Credit Card Customers Segmentation

Importing Directories

In [1]:
```
import numpy as np
import pandas as pd
import plotly.express as px
import plotly.graph_objects as go
import plotly.io as pio
pio.templates.default = "plotly_white"
```

Data Exploration

In [29]:
```
# Importing Dataset

data = pd.read_csv("CreditCard.csv")
data
```

Out[29]:

	Customer_ID	Age	Gender	Geography	Occupation	Annual_Income	Monthly_Inhand_Sal
0	44929	44	Male	India	Entrepreneur	57180.88	4584.073
1	35586	27	Female	France	Accountant	108462.87	9313.572
2	37586	20	Female	Geramany	Musician	99161.82	8055.485
3	16202	30	Male	France	Engineer	39418.47	3099.872
4	42404	19	Female	France	Mechanic	135452.08	11212.673
...	...	...	...	...	...	...	
24995	4007	35	Female	Scotland	Entrepreneur	123665.00	10145.416
24996	29586	56	Male	Spain	Journalist	31226.77	2878.230
24997	37011	29	Female	Scotland	Manager	38158.80	2992.900
24998	16510	36	Female	Geramany	Lawyer	22276.68	2014.390
24999	32952	43	Female	Geramany	Developer	15928.75	1301.395

25000 rows × 24 columns

In [3]:
```
data.columns
```

Out[3]:
```
Index(['Customer_ID', 'Age', 'Gender', 'Geography', 'Occupation',
       'Annual_Income', 'Monthly_Inhand_Salary', 'Num_Bank_Accounts',
       'Num_Credit_Card', 'Interest_Rate', 'Num_of_Loan', 'Type_of_Loan',
       'Delay_from_due_date', 'Num_of_Delayed_Payment', 'Credit_Mix',
       'Outstanding_Debt', 'Credit_Utilization_Ratio', 'Credit_History_Age',
       'Payment_of_Min_Amount', 'Payment_Behaviour', 'Monthly_Balance',
       'Credit_Score', 'Credit_Score_', 'Status'],
      dtype='object')
```

In [4]:
```
data.shape
```

Out[4]: (25000, 24)

In [5]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25000 entries, 0 to 24999
Data columns (total 24 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Customer_ID               25000 non-null  int64  
 1   Age                       25000 non-null  int64  
 2   Gender                    25000 non-null  object 
 3   Geography                 25000 non-null  object 
 4   Occupation                25000 non-null  object 
 5   Annual_Income             25000 non-null  float64
 6   Monthly_Inhand_Salary     25000 non-null  float64
 7   Num_Bank_Accounts         25000 non-null  int64  
 8   Num_Credit_Card           25000 non-null  int64  
 9   Interest_Rate             25000 non-null  int64  
 10  Num_of_Loan               25000 non-null  int64  
 11  Type_of_Loan              25000 non-null  object 
 12  Delay_from_due_date       25000 non-null  int64  
 13  Num_of_Delayed_Payment    25000 non-null  int64  
 14  Credit_Mix                25000 non-null  int64  
 15  Outstanding_Debt          25000 non-null  float64
 16  Credit_Utilization_Ratio  25000 non-null  float64
 17  Credit_History_Age        25000 non-null  int64  
 18  Payment_of_Min_Amount     25000 non-null  object 
 19  Payment_Behaviour         25000 non-null  object 
 20  Monthly_Balance           25000 non-null  float64
 21  Credit_Score              25000 non-null  object 
 22  Credit_Score_             25000 non-null  int64  
 23  Status                    25000 non-null  object 
dtypes: float64(5), int64(11), object(8)
memory usage: 4.6+ MB
```

In [6]:
```
data.describe()
```

Out[6]:

	Customer_ID	Age	Annual_Income	Monthly_Inhand_Salary	Num_Bank_Accounts
count	25000.000000	25000.000000	25000.000000	25000.000000	25000.000000
mean	26110.091320	33.331080	50347.684595	4183.739401	5.379920
std	14382.902183	10.737929	38347.294059	3190.331422	2.594704
min	1006.000000	14.000000	7005.930000	303.645417	0.000000
25%	13746.000000	24.000000	19275.835000	1621.216666	3.000000
50%	26067.000000	33.000000	36916.420000	3079.875833	6.000000
75%	38605.000000	42.000000	70939.340000	5902.424375	7.000000
max	50999.000000	56.000000	179987.280000	15204.633330	11.000000

In [7]:
```
data.isnull().sum()
```

Out[7]:
```
Customer_ID               0
Age                       0
Gender                    0
Geography                 0
```

```
Occupation                0
Annual_Income             0
Monthly_Inhand_Salary     0
Num_Bank_Accounts         0
Num_Credit_Card           0
Interest_Rate             0
Num_of_Loan               0
Type_of_Loan              0
Delay_from_due_date       0
Num_of_Delayed_Payment    0
Credit_Mix                0
Outstanding_Debt          0
Credit_Utilization_Ratio  0
Credit_History_Age        0
Payment_of_Min_Amount     0
Payment_Behaviour         0
Monthly_Balance           0
Credit_Score              0
Credit_Score_             0
Status                    0
dtype: int64
```

In [8]:
```
data["Payment_of_Min_Amount"].value_counts()
```

Out[8]:
```
Yes    13108
No      8844
NM      3048
Name: Payment_of_Min_Amount, dtype: int64
```

In [9]:
```
data["Credit_Score"].value_counts()
```

Out[9]:
```
Standard    13237
Poor         7314
Good         4449
Name: Credit_Score, dtype: int64
```

In [32]:
```
data["Geography"].value_counts()
```

Out[32]:
```
France      5101
Spain       5065
India       5036
Geramany    4930
Scotland    4868
Name: Geography, dtype: int64
```

Data Visualizations

In [36]:
```
# Exploring the Geography feature to know if the occupation of the person affects credit

fig = px.box(data,
             x="Credit_Score",
             y="Geography",
             color="Credit_Score",
             title="Credit Scores Based on Geography",
             color_discrete_map={'Poor':'red',
                                 'Standard':'yellow',
                                 'Good':"Green"})
fig.show()
```

Note: There's no significant difference in the credit scores among the customers of different occupations

occupations.

In [10]:
```
# Exploring the occupation feature to know if the occupation of the person affects credi

fig = px.box(data,
             x="Occupation",
             color="Credit_Score",
             title="Credit Scores Based on Occupation",
             color_discrete_map={'Poor':'red',
                                 'Standard':'yellow',
                                 'Good':"Green"})
fig.show()
```

Note: There's no significant difference in the credit scores among the customers of different occupations.

In [11]:
```
# let's explore impact of the Annual Income of the person's credit scores

fig = px.box(data,
             x = "Credit_Score",
             y = "Annual_Income",
             color = "Credit_Score",
             title = "Credit Score based on Annual Income",
             color_discrete_map={'Poor':'red',
                                 'Standard':'yellow',
                                'Good':"green"})

fig.update_traces(quartilemethod="exclusive")
fig.show()
```

Note : The above visualization predicts that the credit score is proportional to the more you earn annually.

In [12]:
```
data["Type_of_Loan"].nunique()
```

Out[12]: 5734

In [13]:
```
# Now let's see how many loans you can take at a time for a good credit score:

fig = px.box(data,
             x="Credit_Score",
             y="Num_of_Loan",
             color="Credit_Score",
             title="Credit Scores Based on Number of Loans Taken by the Person",
             color_discrete_map={'Poor':'red',
                                 'Standard':'yellow',
                                 'Good':'green'})

fig.update_traces(quartilemethod="exclusive")
fig.show()
```

Note : To have a good credit score, people should not take more than 1 – 3 loans at a time. Having more than three loans at a time will negatively impact your credit scores.

In [14]:
```
# Now let's see if having more bank accounts impacts credit scores or not:

fig = px.box(data
```

```
fig = px.box(data,
             x="Credit_Score",
             y="Num_Bank_Accounts",
             color="Credit_Score",
             title="Credit Scores Based on Number of Bank Accounts",
             color_discrete_map={'Poor':'red',
                                 'Standard':'yellow',
                                 'Good':'green'})

fig.update_traces(quartilemethod="exclusive")
fig.show()
```

Maintaining more than five accounts is not good for having a good credit score. A person should have 3 – 5 bank accounts only. So having more bank accounts doesn't positively impact credit scores.

In [15]:
```
# Now let's see if having more credit cards impacts credit scores or not:

fig = px.box(data,
             x="Credit_Score",
             y="Num_Credit_Card",
             color="Credit_Score",
             title="Credit Scores Based on Number of Credit Cards",
             color_discrete_map={'Poor':'red',
                                 'Standard':'yellow',
                                 'Good':'green'})

fig.update_traces(quartilemethod="exclusive")
fig.show()
```

Just like the number of bank accounts, having more credit cards will not positively impact your credit scores. Having 4 – 5 credit cards is good for your credit score.

In [16]:
```
#Now let's see the impact on credit scores based on how much average interest you pay on

fig = px.box(data,
             x="Credit_Score",
             y="Interest_Rate",
             color="Credit_Score",
             title="Credit Scores Based on the Average Interest rates",
             color_discrete_map={'Poor':'red',
                                 'Standard':'yellow',
                                 'Good':'green'})

fig.update_traces(quartilemethod="exclusive")
fig.show()
```

If the average interest rate is 4 – 11%, the credit score is good. Having an average interest rate of more than 15% is bad for your credit scores.

In [17]:
```
data["Num_of_Loan"].nunique()
```

Out[17]: 10

In [18]:
```
data["Num_of_Loan"].unique()
```

Out[18]: array([0, 4, 3, 1, 6, 5, 9, 8, 7, 2], dtype=int64)

In [19]:
```
#Now let's see how many loans you can take at a time for a good credit score:

fig = px.box(data,
             x="Credit_Score",
             y="Num_of_Loan",
             color="Credit_Score",
             title="Credit Scores Based on Number of Loans Taken by the Person",
             color_discrete_map={'Poor':'red',
                                 'Standard':'yellow',
                                 'Good':'green'})

fig.update_traces(quartilemethod="exclusive")
fig.show()
```

To have a good credit score, you should not take more than 1 – 3 loans at a time. Having more than three loans at a time will negatively impact your credit scores.

In [20]:
```
# Now let's see if delaying payments on the due date impacts your credit scores or not:

fig = px.box(data,
             x="Credit_Score",
             y="Delay_from_due_date",
             color="Credit_Score",
             title="Credit Scores Based on Average Number of Days Delayed for Credit car
             color_discrete_map={'Poor':'red',
                                 'Standard':'yellow',
                                 'Good':'green'})

fig.update_traces(quartilemethod="exclusive")
fig.show()
```

Note: So you can delay your credit card payment 5 – 14 days from the due date. Delaying your payments for more than 17 days from the due date will impact your credit scores negatively.

In [21]:
```
# Now let's have a look at if frequently delaying payments will impact credit scores or

fig = px.box(data,
             x="Credit_Score",
             y="Num_of_Delayed_Payment",
             color="Credit_Score",
             title="Credit Scores Based on Number of Delayed Payments",
             color_discrete_map={'Poor':'red',
                                 'Standard':'yellow',
                                 'Good':'green'})

fig.update_traces(quartilemethod="exclusive")
fig.show()
```

Note: So delaying 4 – 12 payments from the due date will not affect your credit scores. But delaying more than 12 payments from the due date will affect your credit scores negatively.

In [22]:
```
# Now let's see if having more debt will affect credit scores or not:

fig = px.box(data,
             x="Credit_Score",
             y="Outstanding_Debt",
```

```
            color="Credit_Score",
            title="Credit Scores Based on Outstanding Debt",
            color_discrete_map={'Poor':'red',
                                'Standard':'yellow',
                                'Good':'green'})

fig.update_traces(quartilemethod="exclusive")
fig.show()
```

Note: An outstanding debt of dollar 380 – dollar 1150 will not affect your credit scores. But always having a debt of more than dollar 1338 will affect your credit scores negatively.

In [23]:

```
# Now let's see if having a high credit utilization ratio will affect credit scores or n

fig = px.box(data,
            x="Credit_Score",
            y="Credit_Utilization_Ratio",
            color="Credit_Score",
            title="Credit Scores Based on Credit Utilization Ratio",
            color_discrete_map={'Poor':'red',
                                'Standard':'yellow',
                                'Good':'green'})

fig.update_traces(quartilemethod="exclusive")
fig.show()
```

Note: Credit utilization ratio means your total debt divided by your total available credit. According to the above figure, customer credit utilization ratio doesn't affect their credit scores.

In [24]:

```
# Now let's see how the credit history age of a person affects credit scores:

fig = px.box(data,
            x="Credit_Score",
            y="Credit_History_Age",
            color="Credit_Score",
            title="Credit Scores Based on Credit History Age",
            color_discrete_map={'Poor':'red',
                                'Standard':'yellow',
                                'Good':'green'})

fig.update_traces(quartilemethod="exclusive")
fig.show()
```

Note: According to above graph having a long credit history results in better credit scores.

In [25]:

```
# Now let's see if having a low amount at the end of the month affects credit scores or

fig = px.box(data,
            x="Credit_Score",
            y="Monthly_Balance",
            color="Credit_Score",
            title="Credit Scores Based on Monthly Balance Left",
            color_discrete_map={'Poor':'red',
                                'Standard':'yellow',
                                'Good':'green'})

fig.update_traces(quartilemethod="exclusive")
fig.show()
```

```
fig.show()
```

Note: A monthly balance of less than dollars 300 is bad for credit scores. So, having a high monthly balance in your account at the end of the month is good for your credit scores.

In [27]:
```
data.columns
```

Out[27]:
```
Index(['Customer_ID', 'Age', 'Gender', 'Geography', 'Occupation',
       'Annual_Income', 'Monthly_Inhand_Salary', 'Num_Bank_Accounts',
       'Num_Credit_Card', 'Interest_Rate', 'Num_of_Loan', 'Type_of_Loan',
       'Delay_from_due_date', 'Num_of_Delayed_Payment', 'Credit_Mix',
       'Outstanding_Debt', 'Credit_Utilization_Ratio', 'Credit_History_Age',
       'Payment_of_Min_Amount', 'Payment_Behaviour', 'Monthly_Balance',
       'Credit_Score', 'Credit_Score_', 'Status'],
      dtype='object')
```

--- Convert the target variable 'Churn' in a binary numeric variable i.e. Yes=1 ; No = 0

In [30]:
```
data['Status'] = np.where(data.Status == 'Attrited',1,0)
data
```

Out[30]:

	Customer_ID	Age	Gender	Geography	Occupation	Annual_Income	Monthly_Inhand_Sal
0	44929	44	Male	India	Entrepreneur	57180.88	4584.073
1	35586	27	Female	France	Accountant	108462.87	9313.572
2	37586	20	Female	Geramany	Musician	99161.82	8055.485
3	16202	30	Male	France	Engineer	39418.47	3099.872
4	42404	19	Female	France	Mechanic	135452.08	11212.673
...	...	...	...	...	...	...	
24995	4007	35	Female	Scotland	Entrepreneur	123665.00	10145.416
24996	29586	56	Male	Spain	Journalist	31226.77	2878.230
24997	37011	29	Female	Scotland	Manager	38158.80	2992.900
24998	16510	36	Female	Geramany	Lawyer	22276.68	2014.390
24999	32952	43	Female	Geramany	Developer	15928.75	1301.395

25000 rows × 24 columns

-- Convert all the categorical variables into dummy variables

In [37]:
```
data_dummies = pd.get_dummies(data[["Annual_Income", "Monthly_Inhand_Salary",
                  "Num_Bank_Accounts", "Num_Credit_Card",
                  "Interest_Rate", "Num_of_Loan",
                  "Delay_from_due_date", "Num_of_Delayed_Payment",
                  "Credit_Mix", "Outstanding_Debt",
                  "Credit_History_Age", "Monthly_Balance", 'Credit_Score']]
)
data_dummies.head()
```

Out[37]:

	Annual_Income	Monthly_Inhand_Salary	Num_Bank_Accounts	Num_Credit_Card	Interest_Rate

0	57180.88	4584.073333	3	4	9
1	108462.87	9313.572500	0	5	11
2	99161.82	8055.485000	3	6	8
3	39418.47	3099.872500	5	4	1
4	135452.08	11212.673330	0	3	2

In [76]:
```
data_dummies
```

Out[76]:

	Annual_Income	Monthly_Inhand_Salary	Num_Bank_Accounts	Num_Credit_Card	Interest_R
0	57180.88	4584.073333	3	4	
1	108462.87	9313.572500	0	5	
2	99161.82	8055.485000	3	6	
3	39418.47	3099.872500	5	4	
4	135452.08	11212.673330	0	3	
...	...	...	...	...	
24995	123665.00	10145.416670	2	7	
24996	31226.77	2878.230833	5	7	
24997	38158.80	2992.900000	6	8	
24998	22276.68	2014.390000	5	6	
24999	15928.75	1301.395833	9	9	

25000 rows × 15 columns

Credit Score Classification Model

In [38]:
```
x = data_dummies
y = data[["Status"]]
```

In [39]:
```
# To have a glance at correlation between x table features

x.corr()
```

Out[39]:

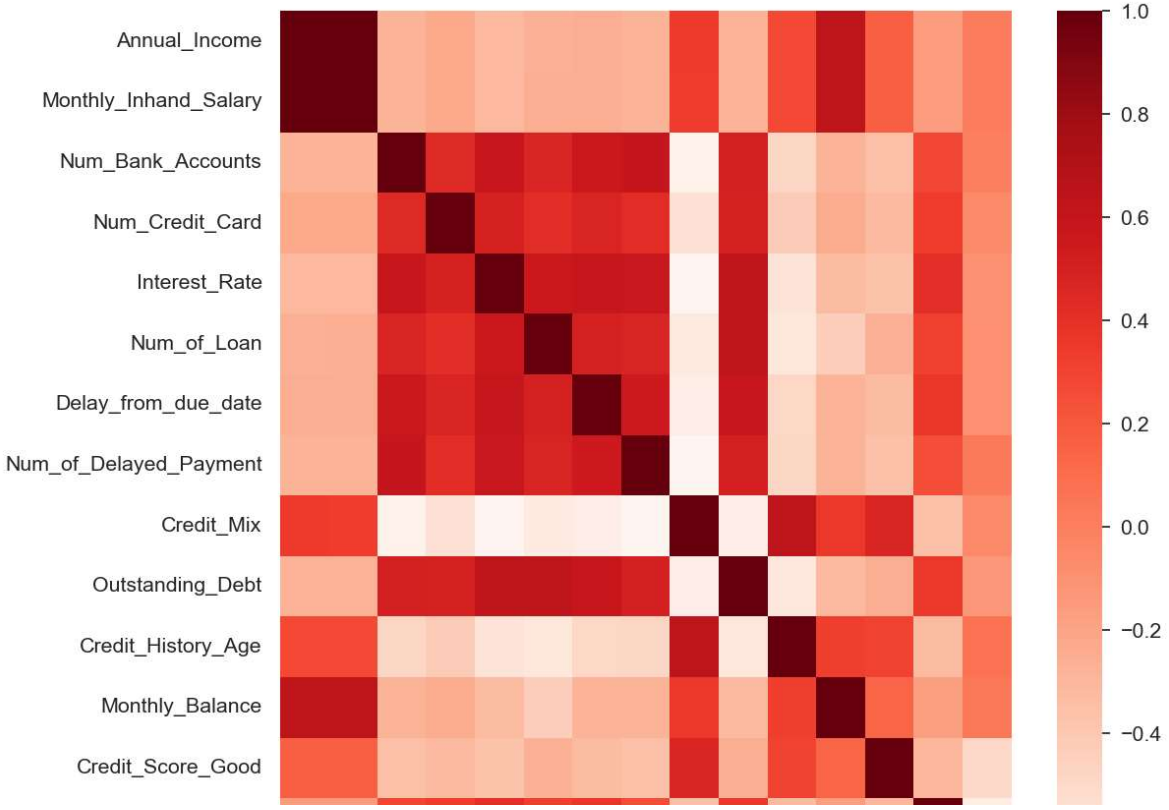
	Annual_Income	Monthly_Inhand_Salary	Num_Bank_Accounts	Num_Cre
Annual_Income	1.000000	0.998150	-0.280478	-(
Monthly_Inhand_Salary	0.998150	1.000000	-0.279822	-(
Num_Bank_Accounts	-0.280478	-0.279822	1.000000	(
Num_Credit_Card	-0.223102	-0.222809	0.443455	
Interest_Rate	-0.308197	-0.307954	0.583391	(
Num_of_Loan	-0.258463	-0.256891	0.473503	(

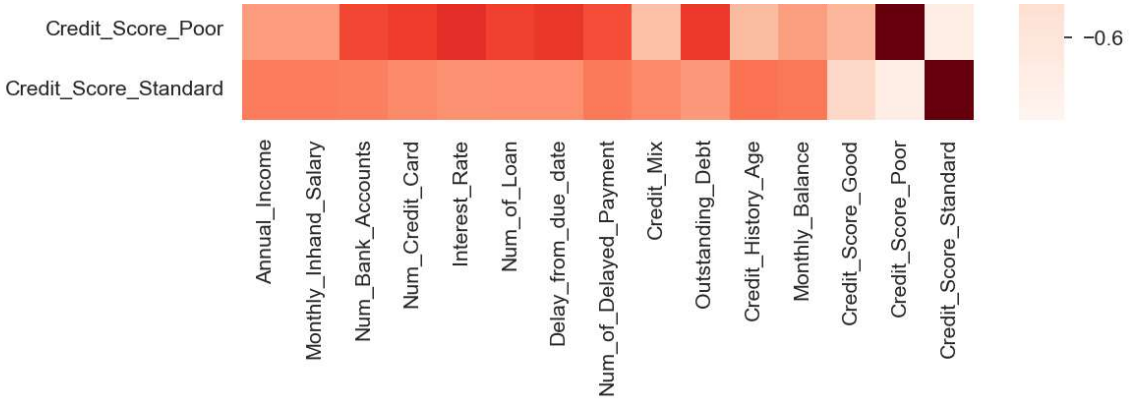
Delay_from_due_date	-0.247363	-0.247101	0.560646	(
Num_of_Delayed_Payment	-0.284821	-0.283725	0.599820	(
Credit_Mix	0.340854	0.339651	-0.722096	-(
Outstanding_Debt	-0.274539	-0.274502	0.506492	(
Credit_History_Age	0.273826	0.273204	-0.481790	-(
Monthly_Balance	0.632449	0.633384	-0.285114	-(
Credit_Score_Good	0.174374	0.171867	-0.346524	-(
Credit_Score_Poor	-0.160723	-0.158991	0.282996	(
Credit_Score_Standard	0.012872	0.013214	0.007598	-(

In [43]:
```
# Plotting graph
%matplotlib inline
import matplotlib.pyplot as plt
plt.style.use("seaborn")
import seaborn as sns
```

In [ ]:
```
# The code is creating a heatmap using the seaborn library to visualize the correlation
plt.figure(figsize=(12,8))
sns.set(font_scale = 1.4)
sns.heatmap(close_returns.corr(), cmap= "Reds", annot= True, annot_kws={"size":15}, vmax
plt.show()
```

In [44]:
```
plt.figure(figsize=(12,12))
sns.set(font_scale = 1.4)
sns.heatmap(x.corr(), cmap="Reds")
plt.show()
```





In [45]:
```
from sklearn.model_selection import train_test_split
```

In [46]:
```
# Split the data into train and test

x_train, x_test, y_train, y_test = train_test_split(x,y, test_size = 0.2, stratify = y,
```

In [47]:
```
print(x.shape, x_train.shape, x_test.shape)
```

```
(25000, 15) (20000, 15) (5000, 15)
```

In [48]:
```
from sklearn.naive_bayes import GaussianNB, MultinomialNB, BernoulliNB
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.naive_bayes import MultinomialNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.ensemble import GradientBoostingClassifier
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score,confusion_matrix,precision_score
```

In [49]:
```
# Model Training

gnb = GaussianNB()
mnb = MultinomialNB()
bnb = BernoulliNB()
svc = SVC(kernel = 'sigmoid', gamma = 1.0)
knc = KNeighborsClassifier()
dtc = DecisionTreeClassifier(max_depth = 5)
lrc = LogisticRegression(solver = 'liblinear', penalty = 'l1')
rfc = RandomForestClassifier(n_estimators = 50, random_state = 2)
abc = AdaBoostClassifier(n_estimators = 50, random_state = 2)
bc = BaggingClassifier(n_estimators = 50, random_state = 2)
etc = ExtraTreesClassifier(n_estimators = 50, random_state = 2)
gbdt = GradientBoostingClassifier(n_estimators = 50, random_state = 2)
xgb = XGBClassifier(n_estimators = 50, random_state = 2)
```

In [50]:
```
clfs = {
    'Support Vector Classification' : svc,
    'KNeighbors Classifier' : knc,
    'Gaussian Naive Bayes' : gnb,
    'Multinomial Naive Bayes': mnb,
    'Bernoulli Naive Bayes' : bnb
```

```
    'Bernoulli Naive Bayes' : bnb,
    'Decision Tree Classifier': dtc,
    'Logistic Regression': lrc,
    'Random Forest Classifier': rfc,
    'AdaBoost Classifier': abc,
    'Bagging Classifier': bc,
    'Extra Trees Classifier': etc,
    'Gradient Boosting Classifier': gbdt,
    'XGB Classifier': xgb
}
```

In [51]:
```
def train_classifier(clf, x_train, y_train, x_test, y_test):
    clf.fit(x_train, y_train)
    y_pred = clf.predict(x_test)
    accuracy = accuracy_score(y_test, y_pred)

    return accuracy
```

In [52]:
```
%%time

train_classifier(svc, x_train, y_train, x_test, y_test)
```

```
C:\Users\lenovo\anaconda3\lib\site-packages\sklearn\utils\validation.py:993: DataConversio
nWarning:

A column-vector y was passed when a 1d array was expected. Please change the shape of y to
(n_samples, ), for example using ravel().

Wall time: 36.4 s
```

Out[52]: 0.5016

In [53]:
```
%%time

accuracy_scores = []

for name, clf in clfs.items():

    accuracy = train_classifier(clf, x_train, y_train, x_test, y_test)

    print("For ", name)
    print("Accuracy - ", accuracy)

    accuracy_scores.append(accuracy)
```

```
C:\Users\lenovo\anaconda3\lib\site-packages\sklearn\utils\validation.py:993: DataConversio
nWarning:

A column-vector y was passed when a 1d array was expected. Please change the shape of y to
(n_samples, ), for example using ravel().

For  Support Vector Classification
Accuracy -  0.5016
C:\Users\lenovo\anaconda3\lib\site-packages\sklearn\neighbors\_classification.py:198: Data
ConversionWarning:

A column-vector y was passed when a 1d array was expected. Please change the shape of y to
(n_samples,), for example using ravel().

C:\Users\lenovo\anaconda3\lib\site-packages\sklearn\neighbors\_classification.py:228: Futu
reWarning:

Unlike other reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode`
typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will change: th
```

```
typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will change: th
e default value of `keepdims` will become False, the `axis` over which the statistic is ta
ken will be eliminated, and the value None will no longer be accepted. Set `keepdims` to T
rue or False to avoid this warning.

For  KNeighbors Classifier
Accuracy -  0.4904
For  Gaussian Naive Bayes
Accuracy -  0.505
For  Multinomial Naive Bayes
Accuracy -  0.5028
For  Bernoulli Naive Bayes
Accuracy -  0.4986
C:\Users\lenovo\anaconda3\lib\site-packages\sklearn\utils\validation.py:993: DataConversio
nWarning:

A column-vector y was passed when a 1d array was expected. Please change the shape of y to
(n_samples, ), for example using ravel().

C:\Users\lenovo\anaconda3\lib\site-packages\sklearn\utils\validation.py:993: DataConversio
nWarning:

A column-vector y was passed when a 1d array was expected. Please change the shape of y to
(n_samples, ), for example using ravel().

C:\Users\lenovo\anaconda3\lib\site-packages\sklearn\utils\validation.py:993: DataConversio
nWarning:

A column-vector y was passed when a 1d array was expected. Please change the shape of y to
(n_samples, ), for example using ravel().

For  Decision Tree Classifier
Accuracy -  0.4842
C:\Users\lenovo\anaconda3\lib\site-packages\sklearn\utils\validation.py:993: DataConversio
nWarning:

A column-vector y was passed when a 1d array was expected. Please change the shape of y to
(n_samples, ), for example using ravel().

C:\Users\lenovo\anaconda3\lib\site-packages\sklearn\svm\_base.py:1206: ConvergenceWarning:

Liblinear failed to converge, increase the number of iterations.

C:\Users\lenovo\AppData\Local\Temp\ipykernel_45084\2342298805.py:2: DataConversionWarning:

A column-vector y was passed when a 1d array was expected. Please change the shape of y to
(n_samples,), for example using ravel().

For  Logistic Regression
Accuracy -  0.501
For  Random Forest Classifier
Accuracy -  0.4862
C:\Users\lenovo\anaconda3\lib\site-packages\sklearn\utils\validation.py:993: DataConversio
nWarning:

A column-vector y was passed when a 1d array was expected. Please change the shape of y to
(n_samples, ), for example using ravel().

For  AdaBoost Classifier
Accuracy -  0.4858
C:\Users\lenovo\anaconda3\lib\site-packages\sklearn\ensemble\_bagging.py:719: DataConversi
onWarning:

A column-vector y was passed when a 1d array was expected. Please change the shape of y to
(n_samples, ), for example using ravel().

For  Bagging Classifier
```

```
For  Bagging Classifier
Accuracy -  0.4878
C:\Users\lenovo\AppData\Local\Temp\ipykernel_45084\2342298805.py:2: DataConversionWarning:

A column-vector y was passed when a 1d array was expected. Please change the shape of y to
(n_samples,), for example using ravel().

For  Extra Trees Classifier
Accuracy -  0.4964
C:\Users\lenovo\anaconda3\lib\site-packages\sklearn\ensemble\_gb.py:494: DataConversionWar
ning:

A column-vector y was passed when a 1d array was expected. Please change the shape of y to
(n_samples, ), for example using ravel().

For  Gradient Boosting Classifier
Accuracy -  0.5002
For  XGB Classifier
Accuracy -  0.4892
Wall time: 1min 20s
```

In [54]:
```
performance_df = pd.DataFrame({'Algorithm':clfs.keys(),'Accuracy':accuracy_scores}).sort
performance_df
```

Out[54]:

	Algorithm	Accuracy
2	Gaussian Naive Bayes	0.5050
3	Multinomial Naive Bayes	0.5028
0	Support Vector Classification	0.5016
6	Logistic Regression	0.5010
11	Gradient Boosting Classifier	0.5002
4	Bernoulli Naive Bayes	0.4986
10	Extra Trees Classifier	0.4964
1	KNeighbors Classifier	0.4904
12	XGB Classifier	0.4892
9	Bagging Classifier	0.4878
7	Random Forest Classifier	0.4862
8	AdaBoost Classifier	0.4858
5	Decision Tree Classifier	0.4842

Data Preprocessing (step to handle imbalanced datasets)

--- SMOTEENN (UpSampling + ENN) It combines two methods: SMOTE for oversampling the minority class and ENN for cleaning the dataset by removing potentially noisy instances.

In [56]:
```
! pip install imbalanced-learn
```

```
Collecting imbalanced-learn
  Downloading imbalanced_learn-0.11.0-py3-none-any.whl (235 kB)
     ------------------------------------ 235.6/235.6 kB 2.1 MB/s eta 0:00:00
Requirement already satisfied: scikit-learn>=1.0.2 in c:\users\lenovo\anaconda3\lib\site-p
ackages (from imbalanced-learn) (1.0.2)
```

```
Collecting joblib>=1.1.1
  Downloading joblib-1.3.2-py3-none-any.whl (302 kB)
     ------------------------------------ 302.2/302.2 kB 4.7 MB/s eta 0:00:00
Requirement already satisfied: numpy>=1.17.3 in c:\users\lenovo\anaconda3\lib\site-package
s (from imbalanced-learn) (1.21.5)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\lenovo\anaconda3\lib\site-
packages (from imbalanced-learn) (2.2.0)
Requirement already satisfied: scipy>=1.5.0 in c:\users\lenovo\anaconda3\lib\site-packages
(from imbalanced-learn) (1.9.1)
Installing collected packages: joblib, imbalanced-learn
  Attempting uninstall: joblib
    Found existing installation: joblib 1.1.0
    Uninstalling joblib-1.1.0:
      Successfully uninstalled joblib-1.1.0
Successfully installed imbalanced-learn-0.11.0 joblib-1.3.2
```

In [57]:
```
from imblearn.combine import SMOTEENN
```

In [64]:
```
X_resampled
```

Out[64]:

	Annual_Income	Monthly_Inhand_Salary	Num_Bank_Accounts	Num_Credit_Card	Interest_Ra
0	7588.460000	663.371667	9	9	2
1	121277.000000	9884.416667	4	1	
2	19657.560000	1349.130000	4	3	
3	30874.430000	2595.869167	4	5	
4	144468.720000	11788.060000	3	3	
...	...	...	...	...	
3000	31720.515149	2691.047707	2	5	
3001	36380.004940	2938.912282	4	6	
3002	92537.830944	7916.357600	1	3	
3003	69278.777677	5980.466628	4	3	
3004	14311.105000	1277.592083	4	3	

3005 rows × 15 columns

In [61]:
```
sm = SMOTEENN()
X_resampled, y_resampled = sm.fit_resample(x,y)
```

In [67]:
```
# Split the data into train and test

xr_train, xr_test, yr_train, yr_test = train_test_split(X_resampled, y_resampled, test_s
```

In [68]:
```
print(X_resampled.shape, xr_train.shape, xr_test.shape)
```

```
(3005, 15) (2404, 15) (601, 15)
```

In [69]:
```
def train_classifier(clf, xr_train, yr_train, xr_test, yr_test):
    clf.fit(xr_train, yr_train)
    yr_pred = clf.predict(xr_test)
```

```
    resampled_accuracy = accuracy_score(yr_test, yr_pred)

    return resampled_accuracy
```

In [70]:
```
%%time

train_classifier(svc, xr_train, yr_train, xr_test, yr_test)
```

```
C:\Users\lenovo\anaconda3\lib\site-packages\sklearn\utils\validation.py:993: DataConversio
nWarning:

A column-vector y was passed when a 1d array was expected. Please change the shape of y to
(n_samples, ), for example using ravel().

Wall time: 538 ms
```

Out[70]: 0.49916805324459235

In [71]:
```
%%time

resampled_accuracy_scores = []

for name, clf in clfs.items():

    accuracy = train_classifier(clf, xr_train, yr_train, xr_test, yr_test)

    print("For ", name)
    print("Accuracy - ", accuracy)

    resampled_accuracy_scores.append(accuracy)
```

```
C:\Users\lenovo\anaconda3\lib\site-packages\sklearn\utils\validation.py:993: DataConversio
nWarning:

A column-vector y was passed when a 1d array was expected. Please change the shape of y to
(n_samples, ), for example using ravel().

For  Support Vector Classification
Accuracy -  0.49916805324459235
For  KNeighbors Classifier
Accuracy -  0.7720465890183028
For  Gaussian Naive Bayes
Accuracy -  0.4925124792013311
For  Multinomial Naive Bayes
Accuracy -  0.540765391014975
For  Bernoulli Naive Bayes
Accuracy -  0.4925124792013311
For  Decision Tree Classifier
Accuracy -  0.5341098169717138
C:\Users\lenovo\anaconda3\lib\site-packages\sklearn\neighbors\_classification.py:198: Data
ConversionWarning:

A column-vector y was passed when a 1d array was expected. Please change the shape of y to
(n_samples,), for example using ravel().

C:\Users\lenovo\anaconda3\lib\site-packages\sklearn\neighbors\_classification.py:228: Futu
reWarning:

Unlike other reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode`
typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will change: th
e default value of `keepdims` will become False, the `axis` over which the statistic is ta
ken will be eliminated, and the value None will no longer be accepted. Set `keepdims` to T
rue or False to avoid this warning.
```

```
C:\Users\lenovo\anaconda3\lib\site-packages\sklearn\utils\validation.py:993: DataConversio
nWarning:

A column-vector y was passed when a 1d array was expected. Please change the shape of y to
(n_samples, ), for example using ravel().

C:\Users\lenovo\anaconda3\lib\site-packages\sklearn\utils\validation.py:993: DataConversio
nWarning:

A column-vector y was passed when a 1d array was expected. Please change the shape of y to
(n_samples, ), for example using ravel().

C:\Users\lenovo\anaconda3\lib\site-packages\sklearn\utils\validation.py:993: DataConversio
nWarning:

A column-vector y was passed when a 1d array was expected. Please change the shape of y to
(n_samples, ), for example using ravel().

C:\Users\lenovo\anaconda3\lib\site-packages\sklearn\utils\validation.py:993: DataConversio
nWarning:

A column-vector y was passed when a 1d array was expected. Please change the shape of y to
(n_samples, ), for example using ravel().

C:\Users\lenovo\anaconda3\lib\site-packages\sklearn\svm\_base.py:1206: ConvergenceWarning:

Liblinear failed to converge, increase the number of iterations.

C:\Users\lenovo\AppData\Local\Temp\ipykernel_45084\770111877.py:2: DataConversionWarning:

A column-vector y was passed when a 1d array was expected. Please change the shape of y to
(n_samples,), for example using ravel().

For  Logistic Regression
Accuracy -  0.5108153078202995
For  Random Forest Classifier
Accuracy -  0.8352745424292846
C:\Users\lenovo\anaconda3\lib\site-packages\sklearn\utils\validation.py:993: DataConversio
nWarning:

A column-vector y was passed when a 1d array was expected. Please change the shape of y to
(n_samples, ), for example using ravel().

For  AdaBoost Classifier
Accuracy -  0.589018302828619
C:\Users\lenovo\anaconda3\lib\site-packages\sklearn\ensemble\_bagging.py:719: DataConversi
onWarning:

A column-vector y was passed when a 1d array was expected. Please change the shape of y to
(n_samples, ), for example using ravel().

For  Bagging Classifier
Accuracy -  0.8519134775374376
C:\Users\lenovo\AppData\Local\Temp\ipykernel_45084\770111877.py:2: DataConversionWarning:

A column-vector y was passed when a 1d array was expected. Please change the shape of y to
(n_samples,), for example using ravel().

For  Extra Trees Classifier
Accuracy -  0.7853577371048253
C:\Users\lenovo\anaconda3\lib\site-packages\sklearn\ensemble\_gb.py:494: DataConversionWar
ning:

A column-vector y was passed when a 1d array was expected. Please change the shape of y to
(n_samples, ), for example using ravel().
```

```
For  Gradient Boosting Classifier
Accuracy -  0.653910149750416
For  XGB Classifier
Accuracy -  0.7903494176372712
Wall time: 5.22 s
```

In [73]:
```
resampled_performance_df = pd.DataFrame({'Algorithm':clfs.keys(),'Accuracy':resampled_ac
resampled_performance_df
```

Out[73]:

	Algorithm	Accuracy
9	Bagging Classifier	0.851913
7	Random Forest Classifier	0.835275
12	XGB Classifier	0.790349
10	Extra Trees Classifier	0.785358
1	KNeighbors Classifier	0.772047
11	Gradient Boosting Classifier	0.653910
8	AdaBoost Classifier	0.589018
3	Multinomial Naive Bayes	0.540765
5	Decision Tree Classifier	0.534110
6	Logistic Regression	0.510815
0	Support Vector Classification	0.499168
2	Gaussian Naive Bayes	0.492512
4	Bernoulli Naive Bayes	0.492512

Model Building

In [74]:
```
model = BaggingClassifier()
model.fit(xr_train, yr_train)
```

```
C:\Users\lenovo\anaconda3\lib\site-packages\sklearn\ensemble\_bagging.py:719: DataConversi
onWarning:

A column-vector y was passed when a 1d array was expected. Please change the shape of y to
(n_samples, ), for example using ravel().
```

Out[74]:
```
BaggingClassifier()
```

In [78]:
```
print("Credit Score Prediction : ")
a = float(input("Annual Income: "))
b = float(input("Monthly Inhand Salary: "))
c = float(input("Number of Bank Accounts: "))
d = float(input("Number of Credit Cards: "))
e = float(input("Interest Rate: "))
f = float(input("Number of Loans: "))
g = float(input("Average number of days delayed by the person: "))
h = float(input("Number of delayed payments: "))
i = input("Credit Mix (Bad: 0, Standard: 1, Good: 3) : ")
j = float(input("Outstanding Debt: "))
k = float(input("Credit History Age: "))
l = float(input("Monthly Balance: "))
m = int(input("Credit Score Good: "))
```

```
n = int(input("Credit Score Poor: "))
o = int(input("Credit Score Standard: "))

features = np.array([[a, b, c, d, e, f, g, h, i, j, k, l, m, n, o]])
print("Predicted Attrition = ", model.predict(features))
```

```
Credit Score Prediction :
Annual Income: 7588.460000
Monthly Inhand Salary: 663.371667
Number of Bank Accounts: 9
Number of Credit Cards: 9
Interest Rate: 29
Number of Loans: 5
Average number of days delayed by the person: 16
Number of delayed payments: 19
Credit Mix (Bad: 0, Standard: 1, Good: 3) : 0
Outstanding Debt: 3948.860000
Credit History Age: 90
```