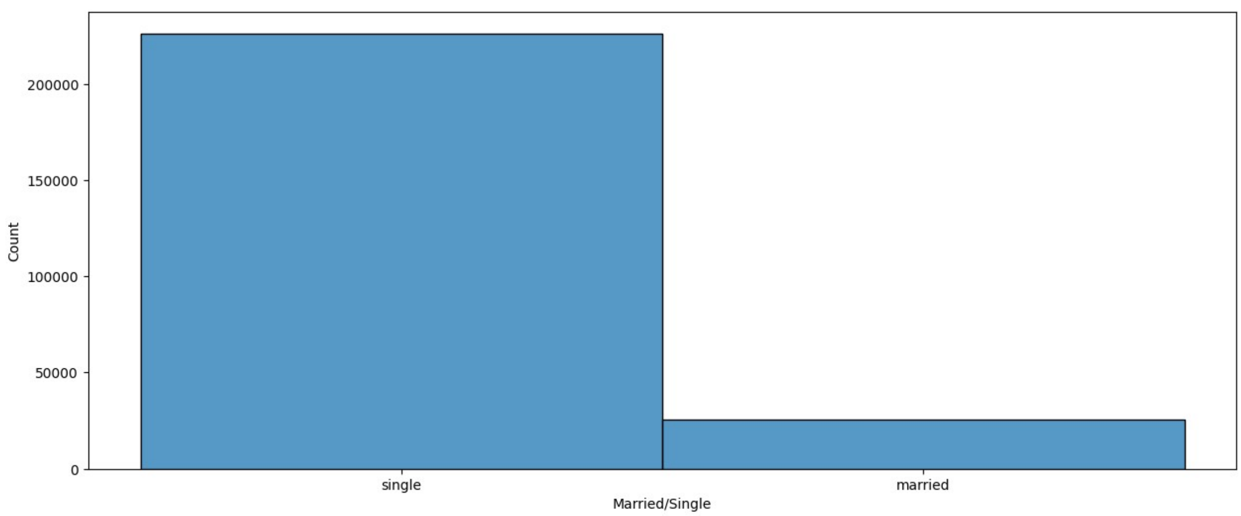
Check for Duplicates:

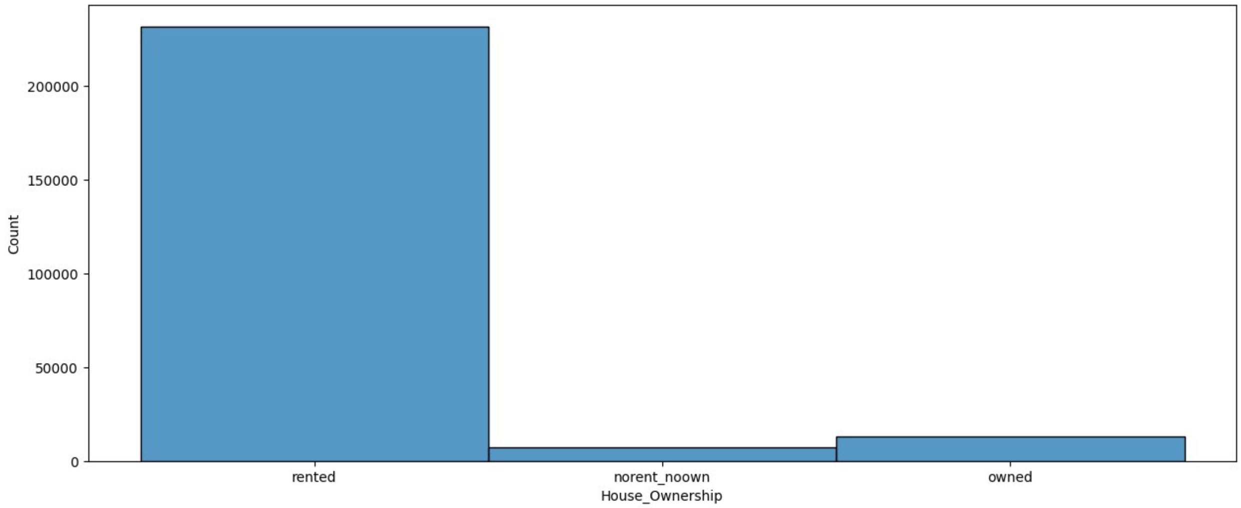
```
df.duplicated().sum()
```

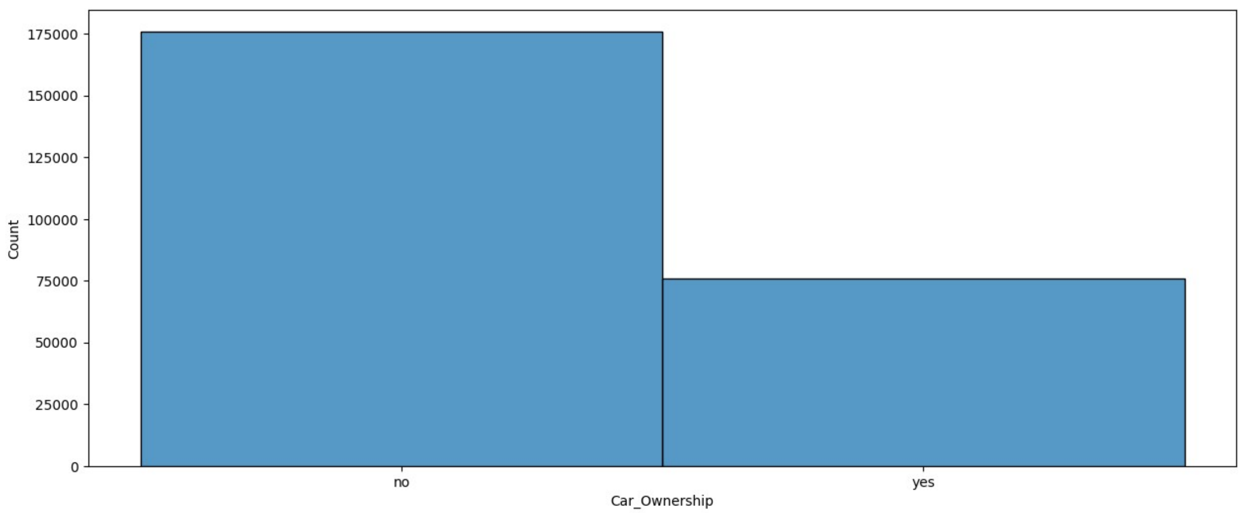
```
0
```

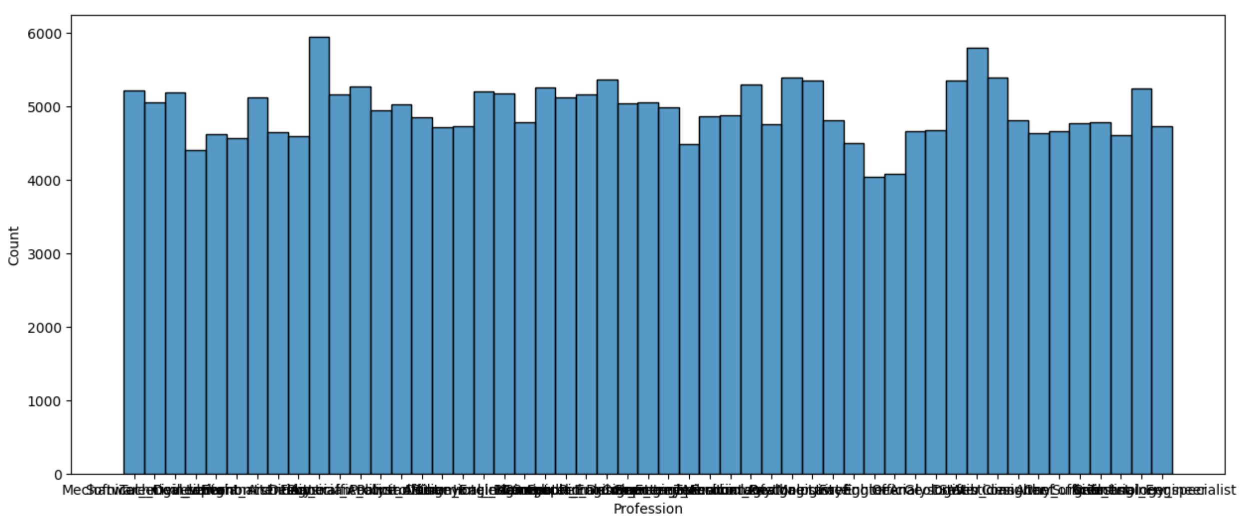
Visualize Feature Distributions:

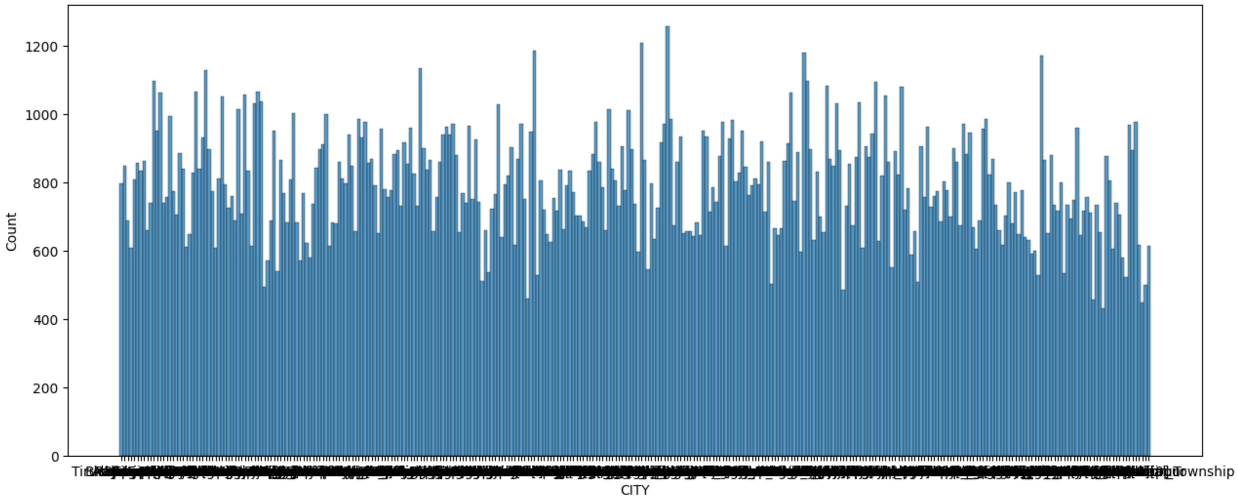
```
import plotly.express as px


for col in df.columns:  # Create boxplots for numerical features
    if df[col].dtype != 'object':
        fig = px.box(df, x=col, template='plotly_dark',
color_discrete_sequence=px.colors.qualitative.Plotly)

fig.update_traces(marker=dict(line=dict(color='rgb(100,100,100)',
width=1)))
        fig.show()

# Create histograms for categorical features
for col in df.columns:
    if df[col].dtype == 'object':
        plt.figure(figsize=(15,6))
        sns.histplot(df,x=col)
        plt.show()
```

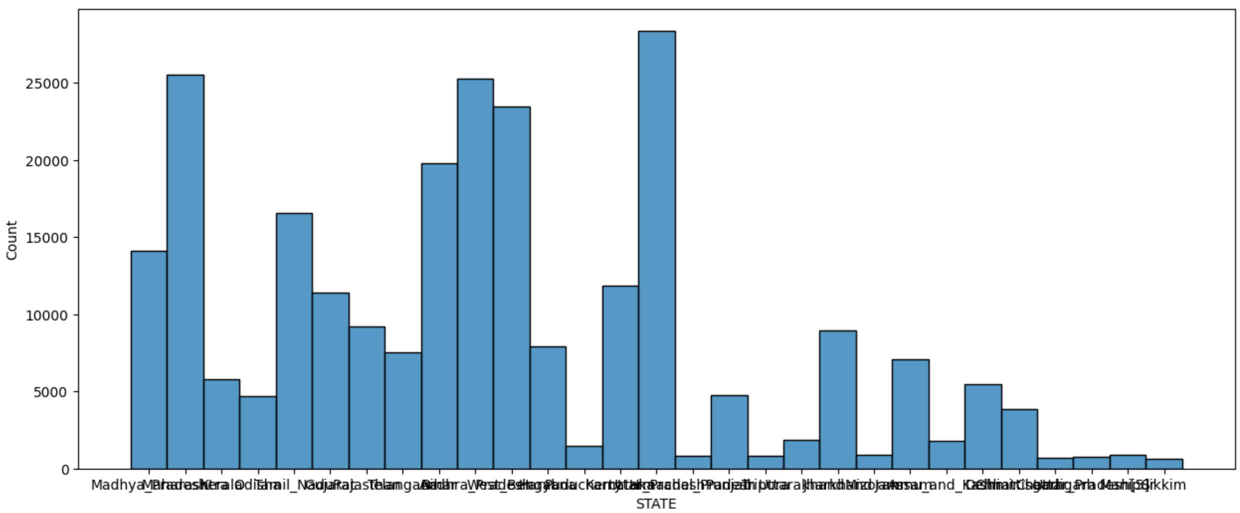



Count
House_Ownership
rented
norent_noown
owned


Count
Car_Ownership
no
yes


Count
Profession





Data Preprocessing:

```
# Drop the 'Id' column as it's not relevant for our analysis
df.drop('Id',axis=1,inplace=True)

from sklearn.preprocessing import LabelEncoder

# Encode categorical features using LabelEncoder
le = LabelEncoder()
df['Profession'] = le.fit_transform(df['Profession'])
df['CITY'] = le.fit_transform(df['CITY'])
df['STATE'] = le.fit_transform(df['STATE'])

df = pd.get_dummies(df,drop_first=True)

from sklearn.preprocessing import StandardScaler

ss = StandardScaler()
```

```
df['Income'] = ss.fit_transform(df[['Income']]) # Standardize
numerical feature ('Income')

df
```

{"type":"dataframe","variable_name":"df"}

Train-Test Split:

```
from sklearn.model_selection import train_test_split

# Features (independent variables)
X = df.drop('Risk_Flag',axis=1)
y=df['Risk_Flag']  # Target variable (dependent variable)

X_train,X_test,y_train,y_test =
train_test_split(X,y,test_size=0.15,random_state=43)
# Split data into training and testing sets

!pip install lazypredict

Collecting lazypredict
  Downloading lazypredict-0.2.12-py2.py3-none-any.whl (12 kB)
Requirement already satisfied: click in
/usr/local/lib/python3.10/dist-packages (from lazypredict) (8.1.7)
Requirement already satisfied: scikit-learn in
/usr/local/lib/python3.10/dist-packages (from lazypredict) (1.2.2)
Requirement already satisfied: pandas in
/usr/local/lib/python3.10/dist-packages (from lazypredict) (2.0.3)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-
packages (from lazypredict) (4.66.4)
Requirement already satisfied: joblib in
/usr/local/lib/python3.10/dist-packages (from lazypredict) (1.4.2)
Requirement already satisfied: lightgbm in
/usr/local/lib/python3.10/dist-packages (from lazypredict) (4.1.0)
Requirement already satisfied: xgboost in
/usr/local/lib/python3.10/dist-packages (from lazypredict) (2.0.3)
Requirement already satisfied: numpy in
/usr/local/lib/python3.10/dist-packages (from lightgbm->lazypredict)
(1.25.2)
Requirement already satisfied: scipy in
/usr/local/lib/python3.10/dist-packages (from lightgbm->lazypredict)
(1.11.4)
Requirement already satisfied: python-dateutil>=2.8.2 in
/usr/local/lib/python3.10/dist-packages (from pandas->lazypredict)
(2.8.2)
Requirement already satisfied: pytz>=2020.1 in
/usr/local/lib/python3.10/dist-packages (from pandas->lazypredict)
(2023.4)
Requirement already satisfied: tzdata>=2022.1 in
```

```
/usr/local/lib/python3.10/dist-packages (from pandas->lazypredict)
(2024.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-learn-
>lazypredict) (3.5.0)
Requirement already satisfied: six>=1.5 in
/usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.2-
>pandas->lazypredict) (1.16.0)
Installing collected packages: lazypredict
Successfully installed lazypredict-0.2.12
```

Find Best Performing Classifier

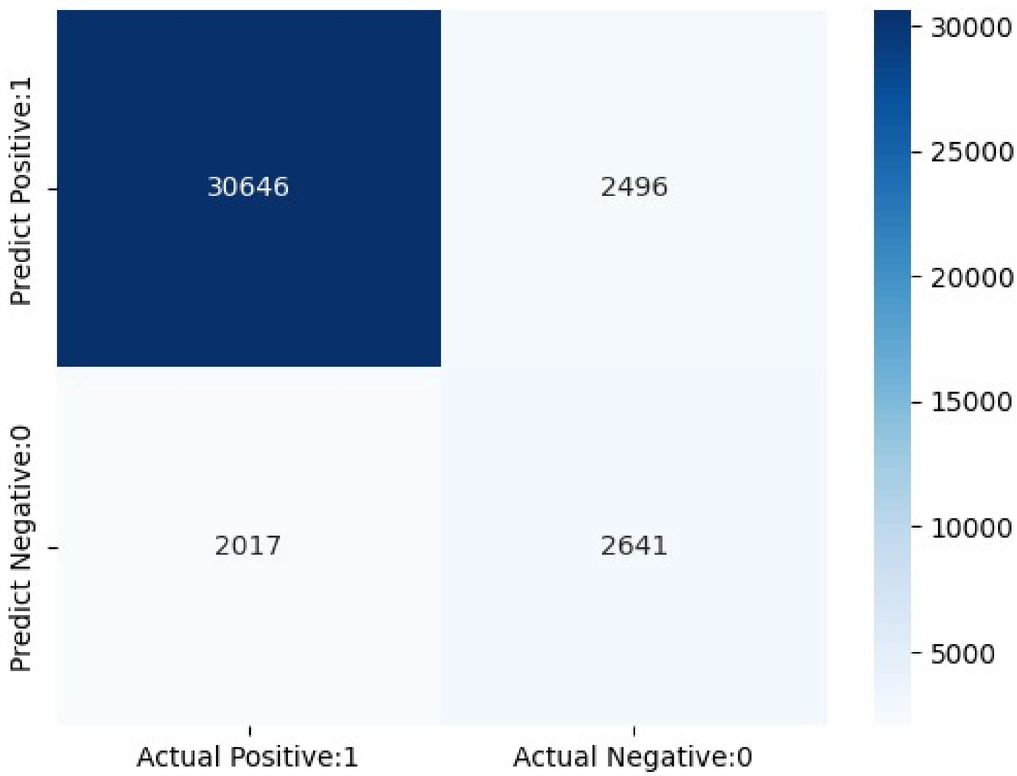
```
import lazypredict
from lazypredict.Supervised import LazyClassifier

# Define high memory classifiers to avoid memory issues
highmem_classifiers =
["LabelSpreading","LabelPropagation","BernoulliNB",'SVC',"NearestCentr
oid","NuSVC","KNeighborsClassifier", "ElasticNetClassifier",
"GradientBoostingClassifier", "HistGradientBoostingClassifier"]
classifiers = [c for c in lazypredict.Supervised.CLASSIFIERS if c[0]
not in highmem_classifiers]

# Run LazyPredict to find the best performing classifier
clf =
LazyClassifier(classifiers=classifiers,verbose=0,ignore_warnings=True)

models,predictions = clf.fit(X_train,X_test,y_train,y_test)
models # Print the list of models tested by LazyPredict

'tuple' object has no attribute '__name__'
Invalid Classifier(s)

 95%|█████████▌| 21/22 [04:16<00:03,  3.12s/it]

[LightGBM] [Info] Number of positive: 26338, number of negative:
187862
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead
of testing was 0.022911 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 688
[LightGBM] [Info] Number of data points in the train set: 214200,
number of used features: 8
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.122960 ->
initscore=-1.964695
[LightGBM] [Info] Start training from score -1.964695

100%|██████████| 22/22 [04:18<00:00, 11.74s/it]
```

```
{"summary":"{\n  \"name\": \"models\",\n  \"rows\": 20,\n  \"fields\":
[\n    {\n      \"column\": \"Model\",\n      \"properties\": {\n
\"dtype\": \"string\",\n        \"num_unique_values\": 20,\n
\"samples\": [\n          \"DecisionTreeClassifier\",\n
\"CalibratedClassifierCV\",\n          \"RidgeClassifierCV\"\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Accuracy\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.04266241785808683,\n        \"min\":
0.6936507936507936,\n        \"max\": 0.8988624338624338,\n
\"num_unique_values\": 11,\n        \"samples\": [\n
0.8892592592592593,\n          0.8809259259259259,\n
0.8767724867724868\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Balanced Accuracy\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.10692655706432809,\n        \"min\":
0.4997950710241233,\n        \"max\": 0.7473072399938032,\n
\"num_unique_values\": 11,\n        \"samples\": [\n
0.6074027217646206,\n          0.7473072399938032,\n          0.5\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"ROC AUC\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.10692655706432808,\n        \"min\": 0.49979507102412335,\n
\"max\": 0.7473072399938031,\n        \"num_unique_values\": 11,\n
\"samples\": [\n          0.6074027217646206,\n
0.7473072399938031,\n          0.5\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"F1 Score\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
0.03977371903877173,\n        \"min\": 0.7331775805199117,\n
\"max\": 0.8963889448064092,\n        \"num_unique_values\": 11,\n
\"samples\": [\n          0.8658881020693475,\n
0.8834818222497602,\n          0.8192042444987104\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Time Taken\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
27.160932590968997,\n        \"min\": 0.11005949974060059,\n
\"max\": 116.99487900733948,\n        \"num_unique_values\": 20,\n
\"samples\": [\n          1.492161750793457,\n
116.99487900733948,\n          0.6113083362579346\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe","variable_name":"models"}
```


```python
from sklearn.tree import DecisionTreeClassifier

dtc = DecisionTreeClassifier() # DecisionTreeClassifier object
model = dtc.fit(X_train,y_train) # Train the Decision Tree Classifier
y_pred = model.predict(X_test) # Make predictions on the test data
```

```
from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report
def model_evaluation(model):
    y_pred = model.predict(X_test)
    cm = confusion_matrix(y_test, y_pred)
    cm_matrix = pd.DataFrame(data=cm, columns=['Actual Positive:1',
'Actual Negative:0'],
                                 index=['Predict Positive:1', 'Predict
Negative:0'])
    print(classification_report(y_test, y_pred))
    sns.heatmap(cm_matrix, annot=True,fmt='d', cmap='Blues')
    TP = cm[0,0]
    TN = cm[1,1]
    FP = cm[0,1]
    FN = cm[1,0]
    print('Accuracy : ', (TP+TN)/(TP+TN+FP+FN))
    print('Classification Error : ',(FP + FN) / float(TP + TN + FP +
FN),"\n" )
    plt.show()
    print("\n","\n")

model_evaluation(model)
```

```
              precision    recall  f1-score   support

           0       0.94      0.92      0.93     33142
           1       0.51      0.57      0.54      4658

    accuracy                           0.88     37800
   macro avg       0.73      0.75      0.74     37800
weighted avg       0.89      0.88      0.88     37800

Accuracy :  0.8806084656084656
Classification Error :  0.1193915343915344
```



Hyperparameter Tuning

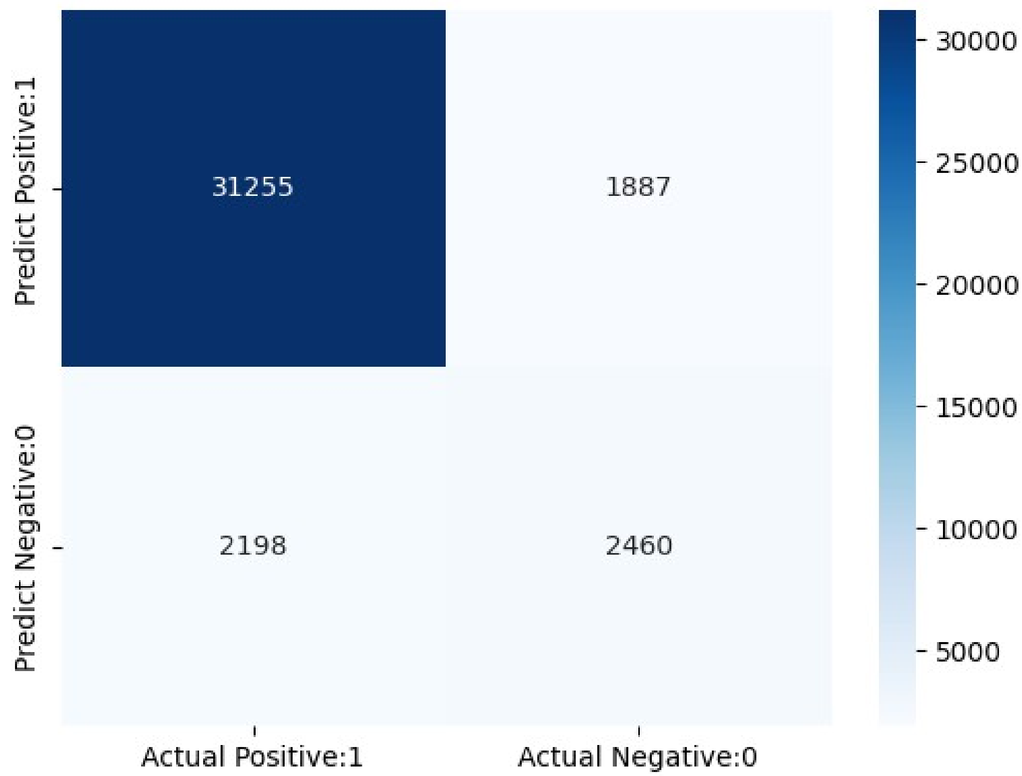
```
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint


param_dist = {
    'criterion': ['gini', 'entropy'],
    'splitter': ['best', 'random'],
    'max_depth': [None, 10, 20, 30, 40, 50],
    'min_samples_split': randint(2, 11),
    'min_samples_leaf': randint(1, 5),
    'max_features': [None, 'auto', 'sqrt', 'log2']
}

dtc = DecisionTreeClassifier()

random_search = RandomizedSearchCV(estimator=dtc,
param_distributions=param_dist, n_iter=100, cv=5, n_jobs=-1, verbose=-
1, random_state=43)
random_search.fit(X_train, y_train)
```

```
best_params = random_search.best_params_
best_model = random_search.best_estimator_

y_pred = best_model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)

print("En iyi hiperparametreler:", best_params)
print("Test seti doğruluğu:", accuracy)
```

```
En iyi hiperparametreler: {'criterion': 'gini', 'max_depth': None,
'max_features': None, 'min_samples_leaf': 4, 'min_samples_split': 8,
'splitter': 'random'}
Test seti doğruluğu: 0.8919312169312169
```

```
model_evaluation(random_search)
```

```
              precision    recall  f1-score   support

           0       0.93      0.94      0.94     33142
           1       0.57      0.53      0.55      4658

    accuracy                           0.89     37800
   macro avg       0.75      0.74      0.74     37800
weighted avg       0.89      0.89      0.89     37800

Accuracy :  0.8919312169312169
Classification Error :  0.10806878306878306
```




```
from imblearn.over_sampling import SMOTE

smote = SMOTE(random_state=42)
X_smote, y_smote = smote.fit_resample(X, y)
X_train_s,X_test_s,y_train_s,y_test_s =
train_test_split(X_smote,y_smote,test_size=0.15,random_state=43)

models_s,predicton_s = clf.fit(X_train_s,X_test_s,y_train_s,y_test_s)
models_s
```

```
'tuple' object has no attribute '__name__'
Invalid Classifier(s)

 95%|█████████▌| 21/22 [08:41<00:07,  7.20s/it]

[LightGBM] [Info] Number of positive: 187875, number of negative:
187831
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead
of testing was 0.009272 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 689
[LightGBM] [Info] Number of data points in the train set: 375706,
```

```
number of used features: 8
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500059 ->
initscore=0.000234
[LightGBM] [Info] Start training from score 0.000234

100%|██████████| 22/22 [08:45<00:00, 23.87s/it]
```


```
{"summary":"{\n  \"name\": \"models_s\",\n  \"rows\": 20,\n
\"fields\": [\n    {\n      \"column\": \"Model\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 20,\n        \"samples\": [\n
\"RandomForestClassifier\",\n
\"PassiveAggressiveClassifier\",\n          \"GaussianNB\"\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Accuracy\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.18113998728416755,\n        \"min\":
0.49241350185514765,\n        \"max\": 0.9274079213296733,\n
\"num_unique_values\": 17,\n        \"samples\": [\n
0.9274079213296733,\n          0.927136436306597,\n
0.8689179813580284\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Balanced Accuracy\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.1811287241756917,\n        \"min\":
0.4923373484779275,\n        \"max\": 0.9274449534480356,\n
\"num_unique_values\": 17,\n        \"samples\": [\n
0.9274449534480356,\n          0.9271738686736194,\n
0.8689244996769151\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"ROC
AUC\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0.1811287241756917,\n        \"min\": 0.4923373484779275,\n
\"max\": 0.9274449534480356,\n        \"num_unique_values\": 17,\n
\"samples\": [\n          0.9274449534480356,\n
0.9271738686736194,\n          0.8689244996769151\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"F1 Score\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
0.19611182712193376,\n        \"min\": 0.33296471522826127,\n
\"max\": 0.9271834934545967,\n        \"num_unique_values\": 17,\n
\"samples\": [\n          0.9271834934545967,\n
0.9269062310813593,\n          0.8689059710863337\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Time Taken\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
61.00020897221006,\n        \"min\": 0.29375505447387695,\n
\"max\": 263.0761320590973,\n        \"num_unique_values\": 20,\n
\"samples\": [\n          91.62440943717957,\n
0.7515401840209961,\n          0.29791831970214844\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe","variable_name":"models_s"}
```

```
from sklearn.ensemble import ExtraTreesClassifier

rf = ExtraTreesClassifier()
model_s = rf.fit(X_train_s,y_train_s)
y_pred_s = model_s.predict(X_test_s)

model_evaluation(model_s)
```

```
              precision    recall  f1-score   support

           0       1.00      0.91      0.95     33142
           1       0.60      0.99      0.75      4658

    accuracy                           0.92     37800
   macro avg       0.80      0.95      0.85     37800
weighted avg       0.95      0.92      0.93     37800

Accuracy :  0.9182275132275133
Classification Error :  0.08177248677248677
```