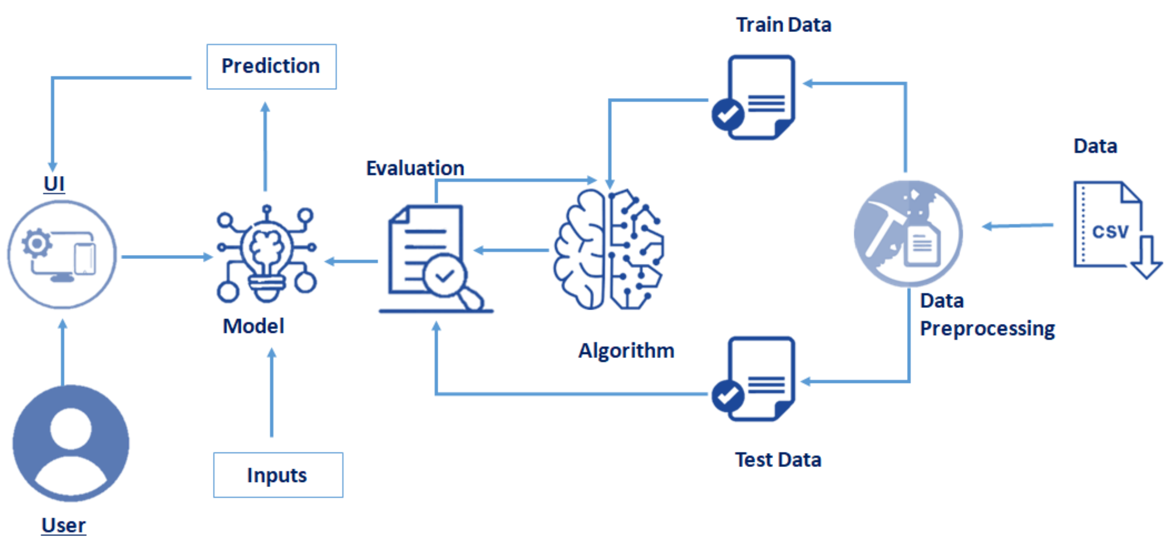
Credit Card Approval Prediction Using IBM Watson Machine Learning

Project Description:

Nowadays, banks receive a lot of applications for issuance of credit cards. Many of them are rejected for many reasons, like high-loan balances, low-income levels, or too many inquiries on an individual's credit report. Manually analyzing these applications is error-prone and a time-consuming process. Luckily, this task can be automated with the power of machine learning and pretty much every bank does so nowadays. In this project, we will build an automatic credit card approval predictor using machine learning techniques, just like the real banks do.

In this project, we will be using regression algorithms such as Decision tree, Random forest, KNN, and xgboost. We will train and test the data with these algorithms. From this the best model is selected and saved in pkl format. We will be doing flask integration and IBM deployment.

Technical Architecture:



Prerequisites:

To complete this project you should have the following software and packages .

Anaconda Navigator :

Anaconda Navigator is a free and open-source distribution of the Python and R programming languages for data science and machine learning related applications. It can be installed on Windows, Linux, and macOS.Conda is an open-source, cross-platform, package management system. Anaconda comes with so very nice tools like JupyterLab, Jupyter Notebook, QtConsole, Spyder, Glueviz, Orange, Rstudio, Visual Studio Code. For this project, we will be using Jupiter notebook and spyder
To install Anaconda navigator and to know how to use Jupyter Notebook a Spyder using Anaconda refer the link below,

https://www.youtube.com/watch?v=5mDYijMfSzs

- **Python packages:**

Open anaconda prompt as administrator

- Type **“pip install numpy”** and click enter.
- Type **“pip install pandas”** and click enter.
- Type **“pip install scikit-learn”** and click enter.
- Type **”pip install matplotlib”** and click enter.
- Type ”**pip install pickle-mixin”** and click enter.
- Type **”pip install seaborn”** and click enter.
- Type **“pip install Flask”** and click enter.

  Refer this link to know about the above libraries.

Flask: Web frame work used for building Web applications

Refer the link below to Install the necessary Packages

https://www.youtube.com/watch?v=akj3_wTploU

Prior Knowledge:

One should have knowledge on the following Concepts :

Supervised and unsupervised learning:

Refer the link below to know about the types of machine learnings
https://www.youtube.com/watch?v=kE5QZ8G_78c&t=5s

Regression Classification and Clustering :

https://www.youtube.com/watch?v=6za9_mh3uTE

Artificial Neural Networks:

https://www.youtube.com/watch?v=DKSZHN7jftI

Convolution Neural Networks :

https://www.youtube.com/watch?v=cleLMnmNMpY

Flask :

https://www.youtube.com/watch?v=lj4I_CvBnt0

Project Objective:

By the end of this project you will:

- Know fundamental concepts and techniques used for machine learning.
- Gain a broad understanding about data.
- Have knowledge on pre-processing the data/capping techniques on outlier and some visualization concepts.
- Gain some ideas on algorithm selection.

Project Flow:

- User interacts with the UI to enter the input.
- Entered input is analyzed by the model which is integrated.

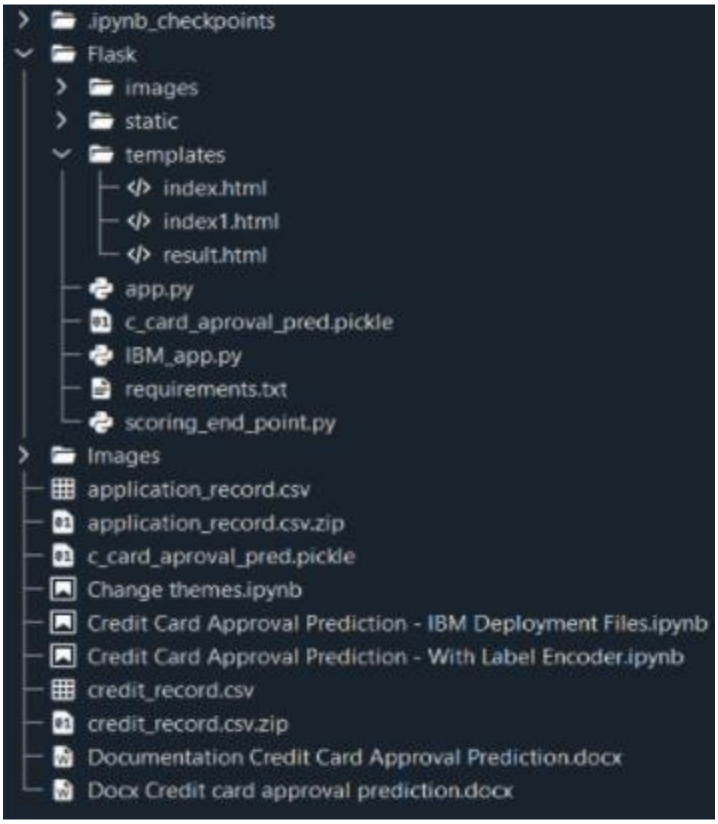
- Once model analyses the input the prediction is showcased on the UI

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

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.
- For IBM deployment IBM_app.py file is used.
- C_card_aproval_pred.pkl is our saved model. Further we will use this model for flask integration.
- Two ipynb files are local machine model building file and ibm deployment file.

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.

Download dataset

Explore and run machine learning code with Kaggle Notebooks | Using data from Credit Card Approval Prediction https://www.kaggle.com/namphuengauawatcharo/credit-card-approval-prediction/data

Visualizing And Analysing The Data

As the dataset is downloaded. Let us read and understand the data properly with the help of some visualization techniques and some analysing techniques.

Note: There is n number of techniques for understanding the data. But here we have used some of it. In an additional way, you can use multiple techniques.

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
```

Read The Dataset

```
app = pd.read_csv('application_record.csv')
credit = pd.read_csv('credit_record.csv')
```

head() method is used to return top n (5 by default) rows of a DataFrame or series.

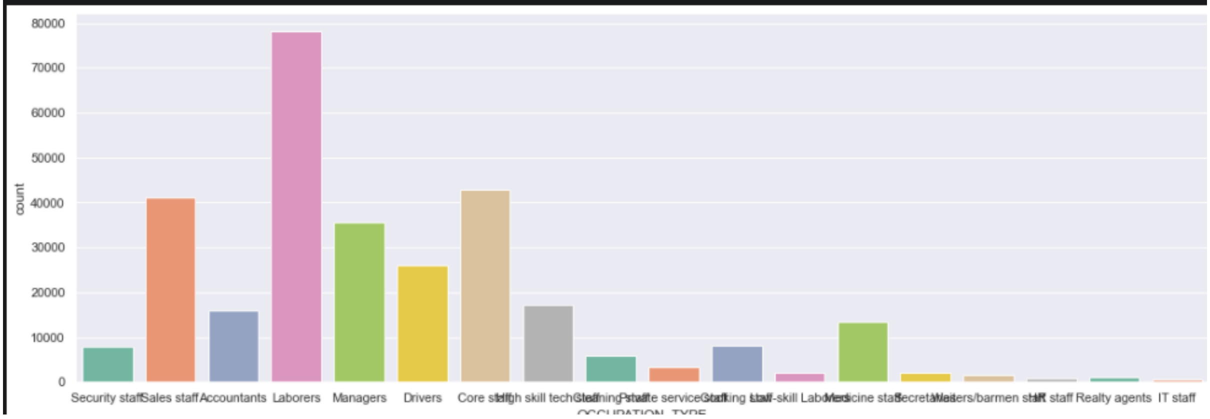
```
app.head()
```

	ID	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	NAME_INCOME_T
0	5008804	M	Y	Y	0	427500.0	Working
1	5008805	M	Y	Y	0	427500.0	Working
2	5008806	M	Y	Y	0	112500.0	Working
3	5008808	F	N	Y	0	270000.0	Commercial associat
4	5008809	F	N	Y	0	270000.0	Commercial associat

Univariate Analysis

```
print("Number of people working status :")
print(app['OCCUPATION_TYPE'].value_counts())
sns.set(rc = {'figure.figsize':(18,6)})
sns.countplot(x='OCCUPATION_TYPE', data=app, palette = 'Set2')
```

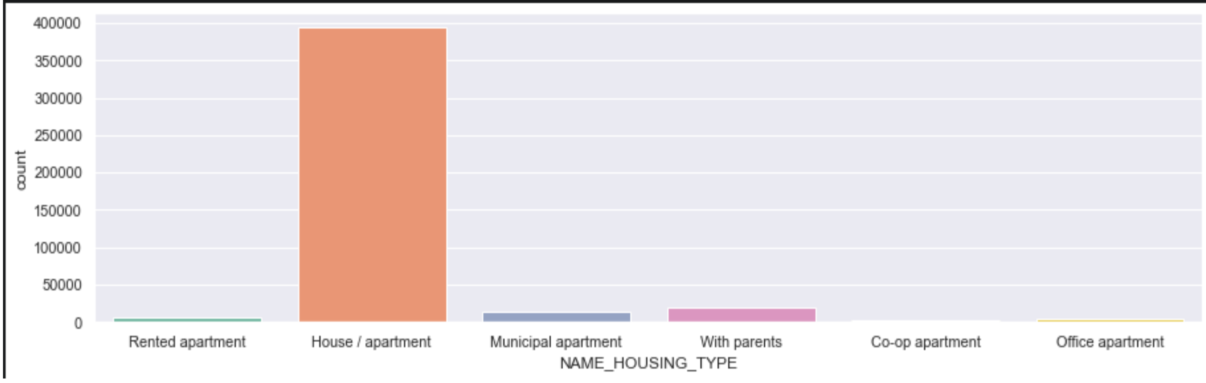
```
Number of people working status :
Laborers                 78240
Core staff               43007
Sales staff              41098
Managers                 35487
Drivers                  26090
High skill tech staff    17289
Accountants              15985
Medicine staff           13520
Cooking staff             8076
Security staff            7993
Cleaning staff            5845
Private service staff     3456
Low-skill Laborers        2140
Secretaries               2044
Waiters/barmen staff      1665
Realty agents             1041
HR staff                   774
IT staff                   604
Name: OCCUPATION_TYPE, dtype: int64

<AxesSubplot:xlabel='OCCUPATION_TYPE', ylabel='count'>
```



- g 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.

```python
print("Types of house of the peoples :")
print(app['NAME_HOUSING_TYPE'].value_counts())
sns.set(rc = {'figure.figsize':(15,4)})
sns.countplot(x='NAME_HOUSING_TYPE', data=app, palette = 'Set2')
```

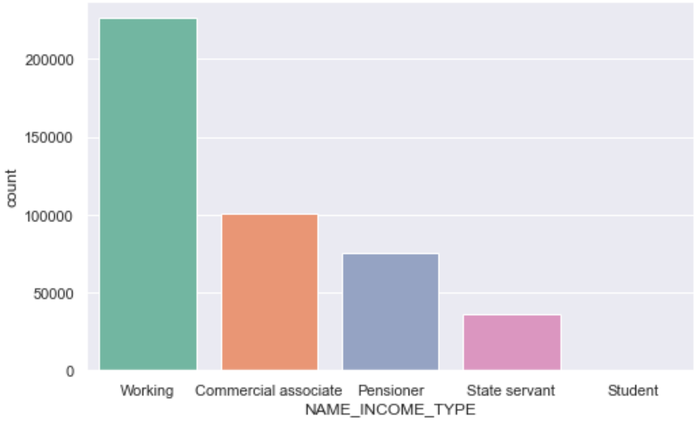
```
Types of house of the peoples :
House / apartment      393831
With parents            19077
Municipal apartment     14214
Rented apartment         5974
Office apartment         3922
Co-op apartment          1539
Name: NAME_HOUSING_TYPE, dtype: int64

<AxesSubplot:xlabel='NAME_HOUSING_TYPE', ylabel='count'>
```



- 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.

```
print("Income Types Of the Person :")
print(app['NAME_INCOME_TYPE'].value_counts())
sns.set(rc = {'figure.figsize':(8,5)})
sns.countplot(x='NAME_INCOME_TYPE', data=app, palette = 'Set2')
```

```
Income Types Of the Person :
Working                 226104
Commercial associate    100757
Pensioner                75493
State servant            36186
Student                     17
Name: NAME_INCOME_TYPE, dtype: int64

<AxesSubplot:xlabel='NAME_INCOME_TYPE', ylabel='count'>
```



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()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 90085 entries, 0 to 438553
Data columns (total 18 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   ID                   90085 non-null  int64  
 1   CODE_GENDER          90085 non-null  object 
 2   FLAG_OWN_CAR         90085 non-null  object 
 3   FLAG_OWN_REALTY      90085 non-null  object 
 4   CNT_CHILDREN         90085 non-null  int64  
 5   AMT_INCOME_TOTAL     90085 non-null  float64
 6   NAME_INCOME_TYPE     90085 non-null  object 
 7   NAME_EDUCATION_TYPE  90085 non-null  object 
 8   NAME_FAMILY_STATUS   90085 non-null  object 
 9   NAME_HOUSING_TYPE    90085 non-null  object 
 10  DAYS_BIRTH           90085 non-null  int64  
 11  DAYS_EMPLOYED        90085 non-null  int64  
 12  FLAG_MOBIL           90085 non-null  int64  
 13  FLAG_WORK_PHONE      90085 non-null  int64  
 14  FLAG_PHONE           90085 non-null  int64  
 15  FLAG_EMAIL           90085 non-null  int64  
 16  OCCUPATION_TYPE      62608 non-null  object 
 17  CNT_FAM_MEMBERS      90085 non-null  float64
dtypes: float64(2), int64(8), object(8)
memory usage: 13.1+ MB
```

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

```python
def unique_values():
    a = app.CODE_GENDER.unique()
    print("-------------CODE_GENDER--------------------------------------------")
    print(a)
    print()
    b = app.FLAG_OWN_CAR.unique()
    print("---------------------FLAG_OWN_CAR------------------------------------")
    print(b)
    print()
    c = app.FLAG_OWN_REALTY.unique()
    print('---------------------FLAG_OWN_REALTY--------------------------------------------')
    print(c)
    print()
    d = app.CNT_CHILDREN.unique()
    print('------------------------CNT_CHILDREN----------------------------------------')
    print(d)
    print()
    e = app.NAME_INCOME_TYPE.unique()
    print( '-------------------------NAME_INCOME_TYPE---------------------------------------')
    print(e)
    print()
    f = app.NAME_EDUCATION_TYPE.unique()
    print('-------------------------NAME_EDUCATION_TYPE---------------------------------------')
    print(f)
    print()
    g = app.NAME_FAMILY_STATUS.unique()
    print('-------------------------NAME_FAMILY_STATUS---------------------------------------')
    print(g)
    print()
    h = app.NAME_HOUSING_TYPE.unique()
    print('--------------------------NAME_HOUSING_TYPE------------------------------------')
    print(h)
    print()
    i = app.OCCUPATION_TYPE.unique()
    print('----------------------OCCUPATION_TYPE-----------------------------------------')
    print(i)
    print()
    j = app.CNT_FAM_MEMBERS.value_counts()
    print('---------------------------CNT_FAM_MEMBERS-------------------------------------')
    print(j)
    print()
    return unique_values
```

Read The Dataset

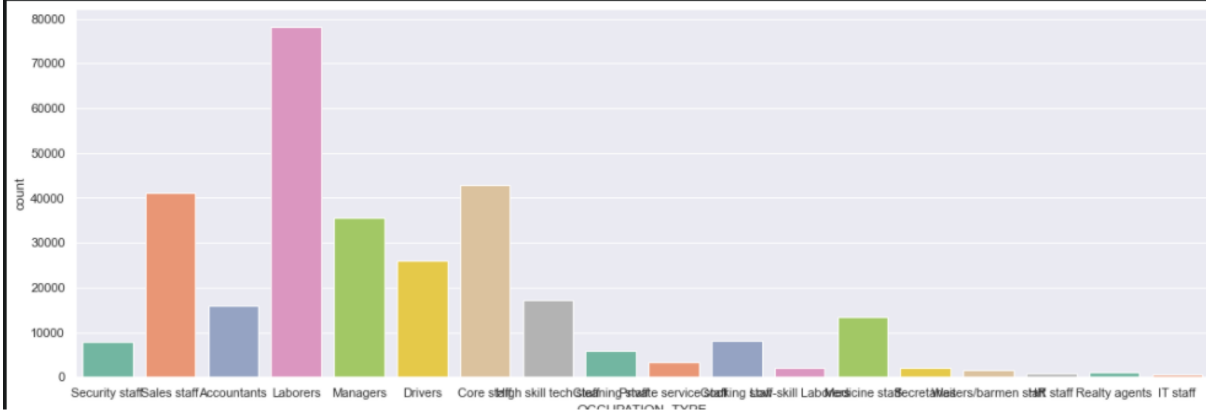
```python
app.head()
```

	ID	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	NAME_INCOME_T
0	5008804	M	Y	Y	0	427500.0	Working
1	5008805	M	Y	Y	0	427500.0	Working
2	5008806	M	Y	Y	0	112500.0	Working
3	5008808	F	N	Y	0	270000.0	Commercial associate
4	5008809	F	N	Y	0	270000.0	Commercial associate

Univariate Analysis

```
print("Number of people working status :")
print(app['OCCUPATION_TYPE'].value_counts())
sns.set(rc = {'figure.figsize':(18,6)})
sns.countplot(x='OCCUPATION_TYPE', data=app, palette = 'Set2')
```

```
Number of people working status :
Laborers                 78240
Core staff               43007
Sales staff              41098
Managers                 35487
Drivers                  26090
High skill tech staff    17289
Accountants              15985
Medicine staff           13520
Cooking staff             8076
Security staff            7993
Cleaning staff            5845
Private service staff     3456
Low-skill Laborers        2140
Secretaries               2044
Waiters/barmen staff      1665
Realty agents             1041
HR staff                   774
IT staff                   604
Name: OCCUPATION_TYPE, dtype: int64

<AxesSubplot:xlabel='OCCUPATION_TYPE', ylabel='count'>
```



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()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 90085 entries, 0 to 438553
Data columns (total 18 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   ID                   90085 non-null  int64  
 1   CODE_GENDER          90085 non-null  object 
 2   FLAG_OWN_CAR         90085 non-null  object 
 3   FLAG_OWN_REALTY      90085 non-null  object 
 4   CNT_CHILDREN         90085 non-null  int64  
 5   AMT_INCOME_TOTAL     90085 non-null  float64
 6   NAME_INCOME_TYPE     90085 non-null  object 
 7   NAME_EDUCATION_TYPE  90085 non-null  object 
 8   NAME_FAMILY_STATUS   90085 non-null  object 
 9   NAME_HOUSING_TYPE    90085 non-null  object 
 10  DAYS_BIRTH           90085 non-null  int64  
 11  DAYS_EMPLOYED        90085 non-null  int64  
 12  FLAG_MOBIL           90085 non-null  int64  
 13  FLAG_WORK_PHONE      90085 non-null  int64  
 14  FLAG_PHONE           90085 non-null  int64  
 15  FLAG_EMAIL           90085 non-null  int64  
 16  OCCUPATION_TYPE      62608 non-null  object 
 17  CNT_FAM_MEMBERS      90085 non-null  float64
dtypes: float64(2), int64(8), object(8)
memory usage: 13.1+ MB
```

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

```python
def unique_values():
    a = app.CODE_GENDER.unique()
    print("--------------CODE_GENDER--------------------------------------------")
    print(a)
    print()
    b = app.FLAG_OWN_CAR.unique()
    print("----------------------FLAG_OWN_CAR------------------------------------")
    print(b)
    print()
    c = app.FLAG_OWN_REALTY.unique()
    print('----------------------FLAG_OWN_REALTY--------------------------------------------')
    print(c)
    print()
    d = app.CNT_CHILDREN.unique()
    print('-------------------------CNT_CHILDREN-----------------------------------------')
    print(d)
    print()
    e = app.NAME_INCOME_TYPE.unique()
    print( '--------------------------NAME_INCOME_TYPE----------------------------------------')
    print(e)
    print()
    f = app.NAME_EDUCATION_TYPE.unique()
    print('--------------------------NAME_EDUCATION_TYPE----------------------------------------')
    print(f)
    print()
    g = app.NAME_FAMILY_STATUS.unique()
    print('--------------------------NAME_FAMILY_STATUS----------------------------------------')
    print(g)
    print()
    h = app.NAME_HOUSING_TYPE.unique()
    print('---------------------------NAME_HOUSING_TYPE---------------------------------------')
    print(h)
    print()
    i = app.OCCUPATION_TYPE.unique()
    print('------------------------OCCUPATION_TYPE------------------------------------------')
    print(i)
    print()
    j = app.CNT_FAM_MEMBERS.value_counts()
    print('----------------------------CNT_FAM_MEMBERS--------------------------------------')
    print(j)
    print()
    return unique_values
```

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()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 90085 entries, 0 to 438553
Data columns (total 18 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   ID                   90085 non-null  int64
 1   CODE_GENDER          90085 non-null  object
 2   FLAG_OWN_CAR         90085 non-null  object
 3   FLAG_OWN_REALTY      90085 non-null  object
 4   CNT_CHILDREN         90085 non-null  int64
 5   AMT_INCOME_TOTAL     90085 non-null  float64
 6   NAME_INCOME_TYPE     90085 non-null  object
 7   NAME_EDUCATION_TYPE  90085 non-null  object
 8   NAME_FAMILY_STATUS   90085 non-null  object
 9   NAME_HOUSING_TYPE    90085 non-null  object
 10  DAYS_BIRTH           90085 non-null  int64
 11  DAYS_EMPLOYED        90085 non-null  int64
 12  FLAG_MOBIL           90085 non-null  int64
 13  FLAG_WORK_PHONE      90085 non-null  int64
 14  FLAG_PHONE           90085 non-null  int64
 15  FLAG_EMAIL           90085 non-null  int64
 16  OCCUPATION_TYPE      62608 non-null  object
 17  CNT_FAM_MEMBERS      90085 non-null  float64
dtypes: float64(2), int64(8), object(8)
memory usage: 13.1+ MB
```

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

```python
def unique_values():
    a = app.CODE_GENDER.unique()
    print("--------------CODE_GENDER--------------------------------------------")
    print(a)
    print()
    b = app.FLAG_OWN_CAR.unique()
    print("----------------------FLAG_OWN_CAR------------------------------------")
    print(b)
    print()
    c = app.FLAG_OWN_REALTY.unique()
    print('----------------------FLAG_OWN_REALTY--------------------------------------------')
    print(c)
    print()
    d = app.CNT_CHILDREN.unique()
    print('-------------------------CNT_CHILDREN-----------------------------------------')
    print(d)
    print()
    e = app.NAME_INCOME_TYPE.unique()
    print( '--------------------------NAME_INCOME_TYPE----------------------------------------')
    print(e)
    print()
    f = app.NAME_EDUCATION_TYPE.unique()
    print('--------------------------NAME_EDUCATION_TYPE----------------------------------------')
    print(f)
    print()
    g = app.NAME_FAMILY_STATUS.unique()
    print('--------------------------NAME_FAMILY_STATUS----------------------------------------')
    print(g)
    print()
    h = app.NAME_HOUSING_TYPE.unique()
    print('---------------------------NAME_HOUSING_TYPE---------------------------------------')
    print(h)
    print()
    i = app.OCCUPATION_TYPE.unique()
    print('------------------------OCCUPATION_TYPE------------------------------------------')
    print(i)
    print()
    j = app.CNT_FAM_MEMBERS.value_counts()
    print('----------------------------CNT_FAM_MEMBERS--------------------------------------')
    print(j)
    print()
    return unique_values
```

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()
```

```
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.

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).

```python
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',
                     'Higher education':'Higher education',
                     'Incomplete higher':'Higher education',
                     'Academic degree':'Academic degree'}
    family_status = {'Single / not married':'Single',
                     'Separated':'Single',
                     'Widow':'Single',
                     'Civil marriage':'Married',
                    'Married':'Married'}
    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
```

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).

```python
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',
                     'Higher education':'Higher education',
                     'Incomplete higher':'Higher education',
                     'Academic degree':'Academic degree'}
    family_status = {'Single / not married':'Single',
                     'Separated':'Single',
                     'Widow':'Single',
                     'Civil marriage':'Married',
                    'Married':'Married'}
    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
```

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()
```

	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
```

```
(1048575, 3)
```

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

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

```
# Data frame to analyze length of time since initial approval of credit card
# Shows 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 # Adding 1 since 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(10)
```

MONTHS_BALANCE	-60	-59	-58	-57	-56	-55	-54	-53	-52	-51	...	window	paid_off	pastdue_1-29	pastdue_30-59	past
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
5001717	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	22	5	17	0	0
5001718	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	39	3	24	2	0
5001719	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	43	41	2	0	0
5001720	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	36	0	29	7	0
5001723	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	31	0	8	0	0

10 rows × 73 columns

Feature Engineering

Converting the multi-classification into binary classification. For a clear understanding refer the below two images

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['pastdue_90-119'].values
+pivot_tb['pastdue_120-149'].values +pivot_tb['pastdue_over_150'].values

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
```

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

9709 rows × 15 columns

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'])
```

inverse_transform : -Transform labels back to original encoding. (Optional)

```python
print("CODE_GENDER",credit_app['CODE_GENDER'].unique())
print(cg.inverse_transform(list(credit_app['CODE_GENDER'].unique())))
print()
print("FLAG_OWN_CAR:",credit_app['FLAG_OWN_CAR'].unique())
print(oc.inverse_transform(list(credit_app['FLAG_OWN_CAR'].unique())))
print()
print("FLAG_OWN_REALTY", credit_app['FLAG_OWN_REALTY'].unique())
print(own_r.inverse_transform(list(credit_app['FLAG_OWN_REALTY'].unique())))
print()
print("NAME_INCOME_TYPE",credit_app['NAME_INCOME_TYPE'].unique())
print(it.inverse_transform(list(credit_app['NAME_INCOME_TYPE'].unique())))
print()
print("NAME_EDUCATION_TYPE",credit_app['NAME_EDUCATION_TYPE'].unique())
print(et.inverse_transform(list(credit_app['NAME_EDUCATION_TYPE'].unique())))
print()
print("NAME_FAMILY_STATUS",credit_app['NAME_FAMILY_STATUS'].unique())
print(fs.inverse_transform(list(credit_app['NAME_FAMILY_STATUS'].unique())))
print()
print("NAME_HOUSING_TYPE",credit_app['NAME_HOUSING_TYPE'].unique())
print(ht.inverse_transform(list(credit_app['NAME_HOUSING_TYPE'].unique())))
```

Output :-

```
CODE_GENDER [1 0]
['M' 'F']

FLAG_OWN_CAR: [1 0]
['Y' 'N']

FLAG_OWN_REALTY [1 0]
['Y' 'N']

NAME_INCOME_TYPE [2 0 1]
['Working' 'Pensioner' 'Student']

NAME_EDUCATION_TYPE [1 2 0]
['Higher education' 'secondary' 'Academic degree']

NAME_FAMILY_STATUS [0 1]
['Married' 'Single']

NAME_HOUSING_TYPE [0 1]
['House / apartment' 'With parents']
```

After encoding values looks like:-

```
credit_app
```

	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	AMT_INCOME_TOTAL	NAME_INCOME_TYPE	NAME_EDUCATIO
0	1	1	1	427500.0	2	1
1	1	1	1	112500.0	2	2
2	0	0	1	270000.0	2	2
3	0	0	1	283500.0	0	1
4	1	1	1	270000.0	2	1
...	...	...	...	...	...	...
9704	0	0	0	180000.0	0	2
9705	0	0	1	112500.0	2	2
9706	1	1	1	90000.0	2	2
9707	0	0	1	157500.0	0	1
9708	1	0	1	112500.0	2	2

9709 rows × 15 columns

Splitting Data Into Train And Test

Duration: 0.5 Hrs

Skill Tags:

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.

For deep understanding refer this link

```
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)
```

Model Building

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.

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))
```

Drop 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.

```python
# dropping duplicate rows
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)
```

Handling Missing Values

```python
app.isnull().mean()
```

```
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.

Data Cleaning And Merging

```python
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',
                     'Higher education':'Higher education',
                     'Incomplete higher':'Higher education',
                     'Academic degree':'Academic degree'}
    family_status = {'Single / not married':'Single',
                     'Separated':'Single',
                     'Widow':'Single',
                     'Civil marriage':'Married',
                    'Married':'Married'}
    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
```

Feature Engineering

```python
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
```

Handling Categorical Value

```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'])
```

Splitting Data Into Train And Test

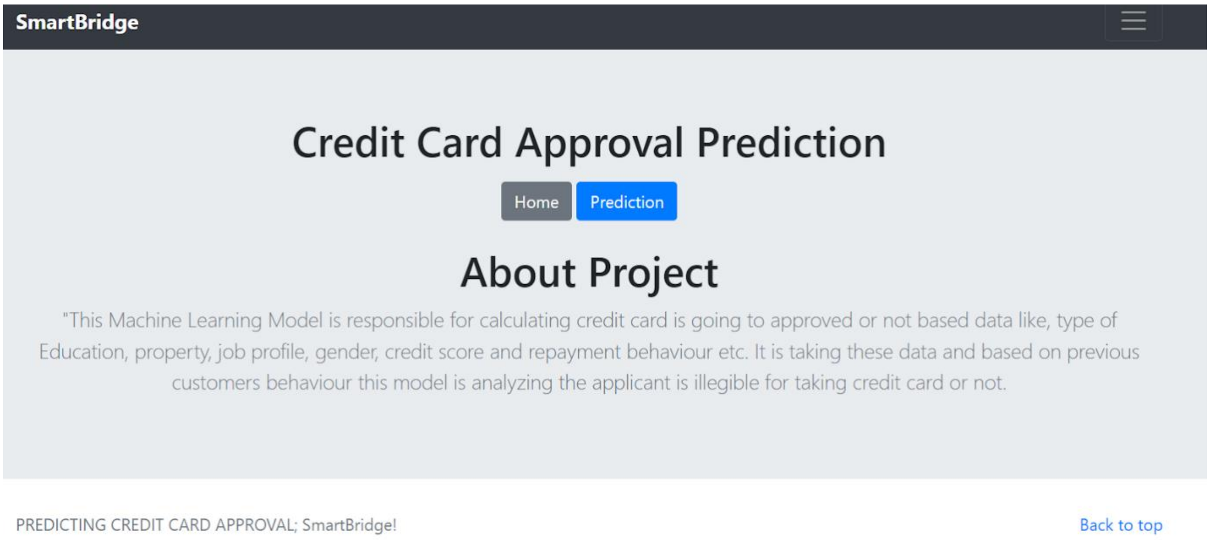

```python
from sklearn.preprocessing import LabelEncoder

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'])
```

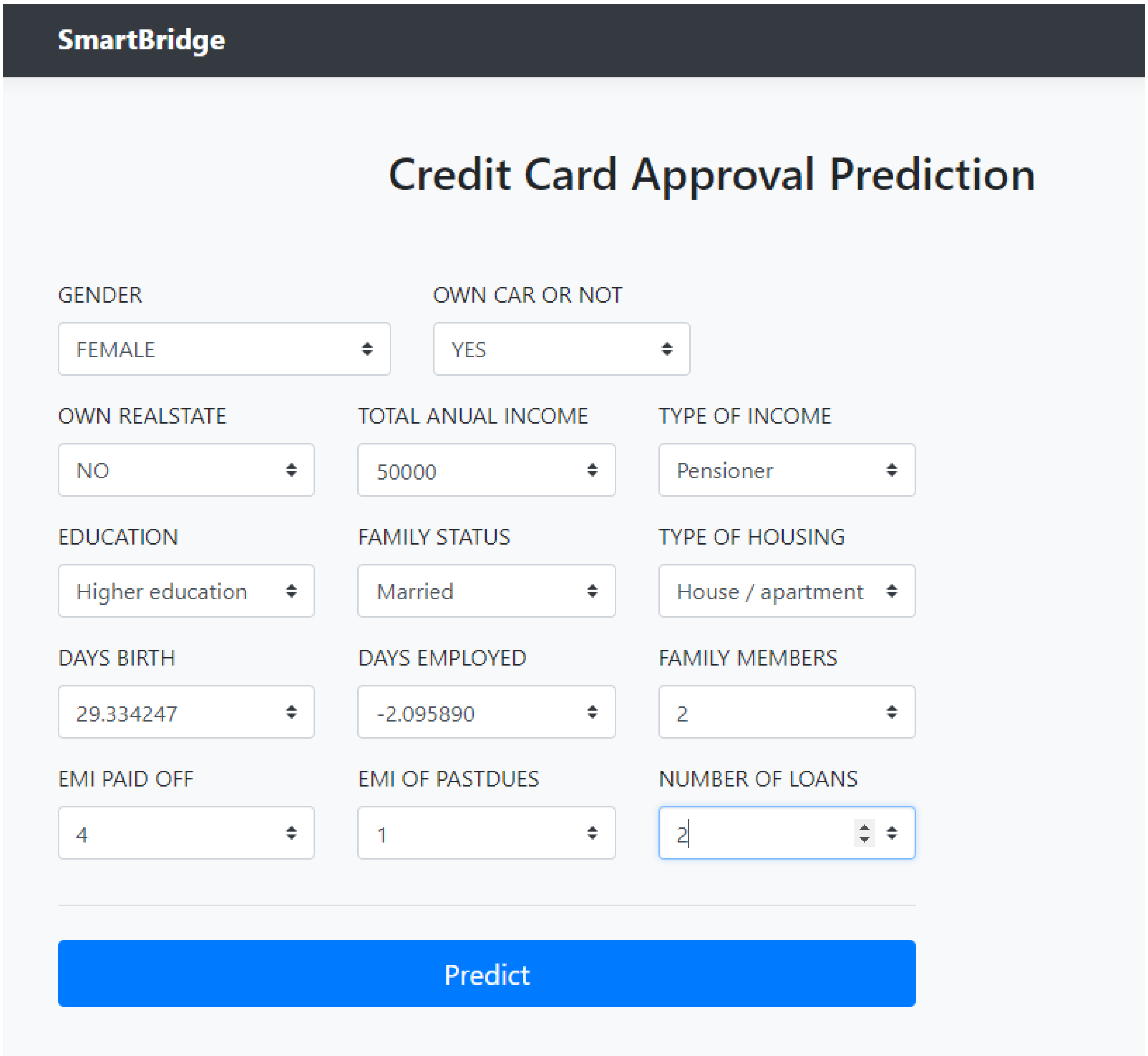

Application Building

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 Page





- Building serverside script

Train The Model On IBM

In this milestone, you will learn how to build a Machine Learning Model and deploy it on the IBM Cloud.