SMARTBRIDGE
Let's Bridge the Gap

Smart
Internz

Credit Card Approval Prediction Using Machine Learning

Project Hand-out, Faculty Development Program

Team members :

Bhavya Katiyar (Lead)
Kanak Soni
Vidushi Chhajed
Saumil Singhal

Mentor name: Satwika

Smartinternz
www.smartinternz.com

Credit Card Approval Prediction Using Machine Learning

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.

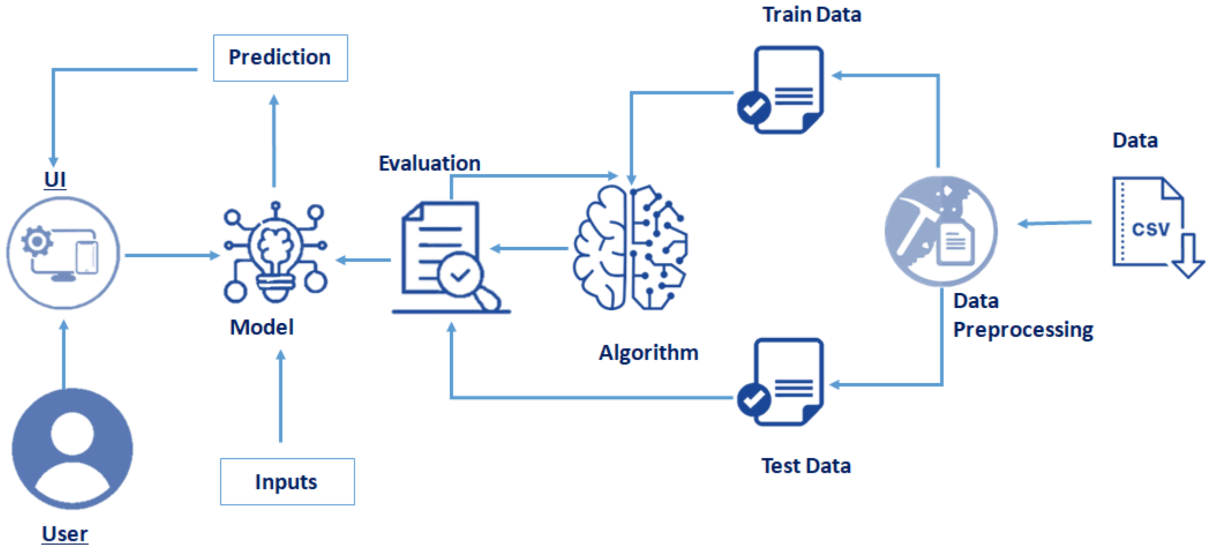
Scenario 2: Personalized Offers

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

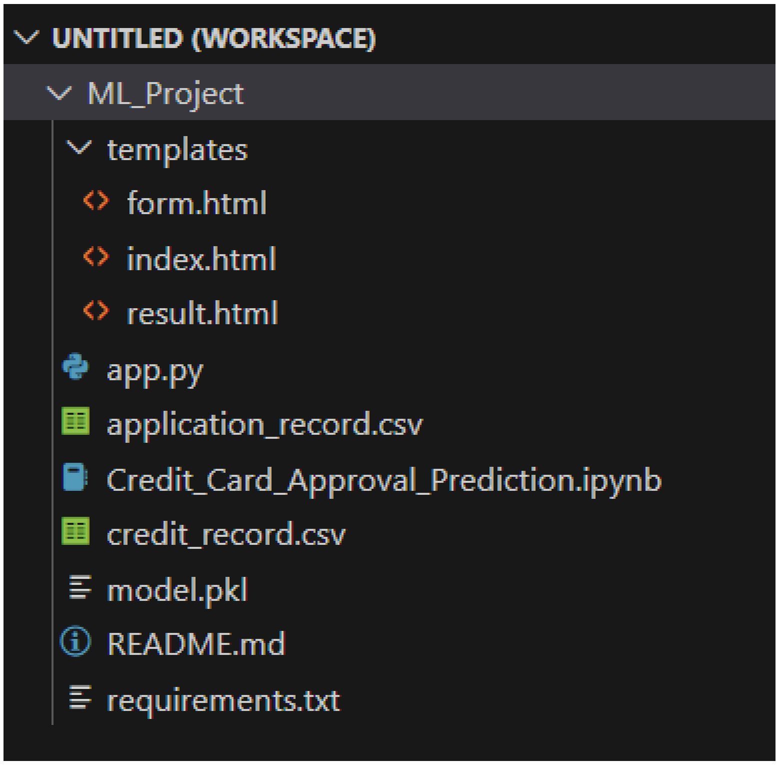
We must have prior knowledge of the following topics to complete this project.

- **ML Concepts**
  - Supervised learning: https://www.javatpoint.com/supervised-machine-learning
  - Unsupervised learning: https://www.javatpoint.com/unsupervised-machine-learning
  - Randomforest: https://www.javatpoint.com/machine-learning-random-forest-algorithm
  - Hyper-parameter tuning techniques: 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

Project Structure

Create a Project folder which contains files as shown below:



```
UNTITLED (WORKSPACE)
  ML_Project
    templates
      form.html
      index.html
      result.html
    app.py
    application_record.csv
    Credit_Card_Approval_Prediction.ipynb
    credit_record.csv
    model.pkl
    README.md
    requirements.txt
```

- 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.
- application_record.csv and credit_record.csv are the Datasets used.
- The Notebook file contains procedures for building the model.

Milestone 1: Define Problem / Problem Understanding

Activity 1: Specify the Business Problem

Credit card providers need a reliable and efficient system to predict whether an applicant is likely to repay their credit responsibly. Traditional manual evaluation methods can be time-consuming and inconsistent, leading to potential financial risks and missed opportunities.

The business problem is to develop a predictive model that can automate credit card application approvals based on historical demographic, financial, and behavioral data.

Activity 2: Business Requirements

A credit card approval prediction system should meet the following business requirements:

- **Accuracy & Reliability:**
  The model should be trained on historical credit and application data to provide reliable predictions with a high accuracy and low false positive rate.

- **Scalability:**
  The system should be capable of handling and processing large volumes of application data efficiently as the user base grows.

- **Adaptability:**
  The model must be flexible to incorporate new features, retrain with newer datasets, or accommodate evolving financial behaviors and credit rules.

- **Regulatory Compliance:**
  Must comply with financial data privacy regulations (e.g., GDPR, RBI guidelines) ensuring secure handling of personal and financial data.

- **User-friendly Interface:**
  A simple and intuitive web interface where bank employees or customers can input information and receive instant feedback.

Activity 3: Literature Survey

The literature review for this project involves studying existing research papers, case studies, and methodologies used for credit scoring and risk analysis using machine learning. Prior works show the application of Logistic Regression, Random Forests, and Gradient Boosting algorithms in predicting customer creditworthiness.

Key areas of focus include:

- Handling class imbalance (e.g., SMOTE)
- Feature engineering from financial and behavioral data
- Model interpretability and explainability
- Integration of ML models into real-time web apps using Flask

The survey also includes evaluating performance metrics (accuracy, F1-score, confusion matrix) and understanding the limitations of static rule-based approval systems.

Activity 4: Social or Business Impact

Social Impact:
This system enables faster, unbiased, and data-driven financial decisions, improving customer experience and ensuring fair credit access for all applicants. It reduces manual effort and human bias in financial approval systems.

Business Impact:
Improves efficiency in the credit approval pipeline, reduces default risk, and enhances profitability for banks. The system can also provide valuable insights into customer behavior, helping financial institutions refine their lending strategies and risk models.

Milestone 2: Data Collection

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: Collecting 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.

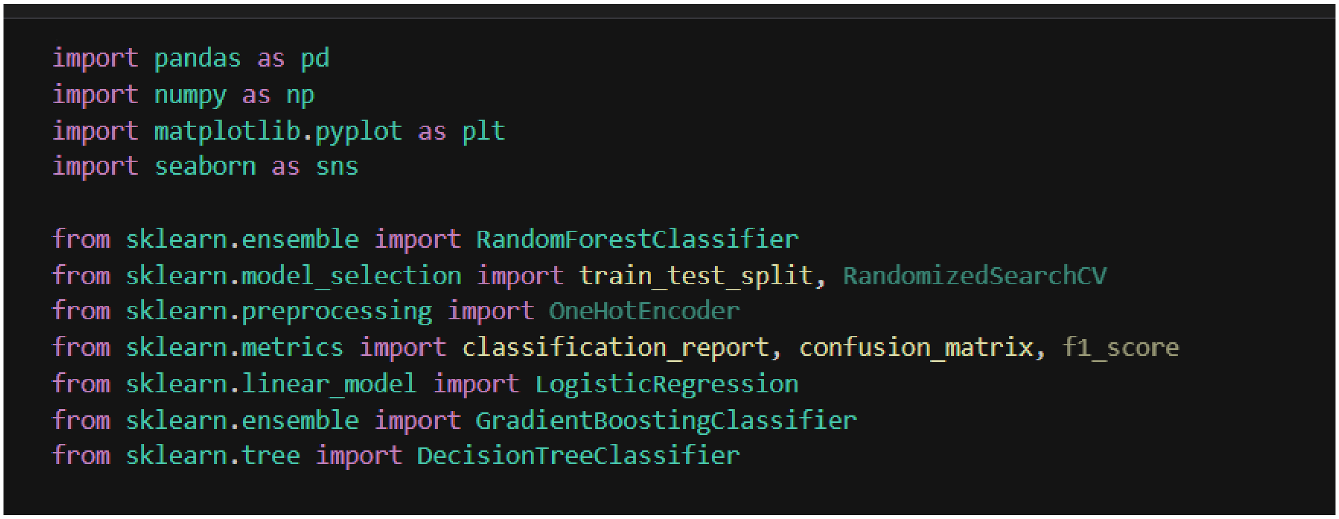
Link: https://www.kaggle.com/namphuengauawatcharo/credit-card-approval-prediction/data

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.

Activity 2: Importing the Libraries

Import the necessary libraries as shown in the image.



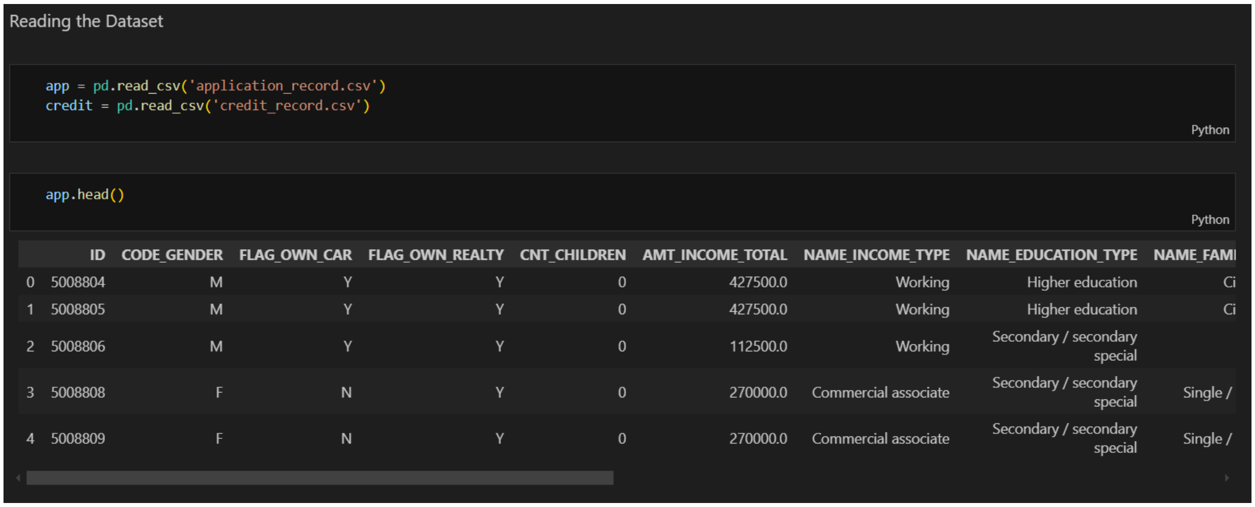
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

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 3: Reading 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.

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



Reading the Dataset

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

```
app.head()
```

	ID	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	NAME_INCOME_TYPE	NAME_EDUCATION_TYPE	NAME_FAMI
0	5008804	M	Y	Y	0	427500.0	Working	Higher education	Ci
1	5008805	M	Y	Y	0	427500.0	Working	Higher education	Ci
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	Single /
4	5008809	F	N	Y	0	270000.0	Commercial associate	Secondary / secondary special	Single /

Milestone 3: Visualizing and Analyzing the data

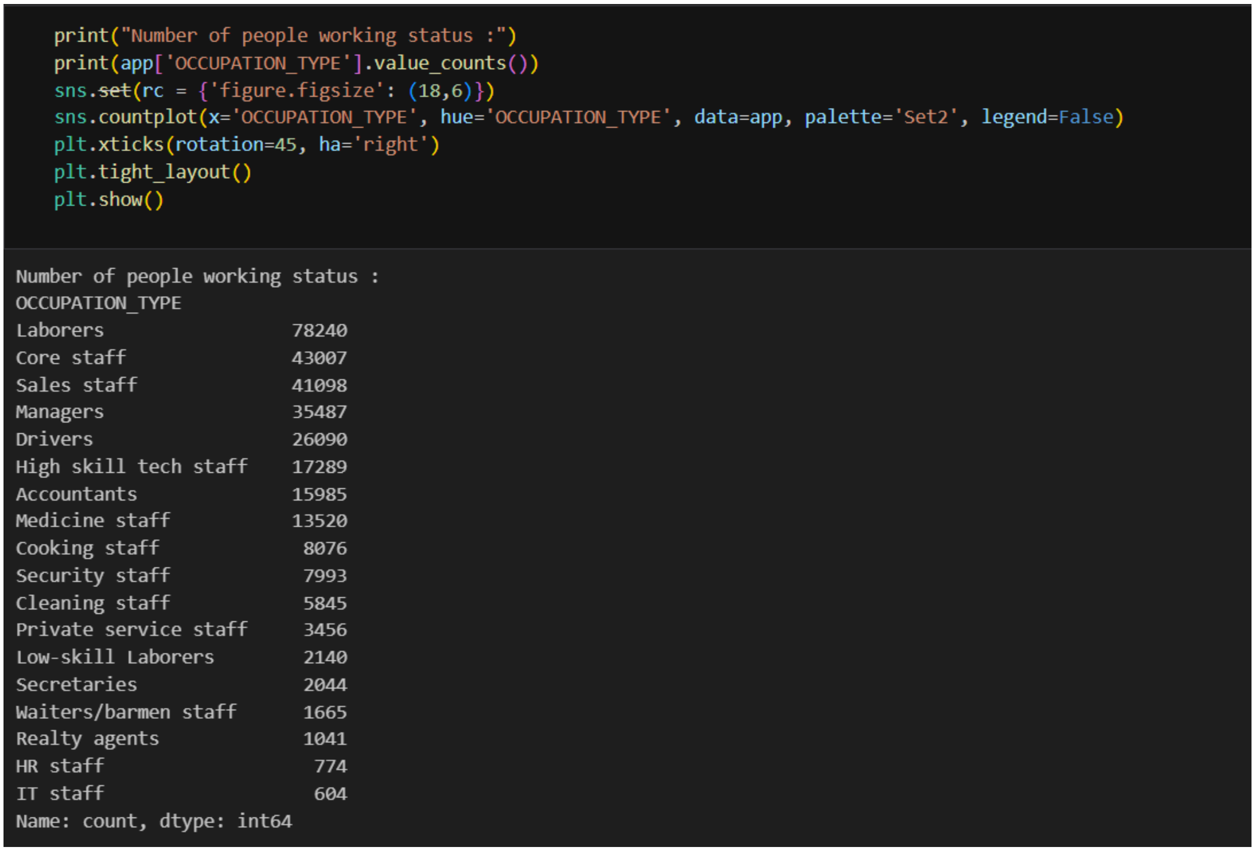
Activity 1: Visual analysis

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 1.1: Univariate analysis

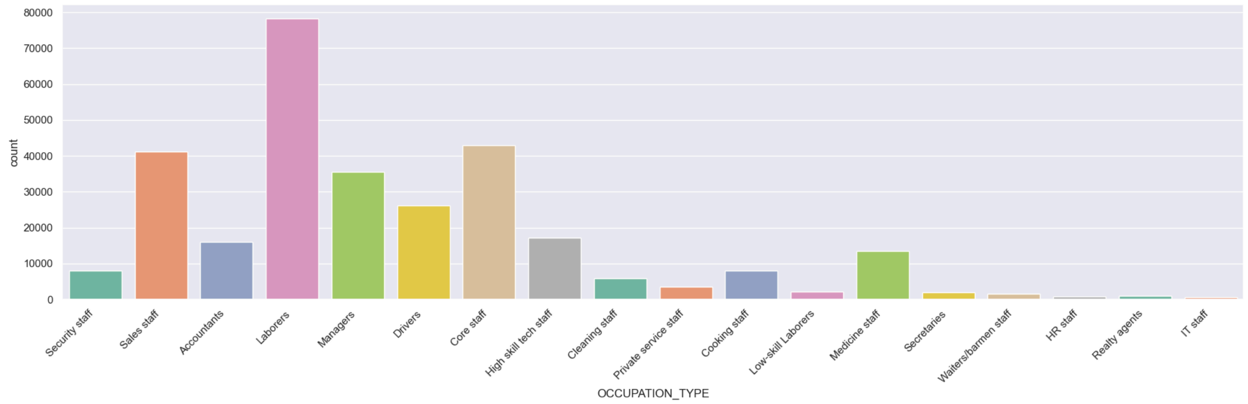
In simple words, univariate analysis is understanding the data with a single feature. Here we have used a count plot.

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

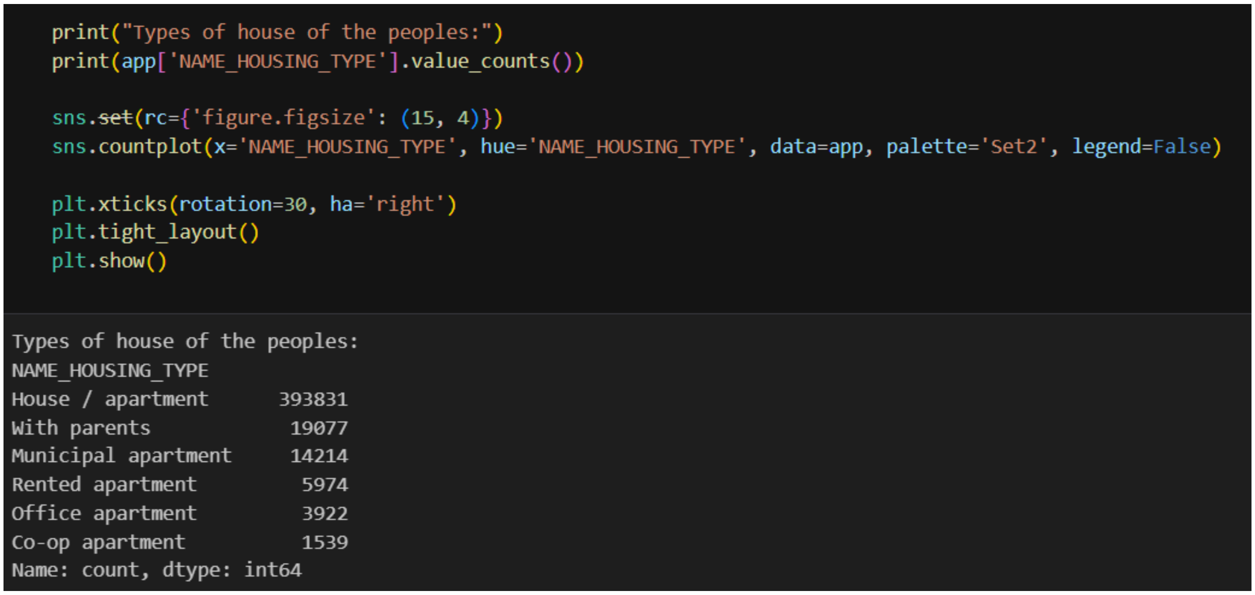


```
print("Number of people working status :")
print(app['OCCUPATION_TYPE'].value_counts())
sns.set(rc = {'figure.figsize': (18,6)})
sns.countplot(x='OCCUPATION_TYPE', hue='OCCUPATION_TYPE', data=app, palette='Set2', legend=False)
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```

```
Number of people working status :
OCCUPATION_TYPE
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: count, dtype: int64
```

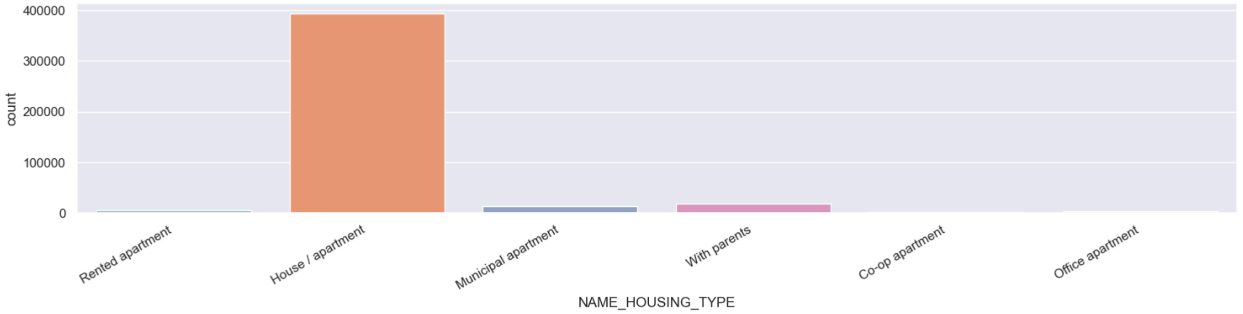


- Count plot is used on housing type feature. With the countplot(), we are going to count the unique category. From the below graph, we found the number of House/apartment are high when compared to other types. For the exact count, value counts() are used.

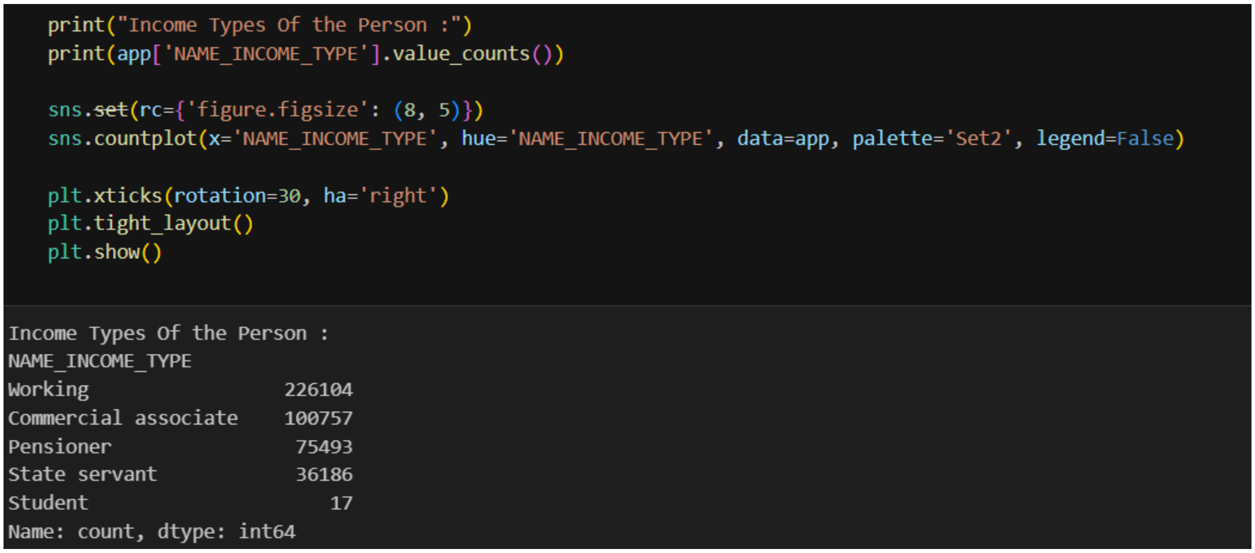


```
print("Types of house of the peoples:")
print(app['NAME_HOUSING_TYPE'].value_counts())

sns.set(rc={'figure.figsize': (15, 4)})
sns.countplot(x='NAME_HOUSING_TYPE', hue='NAME_HOUSING_TYPE', data=app, palette='Set2', legend=False)

plt.xticks(rotation=30, ha='right')
plt.tight_layout()
plt.show()
```

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

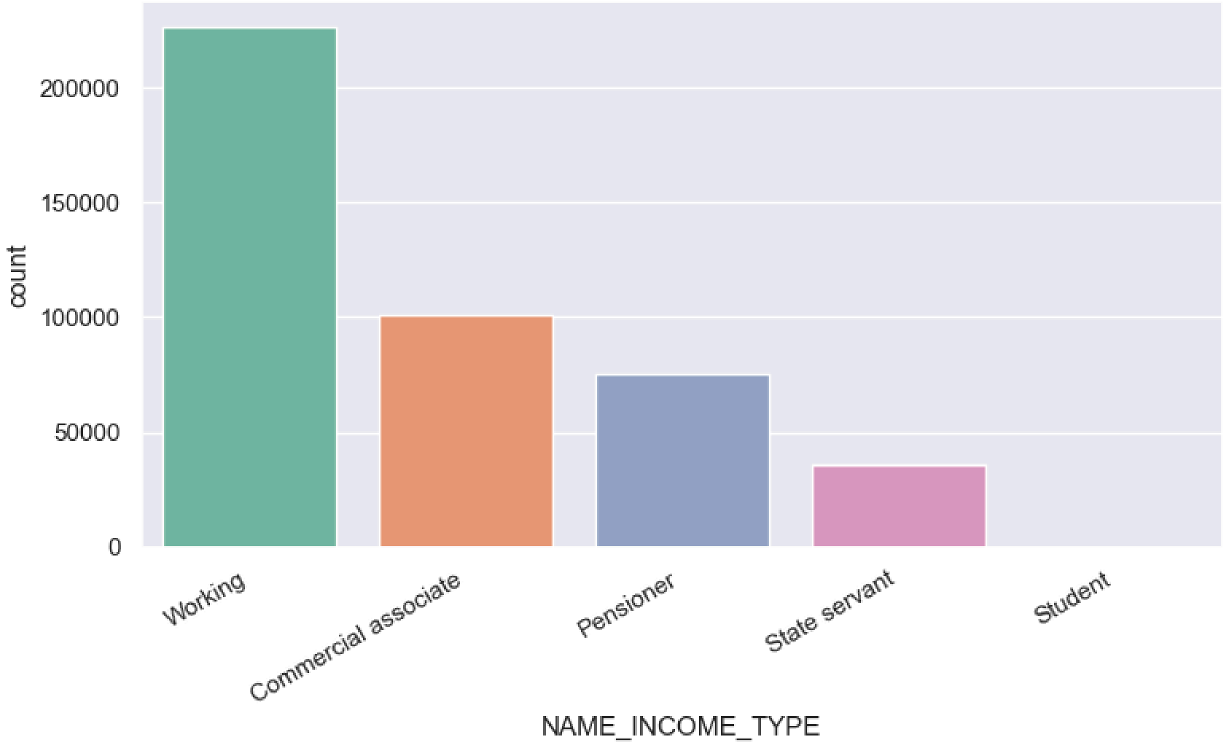


- Count plot is used on income type features. With the countplot(), we are going to count the unique category. From the below graph, we found the number of working applicants are high when compared to other types. For the exact count, value counts() are used.



```
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', hue='NAME_INCOME_TYPE', data=app, palette='Set2', legend=False)

plt.xticks(rotation=30, ha='right')
plt.tight_layout()
plt.show()
```

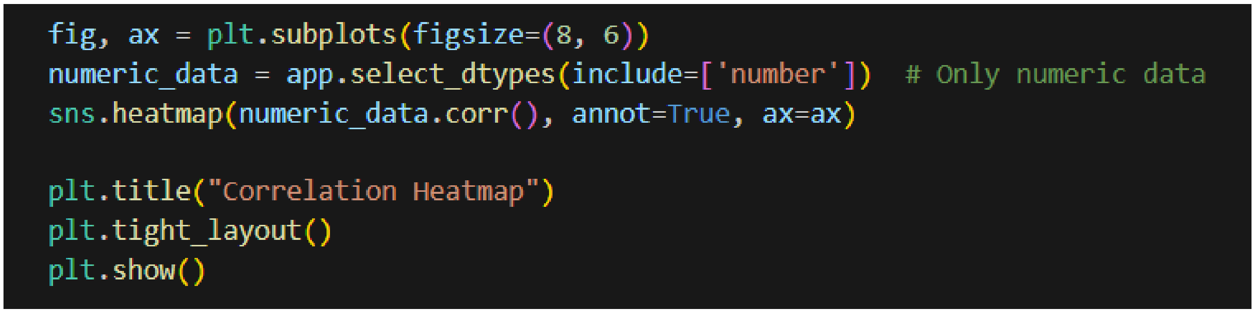
```
Income Types Of the Person :
NAME_INCOME_TYPE
Working                 226104
Commercial associate    100757
Pensioner                75493
State servant            36186
Student                     17
Name: count, dtype: int64
```



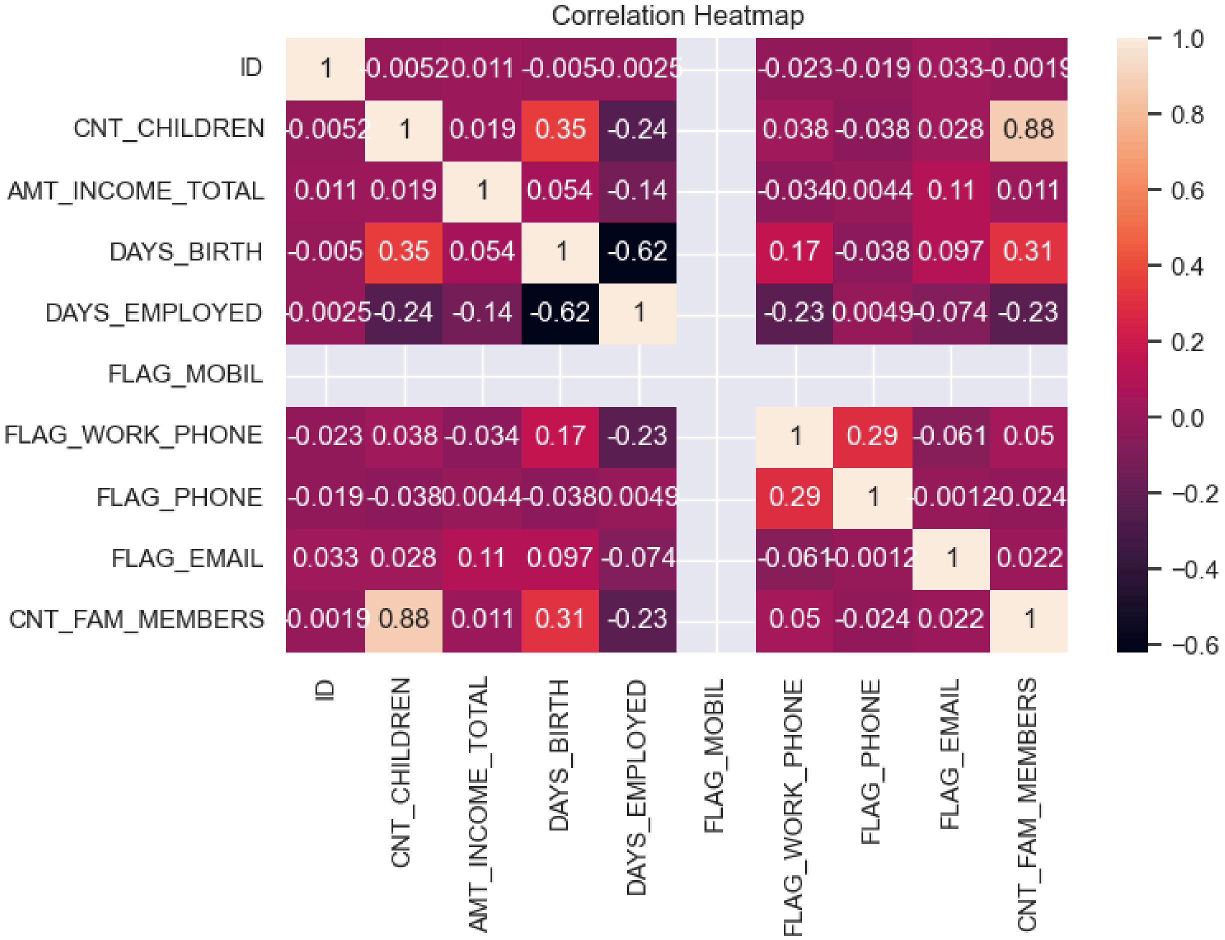
Activity 1.2: Multivariate Analysis

In simple words, multivariate analysis is to find the relation between multiple features. Here we have used heatmap() from the 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.

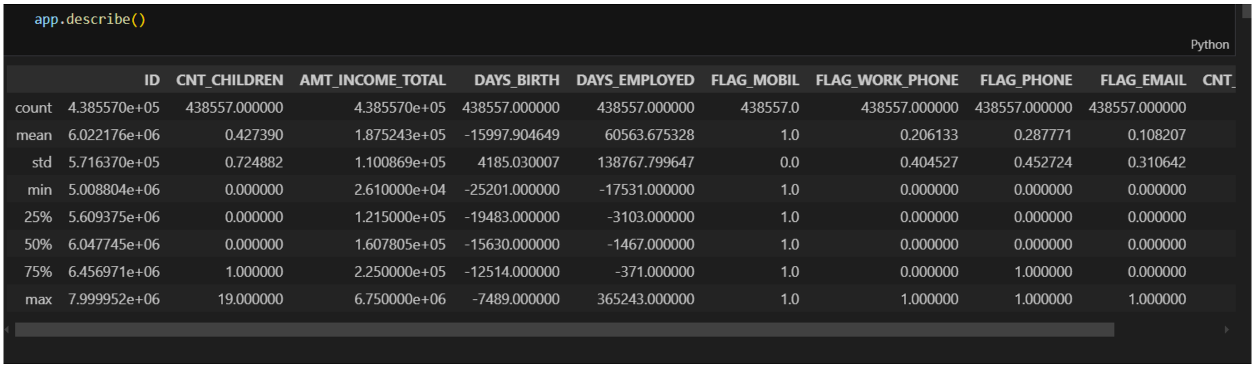


```
fig, ax = plt.subplots(figsize=(8, 6))
numeric_data = app.select_dtypes(include=['number'])  # Only numeric data
sns.heatmap(numeric_data.corr(), annot=True, ax=ax)

plt.title("Correlation Heatmap")
plt.tight_layout()
plt.show()
```



Activity 2: Descriptive Analysis

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.



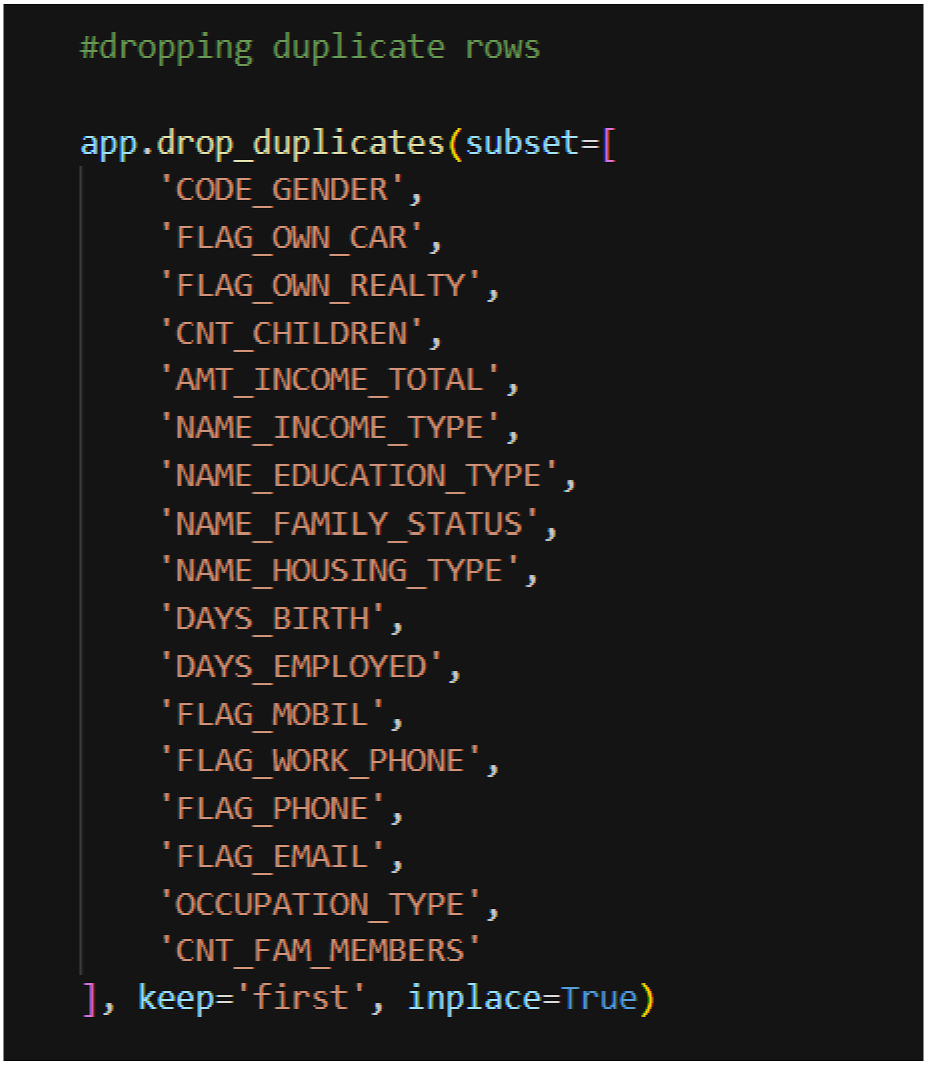
```
app.describe()
```

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

Milestone 4: Data Pre-Processing

Activity 1: Drop unwanted features

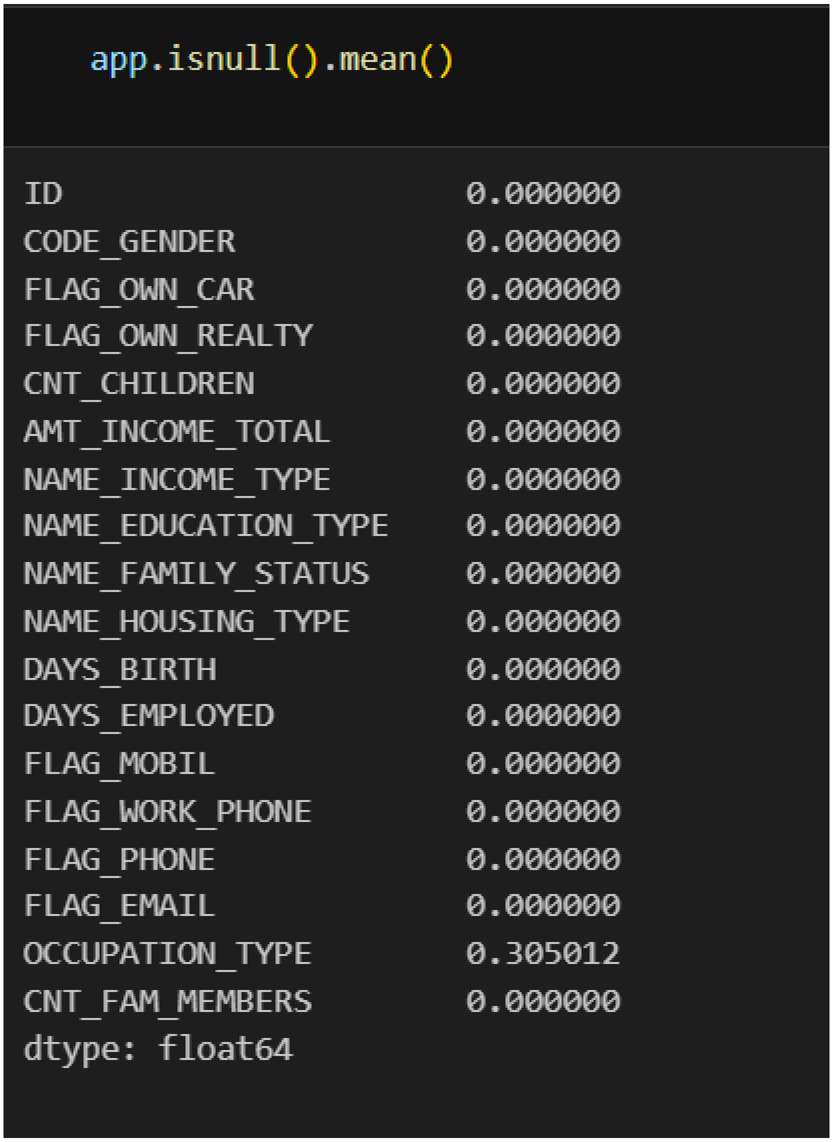
Generally, applicant ids are unique in nature. But in our dataset we found some of the ids are repeating multiple times. To handle this we have to remove the duplicate rows. Drop duplicates() function from pandas is used to remove the duplicate rows. Refer to the below image.



```
#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)
```

Activity 2: Handling missing Values

For checking the null values, df.isnull() function is used. To sum those null values we use the sum() function. 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
```

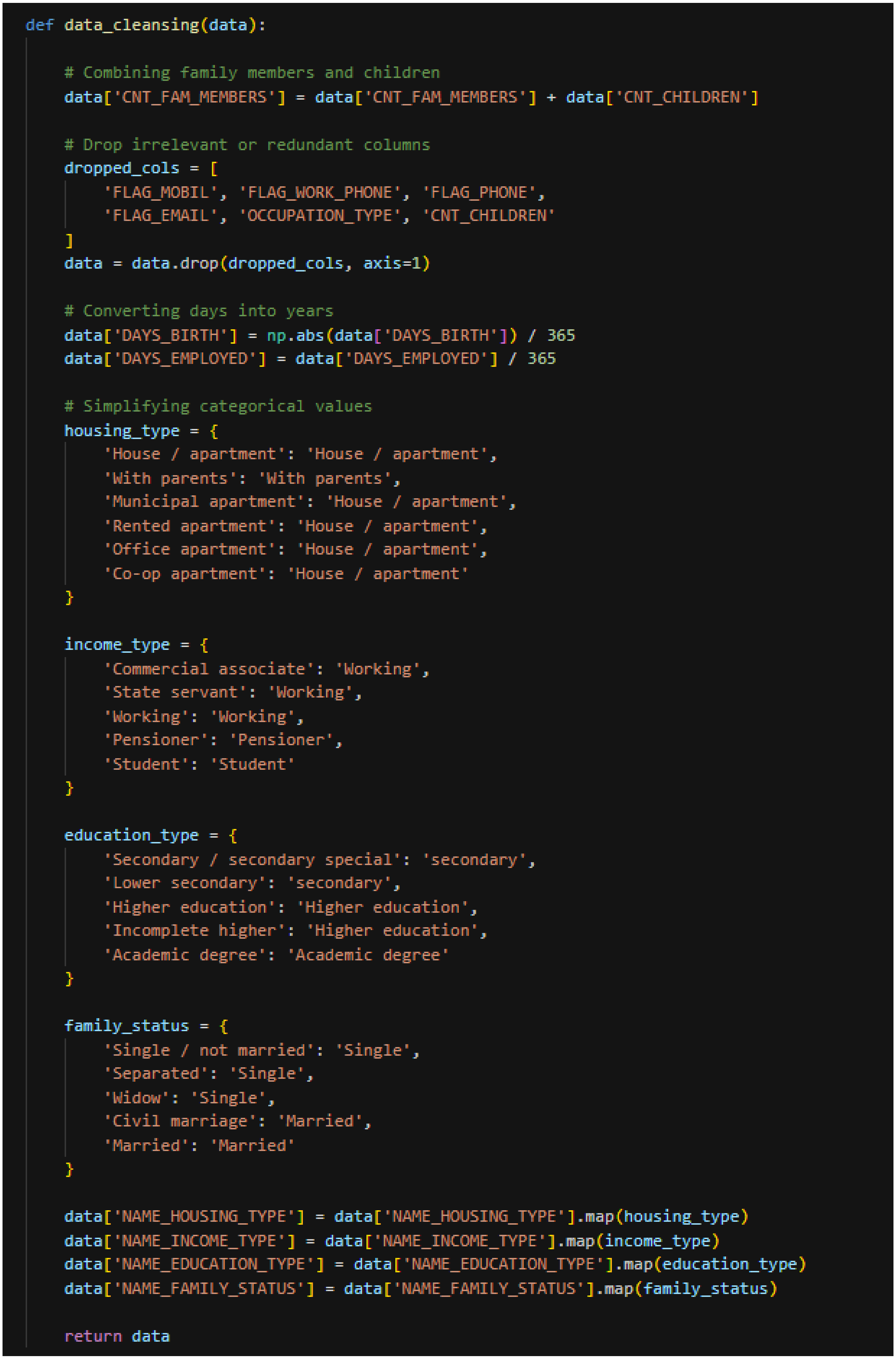
We have null values in the occupation type feature. However, we are going to remove that column in further process. So, let's skip the handling null values process.

Activity 3: Data Cleaning and merging

In this process, we are going to combine two inter-related columns. Our dataset has 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 the number of family members with the number of childrens.
- Six unwanted columns are dropped by drop() function. Refer to the below image to know the column name.

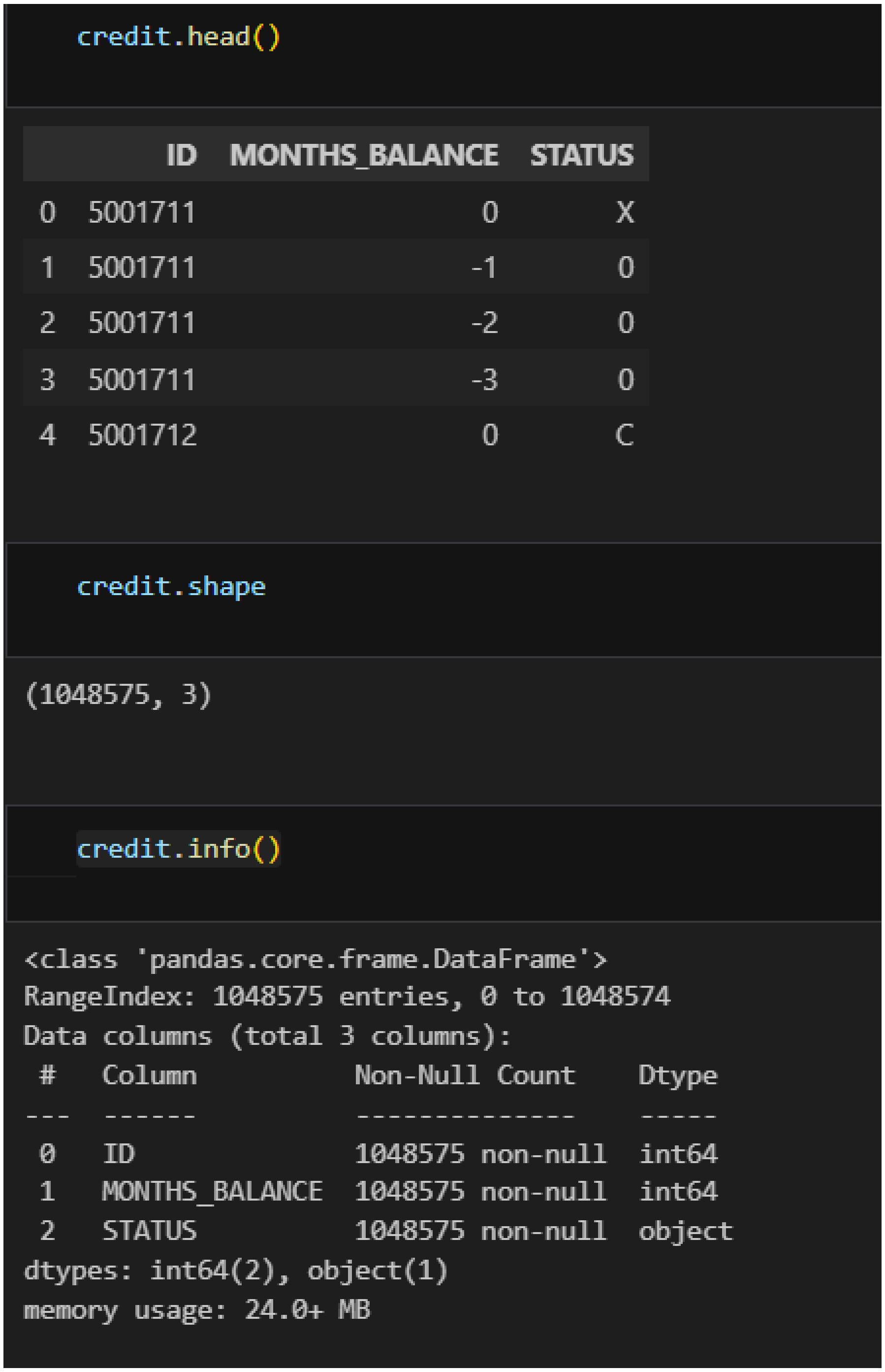
- Days birth and days employed columns have negative values. To convert the negative values to absolute values we use abs() function.
- Feature mapping is done in housing type, income type, education type and family type columns. (This feature mapping step is an optional step).



```python
def data_cleansing(data):

    # Combining family members and children
    data['CNT_FAM_MEMBERS'] = data['CNT_FAM_MEMBERS'] + data['CNT_CHILDREN']

    # Drop irrelevant or redundant columns
    dropped_cols = [
        'FLAG_MOBIL', 'FLAG_WORK_PHONE', 'FLAG_PHONE',
        'FLAG_EMAIL', 'OCCUPATION_TYPE', 'CNT_CHILDREN'
    ]
    data = data.drop(dropped_cols, axis=1)

    # Converting days into years
    data['DAYS_BIRTH'] = np.abs(data['DAYS_BIRTH']) / 365
    data['DAYS_EMPLOYED'] = data['DAYS_EMPLOYED'] / 365

    # Simplifying categorical values
    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)
```

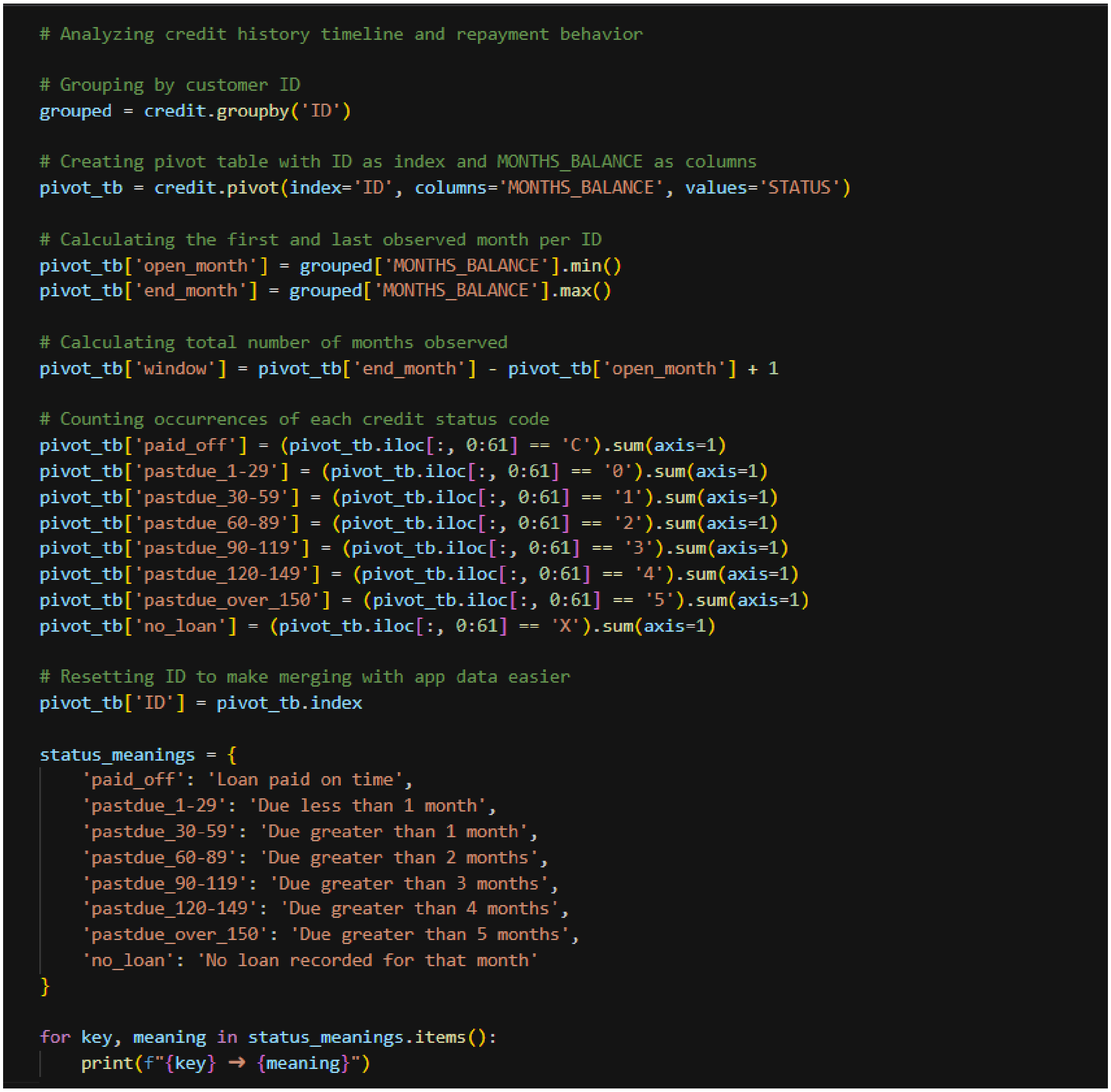
```
credit.info()
```

```
<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
```

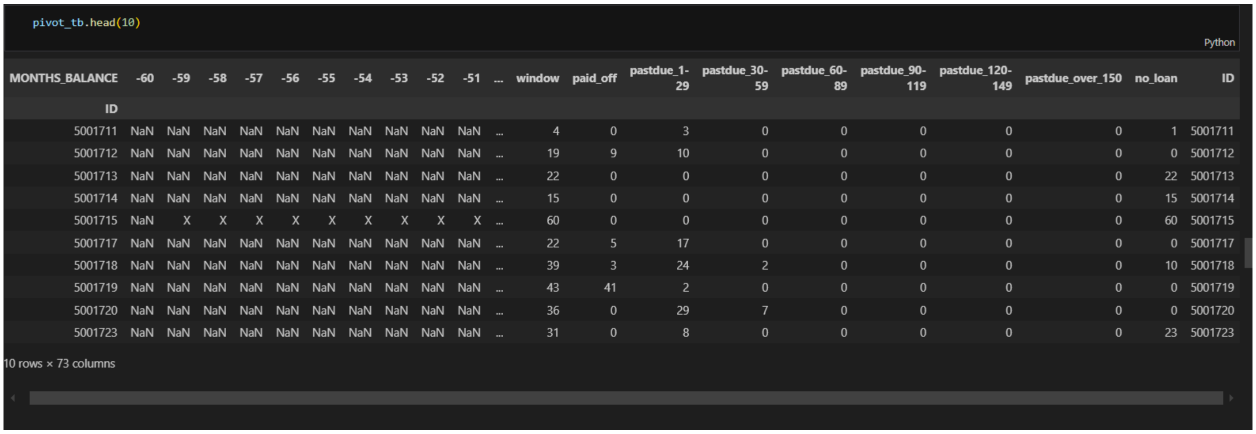
We are grouping the ID column and saving it as a variable ‘grouped’.

We are using it as an index ID and for columns 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 subtracting end_months – open_months



```
# Analyzing credit history timeline and repayment behavior

# Grouping by customer ID
grouped = credit.groupby('ID')

# Creating pivot table with ID as index and MONTHS_BALANCE as columns
pivot_tb = credit.pivot(index='ID', columns='MONTHS_BALANCE', values='STATUS')

# Calculating the first and last observed month per ID
pivot_tb['open_month'] = grouped['MONTHS_BALANCE'].min()
pivot_tb['end_month'] = grouped['MONTHS_BALANCE'].max()

# Calculating total number of months observed
pivot_tb['window'] = pivot_tb['end_month'] - pivot_tb['open_month'] + 1

# Counting occurrences of each credit status code
pivot_tb['paid_off'] = (pivot_tb.iloc[:, 0:61] == 'C').sum(axis=1)
pivot_tb['pastdue_1-29'] = (pivot_tb.iloc[:, 0:61] == '0').sum(axis=1)
pivot_tb['pastdue_30-59'] = (pivot_tb.iloc[:, 0:61] == '1').sum(axis=1)
pivot_tb['pastdue_60-89'] = (pivot_tb.iloc[:, 0:61] == '2').sum(axis=1)
pivot_tb['pastdue_90-119'] = (pivot_tb.iloc[:, 0:61] == '3').sum(axis=1)
pivot_tb['pastdue_120-149'] = (pivot_tb.iloc[:, 0:61] == '4').sum(axis=1)
pivot_tb['pastdue_over_150'] = (pivot_tb.iloc[:, 0:61] == '5').sum(axis=1)
pivot_tb['no_loan'] = (pivot_tb.iloc[:, 0:61] == 'X').sum(axis=1)

# Resetting ID to make merging with app data easier
pivot_tb['ID'] = pivot_tb.index

status_meanings = {
    'paid_off': 'Loan paid on time',
    'pastdue_1-29': 'Due less than 1 month',
    'pastdue_30-59': 'Due greater than 1 month',
    'pastdue_60-89': 'Due greater than 2 months',
    'pastdue_90-119': 'Due greater than 3 months',
    'pastdue_120-149': 'Due greater than 4 months',
    'pastdue_over_150': 'Due greater than 5 months',
    'no_loan': 'No loan recorded for that month'
}

for key, meaning in status_meanings.items():
    print(f"{key} → {meaning}")
```



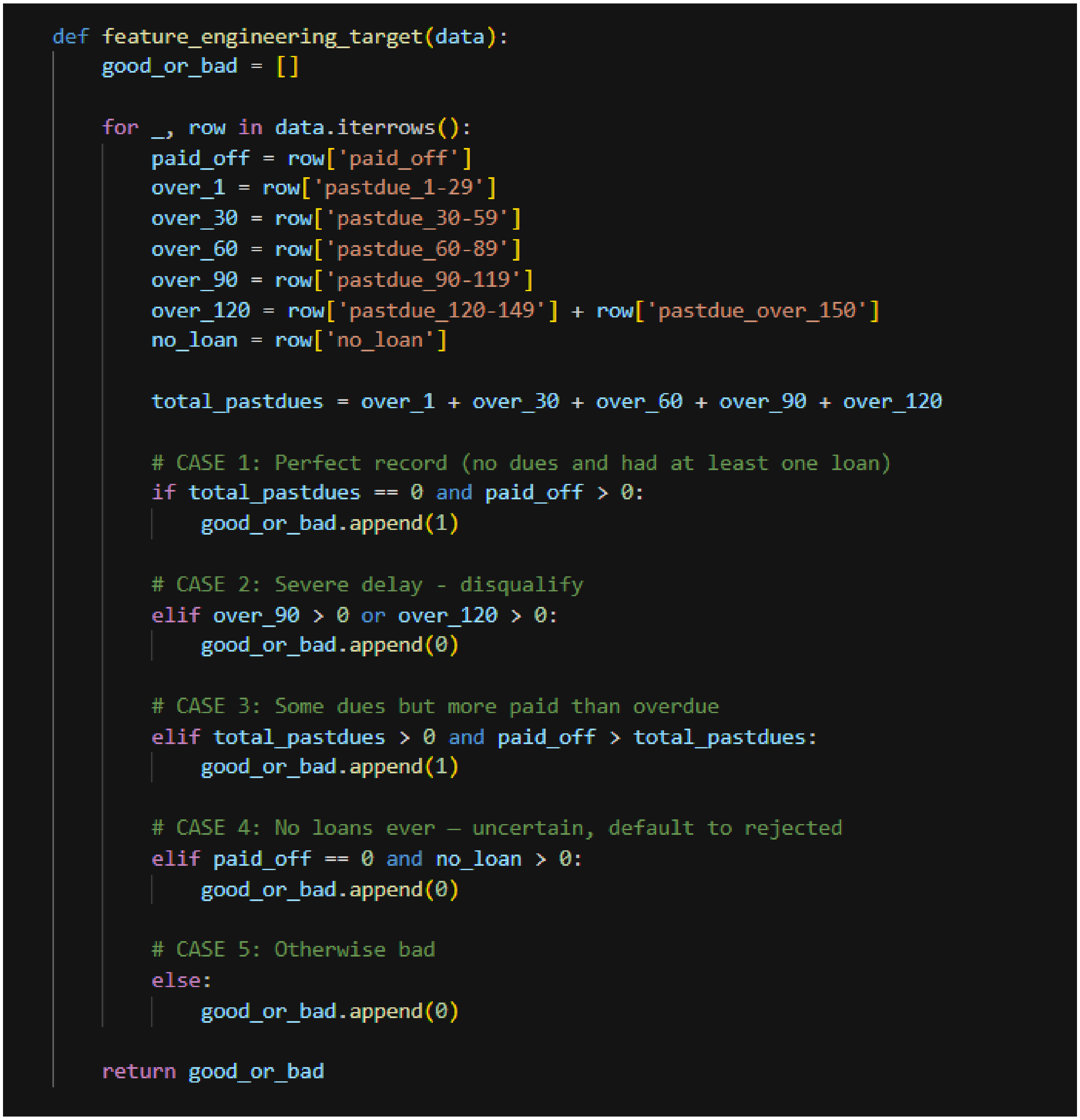
```
pivot_tb.head(10)
```

MONTHS_BALANCE	-60	-59	-58	-57	-56	-55	-54	-53	-52	-51	...	window	paid_off	pastdue_1-29	pastdue_30-59	pastdue_60-89	pastdue_90-119	pastdue_120-149	pastdue_over_150	no_loan	ID
ID																					
5001711	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	4	0	3	0	0	0	0	0	1	5001711
5001712	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	19	9	10	0	0	0	0	0	0	5001712
5001713	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	22	0	0	0	0	0	0	0	22	5001713
5001714	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	15	0	0	0	0	0	0	0	15	5001714
5001715	NaN	X	X	X	X	X	X	X	X	X	...	60	0	0	0	0	0	0	0	60	5001715
5001717	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	22	5	17	0	0	0	0	0	0	5001717
5001718	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	39	3	24	2	0	0	0	0	10	5001718
5001719	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	43	41	2	0	0	0	0	0	0	5001719
5001720	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	36	0	29	7	0	0	0	0	0	5001720
5001723	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	31	0	8	0	0	0	0	0	23	5001723

10 rows × 73 columns

Activity 4: Feature Engineering

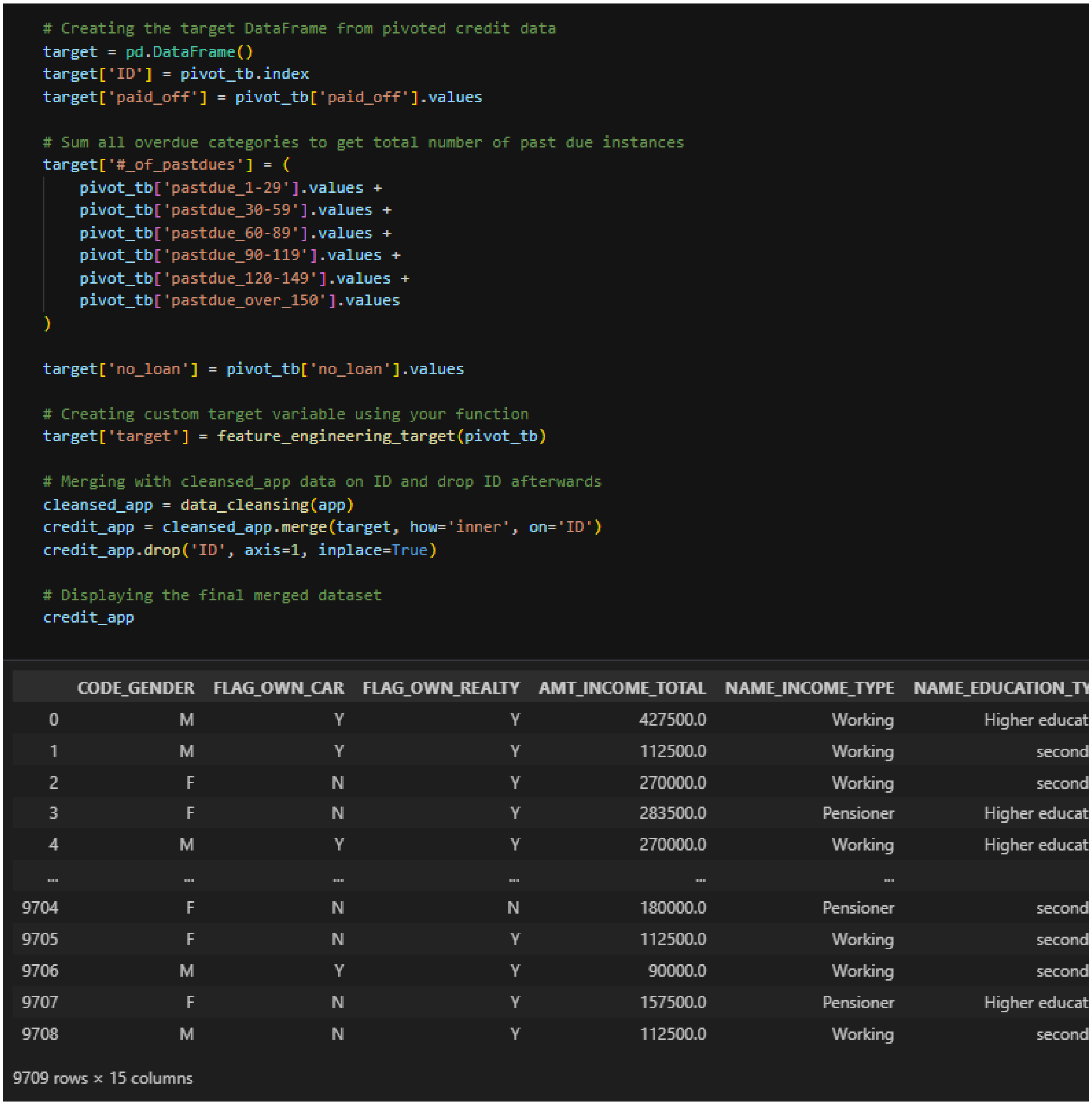
Converting the multi-classification into binary classification. For a clear understanding refer the below two images



```python
def feature_engineering_target(data):
    good_or_bad = []

    for _, 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']

        total_pastdues = over_1 + over_30 + over_60 + over_90 + over_120

        # CASE 1: Perfect record (no dues and had at least one loan)
        if total_pastdues == 0 and paid_off > 0:
            good_or_bad.append(1)

        # CASE 2: Severe delay - disqualify
        elif over_90 > 0 or over_120 > 0:
            good_or_bad.append(0)

        # CASE 3: Some dues but more paid than overdue
        elif total_pastdues > 0 and paid_off > total_pastdues:
            good_or_bad.append(1)

        # CASE 4: No loans ever — uncertain, default to rejected
        elif paid_off == 0 and no_loan > 0:
            good_or_bad.append(0)

        # CASE 5: Otherwise bad
        else:
            good_or_bad.append(0)

    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.



```python
# Creating the target DataFrame from pivoted credit data
target = pd.DataFrame()
target['ID'] = pivot_tb.index
target['paid_off'] = pivot_tb['paid_off'].values

# Sum all overdue categories to get total number of past due instances
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

# Creating custom target variable using your function
target['target'] = feature_engineering_target(pivot_tb)

# Merging with cleansed_app data on ID and drop ID afterwards
cleansed_app = data_cleansing(app)
credit_app = cleansed_app.merge(target, how='inner', on='ID')
credit_app.drop('ID', axis=1, inplace=True)

# Displaying the final merged dataset
credit_app
```

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

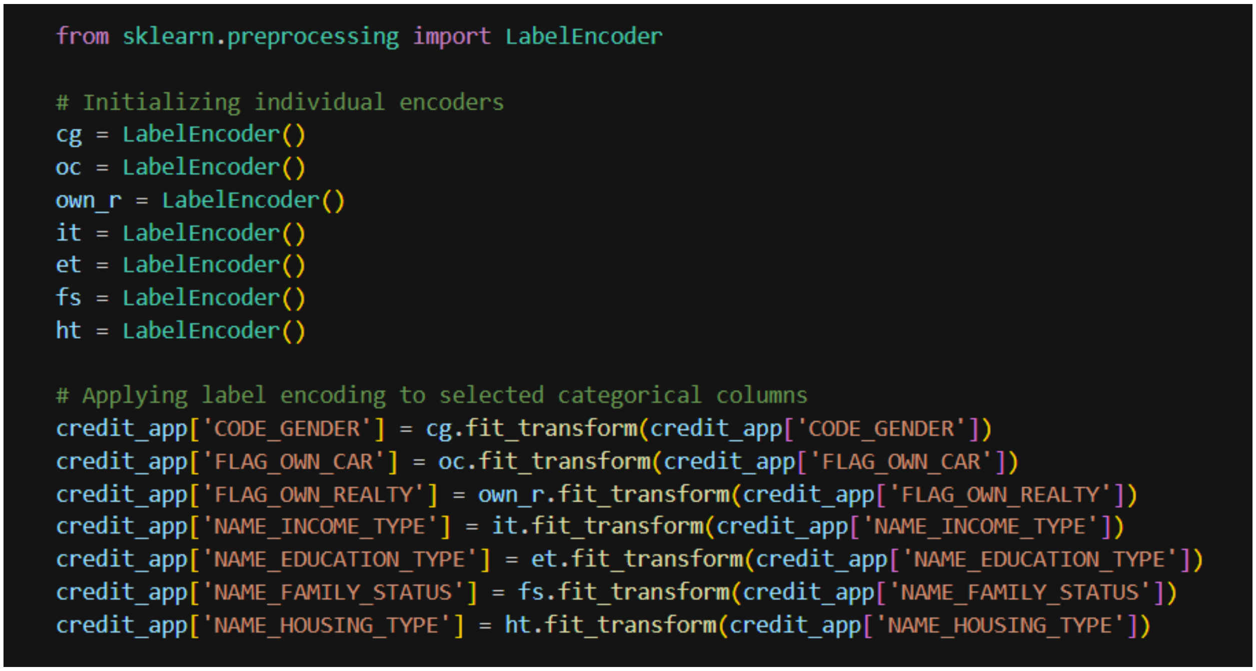
9709 rows × 15 columns

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.

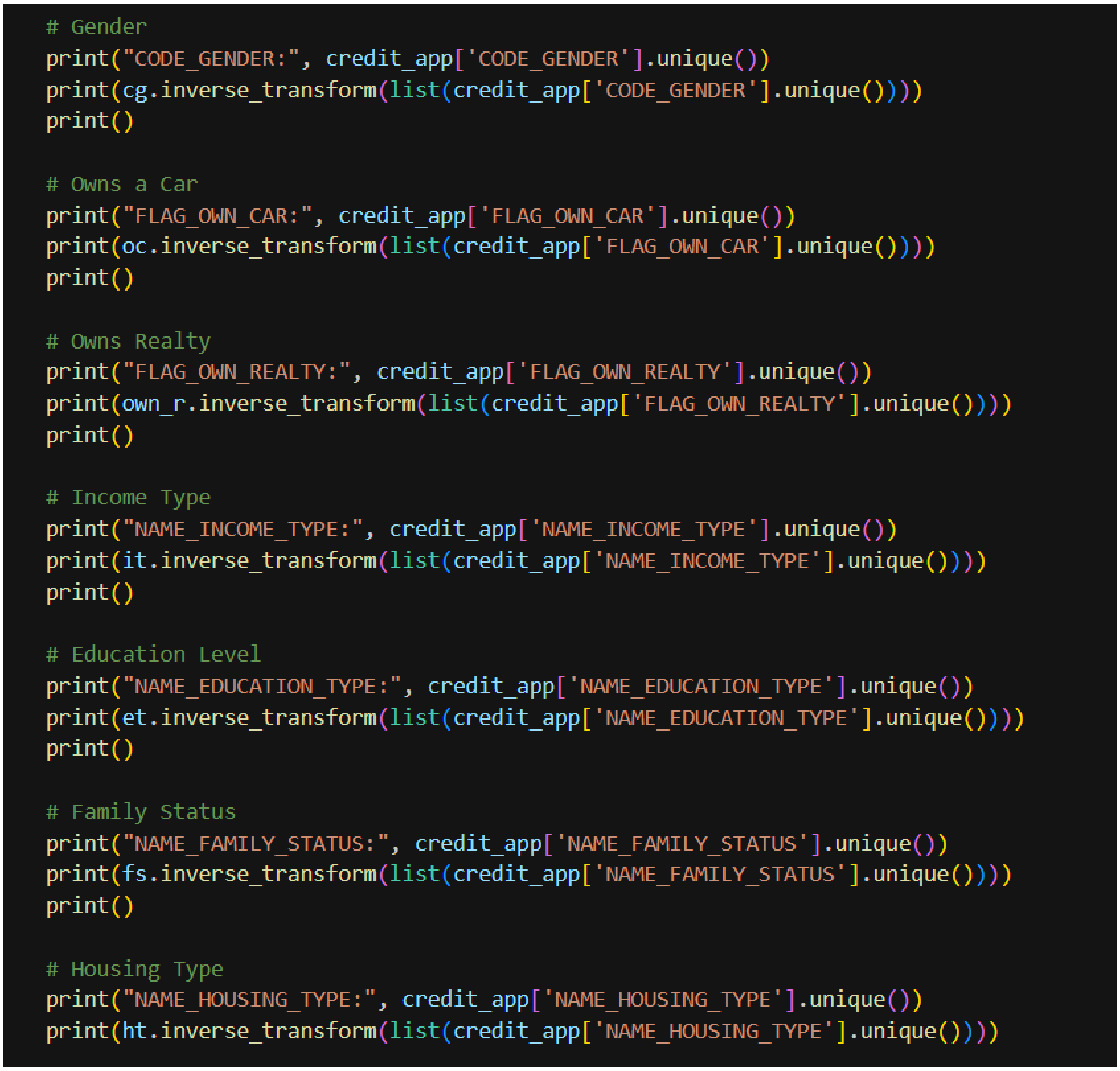
To convert the categorical features into numerical features we use encoding techniques. There are several techniques but in our project we are using label encoding.

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



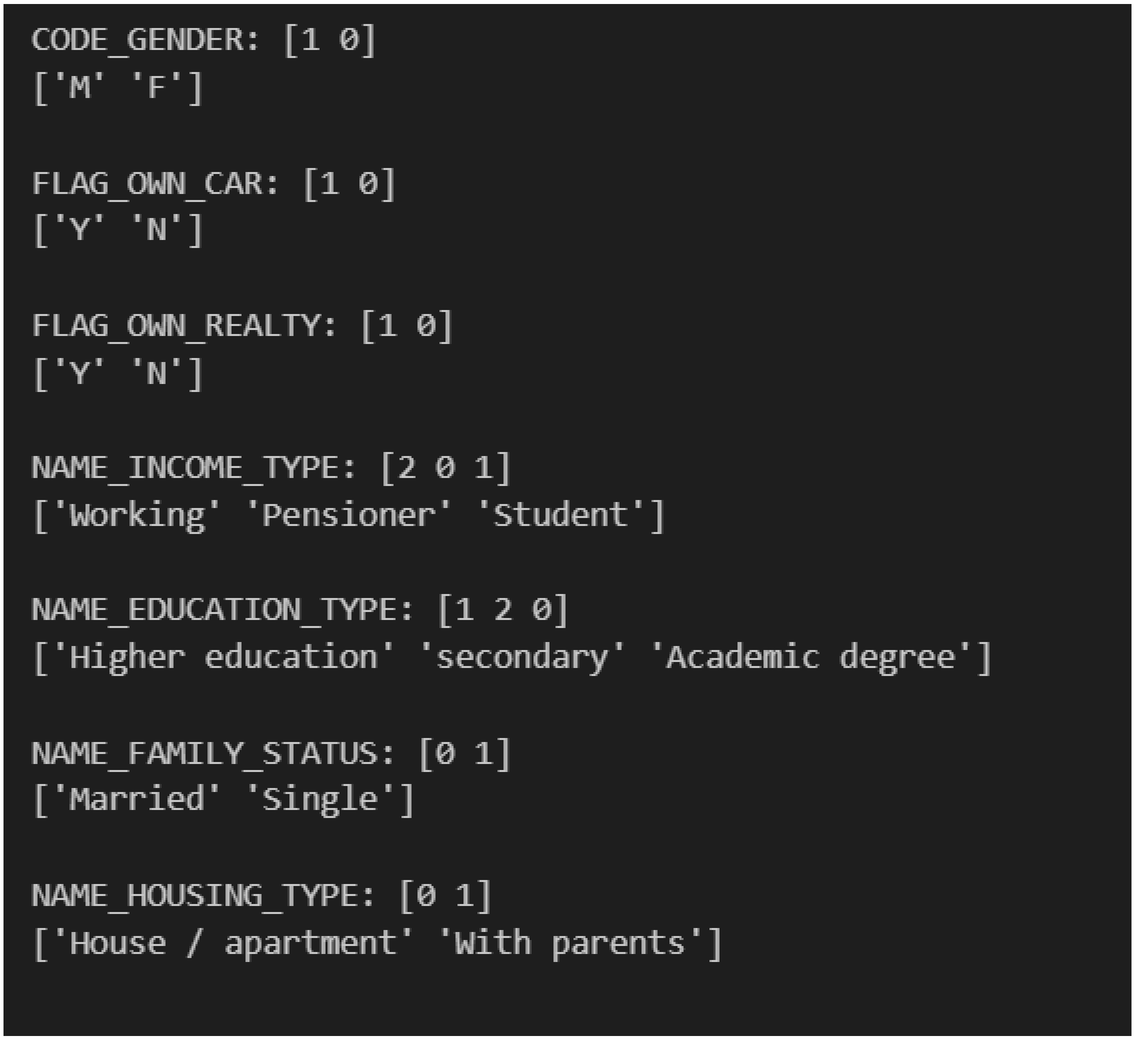
```
from sklearn.preprocessing import LabelEncoder

# Initializing individual encoders
cg = LabelEncoder()
oc = LabelEncoder()
own_r = LabelEncoder()
it = LabelEncoder()
et = LabelEncoder()
fs = LabelEncoder()
ht = LabelEncoder()

# Applying label encoding to selected categorical columns
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)



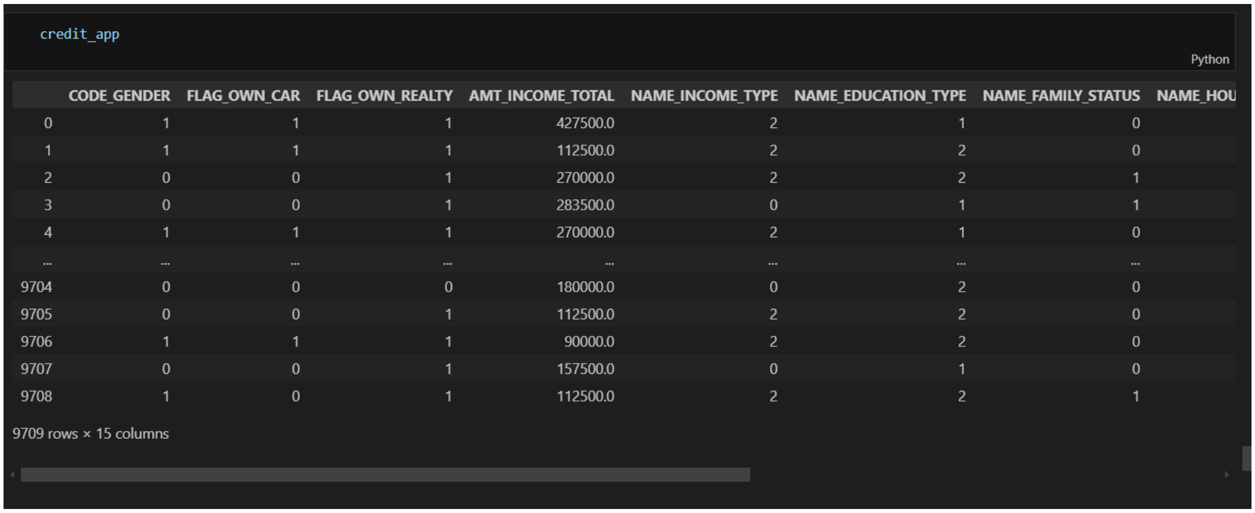
```python
# Gender
print("CODE_GENDER:", credit_app['CODE_GENDER'].unique())
print(cg.inverse_transform(list(credit_app['CODE_GENDER'].unique())))
print()

# Owns a Car
print("FLAG_OWN_CAR:", credit_app['FLAG_OWN_CAR'].unique())
print(oc.inverse_transform(list(credit_app['FLAG_OWN_CAR'].unique())))
print()

# Owns Realty
print("FLAG_OWN_REALTY:", credit_app['FLAG_OWN_REALTY'].unique())
print(own_r.inverse_transform(list(credit_app['FLAG_OWN_REALTY'].unique())))
print()

# Income Type
print("NAME_INCOME_TYPE:", credit_app['NAME_INCOME_TYPE'].unique())
print(it.inverse_transform(list(credit_app['NAME_INCOME_TYPE'].unique())))
print()

# Education Level
print("NAME_EDUCATION_TYPE:", credit_app['NAME_EDUCATION_TYPE'].unique())
print(et.inverse_transform(list(credit_app['NAME_EDUCATION_TYPE'].unique())))
print()

# Family Status
print("NAME_FAMILY_STATUS:", credit_app['NAME_FAMILY_STATUS'].unique())
print(fs.inverse_transform(list(credit_app['NAME_FAMILY_STATUS'].unique())))
print()

# Housing Type
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:-



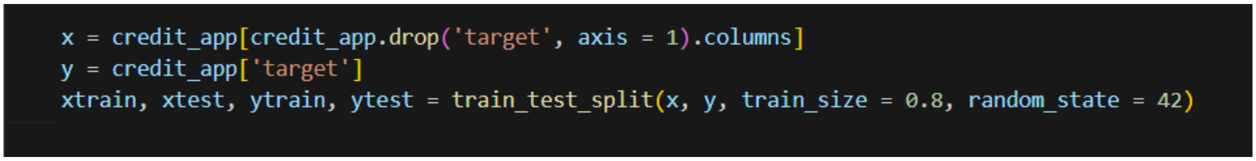
```
credit_app
```

	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	AMT_INCOME_TOTAL	NAME_INCOME_TYPE	NAME_EDUCATION_TYPE	NAME_FAMILY_STATUS	NAME_HOU
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 an independent variable and ‘y’ is a dependent variable.



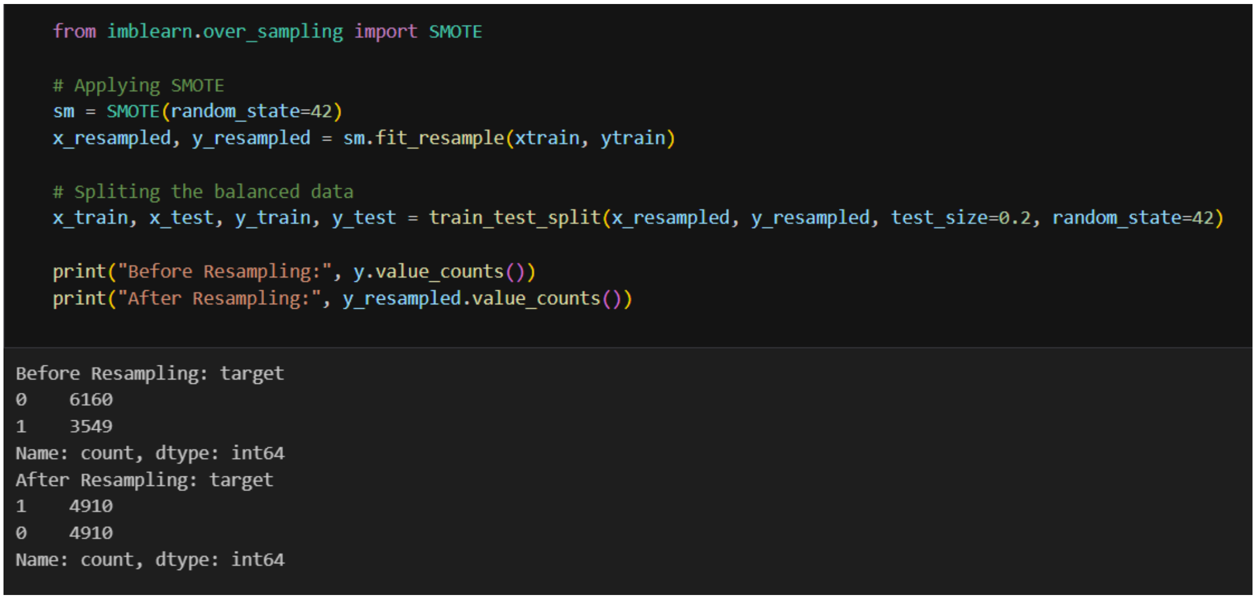
```
x = credit_app[credit_app.drop('target', axis = 1).columns]
y = credit_app['target']
xtrain, xtest, ytrain, ytest = train_test_split(x, y, train_size = 0.8, random_state = 42)
```

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.

But before that lets balance our data using SMOTE:

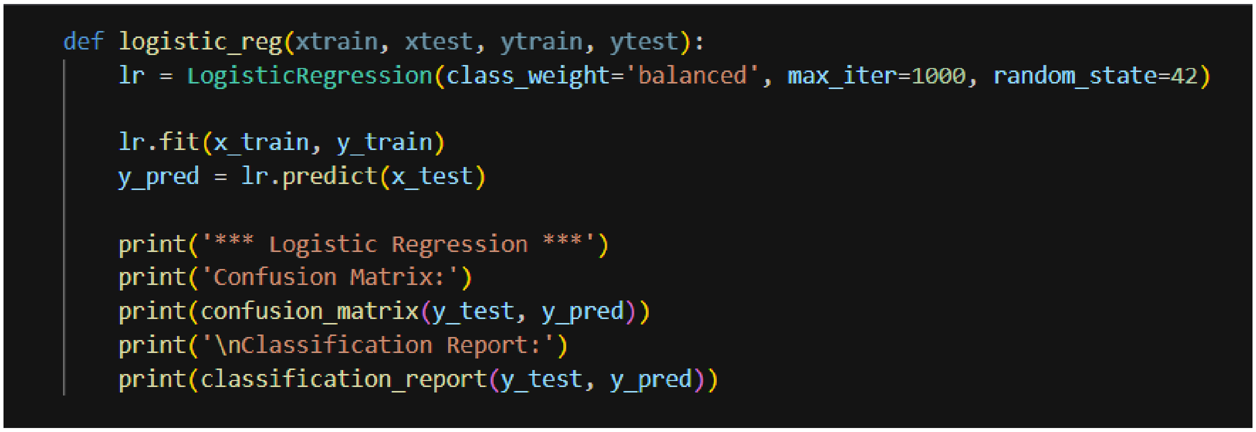


```
from imblearn.over_sampling import SMOTE

# Applying SMOTE
sm = SMOTE(random_state=42)
x_resampled, y_resampled = sm.fit_resample(xtrain, ytrain)

# Spliting the balanced data
x_train, x_test, y_train, y_test = train_test_split(x_resampled, y_resampled, test_size=0.2, random_state=42)

print("Before Resampling:", y.value_counts())
print("After Resampling:", y_resampled.value_counts())
```

```
Before Resampling: target
0    6160
1    3549
Name: count, dtype: int64
After Resampling: target
1    4910
0    4910
Name: count, dtype: int64
```

Activity 1.1: Logistic Regression Model

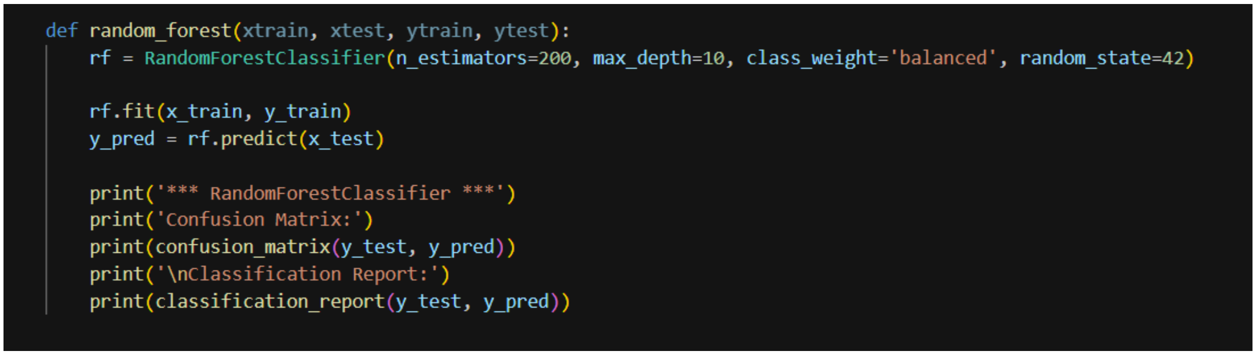
A function named logistic_reg is created and train and test data are passed as the parameters. Inside the function, LogisticRegression() algorithm is initialized and training data is passed to the model with the .fit() function. Test data is predicted with .predict() function and saved in a new variable.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



```
def logistic_reg(xtrain, xtest, ytrain, ytest):
    lr = LogisticRegression(class_weight='balanced', max_iter=1000, random_state=42)

    lr.fit(x_train, y_train)
    y_pred = lr.predict(x_test)

    print('*** Logistic Regression ***')
    print('Confusion Matrix:')
    print(confusion_matrix(y_test, y_pred))
    print('\nClassification Report:')
    print(classification_report(y_test, y_pred))
```

Activity 1.2:Random Forest Classifier

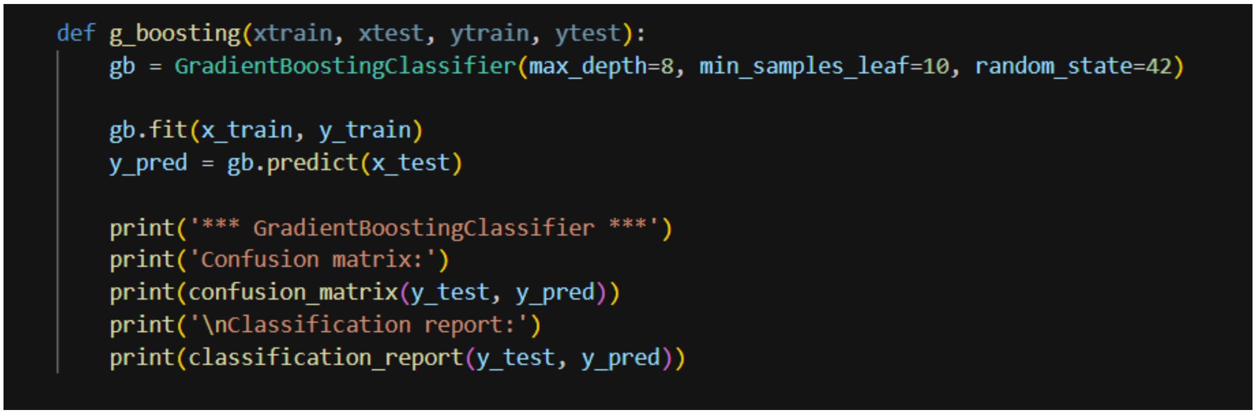
A function named random_forest is created and train and test data are passed as the parameters. Inside the function, RandomForestClassifier() algorithm is initialized and training data is passed to the model with the .fit() function. Test data is predicted with .predict() function and saved in a new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



```
def random_forest(xtrain, xtest, ytrain, ytest):
    rf = RandomForestClassifier(n_estimators=200, max_depth=10, class_weight='balanced', random_state=42)

    rf.fit(x_train, y_train)
    y_pred = rf.predict(x_test)

    print('*** RandomForestClassifier ***')
    print('Confusion Matrix:')
    print(confusion_matrix(y_test, y_pred))
    print('\nClassification Report:')
    print(classification_report(y_test, y_pred))
```

Activity 1.3: Xgboost Model

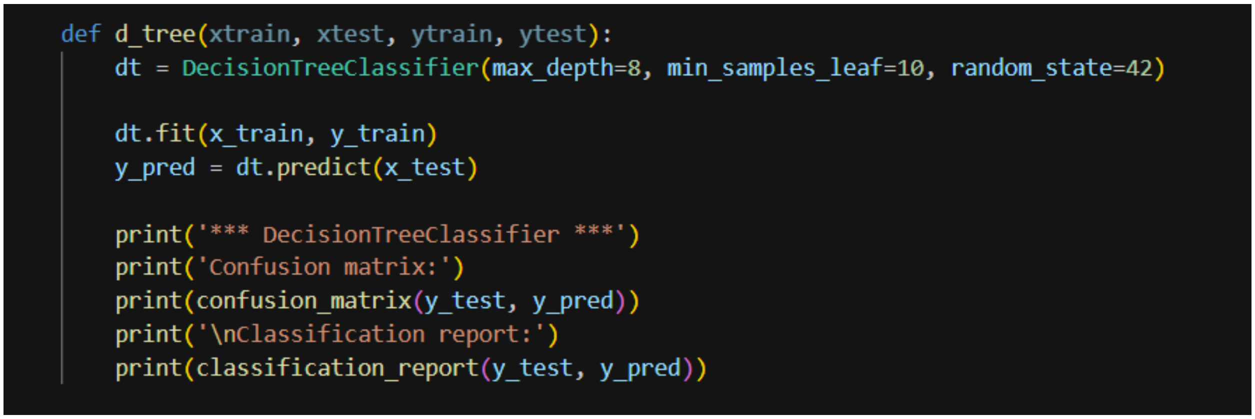
A function named g_boosting is created and train and test data are passed as the parameters. Inside the function, GradientBoostingClassifier() algorithm is initialized and training data is passed to the model with the .fit() function. Test data is predicted with the .predict() function and saved in a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



```
def g_boosting(xtrain, xtest, ytrain, ytest):
    gb = GradientBoostingClassifier(max_depth=8, min_samples_leaf=10, random_state=42)

    gb.fit(x_train, y_train)
    y_pred = gb.predict(x_test)

    print('*** GradientBoostingClassifier ***')
    print('Confusion matrix:')
    print(confusion_matrix(y_test, y_pred))
    print('\nClassification report:')
    print(classification_report(y_test, y_pred))
```

Activity 1.4: Decision tree model

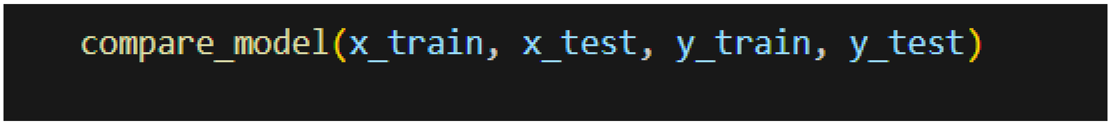
A function named d_tree is created and train and test data are passed as the parameters. Inside the function, DecisionTreeClassifier() algorithm is initialized and training data is passed to the model with the .fit() function. Test data is predicted with the .predict() function and saved in a new variable.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



```
def d_tree(xtrain, xtest, ytrain, ytest):
    dt = DecisionTreeClassifier(max_depth=8, min_samples_leaf=10, random_state=42)

    dt.fit(x_train, y_train)
    y_pred = dt.predict(x_test)

    print('*** DecisionTreeClassifier ***')
    print('Confusion matrix:')
    print(confusion_matrix(y_test, y_pred))
    print('\nClassification report:')
    print(classification_report(y_test, y_pred))
```

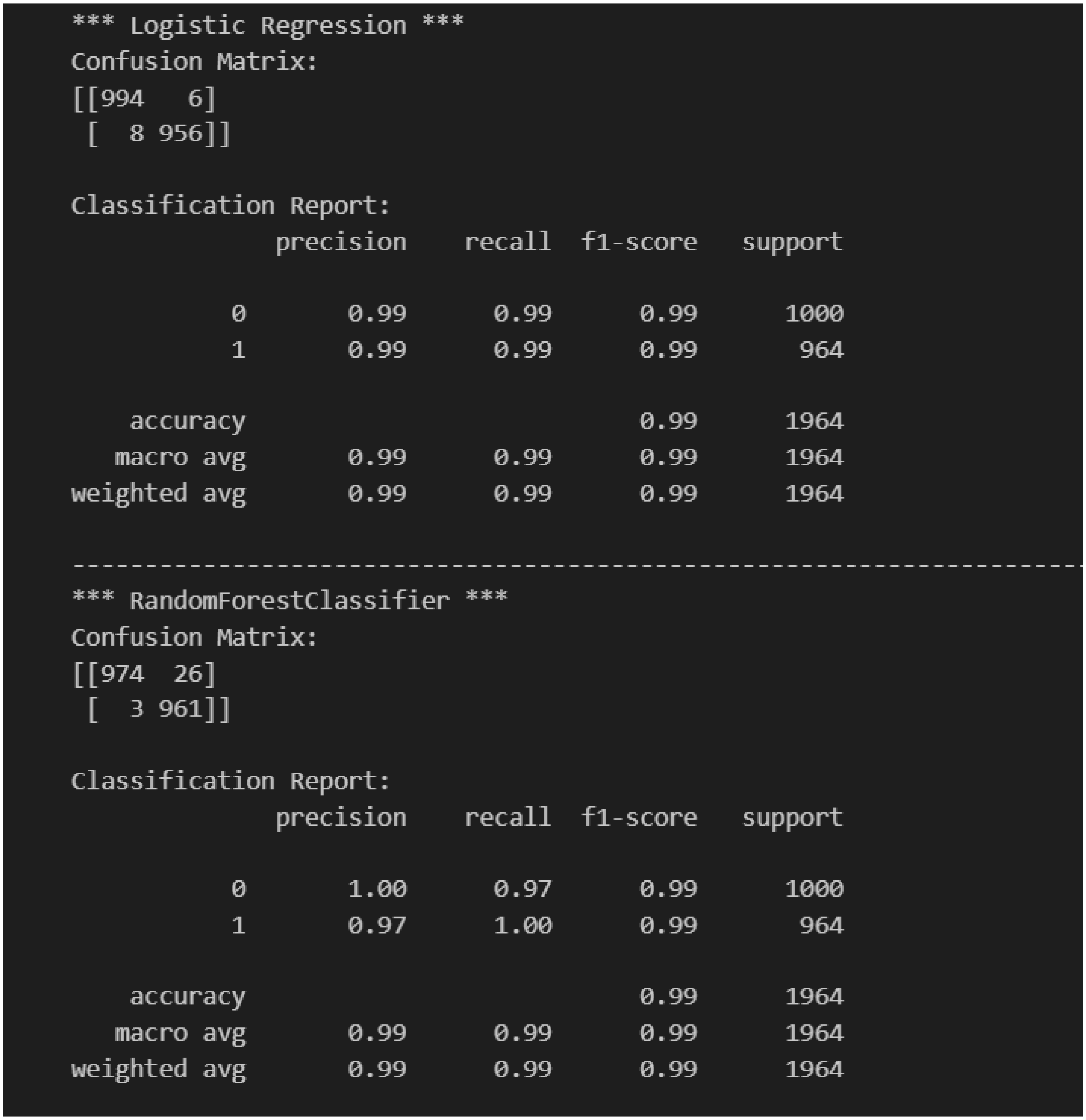
Activity 2: Comparing the models

For comparing the above four models, the compareModel function is defined.

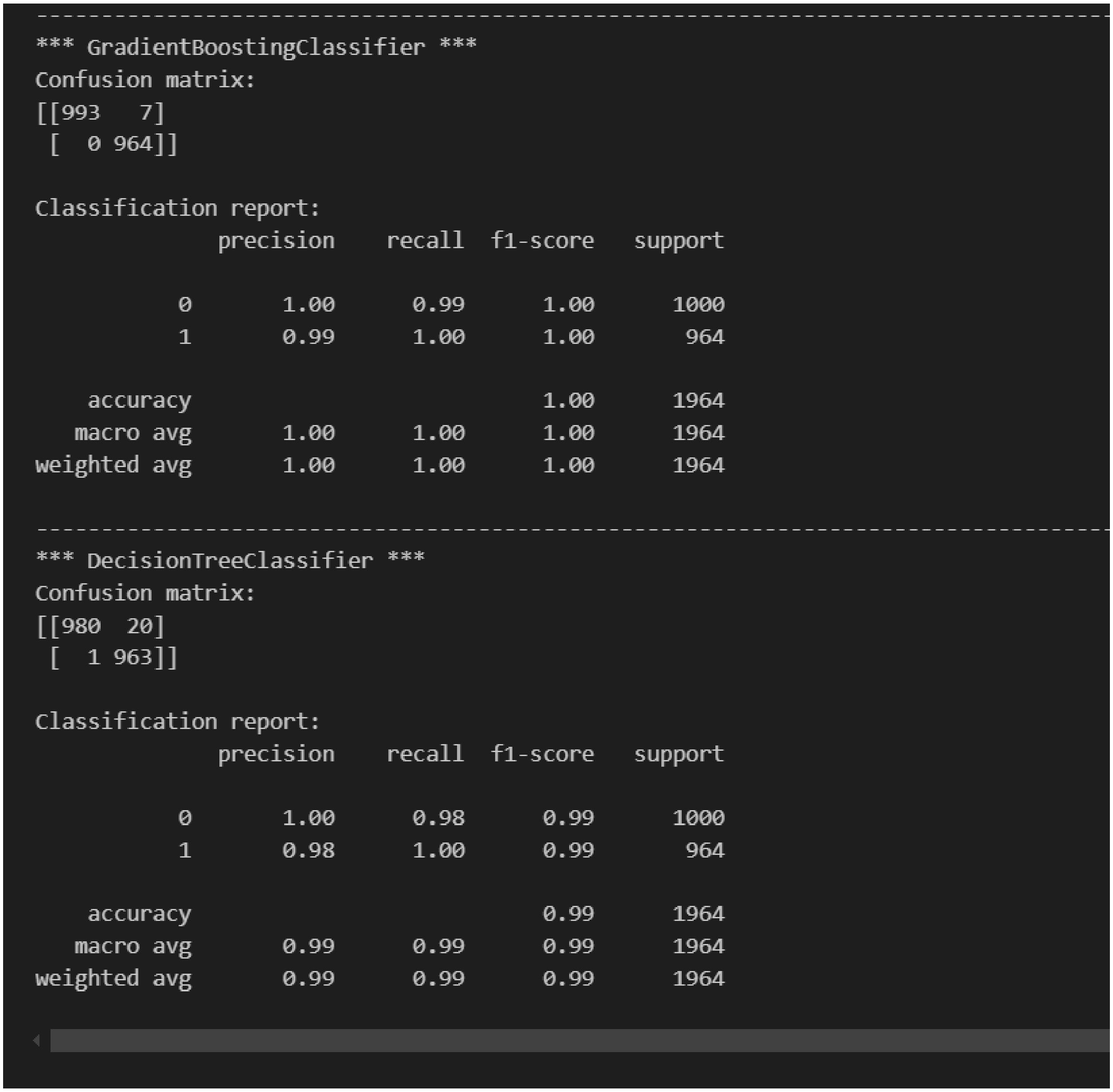


```
compare_model(x_train, x_test, y_train, y_test)
```

After calling the function, the results of models are displayed below as an output. From these four models we found the decision tree model performs well.



```
*** Logistic Regression ***
Confusion Matrix:
[[994   6]
 [  8 956]]

Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.99      0.99      1000
           1       0.99      0.99      0.99       964

    accuracy                           0.99      1964
   macro avg       0.99      0.99      0.99      1964
weighted avg       0.99      0.99      0.99      1964

----------------------------------------------------------------------
*** RandomForestClassifier ***
Confusion Matrix:
[[974  26]
 [  3 961]]

Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.97      0.99      1000
           1       0.97      1.00      0.99       964

    accuracy                           0.99      1964
   macro avg       0.99      0.99      0.99      1964
weighted avg       0.99      0.99      0.99      1964
```



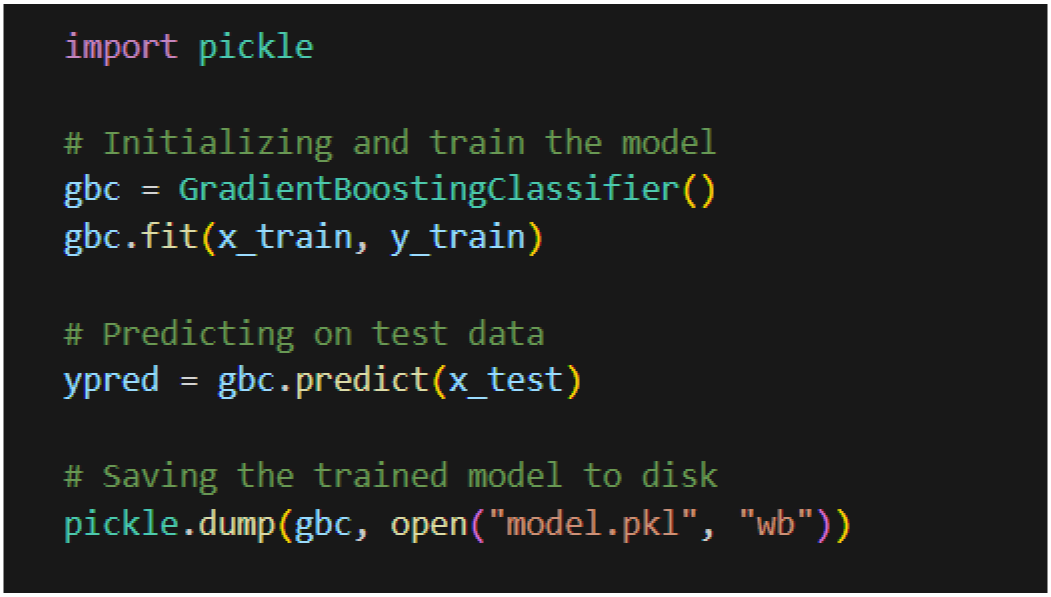
```
-----------------------------------------------------------------------------------
*** GradientBoostingClassifier ***
Confusion matrix:
[[993   7]
 [  0 964]]

Classification report:
              precision    recall  f1-score   support

           0       1.00      0.99      1.00      1000
           1       0.99      1.00      1.00       964

    accuracy                           1.00      1964
   macro avg       1.00      1.00      1.00      1964
weighted avg       1.00      1.00      1.00      1964

-----------------------------------------------------------------------------------
*** DecisionTreeClassifier ***
Confusion matrix:
[[980  20]
 [  1 963]]

Classification report:
              precision    recall  f1-score   support

           0       1.00      0.98      0.99      1000
           1       0.98      1.00      0.99       964

    accuracy                           0.99      1964
   macro avg       0.99      0.99      0.99      1964
weighted avg       0.99      0.99      0.99      1964
```

Finally, we chose the Gradient Boosting Classifier, which gave the best accuracy and F1 score. So, this model is saved and used on flask integration.

Milestone 6: Model Deployment

Activity 1: Saving the best model

Saving the best model after comparing its performance using different evaluation metrics i.e. 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.



```python
import pickle

# Initializing and train the model
gbc = GradientBoostingClassifier()
gbc.fit(x_train, y_train)

# Predicting on test data
ypred = gbc.predict(x_test)

# Saving the trained model to disk
pickle.dump(gbc, open("model.pkl", "wb"))
```

The Xgboost Model is saved by the pickle.dump() function. It saves the model as a .pkl file.

Activity 2: Integrating with Web Framework

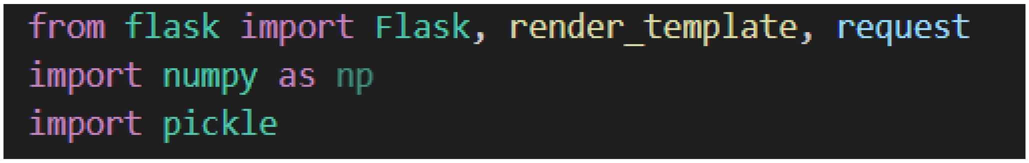
In this section, we will be building a web application that is integrated to the model we built. A UI is provided for the uses where he has to enter the values for predictions. The enter values are given to the saved model and prediction is showcased on the UI. This section has the following tasks Building HTML Pages Building server-side script Run the web application

Activity 2.1: Building Html Page

For this project create HTML files namely index.html and save them in the templates folder.

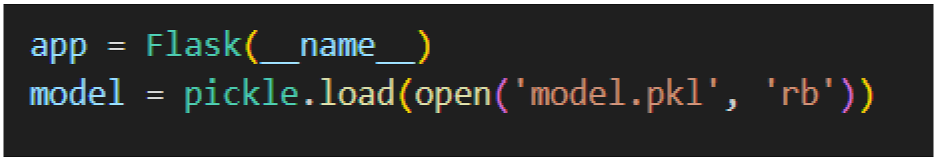
Activity 2.2: Build Python code

Import the libraries



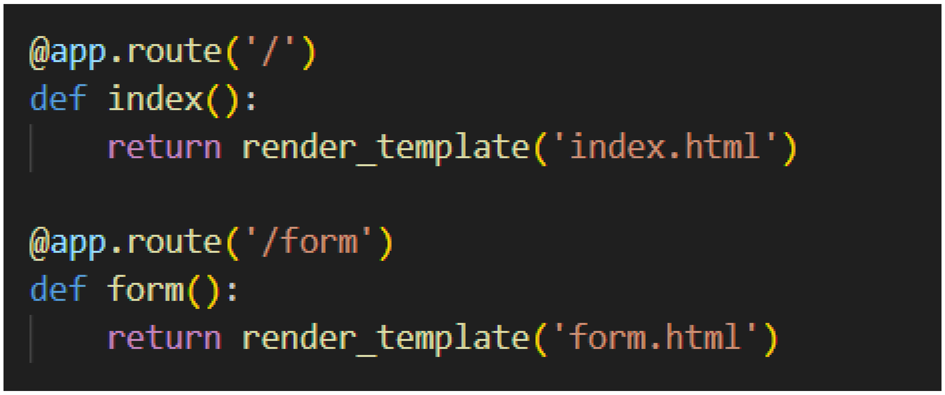
```python
from flask import Flask, render_template, request
import numpy as np
import pickle
```

Load the saved model. Importing the flask module in the project is mandatory.



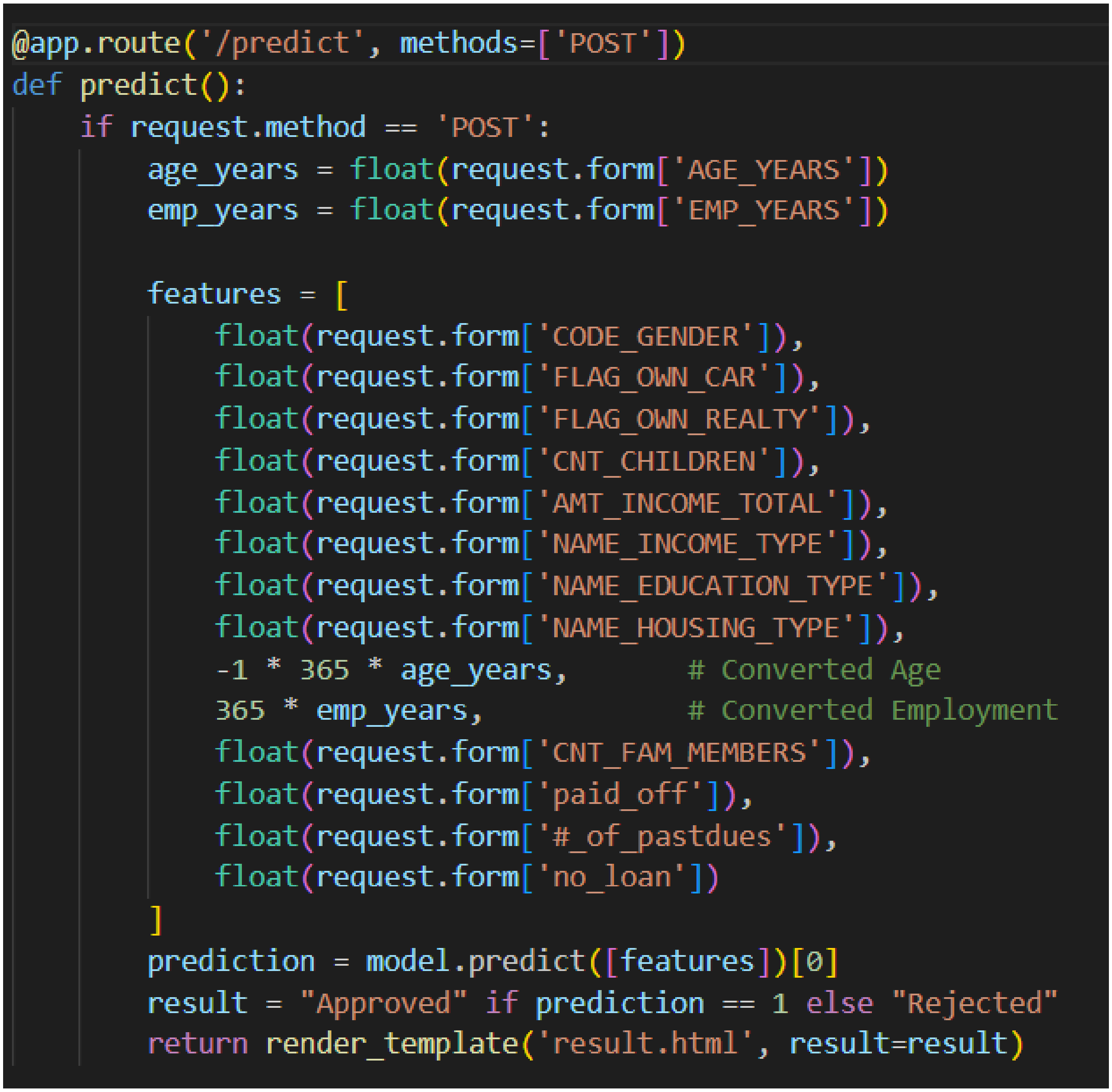
```python
app = Flask(__name__)
model = pickle.load(open('model.pkl', 'rb'))
```

Render HTML page:



```python
@app.route('/')
def index():
    return render_template('index.html')

@app.route('/form')
def form():
    return render_template('form.html')
```

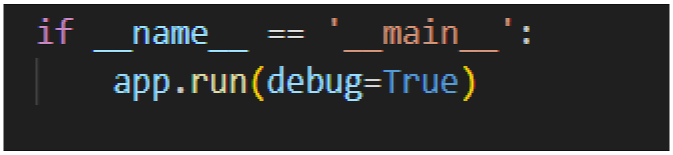
Here we will be using a declared constructor to route to the HTML page which we have created earlier. In the above example, '/ URL is bound with the index.html function. Hence, when the home page of the web server is opened in the browser, the html page will be rendered. Whenever you enter the values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



```python
@app.route('/predict', methods=['POST'])
def predict():
    if request.method == 'POST':
        age_years = float(request.form['AGE_YEARS'])
        emp_years = float(request.form['EMP_YEARS'])

        features = [
            float(request.form['CODE_GENDER']),
            float(request.form['FLAG_OWN_CAR']),
            float(request.form['FLAG_OWN_REALTY']),
            float(request.form['CNT_CHILDREN']),
            float(request.form['AMT_INCOME_TOTAL']),
            float(request.form['NAME_INCOME_TYPE']),
            float(request.form['NAME_EDUCATION_TYPE']),
            float(request.form['NAME_HOUSING_TYPE']),
            -1 * 365 * age_years,       # Converted Age
            365 * emp_years,            # Converted Employment
            float(request.form['CNT_FAM_MEMBERS']),
            float(request.form['paid_off']),
            float(request.form['#_of_pastdues']),
            float(request.form['no_loan'])
        ]
        prediction = model.predict([features])[0]
        result = "Approved" if prediction == 1 else "Rejected"
        return render_template('result.html', result=result)
```

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.

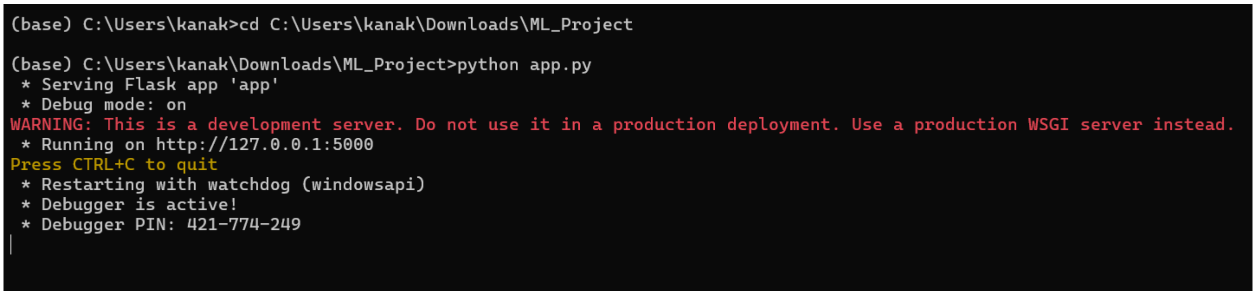
Main Function:



```python
if __name__ == '__main__':
    app.run(debug=True)
```

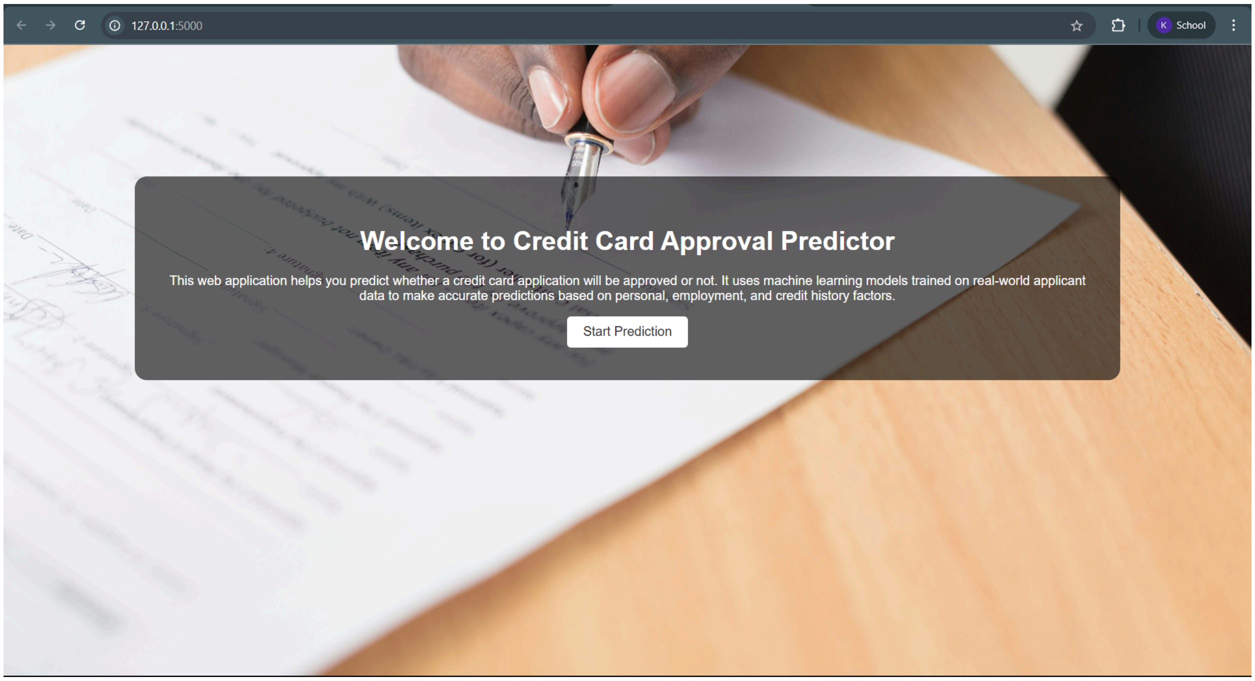
Activity 2.3: Run the web application

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.



```
(base) C:\Users\kanak>cd C:\Users\kanak\Downloads\ML_Project

(base) C:\Users\kanak\Downloads\ML_Project>python app.py
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with watchdog (windowsapi)
 * Debugger is active!
 * Debugger PIN: 421-774-249
```

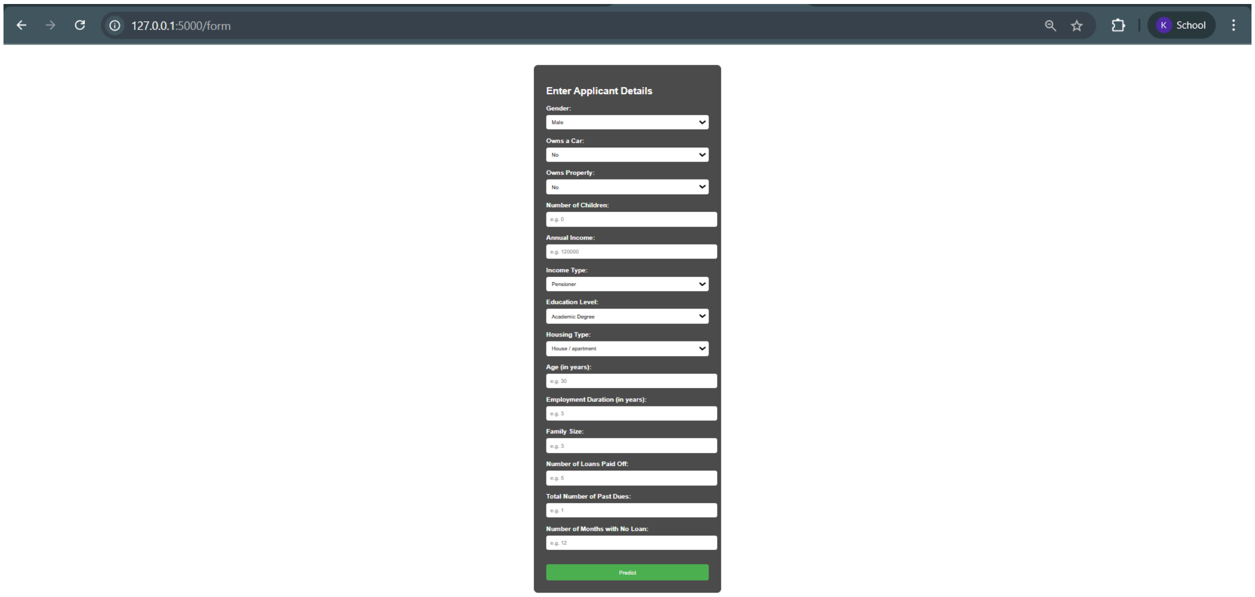
Now, Go the web browser and write the localhost url (http://127.0.0.1:5000) to get the below result.



Welcome to Credit Card Approval Predictor

This web application helps you predict whether a credit card application will be approved or not. It uses machine learning models trained on real-world applicant data to make accurate predictions based on personal, employment, and credit history factors.

Click on “Start Prediction” .



Enter Applicant Details

Gender: Male

Owns a Car: No

Owns Property: No

Number of Children: e.g. 0

Annual Income: e.g. 120000

Income Type: Pensioner

Education Level: Academic Degree

Housing Type: House / apartment

Age (in years): e.g. 30

Employment Duration (in years): e.g. 3

Family Size: e.g. 3

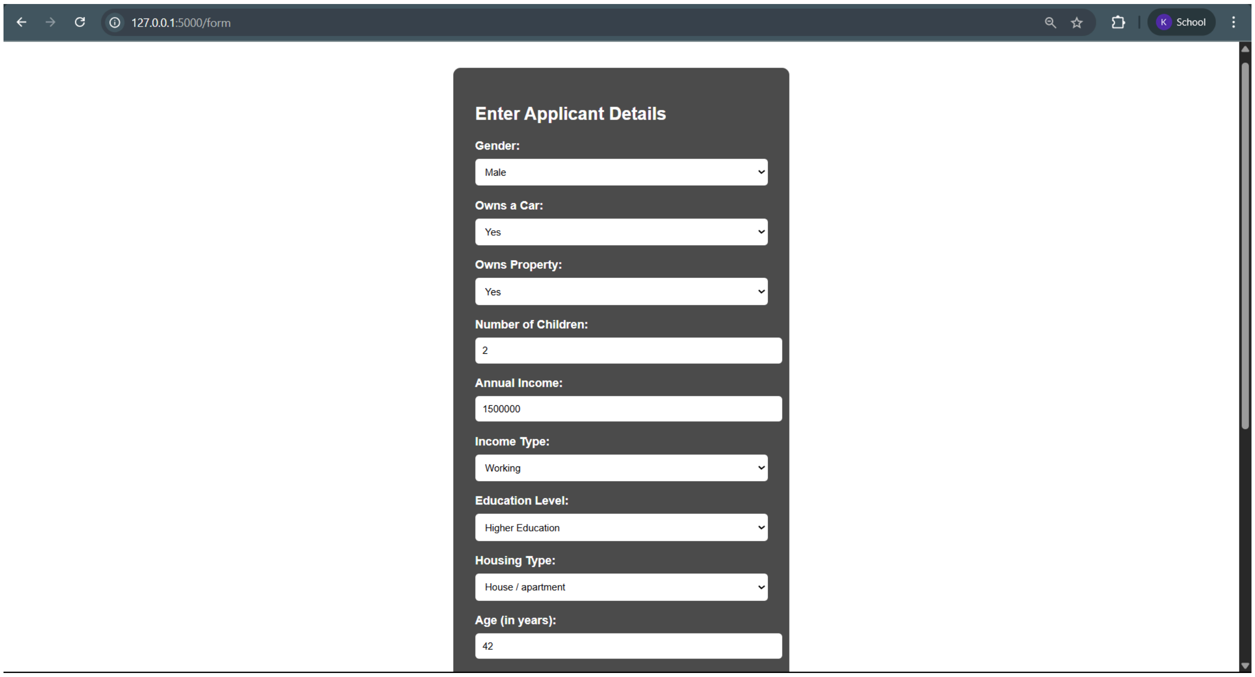
Number of Loans Paid Off: e.g. 5

Total Number of Past Dues: e.g. 1

Number of Months with No Loan: e.g. 12

Predict

The above form page will open up. Now enter the values.



Enter Applicant Details

Gender: Male

Owns a Car: Yes

Owns Property: Yes

Number of Children: 2

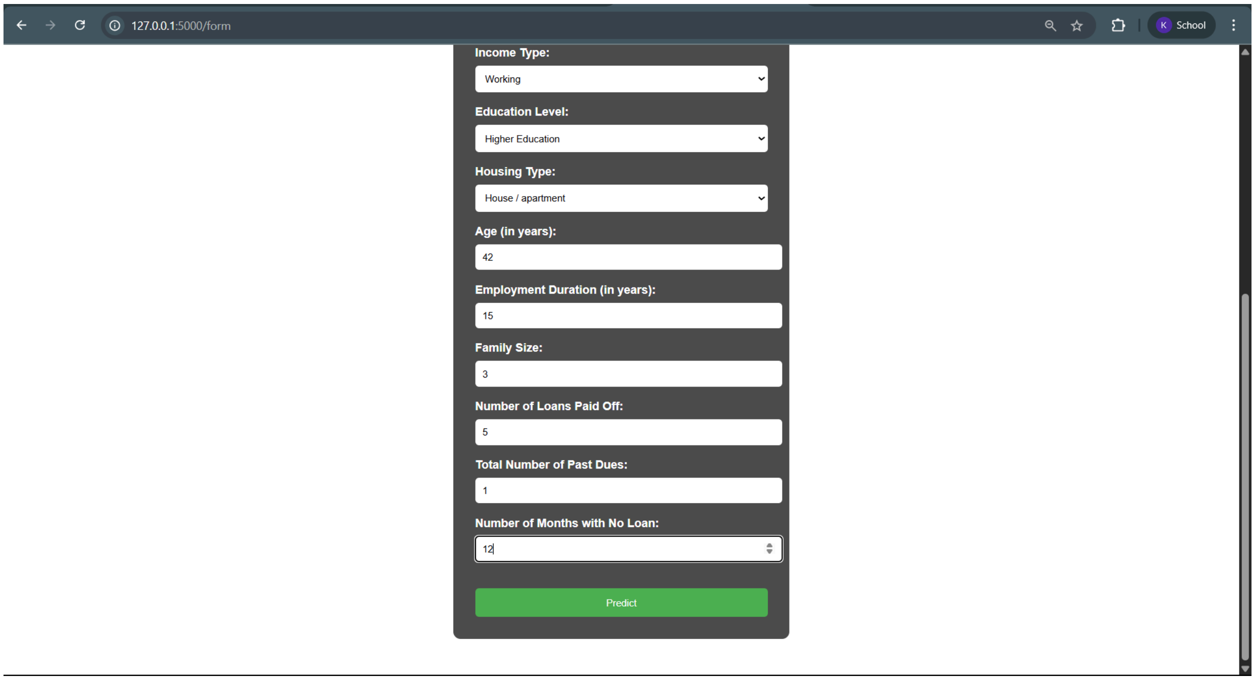
Annual Income: 1500000

Income Type: Working

Education Level: Higher Education

Housing Type: House / apartment

Age (in years): 42



Income Type: Working

Education Level: Higher Education

Housing Type: House / apartment

Age (in years): 42

Employment Duration (in years): 15

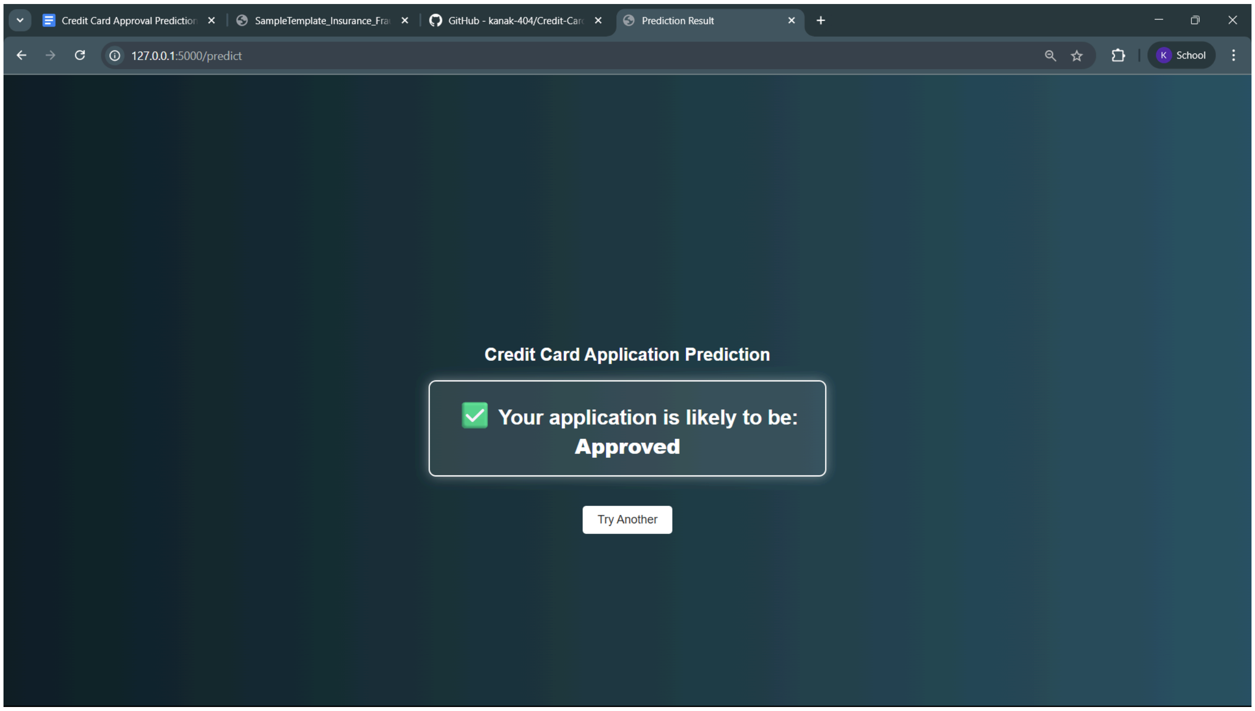
Family Size: 3

Number of Loans Paid Off: 5

Total Number of Past Dues: 1

Number of Months with No Loan: 12

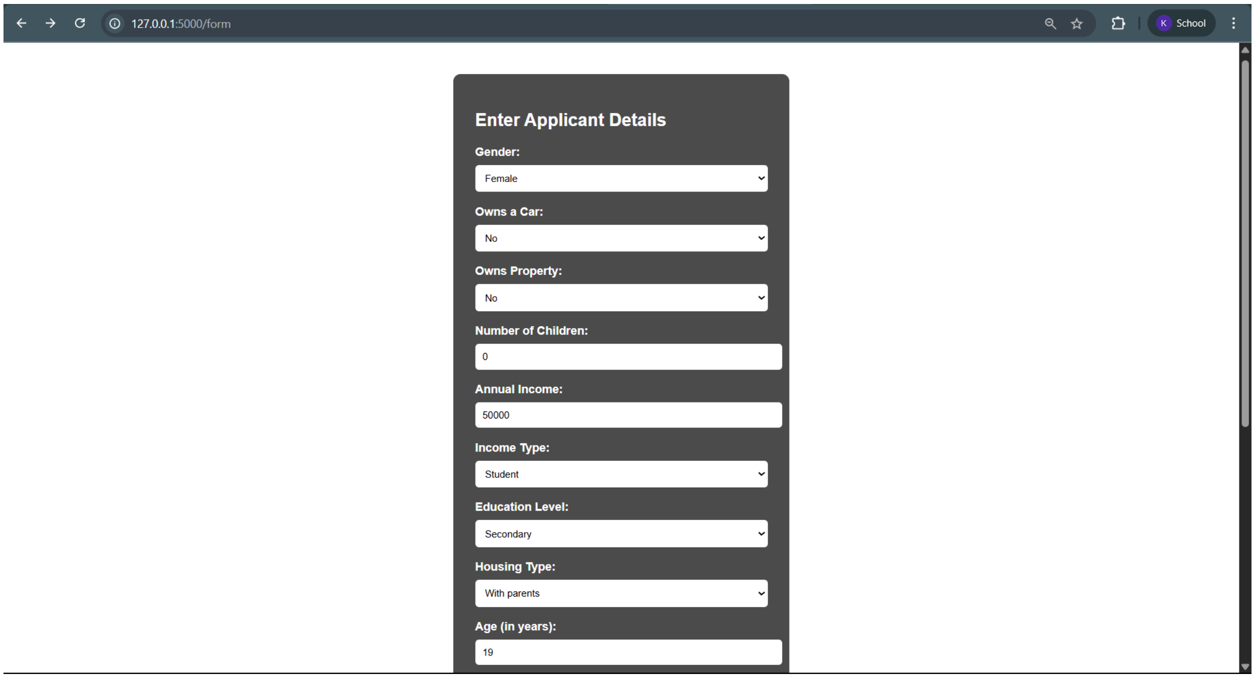
Click on the “Predict” button to get the prediction result.


Credit Card Approval Prediction
SampleTemplate_Insurance_Fra
GitHub - kanak-404/Credit-Car
Prediction Result
127.0.0.1:5000/predict
School

Credit Card Application Prediction

✅ Your application is likely to be: **Approved**

Here,for the given values, the result is “Approved”.
We can try another prediction with different values by clicking on “Try Another”.



Enter Applicant Details

Gender: Female

Owns a Car: No

Owns Property: No

Number of Children: 0

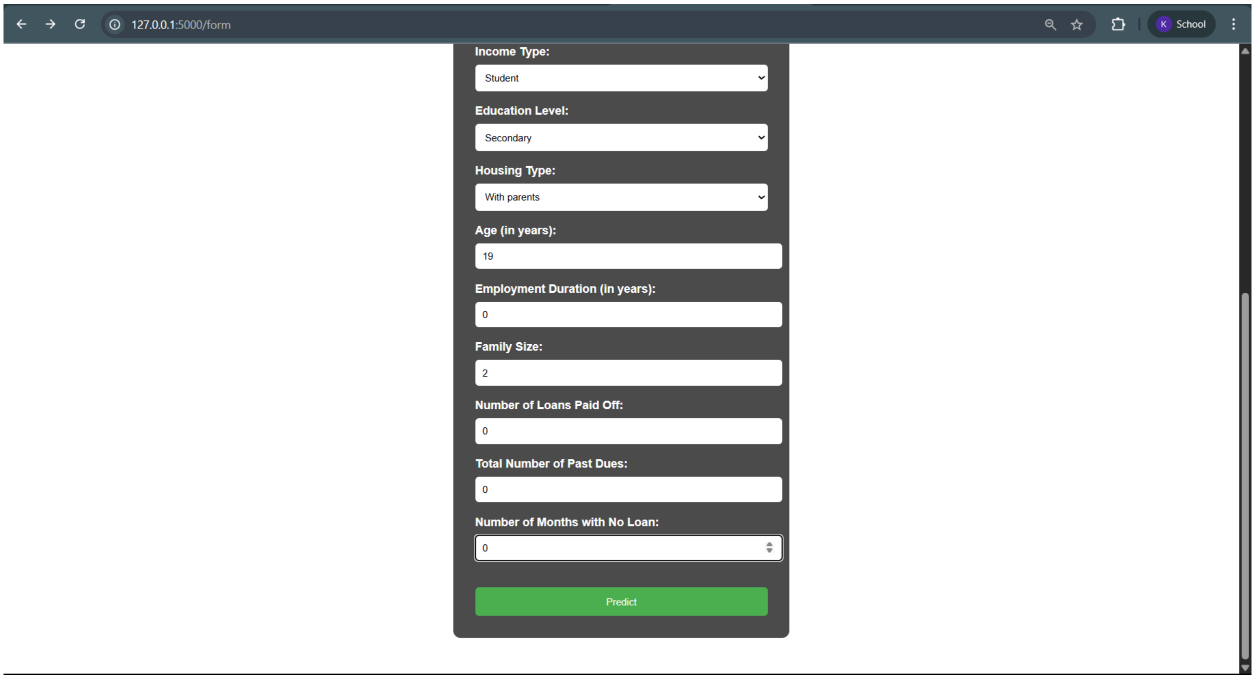
Annual Income: 50000

Income Type: Student

Education Level: Secondary

Housing Type: With parents

Age (in years): 19



Income Type: Student

Education Level: Secondary

Housing Type: With parents

Age (in years): 19

Employment Duration (in years): 0

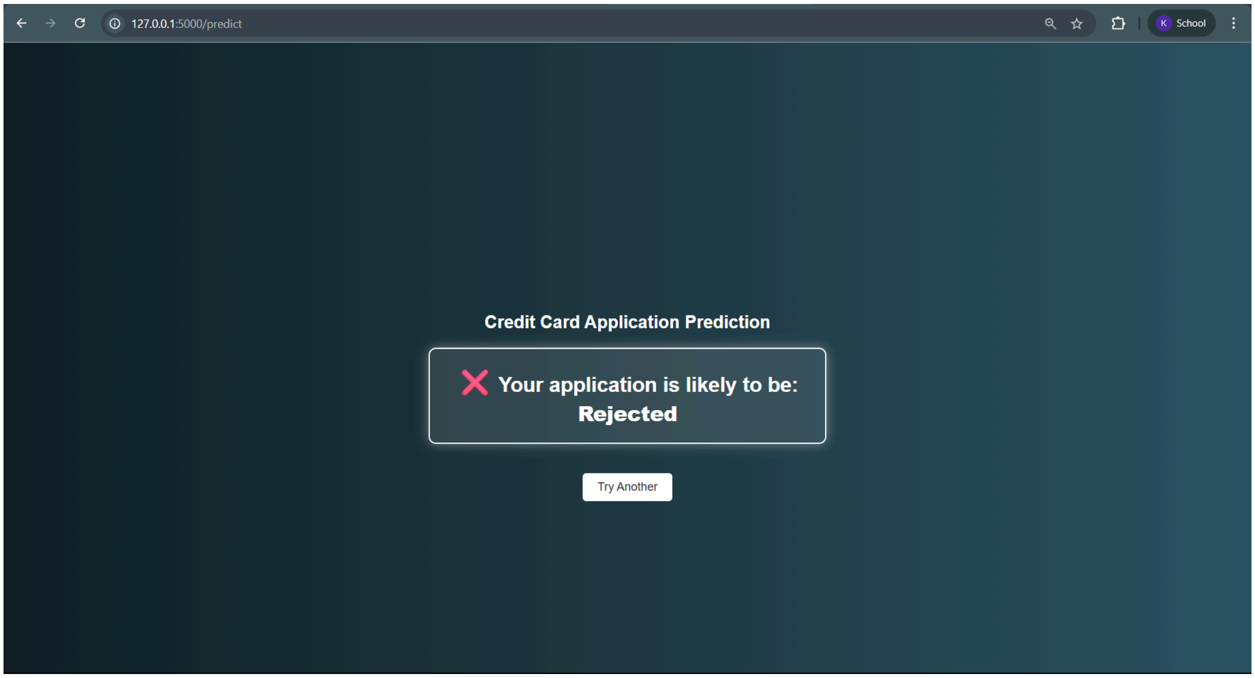
Family Size: 2

Number of Loans Paid Off: 0

Total Number of Past Dues: 0

Number of Months with No Loan: 0

Click on the “Predict” button to get the prediction result.



Credit Card Application Prediction

❌ Your application is likely to be: **Rejected**

Now, the result is “Not Approved”.