Credit Card Approval Prediction Model using Machine Learning

CS675 - Machine Learning
Group 11

Ayush Kale -ak2739@njit.edu
Balu Kiran -bp483@njit.edu
Kissan Hari Subbian Venugopal -ks2279@njit.edu

New Jersey Institute of Technology, Newark, New Jersey-07102
14th December 2022

1 Introduction

The credit card approval process done by financial companies is done considering a variety of factors related to individual applicants, such as credit worthiness, loan and repayment history and income standards. In this project, we aim to build a model that can give conclusive results to whether a financial company can approve credit cards to its customers. This model can help an institution in making an accurate decision on whether a card can be approved or denied in order to avoid fraud, which can cause loss. In this project, we will build a usable automatic credit card approval predictor using machine learning techniques. Using Data Analysis and machine learning we will determine the key features and requirements considered by banks when issuing credit cards to their customers. In this study, we will train our data set using linear regression, Support Vector Machine (SVM), Decision Tree and Random Forest, and then through a comprehensive evaluation process we choose the best model based on the effectiveness of these models in predicting credit card approval.

2 Problem Description

Commercial banks receive a lot of applications for credit cards. Manually analyzing these applications is error-prone, and time-consuming. We can automate the task using machine learning. We can do so by developing an “accept or reject” category of customers. With a large dataset and all of the customer’s demographics available, we can build a Machine Learning model to perform such segmentation. The main goal of this project is to build a robust machine learning model for credit card approval prediction.

2.1 Description of Data Set

The data-set used for this project is publicly available at University of California machine learning repository and Kaggle. This data set consists of the records of consumers from the bank. These records are a mix of both approved and rejected data sets. Complete records are altered into information that cannot be interpreted since data is confidential. The target feature of this dataset is the approval column. The approval can be either yes or no. So, we have considered the output to be a binary categorical attribute taking values 0 or 1. Here category ‘0’ means the approval is declined and ‘1’ means the approval is ‘yes’. Dataset contains alphanumeric values.

2.2 Description of classes in categorical features

- Age (Float)
- Debt (Float)
- Married/Single (non-numeric object type)
- Existing Customer (character, object type)
- Qualification (string object type)
- Employed Years (Float)
- Employed (character, object type)
- Credit Score (Integer)
- Driving License (character, object type)
- Gross Income (Float)
- Result (object)
- Owns Car (Flag, character)
- Owns Realty (Flag, character)

Name	**Description**	**Type of Variable**
Flag_Gender	Gender of the customer	Categorical
Flag_Own_Car	If customer owns a car	Categorical
Flag_Own_Realty	If customer owns property	Categorical
Annual Income	Gross income	Numerical
Marital Status	Marital Status	Categorical
Flag_Mobile	If customer has a mobile phone	Categorical
Flag_Work_Phone	If customer has a work phone	Categorical
Flag_Email	If customer has an email	Categorical
Flag_Employed	If the customer is employed	Categorical
Debt	If customer has any debt	Numerical
Balance	Monthly record	Numerical
Credit Score	Credit Score	Numerical

Employed Years	Number of Years customer was employed for	Numerical
Decision	If the credit card was approved	Categorical

3 Methodology

3.1 Platform, Data-set and Algorithms

Programming Language: Python (Jupyter Notebook)

Operating Systems: Windows 11

Data-Set: **https://www.kaggle.com/datasets/rikdifos/credit-card-approval-prediction/code**

Algorithms:

i) Category 1: Logistic Regression
ii) Category 2: Decision Tree Classifier
iii) Category 3: Random Forest Classification
iv) Category 4: Support Vector Mechanism Classification
v) Category 5: K Nearest Neighbor Classification
vi) Category 6: XGBoost Classification

4 Experiments

4.1 Source Code

Importing Required Libraries

Pandas is an open-source Python package that is most widely used for data science/data analysis and machine learning tasks. It is built on top of another package named Numpy, which provides support for multi-dimensional arrays.

NumPy is a Python library used for working with arrays. It also has functions for working in the domain of linear algebra, Fourier transform, and matrices.

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. Matplotlib makes easy things easy and hard things possible. Create publication quality plots. Make interactive figures that can zoom, pan, update.

Seaborn is a library for making statistical graphics in Python. It builds on top of matplotlib and integrates closely with pandas data structures. Seaborn helps you explore and understand your data.

```python
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import numpy as np
import warnings
warnings.filterwarnings('ignore')
```

```python
app_df = pd.read_csv(r'C:\Users\balup\Downloads\archive (1)\application_record.csv')
credit_df = pd.read_csv(r"C:\Users\balup\Downloads\credit_record.csv (1)\credit_record.csv")
```

```python
app_df.describe()
```

	ID	CNT_CHILDREN	AMT_INCOME_TOTAL	DAYS_BIRTH	DAYS_EMPLOYED	FLAG_MOBIL	FLAG_WORK_PHONE	FLAG_PHONE	FLAG_EMAIL	CNT_FAM_MEMBERS
count	4.385570e+05	438557.000000	4.385570e+05	438557.000000	438557.000000	438557.0	438557.000000	438557.000000	438557.000000	438557.000000
mean	6.022176e+06	0.427390	1.875243e+05	-15997.904649	60563.675328	1.0	0.206133	0.287771	0.108207	2.194465
std	5.716370e+05	0.724882	1.100869e+05	4185.030007	138767.799647	0.0	0.404527	0.452724	0.310642	0.897207
min	5.008804e+06	0.000000	2.610000e+04	-25201.000000	-17531.000000	1.0	0.000000	0.000000	0.000000	1.000000
25%	5.609375e+06	0.000000	1.215000e+05	-19483.000000	-3103.000000	1.0	0.000000	0.000000	0.000000	2.000000
50%	6.047745e+06	0.000000	1.607805e+05	-15630.000000	-1467.000000	1.0	0.000000	0.000000	0.000000	2.000000
75%	6.456971e+06	1.000000	2.250000e+05	-12514.000000	-371.000000	1.0	0.000000	1.000000	0.000000	3.000000
max	7.999952e+06	19.000000	6.750000e+06	-7489.000000	365243.000000	1.0	1.000000	1.000000	1.000000	20.000000

```python
app_df.isnull().sum()
```

```
ID                          0
CODE_GENDER                 0
FLAG_OWN_CAR                0
FLAG_OWN_REALTY             0
CNT_CHILDREN                0
AMT_INCOME_TOTAL            0
NAME_INCOME_TYPE            0
NAME_EDUCATION_TYPE         0
NAME_FAMILY_STATUS          0
NAME_HOUSING_TYPE           0
DAYS_BIRTH                  0
DAYS_EMPLOYED               0
FLAG_MOBIL                  0
FLAG_WORK_PHONE             0
FLAG_PHONE                  0
FLAG_EMAIL                  0
OCCUPATION_TYPE        134203
CNT_FAM_MEMBERS             0
dtype: int64
```

```python
# dropping occupation type which has many null values
app_df.drop('OCCUPATION_TYPE', axis=1, inplace=True)
```

```python
# Checking duplicates in 'ID' column
len(app_df['ID']) - len(app_df['ID'].unique())
```

```
47
```

```python
# Dropping duplicate entries from ID column
app_df = app_df.drop_duplicates('ID', keep='last')
```

```python
# Checking Non-Numerical Columns
cat_columns = app_df.columns[(app_df.dtypes =='object').values].tolist()
cat_columns
```

```
['CODE_GENDER',
 'FLAG_OWN_CAR',
 'FLAG_OWN_REALTY',
 'NAME_INCOME_TYPE',
 'NAME_EDUCATION_TYPE',
 'NAME_FAMILY_STATUS',
 'NAME_HOUSING_TYPE']
```

```python
# Checking Numerical Columns
app_df.columns[(app_df.dtypes !='object').values].tolist()
```

```
['ID',
 'CNT_CHILDREN',
 'AMT_INCOME_TOTAL',
 'DAYS_BIRTH',
 'DAYS_EMPLOYED',
 'FLAG_MOBIL',
 'FLAG_WORK_PHONE',
 'FLAG_PHONE',
 'FLAG_EMAIL',
 'CNT_FAM_MEMBERS']
```

```python
# Checking unique values from Categorical Columns

for i in app_df.columns[(app_df.dtypes =='object').values].tolist():
    print(i,'\n')
    print(app_df[i].value_counts())
    print('-----------------------------------------------')
```

```
CODE_GENDER

F    294412
M    144098
Name: CODE_GENDER, dtype: int64
-----------------------------------------------
FLAG_OWN_CAR

N    275428
Y    163082
Name: FLAG_OWN_CAR, dtype: int64
-----------------------------------------------
FLAG_OWN_REALTY

Y    304043
N    134467
Name: FLAG_OWN_REALTY, dtype: int64
-----------------------------------------------
NAME_INCOME_TYPE

Working                 226087
Commercial associate    100739
Pensioner                75483
State servant            36184
Student                     17
Name: NAME_INCOME_TYPE, dtype: int64
-----------------------------------------------
NAME_EDUCATION_TYPE

Secondary / secondary special    301789
Higher education                 117509
Incomplete higher                 14849
Lower secondary                    4051
Academic degree                     312
Name: NAME_EDUCATION_TYPE, dtype: int64
-----------------------------------------------
NAME_FAMILY_STATUS

Married                 299798
Single / not married     55268
Civil marriage           36524
Separated                27249
Widow                    19671
Name: NAME_FAMILY_STATUS, dtype: int64
-----------------------------------------------
NAME_HOUSING_TYPE

House / apartment      393788
With parents            19074
Municipal apartment     14213
Rented apartment         5974
Office apartment         3922
Co-op apartment          1539
Name: NAME_HOUSING_TYPE, dtype: int64
-----------------------------------------------
```

```python
# Checking unique values from Numerical Columns
app_df['CNT_CHILDREN'].value_counts()
```

```
0     304038
1      88518
2      39879
3       5430
4        486
5        133
7          9
9          5
12         4
6          4
14         3
19         1
Name: CNT_CHILDREN, dtype: int64
```

```python
# Checking Min , Max values from 'DAYS_BIRTH' column
print('Min DAYS_BIRTH :', app_df['DAYS_BIRTH'].min(),'\nMax DAYS_BIRTH :', app_df['DAYS_BIRTH'].max())
```

```
Min DAYS_BIRTH : -25201
Max DAYS_BIRTH : -7489
```

```python
# Converting 'DAYS_BIRTH' values from Day to Years
app_df['DAYS_BIRTH'] = round(app_df['DAYS_BIRTH']/-365,0)
app_df.rename(columns={'DAYS_BIRTH':'AGE_YEARS'}, inplace=True)
```

```python
# Checking unique values greater than 0
app_df[app_df['DAYS_EMPLOYED']>0]['DAYS_EMPLOYED'].unique()
```

```
array([365243], dtype=int64)
```

```python
# As mentioned in document, if 'DAYS_EMPLOYED' is positive no, it means person currently unemployed, hence replacing it with 0
app_df['DAYS_EMPLOYED'].replace(365243, 0, inplace=True)
```

```python
# Converting 'DAYS_EMPLOYED' values from Day to Years
app_df['DAYS_EMPLOYED'] = abs(round(app_df['DAYS_EMPLOYED']/-365,0))
app_df.rename(columns={'DAYS_EMPLOYED':'YEARS_EMPLOYED'}, inplace=True)
```

```python
app_df['FLAG_MOBIL'].value_counts()
```

```
1    438510
Name: FLAG_MOBIL, dtype: int64
```

```python
# As all the values in column are 1, hence dropping column
app_df.drop('FLAG_MOBIL', axis=1, inplace=True)
```

```python
app_df['FLAG_WORK_PHONE'].value_counts()
```

```
0    348118
1     90392
Name: FLAG_WORK_PHONE, dtype: int64
```

```python
# This column only contains 0 & 1 values for Mobile no submitted, hence dropping column
app_df.drop('FLAG_WORK_PHONE', axis=1, inplace=True)
```

```python
app_df['FLAG_PHONE'].value_counts()
```

```
0    312323
1    126187
Name: FLAG_PHONE, dtype: int64
```

```python
# This column only contains 0 & 1 values for Phone no submitted, hence dropping column
app_df.drop('FLAG_PHONE', axis=1, inplace=True)
```

```python
app_df['FLAG_EMAIL'].value_counts()
```

```
0    391062
1     47448
Name: FLAG_EMAIL, dtype: int64
```

```python
# This column only contains 0 & 1 values for Email submitted, hence dropping column
app_df.drop('FLAG_EMAIL', axis=1, inplace=True)
```

```python
app_df['CNT_FAM_MEMBERS'].value_counts()
```

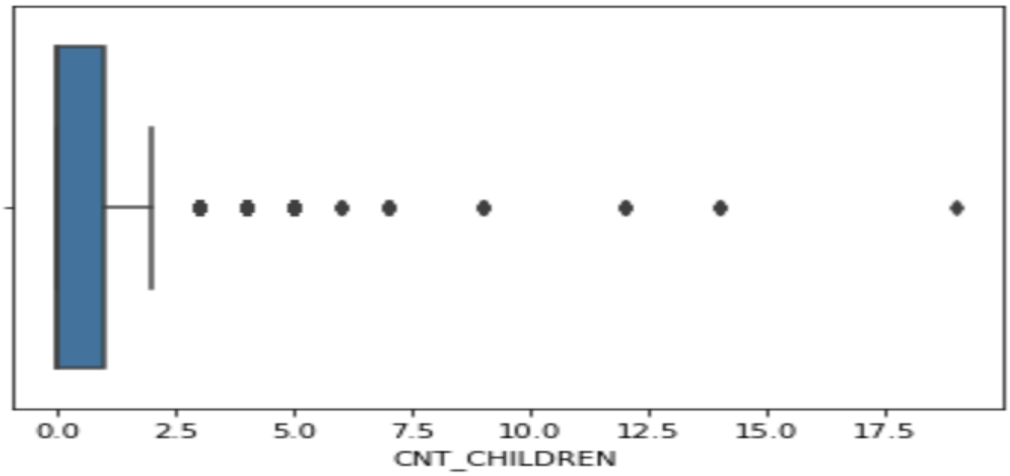
```
2.0     233867
1.0      84483
3.0      77119
4.0      37351
5.0       5081
6.0        459
7.0        124
9.0          9
11.0         5
14.0         4
8.0          4
15.0         3
20.0         1
Name: CNT_FAM_MEMBERS, dtype: int64
```

```python
app_df.head()
```

	ID	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	NAME_INCOME_TYPE	NAME_EDUCATION_TYPE	NAME_FAMILY_STATUS	NAME_HOUSING_TYPE	AGE_YEARS	YEARS_EMPLOYED	CNT_FAM_MEMBERS
0	5008804	M	Y	Y	0	427500.0	Working	Higher education	Civil marriage	Rented apartment	33.0	12.0	2.0
1	5008805	M	Y	Y	0	427500.0	Working	Higher education	Civil marriage	Rented apartment	33.0	12.0	2.0
2	5008806	M	Y	Y	0	112500.0	Working	Secondary / secondary special	Married	House / apartment	59.0	3.0	2.0
3	5008808	F	N	Y	0	270000.0	Commercial associate	Secondary / secondary special	Single / not married	House / apartment	52.0	8.0	1.0
4	5008809	F	N	Y	0	270000.0	Commercial associate	Secondary / secondary special	Single / not married	House / apartment	52.0	8.0	1.0

```
#create plot to detect outliers
sns.boxplot(app_df['CNT_CHILDREN'])
```

<AxesSubplot:xlabel='CNT_CHILDREN'>



Visualization

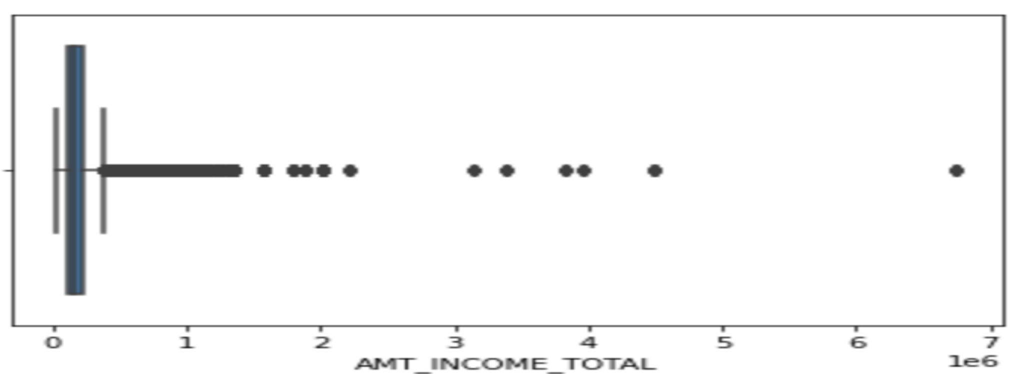
We identified outliers in the dataset using box plots. The data values which exceeded the max value in the box plot are considered as outliers. Then we used the IQR formula to remove them, after that we can see the data without outliers in the modified boxplot.

It is one of the data visualization methods, where the data is distributed on a box and whisker (also known as box-whisker-plot).

Using the IQR, the outlier data points are the ones falling below Q1–1.5 IQR or above Q3 + 1.5 IQR. The Q1 is the 25th percentile and Q3 is the 75th percentile of the dataset, and IQR represents the interquartile range calculated by Q3 minus Q1 (Q3–Q1).

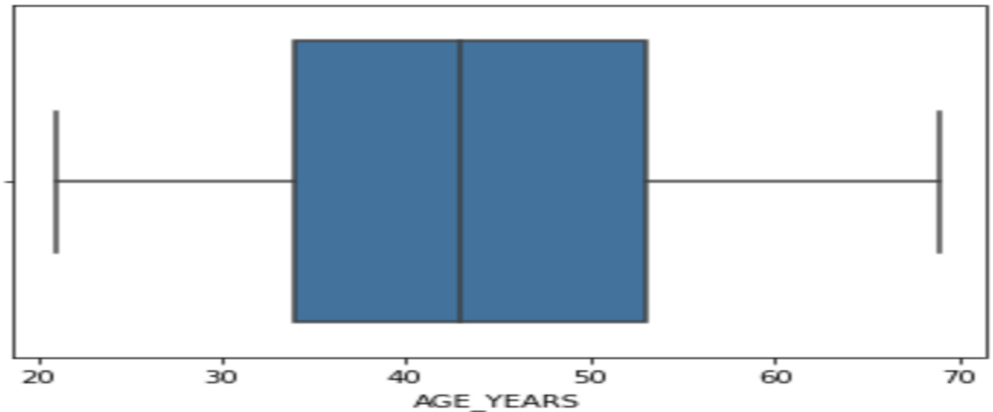
```
sns.boxplot(app_df['AMT_INCOME_TOTAL'])
```

<AxesSubplot:xlabel='AMT_INCOME_TOTAL'>



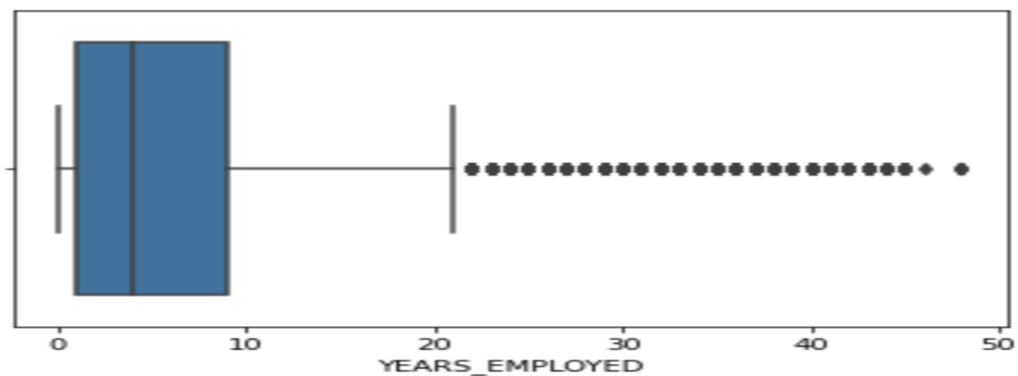
```
sns.boxplot(app_df['AGE_YEARS'])
```

<AxesSubplot:xlabel='AGE_YEARS'>



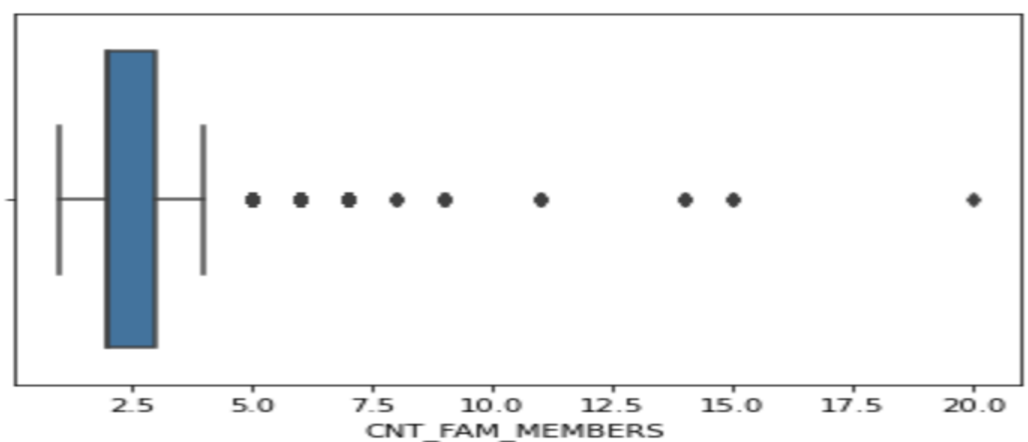

```
sns.boxplot(app_df['YEARS_EMPLOYED'])
```

<AxesSubplot:xlabel='YEARS_EMPLOYED'>




```
sns.boxplot(app_df['CNT_FAM_MEMBERS'])
```

<AxesSubplot:xlabel='CNT_FAM_MEMBERS'>



Removing Outliers

```
high_bound = app_df['CNT_CHILDREN'].quantile(0.999)
print('high_bound :', high_bound)
low_bound = app_df['CNT_CHILDREN'].quantile(0.001)
print('low_bound :', low_bound)
```

```
high_bound : 4.0
low_bound : 0.0
```

```
app_df = app_df[(app_df['CNT_CHILDREN']>=low_bound) & (app_df['CNT_CHILDREN']<=high_bound)]
```

```
high_bound = app_df['AMT_INCOME_TOTAL'].quantile(0.999)
print('high_bound :', high_bound)
low_bound = app_df['AMT_INCOME_TOTAL'].quantile(0.001)
print('low_bound :', low_bound)
```

```
high_bound : 990000.0
low_bound : 36000.0
```

```
app_df = app_df[(app_df['AMT_INCOME_TOTAL']>=low_bound) & (app_df['AMT_INCOME_TOTAL']<=high_bound)]
```

```
high_bound = app_df['YEARS_EMPLOYED'].quantile(0.999)
print('high_bound :', high_bound)
low_bound = app_df['YEARS_EMPLOYED'].quantile(0.001)
print('low_bound :', low_bound)
```

```
high_bound : 40.0
low_bound : 0.0
```

```
app_df = app_df[(app_df['YEARS_EMPLOYED']>=low_bound) & (app_df['YEARS_EMPLOYED']<=high_bound)]
```

```
high_bound = app_df['CNT_FAM_MEMBERS'].quantile(0.999)
print('high_bound :', high_bound)
low_bound = app_df['CNT_FAM_MEMBERS'].quantile(0.001)
print('low_bound :', low_bound)
```

```
high_bound : 6.0
low_bound : 1.0
```

```
app_df = app_df[(app_df['CNT_FAM_MEMBERS']>=low_bound) & (app_df['CNT_FAM_MEMBERS']<=high_bound)]
```

```
app_df.head()
```

	ID	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	NAME_INCOME_TYPE	NAME_EDUCATION_TYPE	NAME_FAMILY_STATUS	NAME_HOUSING_TYPE	AGE_YEARS	YEARS_EMPLOYED	CNT_FAM_MEMBERS
0	5008804	M	Y	Y	0	427500.0	Working	Higher education	Civil marriage	Rented apartment	33.0	12.0	2.0
1	5008805	M	Y	Y	0	427500.0	Working	Higher education	Civil marriage	Rented apartment	33.0	12.0	2.0
2	5008806	M	Y	Y	0	112500.0	Working	Secondary / secondary special	Married	House / apartment	59.0	3.0	2.0
3	5008808	F	N	Y	0	270000.0	Commercial associate	Secondary / secondary special	Single / not married	House / apartment	52.0	8.0	1.0
4	5008809	F	N	Y	0	270000.0	Commercial associate	Secondary / secondary special	Single / not married	House / apartment	52.0	8.0	1.0

Credit Record

```
credit_df.head()
```

	ID	MONTHS_BALANCE	STATUS
0	5001711	0	X
1	5001711	-1	0
2	5001711	-2	0
3	5001711	-3	0
4	5001712	0	C

```
app_df.isnull().sum()
```

```
ID                     0
CODE_GENDER            0
FLAG_OWN_CAR           0
FLAG_OWN_REALTY        0
CNT_CHILDREN           0
AMT_INCOME_TOTAL       0
NAME_INCOME_TYPE       0
NAME_EDUCATION_TYPE    0
NAME_FAMILY_STATUS     0
NAME_HOUSING_TYPE      0
AGE_YEARS              0
YEARS_EMPLOYED         0
CNT_FAM_MEMBERS        0
dtype: int64
```

```
credit_df['STATUS'].value_counts()
```

```
C    442031
0    383120
X    209230
1     11090
5      1693
2       868
3       320
4       223
Name: STATUS, dtype: int64
```

```python
# categorizing 'STATUS' column to binary classification   0 : Good Client and 1 : bad client
credit_df['STATUS'].replace(['C', 'X'],0, inplace=True)
credit_df['STATUS'].replace(['2','3','4','5'],1, inplace=True)
credit_df['STATUS'] = credit_df['STATUS'].astype('int')
```

```python
credit_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1048575 entries, 0 to 1048574
Data columns (total 3 columns):
 #   Column          Non-Null Count    Dtype
---  ------          --------------    -----
 0   ID              1048575 non-null  int64
 1   MONTHS_BALANCE  1048575 non-null  int64
 2   STATUS          1048575 non-null  int32
dtypes: int32(1), int64(2)
memory usage: 20.0 MB
```

```python
credit_df['STATUS'].value_counts(normalize=True)*100
```

```
0     98.646353
1      1.353647
Name: STATUS, dtype: float64
```

```python
credit_df_trans = credit_df.groupby('ID').agg(max).reset_index()
```

```python
credit_df_trans.drop('MONTHS_BALANCE', axis=1, inplace=True)
credit_df_trans.head()
```

	ID	STATUS
0	5001711	0
1	5001712	0
2	5001713	0
3	5001714	0
4	5001715	0

```python
credit_df_trans['STATUS'].value_counts(normalize=True)*100
```

```
0    88.365771
1    11.634229
Name: STATUS, dtype: float64
```

Merging Dataframes

```
# merging the two datasets based on 'ID'
final_df = pd.merge(app_df, credit_df_trans, on='ID', how='inner')
final_df.head()
```

	ID	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	NAME_INCOME_TYPE	NAME_EDUCATION_TYPE	NAME_FAMILY_STATUS	NAME_HOUSING_TYPE	AGE_YEARS	YEARS_EMPLOYED	CNT_FAM_MEMBERS
0	5008804	M	Y	Y	0	427500.0	Working	Higher education	Civil marriage	Rented apartment	33.0	12.0	2.0
1	5008805	M	Y	Y	0	427500.0	Working	Higher education	Civil marriage	Rented apartment	33.0	12.0	2.0
2	5008806	M	Y	Y	0	112500.0	Working	Secondary / secondary special	Married	House / apartment	59.0	3.0	2.0
3	5008808	F	N	Y	0	270000.0	Commercial associate	Secondary / secondary special	Single / not married	House / apartment	52.0	8.0	1.0
4	5008809	F	N	Y	0	270000.0	Commercial associate	Secondary / secondary special	Single / not married	House / apartment	52.0	8.0	1.0

```
# dropping 'ID' column as it is having only unique values (not required for ML Model)
final_df.drop('ID', axis=1, inplace=True)
```

```
# checking if there are still duplicate rows in Final Dataframe
len(final_df) - len(final_df.drop_duplicates())
```

```
25268
```

```
# Dropping duplicate records
final_df = final_df.drop_duplicates()
final_df.reset_index(drop=True ,inplace=True)
```

```
final_df.isnull().sum()
```

```
CODE_GENDER            0
FLAG_OWN_CAR           0
FLAG_OWN_REALTY        0
CNT_CHILDREN           0
AMT_INCOME_TOTAL       0
NAME_INCOME_TYPE       0
NAME_EDUCATION_TYPE    0
NAME_FAMILY_STATUS     0
NAME_HOUSING_TYPE      0
AGE_YEARS              0
YEARS_EMPLOYED         0
CNT_FAM_MEMBERS        0
STATUS                 0
dtype: int64
```

```
final_df['STATUS'].value_counts(normalize=True)*100
```

```
0    78.513294
1    21.486706
Name: STATUS, dtype: float64
```

Data pre-processing and featuring

Machine Learning Model

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(features,label,test_size=0.2,random_state = 10)
```

1. **Logistic Regression**

Logistic regression is a statistical analysis method to predict a binary outcome, such as yes or no, based on prior observations of a data set.

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix

log_model = LogisticRegression()
log_model.fit(x_train, y_train)

print('Logistic Model Accuracy : ', log_model.score(x_test, y_test)*100, '%')

prediction = log_model.predict(x_test)
print('\nConfusion matrix :')
print(confusion_matrix(y_test, prediction))

print('\nClassification report:')
print(classification_report(y_test, prediction))
```

```
Logistic Model Accuracy :  78.84267631103074 %

Confusion matrix :
[[1744    0]
 [ 468    0]]

Classification report:
              precision    recall  f1-score   support

           0       0.79      1.00      0.88      1744
           1       0.00      0.00      0.00       468

    accuracy                           0.79      2212
   macro avg       0.39      0.50      0.44      2212
weighted avg       0.62      0.79      0.70      2212
```

2. Decision Tree classification

A decision tree is a non-parametric supervised learning algorithm, which is utilized for both classification and regression tasks. It has a hierarchical tree structure, which consists of a root node, branches, internal nodes and leaf nodes.

```
from sklearn.tree import DecisionTreeClassifier

decision_model = DecisionTreeClassifier(max_depth=12,min_samples_split=8)

decision_model.fit(x_train, y_train)

print('Decision Tree Model Accuracy : ', decision_model.score(x_test, y_test)*100, '%')

prediction = decision_model.predict(x_test)
print('\nConfusion matrix :')
print(confusion_matrix(y_test, prediction))

print('\nClassification report:')
print(classification_report(y_test, prediction))
```

```
Decision Tree Model Accuracy :  73.4629294755877 %

Confusion matrix :
[[1613  131]
 [ 456   12]]

Classification report:
              precision    recall  f1-score   support

           0       0.78      0.92      0.85      1744
           1       0.08      0.03      0.04       468

    accuracy                           0.73      2212
   macro avg       0.43      0.48      0.44      2212
weighted avg       0.63      0.73      0.68      2212
```

3. Random Forest classification

Random forests are a variant of the bagging method with the basic difference that the basic classier or regressor in random forests is always a decision tree.

Another property of random forests is that when training a tree, the search for the optimum split is limited to a subset of the original features chosen at random. Each split node has a different set of random subsets. The idea is to add more randomness to the learning mechanism in order to try to decorrelate the prediction errors of the individual trees

```
from sklearn.ensemble import RandomForestClassifier

RandomForest_model = RandomForestClassifier(n_estimators=250,
                                            max_depth=12,
                                            min_samples_leaf=16)

RandomForest_model.fit(x_train, y_train)

print('Random Forest Model Accuracy : ', RandomForest_model.score(x_test, y_test)*100, '%')

prediction = RandomForest_model.predict(x_test)
print('\nConfusion matrix :')
print(confusion_matrix(y_test, prediction))

print('\nClassification report:')
print(classification_report(y_test, prediction))
```

```
Random Forest Model Accuracy :  78.84267631103074 %

Confusion matrix :
[[1744    0]
 [ 468    0]]

Classification report:
              precision    recall  f1-score   support

           0       0.79      1.00      0.88      1744
           1       0.00      0.00      0.00       468

    accuracy                           0.79      2212
   macro avg       0.39      0.50      0.44      2212
weighted avg       0.62      0.79      0.70      2212
```

4. Support Vector Machine classification

Support Vector Machine(SVM) is a supervised machine learning algorithm used for both classification and regression. The objective of the SVM algorithm is to find a hyperplane in an N-dimensional space that distinctly classifies the data points.

```python
from sklearn.svm import SVC

svc_model = SVC()

svc_model.fit(x_train, y_train)

print('Support Vector Classifier Accuracy : ', svc_model.score(x_test, y_test)*100, '%')

prediction = svc_model.predict(x_test)
print('\nConfusion matrix :')
print(confusion_matrix(y_test, prediction))

print('\nClassification report:')
print(classification_report(y_test, prediction))
```

```
Support Vector Classifier Accuracy :  78.84267631103074 %

Confusion matrix :
[[1744    0]
 [ 468    0]]

Classification report:
              precision    recall  f1-score   support

           0       0.79      1.00      0.88      1744
           1       0.00      0.00      0.00       468

    accuracy                           0.79      2212
   macro avg       0.39      0.50      0.44      2212
weighted avg       0.62      0.79      0.70      2212
```

5. K Nearest Neighbor classification

The k-nearest neighbors algorithm is a non-parametric supervised learning method.*k*-NN is a type of classification where the function is only approximated locally and all computation is deferred until function evaluation. Since this algorithm relies on distance for classification, if the features represent different physical units or come in vastly different scales then normalizing the training data can improve its accuracy dramatically.

```
from sklearn.neighbors import KNeighborsClassifier

knn_model = KNeighborsClassifier(n_neighbors = 7)

knn_model.fit(x_train, y_train)

print('KNN Model Accuracy : ', knn_model.score(x_test, y_test)*100, '%')

prediction = knn_model.predict(x_test)
print('\nConfusion matrix :')
print(confusion_matrix(y_test, prediction))

print('\nClassification report:')
print(classification_report(y_test, prediction))
```

```
KNN Model Accuracy :  77.03435804701627 %

Confusion matrix :
[[1689   55]
 [ 453   15]]

Classification report:
              precision    recall  f1-score   support

           0       0.79      0.97      0.87      1744
           1       0.21      0.03      0.06       468

    accuracy                           0.77      2212
   macro avg       0.50      0.50      0.46      2212
weighted avg       0.67      0.77      0.70      2212
```

6. XGBoost classification

XGBoost is a popular and efficient open-source implementation of the gradient boosted trees algorithm. Gradient boosting is a supervised learning algorithm, which attempts to accurately predict a target variable by combining the estimates of a set of simpler, weaker models.

```
from xgboost import XGBClassifier

XGB_model = XGBClassifier()

XGB_model.fit(x_train, y_train)

print('XGBoost Model Accuracy : ', XGB_model.score(x_test, y_test)*100, '%')

prediction = XGB_model.predict(x_test)
print('\nConfusion matrix :')
print(confusion_matrix(y_test, prediction))

print('\nClassification report:')
print(classification_report(y_test, prediction))
```

```
XGBoost Model Accuracy :  75.72332730560579 %

Confusion matrix :
[[1664   80]
 [ 457   11]]

Classification report:
              precision    recall  f1-score   support

           0       0.78      0.95      0.86      1744
           1       0.12      0.02      0.04       468

    accuracy                           0.76      2212
   macro avg       0.45      0.49      0.45      2212
weighted avg       0.64      0.76      0.69      2212
```

5 Conclusions and Future Works

5.1 Validation

K-Fold Cross Validation

```
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
kfold = KFold(10)
```

```
# Logistic Regression

results=cross_val_score(log_model,features,label,cv=kfold)
print(results*100,'\n')

print(np.mean(results)*100)
```

```
[78.93309222 81.55515371 82.36889693 79.74683544 74.23146474 83.18264014
 81.19349005 80.19891501 80.09049774 63.6199095 ]

78.51208954857502
```

```
# Random Forest classification

results=cross_val_score(RandomForest_model,features,label,cv=kfold)
print(results*100,'\n')

print(np.mean(results)*100)
```

```
[78.93309222 81.55515371 82.36889693 79.74683544 74.23146474 83.18264014
 81.19349005 80.19891501 80.09049774 63.6199095 ]

78.51208954857502
```

```
# Decision Tree classification

results=cross_val_score(decision_model,features,label,cv=kfold)
print(results*100,'\n')

print(np.mean(results)*100)
```

```
[77.93851718 79.65641953 80.47016275 78.48101266 72.78481013 81.73598553
 80.1084991  78.11934901 77.37556561 63.07692308]

76.97472445648172
```

```
# Support Vector Machine classification

results=cross_val_score(svc_model,features,label,cv=kfold)
print(results*100,'\n')

print(np.mean(results)*100)
```

```
[78.93309222 81.55515371 82.36889693 79.74683544 74.23146474 83.18264014
 81.19349005 80.19891501 80.09049774 63.6199095 ]

78.51208954857502
```

```
# K Nearest Neighbor classification

results=cross_val_score(knn_model,features,label,cv=kfold)
print(results*100,'\n')

print(np.mean(results)*100)
```

```
[78.02893309 80.37974684 81.10307414 78.57142857 73.68896926 82.00723327
 80.1084991  79.20433996 79.09502262 63.25791855]

77.54451654079352
```

```
# XGBoost classification

results=cross_val_score(XGB_model,features,label,cv=kfold)
print(results*100,'\n')

print(np.mean(results)*100)
```

```
[78.57142857 80.8318264  81.91681736 78.93309222 73.41772152 82.82097649
 80.8318264  79.65641953 79.00452489 63.52941176]

77.95140451506795
```

5.2 Model Result

	Model	Accuracy %
0	Logistic Regression	78.842676
1	Decision Tree Classifier	73.462929
2	Random Forest classification	78.842676
3	Support Vector Machine classification	78.842676
4	K Nearest Neighbor classification	77.034358
5	XGBoost classification	75.723327

6 References

1. Koh, H. C., & Chan, K. L. G. (2002). Data mining and customer relationship marketing in the banking industry. Singapore Management Review, 24(2), 1–27. 2. S. B. Kotsiantis. Supervised Machine Learning: A Review of Classification Techniques. Informatica 31 (2007) 249-268 Web.

2. https://archive.ics.uci.edu/ml/machine-learning-databases/00350/ retrieved October 5th, 2022 from https://archive.ics.uci.edu/ml/index.php

3. Seanny (2018),Credit Card Dataset for Machine Learning, Version 1,Retrieved October 5th, 2022 from https://www.kaggle.com/datasets/rikdifos/credit-card-approval-prediction