Sri Lanka Institute of Information Technology

Credit Card Approval Prediction

Final Report

Fundamental Data Mining – Mini Project

Submitted by:

1. IT20606756 – Senanayake P.M
2. IT20606442 – De Silva S.D.G.S
3. IT20600334 – Withanage P.W.E.L
4. IT20622114 – Kahawita M.H
5. IT20298494 – Sewwandi W.M.C

<<26-10-2022>>

Git repository link - https://github.com/SenanayakeX/FDM-Mini-Project

Table of Content

Introduction

In the banking sector, using credit scores to determine credit card approvals is a typical risk management technique. To estimate the likelihood of future defaults and credit card borrowing, it uses the personal information and data provided by credit card applicants. The bank has the authority to choose whether or not to give the applicant a credit card. The degree of risk can be accurately measured by credit scores.

For the following assignment, the group will focus on one of the bank's key operational areas and problems or limitation that are encountered.

Data Description

To analyze data we used two data sets

application_record.csv
https://www.kaggle.com/datasets/rikdifos/credit-card-approval-prediction?select=application_record.csv

credit_record.csv
https://www.kaggle.com/datasets/rikdifos/credit-card-approval-prediction?select=credit_record.csv

CODE_GENDER – Male or Female
FLAG_OWN_CAR – Owned a car
FLAG_OWN_REALTY – Owned property
CNT_CHILDREN – Number of Children
AMT_INCOME_TOTAL – Income amount
NAME_INCOME_TYPE – Income Category
NAME_EDUCATION_TYPE – Education level
NAME_FAMILY_STATUS – Marital status
NAME_HOUSING_TYPE – Way of living
DAYS_EMPLOYED – Start date
OCCUPATION_TYPE – Occupation
CNT_FAM_MEMBERS – Family size
MONTHS_BALANCE – record month: The month of the extracted data is the starting point, backwards, 0 is the current month, -1 is the previous month, and so on

STATUS – 0: 1-29 days past due 1: 30-59 days past due 2: 60-89 days overdue 3: 90-119 days overdue 4: 120-149 days overdue 5: Overdue or bad debts, write-offs for more than 150 days C: paid off that month X: No loan for the month

Data Visualization

```
!pip install matplotlib
import matplotlib.pyplot as plt

plt.figure(figsize=(10,10))

cols_to_plot = ["CNT_CHILDREN","AMT_INCOME_TOTAL","DAYS_BIRTH","DAYS_EMPLOYED"]
app_df[cols_to_plot].hist(edgecolor='black', linewidth=1.2)
fig=plt.gcf()
fig.set_size_inches(12,6)
```

1] ✓ 1.7s Python

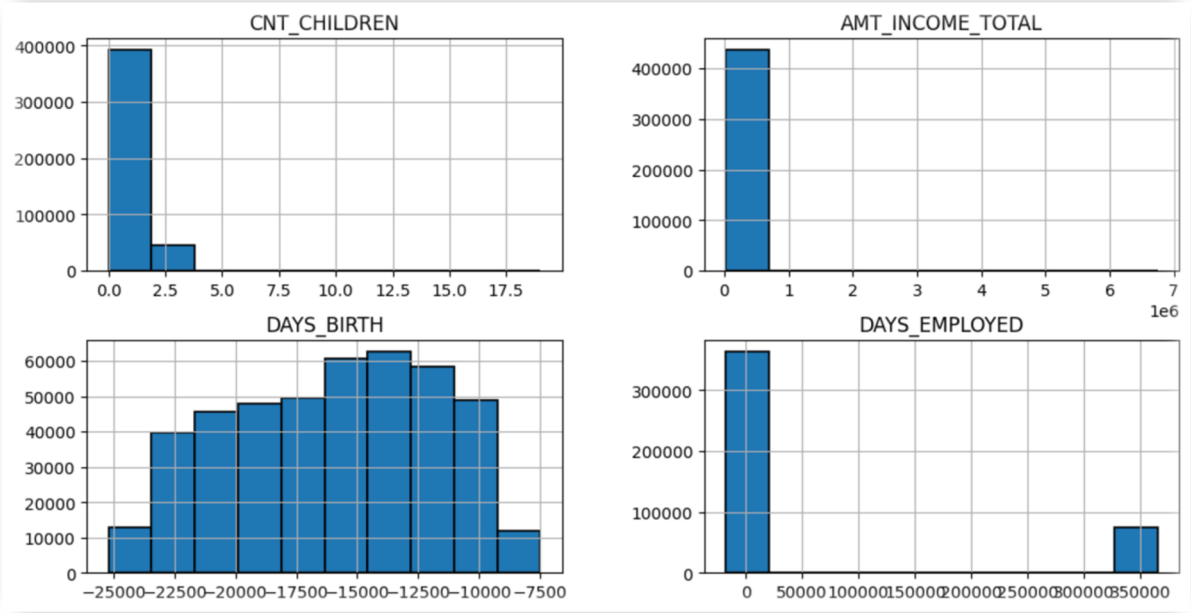


Fig 1. 1: Data Visualization

```
!pip install seaborn
import seaborn as sns
fig, axes = plt.subplots(1,2)

g1=sns.countplot(y=app_df.NAME_INCOME_TYPE,linewidth=1.2, ax=axes[0])
g1.set_title("Customer Distribution by Income Type")
g1.set_xlabel("Count")

g2=sns.countplot(y=app_df.NAME_FAMILY_STATUS,linewidth=1.2, ax=axes[1])
g2.set_title("Customer Distribution by Family Status")
g2.set_xlabel("Count")

fig.set_size_inches(14,5)

plt.tight_layout()


plt.show()
```

✓ 1.2s Python

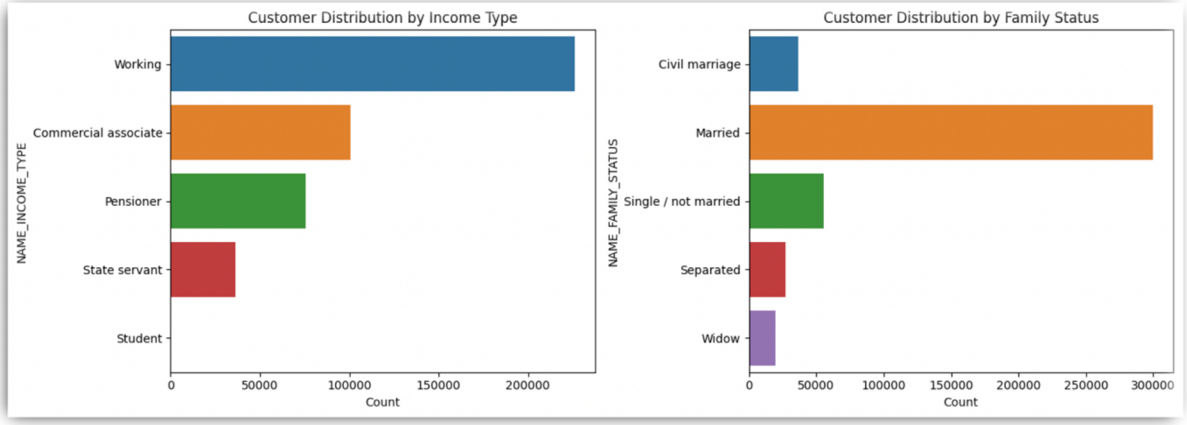


Fig 1. 1. 1: Data Visualization of income types and family status

```
fig, axes = plt.subplots(1,2)

g1= sns.countplot(y=app_df.NAME_HOUSING_TYPE,linewidth=1.2, ax=axes[0])
g1.set_title("Customer Distribution by Housing Type")
g1.set_xlabel("Count")
g1.set_ylabel("Housing Type")

g2= sns.countplot(y=app_df.NAME_EDUCATION_TYPE, ax=axes[1])
g2.set_title("Customer Distribution by Education")
g2.set_xlabel("Count")
g2.set_ylabel("Education Type")

fig.set_size_inches(14,5)

plt.tight_layout()

plt.show()
```

✓ 0.5s Python

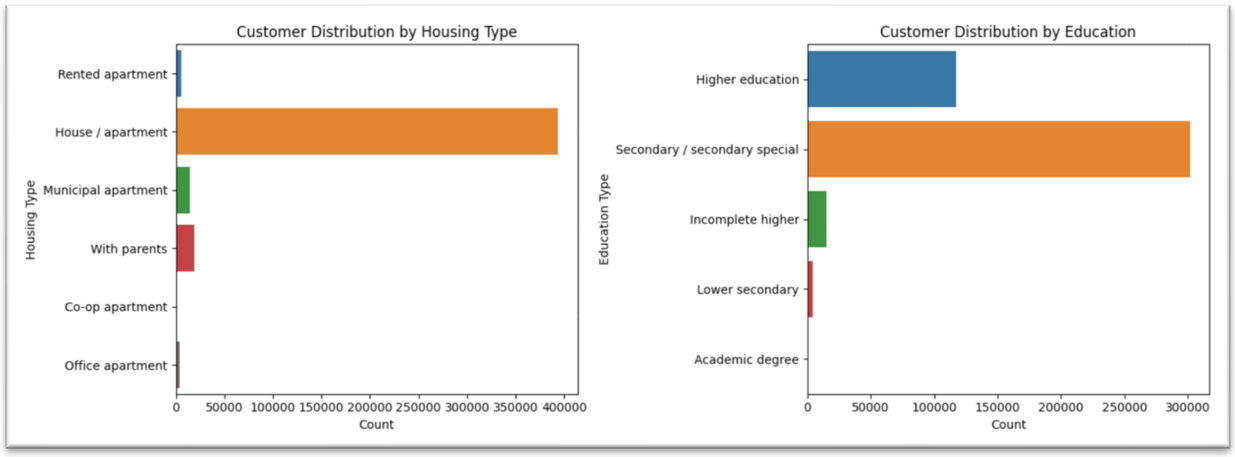


Fig 1. 1. 2: Data Visualization of housing type and education

```
fig, axes = plt.subplots(1,3)

g1= app_df['CODE_GENDER'].value_counts().plot.pie(explode=[0.1,0.1], ax=axes[0])
g1.set_title("Customer Distribution by Gender")

g2= app_df['FLAG_OWN_CAR'].value_counts().plot.pie(explode=[0.1,0.1], ax=axes[1])
g2.set_title("Car Ownership")

g3= app_df['FLAG_OWN_REALTY'].value_counts().plot.pie(explode=[0.1,0.1], ax=axes[2])
g3.set_title("Realty Ownership")

fig.set_size_inches(14,5)

plt.tight_layout()

plt.show()
```

✓ 0.2s Python

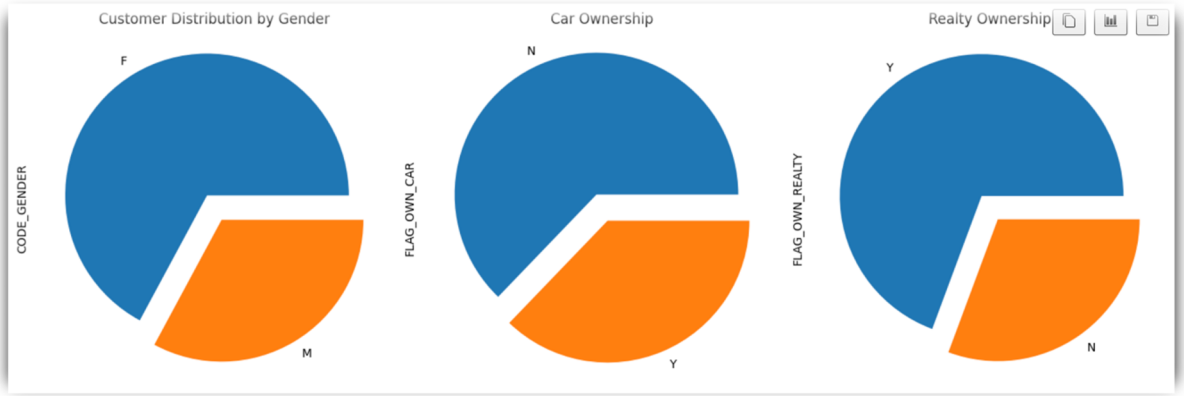


Fig 1. 1. 3 : Data Visualization of gender, car ownership and reality ownership (Pie charts)

```
f, ax = plt.subplots(figsize=(15,15))
cmap = sns.diverging_palette(230, 20, as_cmap=True)
corr = df.corr().round(4)
mask = np.triu(np.ones_like(corr, dtype=bool))
sns.heatmap(corr, annot=True, mask=mask, cmap=cmap)
```

✓ 0.6s Python

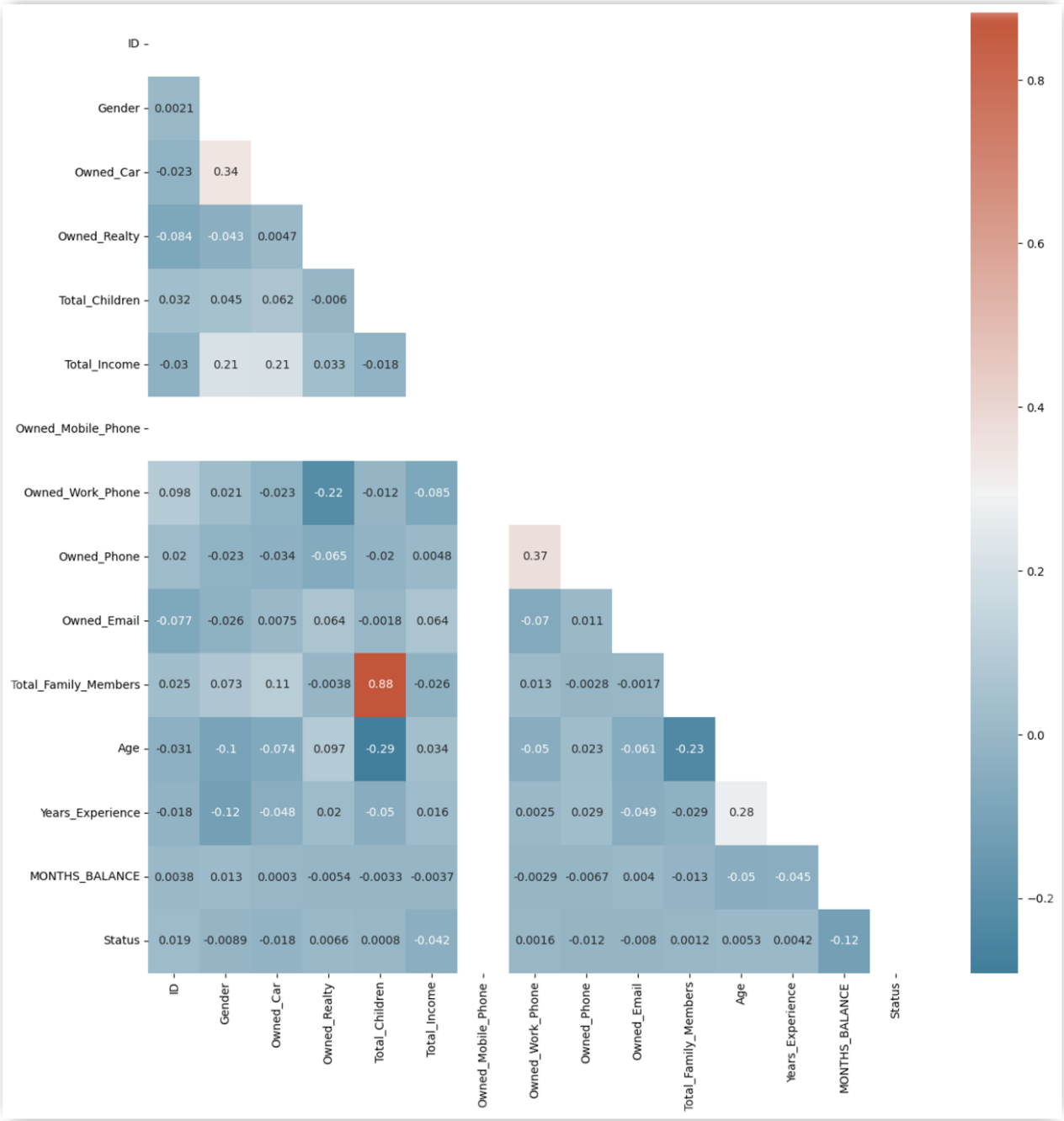


Fig 1. 2: Data Visualization

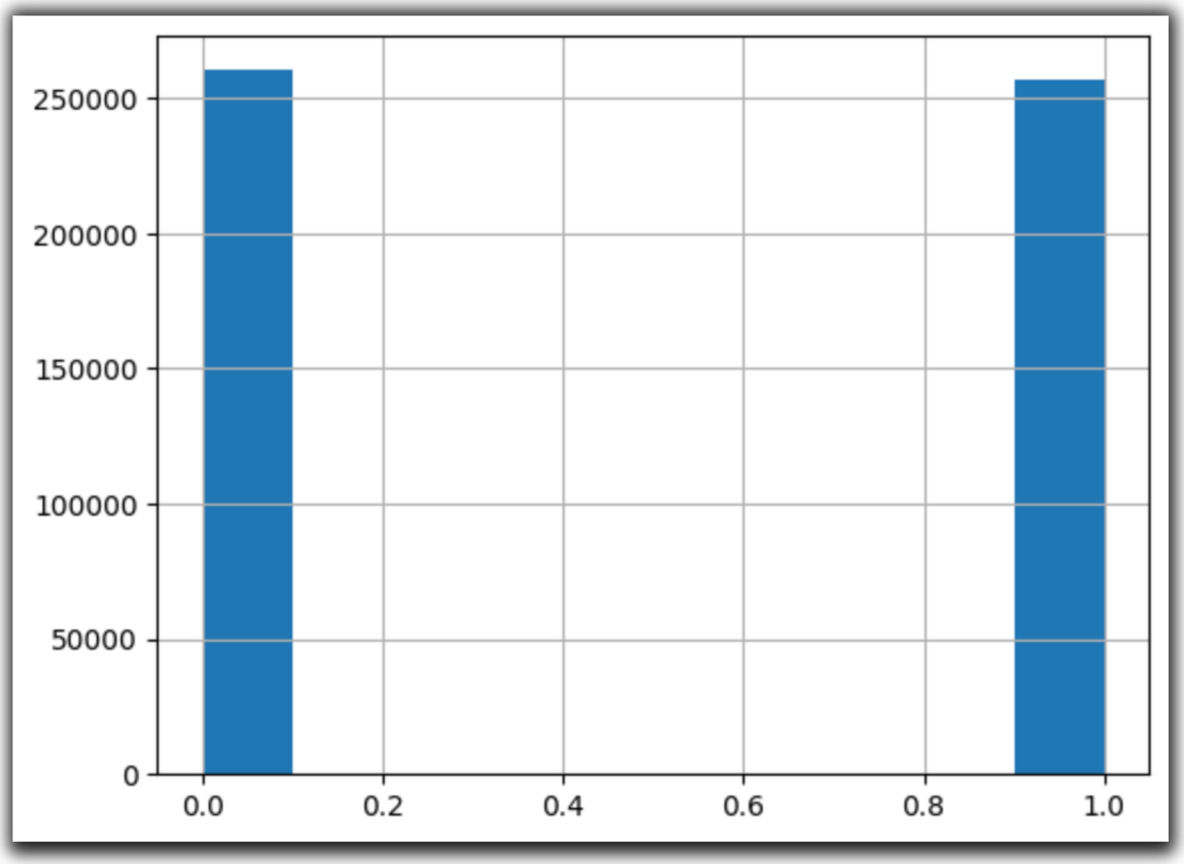


Fig 1. 2. 1: Status

Data Pre-processing

Data preprocessing is initially performed to transform the raw data in a useful and efficient format. Here mainly we used ‘pandas’ and ‘numpy’ libraries.

```
import numpy as np
import pandas as pd
```
✓ 1.4s Python

```
app_df = pd.read_csv('../Data Set/application_record.csv')
app_df
```
✓ 0.4s Python

	ID	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	NAME_INCOME_TYPE	NAME_EDUCATION_TYPE	NAME_F
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	Sin
4	5008809	F	N	Y	0	270000.0	Commercial associate	Secondary / secondary special	Sin
...	...	...	...	...	...	...	...	...	
438552	6840104	M	N	Y	0	135000.0	Pensioner	Secondary / secondary special	
438553	6840222	F	N	N	0	103500.0	Working	Secondary / secondary special	Sin
438554	6841878	F	N	N	0	54000.0	Commercial associate	Higher education	Sin
438555	6842765	F	N	Y	0	72000.0	Pensioner	Secondary / secondary special	
438556	6842885	F	N	Y	0	121500.0	Working	Secondary / secondary special	

438557 rows × 18 columns

Fig 1. 3 : Displaying data set in a data frame

Then we find out the duplicate values and remove those values.

```
app_df.drop_duplicates(subset=['ID']).count()
```
✓ 0.2s Python

```
ID                   438510
CODE_GENDER          438510
FLAG_OWN_CAR         438510
FLAG_OWN_REALTY      438510
CNT_CHILDREN         438510
AMT_INCOME_TOTAL     438510
NAME_INCOME_TYPE     438510
NAME_EDUCATION_TYPE  438510
NAME_FAMILY_STATUS   438510
NAME_HOUSING_TYPE    438510
DAYS_BIRTH           438510
DAYS_EMPLOYED        438510
FLAG_MOBIL           438510
FLAG_WORK_PHONE      438510
FLAG_PHONE           438510
FLAG_EMAIL           438510
OCCUPATION_TYPE      304318
CNT_FAM_MEMBERS      438510
dtype: int64
```

```
app_df.drop_duplicates(subset=['ID'], keep='last', inplace=True)
```
✓ 0.6s Python

Fig 1. 4: Drop duplicate

In raw dataset age represented by days. We convert it to years.

```
app_df['AGE'] = np.ceil(pd.to_timedelta(app_df['DAYS_BIRTH'], unit='D').dt.days / -365.25)
```
✓ 0.4s Python

```
app_df['AGE'].unique()
```
✓ 0.2s Python

```
array([33., 59., 53., 62., 47., 49., 30., 52., 28., 31., 35., 57., 44.,
       45., 46., 34., 56., 43., 38., 29., 58., 55., 40., 25., 21., 39.,
       41., 37., 36., 42., 60., 27., 51., 54., 63., 61., 64., 26., 23.,
       24., 65., 48., 32., 50., 66., 22., 67., 68., 69.])
```

Fig 1. 5: *Age representation*

Then we converted days of employed to years of employed.

```
app_df.loc[(app_df['DAYS_EMPLOYED'] > 0), 'DAYS_EMPLOYED'] = 0
```
✓ 0.1s Python

```
app_df['YEARS_EMPLOYED'] = np.ceil(pd.to_timedelta(app_df['DAYS_EMPLOYED'], unit='D').dt.days / -365.25)
```
✓ 0.3s Python

```
app_df['YEARS_EMPLOYED'].unique()
```
✓ 0.2s Python

```
array([13.,  4.,  9., -0.,  3.,  5.,  6., 20., 15., 14.,  8.,  7., 18.,
       30.,  2., 16., 12.,  1., 11., 24., 25., 21., 10., 28., 27., 19.,
       22., 23., 17., 29., 39., 33., 32., 37., 38., 31., 40., 26., 35.,
       34., 42., 41., 36., 44., 43., 45., 48., 46.])
```

Fig 1. 6: *years of experience*

Then we import second dataset

```
url2 = '../Data Set/credit_record.csv'
credit_df = pd.read_csv(url2, encoding='latin-1')
credit_df
```
✓ 0.1s Python

	ID	MONTHS_BALANCE	STATUS
0	5001711	0	X
1	5001711	-1	0
2	5001711	-2	0
3	5001711	-3	0
4	5001712	0	C
...	...	...	...
1048570	5150487	-25	C
1048571	5150487	-26	C
1048572	5150487	-27	C
1048573	5150487	-28	C
1048574	5150487	-29	C

1048575 rows × 3 columns

Fig 1.7 *Displaying dataset in data frame*

Define good debt and bad debt and calculating them.

```
credit_df['STATUS2'].unique()
```
✓ 0.4s Python

```
array(['X', '0', 'C', '1', '3', '2', '4', '5'], dtype=object)
```

```
credit_df = credit_df.replace({'STATUS2' :
                               {'C' : 'Good_Debt',
                                'X' : 'Bad_Debt',
                                '0' : 'Bad_Debt',
                                '1' : 'Good_Debt',
                                '2' : 'Good_Debt',
                                '3' : 'Good_Debt',
                                '4' : 'Good_Debt',
                                '5' : 'Good_Debt'}})
```
✓ 0.2s Python

```
credit_df.value_counts(subset=['ID', 'STATUS2']).unstack(fill_value=0)
```
✓ 0.9s Python

STATUS2	Bad_Debt	Good_Debt
ID		
5001711	4	0
5001712	10	9
5001713	22	0
5001714	15	0
5001715	60	0
...	...	...
5150482	12	6
5150483	18	0
5150484	12	1
5150485	2	0
5150487	0	30

45985 rows × 2 columns

Fig 1.9: good debt and bad debt

Define status using good debt and bad debt

```
result_df.loc[(result_df['Good_Debt'] > result_df['Bad_Debt']), 'Status'] = 1
```
✓ 0.1s Python

```
result_df.loc[(result_df['Good_Debt'] <= result_df['Bad_Debt']), 'Status'] = 0
```
✓ 0.1s Python

```
result_df['Status'] = result_df['Status'].astype(int)
```
✓ 0.1s Python

```
result_df.head()
```
✓ 0.2s Python

STATUS2	ID	Bad_Debt	Good_Debt	Status
0	5001711	4	0	0
1	5001712	10	9	0
2	5001713	22	0	0
3	5001714	15	0	0
4	5001715	60	0	0

Fig 2.0: status

Merge two dataset to create final dataset

```
df = app_clean_df.merge(test, how='inner', on=['ID'])
✓ 0.1s                                                    Python

df.head()
✓ 0.2s                                                    Python
```

	ID	Gender	Owned_Car	Owned_Realty	Total_Children	Total_Income	Income_Type	Education_Type	Family_Status	Housing_Type	Owned_Mobile_Phone	Ow
0	5008806	1	1	1	0	112500.0	Working	Secondary / secondary special	Married	House / apartment	1	
1	5008806	1	1	1	0	112500.0	Working	Secondary / secondary special	Married	House / apartment	1	
2	5008806	1	1	1	0	112500.0	Working	Secondary / secondary special	Married	House / apartment	1	
3	5008806	1	1	1	0	112500.0	Working	Secondary / secondary special	Married	House / apartment	1	
4	5008806	1	1	1	0	112500.0	Working	Secondary / secondary special	Married	House / apartment	1	

Fig 2.1: *final dataset in data frame*

Draw boxplots two find columns that need to ignore.

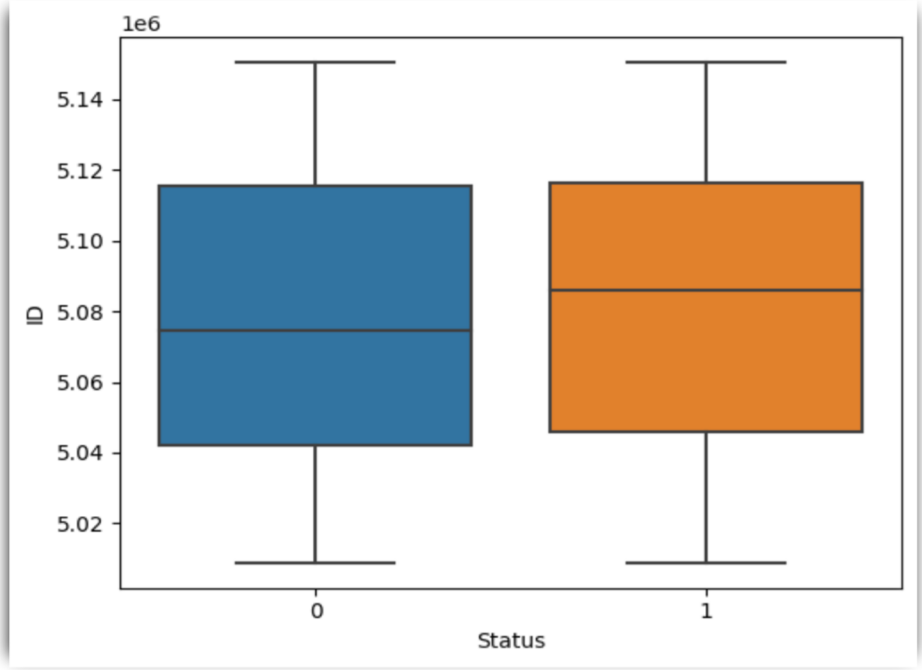


Fig 2.1.0: *Boxplot for ID*

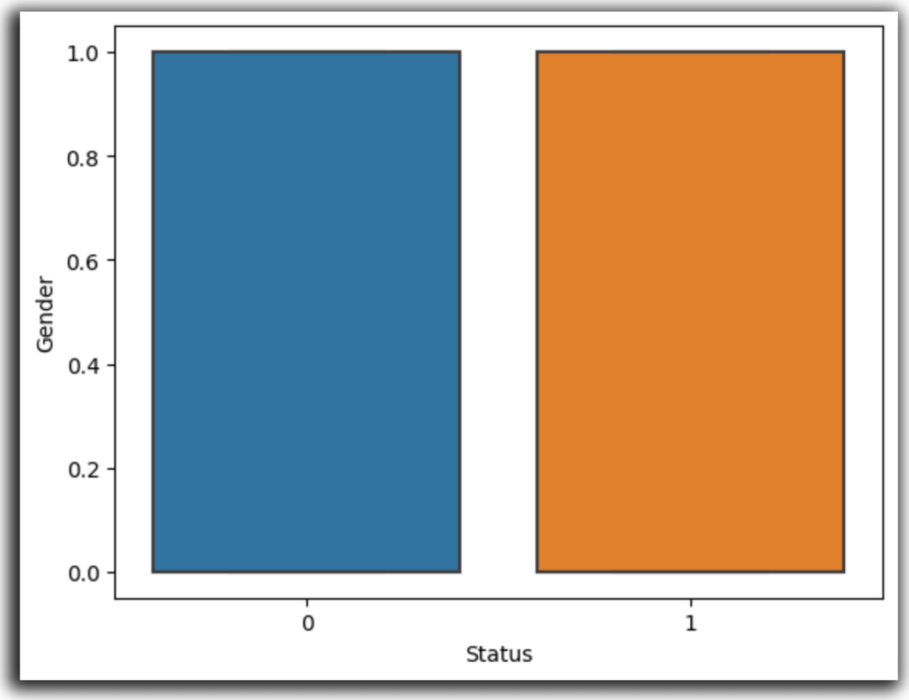


Fig 2.1.1: Boxplot for Gender

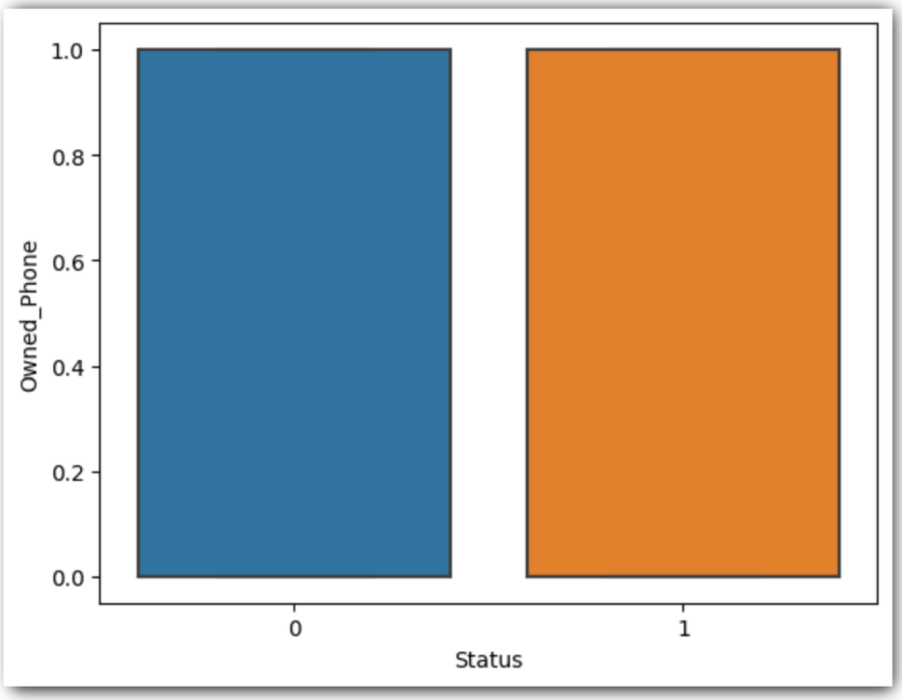


Fig 2.1.2: Boxplot for owned phone

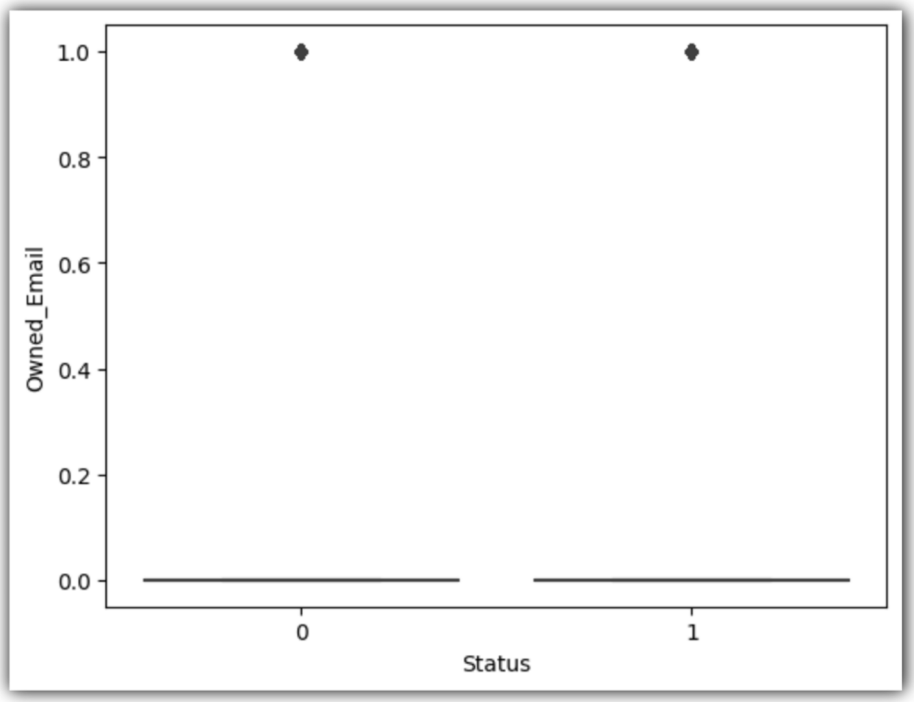


Fig 2.1.3: Boxplot for owned email

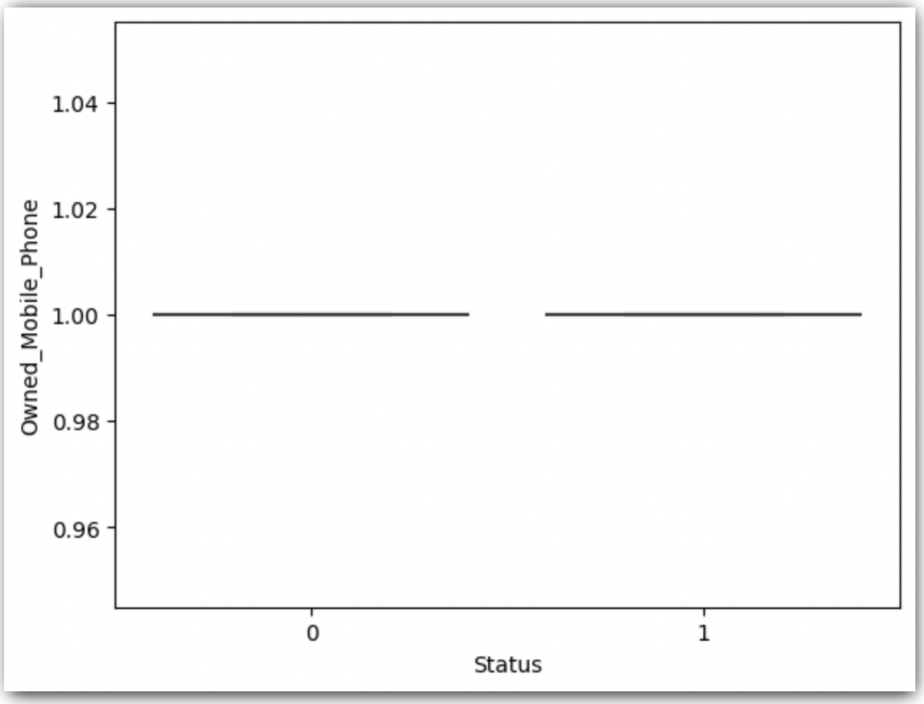


Fig 2.1.4: Boxplot for owned mobile phone

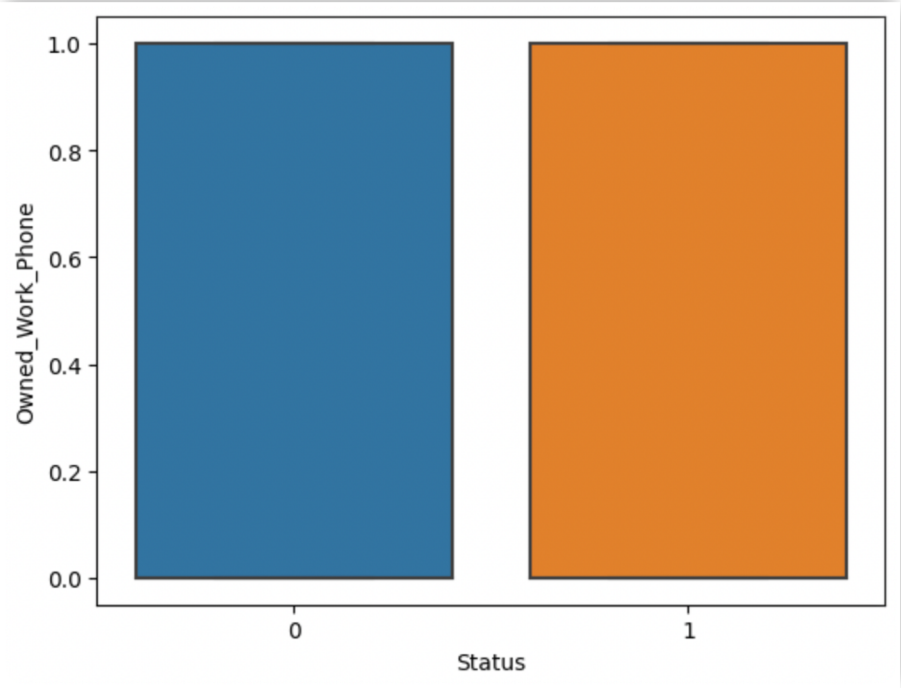


Fig 2.1.5: Boxplot for owned work phone

Removing unwanted columns

```
df = df.drop(columns=['ID', 'Gender', 'Owned_Mobile_Phone', 'Owned_Work_Phone', 'Owned_Phone', 'Owned_Email'])
✓ 0.3s                                                                                                    Python
```

Fig 2.2: dropping unwanted columns

Find out null columns

```
df.isnull().sum()
```
✓ 0.1s — Python

```
Owned_Car              0
Owned_Realty           0
Total_Children         0
Total_Income           0
Income_Type            0
Education_Type         0
Family_Status          0
Housing_Type           0
Job_Title              0
Total_Family_Members   0
Age                    0
Years_Experience       0
MONTHS_BALANCE         0
Status                 0
dtype: int64
```

Fig 2.3: checking null values

Group job titles

```
df['Job_Title'].value_counts()
```
✓ 0.3s — Python

```
Laborers                127435
Core staff               73723
Sales staff              68963
Managers                 63476
Drivers                  46483
High skill tech staff    30323
Accountants              25785
Medicine staff           24463
Cooking staff            13248
Security staff           11983
Cleaning staff           11144
Private service staff     6638
Low-skill Laborers        3617
Secretaries               3149
Waiters/barmen staff      2506
HR staff                  1588
IT staff                  1319
Realty agents             1260
Name: Job_Title, dtype: int64
```

```
def add_job(inpt):
    if inpt == 'Cooking staff' or inpt == 'Cleaning staff' or inpt == 'Waiters/barmen staff':
        return 'Dining staff'
    elif inpt == 'Security staff' or inpt == 'Private service staff' or inpt == 'Secretaries' or inpt == 'HR staff' or inpt == 'Realty agents' or inpt == 'IT staff':
        return 'Office staff'
    else:
        return inpt
```
✓ 0.1s — Python

```
df['Job_Title'] = df['Job_Title'].apply(add_job)
```
✓ 0.8s — Python

```
df['Job_Title'].value_counts()
```
✓ 0.3s — Python

```
Laborers                127435
Core staff               73723
Sales staff              68963
Managers                 63476
Drivers                  46483
High skill tech staff    30323
Dining staff             26898
```

fig 2.4: job title

Standardization

In our dataset all the features were not numeric therefore we have to follow the procedure to convert categorical data into numeric data.

- 1 represents males and 0 represent females

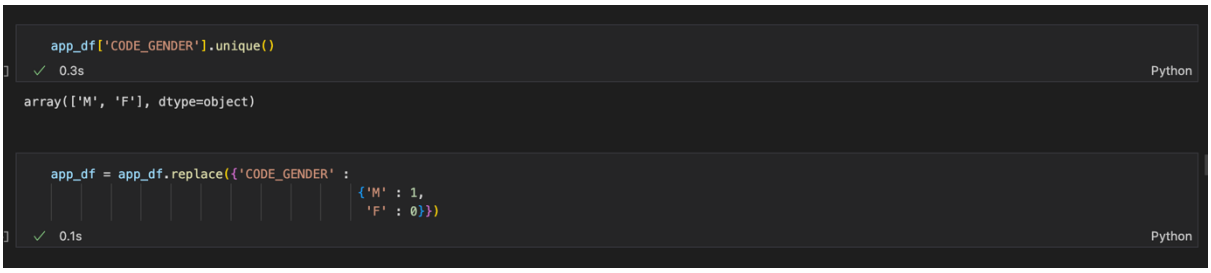


Fig 2.5: *gender representation*

- 1 represents owning a car and 0 represent not owning a car

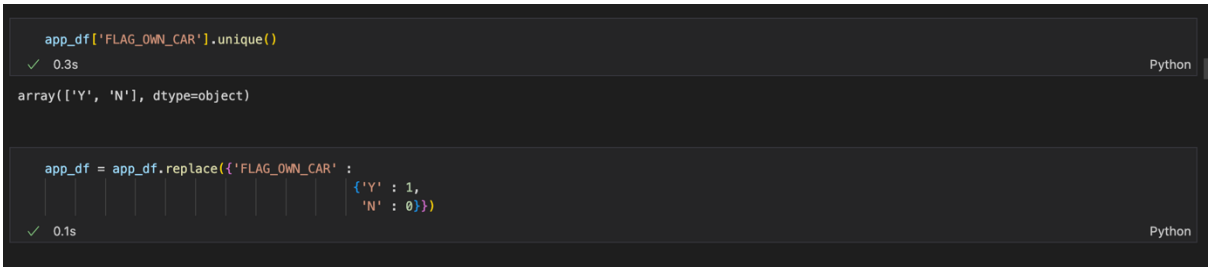


Fig 2.6: *car representation*

- 1 represents owning a reality and 0 represent not owning a reality

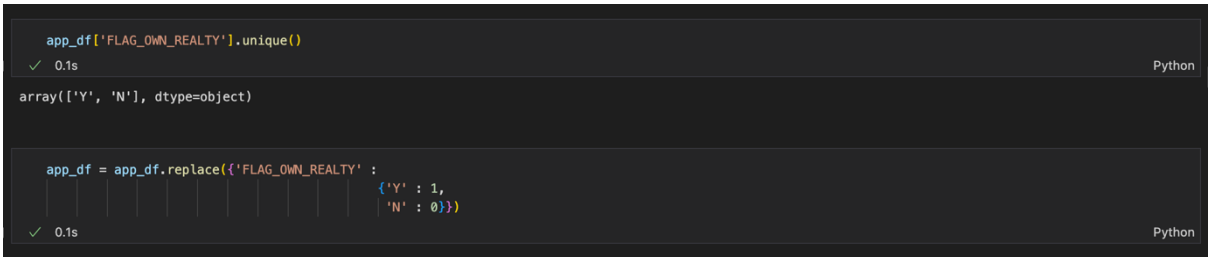


Fig 2.7: *reality representation*

Proposed Data Mining Solution

In enormous data sets, data mining seeks out hidden, true, and potentially helpful patterns. Finding unexpected or previously unknown relationships among the data is the main goal of data mining. We utilized the models listed below to make our forecasts because we needed to predict both new cases and fatalities.

Classification

Classification analysis is a data analysis task within data-mining, that identifies and assigns categories to a collection of data to allow for more accurate analysis. The classification method makes use of mathematical techniques such as decision trees, linear programming, neural network and statistics.

Naive Bayes Algorithm

It is a classification technique based on Bayes' Theorem with an assumption of independence among predictors. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature.

$$P(x_i \mid y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right)$$

Association Rule

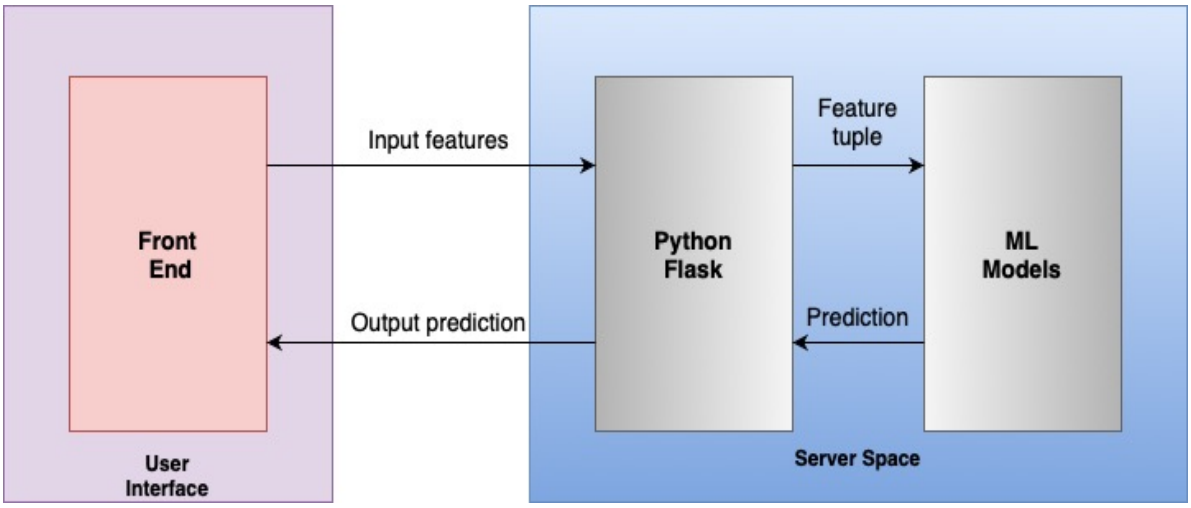
Association rule mining finds interesting associations and relationships among large sets of data items. This rule shows how frequently an itemset occurs in a transaction.

Market Based Analysis

Market based analysis is one of the key techniques used by large relations to show association between items. It allows retailers to identify relationship between the items that people buy together frequently.

Deployment Implementation

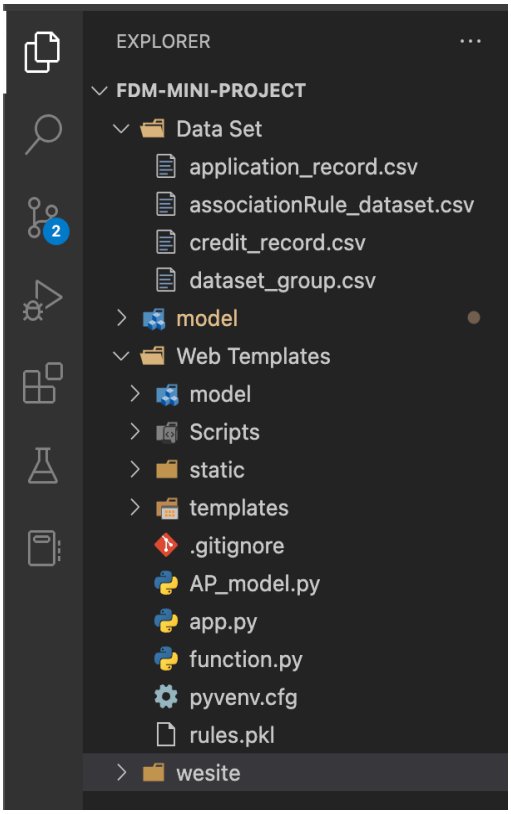
The models were developed and trained using python



The Project Structure

The Flask file structure should be in a specific folder layout otherwise errors may occur while loading.

The Flask File structure used in our project is given below.



Static Folder - This contains all your static assets like CSS files, and images.

Templates Folder - This is the default location that template HTML files must be in for Flask to render them properly. Any page that interacts with your models will be in here.

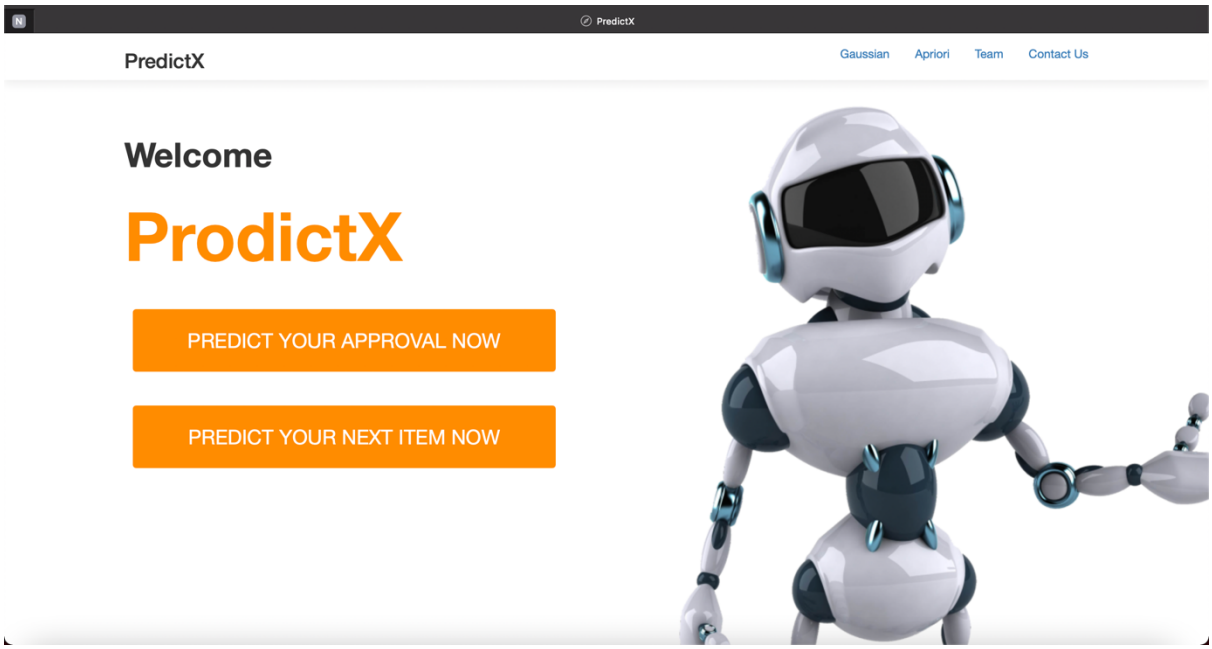
Index.html - This is the front-facing HTML file that users can interact with

App.py - Exits in root directory. This contains functions that connect model and the user inputs

User Interface

A simple user friendly interface was developed.

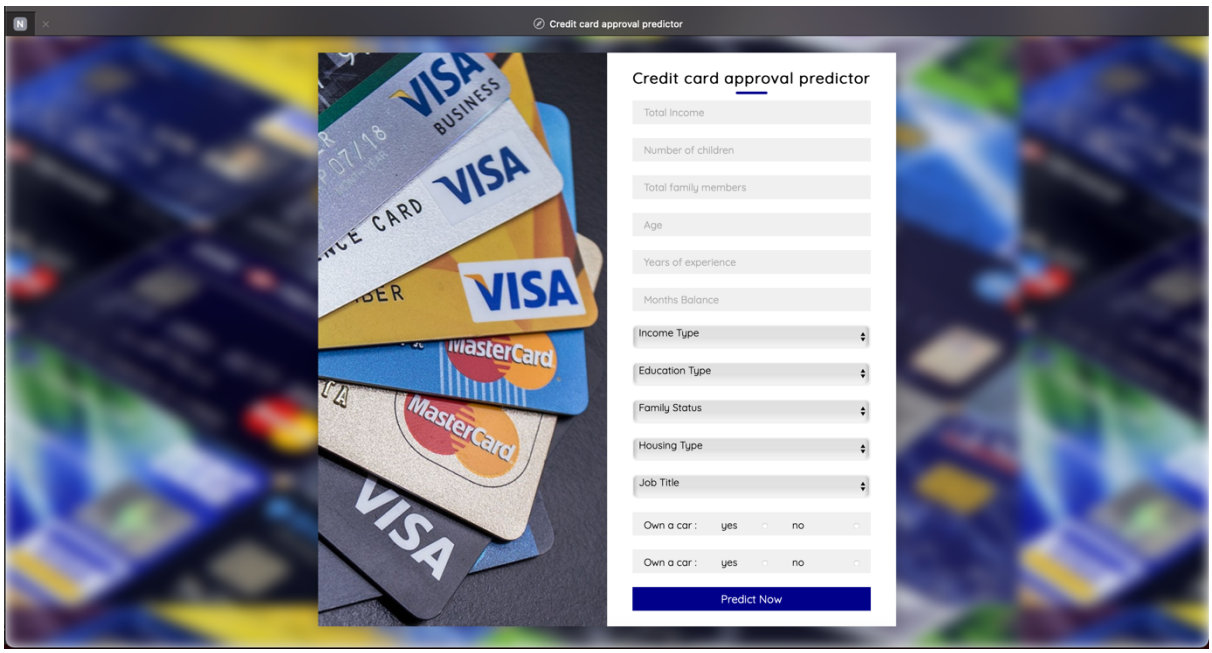
- Home page



- Credit card Approval Prediction

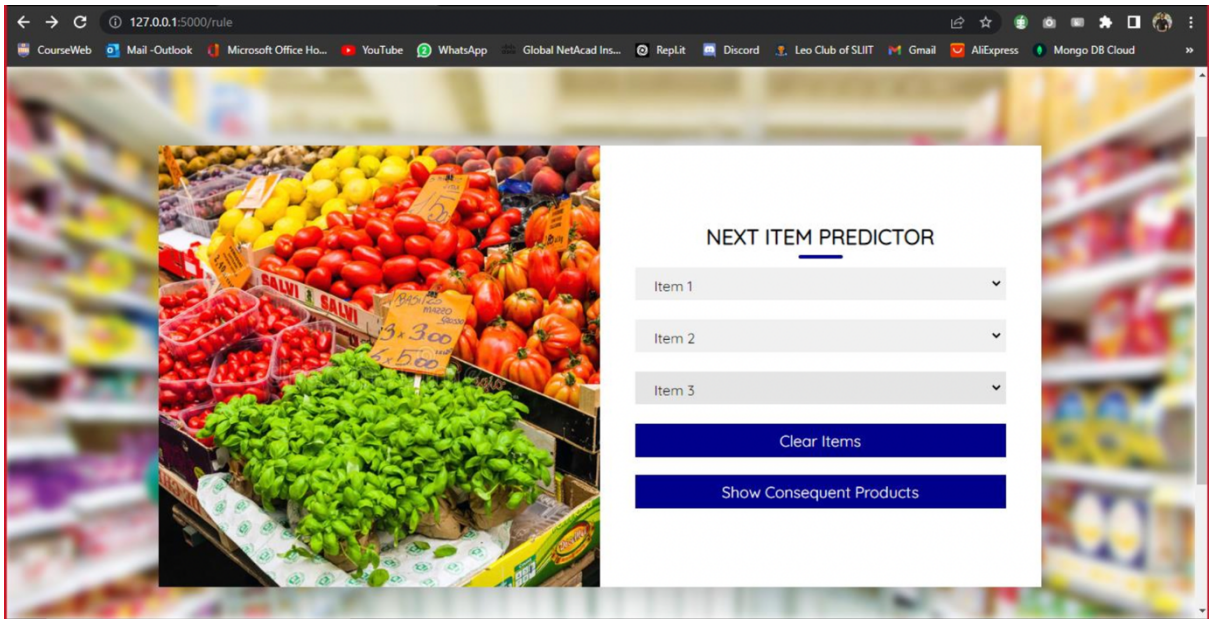
  For Credit card approval prediction user have to enter total Income, total number of children, total family members, Age, years of experience and months balance. Then

user has to select income type, education type, family status, job title, car owner ship and reality ownership.



- Product Recommendation

  To retrieve consequent products user has to select three items.



Benefits of Proposed Solution

Credit card approval prediction

- **Risk Control**

Model predicts the default behavior of the credit card applicants or holders, and offers banks efficient ways to approve credit cards and manage risk.

- **Data re-usability**

Data gathered from future transactions can be used again to improve accuracy of the model. Which will help to save future cost

Market based Analysis

- **Identifies customer behavior and pattern**

Marketers can predict with some degree of precision what products each client will likely buy next by looking at their shopping habits.

- **Arranging Display**

Display format of super market can change according to the customer's shopping behavior.

- **Help in arranging prices and campaign planning**

Marketers can set prices and promotions by paying attention to clients' buying trends.

Conclusion

- Models used: Gaussian, Apriori
- Reasons to select Models:
  In order to train the model, choose the algorithm with the best accuracy after calculating the accuracy of the most accurate output predictions.

- Our proposed system will help the bank management to solve problems they face when they approve credit card applications. It will help them to save the time they waste. And also helps Customers to improve their potential to get a credit card.

- Our proposed systems will help to marketers to have better idea about customers' shopping behaviour and it'll help to increase their sales.