



# Credit Card Approval Prediction

Project Hand-out:

1. Utsav Johri
2. Hrijumana Dubey
3. Akshat Vishwakarma
4. Siddharth Singh

SmartInternz
www.smartinternz.com

# Credit Card Approval Prediction

Credit Card Approval Prediction is a machine learning-based system designed to assess the likelihood of a credit card application being approved or denied. By analyzing various factors such as income, credit score, employment status, and financial history, this project aims to provide accurate predictions to financial institutions, helping them streamline the approval process, minimize risk, and improve customer satisfaction.

## Scenario 1: Risk Assessment

Financial institutions can use the credit card approval predictions to evaluate the risk associated with each applicant. By considering factors such as credit history, income stability, and debt-to-income ratio, banks can make informed decisions about approving or denying credit card applications, reducing the risk of defaults and losses.

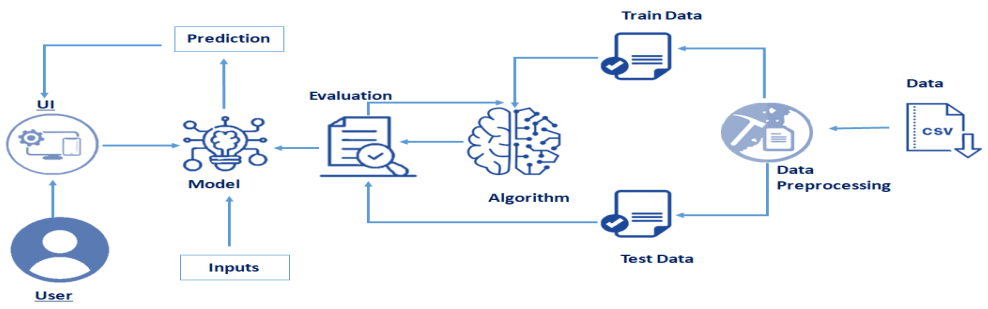
## Scenario 2: Personalized Offers

Based on the prediction results, banks can tailor personalized credit card offers to different customer segments. For example, applicants with higher credit scores and stable incomes may receive offers for premium cards with exclusive benefits, while those with limited credit history may be offered starter cards with lower credit limits.

## Scenario 3: Fraud Detection

Credit card approval predictions can also be utilized for fraud detection purposes. By analyzing application data and detecting inconsistencies or red flags, financial institutions can identify potential fraudulent activities early on, preventing unauthorized access and protecting both customers and the institution from financial losses.

## Technical Architecture:



# Project Flow:

- User interacts with the UI to enter the input.
- Entered input is analyzed by the model which is integrated.
- Once model analyses the input the prediction is showcased on the UI

To accomplish this, we have to complete all the activities listed below,

- Data Collection.
    - Collect the dataset or Create the dataset
- Data Visualization
    - Univariate analysis
    - Multivariate analysis
    - Descriptive analysis
- Data Pre-processing
    - Checking for null values
    - Drop unwanted features
    - Data Cleaning and merging
    - Handling categorical data
    - Splitting Data into Train and Test.
- Model Building
    - Import the model building Libraries
    - Initializing the model
    - Training and testing the model
    - Evaluation of Model
    - Save the Model
- Application Building
    - Create an HTML file
    - Build a Python Code

# Prior knowledge

You must have prior knowledge of following topics to complete this project.

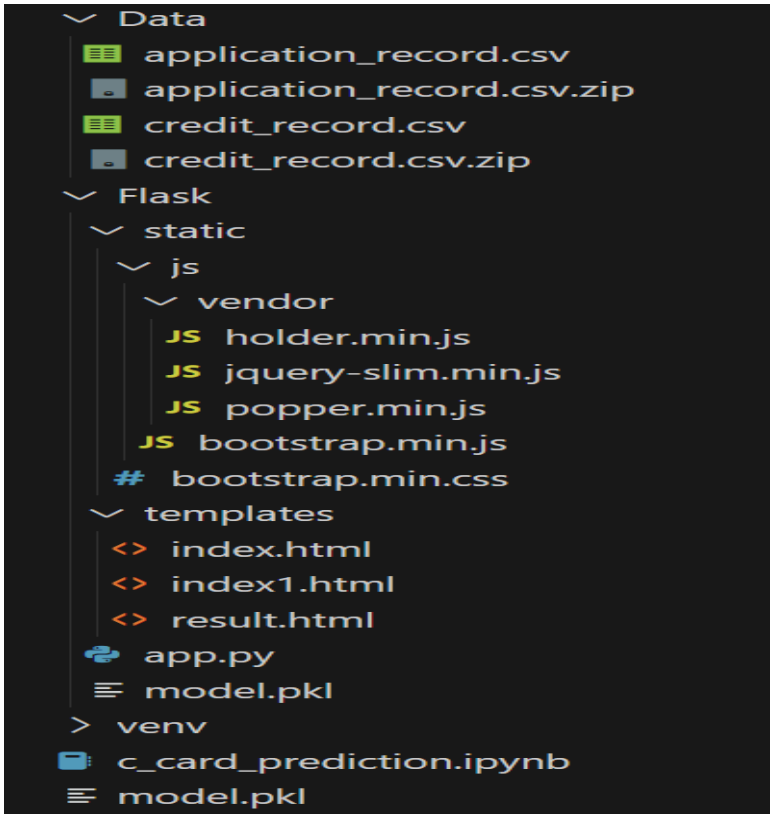
- **ML Concepts**
    - Supervised Learning: https://www.javatpoint.com/supervised-machine-learning
    - Unsupervised Learning: https://www.javatpoint.com/unsupervised-machine-learning

- Random Forest: https://www.javatpoint.com/machine-learning-random-forest-algorithm
- Hyper-parameter tuning:https://towardsdatascience.com/hyperparameter-tuning-the-random-forest-in-python-using-scikit-learn-28d2aa77dd74
- Evaluation metrics: https://www.analyticsvidhya.com/blog/2019/08/11-important-model-evaluation-error-metrics/

- **Flask Basics**: https://youtu.be/lj4I_CvBnt0?si=PVLFbWJHJM1uUjBX

# Project Structure

Create a Project folder which contains files as shown below:



● We are building a flask application which needs HTML pages stored in the templates folder and a python script app.py for scripting.
● 'model.pkl' is our saved model. Further we will use this model for flask integration.
● Data Folder contains the Dataset used
● The Notebook file contains procedure for building the model.

# Milestone 1: Define Problem / Problem Understanding

## Activity 1: Specify the business problem

Financial institutions face challenges in accurately and efficiently assessing credit card applications. Manual processes or outdated systems may lead to delayed decisions, increased risk of default, or missed opportunities to serve customers effectively.

The goal of this project is to develop a machine learning-based **Credit Card Approval Prediction** system that analyzes applicant data such as income, credit score, employment status, and financial history to predict the likelihood of credit card approval. This helps financial institutions streamline the approval process, minimize risk, detect potential fraud, and deliver personalized credit solutions to different customer segments.

## Activity 2: Business requirements

A credit card approval prediction project can involve various business requirements based on the needs of financial institutions. Some potential requirements may include:

- **Accurate and up-to-date predictions:** The system should utilize recent and reliable applicant data (e.g., income, credit score, employment status) to ensure prediction accuracy and relevance to current financial standards.

- **Risk assessment capability:** The system should be able to evaluate the financial risk of approving a credit card application by analyzing credit history, income stability, and debt-to-income ratios.

- **Compliance:** The system should comply with all relevant banking and financial regulations, including data privacy and fair lending practices.

- **Personalization:** The system should be capable of generating personalized credit card offers based on applicant profiles, helping banks provide targeted services.

- **Fraud detection:** The system should include mechanisms to flag suspicious or inconsistent application data to help prevent fraudulent approvals.

- **User-friendly interface:** The prediction system should be intuitive and easy to use for bank staff and financial analysts, ensuring smooth integration into existing workflows.

## Activity 3: Literature Survey

A literature survey for a credit card approval prediction project involves reviewing relevant research and practices in credit scoring and machine learning. Key findings include:

- **Common Algorithms**:
  - Logistic Regression, Decision Trees, Random Forests, Support Vector Machines (SVM), and Gradient Boosting are frequently used for prediction tasks in financial services.
- **Evaluation Metrics**:
  - Accuracy, Precision, Recall, F1-score, and ROC-AUC are standard metrics used to evaluate model performance.
- **Challenges Identified**:
  - Bias in training data leading to unfair predictions.
  - Lack of interpretability in complex models.
  - Limited support for real-time predictions in many systems.
- **Popular Datasets Used**:
  - UCI Credit Approval Dataset, German Credit Data, and Kaggle financial datasets.
- **Best Practices**:
  - Emphasis on data preprocessing (handling missing data, scaling, encoding).
  - Fairness-aware modeling and explainable AI techniques are gaining importance.

These insights will support the design and implementation of a robust and ethical credit card approval prediction system.

## Activity 4: Social or Business Impact.

- **Social Impact:** Improved financial inclusion and customer experience — By

accurately predicting credit card approval outcomes, the system helps financial institutions make fair and informed decisions. This reduces unnecessary application rejections and improves customer satisfaction.

- **Business Model/Impact:** Enhanced decision-making and operational efficiency — Financial institutions can reduce risk, detect potential fraud early, and offer personalized credit card solutions. This helps streamline processes, minimize losses from defaults, and expand customer outreach through targeted offers.

# Milestone 2: Data Collection

ML depends heavily on data. It is the most crucial aspect that makes algorithm training possible. So, this section allows you to download the required dataset.

### Activity 1: Collect the dataset

There are many popular open sources for collecting the data. Eg: kaggle.com, UCI repository, etc. In this project we have used .csv data. This data is downloaded from kaggle.com. Please refer to the link given below to download the dataset.

Link: https://www.kaggle.com/namphuengauawatcharo/credit-card-approval-prediction/data

### Activity 1.1: Importing the libraries

Import the necessary libraries as shown in the image.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
#from imblearn.combine import SMOTETomek
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split, RandomizedSearchCV
from sklearn.preprocessing import OneHotEncoder
from sklearn.metrics import classification_report, confusion_matrix, f1_score
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.tree import DecisionTreeClassifier
```

## Activity 1.2: Read the Dataset

Our dataset format might be in .csv, excel files, .txt, .json, etc. We can read the dataset with the help of pandas. In pandas we have a function called read_csv() to read the dataset. As a parameter we have to give the directory of the csv file.

```
app = pd.read_csv('C:/Users/johri/Credit_Card_Approval_Prediction/Data/application_record.csv')
credit = pd.read_csv('C:/Users/johri/Credit_Card_Approval_Prediction/Data/credit_record.csv')
```

```
app.head()
```

✓ 0.0s Python

| | ID | CODE_GENDER | FLAG_OWN_CAR | FLAG_OWN_REALTY | CNT_CHILDREN | AMT_INCOME_TOTAL | NAME_INCOME_TYPE | NAME_EDUCATION_TYPE | N |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 5008804 | M | Y | Y | 0 | 427500.0 | Working | Higher education | |
| 1 | 5008805 | M | Y | Y | 0 | 427500.0 | Working | Higher education | |
| 2 | 5008806 | M | Y | Y | 0 | 112500.0 | Working | Secondary / secondary special | |
| 3 | 5008808 | F | N | Y | 0 | 270000.0 | Commercial associate | Secondary / secondary special | |
| 4 | 5008809 | F | N | Y | 0 | 270000.0 | Commercial associate | Secondary / secondary special | |

# Milestone 3: Exploratory Data Analysis

As the dataset is downloaded. Let us read and understand the data properly with the help of some visualisation techniques and some analysing techniques.
**Note:** There are a number of techniques for understanding the data. But here we have used some of it. In an additional way, you can use multiple techniques.

## Activity 1: Descriptive statistical

Descriptive analysis is to study the basic features of data with the statistical process. Here pandas has a worthy function called describe. With this describe function we can understand the unique, top and frequent values of categorical features. And we can find mean, std, min, max and percentile values of continuous features.

```
app.describe()
```

✓ 0.1s Python

| | ID | CNT_CHILDREN | AMT_INCOME_TOTAL | DAYS_BIRTH | DAYS_EMPLOYED | FLAG_MOBIL | FLAG_WORK_PHONE | FLAG_PHONE | FLAG_EM |
|---|---|---|---|---|---|---|---|---|---|
| count | 4.385570e+05 | 438557.000000 | 4.385570e+05 | 438557.000000 | 438557.000000 | 438557.0 | 438557.000000 | 438557.000000 | 438557.000( |
| mean | 6.022176e+06 | 0.427390 | 1.875243e+05 | -15997.904649 | 60563.675328 | 1.0 | 0.206133 | 0.287771 | 0.108 |
| std | 5.716370e+05 | 0.724882 | 1.100869e+05 | 4185.030007 | 138767.799647 | 0.0 | 0.404527 | 0.452724 | 0.310( |
| min | 5.008804e+06 | 0.000000 | 2.610000e+04 | -25201.000000 | -17531.000000 | 1.0 | 0.000000 | 0.000000 | 0.000( |
| 25% | 5.609375e+06 | 0.000000 | 1.215000e+05 | -19483.000000 | -3103.000000 | 1.0 | 0.000000 | 0.000000 | 0.000( |
| 50% | 6.047745e+06 | 0.000000 | 1.607805e+05 | -15630.000000 | -1467.000000 | 1.0 | 0.000000 | 0.000000 | 0.000( |
| 75% | 6.456971e+06 | 1.000000 | 2.250000e+05 | -12514.000000 | -371.000000 | 1.0 | 0.000000 | 1.000000 | 0.000( |
| max | 7.999952e+06 | 19.000000 | 6.750000e+06 | -7489.000000 | 365243.000000 | 1.0 | 1.000000 | 1.000000 | 1.000( |

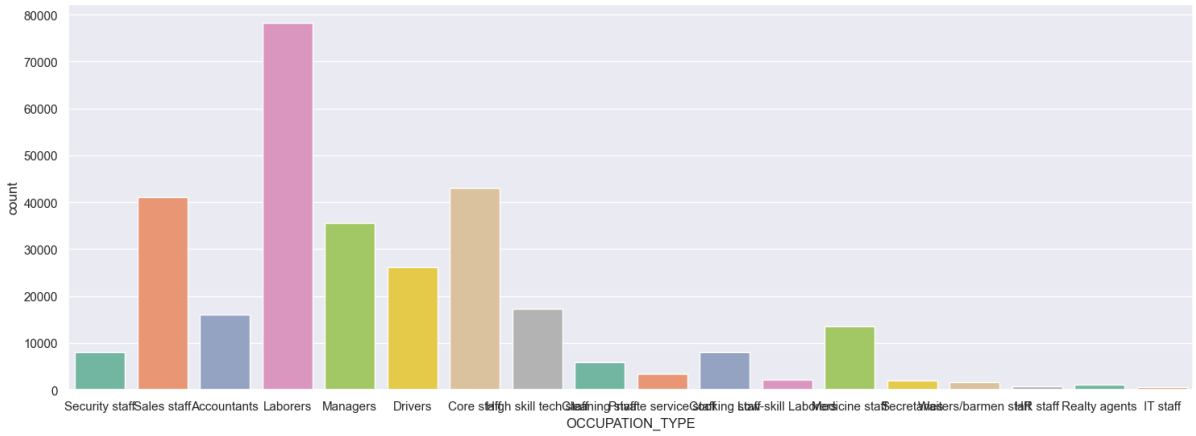
## Activity 2: Visual analysis

Visual analysis is the process of using visual representations, such as charts, plots, and graphs, to explore and understand data. It is a way to quickly identify patterns, trends, and outliers in the data, which can help to gain insights and make informed decisions.

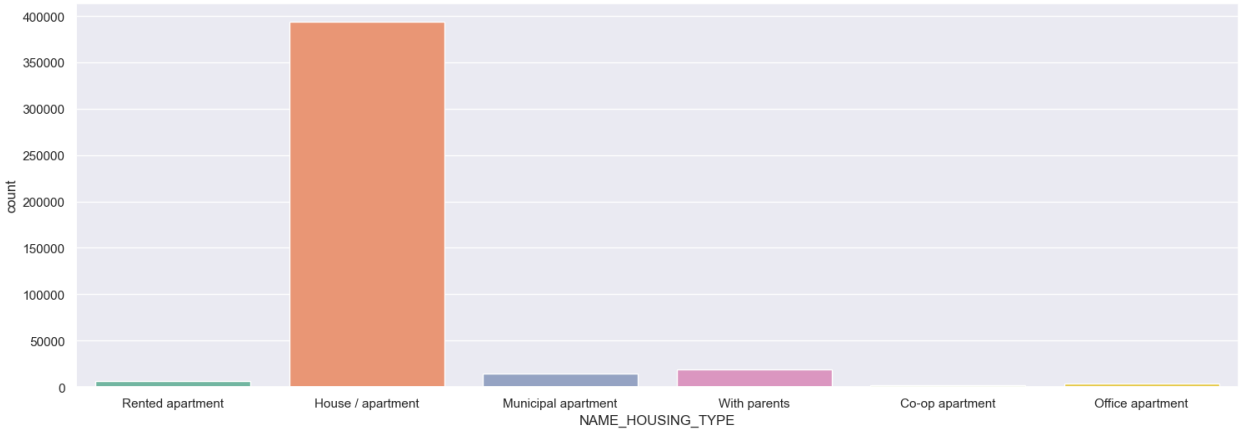
## Activity 2.1: Univariate analysis

In simple words, univariate analysis is understanding the data with single feature. Here we have displayed three different graphs of countplot. Seaborn package provides a wonderful function countplot. It is more useful for categorical features. With the help of countplot, we can find the number of unique values in the feature.

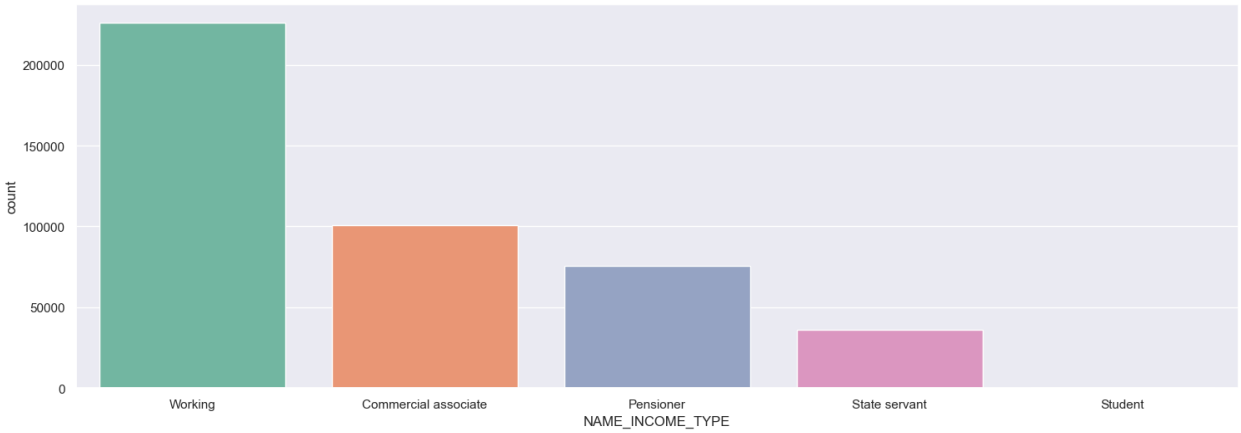
- Count plot is used on occupation type feature. With the countplot(), we are going to count the unique category. From the below graph, we found the number of labors are high when compared to other types. For the exact count, value counts() are used.




- Count plot is used on housing type feature. With the countplot(), we are going to count the unique category. From the below graph, we found the number of House/appartment are high when compared to other types. For the exact count, value counts() are used.



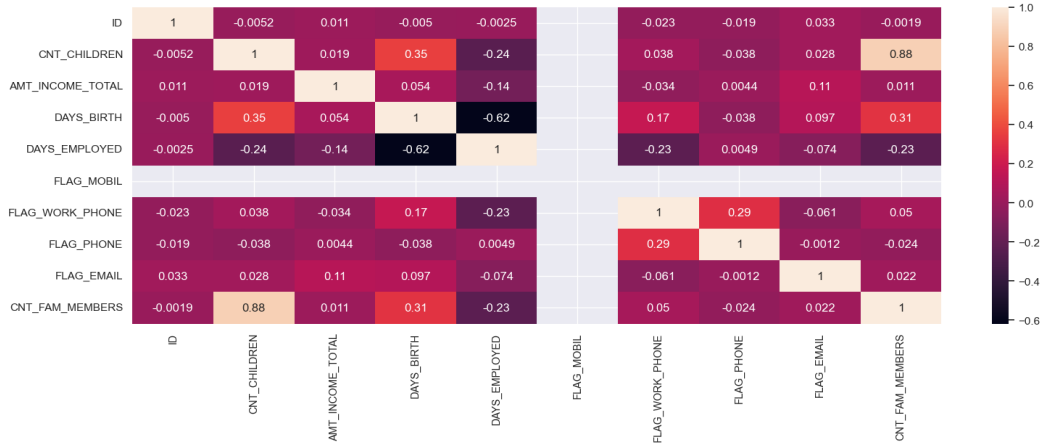

- Count plot is used on income type feature. With the countplot(), we are going to count the unique category. From the below graph, we found the number of working applicant are high when compared to other types. For the exact count, value counts() are used.




## Activity 2.2: Multivariate analysis

In simple words, multivariate analysis is to find the relation between multiple features. Here we have used heatmap from seaborn package. To visualize the correlation of the features, heatmap() function is used. As a parameter corr() function should be passed inside heatmap. And to display the correlation percentage annot() function is used.

· From the below image, we came to a conclusion that there are some features which are highly correlated.
· These Highly correlated features should be dropped




# Milestone 4: Data Pre-processing

As we have understood how the data is. Let's pre-process the collected data. The download data set is not suitable for training the machine learning model as it might have so much randomness so we need to clean the dataset properly in order to fetch good results. This activity includes the following steps.

- Handling missing values
- Handling categorical data
- Handling outliers
- Scaling Techniques
- Splitting dataset into training and test set

Note: These are the general steps of pre-processing the data before using it for machine learning. Depending on the condition of your dataset, you may or may not have to go through all these steps.

- To find the data type of columns info() function is used. It gives small information about the features.

```
app.info()
✓ 0.1s                                                    Python
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 438557 entries, 0 to 438556
Data columns (total 18 columns):
 #   Column               Non-Null Count   Dtype
---  ------               --------------   -----
 0   ID                   438557 non-null  int64
 1   CODE_GENDER          438557 non-null  object
 2   FLAG_OWN_CAR         438557 non-null  object
 3   FLAG_OWN_REALTY      438557 non-null  object
 4   CNT_CHILDREN         438557 non-null  int64
 5   AMT_INCOME_TOTAL     438557 non-null  float64
 6   NAME_INCOME_TYPE     438557 non-null  object
 7   NAME_EDUCATION_TYPE  438557 non-null  object
 8   NAME_FAMILY_STATUS   438557 non-null  object
```

- Unique() method is used to find the unique values of features. A function is defined below to find the unique values of features.

```
def unique_values():
    a = app.CODE_GENDER.unique()
    print("----------CODE_GENDER----------\n")
    print(a)
    print()
    b = app.FLAG_OWN_CAR.unique()
    print("----------FLAG_OWN_CAR----------\n")
    print(b)
    print()
    c= app.FLAG_OWN_REALTY.unique()
    print("----------FLAG_OWN_REALTY----------\n")
    print(c)
    print()
    d = app.CNT_CHILDREN.unique()
    print("----------CNT_CHILDREN----------\n")
    print(d)
    print()
    e = app.NAME_INCOME_TYPE.unique()
    print("----------NAME_INCOME_TYPE----------\n")
    print(e)
    print()
    f = app.NAME_EDUCATION_TYPE.unique()
    print("----------NAME_EDUCATION_TYPE----------\n")
    print(f)
    print()
    g = app.NAME_FAMILY_STATUS.unique()
    print("----------NAME_FAMILY_STATUS----------\n")
    print(g)
    print()
    h = app.NAME_HOUSING_TYPE.unique()
    print("----------NAME_HOUSING_TYPE----------\n")
    print(h)
    print()
    i = app.OCCUPATION_TYPE.unique()
    print("----------OCCUPATION_TYPE----------\n")
    print(i)
    print()
```

```
print()
j = app.CNT_FAM_MEMBERS.unique()
print("----------CNT_FAM_MEMBERS----------\n")
print(j)
print()
return unique_values
```

## Activity 1: Dropping unwanted features

Generally, applicant ids are unique in nature. But in our dataset we found some of the ids are repeating multiple times. To handle this we have to remove the duplicate rows. Drop duplicates() function from pandas is used to remove the duplicate rows. Refer the below diagram.

```
app.drop_duplicates(subset=['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', 'DAYS_BIRTH',
                            'DAYS_EMPLOYED', 'FLAG_MOBIL', 'FLAG_WORK_PHONE', 'FLAG_PHONE',
                            'FLAG_EMAIL', 'OCCUPATION_TYPE', 'CNT_FAM_MEMBERS'], keep = 'first', inplace=True)
```

## Activity 2: Handling missing values

For checking the null values, df.isnull() function is used. To sum those null values we use sum() function to it. mean() function is used to find the impact of null values in features. From the below image we found, our dataset has no null values.

```
app.isnull().mean()
✓ 0.0s

ID                     0.000000
CODE_GENDER            0.000000
FLAG_OWN_CAR           0.000000
FLAG_OWN_REALTY        0.000000
CNT_CHILDREN           0.000000
AMT_INCOME_TOTAL       0.000000
NAME_INCOME_TYPE       0.000000
NAME_EDUCATION_TYPE    0.000000
NAME_FAMILY_STATUS     0.000000
NAME_HOUSING_TYPE      0.000000
DAYS_BIRTH             0.000000
DAYS_EMPLOYED          0.000000
FLAG_MOBIL             0.000000
FLAG_WORK_PHONE        0.000000
FLAG_PHONE             0.000000
FLAG_EMAIL             0.000000
OCCUPATION_TYPE        0.305012
CNT_FAM_MEMBERS        0.000000
dtype: float64
```

We have null values in the occupation type feature. However, we are going to remove that column in further process. So, let's skip the handling null values process.

## Activity 3: Data cleaning and merging

In this process, we are going to combine two inter-related columns. Our dataset have some negative values. Those negative values are converted into absolute values. Feature mapping is used on some categorical columns.

- A function data_cleaning() is defined. A column is created by adding number of family members with number of childrens.
- Six unwanted columns are dropped by drop() function. Refer the below image to know the columns name.
- Days birth and days employed columns have negative values. To convert the negative values to absolute values we use abs() function.
- Feature mapping are done in housing type, income type, education type and family type columns. (This feature mapping step is an optional step).

```
def data_cleansing(data):
    # Adding number of family members with number of children to get overall family members.
    data['CNT_FAM_MEMBERS'] = data['CNT_FAM_MEMBERS'] + data['CNT_CHILDREN']

    dropped_cols = ['FLAG_MOBIL', 'FLAG_WORK_PHONE', 'FLAG_PHONE',
                    'FLAG_EMAIL', 'OCCUPATION_TYPE', 'CNT_CHILDREN']
    data = data.drop(dropped_cols, axis = 1)

    # converting birth years and days employed to years.
    data['DAYS_BIRTH'] = np.abs(data['DAYS_BIRTH']/365) #Absolute
    data['DAYS_EMPLOYED'] = data['DAYS_EMPLOYED']/365

    #Cleaning up categorical values to lower the count of dummy variables.
    housing_type = {'House / apartment' : 'House / apartment',
                    'With parents' : 'With parents',
                    'Municipal apartment' : 'House / apartment',
                    'Rented apartment' : 'House / apartment',
                    'Office apartment' : 'House / apartment',
                    'Co-op apartment' : 'House / apartment'}

    income_type = {'Commercial associate' : 'Working',
                   'State servant' : 'Working',
                   'Working' : 'Working',
                   'Pensioner' : 'Pensioner',
                   'Student' : 'Student'}

    education_type = {'Secondary / secondary special':'secondary',
                      'Lower secondary' : 'secondary',
                      'Incomplete higher' : 'Higher education',
                      'Higher education' : 'Higher education',
                      'Academic degree' : 'Academic degree'}
```

```
family_status = {'Civil marriage' : 'Married',
                 'Married' : 'Married',
                 'Single / not married' : 'Single',
                 'Separated' : 'Single',
                 'Widow' : 'Single'}

data['NAME_HOUSING_TYPE'] = data['NAME_HOUSING_TYPE'].map(housing_type)
data['NAME_INCOME_TYPE'] = data['NAME_INCOME_TYPE'].map(income_type)
data['NAME_EDUCATION_TYPE'] = data['NAME_EDUCATION_TYPE'].map(education_type)
data['NAME_FAMILY_STATUS'] = data['NAME_FAMILY_STATUS'].map(family_status)

return data
```

## Activity 3.1: Understanding data

Let's move to our second dataframe(cr).

To display the first five columns head() function is used. The info() method is used to find the data types of the columns.

```
credit.head()
```

✓ 0.0s

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

```
credit.shape
```
✓ 0.0s

```
(1048575, 3)
```

```
credit.info()
```
✓ 0.0s

```
<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  object
dtypes: int64(2), object(1)
memory usage: 24.0+ MB
```

## Activity 3.2: Creating 'pivot_tb'

We are grouping the ID column and saving it as a variable 'grouped'.

We are using as an index ID and for column we are using MONTHS_BALANCE and STATUS as a value.

- Minimum MONTHS_BALANCE as a open_month
- Maximum MONTHS_BALANCE as a end_months
- And for window we are substracting end_months – open_months

```
# Grouping the ID column and saving it as a variable ‘grouped’.
# Using as an index ID and for column we are using MONTHS_BALANCE and STATUS as a value.
# Minimum MONTHS_BALANCE as a open_month
# Maximum MONTHS_BALANCE as a end_months
# And for window we are substracting (end_months) - (open_months)
# Shows the number of past dues, paid off and no loan status
grouped = credit.groupby('ID')

pivot_tb = credit.pivot(index = 'ID', columns = 'MONTHS_BALANCE', values = 'STATUS')
pivot_tb['open_month'] = grouped['MONTHS_BALANCE'].min()
pivot_tb['end_month'] = grouped['MONTHS_BALANCE'].max()
pivot_tb['window'] = pivot_tb['end_month'] - pivot_tb['open_month']
pivot_tb['window'] += 1 #because month starts at 0

#counting number of past dues, paid offs and no loans
pivot_tb['paid_off'] = pivot_tb[pivot_tb.iloc[:,0:61] == 'C'].count(axis = 1)
pivot_tb['pastdue_1-29'] = pivot_tb[pivot_tb.iloc[:,0:61] == '0'].count(axis = 1)
pivot_tb['pastdue_30-59'] = pivot_tb[pivot_tb.iloc[:,0:61] == '1'].count(axis = 1)
pivot_tb['pastdue_60-89'] = pivot_tb[pivot_tb.iloc[:,0:61] == '2'].count(axis = 1)
pivot_tb['pastdue_90-119'] = pivot_tb[pivot_tb.iloc[:,0:61] == '3'].count(axis = 1)
pivot_tb['pastdue_120-149'] = pivot_tb[pivot_tb.iloc[:,0:61] == '4'].count(axis = 1)
pivot_tb['pastdue_over_150'] = pivot_tb[pivot_tb.iloc[:,0:61] == '5'].count(axis = 1)
pivot_tb['no_loan'] = pivot_tb[pivot_tb.iloc[:,0:61] == 'X'].count(axis = 1)
#setting ID column to merge with app data
pivot_tb['ID'] = pivot_tb.index
```

Output:

- Paid_off means loan paid on time
- Pastdue_1-29 means due less than 1 month
- Pastdue_30-59 means due greater than 1 month
- Pastdue_60-89 means due greater than 2 month
- Pastdue_90-119 means due greater than 3 month
- Pastdue_120-149 means due greater than 4 month
- Pastdue_over-150 means due greater than 5 month
- X means no_loan

```
pivot_tb.head()
```
0.0s Python

| ONTHS_BALANCE | -60 | -59 | -58 | -57 | -56 | -55 | -54 | -53 | -52 | -51 | ... | window | paid_off | pastdue_1-29 | pastdue_30-59 | pastdue_60-89 | pastdue_90 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ID | | | | | | | | | | | | | | | | | |
| 5001711 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | 4 | 0 | 3 | 0 | 0 | |
| 5001712 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | 19 | 9 | 10 | 0 | 0 | |
| 5001713 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | 22 | 0 | 0 | 0 | 0 | |
| 5001714 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | 15 | 0 | 0 | 0 | 0 | |
| 5001715 | NaN | X | X | X | X | X | X | X | X | X | ... | 60 | 0 | 0 | 0 | 0 | |

## Activity 4: Feature Engineering

A ratio based method was used to create the target variable. For example, given a client with a time period of 60 months, if the client had paid off loan 40 times and was late 20 times, this would be considered a fairly good client given that there were more loans that were paid off on time compared to late payments. If a client had no loans throughout the initial approval of the credit card account, by default, this would be considered a good client as well. To identify a bad client, the number of past dues would exceed the number of loans paid off or if the client only has past dues. It may be better to incorporate a set difference between number of paid off loans and number of past dues. Meaning, there needs to be a significant gap between paid off loans and past dues. If a person has 50 past dues and 51 paid off loans, based on the ratio method, this would be considered good. However the difference is only 1 and this may not be a good sign of a good client. For simplicity sake, I will not adjust the algorithm further and keep it at ratio decisioning. Code is also not optimal, adjustment may be needed for the code to compute faster.

```
def feature_engineering_target(data):
    good_or_bad = []
    for index, row in data.iterrows():
        paid_off = row['paid_off']
        over_1 = row['pastdue_1-29']
        over_30 = row['pastdue_30-59']
        over_60 = row['pastdue_60-89']
        over_90 = row['pastdue_90-119']
        over_120 = row['pastdue_120-149'] + row['pastdue_over_150']
        no_loan = row['no_loan']

        overall_pastdues = over_1 + over_30 + over_60 + over_90 + over_120

        if overall_pastdues == 0:
            if paid_off >= no_loan or paid_off <= no_loan:
                good_or_bad.append(1)
            elif paid_off == 0 and no_loan == 1:
                good_or_bad.append(1)
        elif overall_pastdues != 0:
            if paid_off > overall_pastdues:
                good_or_bad.append(1)
            elif paid_off <= overall_pastdues:
                good_or_bad.append(0)
        elif paid_off == 0 and no_loan != 0:
            if overall_pastdues <= no_loan or overall_pastdues >= no_loan:
                good_or_bad.append(0)
        else:
            good_or_bad.append(1)

    return good_or_bad
```

Converting our credit data into binary format because at last we need to predict whether a person is eligible for credit card or not?

Merging two data frames with merge() function.

```
target = pd.DataFrame()
target['ID'] = pivot_tb.index
target['paid_off'] = pivot_tb['paid_off'].values
target['#_of_pastdues'] = pivot_tb['pastdue_1-29'].values + pivot_tb['pastdue_30-59'].values + pivot_tb['pastdue_60-89'].values + pivot_tb['p

target['no_loan'] = pivot_tb['no_loan'].values
target['target'] = feature_engineering_target(pivot_tb)
credit_app = cleansed_app.merge(target, how = 'inner', on = 'ID')
credit_app.drop('ID', axis=1, inplace=True)
```

```
credit_app
✓ 0.0s                                                                                                                    Python
```

| | CODE_GENDER | FLAG_OWN_CAR | FLAG_OWN_REALTY | AMT_INCOME_TOTAL | NAME_INCOME_TYPE | NAME_EDUCATION_TYPE | NAME_FAMILY_STATUS | N |
|---|---|---|---|---|---|---|---|---|
| 0 | M | Y | Y | 427500.0 | Working | Higher education | Married | |
| 1 | M | Y | Y | 112500.0 | Working | secondary | Married | |
| 2 | F | N | Y | 270000.0 | Working | secondary | Single | |
| 3 | F | N | Y | 283500.0 | Pensioner | Higher education | Single | |
| 4 | M | Y | Y | 270000.0 | Working | Higher education | Married | |
| ... | ... | ... | ... | ... | ... | ... | ... | |
| 9704 | F | N | N | 180000.0 | Pensioner | secondary | Married | |
| 9705 | F | N | Y | 112500.0 | Working | secondary | Married | |
| 9706 | M | Y | Y | 90000.0 | Working | secondary | Married | |
| 9707 | F | N | Y | 157500.0 | Pensioner | Higher education | Married | |
| 9708 | M | N | Y | 112500.0 | Working | secondary | Single | |

9709 rows × 15 columns

## Activity 5: Handling Categorical Values

As we can see our dataset has categorical data we must convert the categorical data to integer encoding or binary encoding.

To convert the categorical features into numerical features we use encoding techniques. There are several techniques but in our project we are using label encoding.

- Label encoder is initialized and categorical feature is passed as parameter for fit_transform() function. Label encoding uses alphabetical ordering. For the feature names refer the below diagram.

```python
from sklearn.preprocessing import LabelEncoder

cg = LabelEncoder()
oc = LabelEncoder()
own_r = LabelEncoder()
it = LabelEncoder()
et = LabelEncoder()
fs = LabelEncoder()
ht = LabelEncoder()

credit_app['CODE_GENDER'] = cg.fit_transform(credit_app['CODE_GENDER'])
credit_app['FLAG_OWN_CAR'] = oc.fit_transform(credit_app['FLAG_OWN_CAR'])
credit_app['FLAG_OWN_REALTY'] = own_r.fit_transform(credit_app['FLAG_OWN_REALTY'])
credit_app['NAME_INCOME_TYPE'] = it.fit_transform(credit_app['NAME_INCOME_TYPE'])
credit_app['NAME_EDUCATION_TYPE'] = et.fit_transform(credit_app['NAME_EDUCATION_TYPE'])
credit_app['NAME_FAMILY_STATUS'] = fs.fit_transform(credit_app['NAME_FAMILY_STATUS'])
credit_app['NAME_HOUSING_TYPE'] = ht.fit_transform(credit_app['NAME_HOUSING_TYPE'])
```

After encoding values looks like:-

```
credit_app
✓ 0.0s                                                                    Python
```

| | CODE_GENDER | FLAG_OWN_CAR | FLAG_OWN_REALTY | AMT_INCOME_TOTAL | NAME_INCOME_TYPE | NAME_EDUCATION_TYPE | NAME_FAMILY_STATUS | N |
|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 427500.0 | 2 | 1 | 0 | |
| 1 | 1 | 1 | 1 | 112500.0 | 2 | 2 | 0 | |
| 2 | 0 | 0 | 1 | 270000.0 | 2 | 2 | 1 | |
| 3 | 0 | 0 | 1 | 283500.0 | 0 | 1 | 1 | |
| 4 | 1 | 1 | 1 | 270000.0 | 2 | 1 | 0 | |
| ... | ... | ... | ... | ... | ... | ... | ... | |
| 9704 | 0 | 0 | 0 | 180000.0 | 0 | 2 | 0 | |
| 9705 | 0 | 0 | 1 | 112500.0 | 2 | 2 | 0 | |
| 9706 | 1 | 1 | 1 | 90000.0 | 2 | 2 | 0 | |
| 9707 | 0 | 0 | 1 | 157500.0 | 0 | 1 | 0 | |
| 9708 | 1 | 0 | 1 | 112500.0 | 2 | 2 | 1 | |

```
9709 rows × 15 columns
```

## Activity 6: Splitting data into train and test

Now let's split the Dataset into train and test sets. For splitting training and testing data we are using the train_test_split() function from sklearn. As parameters, we are passing x, y, train_size, random_state. X is independent variable and y is dependent variable.

Link: https://www.geeksforgeeks.org/how-to-split-a-dataset-into-train-and-test-sets-using-python/

```
x = credit_app[credit_app.drop('target', axis = 1).columns]
y = credit_app['target']
xtrain, xtest, ytrain, ytest = train_test_split(x,y, train_size=0.8, random_state=0)
```

# Milestone 5: Model Building

## Activity 1: Training the model in multiple algorithms

Now our data is cleaned and it's time to build the model. We can train our data on different algorithms. For this project we are applying four classification algorithms. The best model is saved based on its performance. To evaluate the performance confusion matrix and classification report is used.

## Activity 1.1: Logistic Regression Model

A function named logistic_reg is created and train and test data are passed as the parameters. Inside the function, LogisticRegression() algorithm is initialized and training data is passed to the model with .fit() function. Test data is predicted with .predict() function and saved in new variable. For evaluating the model, confusion matrix and classification report is done. Refer the below image.

```
def logistic_reg(xtrain, xtest, ytrain, ytest):
    lr=LogisticRegression(solver='liblinear')
    lr.fit(xtrain, ytrain)
    ypred=lr.predict(xtest)
    print('***LogisticRegression***')
    print('Confusion matrix')
    print(confusion_matrix(ytest,ypred))
    print('Classification report')
    print(classification_report(ytest,ypred))
```

## Activity 1.2: Random Forest Classifier

A function named random_forest is created and train and test data are passed as the parameters. Inside the function, RandomForestClassifier() algorithm is

initialized and training data is passed to the model with .fit() function. Test data is predicted with .predict() function and saved in new variable. For evaluating the model, confusion matrix and classification report is done. Refer the below image.

```
def random_forest(xtrain, xtest, ytrain, ytest):
    rf = RandomForestClassifier()
    rf.fit(xtrain, ytrain)
    ypred=rf.predict(xtest)
    print('***RandomForestClassifier***')
    print('Confusion matrix')
    print(confusion_matrix(ytest,ypred))
    print('Classification report')
    print(classification_report(ytest,ypred))
```

**Activity 1.3: Xgboost Model**

A function named g_boosting is created and train and test data are passed as the parameters. Inside the function, GradientBoostingClassifier() algorithm is initialized and training data is passed to the model with .fit() function. Test data is predicted with .predict() function and saved in new variable. For evaluating the model, confusion matrix and classification report is done. Refer the below image.

```
def g_boosting(xtrain, xtest, ytrain, ytest):
    gb=GradientBoostingClassifier()
    gb.fit(xtrain, ytrain)
    ypred=gb.predict(xtest)
    print('***GradientBoostingClassifier***')
    print('Confusion matrix')
    print(confusion_matrix(ytest,ypred))
    print('Classification report')
    print(classification_report(ytest,ypred))
```

**Activity 1.4: Decision Tree Model**

A function named d_tree is created and train and test data are passed as the parameters. Inside the function, DecisionTreeClassifier() algorithm is initialized and training data is passed to the model with .fit() function. Test data is predicted with .predict() function and saved in new variable. For evaluating the model, confusion matrix and classification report is done. Refer the below image.

```
def d_tree(xtrain, xtest, ytrain, ytest):
    dt=DecisionTreeClassifier()
    dt.fit(xtrain, ytrain)
    ypred=dt.predict(xtest)
    print('***DecisionTreeClassifier***')
    print('Confusion matrix')
    print(confusion_matrix(ytest,ypred))
    print('Classification report')
    print(classification_report(ytest,ypred))
```

# Milestone 6: Performance Testing

Multiple evaluation metrics means evaluating the model's performance on a test set using different performance measures. This can provide a more comprehensive understanding of the model's strengths and weaknesses. We are using evaluation metrics for classification tasks including accuracy, precision, recall, support and F1-score.

**Activity 1: Comparing The Models**

For comparing the above four models compareModel function is defined. After calling the function, the results of models are displayed below as an output. From these four model we found decision tree model performs well. Accuracy of decision

tree model is 99%. So, this model is saved and used on flask integration.

```
def compare_model(xtrain, xtest, ytrain, ytest):
    logistic_reg(xtrain, xtest, ytrain, ytest)
    print('-'*100)
    random_forest(xtrain, xtest, ytrain, ytest)
    print('-'*100)
    d_tree(xtrain, xtest, ytrain, ytest)
    print('-'*100)
    g_boosting(xtrain, xtest, ytrain, ytest)
```

```
    compare_model(xtrain, xtest, ytrain, ytest)
 ✓  1.4s

***LogisticRegression***
Confusion matrix
[[1025   12]
 [   4  901]]
Classification report
              precision    recall  f1-score   support

           0       1.00      0.99      0.99      1037
           1       0.99      1.00      0.99       905

    accuracy                           0.99      1942
   macro avg       0.99      0.99      0.99      1942
weighted avg       0.99      0.99      0.99      1942

----------------------------------------------------------------------------------------------------
***RandomForestClassifier***
Confusion matrix
[[1028    9]
 [   6  899]]
Classification report
              precision    recall  f1-score   support

           0       0.99      0.99      0.99      1037
           1       0.99      0.99      0.99       905
...
    accuracy                           1.00      1942
   macro avg       1.00      1.00      1.00      1942
weighted avg       1.00      1.00      1.00      1942
```

```
------------------------------------------------------------
***DecisionTreeClassifier***
Confusion matrix
[[1035    2]
 [   1  904]]
Classification report
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      1037
           1       1.00      1.00      1.00       905

    accuracy                           1.00      1942
   macro avg       1.00      1.00      1.00      1942
weighted avg       1.00      1.00      1.00      1942

------------------------------------------------------------
***GradientBoostingClassifier***
Confusion matrix
[[1036    1]
 [   4  901]]
Classification report
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      1037
           1       1.00      1.00      1.00       905

    accuracy                           1.00      1942
   macro avg       1.00      1.00      1.00      1942
weighted avg       1.00      1.00      1.00      1942
```

To get deep knowledge in confusion matrix and classification report refer the below links:

Link 1: https://www.analyticsvidhya.com/blog/2020/04/confusion-matrix-machine-learning/%20,

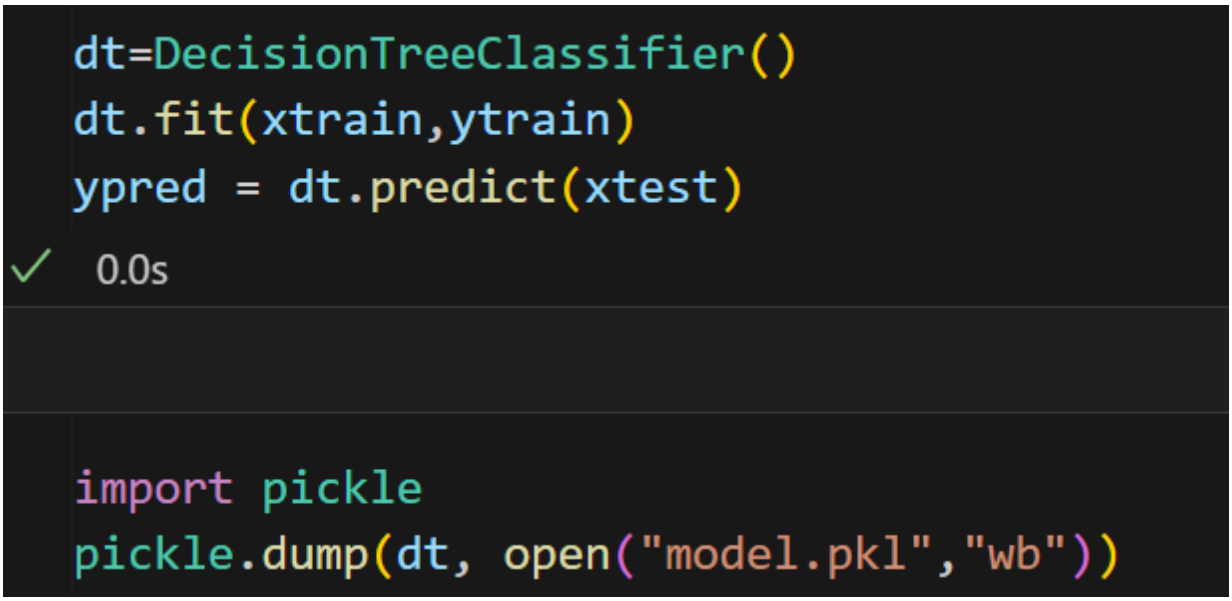
Link 2: https://medium.com/@kohlishivam5522/understanding-a-classification-report-for-your-machine-learning-model-88815e2ce397

# Milestone 7: Model Deployment

## Activity 1: Save the best model

Saving the best model after comparing its performance using different evaluation

metrics means selecting the model with the highest performance.This can be useful in avoiding the need to retrain the model every time it is needed and also to be able to use it in the future.
Decision tree model is saved by pickle.dump() function. It saves the model as .pkl file




## Activity 2: Integrate with Web Framework

In this section, we will be building a web application that is integrated to the model we built. A UI is provided for the uses where he has to enter the values for predictions. The enter values are given to the saved model and prediction is showcased on the UI.

This section has the following tasks

- Building HTML Pages
- Building serverside script

## Activity 2.1: Building Html Page

In our project we have created 3 HTML pages. They are,

1. index.html
2. index1.html

3. result.html

Save them in the templates folder. Refer to this link:

Link:https://drive.google.com/file/d/1zRabWtnCJMyOP4zVkNrR7gXkK2c5FbiV/view?usp=drive_link

**Activity 2.2: Build Python code:**

Import the libraries

```
#importing from flask_request/dependencies
from flask import Flask, request, render_template
import numpy as np
import pandas as pd
import pickle
import os
```

Load the saved model. Importing the flask module in the project is mandatory. An object of Flask class is our WSGI application. Flask constructor takes the name of the current module ( name ) as argument.

```
app=Flask(__name__) # initializing a flask app

with open(r'C:/Users/johri/Credit_Card_Approval_Prediction/Flask/model.pkl','rb') as handle:
    model = pickle.load(handle)
```

Render HTML page:

```
@app.route('/') # route to display the home page
def home():
    return render_template('index.html') #rendering the home page
@app.route('/Prediction',methods=['POST','GET'])
def prediction(): # route which will take you to the prediction page
    return render_template('index1.html')
@app.route('/home',methods=['POST','GET'])
def my_home():
    return render_template('index.html')
```

Here we will be using a declared constructor to route to the HTML page which we have created earlier.
In the above example, '/' URL is bound with the index.html function. Hence, when the home page of the web server is opened in the browser, the html page will be rendered. Whenever you enter the values from the html page the values can be retrieved using POST Method. Retrieves the value from UI:

```
@app.route('/predict',methods=["POST","GET"]) # route to show the predictions in a web UI
def predict():
    try:
        # reading the inputs given by the user
        input_feature = [float(x) for x in request.form.values()]
        features_values = np.array([input_feature])
        feature_name = ['CODE_GENDER','FLAG_OWN_CAR','FLAG_OWN_REALTY',
                        'AMT_INCOME_TOTAL','NAME_INCOME_TYPE','NAME_EDUCATION_TYPE','NAME_FAMILY_STATUS',
                    'NAME_HOUSING_TYPE','DAYS_BIRTH','DAYS_EMPLOYED','CNT_FAM_MEMBERS','paid_off','#_of_pastdues','no_loan']
        x=pd.DataFrame(features_values,columns=feature_name)

        # predictions using the loaded model file
        pred=model.predict(x)
        print(pred)
        if(pred==0):
            prediction = "Eligible"
        else:
            prediction = "Not Eligible"

        #prediction="Prediction is:"+str(pred)

        # showing the prediction results in a UI
        return render_template("result.html",prediction=prediction)
    except Exception as e:
        return f"Something went wrong: {e}"
```

Here we are routing our app to predict() function. This function retrieves all the values from the HTML page using Post request. That is stored in an array. This array is passed to the model.predict() function. This function returns the prediction. And this prediction value will be rendered to the text that we have mentioned in the submit.html page earlier.

Main Function:

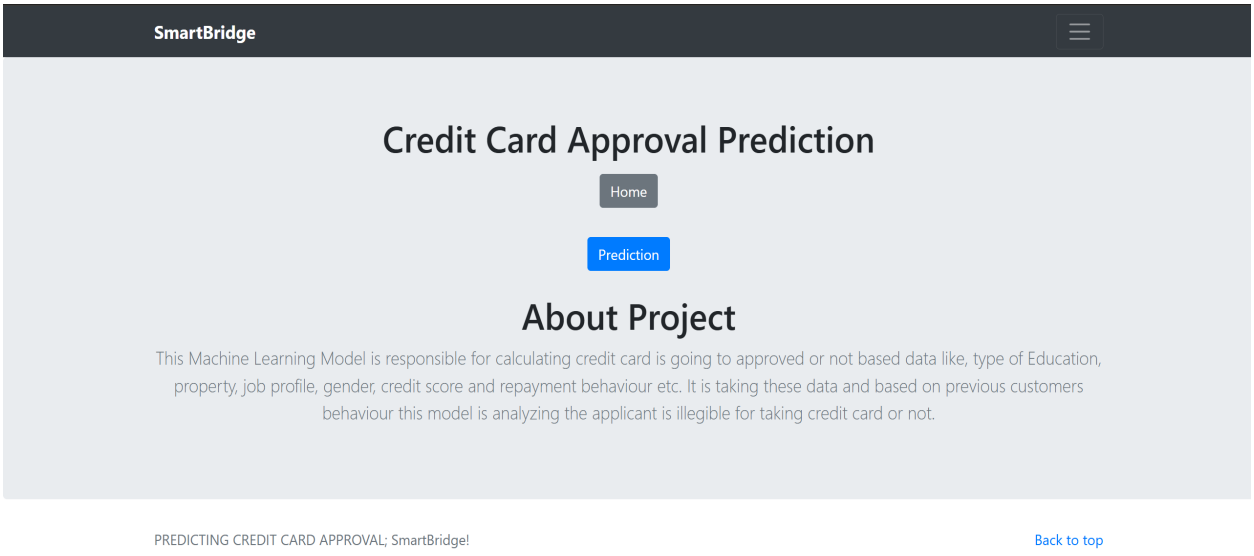
```
if __name__=="__main__":

    # Running the app
    port=int(os.environ.get('PORT', 5000))
    app.run(port=port,debug=False,use_reloader=False)
```

## Activity 2.3: Run the web application

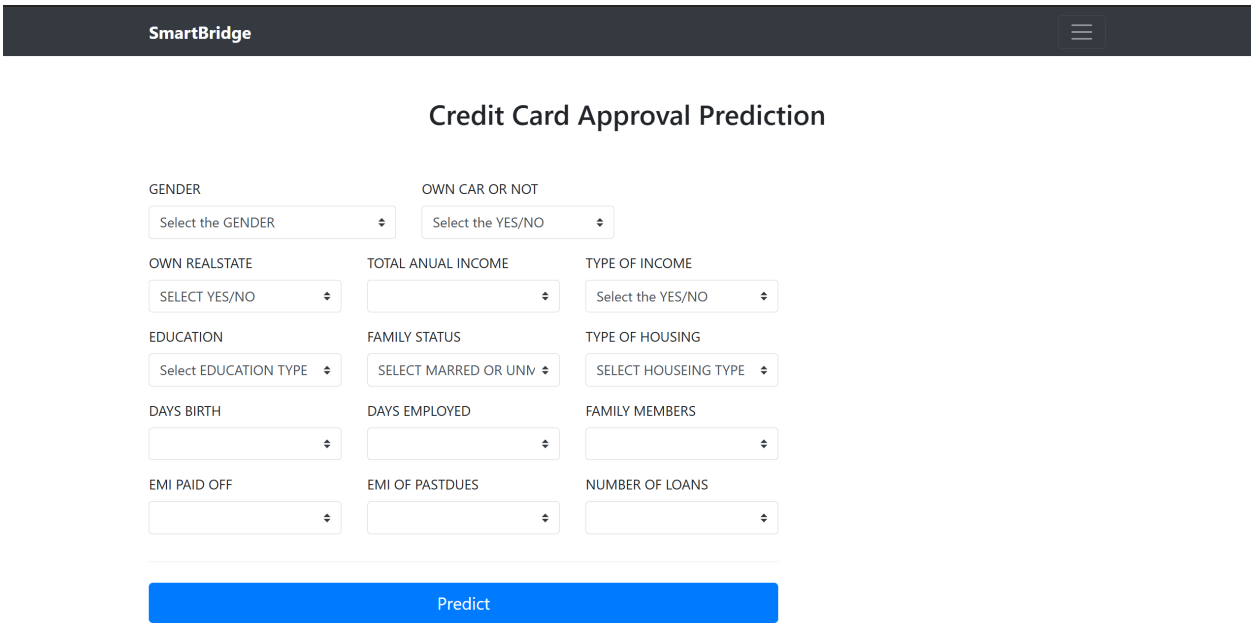
- Open anaconda prompt from the start menu
- Navigate to the folder where your python script is.
- Now type “python app.py” command
- Navigate to the localhost where you can view your web page.
- Click on the predict button from the top left corner, enter the inputs, click on the submit button, and see the result/prediction on the web.

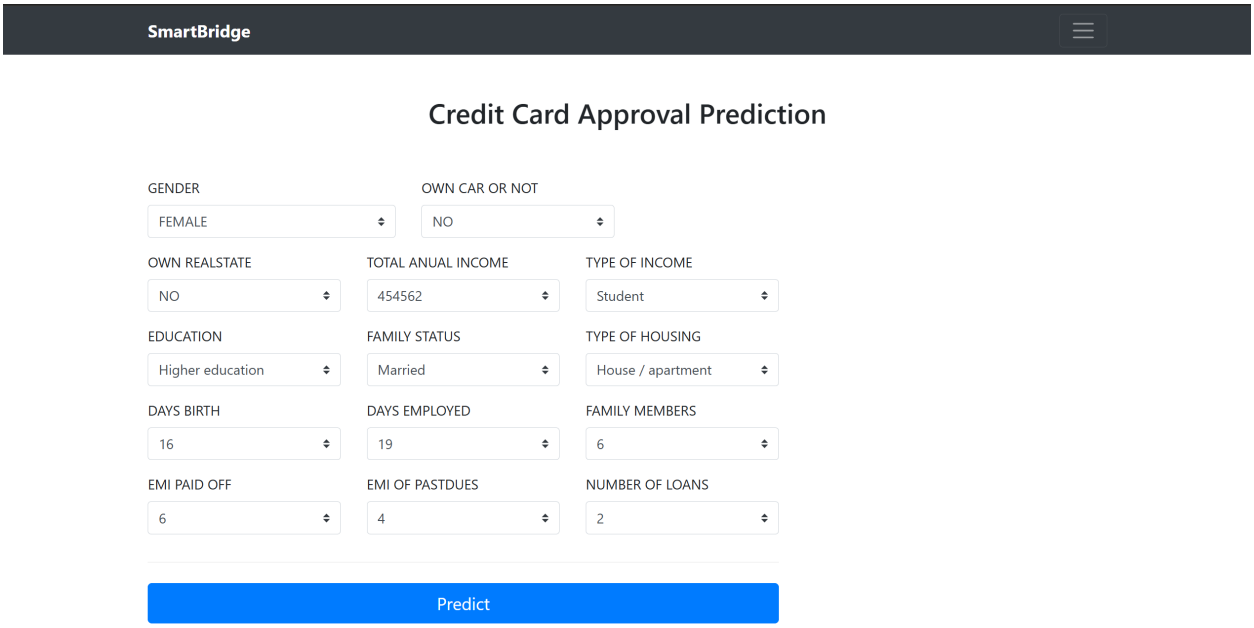
```
PS C:\Users\johri\Credit_Card_Approval_Prediction> python
 * Serving Flask app 'app'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
```

Now, go the web browser and write the localhost url (http://127.0.0.1:5000) to get the below result. Let’s look our index page.

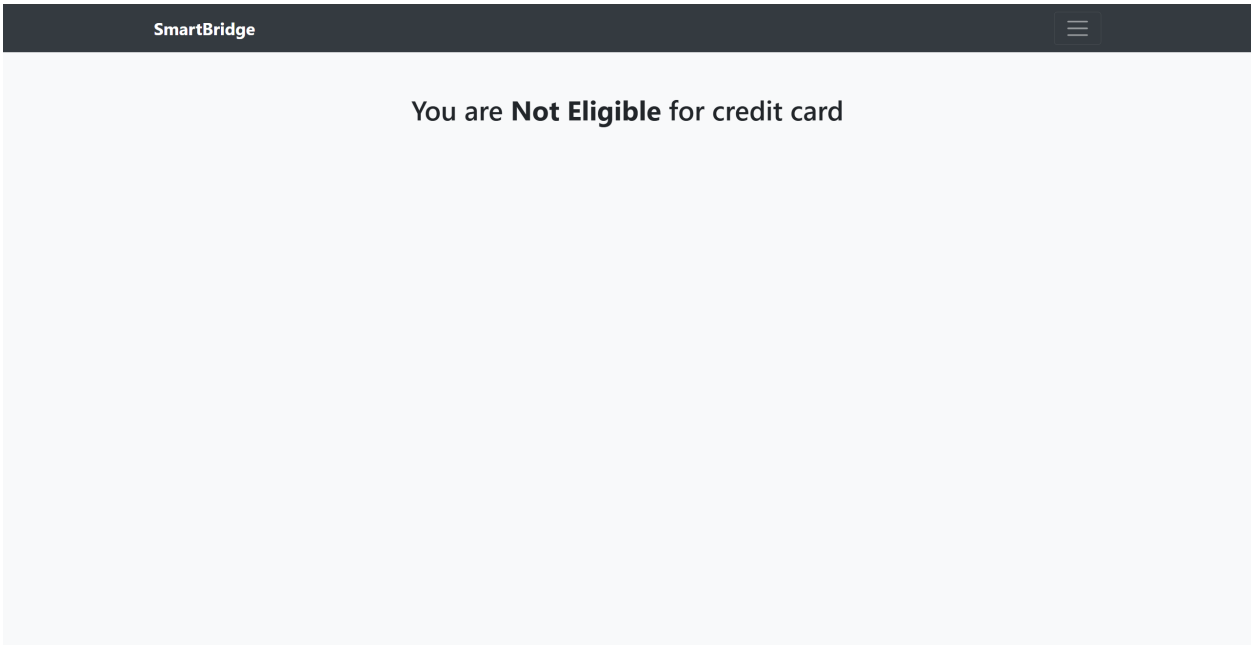



Click on predict button to give your inputs. It routes to index1.html page.




To predict your credit card eligibility, click on predict button. The output will be displayed on result.html page.




# Conclusion:

The Credit Card Approval Prediction project demonstrates the effectiveness of machine learning in automating and enhancing the decision-making process for financial institutions. By analyzing key applicant data such as income, credit score, employment status, and financial history, the system can accurately predict the likelihood of credit card approval.

This predictive capability not only helps banks minimize risks and reduce defaults but also supports better customer segmentation and personalized offerings. Additionally, the integration of fraud detection and risk assessment mechanisms further strengthens the reliability and security of the approval process.

Overall, the project offers a scalable, data-driven solution that improves operational efficiency, ensures fair decision-making, and contributes to better financial inclusion and customer experience.