Project proposal to predict credit card approval - by Bhavesh Mewara (S6031)

Objective

Banks receive numerous credit card applications, but manual analysis, prone to errors and delays, often leads to rejections based on factors like high debt, low income, or excessive credit inquiries. Thankfully, machine learning offers a swift and accurate solution. This guide simplifies the process, illustrating how to create a credit card approval predictor using machine learning, aligning with the automated practices adopted by many banks. By automating the evaluation, the guide aims to make credit decisions faster, more reliable, and accessible, ensuring an efficient and error-free approach for users

###**Section 1:(Question to Answer)**

Q1. Why is this Proposal important in today's world? How to a good client is worthy for a bank?

- Our proposal is vital today as it employs machine learning to predict ideal clients for banks. Predicting wisely helps banks minimize risks and make informed lending decisions, fostering financial stability. This modern approach ensures efficient, secure, and customer-friendly practices, contributing to a bank's success in today's dynamic *world*

Q2.How is it going to impact the banking sector?

- Implementing our proposal with machine learning will revolutionize the banking sector, streamlining credit decisions, minimizing risks, and enhancing operational efficiency. This innovation ensures a more responsive and competitive financial landscape.

Q3. If any, what is the gap in the knowledge or how your proposed method can be helpful if required in the future for any bank in India?

- Our idea helps banks in India by using advanced technology and machine learning to better understand people's credit. It can change and improve over time, keeping up with new information. This makes it valuable for Indian banks and others, ensuring better decisions about loans.

###**Section 2: Initial Hypotheses**

In the Data Analysis (DA) track, we will aim to identify patterns in the data and important features that may impact a Machine Learning (ML) model. Our initial hypotheses are:

- Hypothesis 1: Income type, annual income, education level, Age_in_Years and Experience_years are crucial factors in predicting credit card approval.
- Hypothesis 2: Car ownership, property ownership, and family size may influence credit card approval decisions.
- Hypothesis 3: Gender, marital status and housing type may also play a role in credit card approval.

###**Section 3: Data Analysis Approach**

Q1. What approach you are going to take in order to approve and disapprove your hypothesis?

- I will do univariate, bivariate and multivariate analysis to understand relationship between features and target variable(lalel) after that i will perform the test like T-test, chisquare to test my hypothesis

Q2. What Feature Engineering Techniques will be relevant to your Project?

The Feature engineering techniques to be used in my project are data cleaning, Outlier treatment, categorical encoding, feature selection, feature scaling.

Q3. Please Justify your data analysis approch

Approch in this project are:

- Understanding and Exploration of Data
- EDA (Exploratory Data Analysis)
- Data Preprocessing
- Model Training & Model Evalution

###**Section 4: Machine Learning Approach**

Q1. Which method you will use for ML based predictions for credit card approval ?

The model used are:

1. Logistic Regression
2. Decision tree classifier
3. XgboostClassifier
4. SVM
5. KNN

Q2. Justification for Model Selection:

Support Vector Machines (SVM) are ideal for well-defined class separations, k-Nearest Neighbors (KNN) suits irregular boundaries in smaller datasets. Logistic Regression is apt for binary classification with linear relationships. Decision Trees offer interpretability and capture complex data structures, but may overfit. XGBoost, an ensemble method, balances tree-based power with regularization, excelling in diverse tasks but requiring more tuning. Model choice depends on data characteristics, interpretability needs, and trade-offs between simplicity and predictive performance. Evaluation through metrics and considering computational efficiency is crucial for a balanced model selection process.

- The most appropriate model for my project is Xgboostclassifer

Q3. Steps to Improve Model Accuracy?

- Feature selection to identify the most relevant variables. Hyperparameter tuning for model optimization. Cross-validation to assess model performance. Evaluation metrics, such as accuracy, precision, recall, and F1-score, to justify the chosen model.

Q4. Comparison of Models:

- We will compare the performance of at least four machine learning models using relevant cost functions and visualization tools to determine the most suitable model for credit card approval prediction

PREDICT CREDIT CARD APPROVAL

```
# import all Required library

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

#1. Understanding and Exploration of Data

Load the dataset

```
credit = pd.read_csv('/content/Credit_card.csv')
label = pd.read_csv('/content/Credit_card_label.csv')
```

Mergeing or concat both the datasets

```


df = credit.merge(label, how='inner', on='Ind_ID')
```

head of the dataset

```
df.head(3)

    Ind_ID GENDER Car_Owner Propert_Owner  CHILDREN  Annual_income  \
0  5008827      M         Y             Y         0       180000.0
1  5009744      F         Y             N         0       315000.0
2  5009746      F         Y             N         0       315000.0

            Type_Income         EDUCATION Marital_status
Housing_type  \
0             Pensioner  Higher education        Married  House /
apartment
1  Commercial associate  Higher education        Married  House /
apartment
2  Commercial associate  Higher education        Married  House /
apartment

   Birthday_count  Employed_days  Mobile_phone  Work_Phone  Phone
EMAIL_ID  \
0        -18772.0         365243             1           0      0
0
```

```
1        -13557.0           -586               1            1       1
0
2             NaN           -586               1            1       1
0

  Type_Occupation  Family_Members  label
0             NaN               2      1
1             NaN               2      1
2             NaN               2      1
```

Tail of the dataset

```
df.tail(3)
```

```
       Ind_ID GENDER Car_Owner Propert_Owner  CHILDREN  Annual_income
\
1545  5115992      M         Y             Y         2       180000.0

1546  5118219      M         Y             N         0       270000.0

1547  5053790      F         Y             Y         0       225000.0


     Type_Income                      EDUCATION  Marital_status  \
1545     Working               Higher education         Married
1546     Working  Secondary / secondary special  Civil marriage
1547     Working               Higher education         Married

           Housing_type  Birthday_count  Employed_days
Mobile_phone  \
1545  House / apartment        -13174.0          -2477            1

1546  House / apartment        -15292.0           -645            1

1547  House / apartment        -16601.0          -2859            1


      Work_Phone  Phone  EMAIL_ID Type_Occupation  Family_Members
label
1545           0      0         0        Managers               4
0
1546           1      1         0         Drivers               2
0
1547           0      0         0             NaN               2
0
```

Shape of the Dataset

```
df.shape
```

```
(1548, 19)
```

Dataset Info

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1548 entries, 0 to 1547
Data columns (total 19 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Ind_ID           1548 non-null   int64
 1   GENDER           1541 non-null   object
 2   Car_Owner        1548 non-null   object
 3   Propert_Owner    1548 non-null   object
 4   CHILDREN         1548 non-null   int64
 5   Annual_income    1525 non-null   float64
 6   Type_Income      1548 non-null   object
 7   EDUCATION        1548 non-null   object
 8   Marital_status   1548 non-null   object
 9   Housing_type     1548 non-null   object
 10  Birthday_count   1526 non-null   float64
 11  Employed_days    1548 non-null   int64
 12  Mobile_phone     1548 non-null   int64
 13  Work_Phone       1548 non-null   int64
 14  Phone            1548 non-null   int64
 15  EMAIL_ID         1548 non-null   int64
 16  Type_Occupation  1060 non-null   object
 17  Family_Members   1548 non-null   int64
 18  label            1548 non-null   int64
dtypes: float64(2), int64(9), object(8)
memory usage: 241.9+ KB
```

Renaming Some Columns name

```
df.rename(columns={'GENDER':'Gender','CHILDREN':'Children','EDUCATION'
:'Education','EMAIL_ID':'Email_ID','Type_Occupation':'Occupation_Type'
,'Propert_Owner':'Property_Owner','Type_Income':'Income_Type'},inplace
=True)
```

checking all columns

```
df.columns
```

```
Index(['Ind_ID', 'Gender', 'Car_Owner', 'Property_Owner', 'Children',
       'Annual_income', 'Income_Type', 'Education', 'Marital_status',
       'Housing_type', 'Birthday_count', 'Employed_days',
'Mobile_phone',
```

```
       'Work_Phone', 'Phone', 'Email_ID', 'Occupation_Type',
'Family_Members',
       'label'],
      dtype='object')
```

Numerical AND Categorical Features

```
cat_features = df.dtypes[df.dtypes == 'object'].index
cat_features

Index(['Gender', 'Car_Owner', 'Property_Owner', 'Income_Type',
'Education',
       'Marital_status', 'Housing_type', 'Occupation_Type'],
      dtype='object')

num_features = df.select_dtypes(include =['int','float']).columns
num_features

Index(['Ind_ID', 'Children', 'Annual_income', 'Birthday_count',
       'Employed_days', 'Mobile_phone', 'Work_Phone', 'Phone',
'Email_ID',
       'Family_Members', 'label'],
      dtype='object')
```

Describe function for Numerical & Categorical Features

```
df[cat_features].describe()

       Gender Car_Owner Property_Owner Income_Type  \
count    1541      1548           1548        1548
unique      2         2              2           4
top         F         N              Y     Working
freq      973       924           1010         798

                            Education Marital_status
Housing_type  \
count                            1548           1548
1548
unique                              5              5
6
top     Secondary / secondary special        Married  House /
apartment
freq                             1031           1049
1380

       Occupation_Type
count             1060
unique              18
top           Laborers
freq               268
```

```
df[num_features].describe()
```

```
                Ind_ID     Children  Annual_income  Birthday_count  \
count  1.548000e+03  1548.000000   1.525000e+03     1526.000000
mean   5.078920e+06     0.412791   1.913993e+05   -16040.342071
std    4.171759e+04     0.776691   1.132530e+05     4229.503202
min    5.008827e+06     0.000000   3.375000e+04   -24946.000000
25%    5.045070e+06     0.000000   1.215000e+05   -19553.000000
50%    5.078842e+06     0.000000   1.665000e+05   -15661.500000
75%    5.115673e+06     1.000000   2.250000e+05   -12417.000000
max    5.150412e+06    14.000000   1.575000e+06    -7705.000000

       Employed_days  Mobile_phone   Work_Phone        Phone
Email_ID  \
count    1548.000000        1548.0  1548.000000  1548.000000
1548.000000
mean    59364.689922           1.0     0.208010     0.309432
0.092377
std    137808.062701           0.0     0.406015     0.462409
0.289651
min    -14887.000000           1.0     0.000000     0.000000
0.000000
25%     -3174.500000           1.0     0.000000     0.000000
0.000000
50%     -1565.000000           1.0     0.000000     0.000000
0.000000
75%      -431.750000           1.0     0.000000     1.000000
0.000000
max    365243.000000           1.0     1.000000     1.000000
1.000000

       Family_Members        label
count     1548.000000  1548.000000
mean         2.161499     0.113049
std          0.947772     0.316755
min          1.000000     0.000000
25%          2.000000     0.000000
50%          2.000000     0.000000
75%          3.000000     0.000000
max         15.000000     1.000000
```

```
df.nunique() == 1
```

```
Ind_ID              False
Gender              False
Car_Owner           False
Property_Owner      False
Children            False
Annual_income       False
Income_Type         False
```

```
Education            False
Marital_status       False
Housing_type         False
Birthday_count       False
Employed_days        False
Mobile_phone          True
Work_Phone           False
Phone                False
Email_ID             False
Occupation_Type      False
Family_Members       False
label                False
dtype: bool
```

Dropping the mobile_phone column bez. its fill with only one value

```
df.drop(columns=['Mobile_phone'],inplace=True)
```

checking for duplicated values

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

0
```

Cheking the missing values(null)

```
df.isnull().sum()

Ind_ID               0
Gender               7
Car_Owner            0
Property_Owner       0
Children             0
Annual_income       23
Income_Type          0
Education            0
Marital_status       0
Housing_type         0
Birthday_count      22
Employed_days        0
Work_Phone           0
Phone                0
Email_ID             0
Occupation_Type    488
Family_Members       0
label                0
dtype: int64
```

missing values in the below features:

- Gender
- Annual_income
- Birthday_count
- Occupation_Type

```
# let check how much percentage of null values they have

df.isnull().mean()*100

Ind_ID              0.000000
Gender              0.452196
Car_Owner           0.000000
Property_Owner      0.000000
Children            0.000000
Annual_income       1.485788
Income_Type         0.000000
Education           0.000000
Marital_status      0.000000
Housing_type        0.000000
Birthday_count      1.421189
Employed_days       0.000000
Work_Phone          0.000000
Phone               0.000000
Email_ID            0.000000
Occupation_Type    31.524548
Family_Members      0.000000
label               0.000000
dtype: float64

# Handling missing for gender, Annual_income, Occuption_Type,
birthday_count columns
```

GENDER is categorical variable so fill with MODE

```
df['Gender'] = df['Gender'].fillna(df['Gender'].mode()[0])

df["Gender"].value_counts()

F    980
M    568
Name: Gender, dtype: int64
```

Dropping the Occuption column as multiple values are missing

```
df.drop(columns=['Occupation_Type'], inplace=True)
```

Annual_income, Age_in_Years are numerical variable so fill with mean & mode

```
df['Annual_income'] =
df['Annual_income'].fillna(df['Annual_income'].mean())
```

```
df['Birthday_count'] = 
df['Birthday_count'].fillna(df['Birthday_count'].mean())

df.isnull().sum()
```

```
Ind_ID             0
Gender             0
Car_Owner          0
Property_Owner     0
Children           0
Annual_income      0
Income_Type        0
Education          0
Marital_status     0
Housing_type       0
Birthday_count     0
Employed_days      0
Work_Phone         0
Phone              0
Email_ID           0
Family_Members     0
label              0
dtype: int64
```

Converting Birthday_count and Employeed_days into Years

```
# Creating new column by dividing the birthday_count with 365, by 
which we get age in years

df['Age_in_Years'] = round(np.abs(df['Birthday_count']/365),0)

 # Crearting new column which gives experience in years by dividing 
employed_days with 365

df['Experience_years']= round(np.abs(df['Employed_days'])/365,0)


df['Experience_years'].sample(5)
```

```
581     1001.0
858        3.0
1267       1.0
1052    1001.0
332       23.0
Name: Experience_years, dtype: float64
```

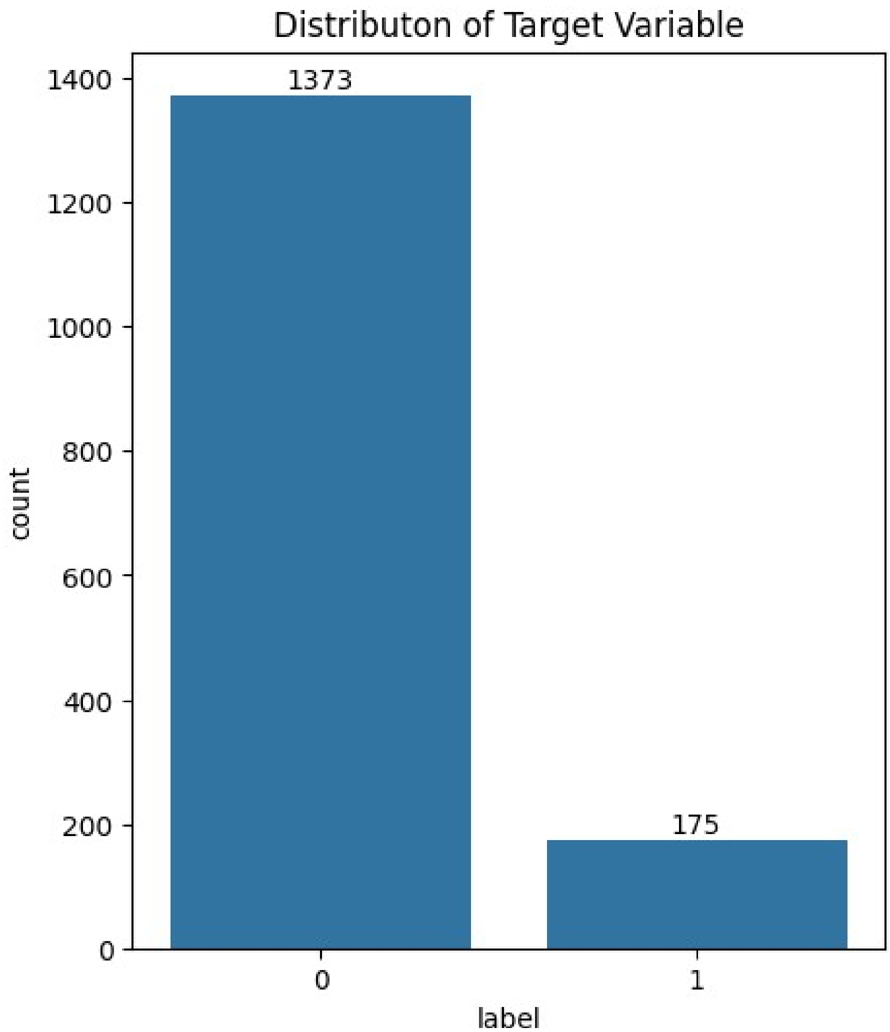
```
# Dropping Unnecessary columns

df.drop(columns=['Birthday_count','Employed_days'], inplace=True)

df.head(3)
```

```
   Ind_ID Gender Car_Owner Property_Owner  Children  Annual_income  \
0  5008827      M         Y              Y         0       180000.0
1  5009744      F         Y              N         0       315000.0
2  5009746      F         Y              N         0       315000.0

            Income_Type         Education Marital_status
Housing_type  \
0             Pensioner  Higher education        Married  House /
apartment
1  Commercial associate  Higher education        Married  House /
apartment
2  Commercial associate  Higher education        Married  House /
apartment

   Work_Phone  Phone  Email_ID  Family_Members  label  Age_in_Years  \
0           0      0         0               2      1          51.0
1           1      1         0               2      1          37.0
2           1      1         0               2      1          44.0

   Experience_years
0            1001.0
1               2.0
2               2.0
```

2. Explorartory Data Analysis (EDA)

Understanding Label Column

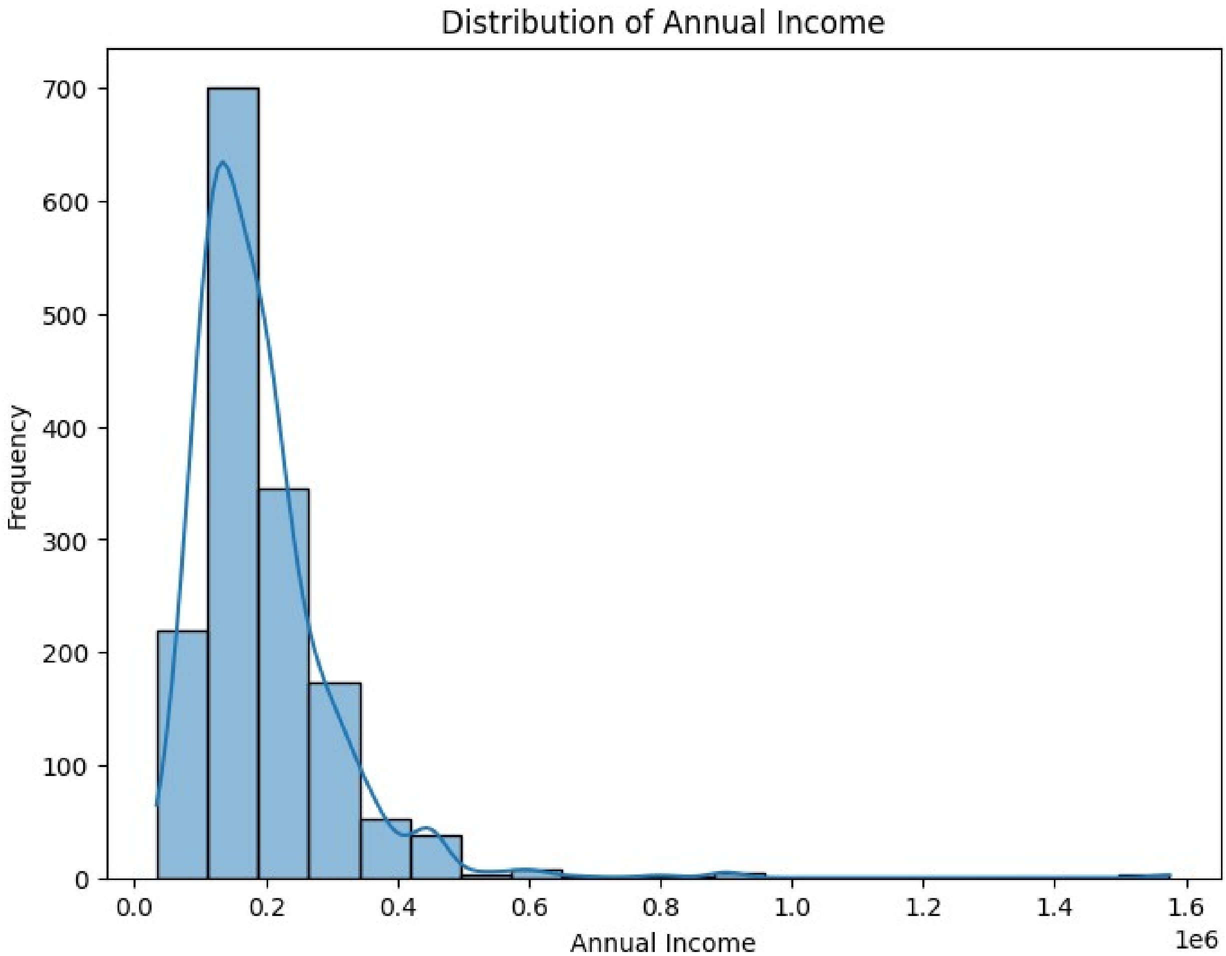
```
plt.figure(figsize=(5,6))
bx = sns.countplot(data=df, x='label')
plt.title('Distributon of Target Variable')

for bars in bx.containers:
  bx.bar_label(bars)


plt.show()
```



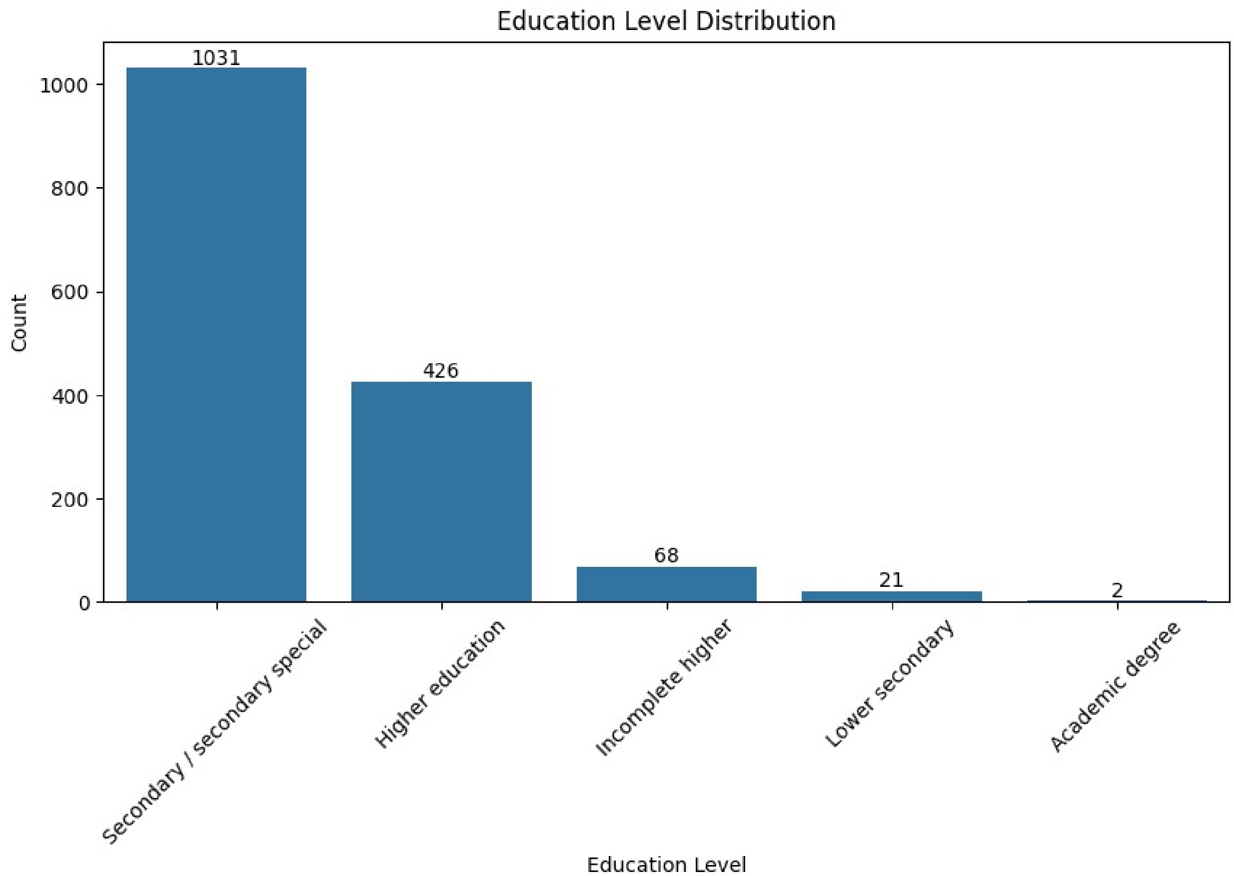
So with the help of graph its clearly visible that datatset is imbalanced as there is huge difference b/w approved(0) and rejected(1)

###2.1 Univariate Analysis

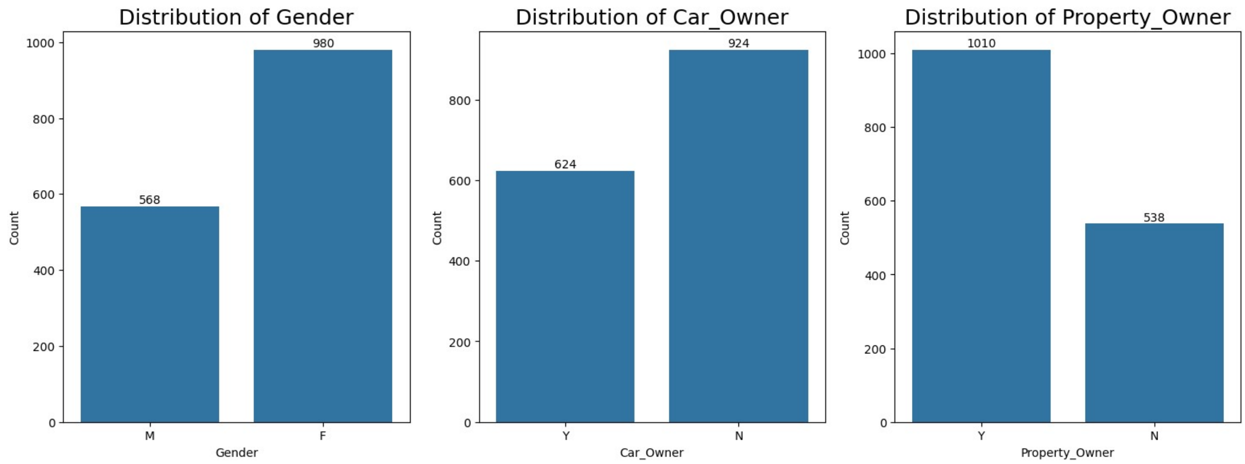
```
# here Annual_income is in numerical form so i used histogram
(univariate)

plt.figure(figsize=(8, 6))
sns.histplot(df['Annual_income'], bins=20, kde=True)
plt.title('Distribution of Annual Income')
plt.xlabel('Annual Income')
plt.ylabel('Frequency')
plt.show()
```



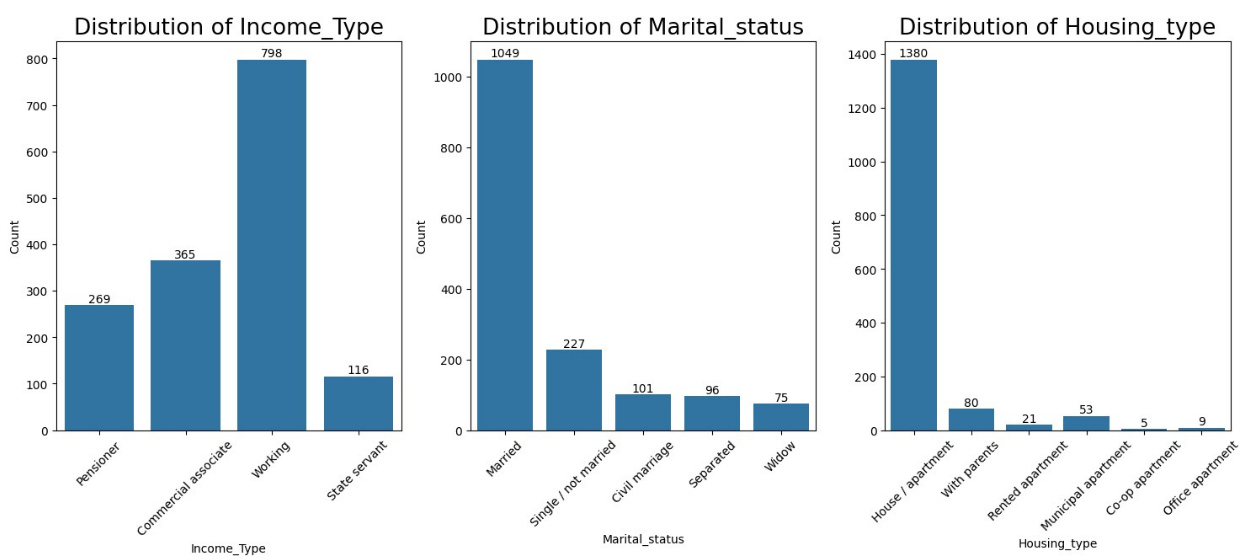

```
# Education Distribution

plt.figure(figsize=(10,5))

eds = df.Education.value_counts().index
bx = sns.barplot(x=eds, y=df.Education.value_counts().values)
plt.xlabel('Education Level')
plt.ylabel('Count')
plt.title('Education Level Distribution')
plt.xticks(rotation=45)

for bars in bx.containers:
  bx.bar_label(bars)

plt.show()
```



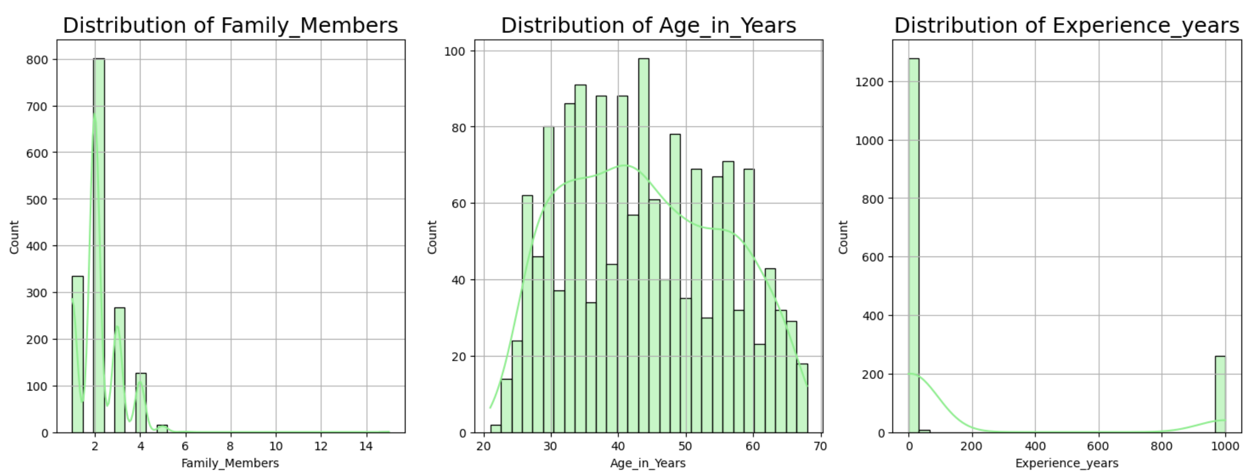

```
cols = ['Gender', 'Car_Owner', 'Property_Owner']

# Set up the figure and axes

fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(18, 6))

# Plot univariate distributions for each column
for i, col in enumerate(cols):
    bx = sns.countplot(data=df , x=col, ax=axes[i])
    axes[i].set_title(f'Distribution of {col}' , fontsize=18 )
    axes[i].set_ylabel('Count')
    axes[i].tick_params(axis='x')

    for bars in bx.containers:
      bx.bar_label(bars)

plt.show()
```




```
cols = ['Income_Type', 'Marital_status', 'Housing_type']

# set up figures and axes

fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(18,6))

# plot univariate distribution of cols

for i, col in enumerate(cols):
  bx = sns.countplot(data=df, x=col, ax=axes[i])
  axes[i].set_title(f'Distribution of {col}', fontsize=19)
  axes[i].set_ylabel('Count')
  axes[i].tick_params(axis='x', rotation=45)

  for bars in bx.containers:
    bx.bar_label(bars)

plt.show()
```



```python
cols = ['Family_Members', 'Age_in_Years', 'Experience_years']

# Set up the figure and axes
fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(18, 6))

# Plot univariate distributions for each column
for i, col in enumerate(cols):
    if df[col].dtype in ['int64', 'float64']:
        sns.histplot(data=df, x=col, ax=axes[i], bins=30, kde=True ,
color ='lightgreen')
    else:
        sns.countplot(data=df, x=col, ax=axes[i], color ='green')

    axes[i].set_title(f'Distribution of {col}', fontsize = 18)
    axes[i].set_ylabel('Count')
    axes[i].tick_params(axis='x')
    axes[i].grid(True)


plt.show()
```



2.2 Bivariate Analysis

Numericals v/s Label columns

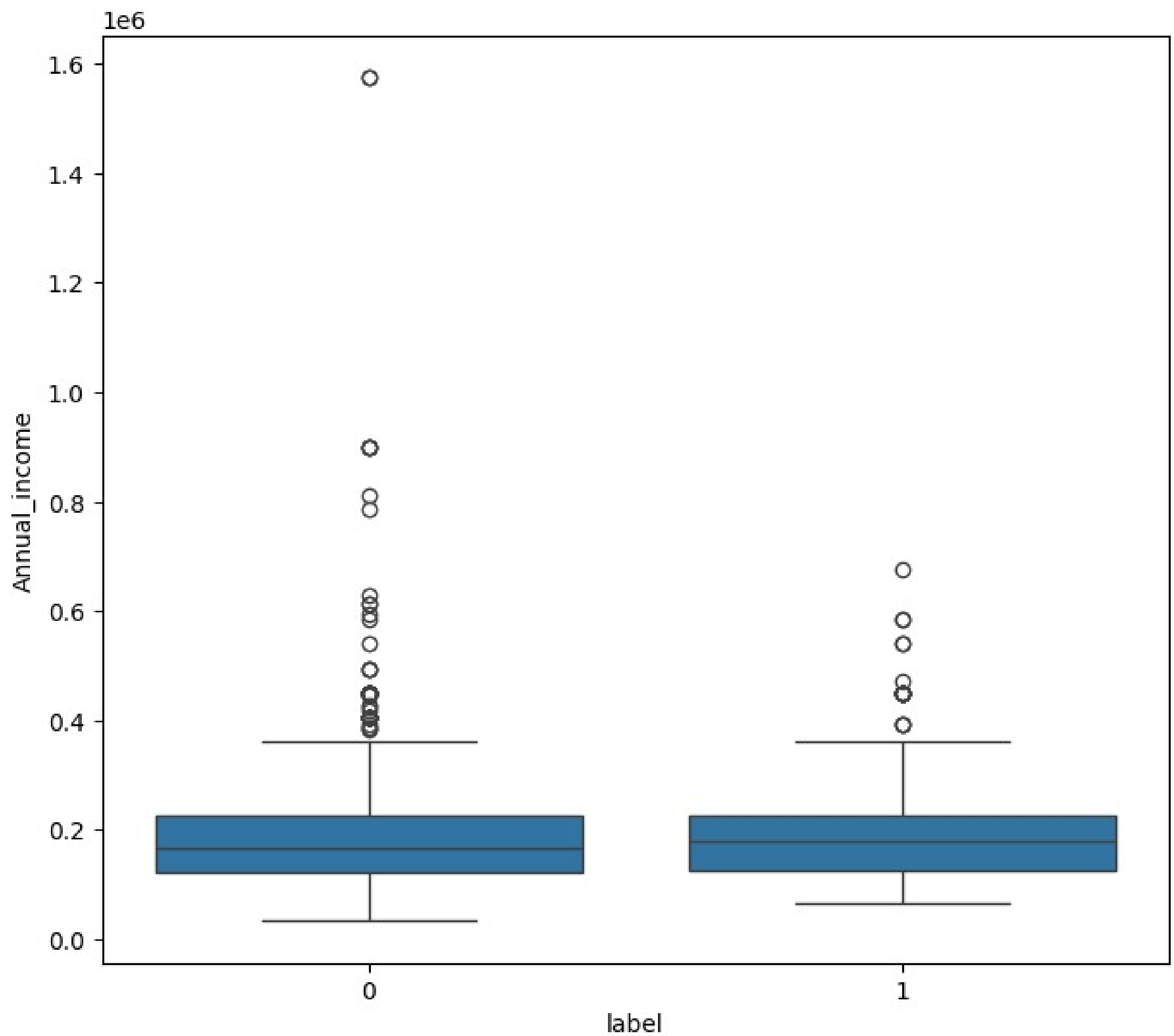
```python
num_features
```

```
Index(['Ind_ID', 'Children', 'Annual_income', 'Birthday_count',
       'Employed_days', 'Mobile_phone', 'Work_Phone', 'Phone',
'Email_ID',
       'Family_Members', 'label'],
      dtype='object')
```

Annual Income v/s Label

```
plt.figure(figsize=(8,7))
sns.boxplot(x='label', y='Annual_income', data=df)
```

```
<Axes: xlabel='label', ylabel='Annual_income'>
```



We can see some difference it in so we will do hypothesis testing (using t-test)

- Null Hypothesis (H0): No significant difference in credit card approval rates based on annual income
- Alternatives Hypothesis(H1): There is a significant difference in credit card approval rates based on annual income

```
import pandas as pd
from scipy.stats import ttest_ind
```

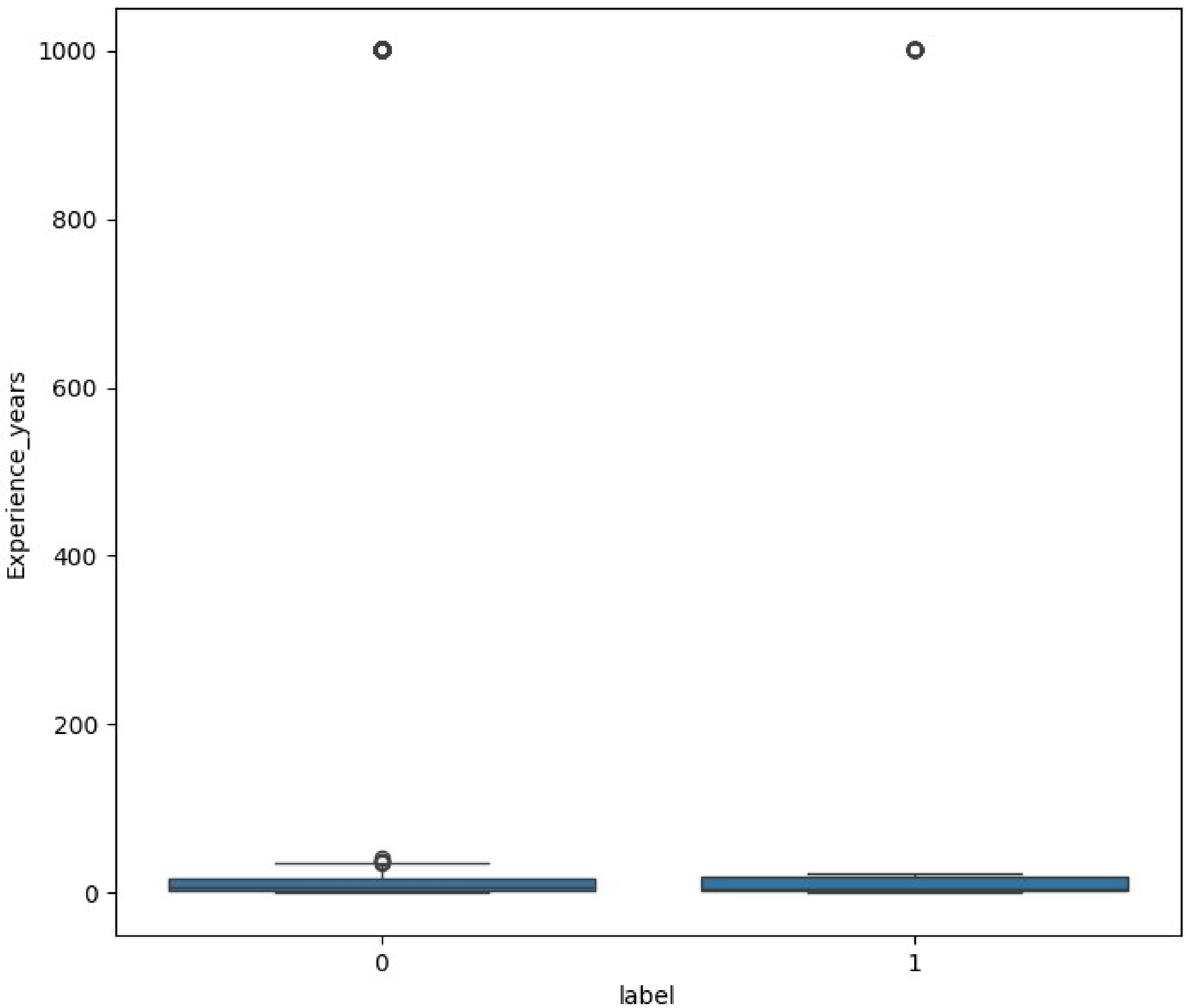
```python
# split data into 2 groups based on label(0 for approval, 1 for
rejected)

approval_group = df[df['label']==0]['Annual_income']
rejected_group = df[df['label']==1]['Annual_income']

# perform ttest

t_statistic, p_value = ttest_ind(approval_group.dropna(),
rejected_group.dropna(), equal_var=False)

# output of result

print('T_statistic:',t_statistic)
print('p_value:',p_value)

if p_value < 0.05:
  print('Reject the null Hypothesis')
else:
  print('Fail to reject Null Hypothesis ')
```

```
T_statistic: -1.0343837479460318
p_value: 0.3021025557793681
Fail to reject Null Hypothesis
```

Experience_years V/s label

```python
plt.figure(figsize=(8,7))
sns.boxplot(x='label', y='Experience_years', data=df)
```

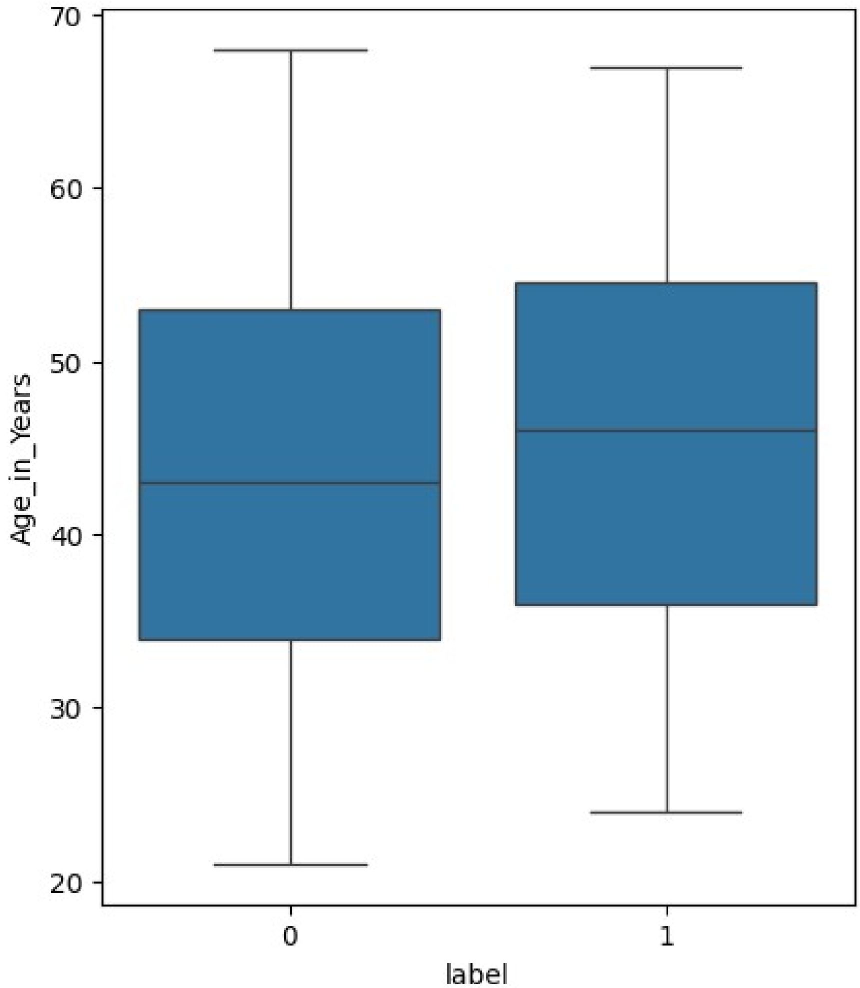
```
<Axes: xlabel='label', ylabel='Experience_years'>
```



Age_in_Years vs Label

```
plt.figure(figsize=(5,6))
sns.boxplot(x='label',y='Age_in_Years', data =df)
```

```
<Axes: xlabel='label', ylabel='Age_in_Years'>
```

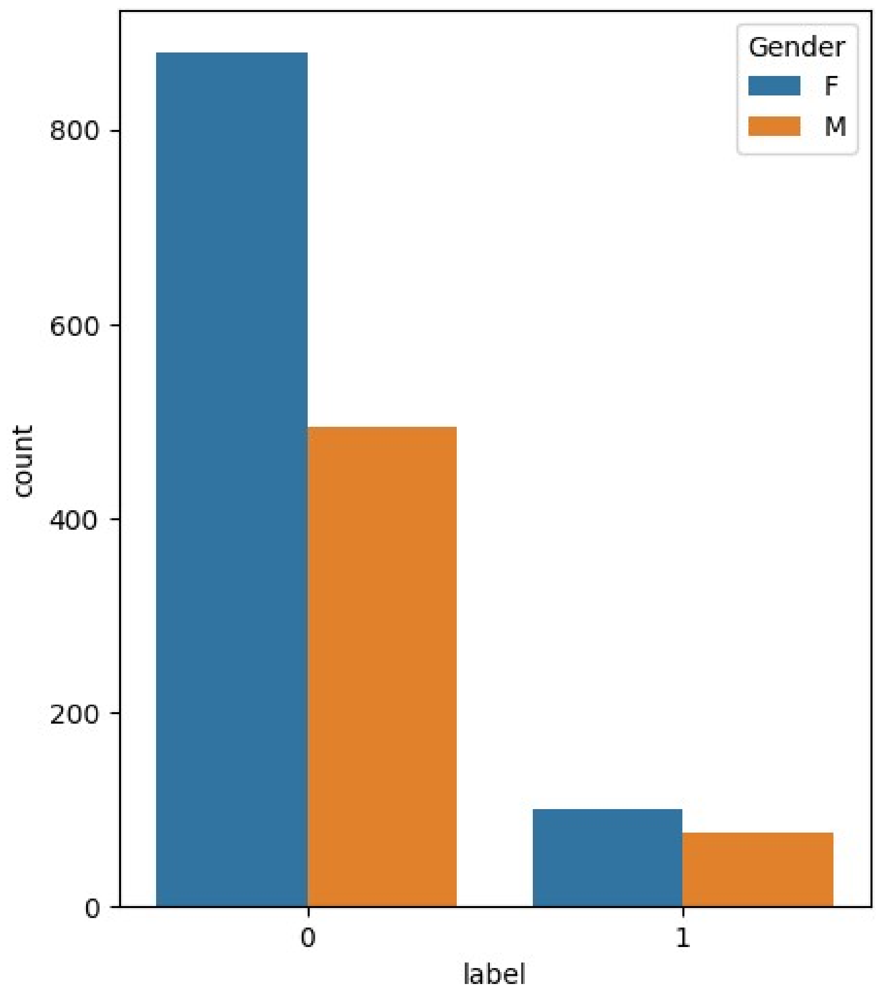


Age_in_Years - age Group in kind of normally distributed as we seen in univariate analysis, but there is difference between Avg. age of Approval and Rejected

Gender v/s Label

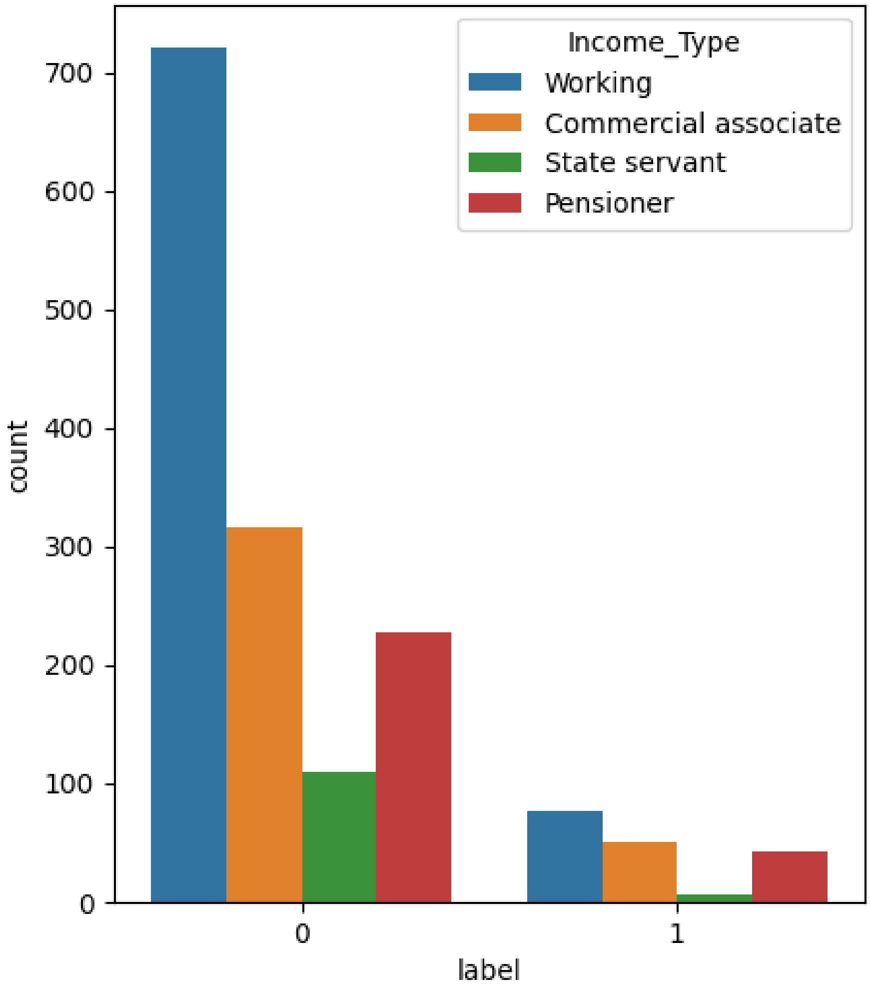
```
plt.figure(figsize=(5,6))
sns.countplot(x='label', hue='Gender', data=df)
```

```
<Axes: xlabel='label', ylabel='count'>
```



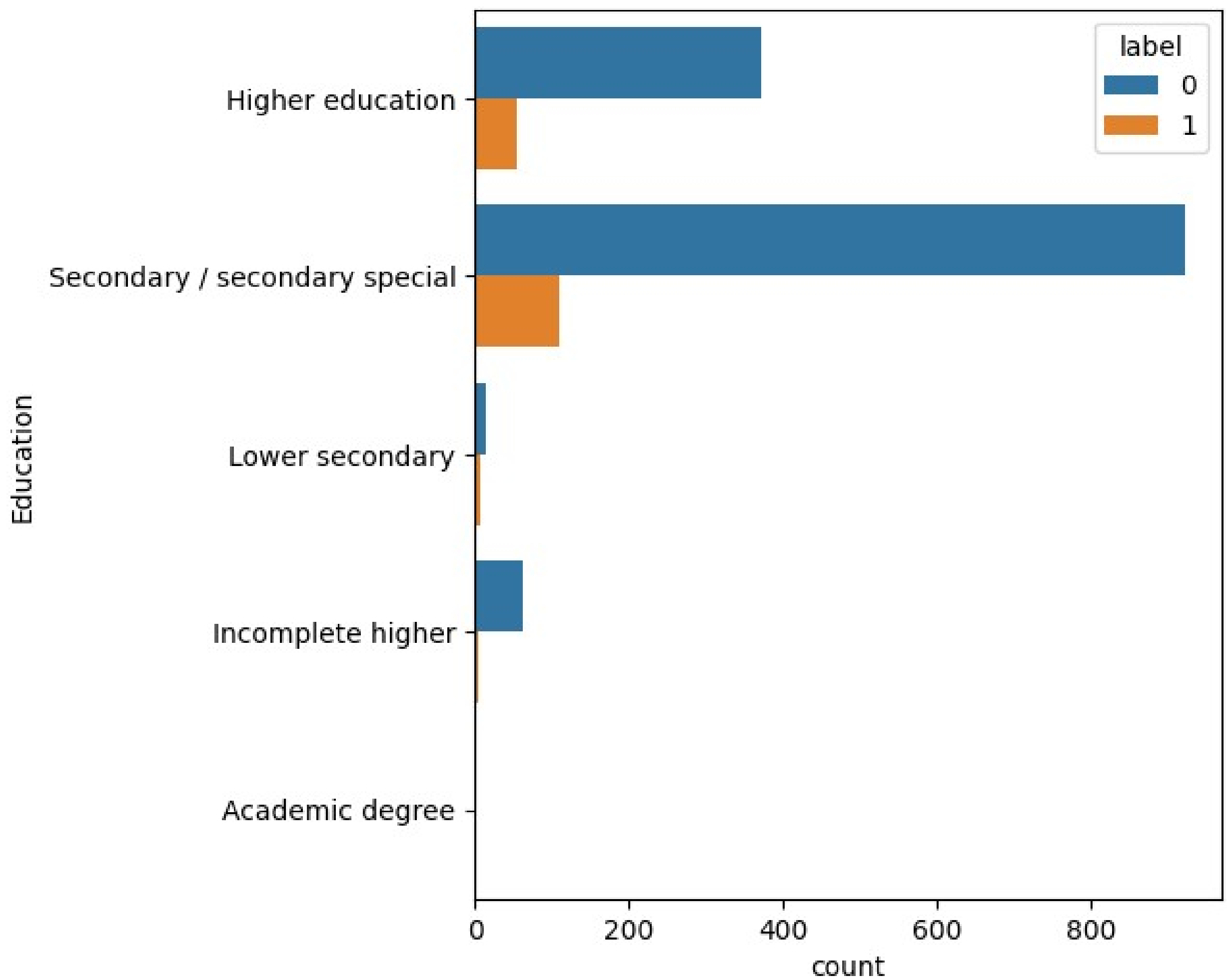
Income_type v/s Label

```
plt.figure(figsize=(5,6))
sns.countplot(x='label', hue='Income_Type', data=df)

<Axes: xlabel='label', ylabel='count'>
```




```
plt.figure(figsize=(5,6))
sns.countplot(y='Education', hue='label', data=df)
```

```
<Axes: xlabel='count', ylabel='Education'>
```



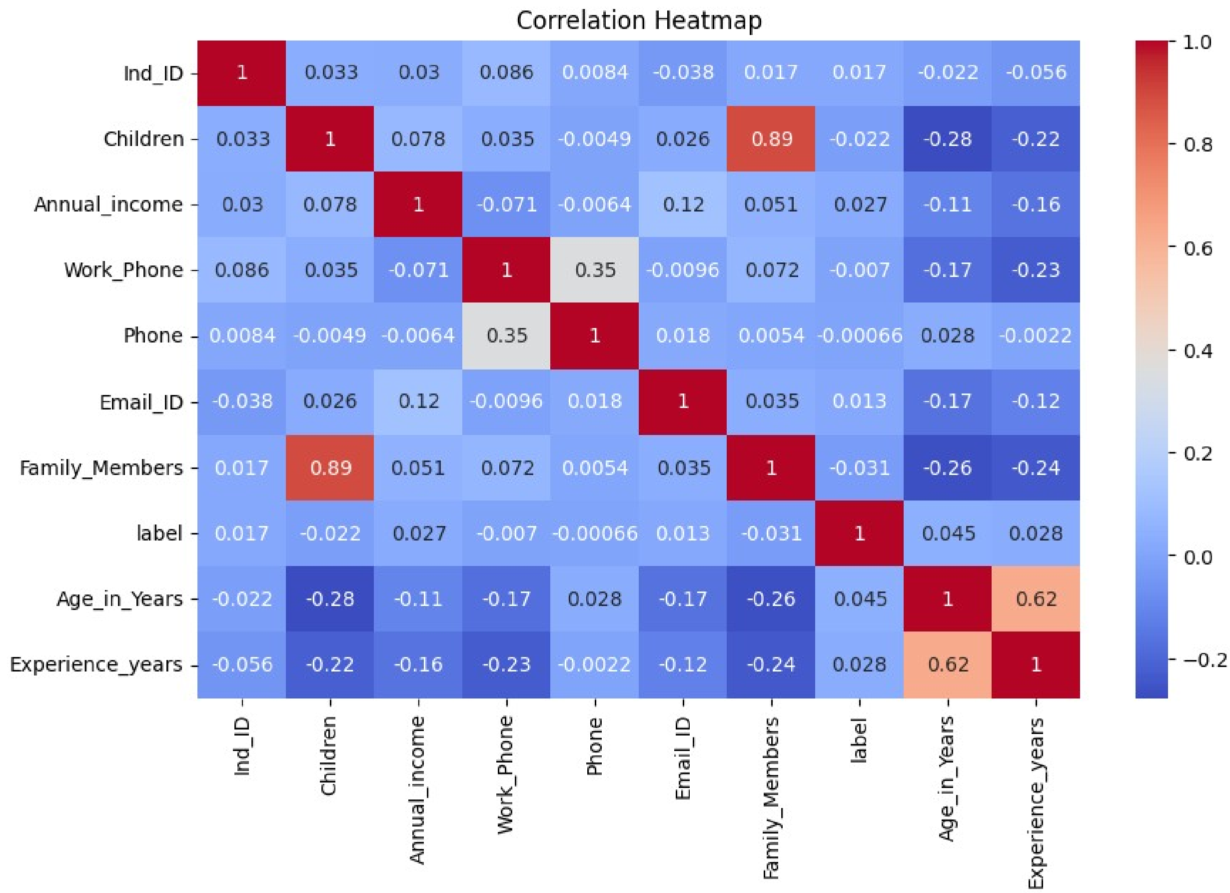

```
df.corr()

<ipython-input-48-2f6f6606aa2c>:1: FutureWarning: The default value of
numeric_only in DataFrame.corr is deprecated. In a future version, it
will default to False. Select only valid columns or specify the value
of numeric_only to silence this warning.
  df.corr()

                    Ind_ID  Children  Annual_income  Work_Phone
Phone  \
Ind_ID            1.000000  0.032535       0.029888    0.085794
0.008403
Children          0.032535  1.000000       0.078111    0.035014 -
0.004908
Annual_income     0.029888  0.078111       1.000000   -0.070541 -
0.006384
Work_Phone        0.085794  0.035014      -0.070541    1.000000
0.352439
Phone             0.008403 -0.004908      -0.006384    0.352439
```

```
1.000000
Email_ID          -0.037923  0.025776       0.121842   -0.009594
0.018105
Family_Members     0.016950  0.890248       0.050677    0.072228
0.005372
label              0.016796 -0.021646       0.026875   -0.007046 -
0.000664
Age_in_Years      -0.022025 -0.276852      -0.109767   -0.172196
0.028307
Experience_years  -0.055900 -0.219702      -0.159497   -0.230323 -
0.002248

                   Email_ID  Family_Members     label  Age_in_Years  \
Ind_ID            -0.037923        0.016950  0.016796     -0.022025
Children           0.025776        0.890248 -0.021646     -0.276852
Annual_income      0.121842        0.050677  0.026875     -0.109767
Work_Phone        -0.009594        0.072228 -0.007046     -0.172196
Phone              0.018105        0.005372 -0.000664      0.028307
Email_ID           1.000000        0.035098  0.012921     -0.166469
Family_Members     0.035098        1.000000 -0.030709     -0.263470
label              0.012921       -0.030709  1.000000      0.044841
Age_in_Years      -0.166469       -0.263470  0.044841      1.000000
Experience_years  -0.121353       -0.238921  0.028468      0.622225

                   Experience_years
Ind_ID                    -0.055900
Children                  -0.219702
Annual_income             -0.159497
Work_Phone                -0.230323
Phone                     -0.002248
Email_ID                  -0.121353
Family_Members            -0.238921
label                      0.028468
Age_in_Years               0.622225
Experience_years           1.000000
```

2.3 Multivariate Analysis

```
plt.figure(figsize = (10,6))
sns.heatmap(df.corr(), annot=True , cmap='coolwarm')
plt.title("Correlation Heatmap")
plt.show()
```

```
<ipython-input-49-c01bbd619d2f>:2: FutureWarning: The default value of
numeric_only in DataFrame.corr is deprecated. In a future version, it
will default to False. Select only valid columns or specify the value
of numeric_only to silence this warning.
  sns.heatmap(df.corr(), annot=True , cmap='coolwarm')
```



Interpretation

- Children and Family members have linear corelation
- Age and experience also show some corelation
- Age has some -ve correalation with the family_members and children
- Another positive correlation between phone and work phone.
- Age_in_years is highly correlated with Experience_years

Before data go for futher step or process. we have to export cleaned data to used in SOL Query

```
df.to_csv('Credit_and_label.csv', index=False)
```

Creating new dataframe to be used for SQL Query

```
clean_df = df.copy()

df.drop(columns=['Ind_ID'], inplace=True)
```

3 Data Preprocessing or Data Engineering

```
df.head()
```

```
  Gender Car_Owner Property_Owner  Children  Annual_income  \
0      M         Y              Y         0   180000.00000
1      F         Y              N         0   315000.00000
2      F         Y              N         0   315000.00000
3      F         Y              N         0   191399.32623
4      F         Y              N         0   315000.00000

            Income_Type         Education Marital_status
Housing_type  \
0             Pensioner  Higher education        Married  House /
apartment
1  Commercial associate  Higher education        Married  House /
apartment
2  Commercial associate  Higher education        Married  House /
apartment
3  Commercial associate  Higher education        Married  House /
apartment
4  Commercial associate  Higher education        Married  House /
apartment

   Work_Phone  Phone  Email_ID  Family_Members  label  Age_in_Years  \
0           0      0         0               2      1          51.0
1           1      1         0               2      1          37.0
2           1      1         0               2      1          44.0
3           1      1         0               2      1          37.0
4           1      1         0               2      1          37.0

   Experience_years
0            1001.0
1               2.0
2               2.0
3               2.0
4               2.0
```

3.1 Checking & Handling Outliers

```python
# Checking the Outlier

Outliers = []

def detect_outliers(data):
  threshold = 3
  mean = np.mean(data)
  std = np.std(data)

  for i in data:
    z_score=(i-mean)/std
    if np.abs(z_score) > threshold:
      Outliers.append(i)
```

```
    return Outliers
```

```
detect_outliers(df['Experience_years'])
```

```
[]
```

```
detect_outliers(df['Annual_income'])
detect_outliers(df['Family_Members'])
detect_outliers(df['Children'])
```

```
[540000.0,
 540000.0,
 675000.0,
 585000.0,
 585000.0,
 1575000.0,
 1575000.0,
 900000.0,
 540000.0,
 612000.0,
 612000.0,
 787500.0,
 594000.0,
 585000.0,
 900000.0,
 900000.0,
 900000.0,
 630000.0,
 810000.0,
 6,
 15,
 4,
 3,
 3,
 3,
 3,
 14,
 3,
 3,
 3,
 3,
 3,
 3,
 3,
 3,
 3,
 3,
 3,
 3]
```

so with the help of function i find outliers in (annual_income, family_members, children, Experience_years)

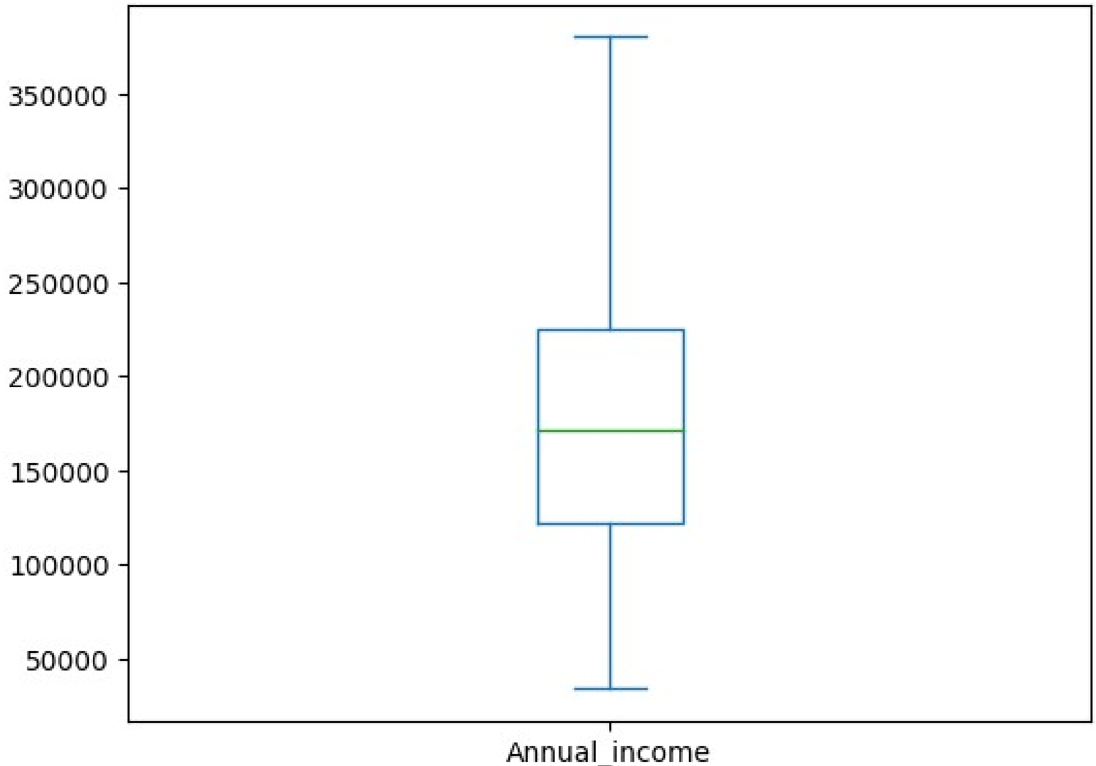
Annual_income

```
df.describe()['Annual_income']

q1 = df.describe()['Annual_income']['25%']
q3 = df.describe()['Annual_income']['75%']

print(q1)
print(q3)

IQR=q3-q1
print(IQR)


lower_fence= q1-1.5*IQR
upper_fence= q3+1.5*IQR

print(lower_fence)
print(upper_fence)
```

```
121500.0
225000.0
103500.0
-33750.0
380250.0
```

```
df['Annual_income']=df['Annual_income'].clip(lower_fence,upper_fence)

df['Annual_income'].plot(kind='box')
```

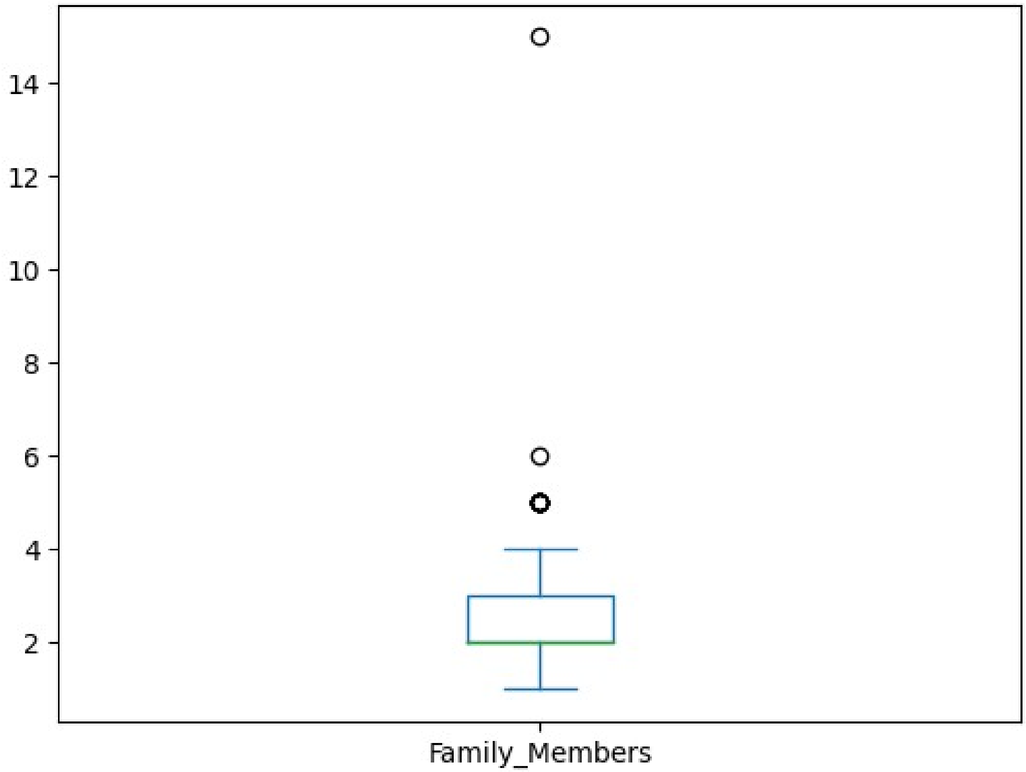
```
<Axes: >
```



Family_Menbers

```
df['Family_Members'].plot(kind='box')
```

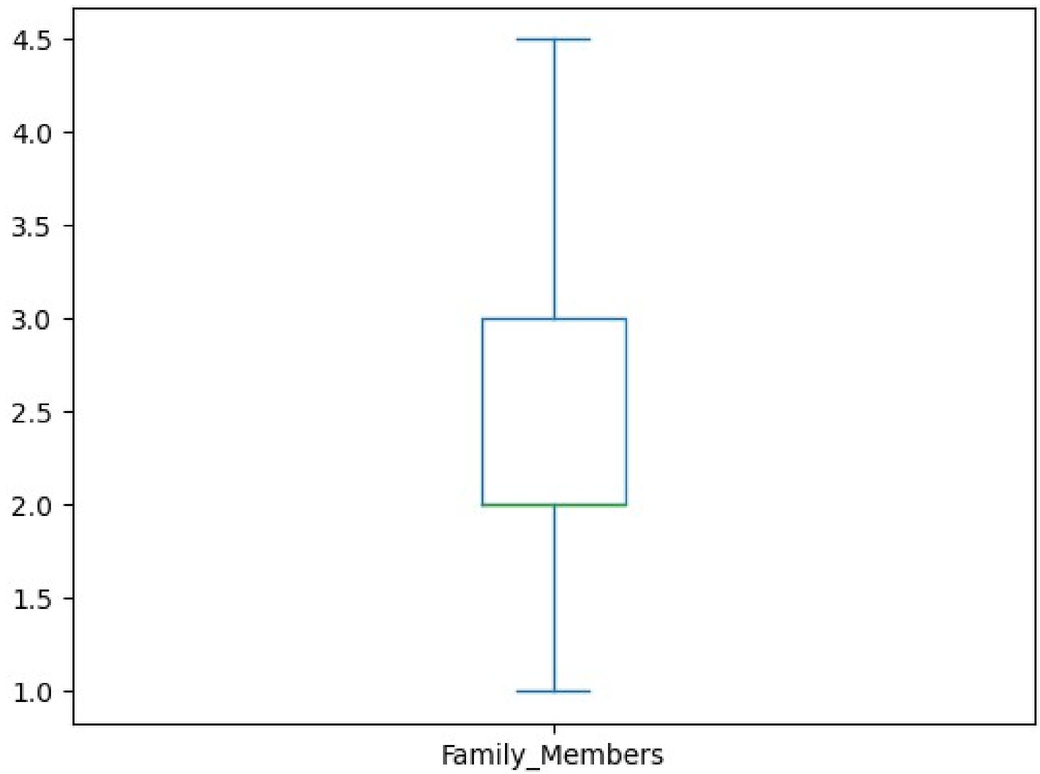
```
<Axes: >
```




```
df.describe()['Family_Members']

q1 = df.describe()['Family_Members']['25%']
q3 = df.describe()['Family_Members']['75%']

print(q1)
print(q3)

IQR=q3-q1
print(IQR)


lower_fence= q1-1.5*IQR
upper_fence= q3+1.5*IQR

print(lower_fence)
print(upper_fence)

2.0
3.0
1.0
0.5
4.5
```

```
df['Family_Members']=df['Family_Members'].clip(lower_fence,upper_fence)

df['Family_Members'].plot(kind='box')
```

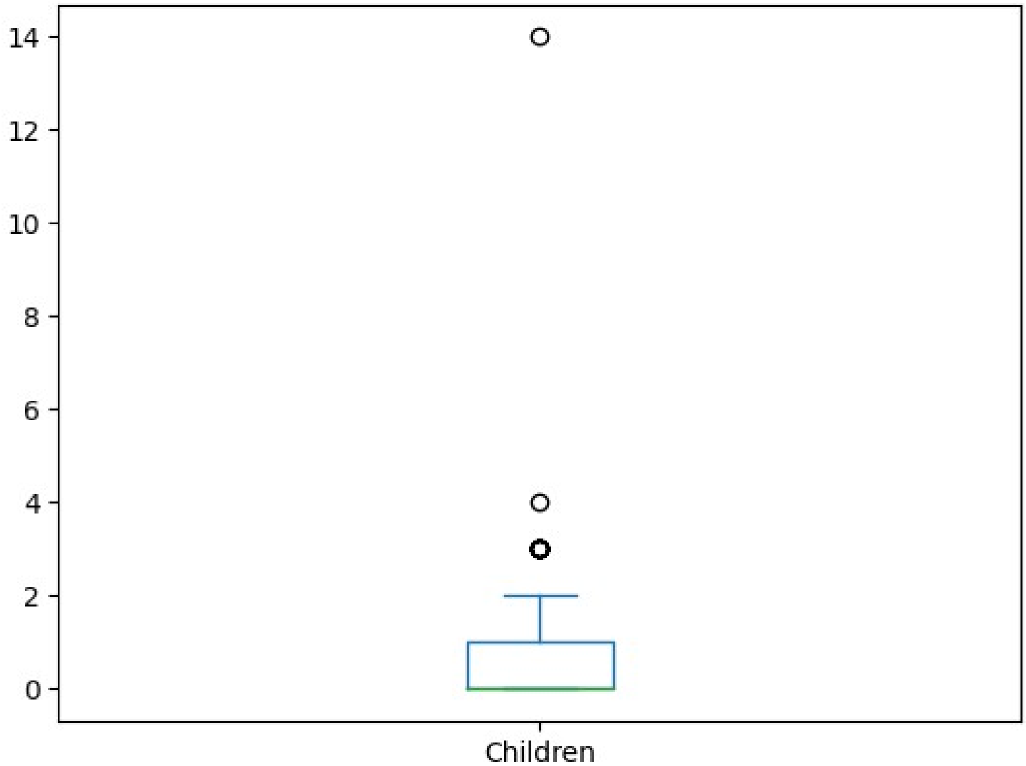
```
<Axes: >
```



Children

```
df['Children'].plot(kind='box')
```

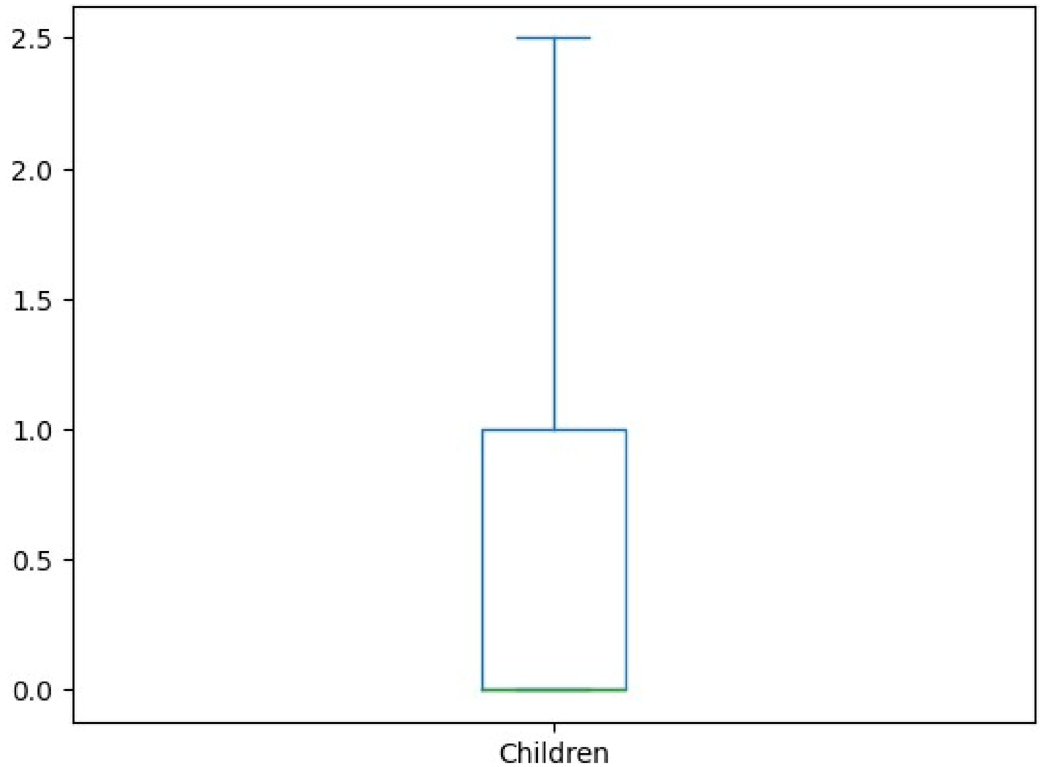
```
<Axes: >
```




```
df.describe()['Children']

q1 = df.describe()['Children']['25%']
q3 = df.describe()['Children']['75%']

print(q1)
print(q3)

IQR=q3-q1
print(IQR)


lower_fence= q1-1.5*IQR
upper_fence= q3+1.5*IQR

print(lower_fence)
print(upper_fence)

0.0
1.0
1.0
-1.5
2.5


df['Children']=df['Children'].clip(lower_fence,upper_fence)
```

```
df['Children'].plot(kind='box')
```

```
<Axes: >
```



3.2 Categorical Encoding

```
# Binary Encoding

# for gender column

df['Gender'] = df['Gender'].map({'M':1,'F':0})

# for Car_Owner Column

df['Car_Owner'] = df['Car_Owner'].map({'Y':1,'N':0})

# for Property_Owner

df['Property_Owner'] = df['Property_Owner'].map({'Y':1,'N':0})

# Education is in order so i am using ordinal encoding or map

df['Education'].value_counts()
```

```
Secondary / secondary special    1031
Higher education                  426
```

```
Incomplete higher                   68
Lower secondary                     21
Academic degree                      2
Name: Education, dtype: int64

# importing ordinal encoder
from sklearn.preprocessing import OrdinalEncoder

# values to ordinal
cols_order = ['Lower secondary','Secondary / secondary
special','Incomplete higher','Higher education', 'Academic degree']

x = OrdinalEncoder(categories=[cols_order])

df['Education'] = x.fit_transform(df[['Education']])

# columns to onehot encoding

df= pd.get_dummies(df,
columns=['Income_Type','Marital_status','Housing_type'],
drop_first=True)

df.sample(5)

      Gender  Car_Owner  Property_Owner  Children  Annual_income
Education  \
194        1          0               1       0.0       126000.0
1.0
519        0          0               1       0.0       135000.0
1.0
835        0          0               1       2.0       103500.0
1.0
1506       0          1               1       0.0       157500.0
1.0
1513       0          0               0       0.0        94500.0
3.0

      Work_Phone  Phone  Email_ID  Family_Members  ...
Income_Type_Working  \
194            0      0         0             2.0  ...
0
519            0      1         0             2.0  ...
0
835            1      1         0             4.0  ...
1
1506           0      0         0             2.0  ...
1
1513           0      0         0             2.0  ...
1

      Marital_status_Married  Marital_status_Separated  \
```

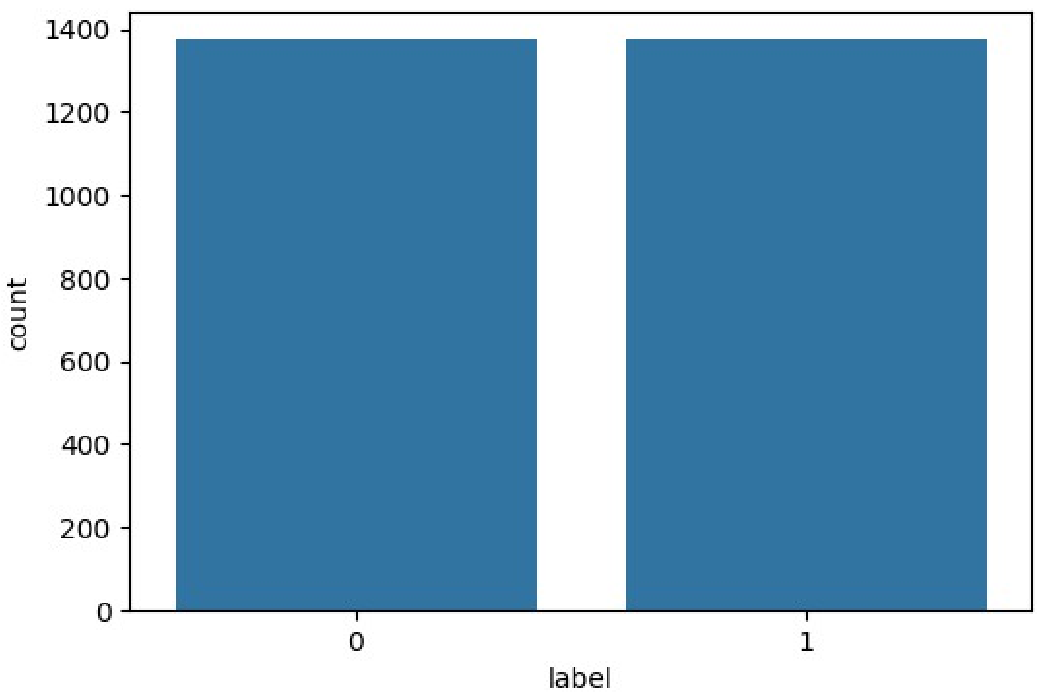
```
194                          1                          0
519                          1                          0
835                          1                          0
1506                         1                          0
1513                         1                          0

      Marital_status_Single / not married  Marital_status_Widow  \
194                                     0                     0
519                                     0                     0
835                                     0                     0
1506                                    0                     0
1513                                    0                     0

      Housing_type_House / apartment  Housing_type_Municipal apartment
\
194                                0                                 0

519                                1                                 0

835                                1                                 0

1506                               1                                 0

1513                               1                                 0


      Housing_type_Office apartment  Housing_type_Rented apartment  \
194                               0                              1
519                               0                              0
835                               0                              0
1506                              0                              0
1513                              0                              0

      Housing_type_With parents
194                           0
519                           0
835                           0
1506                          0
1513                          0

[5 rows x 25 columns]
```

3.3 Feature Splitting

```python
# Separate features and target variable

X = df.drop(columns=['label'])
y = df['label']
```

3.4 Performing SMOTE to handle imbalance in the dataset(target variable)

```
# importing SMOTE
from imblearn.over_sampling import SMOTE

oversample = SMOTE()

X, y = oversample.fit_resample(X, y)

# checking values after applying smote

y.value_counts()

1    1373
0    1373
Name: label, dtype: int64

plt.figure(figsize=(6,4))
sns.countplot(x=y)

<Axes: xlabel='label', ylabel='count'>
```




```
# Required imports
from sklearn.model_selection import train_test_split, cross_val_score,
RandomizedSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
```

```
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score, classification_report, confusion_matrix,
ConfusionMatrixDisplay, PrecisionRecallDisplay
from sklearn.feature_selection import SelectKBest,
SelectPercentile,RFE, SelectFromModel
```

3.5 Performing train-test-split

```
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=45)

# training data

X_train.shape

(2196, 24)

# test data

X_test.shape

(550, 24)
```

3.6 Feature Selection

```
rfe = RFE(estimator=RandomForestClassifier(),n_features_to_select=15,
step=2, verbose=3)
rfe.fit(X_train, y_train)

Fitting estimator with 24 features.
Fitting estimator with 22 features.
Fitting estimator with 20 features.
Fitting estimator with 18 features.
Fitting estimator with 16 features.

RFE(estimator=RandomForestClassifier(), n_features_to_select=15,
step=2,
    verbose=3)
```

3.7 Feature Scaling

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
```

```
X_train_scaled = scaler.fit_transform(X_train)

X_test_scaled = scaler.transform(X_test)
```

4 Model Trainig AND Evalution

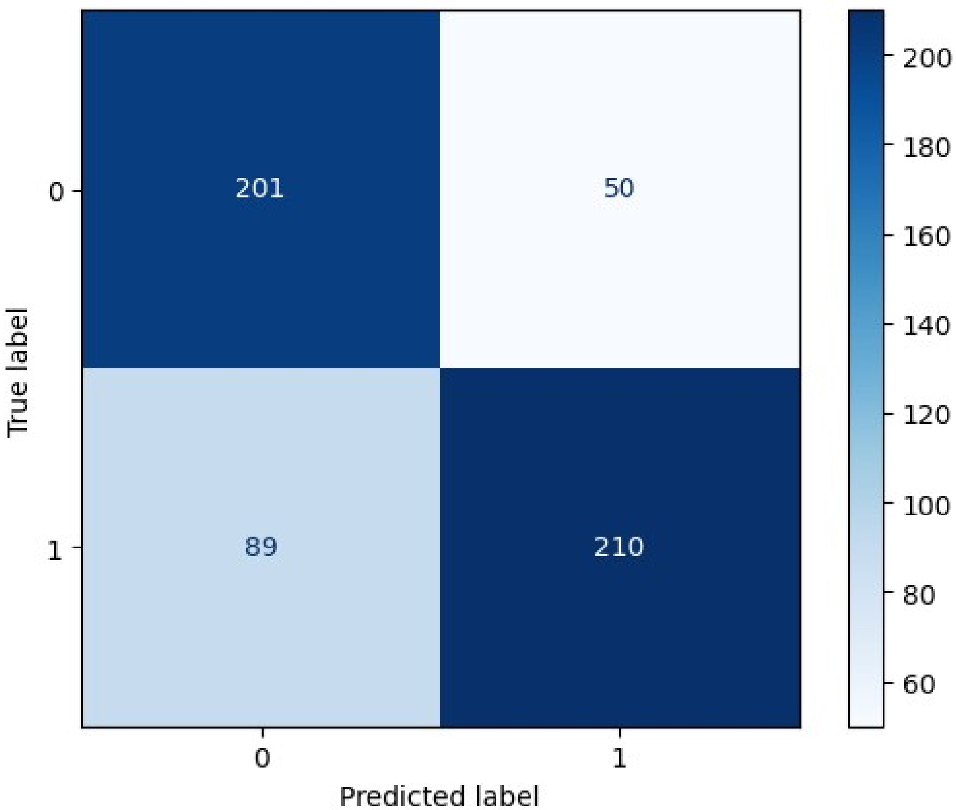
- Creating the list to store all classification_reports of difffernet model.
- This List will help to create a dictionary which at last will represents as Dataframe

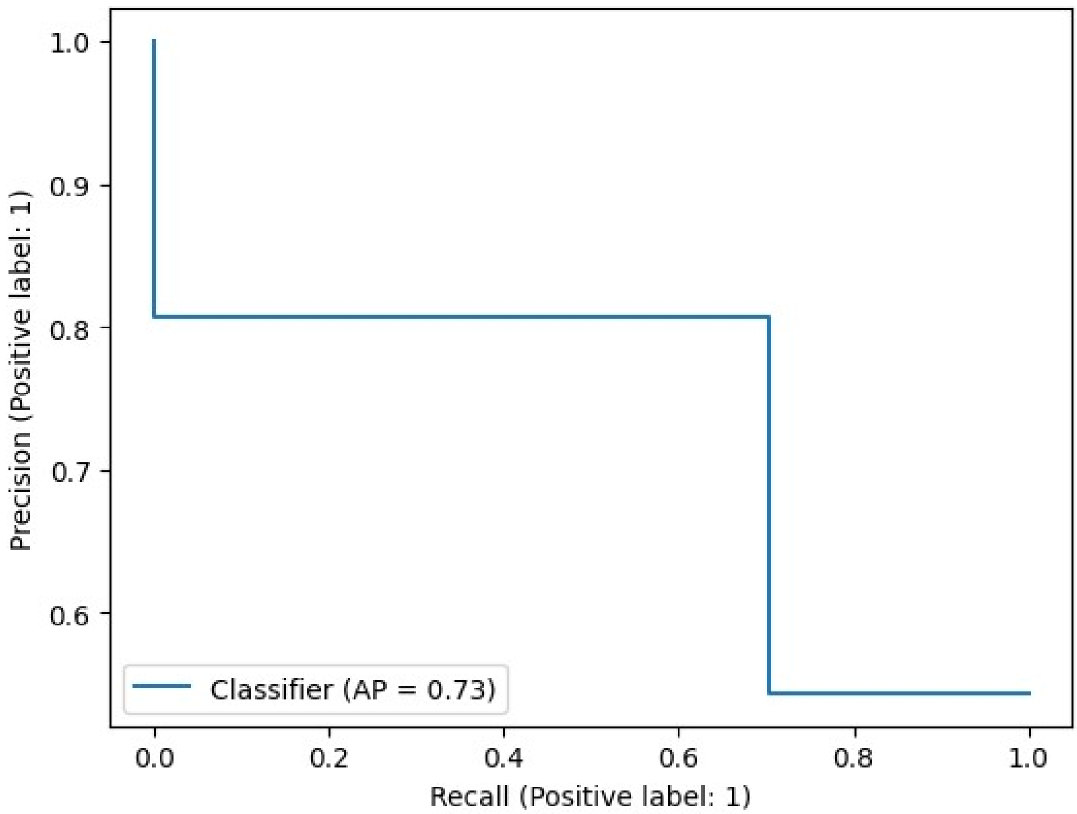
```
models = []
accuracy_scores = []
precision_scores = []
recall_scores = []
f1_scores = []

# creating a function in which model is called and perform all desired
action inside the function

def train_and_evaluate_model(model):
  model.fit(X_train_scaled,y_train)
  y_pred = model.predict(X_test_scaled)
  print("Classification Report of model:")
  print(classification_report(y_test,y_pred))
  print('Accuracy:',accuracy_score(y_test,y_pred))
  ConfusionMatrixDisplay.from_predictions(y_test,y_pred, cmap='Blues')
  PrecisionRecallDisplay.from_predictions(y_test,y_pred)
  accuracy = 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')
  accuracy_scores.append(accuracy)
  precision_scores.append(precision)
  recall_scores.append(recall)
  f1_scores.append(f1)
  models.append(model)
```

4.1 Logistic Regression

```
train_and_evaluate_model(LogisticRegression())

Classification Report of model:
              precision    recall  f1-score   support

           0       0.69      0.80      0.74       251
           1       0.81      0.70      0.75       299

    accuracy                           0.75       550
```

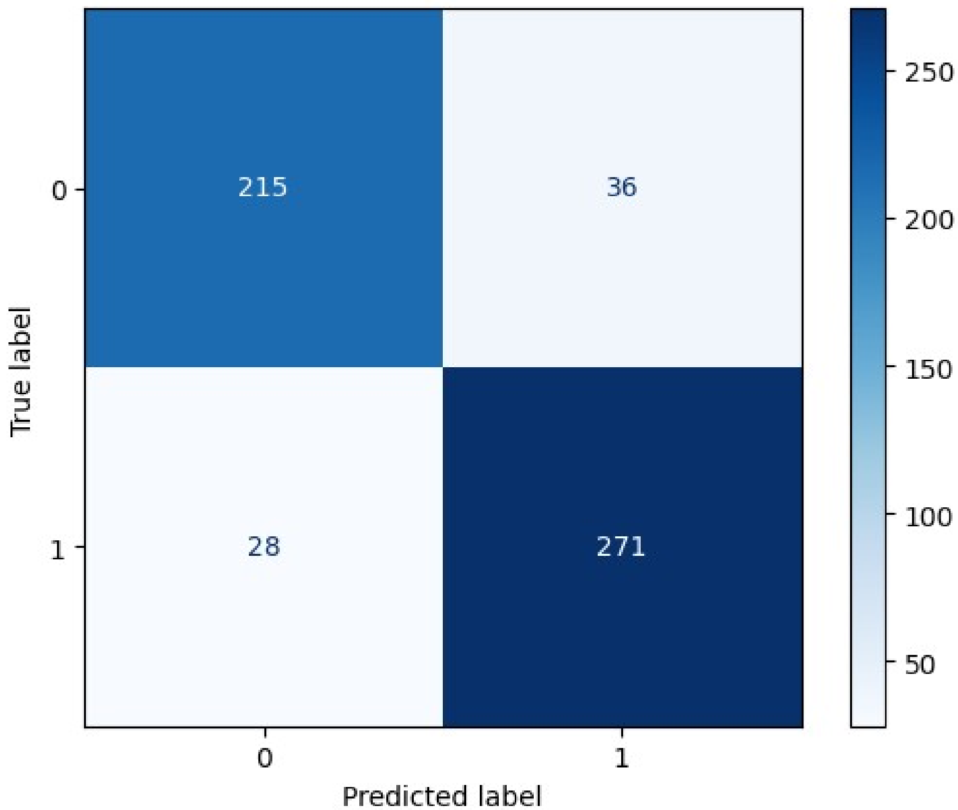
```
   macro avg       0.75      0.75      0.75       550
weighted avg       0.76      0.75      0.75       550

Accuracy: 0.7472727272727273
```



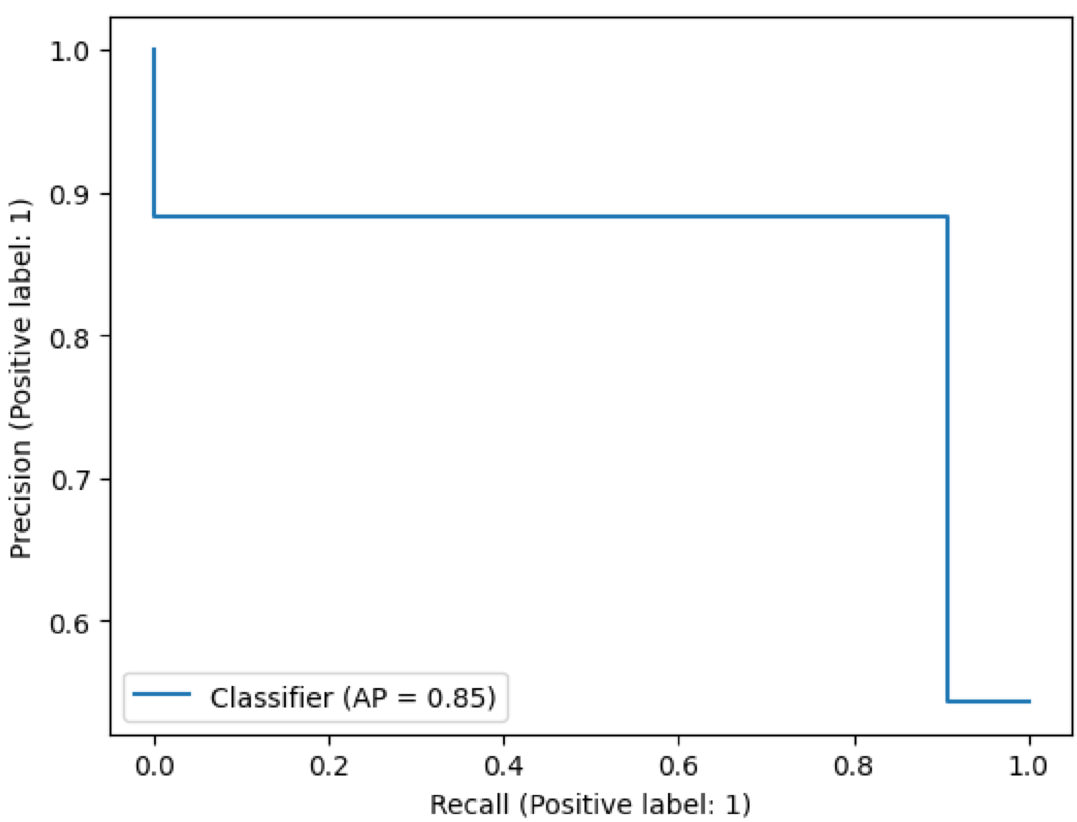


4.2 DecisionTreeClassifier

```
train_and_evaluate_model(DecisionTreeClassifier())

Classification Report of model:
              precision    recall  f1-score   support

           0       0.88      0.86      0.87       251
           1       0.88      0.91      0.89       299

    accuracy                           0.88       550
   macro avg       0.88      0.88      0.88       550
weighted avg       0.88      0.88      0.88       550

Accuracy: 0.8836363636363637
```



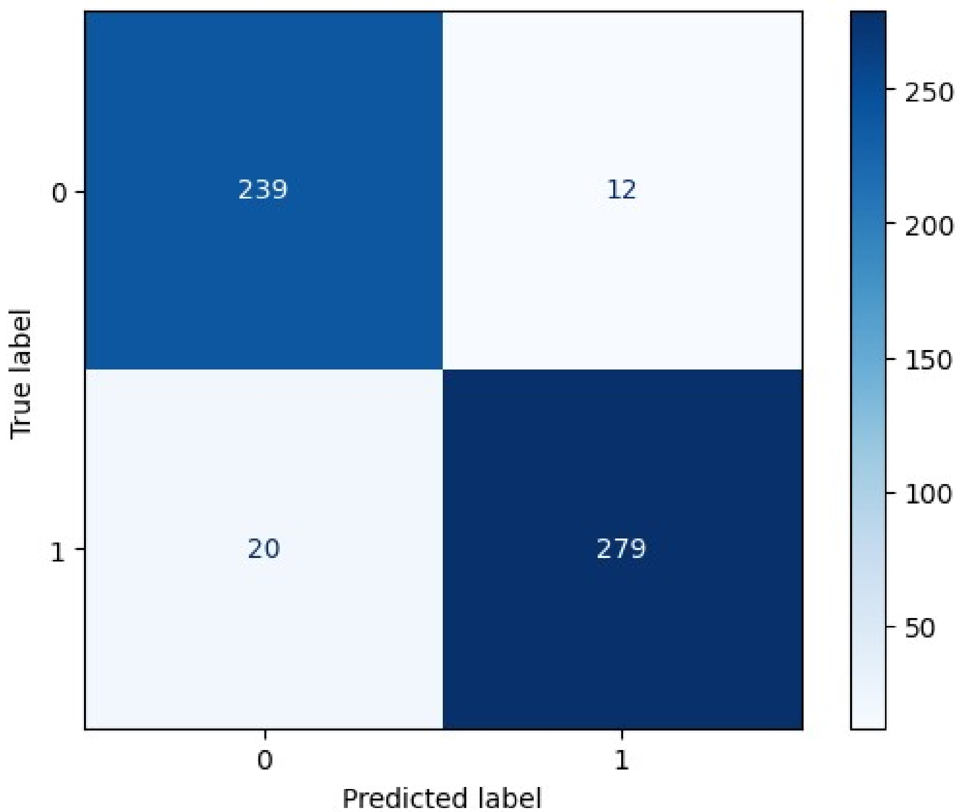
215
36
28
271
0
1
True label
Predicted label
250
200
150
100
50

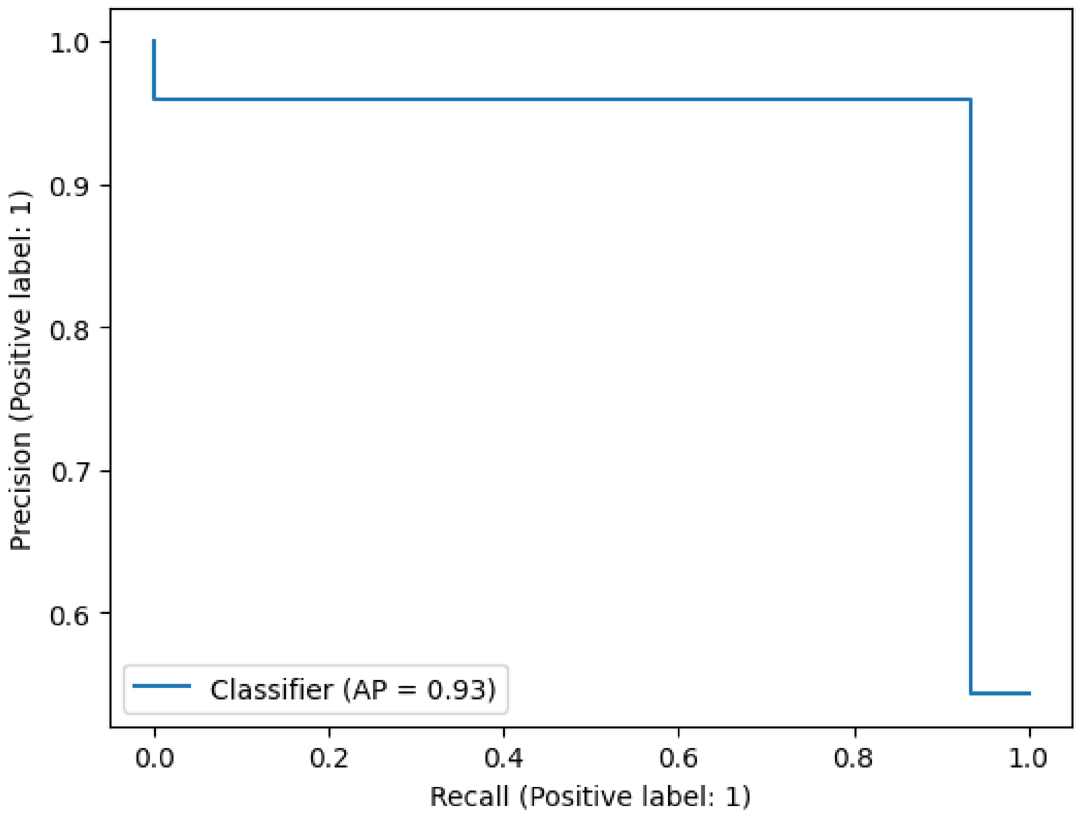


Classifier (AP = 0.85)
Precision (Positive label: 1)
Recall (Positive label: 1)
1.0
0.9
0.8
0.7
0.6
0.0
0.2
0.4
0.6
0.8
1.0

4.3 XGBClassifier

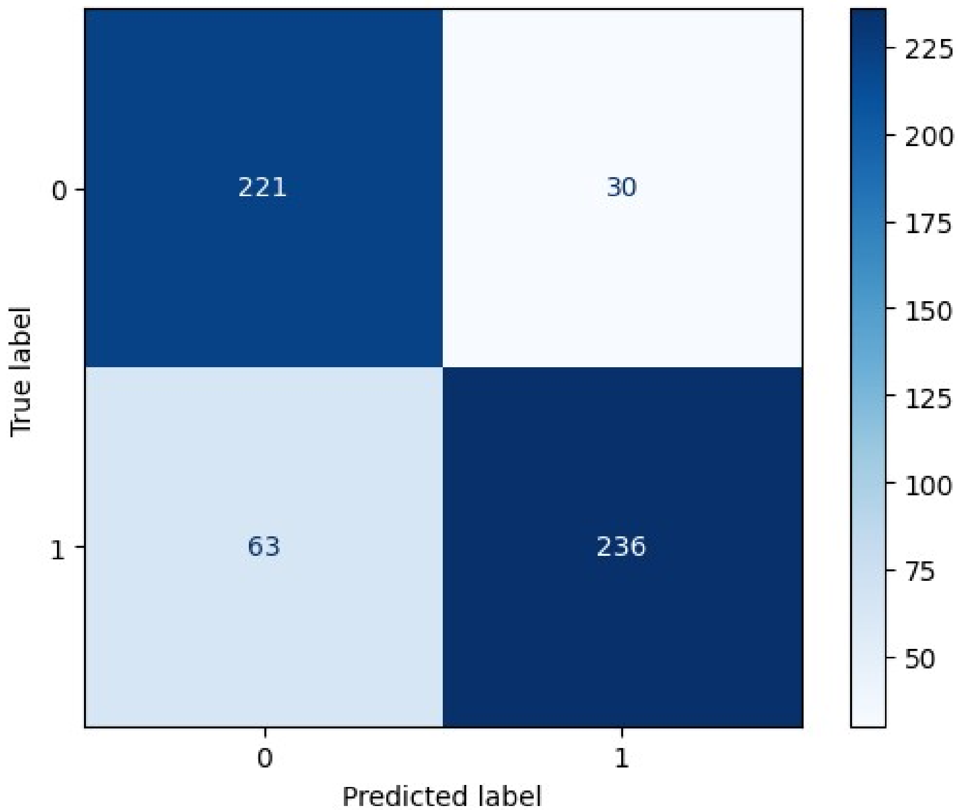
```
train_and_evaluate_model(XGBClassifier())

Classification Report of model:
              precision    recall  f1-score   support

           0       0.92      0.95      0.94       251
           1       0.96      0.93      0.95       299

    accuracy                           0.94       550
   macro avg       0.94      0.94      0.94       550
weighted avg       0.94      0.94      0.94       550

Accuracy: 0.9418181818181818
```

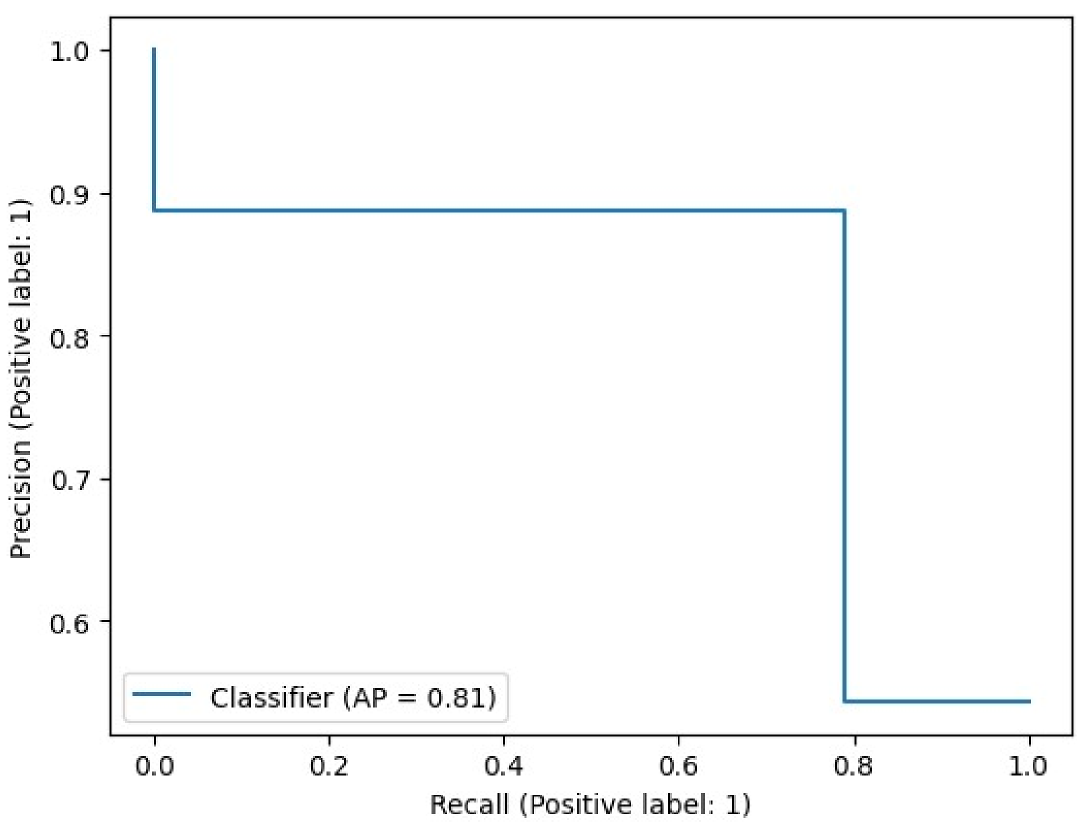




4.4 Suppot vector machine

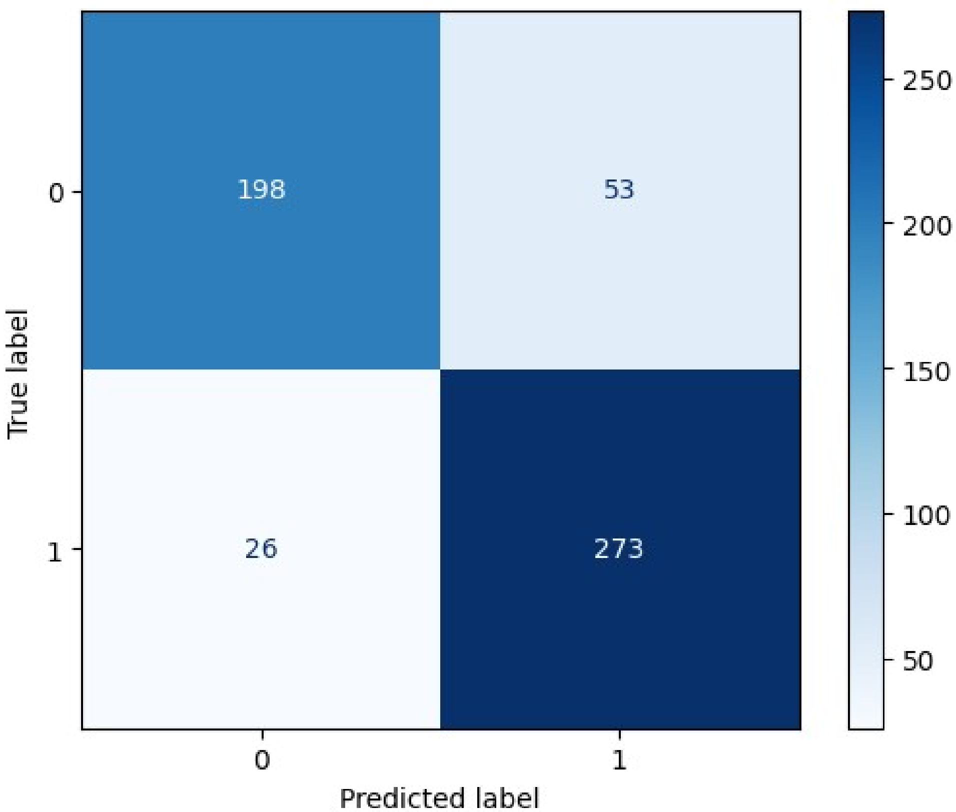
```
train_and_evaluate_model(SVC())

Classification Report of model:
              precision    recall  f1-score   support

           0       0.78      0.88      0.83       251
           1       0.89      0.79      0.84       299

    accuracy                           0.83       550
   macro avg       0.83      0.83      0.83       550
weighted avg       0.84      0.83      0.83       550

Accuracy: 0.8309090909090909
```

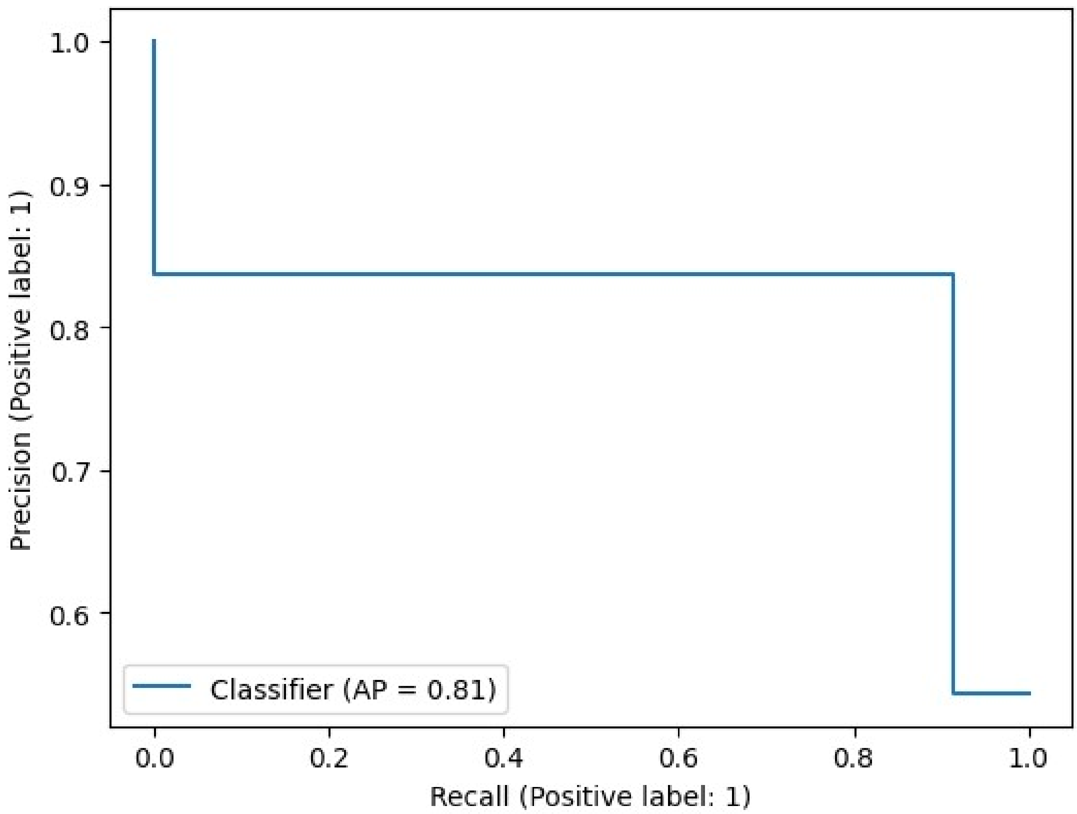

221
30
63
236
0
1
0
1
True label
Predicted label
225
200
175
150
125
100
75
50


1.0
0.9
0.8
0.7
0.6
Precision (Positive label: 1)
Classifier (AP = 0.81)
0.0
0.2
0.4
0.6
0.8
1.0
Recall (Positive label: 1)

4.5 KNN

```
train_and_evaluate_model(KNeighborsClassifier())

Classification Report of model:
              precision    recall  f1-score   support

           0       0.88      0.79      0.83       251
           1       0.84      0.91      0.87       299

    accuracy                           0.86       550
   macro avg       0.86      0.85      0.85       550
weighted avg       0.86      0.86      0.86       550

Accuracy: 0.8563636363636363
```





5 Model Selection

```
model_compare = pd.DataFrame({'Model': models,
                              'Accuracy': accuracy_scores,
                              'Precision': precision_scores,
                              'Recall': recall_scores,
                              'F1':
f1_scores}).sort_values('Accuracy',ascending=False)

model_compare

                                               Model  Accuracy
Precision  \
2  XGBClassifier(base_score=None, booster=None, c...  0.941818
0.940771
1                           DecisionTreeClassifier()  0.883636
0.883755
4                             KNeighborsClassifier()  0.856364
0.860676
3                                              SVC()  0.830909
0.832694
0                               LogisticRegression()  0.747273
0.750398
```

```
      Recall        F1
2  0.942651  0.941509
1  0.881464  0.882417
4  0.850944  0.853642
3  0.834888  0.830783
0  0.751569  0.747205
```

```
best_model = model_compare.iloc[0]['Model']
best_model
```

```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None,
early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None,
feature_types=None,
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None,
max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan,
monotone_constraints=None,
              multi_strategy=None, n_estimators=None, n_jobs=None,
              num_parallel_tree=None, random_state=None, ...)
```

--> I have selected XGBOOST model bez it give superior performance in terms of accuracy, precision, and recall when compared to other models.

--> Interpretability and Feature Importance: XGBoost provides clear insights into the significance of different features in making predictions

The Best ML Model is XGB Classifier

cross validation for best model

```
avg_cv_scores = cross_val_score(best_model, X_test_scaled, y_test,
scoring ='accuracy', cv=5, verbose=2)
mean_score = round(np.mean(avg_cv_scores),4)
print(f'Mean cross Validation Performance: {mean_score*100}%')
```

```
[CV] END .................................................... total
time=   0.1s
[CV] END .................................................... total
time=   0.1s
[CV] END .................................................... total
time=   0.1s
[CV] END .................................................... total
time=   0.1s
```

```
[CV] END .................................................... total
time=   0.1s
Mean cross Validation Performance: 85.27%
```

5.1 Hyperparameter Tuning (to imporve or enhance model performance)

5.1.1 Logistic Regression

```python
# Tuning for logistic Regression

param_grid = {
    'penalty': ['l1', 'l2'],
    'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000]
}

grid_lr = RandomizedSearchCV(LogisticRegression(), param_grid,
verbose=3, cv=4)

train_and_evaluate_model(grid_lr)
```

```
Fitting 4 folds for each of 10 candidates, totalling 40 fits
[CV 1/4] END ....................C=10, penalty=l1;, score=nan total
time=   0.0s
[CV 2/4] END ....................C=10, penalty=l1;, score=nan total
time=   0.0s
[CV 3/4] END ....................C=10, penalty=l1;, score=nan total
time=   0.0s
[CV 4/4] END ....................C=10, penalty=l1;, score=nan total
time=   0.0s
[CV 1/4] END .................C=0.1, penalty=l2;, score=0.767 total
time=   0.0s
[CV 2/4] END .................C=0.1, penalty=l2;, score=0.787 total
time=   0.0s
[CV 3/4] END .................C=0.1, penalty=l2;, score=0.767 total
time=   0.0s
[CV 4/4] END .................C=0.1, penalty=l2;, score=0.760 total
time=   0.0s
[CV 1/4] END ...................C=100, penalty=l1;, score=nan total
time=   0.0s
[CV 2/4] END ...................C=100, penalty=l1;, score=nan total
time=   0.0s
[CV 3/4] END ...................C=100, penalty=l1;, score=nan total
time=   0.0s
[CV 4/4] END ...................C=100, penalty=l1;, score=nan total
time=   0.0s
[CV 1/4] END ..................C=1000, penalty=l1;, score=nan total
time=   0.0s
[CV 2/4] END ..................C=1000, penalty=l1;, score=nan total
```

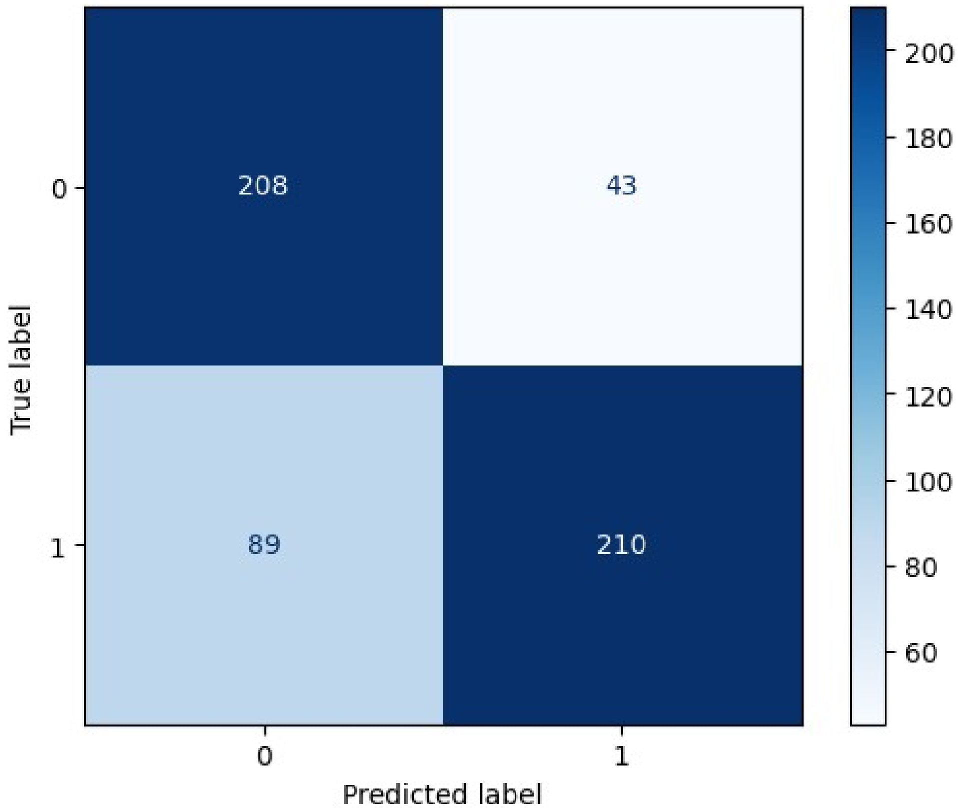
```
time=   0.0s
[CV 3/4] END ..................C=1000, penalty=l1;, score=nan total
time=   0.0s
[CV 4/4] END ..................C=1000, penalty=l1;, score=nan total
time=   0.0s
[CV 1/4] END .....................C=1, penalty=l1;, score=nan total
time=   0.0s
[CV 2/4] END .....................C=1, penalty=l1;, score=nan total
time=   0.0s
[CV 3/4] END .....................C=1, penalty=l1;, score=nan total
time=   0.0s
[CV 4/4] END .....................C=1, penalty=l1;, score=nan total
time=   0.0s
[CV 1/4] END ................C=1000, penalty=l2;, score=0.792 total
time=   0.0s
[CV 2/4] END ................C=1000, penalty=l2;, score=0.792 total
time=   0.0s

/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/
_logistic.py:458: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic
.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic
.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
```

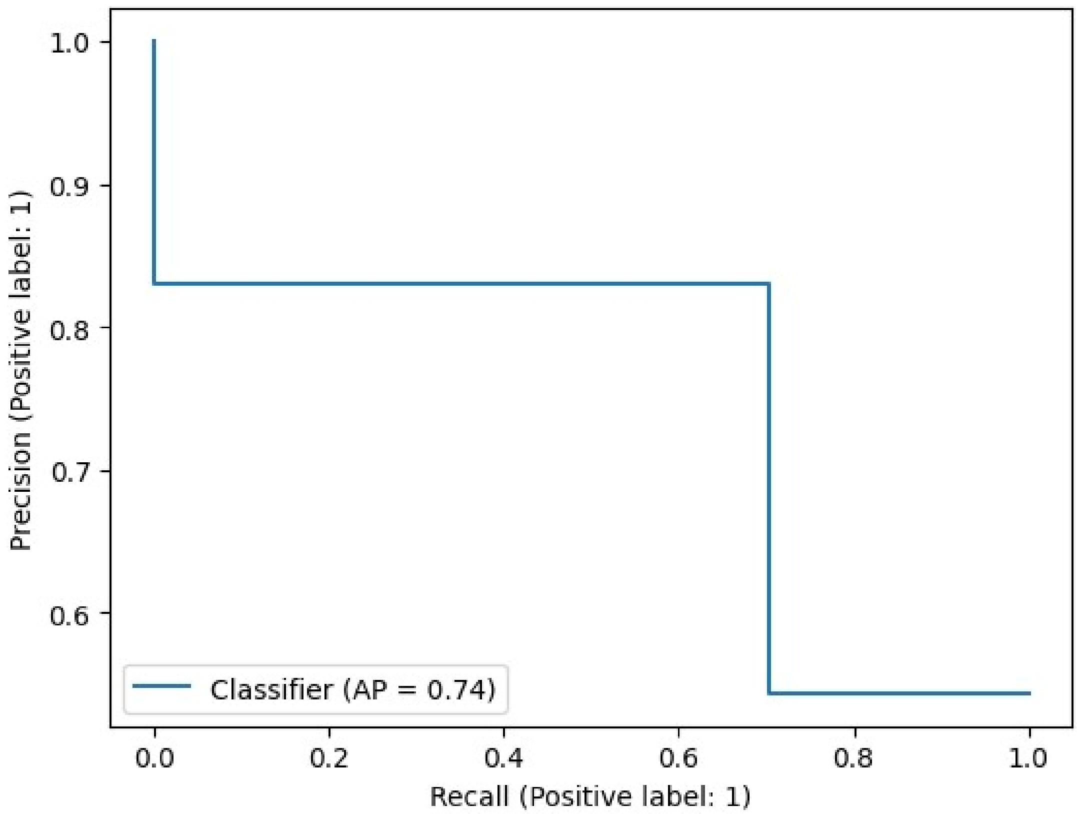
```
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic
.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic
.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(

[CV 3/4] END ................C=1000, penalty=l2;, score=0.781 total
time=   0.0s
[CV 4/4] END ................C=1000, penalty=l2;, score=0.789 total
time=   0.0s
[CV 1/4] END .................C=100, penalty=l2;, score=0.796 total
time=   0.0s
[CV 2/4] END .................C=100, penalty=l2;, score=0.791 total
time=   0.0s
[CV 3/4] END .................C=100, penalty=l2;, score=0.781 total
time=   0.0s

/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/
_logistic.py:458: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
```

```
https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic
.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_valid
ation.py:378: FitFailedWarning:
20 fits failed out of a total of 40.
The score on these train-test partitions for these parameters will be
set to nan.
If these failures are not expected, you can try to debug them by
setting error_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------
----------
20 fits failed with the following error:
Traceback (most recent call last):
  File
"/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_vali
dation.py", line 686, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File
"/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logisti
c.py", line 1162, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File
"/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logisti
c.py", line 54, in _check_solver
    raise ValueError(
ValueError: Solver lbfgs supports only 'l2' or 'none' penalties, got
l1 penalty.

  warnings.warn(some_fits_failed_message, FitFailedWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_searc
h.py:952: UserWarning: One or more of the test scores are non-finite:
[       nan 0.77003643        nan        nan        nan 0.78870674
 0.78870674        nan 0.73770492 0.71584699]
  warnings.warn(
```

```
[CV 4/4] END .................C=100, penalty=l2;, score=0.787 total
time=   0.1s
[CV 1/4] END ..................C=0.01, penalty=l1;, score=nan total
time=   0.0s
[CV 2/4] END ..................C=0.01, penalty=l1;, score=nan total
time=   0.0s
[CV 3/4] END ..................C=0.01, penalty=l1;, score=nan total
time=   0.0s
[CV 4/4] END ..................C=0.01, penalty=l1;, score=nan total
time=   0.0s
[CV 1/4] END ................C=0.01, penalty=l2;, score=0.712 total
time=   0.0s
[CV 2/4] END ................C=0.01, penalty=l2;, score=0.769 total
time=   0.0s
[CV 3/4] END ................C=0.01, penalty=l2;, score=0.736 total
time=   0.0s
[CV 4/4] END ................C=0.01, penalty=l2;, score=0.734 total
time=   0.0s
[CV 1/4] END ...............C=0.001, penalty=l2;, score=0.689 total
time=   0.0s
[CV 2/4] END ...............C=0.001, penalty=l2;, score=0.747 total
time=   0.0s
[CV 3/4] END ...............C=0.001, penalty=l2;, score=0.721 total
time=   0.0s
[CV 4/4] END ...............C=0.001, penalty=l2;, score=0.707 total
time=   0.0s
Classification Report of model:
              precision    recall  f1-score   support

           0       0.70      0.83      0.76       251
           1       0.83      0.70      0.76       299

    accuracy                           0.76       550
   macro avg       0.77      0.77      0.76       550
weighted avg       0.77      0.76      0.76       550

Accuracy: 0.76

/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/
_logistic.py:458: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
```

```
regression
  n_iter_i = _check_optimize_result(
```





After tuning, there is increase in accuracy to 0.76

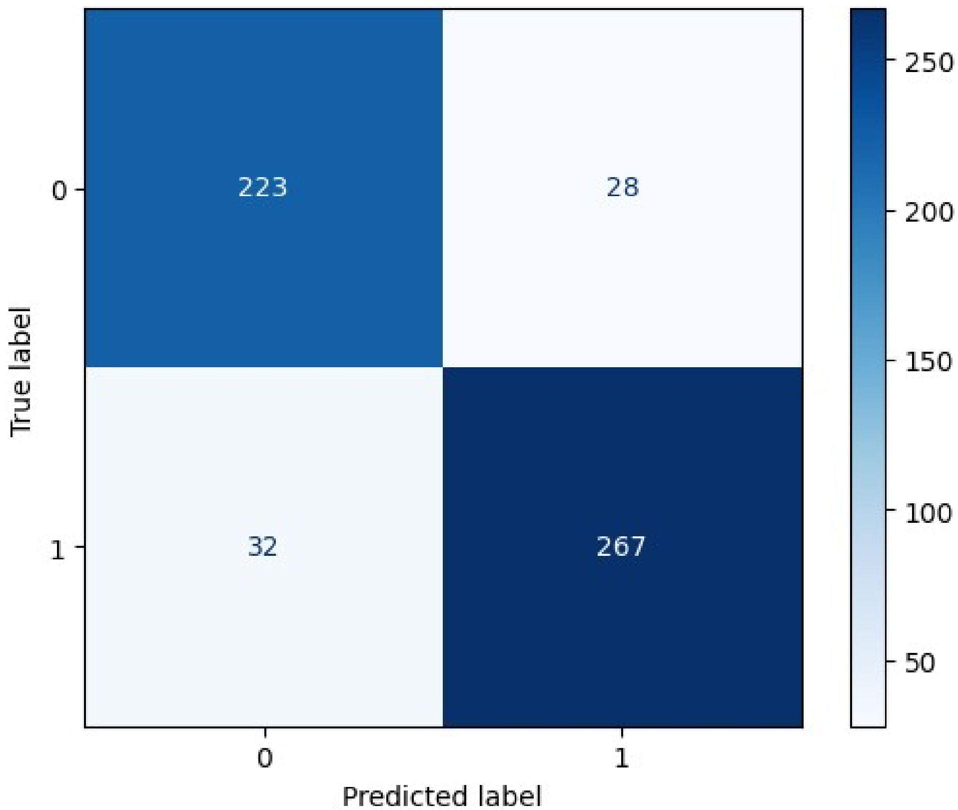
##DecisionTreeClassifier

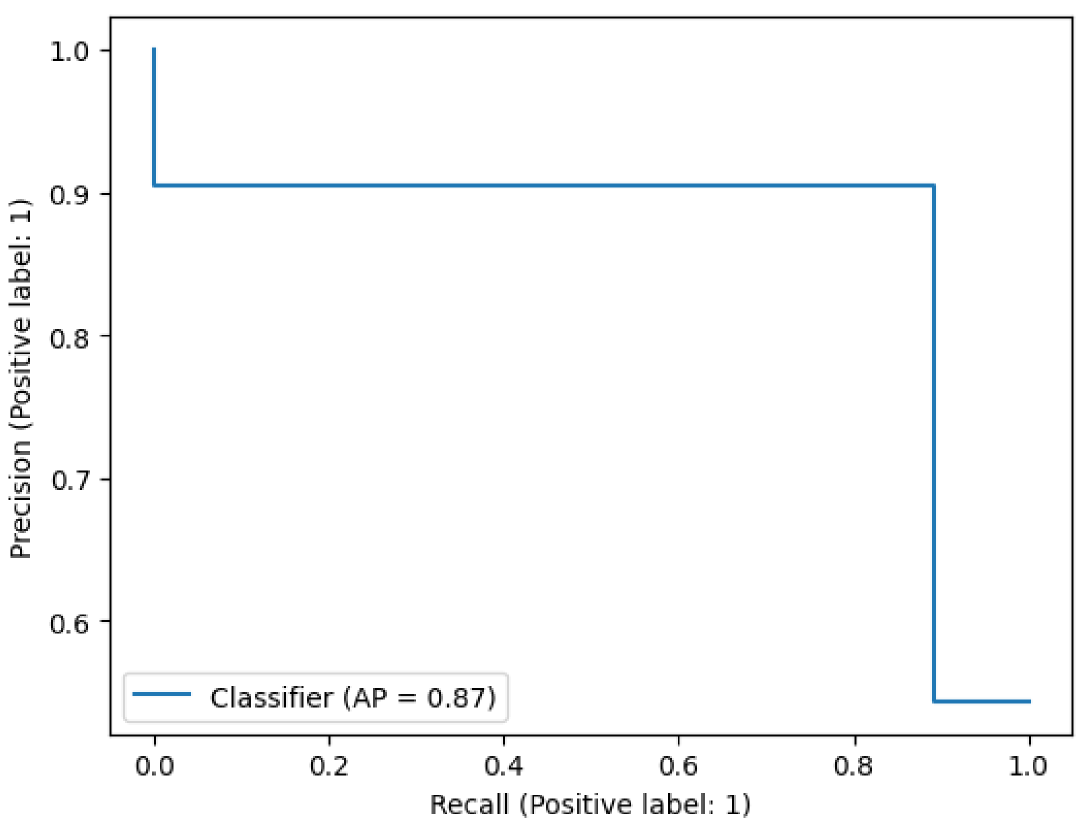
```
# Tuning for DecisionTreeClassifier

param_grid = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 10, 20, 30, 40, 50],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

grid_dt = RandomizedSearchCV(DecisionTreeClassifier(), param_grid,
verbose=3, cv=4)

train_and_evaluate_model(grid_dt)
```

```
Fitting 4 folds for each of 10 candidates, totalling 40 fits
[CV 1/4] END criterion=gini, max_depth=40, min_samples_leaf=2,
min_samples_split=2;, score=0.876 total time=   0.0s
[CV 2/4] END criterion=gini, max_depth=40, min_samples_leaf=2,
min_samples_split=2;, score=0.874 total time=   0.0s
[CV 3/4] END criterion=gini, max_depth=40, min_samples_leaf=2,
min_samples_split=2;, score=0.851 total time=   0.0s
```

```
[CV 4/4] END criterion=gini, max_depth=40, min_samples_leaf=2,
min_samples_split=2;, score=0.891 total time=   0.0s
[CV 1/4] END criterion=entropy, max_depth=20, min_samples_leaf=4,
min_samples_split=10;, score=0.851 total time=   0.0s
[CV 2/4] END criterion=entropy, max_depth=20, min_samples_leaf=4,
min_samples_split=10;, score=0.889 total time=   0.0s
[CV 3/4] END criterion=entropy, max_depth=20, min_samples_leaf=4,
min_samples_split=10;, score=0.842 total time=   0.0s
[CV 4/4] END criterion=entropy, max_depth=20, min_samples_leaf=4,
min_samples_split=10;, score=0.874 total time=   0.0s
[CV 1/4] END criterion=gini, max_depth=10, min_samples_leaf=2,
min_samples_split=10;, score=0.834 total time=   0.0s
[CV 2/4] END criterion=gini, max_depth=10, min_samples_leaf=2,
min_samples_split=10;, score=0.847 total time=   0.0s
[CV 3/4] END criterion=gini, max_depth=10, min_samples_leaf=2,
min_samples_split=10;, score=0.821 total time=   0.0s
[CV 4/4] END criterion=gini, max_depth=10, min_samples_leaf=2,
min_samples_split=10;, score=0.847 total time=   0.0s
[CV 1/4] END criterion=entropy, max_depth=30, min_samples_leaf=4,
min_samples_split=10;, score=0.858 total time=   0.0s
[CV 2/4] END criterion=entropy, max_depth=30, min_samples_leaf=4,
min_samples_split=10;, score=0.889 total time=   0.0s
[CV 3/4] END criterion=entropy, max_depth=30, min_samples_leaf=4,
min_samples_split=10;, score=0.851 total time=   0.0s
[CV 4/4] END criterion=entropy, max_depth=30, min_samples_leaf=4,
min_samples_split=10;, score=0.869 total time=   0.0s
[CV 1/4] END criterion=gini, max_depth=10, min_samples_leaf=1,
min_samples_split=5;, score=0.851 total time=   0.0s
[CV 2/4] END criterion=gini, max_depth=10, min_samples_leaf=1,
min_samples_split=5;, score=0.856 total time=   0.0s
[CV 3/4] END criterion=gini, max_depth=10, min_samples_leaf=1,
min_samples_split=5;, score=0.823 total time=   0.0s
[CV 4/4] END criterion=gini, max_depth=10, min_samples_leaf=1,
min_samples_split=5;, score=0.851 total time=   0.0s
[CV 1/4] END criterion=gini, max_depth=50, min_samples_leaf=1,
min_samples_split=5;, score=0.874 total time=   0.0s
[CV 2/4] END criterion=gini, max_depth=50, min_samples_leaf=1,
min_samples_split=5;, score=0.883 total time=   0.0s
[CV 3/4] END criterion=gini, max_depth=50, min_samples_leaf=1,
min_samples_split=5;, score=0.852 total time=   0.0s
[CV 4/4] END criterion=gini, max_depth=50, min_samples_leaf=1,
min_samples_split=5;, score=0.887 total time=   0.0s
[CV 1/4] END criterion=entropy, max_depth=50, min_samples_leaf=1,
min_samples_split=10;, score=0.851 total time=   0.0s
[CV 2/4] END criterion=entropy, max_depth=50, min_samples_leaf=1,
min_samples_split=10;, score=0.900 total time=   0.0s
[CV 3/4] END criterion=entropy, max_depth=50, min_samples_leaf=1,
min_samples_split=10;, score=0.843 total time=   0.0s
[CV 4/4] END criterion=entropy, max_depth=50, min_samples_leaf=1,
```

```
min_samples_split=10;, score=0.874 total time=   0.0s
[CV 1/4] END criterion=gini, max_depth=20, min_samples_leaf=2,
min_samples_split=5;, score=0.876 total time=   0.0s
[CV 2/4] END criterion=gini, max_depth=20, min_samples_leaf=2,
min_samples_split=5;, score=0.860 total time=   0.0s
[CV 3/4] END criterion=gini, max_depth=20, min_samples_leaf=2,
min_samples_split=5;, score=0.838 total time=   0.0s
[CV 4/4] END criterion=gini, max_depth=20, min_samples_leaf=2,
min_samples_split=5;, score=0.893 total time=   0.0s
[CV 1/4] END criterion=gini, max_depth=40, min_samples_leaf=2,
min_samples_split=5;, score=0.882 total time=   0.0s
[CV 2/4] END criterion=gini, max_depth=40, min_samples_leaf=2,
min_samples_split=5;, score=0.865 total time=   0.0s
[CV 3/4] END criterion=gini, max_depth=40, min_samples_leaf=2,
min_samples_split=5;, score=0.845 total time=   0.0s
[CV 4/4] END criterion=gini, max_depth=40, min_samples_leaf=2,
min_samples_split=5;, score=0.883 total time=   0.0s
[CV 1/4] END criterion=gini, max_depth=10, min_samples_leaf=4,
min_samples_split=10;, score=0.838 total time=   0.0s
[CV 2/4] END criterion=gini, max_depth=10, min_samples_leaf=4,
min_samples_split=10;, score=0.858 total time=   0.0s
[CV 3/4] END criterion=gini, max_depth=10, min_samples_leaf=4,
min_samples_split=10;, score=0.805 total time=   0.0s
[CV 4/4] END criterion=gini, max_depth=10, min_samples_leaf=4,
min_samples_split=10;, score=0.840 total time=   0.0s
Classification Report of model:
              precision    recall  f1-score   support

           0       0.87      0.89      0.88       251
           1       0.91      0.89      0.90       299

    accuracy                           0.89       550
   macro avg       0.89      0.89      0.89       550
weighted avg       0.89      0.89      0.89       550

Accuracy: 0.8909090909090909
```


223
28
32
267
0
1
True label
0
1
Predicted label
250
200
150
100
50


1.0
0.9
0.8
0.7
0.6
Precision (Positive label: 1)
Classifier (AP = 0.87)
0.0
0.2
0.4
0.6
0.8
1.0
Recall (Positive label: 1)

After tuning, there is increase in accuracy to 0.89

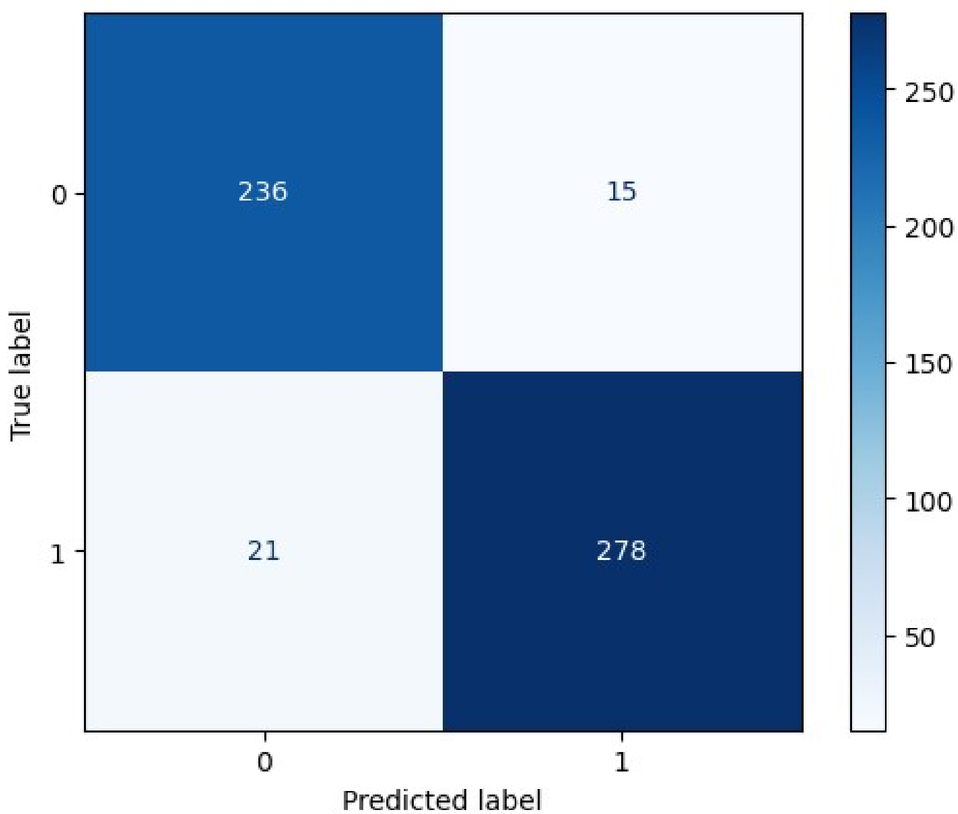
XGBOOST CLASSIFIER

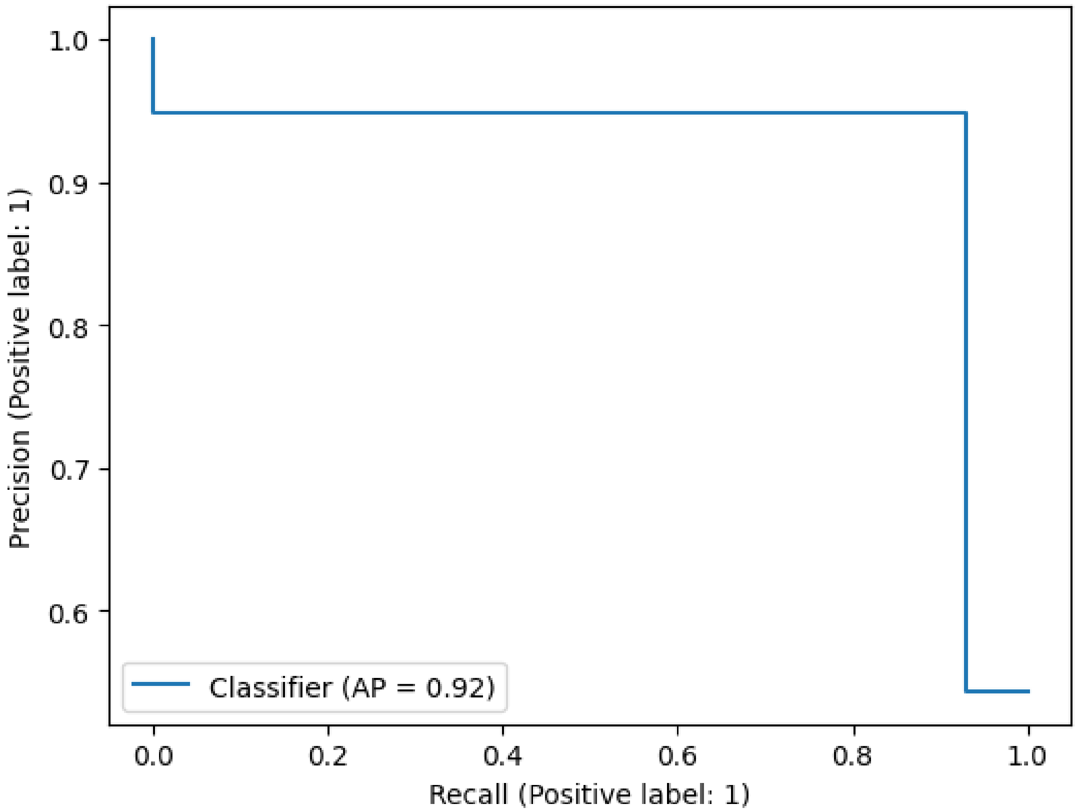
```
# Tuning for XGBOOST CLASSIFIER

param_grid = {
    'learning_rate': [0.01, 0.1, 0.2],
    'max_depth': [3, 5, 7],
    'subsample': [0.8, 1.0],
    'colsample_bytree': [0.8, 1.0],
    'n_estimators': [100, 200, 300]
}

grid_XG = RandomizedSearchCV(XGBClassifier(), param_grid, verbose=3,
cv=4)

train_and_evaluate_model(grid_XG)
```

```
Fitting 4 folds for each of 10 candidates, totalling 40 fits
[CV 1/4] END colsample_bytree=0.8, learning_rate=0.01, max_depth=7,
n_estimators=100, subsample=0.8;, score=0.920 total time=   0.2s
[CV 2/4] END colsample_bytree=0.8, learning_rate=0.01, max_depth=7,
n_estimators=100, subsample=0.8;, score=0.918 total time=   0.1s
[CV 3/4] END colsample_bytree=0.8, learning_rate=0.01, max_depth=7,
n_estimators=100, subsample=0.8;, score=0.863 total time=   0.1s
[CV 4/4] END colsample_bytree=0.8, learning_rate=0.01, max_depth=7,
n_estimators=100, subsample=0.8;, score=0.905 total time=   0.1s
[CV 1/4] END colsample_bytree=1.0, learning_rate=0.1, max_depth=3,
n_estimators=100, subsample=1.0;, score=0.887 total time=   0.1s
[CV 2/4] END colsample_bytree=1.0, learning_rate=0.1, max_depth=3,
n_estimators=100, subsample=1.0;, score=0.902 total time=   0.1s
[CV 3/4] END colsample_bytree=1.0, learning_rate=0.1, max_depth=3,
n_estimators=100, subsample=1.0;, score=0.871 total time=   0.1s
[CV 4/4] END colsample_bytree=1.0, learning_rate=0.1, max_depth=3,
n_estimators=100, subsample=1.0;, score=0.891 total time=   0.1s
[CV 1/4] END colsample_bytree=1.0, learning_rate=0.2, max_depth=3,
n_estimators=100, subsample=0.8;, score=0.909 total time=   0.1s
[CV 2/4] END colsample_bytree=1.0, learning_rate=0.2, max_depth=3,
n_estimators=100, subsample=0.8;, score=0.922 total time=   0.1s
[CV 3/4] END colsample_bytree=1.0, learning_rate=0.2, max_depth=3,
n_estimators=100, subsample=0.8;, score=0.880 total time=   1.2s
[CV 4/4] END colsample_bytree=1.0, learning_rate=0.2, max_depth=3,
n_estimators=100, subsample=0.8;, score=0.903 total time=   0.3s
[CV 1/4] END colsample_bytree=0.8, learning_rate=0.2, max_depth=3,
n_estimators=100, subsample=0.8;, score=0.909 total time=   0.3s
[CV 2/4] END colsample_bytree=0.8, learning_rate=0.2, max_depth=3,
n_estimators=100, subsample=0.8;, score=0.920 total time=   0.2s
[CV 3/4] END colsample_bytree=0.8, learning_rate=0.2, max_depth=3,
n_estimators=100, subsample=0.8;, score=0.893 total time=   0.1s
```

```
[CV 4/4] END colsample_bytree=0.8, learning_rate=0.2, max_depth=3,
n_estimators=100, subsample=0.8;, score=0.909 total time=   0.1s
[CV 1/4] END colsample_bytree=1.0, learning_rate=0.2, max_depth=3,
n_estimators=300, subsample=0.8;, score=0.931 total time=   0.2s
[CV 2/4] END colsample_bytree=1.0, learning_rate=0.2, max_depth=3,
n_estimators=300, subsample=0.8;, score=0.951 total time=   0.1s
[CV 3/4] END colsample_bytree=1.0, learning_rate=0.2, max_depth=3,
n_estimators=300, subsample=0.8;, score=0.918 total time=   0.1s
[CV 4/4] END colsample_bytree=1.0, learning_rate=0.2, max_depth=3,
n_estimators=300, subsample=0.8;, score=0.916 total time=   0.1s
[CV 1/4] END colsample_bytree=0.8, learning_rate=0.1, max_depth=5,
n_estimators=100, subsample=0.8;, score=0.923 total time=   0.1s
[CV 2/4] END colsample_bytree=0.8, learning_rate=0.1, max_depth=5,
n_estimators=100, subsample=0.8;, score=0.929 total time=   0.1s
[CV 3/4] END colsample_bytree=0.8, learning_rate=0.1, max_depth=5,
n_estimators=100, subsample=0.8;, score=0.907 total time=   0.1s
[CV 4/4] END colsample_bytree=0.8, learning_rate=0.1, max_depth=5,
n_estimators=100, subsample=0.8;, score=0.927 total time=   0.1s
[CV 1/4] END colsample_bytree=0.8, learning_rate=0.1, max_depth=5,
n_estimators=300, subsample=0.8;, score=0.949 total time=   0.2s
[CV 2/4] END colsample_bytree=0.8, learning_rate=0.1, max_depth=5,
n_estimators=300, subsample=0.8;, score=0.953 total time=   0.2s
[CV 3/4] END colsample_bytree=0.8, learning_rate=0.1, max_depth=5,
n_estimators=300, subsample=0.8;, score=0.931 total time=   0.2s
[CV 4/4] END colsample_bytree=0.8, learning_rate=0.1, max_depth=5,
n_estimators=300, subsample=0.8;, score=0.929 total time=   0.2s
[CV 1/4] END colsample_bytree=1.0, learning_rate=0.01, max_depth=3,
n_estimators=300, subsample=1.0;, score=0.823 total time=   0.1s
[CV 2/4] END colsample_bytree=1.0, learning_rate=0.01, max_depth=3,
n_estimators=300, subsample=1.0;, score=0.825 total time=   0.1s
[CV 3/4] END colsample_bytree=1.0, learning_rate=0.01, max_depth=3,
n_estimators=300, subsample=1.0;, score=0.829 total time=   0.1s
[CV 4/4] END colsample_bytree=1.0, learning_rate=0.01, max_depth=3,
n_estimators=300, subsample=1.0;, score=0.849 total time=   0.2s
[CV 1/4] END colsample_bytree=1.0, learning_rate=0.2, max_depth=5,
n_estimators=100, subsample=0.8;, score=0.938 total time=   0.1s
[CV 2/4] END colsample_bytree=1.0, learning_rate=0.2, max_depth=5,
n_estimators=100, subsample=0.8;, score=0.945 total time=   0.1s
[CV 3/4] END colsample_bytree=1.0, learning_rate=0.2, max_depth=5,
n_estimators=100, subsample=0.8;, score=0.925 total time=   0.1s
[CV 4/4] END colsample_bytree=1.0, learning_rate=0.2, max_depth=5,
n_estimators=100, subsample=0.8;, score=0.933 total time=   0.1s
[CV 1/4] END colsample_bytree=0.8, learning_rate=0.01, max_depth=5,
n_estimators=200, subsample=1.0;, score=0.887 total time=   0.2s
[CV 2/4] END colsample_bytree=0.8, learning_rate=0.01, max_depth=5,
n_estimators=200, subsample=1.0;, score=0.905 total time=   0.2s
[CV 3/4] END colsample_bytree=0.8, learning_rate=0.01, max_depth=5,
n_estimators=200, subsample=1.0;, score=0.867 total time=   0.2s
[CV 4/4] END colsample_bytree=0.8, learning_rate=0.01, max_depth=5,
```

```
n_estimators=200, subsample=1.0;, score=0.900 total time=   0.2s
Classification Report of model:
              precision    recall  f1-score   support

           0       0.92      0.94      0.93       251
           1       0.95      0.93      0.94       299

    accuracy                           0.93       550
   macro avg       0.93      0.94      0.93       550
weighted avg       0.93      0.93      0.93       550

Accuracy: 0.9345454545454546
```





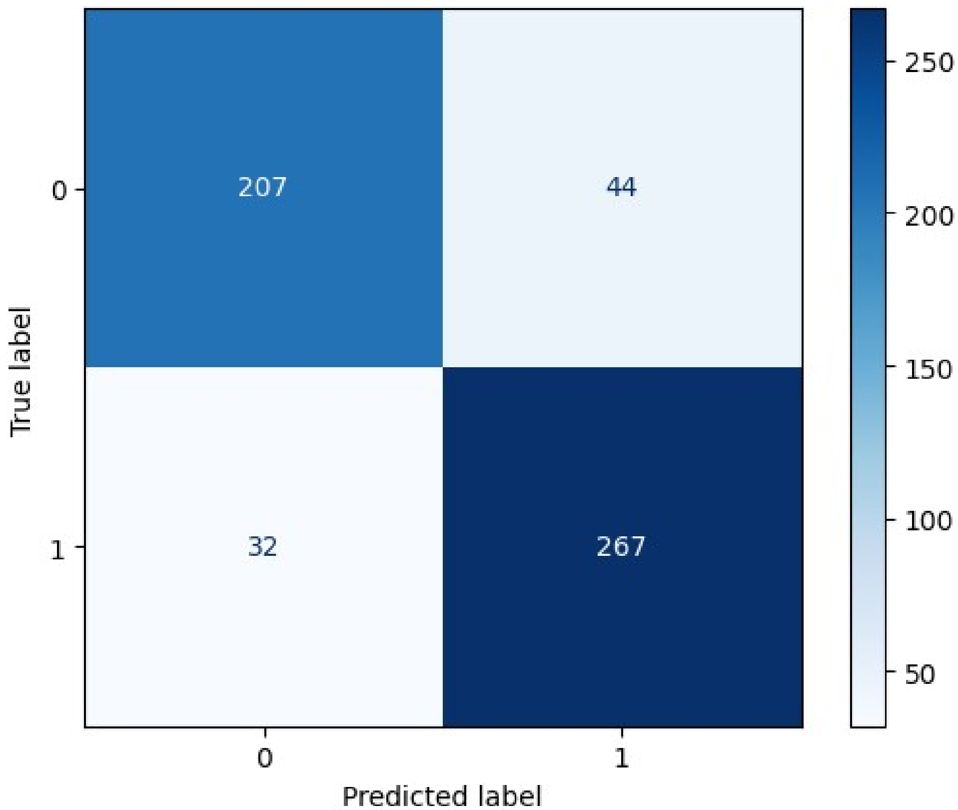
After tuning, there is increase in accuracy to 0.94

SVM

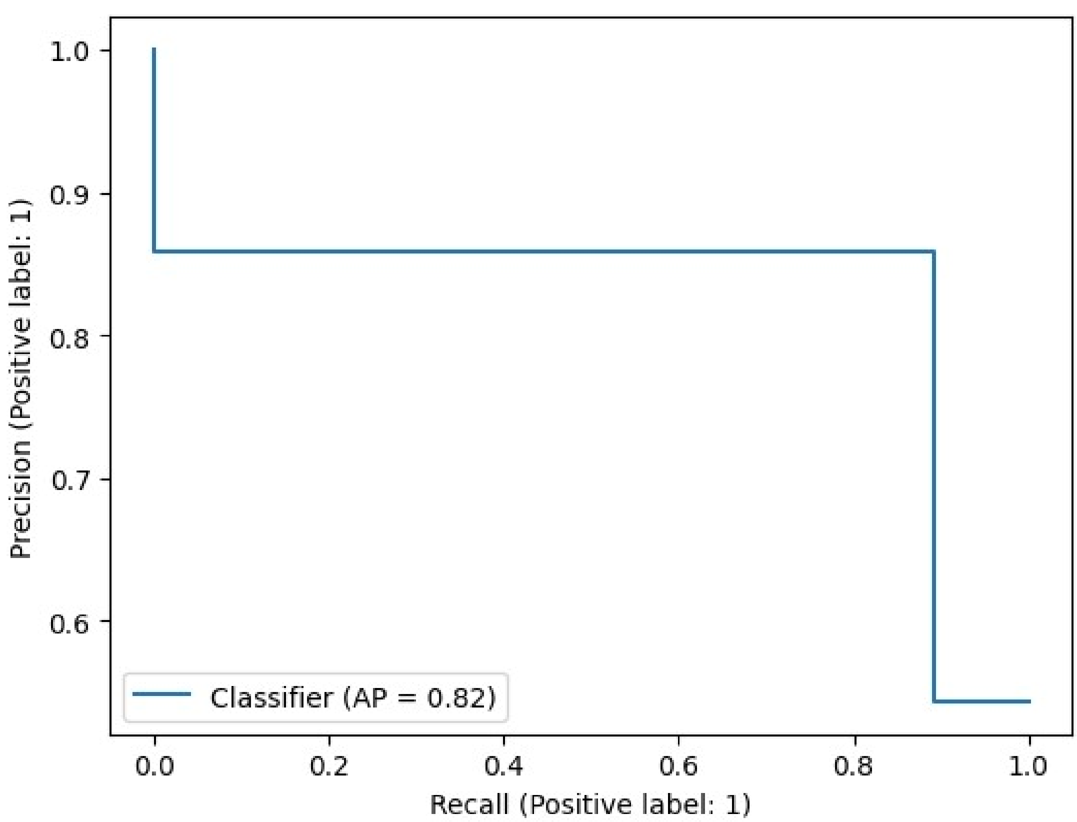
```
# Tuning for svc

param_grid = {
    'C': [0.1, 1, 10],
    'kernel': ['linear', 'rbf', 'poly'],
    'gamma': ['scale', 'auto', 0.1, 1]
}

grid_svc = RandomizedSearchCV(SVC(), param_grid, verbose=3, cv=4)

train_and_evaluate_model(grid_svc)

Fitting 4 folds for each of 10 candidates, totalling 40 fits
[CV 1/4] END ......C=0.1, gamma=0.1, kernel=rbf;, score=0.770 total
time=   0.3s
[CV 2/4] END ......C=0.1, gamma=0.1, kernel=rbf;, score=0.783 total
time=   0.3s
[CV 3/4] END ......C=0.1, gamma=0.1, kernel=rbf;, score=0.754 total
time=   0.3s
[CV 4/4] END ......C=0.1, gamma=0.1, kernel=rbf;, score=0.767 total
time=   0.3s
```

```
[CV 1/4] END .......C=1, gamma=auto, kernel=rbf;, score=0.778 total
time=   0.2s
[CV 2/4] END .......C=1, gamma=auto, kernel=rbf;, score=0.812 total
time=   0.3s
[CV 3/4] END .......C=1, gamma=auto, kernel=rbf;, score=0.778 total
time=   0.2s
[CV 4/4] END .......C=1, gamma=auto, kernel=rbf;, score=0.783 total
time=   0.3s
[CV 1/4] END ........C=10, gamma=1, kernel=poly;, score=0.867 total
time=   1.0s
[CV 2/4] END ........C=10, gamma=1, kernel=poly;, score=0.852 total
time=   0.7s
[CV 3/4] END ........C=10, gamma=1, kernel=poly;, score=0.860 total
time=   0.4s
[CV 4/4] END ........C=10, gamma=1, kernel=poly;, score=0.862 total
time=   0.8s
[CV 1/4] END .......C=10, gamma=0.1, kernel=rbf;, score=0.829 total
time=   0.1s
[CV 2/4] END .......C=10, gamma=0.1, kernel=rbf;, score=0.849 total
time=   0.1s
[CV 3/4] END .......C=10, gamma=0.1, kernel=rbf;, score=0.820 total
time=   0.1s
[CV 4/4] END .......C=10, gamma=0.1, kernel=rbf;, score=0.843 total
time=   0.2s
[CV 1/4] END .....C=0.1, gamma=auto, kernel=rbf;, score=0.721 total
time=   0.3s
[CV 2/4] END .....C=0.1, gamma=auto, kernel=rbf;, score=0.785 total
time=   0.3s
[CV 3/4] END .....C=0.1, gamma=auto, kernel=rbf;, score=0.756 total
time=   0.2s
[CV 4/4] END .....C=0.1, gamma=auto, kernel=rbf;, score=0.730 total
time=   0.2s
[CV 1/4] END ......C=10, gamma=1, kernel=linear;, score=0.796 total
time=   0.2s
[CV 2/4] END ......C=10, gamma=1, kernel=linear;, score=0.774 total
time=   0.2s
[CV 3/4] END ......C=10, gamma=1, kernel=linear;, score=0.758 total
time=   0.2s
[CV 4/4] END ......C=10, gamma=1, kernel=linear;, score=0.787 total
time=   0.2s
[CV 1/4] END ......C=1, gamma=auto, kernel=poly;, score=0.643 total
time=   0.1s
[CV 2/4] END ......C=1, gamma=auto, kernel=poly;, score=0.641 total
time=   0.1s
[CV 3/4] END ......C=1, gamma=auto, kernel=poly;, score=0.638 total
time=   0.1s
[CV 4/4] END ......C=1, gamma=auto, kernel=poly;, score=0.658 total
time=   0.1s
[CV 1/4] END ..C=10, gamma=scale, kernel=linear;, score=0.796 total
```

```
time=   0.2s
[CV 2/4] END ..C=10, gamma=scale, kernel=linear;, score=0.774 total
time=   0.2s
[CV 3/4] END ..C=10, gamma=scale, kernel=linear;, score=0.758 total
time=   0.2s
[CV 4/4] END ..C=10, gamma=scale, kernel=linear;, score=0.787 total
time=   0.2s
[CV 1/4] END ...C=0.1, gamma=scale, kernel=poly;, score=0.776 total
time=   0.1s
[CV 2/4] END ...C=0.1, gamma=scale, kernel=poly;, score=0.807 total
time=   0.1s
[CV 3/4] END ...C=0.1, gamma=scale, kernel=poly;, score=0.770 total
time=   0.2s
[CV 4/4] END ...C=0.1, gamma=scale, kernel=poly;, score=0.787 total
time=   0.2s
[CV 1/4] END .....C=1, gamma=scale, kernel=poly;, score=0.816 total
time=   0.2s
[CV 2/4] END .....C=1, gamma=scale, kernel=poly;, score=0.845 total
time=   0.2s
[CV 3/4] END .....C=1, gamma=scale, kernel=poly;, score=0.798 total
time=   0.2s
[CV 4/4] END .....C=1, gamma=scale, kernel=poly;, score=0.832 total
time=   0.2s
Classification Report of model:
              precision    recall  f1-score   support

           0       0.87      0.82      0.84       251
           1       0.86      0.89      0.88       299

    accuracy                           0.86       550
   macro avg       0.86      0.86      0.86       550
weighted avg       0.86      0.86      0.86       550

Accuracy: 0.8618181818181818
```



207
44
32
267
0
1
True label
Predicted label
250
200
150
100
50



Precision (Positive label: 1)
Recall (Positive label: 1)
Classifier (AP = 0.82)
1.0
0.9
0.8
0.7
0.6
0.0
0.2
0.4
0.6
0.8
1.0

After tuning, there is increase in accuracy to 0.86

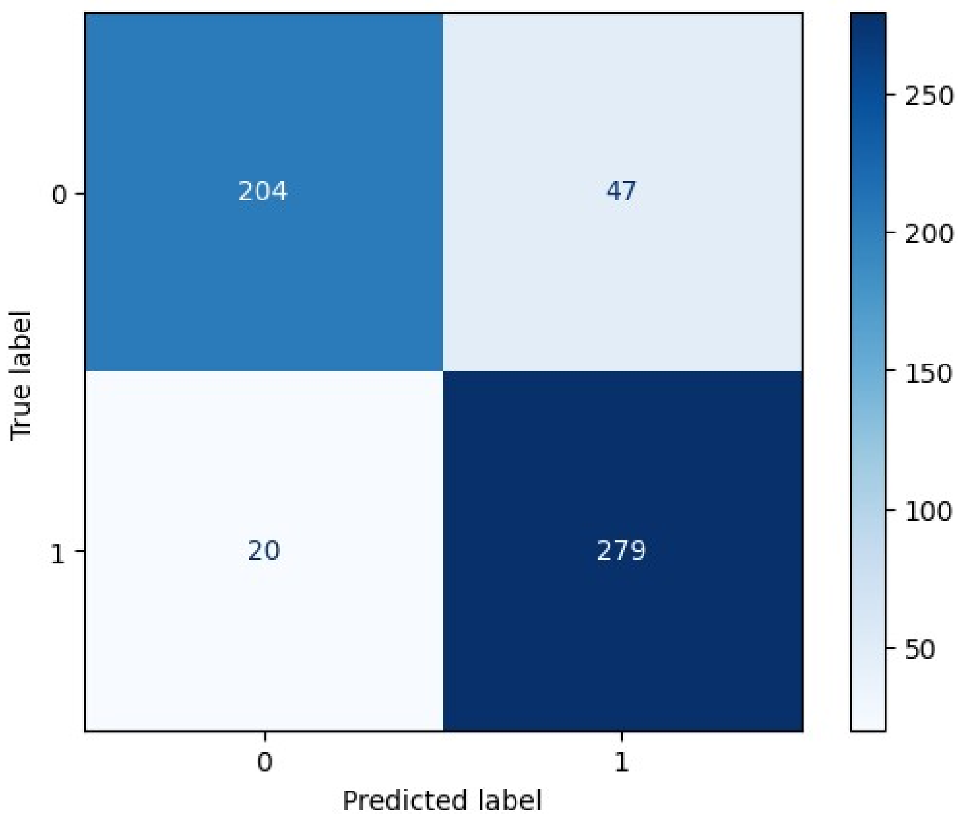
k-Nearest Neighbors (KNN)

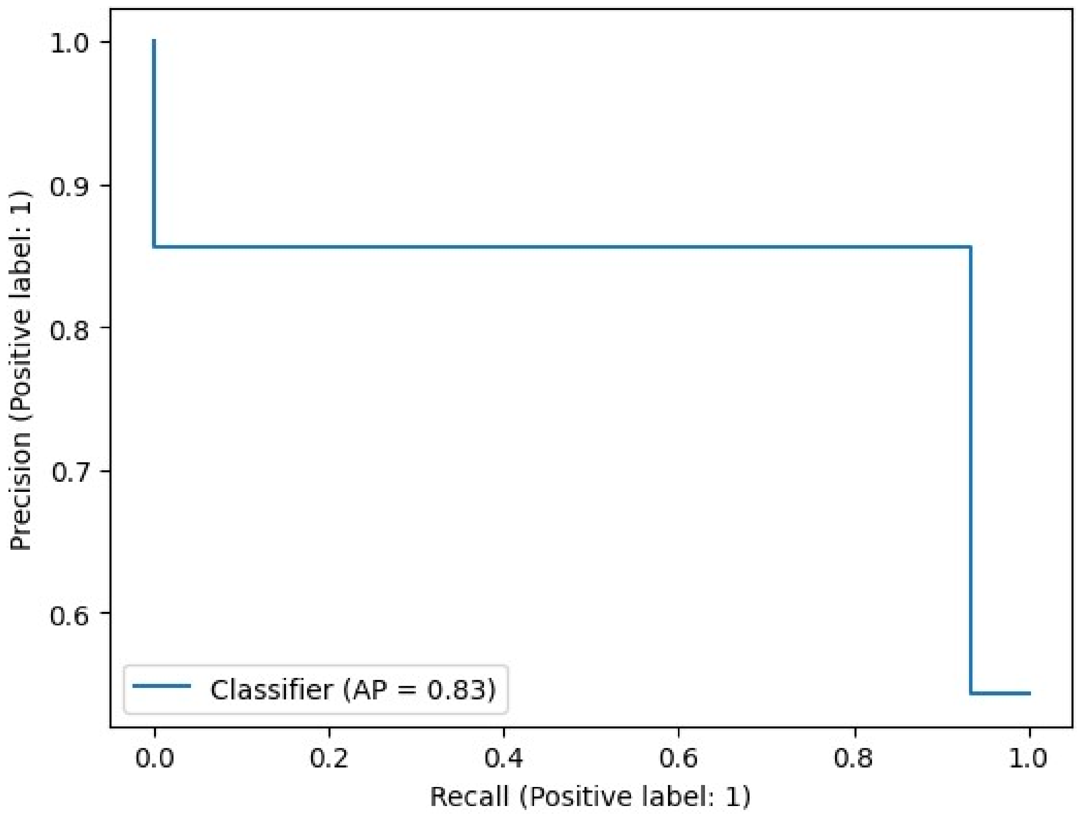
```
# Tuning for k-Nearest Neighbors (KNN)

param_grid = {
    'n_neighbors': [3, 5, 7, 9],
    'weights': ['uniform', 'distance'],
    'p': [1, 2]  # p=1 for Manhattan distance, p=2 for Euclidean
distance
}


grid_knn = RandomizedSearchCV(KNeighborsClassifier(), param_grid,
verbose=3, cv=4)

train_and_evaluate_model(grid_knn)
```

```
Fitting 4 folds for each of 10 candidates, totalling 40 fits
[CV 1/4] END n_neighbors=9, p=1, weights=distance;, score=0.862 total
time=   0.2s
[CV 2/4] END n_neighbors=9, p=1, weights=distance;, score=0.858 total
time=   0.0s
[CV 3/4] END n_neighbors=9, p=1, weights=distance;, score=0.862 total
time=   0.0s
[CV 4/4] END n_neighbors=9, p=1, weights=distance;, score=0.872 total
time=   0.0s
[CV 1/4] END n_neighbors=9, p=2, weights=distance;, score=0.854 total
time=   0.0s
[CV 2/4] END n_neighbors=9, p=2, weights=distance;, score=0.852 total
time=   0.0s
[CV 3/4] END n_neighbors=9, p=2, weights=distance;, score=0.847 total
time=   0.0s
[CV 4/4] END n_neighbors=9, p=2, weights=distance;, score=0.856 total
time=   0.0s
[CV 1/4] END n_neighbors=5, p=2, weights=distance;, score=0.874 total
time=   0.0s
[CV 2/4] END n_neighbors=5, p=2, weights=distance;, score=0.856 total
time=   0.0s
[CV 3/4] END n_neighbors=5, p=2, weights=distance;, score=0.865 total
time=   0.0s
[CV 4/4] END n_neighbors=5, p=2, weights=distance;, score=0.869 total
time=   0.0s
[CV 1/4] END n_neighbors=7, p=1, weights=distance;, score=0.863 total
time=   0.0s
[CV 2/4] END n_neighbors=7, p=1, weights=distance;, score=0.862 total
time=   0.1s
[CV 3/4] END n_neighbors=7, p=1, weights=distance;, score=0.872 total
time=   0.0s
[CV 4/4] END n_neighbors=7, p=1, weights=distance;, score=0.882 total
```

```
time=   0.0s
[CV 1/4] END n_neighbors=3, p=1, weights=uniform;, score=0.878 total
time=   0.1s
[CV 2/4] END n_neighbors=3, p=1, weights=uniform;, score=0.860 total
time=   0.1s
[CV 3/4] END n_neighbors=3, p=1, weights=uniform;, score=0.854 total
time=   0.1s
[CV 4/4] END n_neighbors=3, p=1, weights=uniform;, score=0.880 total
time=   0.1s
[CV 1/4] END n_neighbors=5, p=1, weights=uniform;, score=0.863 total
time=   0.1s
[CV 2/4] END n_neighbors=5, p=1, weights=uniform;, score=0.842 total
time=   0.1s
[CV 3/4] END n_neighbors=5, p=1, weights=uniform;, score=0.840 total
time=   0.1s
[CV 4/4] END n_neighbors=5, p=1, weights=uniform;, score=0.851 total
time=   0.1s
[CV 1/4] END n_neighbors=7, p=2, weights=uniform;, score=0.814 total
time=   0.0s
[CV 2/4] END n_neighbors=7, p=2, weights=uniform;, score=0.816 total
time=   0.0s
[CV 3/4] END n_neighbors=7, p=2, weights=uniform;, score=0.809 total
time=   0.0s
[CV 4/4] END n_neighbors=7, p=2, weights=uniform;, score=0.827 total
time=   0.0s
[CV 1/4] END n_neighbors=3, p=2, weights=uniform;, score=0.854 total
time=   0.0s
[CV 2/4] END n_neighbors=3, p=2, weights=uniform;, score=0.858 total
time=   0.0s
[CV 3/4] END n_neighbors=3, p=2, weights=uniform;, score=0.834 total
time=   0.0s
[CV 4/4] END n_neighbors=3, p=2, weights=uniform;, score=0.858 total
time=   0.0s
[CV 1/4] END n_neighbors=7, p=1, weights=uniform;, score=0.823 total
time=   0.1s
[CV 2/4] END n_neighbors=7, p=1, weights=uniform;, score=0.829 total
time=   0.1s
[CV 3/4] END n_neighbors=7, p=1, weights=uniform;, score=0.825 total
time=   0.1s
[CV 4/4] END n_neighbors=7, p=1, weights=uniform;, score=0.836 total
time=   0.1s
[CV 1/4] END n_neighbors=9, p=2, weights=uniform;, score=0.805 total
time=   0.1s
[CV 2/4] END n_neighbors=9, p=2, weights=uniform;, score=0.811 total
time=   0.0s
[CV 3/4] END n_neighbors=9, p=2, weights=uniform;, score=0.794 total
time=   0.0s
[CV 4/4] END n_neighbors=9, p=2, weights=uniform;, score=0.807 total
time=   0.0s
```

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

           0       0.91      0.81      0.86       251
           1       0.86      0.93      0.89       299

    accuracy                           0.88       550
   macro avg       0.88      0.87      0.88       550
weighted avg       0.88      0.88      0.88       550

Accuracy: 0.8781818181818182
```





After tuning, there is increase in accuracy to 0.87

Also after hypertuning XGBOOST is Performing as Best Model.

#Conclusion

- In this project, we embarked on the journey of building a predictive model for credit card approval. After exploring multiple algorithms, we finalized XGBoost due to its robustness and efficiency in handling tabular data with a mix of different variable types.
- We utilized XGBoost, a gradient boosting algorithm, known for its high performance in classification problems
- Performance Metrics: The model after hyperparameter tuning achieved an accuracy of approximately 94%. Notably, the precision and recall were also high, indicating 94% and 94% respectively. The F1-Score, which is the harmonic mean of precision and recall, stood at 94%, further cementing the model's reliability.
- Through diligent preprocessing, model selection, and hyperparameter tuning, we've crafted a robust model for credit card approval predictions. This model not only

boasts high accuracy but also ensures a balanced trade-off between precision and recall, thus making it a valuable asset for financial institutions aiming to streamline their credit card approval processes.

SQL QUERY

```
import duckdb
conn = duckdb.connect()

conn.register('clean_df',clean_df)

<duckdb.duckdb.DuckDBPyConnection at 0x7c3f5cbf3c70>
```

Q1. Group the customers based on their income type and find the avg. of their annual income

```
fixed_query = """
select Income_Type, Avg(Annual_income) as Avg_Income
from clean_df
group by Income_Type;
"""

result = conn.execute(fixed_query).fetchdf()
print(result)

            Income_Type     Avg_Income
0             Pensioner  155713.746487
1               Working  181191.434321
2         State servant  211422.413793
3  Commercial associate  233653.135917
```

Q2. Find the female owners of cars and property

```
fixed_query = """
select count(*) as 'Female who own Car and property'
from clean_df
where Gender='F' and Car_Owner = 'Y' and Property_Owner ='Y'
"""

result = conn.execute(fixed_query).fetchdf()
print(result)

   Female who own Car and property
0                              179
```

Q3. Find the male Customers who are staying with their Families

```
fixed_query = """
select count(*) as 'Male customers staying with their families'
from clean_df
where Gender = 'M' and Family_Members>1
"""

result = conn.execute(fixed_query).fetchdf()
print(result)
```

```
   Male customers staying with their families
0                                         470
```

Q4. Please list the top five people having the highest income?

```
fixed_query = """
select * from clean_df
order by Annual_income desc
limit 5
"""

result = conn.execute(fixed_query).fetchdf()
print(result)
```

```
    Ind_ID Gender Car_Owner Property_Owner  Children  Annual_income  \
0  5143231      F         Y              Y         1      1575000.0
1  5143235      F         Y              Y         1      1575000.0
2  5090470      M         N              Y         1       900000.0
3  5079016      M         Y              Y         2       900000.0
4  5079017      M         Y              Y         2       900000.0

            Income_Type                      Education
Marital_status  \
0  Commercial associate               Higher education  Single / not
married
1  Commercial associate               Higher education  Single / not
married
2               Working  Secondary / secondary special
Married
3  Commercial associate               Higher education
Married
4  Commercial associate               Higher education
Married

        Housing_type  Work_Phone  Phone  Email_ID  Family_Members
label  \
0  House / apartment           0      0         0               2
0
1  House / apartment           0      0         0               2
0
2  House / apartment           0      0         0               3
```

```
0
3  House / apartment           0      0         0              4
0
4  House / apartment           0      0         0              4
0

   Age_in_Years  Experience_years
0          28.0               7.0
1          28.0               7.0
2          42.0              12.0
3          27.0               3.0
4          27.0               3.0
```

Q5. How many married people are having bad credit

```
fixed_query = """
select count(*) as 'married people having bad credit'
from clean_df
where Marital_status ='Married' and label = 1
"""

result = conn.execute(fixed_query).fetchdf()
print(result)

   married people having bad credit
0                               114
```

Q6. What is the Highest Education level and what is the total count

```
fixed_query = """
select count(*) as 'Highest_degree_count'
from clean_df
where Education='Academic degree'
"""

result = conn.execute(fixed_query).fetchdf()
print(result)

   Highest_degree_count
0                     2
```

Q7. Between married males and females , eho is having more bad credit?

```
fixed_query = """
select Gender, count(Gender) as 'count'
from clean_df
where Marital_status='Married' and label=1
group by Gender
order by count desc
```

```
limit 1
"""

result = conn.execute(fixed_query).fetchdf()
print(result)
```

```
  Gender  count
0      F     63
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
%%capture
!wget -nc https://raw.githubusercontent.com/brpy/colab-
pdf/master/colab_pdf.py
from colab_pdf import colab_pdf
colab_pdf('capstone project(credit card).ipynb')
```