**Part 1 Data Exploration**

In [1]:

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

import warnings

warnings.filterwarnings("ignore")

In [3]:

df = pd.read\_csv("/content/drive/MyDrive/BankChurnAnalysis/BankChurners.csv")

In [4]:

df.head()

Out[4]:

|  | **CLIENTNUM** | **Attrition\_Flag** | **Customer\_Age** | **Gender** | **Dependent\_count** | **Education\_Level** | **Marital\_Status** | **Income\_Category** | **Card\_Category** | **Months\_on\_book** | **...** | **Credit\_Limit** | **Total\_Revolving\_Bal** | **Avg\_Open\_To\_Buy** | **Total\_Amt\_Chng\_Q4\_Q1** | **Total\_Trans\_Amt** | **Total\_Trans\_Ct** | **Total\_Ct\_Chng\_Q4\_Q1** | **Avg\_Utilization\_Ratio** | **Naive\_Bayes\_Classifier\_Attrition\_Flag\_Card\_Category\_Contacts\_Count\_12\_mon\_Dependent\_count\_Education\_Level\_Months\_Inactive\_12\_mon\_1** | **Naive\_Bayes\_Classifier\_Attrition\_Flag\_Card\_Category\_Contacts\_Count\_12\_mon\_Dependent\_count\_Education\_Level\_Months\_Inactive\_12\_mon\_2** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 768805383 | Existing Customer | 45 | M | 3 | High School | Married | 60𝐾−80K | Blue | 39 | ... | 12691.0 | 777 | 11914.0 | 1.335 | 1144 | 42 | 1.625 | 0.061 | 0.000093 | 0.99991 |
| **1** | 818770008 | Existing Customer | 49 | F | 5 | Graduate | Single | Less than $40K | Blue | 44 | ... | 8256.0 | 864 | 7392.0 | 1.541 | 1291 | 33 | 3.714 | 0.105 | 0.000057 | 0.99994 |
| **2** | 713982108 | Existing Customer | 51 | M | 3 | Graduate | Married | 80𝐾−120K | Blue | 36 | ... | 3418.0 | 0 | 3418.0 | 2.594 | 1887 | 20 | 2.333 | 0.000 | 0.000021 | 0.99998 |
| **3** | 769911858 | Existing Customer | 40 | F | 4 | High School | Unknown | Less than $40K | Blue | 34 | ... | 3313.0 | 2517 | 796.0 | 1.405 | 1171 | 20 | 2.333 | 0.760 | 0.000134 | 0.99987 |
| **4** | 709106358 | Existing Customer | 40 | M | 3 | Uneducated | Married | 60𝐾−80K | Blue | 21 | ... | 4716.0 | 0 | 4716.0 | 2.175 | 816 | 28 | 2.500 | 0.000 | 0.000022 | 0.99998 |

5 rows × 23 columns

**1.1 What’s the shape of this dataset? How many features do we have in this dataset? Is there any null values?**

In [5]:

df.shape

Out[5]:

(10127, 23)

In [6]:

# Create a Boolean mask of null values

null\_mask = df.isnull()

# Sum the mask to find the total number of null values

num\_nulls = null\_mask.sum()

print(num\_nulls)

CLIENTNUM 0

Attrition\_Flag 0

Customer\_Age 0

Gender 0

Dependent\_count 0

Education\_Level 0

Marital\_Status 0

Income\_Category 0

Card\_Category 0

Months\_on\_book 0

Total\_Relationship\_Count 0

Months\_Inactive\_12\_mon 0

Contacts\_Count\_12\_mon 0

Credit\_Limit 0

Total\_Revolving\_Bal 0

Avg\_Open\_To\_Buy 0

Total\_Amt\_Chng\_Q4\_Q1 0

Total\_Trans\_Amt 0

Total\_Trans\_Ct 0

Total\_Ct\_Chng\_Q4\_Q1 0

Avg\_Utilization\_Ratio 0

Naive\_Bayes\_Classifier\_Attrition\_Flag\_Card\_Category\_Contacts\_Count\_12\_mon\_Dependent\_count\_Education\_Level\_Months\_Inactive\_12\_mon\_1 0

Naive\_Bayes\_Classifier\_Attrition\_Flag\_Card\_Category\_Contacts\_Count\_12\_mon\_Dependent\_count\_Education\_Level\_Months\_Inactive\_12\_mon\_2 0

dtype: int64

In this dataset,it involves 23 features. we have 23 columns with 10127 rows and based on the talbe above, we don't have any null values.

**1.2 Is there any data that we need to remove in this dataset?**

In [7]:

df = df.drop(df.columns[[-1, -2]],axis = 1)

df.head()

Out[7]:

|  | **CLIENTNUM** | **Attrition\_Flag** | **Customer\_Age** | **Gender** | **Dependent\_count** | **Education\_Level** | **Marital\_Status** | **Income\_Category** | **Card\_Category** | **Months\_on\_book** | **...** | **Months\_Inactive\_12\_mon** | **Contacts\_Count\_12\_mon** | **Credit\_Limit** | **Total\_Revolving\_Bal** | **Avg\_Open\_To\_Buy** | **Total\_Amt\_Chng\_Q4\_Q1** | **Total\_Trans\_Amt** | **Total\_Trans\_Ct** | **Total\_Ct\_Chng\_Q4\_Q1** | **Avg\_Utilization\_Ratio** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 768805383 | Existing Customer | 45 | M | 3 | High School | Married | 60𝐾−80K | Blue | 39 | ... | 1 | 3 | 12691.0 | 777 | 11914.0 | 1.335 | 1144 | 42 | 1.625 | 0.061 |
| **1** | 818770008 | Existing Customer | 49 | F | 5 | Graduate | Single | Less than $40K | Blue | 44 | ... | 1 | 2 | 8256.0 | 864 | 7392.0 | 1.541 | 1291 | 33 | 3.714 | 0.105 |
| **2** | 713982108 | Existing Customer | 51 | M | 3 | Graduate | Married | 80𝐾−120K | Blue | 36 | ... | 1 | 0 | 3418.0 | 0 | 3418.0 | 2.594 | 1887 | 20 | 2.333 | 0.000 |
| **3** | 769911858 | Existing Customer | 40 | F | 4 | High School | Unknown | Less than $40K | Blue | 34 | ... | 4 | 1 | 3313.0 | 2517 | 796.0 | 1.405 | 1171 | 20 | 2.333 | 0.760 |
| **4** | 709106358 | Existing Customer | 40 | M | 3 | Uneducated | Married | 60𝐾−80K | Blue | 21 | ... | 1 | 0 | 4716.0 | 0 | 4716.0 | 2.175 | 816 | 28 | 2.500 | 0.000 |

5 rows × 21 columns

In this dataset, the last two columns are not needed, so we remove them to keep data organized and clean

**1.4 If this dataset is imbalanced?**

In [8]:

target = df["Attrition\_Flag"] # Target variable

In [9]:

target.value\_counts()

Out[9]:

Existing Customer 8500

Attrited Customer 1627

Name: Attrition\_Flag, dtype: int64

In [10]:

target.value\_counts().plot.pie(autopct='%.2f',figsize=(6, 5))

Out[10]:

* As we can see, this dataset is not balanced at all where lable 0 make up the majority of the dataset and leaving lable 1 fewer examples.
* It will casue problems becasue a biased model will be built based on imbalanced data which is accurate to predict the majority class but fails to predict the minority class.
* This problem will be resolved using random over-sampling technique.

**Part 2 Data Exploration for categorical data**

**Univariate analysis for numerical data**

**2.1 How many unique values are there, and what are the frequencies of these values? Shall we convert any of the categorical variables to numerical to get better results?**

In [11]:

df["Attrition\_Flag"].replace(['Existing Customer', 'Attrited Customer'],[0, 1], inplace=True)

df["Attrition\_Flag"].value\_counts()

Out[11]:

0 8500

1 1627

Name: Attrition\_Flag, dtype: int64

In [12]:

df["Gender"].replace(['F', 'M'],[0, 1], inplace=True)

df["Gender"].value\_counts()

Out[12]:

0 5358

1 4769

Name: Gender, dtype: int64

In [13]:

df['Education\_Level'].replace(['Uneducated', 'High School','College','Graduate','Post-Graduate','Doctorate','Unknown'],[0, 1, 2, 3, 4,5,np.nan], inplace=True)

df['Education\_Level'].value\_counts(dropna=False)

Out[13]:

3.0 3128

1.0 2013

NaN 1519

0.0 1487

2.0 1013

4.0 516

5.0 451

Name: Education\_Level, dtype: int64

In [14]:

df['Education\_Level'].replace([np.nan],[3], inplace=True)

df['Education\_Level'].value\_counts(dropna=False)

Out[14]:

3.0 4647

1.0 2013

0.0 1487

2.0 1013

4.0 516

5.0 451

Name: Education\_Level, dtype: int64

In [15]:

df["Income\_Category"].replace(['Less than $40K', '$40K - $60K','$60K - $80K','$80K - $120K','$120K +','Unknown'],[0, 1, 2, 3, 4,np.nan], inplace=True)

df['Income\_Category'].value\_counts(dropna=False)

Out[15]:

0.0 3561

1.0 1790

3.0 1535

2.0 1402

NaN 1112

4.0 727

Name: Income\_Category, dtype: int64

In [16]:

df["Income\_Category"].replace([np.nan],[0], inplace=True)

df['Income\_Category'].value\_counts()

Out[16]:

0.0 4673

1.0 1790

3.0 1535

2.0 1402

4.0 727

Name: Income\_Category, dtype: int64

In [17]:

df["Card\_Category"].replace(['Blue', 'Silver', 'Gold', 'Platinum'],[0, 1, 2, 3], inplace=True)

df["Card\_Category"].value\_counts()

Out[17]:

0 9436

1 555

2 116

3 20

Name: Card\_Category, dtype: int64

In this section, we convert all the categorical to numerical values.

In [18]:

df\_cat = df[['Attrition\_Flag','Gender','Education\_Level','Marital\_Status','Income\_Category','Card\_Category']]

In [19]:

plt.figure(figsize=(28, 20))

plt.subplot(3,3,1)

sns.countplot(x=df["Attrition\_Flag"],data=df)

plt.subplot(3,3,2)

sns.countplot(x=df["Gender"],hue="Attrition\_Flag",order = df['Gender'].value\_counts().index,data=df)

plt.subplot(3,3,3)

ax2 = sns.countplot(x=df["Education\_Level"],hue="Attrition\_Flag",order = df['Education\_Level'].value\_counts().index, data=df)

plt.subplot(3,3,4)

ax3 = sns.countplot(x=df["Income\_Category"],hue="Attrition\_Flag",order = df['Income\_Category'].value\_counts().index,data=df)

plt.subplot(3,3,5)

ax4 = sns.countplot(x=df["Marital\_Status"],hue="Attrition\_Flag",order = df['Marital\_Status'].value\_counts().index,data=df)

plt.subplot(3,3,6)

ax5 = sns.countplot(x=df["Card\_Category"],hue="Attrition\_Flag",order = df['Card\_Category'].value\_counts().index,data=df)

**Part 3 Data Exploration for numerical data**

**Plot histogram of numerical data**

In [20]:

df\_num = df[['Customer\_Age','Dependent\_count','Months\_on\_book','Total\_Relationship\_Count','Months\_Inactive\_12\_mon','Contacts\_Count\_12\_mon','Credit\_Limit','Total\_Revolving\_Bal','Avg\_Open\_To\_Buy','Total\_Amt\_Chng\_Q4\_Q1','Total\_Trans\_Amt','Total\_Trans\_Ct','Total\_Ct\_Chng\_Q4\_Q1','Avg\_Utilization\_Ratio']]

In [21]:

plt.figure(figsize=(22, 20))

plt.subplot(4,4,1)

sns.histplot(x=df["Customer\_Age"],kde=True,hue=df["Attrition\_Flag"],data=df)

plt.subplot(4,4,2)

sns.histplot(x=df["Dependent\_count"],hue=df["Attrition\_Flag"],kde=False,data=df)

plt.subplot(4,4,3)

sns.histplot(x=df["Months\_on\_book"],hue=df["Attrition\_Flag"],kde=True,data=df)

plt.subplot(4,4,4)

sns.histplot(x=df["Total\_Relationship\_Count"],hue=df["Attrition\_Flag"],kde=False,data=df)

plt.subplot(4,4,5)

sns.histplot(x=df["Months\_Inactive\_12\_mon"],hue=df["Attrition\_Flag"],kde=False,data=df)

plt.subplot(4,4,6)

sns.histplot(x=df["Contacts\_Count\_12\_mon"],hue=df["Attrition\_Flag"],kde=False,data=df)

plt.subplot(4,4,7)

sns.histplot(x=df["Credit\_Limit"],hue=df["Attrition\_Flag"],kde=True,data=df)

plt.subplot(4,4,8)

sns.histplot(x=df["Total\_Revolving\_Bal"],hue=df["Attrition\_Flag"],kde=True,data=df)

plt.subplot(4,4,9)

sns.histplot(x=df["Avg\_Open\_To\_Buy"],hue=df["Attrition\_Flag"],kde=True,data=df)

plt.subplot(4,4,10)

sns.histplot(x=df["Total\_Amt\_Chng\_Q4\_Q1"],hue=df["Attrition\_Flag"],kde=True,data=df)

plt.subplot(4,4,11)

sns.histplot(x=df["Total\_Trans\_Ct"],hue=df["Attrition\_Flag"],kde=True,data=df)

plt.subplot(4,4,12)

sns.histplot(x=df["Total\_Ct\_Chng\_Q4\_Q1"],hue=df["Attrition\_Flag"],kde=True,data=df)

plt.subplot(4,4,13)

sns.histplot(x=df["Avg\_Utilization\_Ratio"],hue=df["Attrition\_Flag"],kde=True,data=df)

Out[21]:

**3.1 Is there any outliers in any of these features in this dataset?**

From the histograms above, we do not notice any outliers.

**3.2 Are there any correlations or associations between the data and label?**

In [22]:

df\_num["Flag"]=df['Attrition\_Flag']

In [23]:

df\_num.head()

Out[23]:

|  | **Customer\_Age** | **Dependent\_count** | **Months\_on\_book** | **Total\_Relationship\_Count** | **Months\_Inactive\_12\_mon** | **Contacts\_Count\_12\_mon** | **Credit\_Limit** | **Total\_Revolving\_Bal** | **Avg\_Open\_To\_Buy** | **Total\_Amt\_Chng\_Q4\_Q1** | **Total\_Trans\_Amt** | **Total\_Trans\_Ct** | **Total\_Ct\_Chng\_Q4\_Q1** | **Avg\_Utilization\_Ratio** | **Flag** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 45 | 3 | 39 | 5 | 1 | 3 | 12691.0 | 777 | 11914.0 | 1.335 | 1144 | 42 | 1.625 | 0.061 | 0 |
| **1** | 49 | 5 | 44 | 6 | 1 | 2 | 8256.0 | 864 | 7392.0 | 1.541 | 1291 | 33 | 3.714 | 0.105 | 0 |
| **2** | 51 | 3 | 36 | 4 | 1 | 0 | 3418.0 | 0 | 3418.0 | 2.594 | 1887 | 20 | 2.333 | 0.000 | 0 |
| **3** | 40 | 4 | 34 | 3 | 4 | 1 | 3313.0 | 2517 | 796.0 | 1.405 | 1171 | 20 | 2.333 | 0.760 | 0 |
| **4** | 40 | 3 | 21 | 5 | 1 | 0 | 4716.0 | 0 | 4716.0 | 2.175 | 816 | 28 | 2.500 | 0.000 | 0 |

In [24]:

sns.set(rc={'figure.figsize':(15,10)})

sns.heatmap(df\_num.corr(),annot=True,cmap='RdBu\_r')

Out[24]:

**3.3 What are the top 5 numerical features that most correlated on churn?**

In [25]:

correlations = df.corr()

# Take the absolute value of the correlations

correlations = correlations.abs()

# Sort the correlations in descending order

correlations = correlations.sort\_values(by="Attrition\_Flag", ascending=False)

# Select the top 5 variables that are most correlated with the target

top\_5 = correlations.head(6)

index\_labels = top\_5.index.tolist()

# Print the index labels of the top 5 variables

print("Top 5 correlated numericalvariables:")

for label in index\_labels[1:]:

print(label)

Top 5 correlated numericalvariables:

Total\_Trans\_Ct

Total\_Ct\_Chng\_Q4\_Q1

Total\_Revolving\_Bal

Contacts\_Count\_12\_mon

Avg\_Utilization\_Ratio

**Feature Enginearing**

In [26]:

df["Recolving\_Bal\_Per\_Relationship"]=df["Total\_Revolving\_Bal"]/df["Total\_Relationship\_Count"]

In [27]:

df.head()

Out[27]:

|  | **CLIENTNUM** | **Attrition\_Flag** | **Customer\_Age** | **Gender** | **Dependent\_count** | **Education\_Level** | **Marital\_Status** | **Income\_Category** | **Card\_Category** | **Months\_on\_book** | **...** | **Contacts\_Count\_12\_mon** | **Credit\_Limit** | **Total\_Revolving\_Bal** | **Avg\_Open\_To\_Buy** | **Total\_Amt\_Chng\_Q4\_Q1** | **Total\_Trans\_Amt** | **Total\_Trans\_Ct** | **Total\_Ct\_Chng\_Q4\_Q1** | **Avg\_Utilization\_Ratio** | **Recolving\_Bal\_Per\_Relationship** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 768805383 | 0 | 45 | 1 | 3 | 1.0 | Married | 2.0 | 0 | 39 | ... | 3 | 12691.0 | 777 | 11914.0 | 1.335 | 1144 | 42 | 1.625 | 0.061 | 155.4 |
| **1** | 818770008 | 0 | 49 | 0 | 5 | 3.0 | Single | 0.0 | 0 | 44 | ... | 2 | 8256.0 | 864 | 7392.0 | 1.541 | 1291 | 33 | 3.714 | 0.105 | 144.0 |
| **2** | 713982108 | 0 | 51 | 1 | 3 | 3.0 | Married | 3.0 | 0 | 36 | ... | 0 | 3418.0 | 0 | 3418.0 | 2.594 | 1887 | 20 | 2.333 | 0.000 | 0.0 |
| **3** | 769911858 | 0 | 40 | 0 | 4 | 1.0 | Unknown | 0.0 | 0 | 34 | ... | 1 | 3313.0 | 2517 | 796.0 | 1.405 | 1171 | 20 | 2.333 | 0.760 | 839.0 |
| **4** | 709106358 | 0 | 40 | 1 | 3 | 0.0 | Married | 2.0 | 0 | 21 | ... | 0 | 4716.0 | 0 | 4716.0 | 2.175 | 816 | 28 | 2.500 | 0.000 | 0.0 |

5 rows × 22 columns

**One hot encoding**

In [28]:

df1 = pd.get\_dummies(df, columns=['Marital\_Status'],drop\_first=True,prefix='Is')

df1.head()

Out[28]:

|  | **CLIENTNUM** | **Attrition\_Flag** | **Customer\_Age** | **Gender** | **Dependent\_count** | **Education\_Level** | **Income\_Category** | **Card\_Category** | **Months\_on\_book** | **Total\_Relationship\_Count** | **...** | **Avg\_Open\_To\_Buy** | **Total\_Amt\_Chng\_Q4\_Q1** | **Total\_Trans\_Amt** | **Total\_Trans\_Ct** | **Total\_Ct\_Chng\_Q4\_Q1** | **Avg\_Utilization\_Ratio** | **Recolving\_Bal\_Per\_Relationship** | **Is\_Married** | **Is\_Single** | **Is\_Unknown** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 768805383 | 0 | 45 | 1 | 3 | 1.0 | 2.0 | 0 | 39 | 5 | ... | 11914.0 | 1.335 | 1144 | 42 | 1.625 | 0.061 | 155.4 | 1 | 0 | 0 |
| **1** | 818770008 | 0 | 49 | 0 | 5 | 3.0 | 0.0 | 0 | 44 | 6 | ... | 7392.0 | 1.541 | 1291 | 33 | 3.714 | 0.105 | 144.0 | 0 | 1 | 0 |
| **2** | 713982108 | 0 | 51 | 1 | 3 | 3.0 | 3.0 | 0 | 36 | 4 | ... | 3418.0 | 2.594 | 1887 | 20 | 2.333 | 0.000 | 0.0 | 1 | 0 | 0 |
| **3** | 769911858 | 0 | 40 | 0 | 4 | 1.0 | 0.0 | 0 | 34 | 3 | ... | 796.0 | 1.405 | 1171 | 20 | 2.333 | 0.760 | 839.0 | 0 | 0 | 1 |
| **4** | 709106358 | 0 | 40 | 1 | 3 | 0.0 | 2.0 | 0 | 21 | 5 | ... | 4716.0 | 2.175 | 816 | 28 | 2.500 | 0.000 | 0.0 | 1 | 0 | 0 |

5 rows × 24 columns

**What are the correlations after the feature engineering?**

In [29]:

correlations\_fe = df1.corr()

# Take the absolute value of the correlations

correlations\_fe = correlations\_fe.abs()

# Sort the correlations in descending order

correlations\_fe = correlations\_fe.sort\_values(by="Attrition\_Flag", ascending=False)

# Select the top 5 variables that are most correlated with the target

top\_5 = correlations\_fe.head(6)

index\_labels = top\_5.index.tolist()

# Print the index labels of the top 5 variables

print("Top 5 correlated numericalvariables:")

for label in index\_labels[1:]:

print(label)

Top 5 correlated numericalvariables:

Total\_Trans\_Ct

Total\_Ct\_Chng\_Q4\_Q1

Total\_Revolving\_Bal

Contacts\_Count\_12\_mon

Avg\_Utilization\_Ratio

**Model1: Logistic Regression**

**Import packages**

In [30]:

pip install shap

Looking in indexes: <https://pypi.org/simple>, <https://us-python.pkg.dev/colab-wheels/public/simple/>

Collecting shap

Downloading shap-0.41.0-cp38-cp38-manylinux\_2\_12\_x86\_64.manylinux2010\_x86\_64.whl (575 kB)

━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 575.9/575.9 KB 11.8 MB/s eta 0:00:00

Requirement already satisfied: numba in /usr/local/lib/python3.8/dist-packages (from shap) (0.56.4)

Requirement already satisfied: scipy in /usr/local/lib/python3.8/dist-packages (from shap) (1.7.3)

Requirement already satisfied: pandas in /usr/local/lib/python3.8/dist-packages (from shap) (1.3.5)

Requirement already satisfied: packaging>20.9 in /usr/local/lib/python3.8/dist-packages (from shap) (21.3)

Collecting slicer==0.0.7

Downloading slicer-0.0.7-py3-none-any.whl (14 kB)

Requirement already satisfied: tqdm>4.25.0 in /usr/local/lib/python3.8/dist-packages (from shap) (4.64.1)

Requirement already satisfied: cloudpickle in /usr/local/lib/python3.8/dist-packages (from shap) (1.5.0)

Requirement already satisfied: numpy in /usr/local/lib/python3.8/dist-packages (from shap) (1.21.6)

Requirement already satisfied: scikit-learn in /usr/local/lib/python3.8/dist-packages (from shap) (1.0.2)

Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /usr/local/lib/python3.8/dist-packages (from packaging>20.9->shap) (3.0.9)

Requirement already satisfied: setuptools in /usr/local/lib/python3.8/dist-packages (from numba->shap) (57.4.0)

Requirement already satisfied: importlib-metadata in /usr/local/lib/python3.8/dist-packages (from numba->shap) (5.2.0)

Requirement already satisfied: llvmlite<0.40,>=0.39.0dev0 in /usr/local/lib/python3.8/dist-packages (from numba->shap) (0.39.1)

Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.8/dist-packages (from pandas->shap) (2.8.2)

Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.8/dist-packages (from pandas->shap) (2022.7)

Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.8/dist-packages (from scikit-learn->shap) (1.2.0)

Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.8/dist-packages (from scikit-learn->shap) (3.1.0)

Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.8/dist-packages (from python-dateutil>=2.7.3->pandas->shap) (1.15.0)

Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.8/dist-packages (from importlib-metadata->numba->shap) (3.11.0)

Installing collected packages: slicer, shap

Successfully installed shap-0.41.0 slicer-0.0.7

In [31]:

from sklearn.linear\_model import LogisticRegression

from sklearn import metrics

from sklearn.metrics import classification\_report, confusion\_matrix

from sklearn.metrics import precision\_recall\_curve

from sklearn.model\_selection import cross\_validate

from sklearn.metrics import average\_precision\_score

from sklearn.model\_selection import train\_test\_split

from imblearn.over\_sampling import SMOTE

from imblearn.over\_sampling import RandomOverSampler

import shap

In [32]:

X = df1.iloc[:,2:] # Features

y = df1["Attrition\_Flag"] # Target variable

**split data without random sampling**

In [33]:

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=321)

**Random Over Sampling**

In [34]:

ros = RandomOverSampler(sampling\_strategy=1)

X\_res\_os, y\_res\_os = ros.fit\_resample(X\_train, y\_train)

y\_res\_os.value\_counts().plot.pie(autopct='%.2f',figsize=(6, 5))

Out[34]:

In [35]:

X\_train\_os, X\_test\_os, y\_train\_os, y\_test\_os = train\_test\_split(X\_res\_os, y\_res\_os, test\_size=0.2, random\_state=223)

**Logistics Regression**

In [41]:

logreg = LogisticRegression(max\_iter=1000,solver="liblinear")

logreg.fit(X\_train\_os, y\_train\_os)

Out[41]:

LogisticRegression(max\_iter=1000, solver='liblinear')

In [42]:

y\_pred\_logreg\_test = logreg.predict(X\_test)

y\_pred\_proba\_logreg\_test = logreg.predict\_proba(X\_test)[:, 1]

**Classification Report**

In [43]:

print(classification\_report(y\_test, y\_pred\_logreg\_test))

precision recall f1-score support

0 0.96 0.84 0.90 1723

1 0.47 0.80 0.59 303

accuracy 0.83 2026

macro avg 0.71 0.82 0.74 2026

weighted avg 0.89 0.83 0.85 2026

rfc\_ap\_df=pd.DataFrame({"ap\_train":ap\_train\_list\_logreg,"ap\_test":ap\_test\_list\_logreg})

avg\_list = [rfc\_ap\_df['ap\_train'].mean(), rfc\_ap\_df['ap\_test'].mean()]

# Add the two rows to the dataframe

rfc\_ap\_df.loc['Average'] = avg\_list

rfc\_ap\_df.head(7)

Out[48]:

|  | **ap\_train** | **ap\_test** |
| --- | --- | --- |
| **0** | 0.9270 | 0.7050 |
| **1** | 0.9160 | 0.7640 |
| **2** | 0.9190 | 0.7510 |
| **3** | 0.9260 | 0.7370 |
| **4** | 0.9250 | 0.7070 |
| **Average** | 0.9226 | 0.7328 |

In [49]:

ap\_train\_list\_logreg=[]

ap\_test\_list\_logreg=[]

for i in range(1,6):

X = df1.iloc[:,2:] # Features

y = df1["Attrition\_Flag"] # Target variable

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=i\*321)

ros = RandomOverSampler(sampling\_strategy=1)

X\_res\_os, y\_res\_os = ros.fit\_resample(X\_train, y\_train)

X\_train\_os, X\_test\_os, y\_train\_os, y\_test\_os = train\_test\_split(X\_res\_os, y\_res\_os, test\_size=0.2, random\_state=i\*357)

logreg = LogisticRegression(max\_iter=1000,solver="liblinear")

logreg.fit(X\_train\_os, y\_train\_os)

y\_pred=logreg.predict(X\_test)

y\_pred\_proba\_logreg = logreg.predict\_proba(X\_test)[:, 1]

y\_pred\_proba\_logreg\_train = logreg.predict\_proba(X\_train\_os)[:, 1]

ap\_train=np.round(metrics.roc\_auc\_score(y\_train\_os, y\_pred\_proba\_logreg\_train),3)

ap\_train\_list\_logreg.append(ap\_train)

ap\_test=np.round(metrics.roc\_auc\_score(y\_test, y\_pred\_proba\_logreg),3)

ap\_test\_list\_logreg.append(ap\_test)

my\_aucplot(y\_test, y\_pred\_proba\_logreg,"roc\_auc = " +str(ap\_test))

rfc\_ap\_df=pd.DataFrame({"roc\_train":ap\_train\_list\_logreg,"roc\_test":ap\_test\_list\_logreg})

avg\_list = [rfc\_ap\_df['roc\_train'].mean(), rfc\_ap\_df['roc\_test'].mean()]

# Add the two rows to the dataframe

rfc\_ap\_df.loc['Average'] = avg\_list

rfc\_ap\_df.head(7)

Out[49]:

|  | **roc\_train** | **roc\_test** |
| --- | --- | --- |
| **0** | 0.9300 | 0.9030 |
| **1** | 0.9220 | 0.9240 |
| **2** | 0.9260 | 0.9270 |
| **3** | 0.9340 | 0.9110 |
| **4** | 0.9260 | 0.9240 |
| **Average** | 0.9276 | 0.9178 |

**Model 2: Decision Tree**

**DECISION TREE CODE**

import pandas as pd #import numpy as np import seaborn as sns import matplotlib.pyplot as plt import pandas as pd import numpy as np import seaborn as sns import matplotlib.pyplot as plt import sklearn.tree import sklearn.ensemble from sklearn.tree import DecisionTreeClassifier # import library function for decision tree classification algorithm from sklearn.ensemble import RandomForestClassifier from sklearn.metrics import classification\_report, confusion\_matrix from sklearn.model\_selection import train\_test\_split #library function to split the dataset into train and test data from sklearn.metrics import accuracy\_score # accuracy of the model classification

data = pd.read\_csv("BankChurners.csv")

# check data types data.dtypes

data.head(2)

# check missing values data.isnull().sum()

# map 'Existing Customer' to 0 and 'Attrited Customer' to 1 mapping = {'Existing Customer':0, 'Attrited Customer':1} data['Attrition\_Flag'] = data['Attrition\_Flag'].map(mapping) data['Attrition\_Flag'].unique()

#The dataset is split into features and target target\_var = 'Attrition\_Flag' X = data.drop(columns=[target\_var]) y= data[target\_var]

#One-hot encoding categorical features one\_encoding= pd.get\_dummies(X)

# dataset split into training and testing data X\_train, X\_test, y\_train, y\_test = train\_test\_split(one\_encoding[['CLIENTNUM','Months\_on\_book','Months\_Inactive\_12\_mon','Credit\_Limit', 'Card\_Category\_Blue', 'Card\_Category\_Gold','Card\_Category\_Platinum', 'Card\_Category\_Silver']], data['Attrition\_Flag'] ,test\_size=0.25, random\_state=42) #test data is 25% and 75% for training data

#create a decision tree model tree\_model = DecisionTreeClassifier() tree\_model.fit(X\_train, y\_train)

#Predict the bankchurn rate for the testing set y\_pred = clf\_model.predict(X\_test) print("Predicted values:", y\_pred)

#The accuracy of the model accuracy = accuracy\_score(y\_test, y\_pred) print("Accuracy:", accuracy\*100, "%")

plt.figure(figsize=(20,10)) plot\_tree(clf\_model, filled=True) plt.show()

y\_pred = clf\_model.predict(X\_test) print("Classification report - n", classification\_report(y\_test,y\_pred))

**CLASSIFICATION REPORT FOR DECISION TREE**

Classification report - n precision recall f1-score support

0 0.85 0.85 0.85 2113

1 0.26 0.26 0.26 419

accuracy 0.75 2532

macro avg 0.55 0.56 0.55 2532

weighted avg 0.75 0.75 0.75 2532

**Model 2: Random Forest Classfier**

#import Random Forest Model

from sklearn.ensemble import RandomForestClassifier

#Create a Gaussian Classifier

rfc=RandomForestClassifier(n\_estimators=100,max\_depth=6,criterion="entropy")

#Train the model using the training sets y\_pred=clf.predict(X\_test)

rfc.fit(X\_train\_os,y\_train\_os)

y\_pred\_rfc=rfc.predict(X\_test)

y\_pred\_proba\_rfc = rfc.predict\_proba(X\_test)[:,1]

**Confusion Matrix**

In [58]:

rfc\_matrix = metrics.confusion\_matrix(y\_test, y\_pred\_rfc)

rfc\_matrix

Out[58]:

array([[1613, 114],

[ 24, 275]])

**Classification Report**

In [59]:

print(classification\_report(y\_test, y\_pred\_rfc))

precision recall f1-score support

0 0.99 0.93 0.96 1727

1 0.71 0.92 0.80 299

accuracy 0.93 2026

macro avg 0.85 0.93 0.88 2026

weighted avg 0.94 0.93 0.94 2026

In [60]:

f1\_rfc=metrics.f1\_score(y\_test, y\_pred\_rfc)

rfc\_ap\_df.head(6)

Out[63]:

|  | **ap\_train** | **ap\_test** |
| --- | --- | --- |
| **0** | 0.9100 | 0.853 |
| **1** | 0.9050 | 0.886 |
| **2** | 0.9140 | 0.881 |
| **3** | 0.9010 | 0.869 |
| **4** | 0.9140 | 0.891 |
| **Average** | 0.9088 | 0.876 |

**Model 3: XG Boost**

**Build model for imbalanced data**

In [66]:

from xgboost import XGBClassifier

XGB\_org = XGBClassifier(max\_depth=3,learning\_rate=0.03,n\_estimators=100)

XGB\_org.fit(X\_train,y\_train)

y\_pred\_XGB\_org=XGB\_org.predict(X\_test)

y\_pred\_proba\_XGB\_org = XGB\_org.predict\_proba(X\_test)[:, 1]

y\_pred\_proba\_XGB\_train = XGB\_org.predict\_proba(X\_train)[:, 1]

In [67]:

print(classification\_report(y\_test, y\_pred\_XGB\_org))

precision recall f1-score support

0 0.96 0.98 0.97 1727

1 0.90 0.75 0.82 299

accuracy 0.95 2026

macro avg 0.93 0.87 0.89 2026

weighted avg 0.95 0.95 0.95 2026

In [68]:

metrics.roc\_auc\_score(y\_test, y\_pred\_proba\_XGB\_org)

Out[68]:

0.9778871087372887

In [69]:

metrics.average\_precision\_score(y\_test, y\_pred\_proba\_XGB\_org)

Out[69]:

0.9074435632018839

In [70]:

from xgboost import XGBClassifier

XGB = XGBClassifier(max\_depth=3,learning\_rate=0.03,n\_estimators=100)

XGB.fit(X\_train\_os,y\_train\_os)

y\_pred\_XGB=XGB.predict(X\_test)

y\_pred\_proba\_XGB = XGB.predict\_proba(X\_test)[:, 1]

y\_pred\_proba\_XGB\_train = XGB.predict\_proba(X\_train)[:, 1]

**Confusion Matrix**

In [71]:

XGB\_matrix = metrics.confusion\_matrix(y\_test, y\_pred\_XGB)

XGB\_matrix

Out[71]:

array([[1595, 132],

[ 20, 279]])

**classication report**

In [72]:

print(classification\_report(y\_test, y\_pred\_XGB))

precision recall f1-score support

0 0.99 0.92 0.95 1727

1 0.68 0.93 0.79 299

accuracy 0.92 2026

macro avg 0.83 0.93 0.87 2026

weighted avg 0.94 0.92 0.93 2026

**Precision-Recall Curve**

In [73]:

ap\_xgb=np.round(average\_precision\_score(y\_test, y\_pred\_proba\_XGB),3)

#calculate precision and recall

precision\_xgb, recall\_xgb, thresholds = precision\_recall\_curve(y\_test, y\_pred\_proba\_XGB)

#create precision recall curve

fig, ax = plt.subplots()

ax.plot(recall\_xgb, precision\_xgb)

#add axis labels to plot

ax.set\_title('Precision-Recall Curve for trainning')

ax.set\_ylabel('Precision')

ax.set\_xlabel('Recall')

ax.text(0.9, 0.35,'ap=' + str(ap\_xgb) , fontsize=15)

#display plot

plt.show()

**ROC-AUC curve**

In [74]:

#define metrics

fpr\_xgb, tpr\_xgb, \_ = metrics.roc\_curve(y\_test, y\_pred\_proba\_XGB)

#create ROC curve

auc\_xgb = metrics.roc\_auc\_score(y\_test, y\_pred\_proba\_XGB)

plt.plot(fpr\_xgb,tpr\_xgb,label="roc\_auc="+str(auc\_xgb))

plt.legend(loc=4)

plt.show()

**Hyper Parameter Tunning**

**Retrain model use best parameters**

In [75]:

XGB\_tuned = XGBClassifier(max\_depth=5,learning\_rate=0.03,n\_estimators=150,radom\_state=4211)

XGB\_tuned.fit(X\_train\_os,y\_train\_os)

Out[75]:

XGBClassifier(learning\_rate=0.03, max\_depth=5, n\_estimators=150,

radom\_state=4211)

In [76]:

y\_pred\_XGB\_best=XGB\_tuned.predict(X\_test)

y\_pred\_proba\_XGB\_best = XGB\_tuned.predict\_proba(X\_test)[:, 1]

**Classification Report**

In [77]:

print(classification\_report(y\_test, y\_pred\_XGB\_best))

precision recall f1-score support

0 0.99 0.96 0.97 1727

1 0.81 0.93 0.87 299

accuracy 0.96 2026

macro avg 0.90 0.95 0.92 2026

weighted avg 0.96 0.96 0.96 2026

In [78]:

f1\_best=metrics.f1\_score(y\_test, y\_pred\_XGB\_best)

**Precision Recall Curve**

In [79]:

ap\_best=metrics.average\_precision\_score(y\_test, y\_pred\_proba\_XGB\_best)

#calculate precision and recall

precision\_best, recall\_best, thresholds = precision\_recall\_curve(y\_test, y\_pred\_proba\_XGB\_best)

#create precision recall curve

fig, ax = plt.subplots()

ax.plot(recall\_best, precision\_best)

#add axis labels to plot

ax.set\_title('Precision-Recall Curve for trainning')

ax.set\_ylabel('Precision')

ax.set\_xlabel('Recall')

ax.text(0.9, 0.35,'ap=' + str(ap\_best) , fontsize=15)

#display plot

plt.show()

**Roc Auc Curve**

In [80]:

#define metrics

fpr\_best, tpr\_best, \_ = metrics.roc\_curve(y\_test, y\_pred\_proba\_XGB\_best)

#create ROC curve

auc\_best = metrics.roc\_auc\_score(y\_test, y\_pred\_proba\_XGB\_best)

plt.plot(fpr\_best,tpr\_best,label="roc\_auc="+str(auc\_best))

plt.legend(loc=4)

plt.show()

**5-Fold cross validation**

In [81]:

ap\_train\_list\_xgb=[]

ap\_test\_list\_xgb=[]

for i in range(1,6):

X = df1.iloc[:,2:] # Features

y = df1["Attrition\_Flag"] # Target variable

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=i\*321)

ros = RandomOverSampler(sampling\_strategy=1, random\_state=i\*322)

X\_res\_os, y\_res\_os = ros.fit\_resample(X\_train, y\_train)

X\_train\_os, X\_test\_os, y\_train\_os, y\_test\_os = train\_test\_split(X\_res\_os, y\_res\_os, test\_size=0.2, random\_state=i\*357)

XGB = XGBClassifier(max\_depth=5,learning\_rate=0.03,n\_estimators=150)

XGB.fit(X\_train\_os,y\_train\_os)

y\_pred=XGB.predict(X\_test)

y\_pred\_proba\_XGB = XGB.predict\_proba(X\_test)[:, 1]

y\_pred\_proba\_XGB\_train = XGB.predict\_proba(X\_train)[:, 1]

ap\_train=np.round(metrics.average\_precision\_score(y\_train, y\_pred\_proba\_XGB\_train),3)

ap\_train\_list\_xgb.append(ap\_train)

ap\_test=np.round(metrics.average\_precision\_score(y\_test, y\_pred\_proba\_XGB),3)

ap\_test\_list\_xgb.append(ap\_test)

myplot(y\_test, y\_pred\_proba\_XGB,"ap = " +str(ap\_test))

rfc\_ap\_df=pd.DataFrame({"ap\_train":ap\_train\_list\_xgb,"ap\_test":ap\_test\_list\_xgb})

# Create a list of the average precision values

avg\_list = [rfc\_ap\_df['ap\_train'].mean(), rfc\_ap\_df['ap\_test'].mean()]

# Add the two rows to the dataframe

rfc\_ap\_df.loc['Average'] = avg\_list

rfc\_ap\_df.head(6)

Out[81]:

|  | **ap\_train** | **ap\_test** |
| --- | --- | --- |
| **0** | 0.983 | 0.9540 |
| **1** | 0.984 | 0.9640 |
| **2** | 0.982 | 0.9550 |
| **3** | 0.983 | 0.9440 |
| **4** | 0.983 | 0.9560 |
| **Average** | 0.983 | 0.9546 |

In [82]:

ap\_train\_list\_xgb=[]

ap\_test\_list\_xgb=[]

for i in range(1,6):

X = df1.iloc[:,2:] # Features

y = df1["Attrition\_Flag"] # Target variable

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=i\*321)

ros = RandomOverSampler(sampling\_strategy=1, random\_state=i\*322)

X\_res\_os, y\_res\_os = ros.fit\_resample(X\_train, y\_train)

X\_train\_os, X\_test\_os, y\_train\_os, y\_test\_os = train\_test\_split(X\_res\_os, y\_res\_os, test\_size=0.2, random\_state=i\*357)

XGB = XGBClassifier(max\_depth=5,learning\_rate=0.02,n\_estimators=150)

XGB.fit(X\_train\_os,y\_train\_os)

y\_pred=XGB.predict(X\_test)

y\_pred\_proba\_xgb\_cv = XGB.predict\_proba(X\_test)[:, 1]

y\_pred\_proba\_xgb\_train\_cv = XGB.predict\_proba(X\_train)[:, 1]

ap\_train=np.round(metrics.roc\_auc\_score(y\_train, y\_pred\_proba\_xgb\_train\_cv),3)

ap\_train\_list\_xgb.append(ap\_train)

ap\_test=np.round(metrics.roc\_auc\_score(y\_test, y\_pred\_proba\_xgb\_cv),4)

ap\_test\_list\_xgb.append(ap\_test)

my\_aucplot(y\_test, y\_pred\_proba\_xgb\_cv,"roc\_auc = " +str(ap\_test))

rfc\_ap\_df=pd.DataFrame({"roc\_train":ap\_train\_list\_xgb,"roc\_test":ap\_test\_list\_xgb})

avg\_list = [rfc\_ap\_df['roc\_train'].mean(), rfc\_ap\_df['roc\_test'].mean()]

# Add the two rows to the dataframe

rfc\_ap\_df.loc['Average'] = avg\_list

rfc\_ap\_df.head(6)

Out[82]:

|  | **roc\_train** | **roc\_test** |
| --- | --- | --- |
| **0** | 0.9950 | 0.98760 |
| **1** | 0.9950 | 0.98880 |
| **2** | 0.9940 | 0.98730 |
| **3** | 0.9950 | 0.98380 |
| **4** | 0.9950 | 0.98790 |
| **Average** | 0.9948 | 0.98708 |

**Feature Importance**

In [83]:

explainer\_xgb = shap.TreeExplainer(XGB\_tuned)

shap\_values\_xgb = explainer\_xgb.shap\_values(X\_test)

shap.summary\_plot(shap\_values\_xgb, X\_test, plot\_type="bar")

In [84]:

importances = XGB\_tuned.feature\_importances\_

# Sort feature importances in descending order

indices = np.argsort(importances)[::-1]

# Rearrange feature names so they match the sorted feature importances

names = [X.columns[i] for i in indices]

# Create plot

plt.figure()

# Create plot title

plt.title("Feature Importance")

# Add bars

plt.bar(range(X.shape[1]), importances[indices])

# Add feature names as x-axis labels

plt.xticks(range(X.shape[1]), names, rotation=90)

# Show plot

plt.show()

In [85]:

shap.summary\_plot(shap\_values\_xgb, X\_test)

**Conclusion**

In [86]:

sting\_ap="AP "

plt.title('Precision Recall Curve')

plt.plot(recall\_logreg, precision\_logreg, label=sting\_ap+"Logistic Regression = "+str(round(ap\_logreg, 2)))

# Plot the second precision-recall curve

plt.plot(recall\_rfc, precision\_rfc, label=sting\_ap+"Random Forest = "+str(round(ap\_rfc,2)))

# Plot the third precision-recall curve

plt.plot(recall\_best, precision\_best, label=sting\_ap+"XG Boost = "+str(round(ap\_best, 2)))

# Add a legend to the graph

plt.legend()

# Show the plot

plt.show()

In [87]:

sting\_auc="auc "

plt.title('Roc Auc Curve')

plt.plot(fpr\_logreg, tpr\_logreg, label=sting\_auc+"Logistic Regression = "+str(round(auc\_logreg, 2)))

# Plot the second ROC AUC curve

plt.plot(fpr\_rfc, tpr\_rfc, label=sting\_auc+"Random Forest = "+str(round(auc\_rfc, 2)))

# Plot the third ROC AUC curve

plt.plot(fpr\_best, tpr\_best, label=sting\_auc+"XG Boost = "+str(round(auc\_best, 2)))

# Add a legend to the graph

plt.legend()

# Show the plot

plt.show()

How many transaction did the customer made in the past? What is the revolving balance? what is the ratio of the number of transactions in Q4 over Q1? How many accounts does this customer have with this bank?

|  | **Total\_Trans\_Ct** | **Total\_Trans\_Amt** | **Total\_Revolving\_Bal** | **Total\_Ct\_Chng\_Q4\_Q1** | **Months\_Inactive\_12\_mon** |
| --- | --- | --- | --- | --- | --- |
| **7337** | 66 | 3391 | 0 | 0.833 | 3 |
| **2183** | 79 | 4107 | 629 | 0.580 | 1 |
| **6051** | 67 | 4002 | 1342 | 0.763 | 3 |
| **2332** | 38 | 1334 | 1960 | 0.407 | 4 |
| **5931** | 40 | 2182 | 0 | 0.429 | 3 |

In [93]:

rfc\_pk.predict(Xd\_test)

Out[93]:

array([0, 0, 0, ..., 0, 0, 1])

In [94]:

my\_features\_rfc = pd.DataFrame({'Total\_Trans\_Ct':[66], 'Total\_Trans\_Amt':[3391], 'Total\_Revolving\_Bal': [0], 'Total\_Ct\_Chng\_Q4\_Q1':[0.833], 'Months\_Inactive\_12\_mon':[3]})

rfc\_pk.predict(my\_features\_rfc)

Out[94]:

array([0])

In [95]:

pickle.dump(rfc\_pk,open('rfc.pkl', 'wb'))

**XG Boost**

In [96]:

X\_pk\_xgb = df1[["Total\_Trans\_Ct","Total\_Revolving\_Bal","Total\_Trans\_Amt","Total\_Ct\_Chng\_Q4\_Q1","Total\_Relationship\_Count"]] # Features

y = df1["Attrition\_Flag"]

In [97]:

Xd\_train, Xd\_test, yd\_train, yd\_test = train\_test\_split(X\_pk\_xgb, y, test\_size=0.2)

ros = RandomOverSampler(sampling\_strategy=1)

Xd\_res\_os, yd\_res\_os = ros.fit\_resample(Xd\_train, yd\_train)

Xd\_train\_os, Xd\_test\_os, yd\_train\_os, yd\_test\_os = train\_test\_split(Xd\_res\_os, yd\_res\_os, test\_size=0.2)

xgb\_pk = XGBClassifier(max\_depth=5,learning\_rate=0.03,n\_estimators=150, objective='binary:logistic')

xgb\_pk.fit(Xd\_train\_os, yd\_train\_os)

y\_pred\_xgb\_pk =xgb\_pk.predict(Xd\_test)

y\_pred\_proba\_xgb\_pk = xgb\_pk.predict\_proba(Xd\_test)[:,1]

In [98]:

cm = confusion\_matrix(yd\_test, y\_pred\_xgb\_pk)

# Print the confusion matrix

print(cm)

[[1594 117]

[ 14 301]]

In [99]:

print(classification\_report(yd\_test, y\_pred\_xgb\_pk))

precision recall f1-score support

0 0.99 0.93 0.96 1711

1 0.72 0.96 0.82 315

accuracy 0.94 2026

macro avg 0.86 0.94 0.89 2026

weighted avg 0.95 0.94 0.94 2026

In [100]:

metrics.roc\_auc\_score(yd\_test, y\_pred\_proba\_xgb\_pk)

Out[100]:

0.987785848802798

In [101]:

average\_precision\_score(yd\_test, y\_pred\_proba\_xgb\_pk)

Out[101]:

0.9365375029625733

In [102]:

xgb\_pk.predict\_proba(Xd\_test)

Out[102]:

array([[0.33541834, 0.66458166],

[0.94738215, 0.05261784],

[0.9509682 , 0.04903177],

...,

[0.98892945, 0.01107053],

[0.9904367 , 0.00956332],

[0.98536605, 0.01463393]], dtype=float32)

In [103]:

xgb\_pk.predict(Xd\_test)

Out[103]:

array([1, 0, 0, ..., 0, 0, 0])

In [104]:

my\_features = pd.DataFrame({'Total\_Trans\_Ct':[21], 'Total\_Revolving\_Bal':[1151], 'Total\_Trans\_Amt': [820], 'Total\_Ct\_Chng\_Q4\_Q1':[0.750], 'Total\_Relationship\_Count':[3]})

xgb\_pk.predict(my\_features)

Out[104]:

array([1])

**Prediction Evaluation**

combine\_df.head(10)

Out[114]:

|  | **Y\_actual** | **Y\_proba** | **Dollar\_Overall\_Trx** | **Y\_predict\_0.1** | **Y\_predict\_0.2** | **Y\_predict\_0.3** | **Y\_predict\_0.4** | **Y\_predict\_0.5** | **Y\_predict\_0.6** | **Y\_predict\_0.7** | **Y\_predict\_0.8** | **Y\_predict\_0.9** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **3698** | 0 | 0.021004 | 3982 | False | False | False | False | False | False | False | False | False |
| **1630** | 0 | 0.035142 | 1830 | False | False | False | False | False | False | False | False | False |
| **1978** | 0 | 0.030771 | 3095 | False | False | False | False | False | False | False | False | False |
| **9592** | 0 | 0.014873 | 15574 | False | False | False | False | False | False | False | False | False |
| **9136** | 1 | 0.893042 | 3922 | True | True | True | True | True | True | True | True | False |
| **3487** | 0 | 0.011190 | 4364 | False | False | False | False | False | False | False | False | False |
| **1983** | 0 | 0.066670 | 1927 | False | False | False | False | False | False | False | False | False |
| **4680** | 1 | 0.949487 | 2522 | True | True | True | True | True | True | True | True | True |
| **3303** | 0 | 0.011996 | 4382 | False | False | False | False | False | False | False | False | False |
| **1048** | 1 | 0.954482 | 694 | True | True | True | True | True | True | True | True | True |

In [115]:

combine\_df.iloc[:,-9:].sum()

Out[115]:

Y\_predict\_0.1 642

Y\_predict\_0.2 488

Y\_predict\_0.3 420

Y\_predict\_0.4 376

Y\_predict\_0.5 346

Y\_predict\_0.6 319

Y\_predict\_0.7 296

Y\_predict\_0.8 270

Y\_predict\_0.9 230

dtype: int64