

CREDIT CARD APPROVAL SYSTEM TERM REPORT

by

**Pratyush Priyam
&
Thota Rahul**

Section: KM118

Roll Numbers: RKM118B35 & RKM118A10



L OVELY
P ROFESSIONAL
U NIVERSITY


**Department of Intelligent Systems,
School of Computer Science Engineering,
Lovely Professional University, Jalandhar
November 2022**

STUDENT DECLARATION

This is to declare that this report has been written by us. No part of the report is copied from other sources. All information included from other sources have been duly acknowledged. We aver that if any part of the report is found to be copied, we shall take full responsibility for it.



Pratyush Priyam
RKM118B35



Thota Rahul
RKM118A10

Place : Lovely Professional University, Jalandhar
Date: 08th November 2022

TABLE OF CONTENTS

TITLE	PAGE NO.
BONAFIDE CERTIFICATE	4
INTRODUCTION	5
SYNTAX AND SCREENSHOTS	6
ALGORITHM USED	9
CONCLUSION	10
BIBLIOGRAPHY	11

BONAFIDE CERTIFICATE

Certified that this project report “Credit Card Approval System using machine learning” is the Bonafede work of Pratyush Priyam and Thota Rahul who carried out the project work under my supervision.

Dr. Dhanpratap Singh
Associate professor
25706
Intelligence System 1

INTRODUCTION

In this project, we will try to make a Credit Card Approval System using Machine Learning via python.

The correct assessment for credit card approval is very important for banks and organisations who lend a credit card to the people. The recent years have seen a huge growth in credit cards and loans. The exact judgement of person to be approved for credit cards allows the organisations to minimize losses and the same time make suitable credit arrangements as per requirement. Due to the huge growth in the number of applicants, there is a need for a more sophisticated method to automate the process and speed it up.

Credit card approval can be beneficial for organisations that lend credit cards, and due to increase in a huge number of the applicant, there is need to automate the task and classify the applicants into if they are eligible for a credit card or not. This helps to avoid organisation losses by avoiding potential defaulters. Here we are not just looking into bank balance but into their personal attributes like gender, married, age, Occupation etc. This can also help cut down the weekslong process into few days. This gives benefit by cutting down costs on credit analysis and faster credit decisions.

```

In [8]: # 1. Loading Dataset
import pandas as pd
df = pd.read_csv('crx.data', sep='\\s+', header=None)
display(df)

0      b 30 83 0 u g w v 1 25 11 01 f g 00202 0 +
1      a 58 67 4 46 u g q h 3 04 11 06 f g 00043 560 +
2      a 24 50 0 5 u g q h 1 5 11 01 f g 00280 824 +
3      b 27 83 1 54 u g w v 3 75 11 05 f g 00100 3 +
4      b 20 17 5 625 u g w v 1 71 11 01 f s 00120 0 +
...
685     b 21 06 10 085 y p e h 1 25 11 01 f g 00260 0 -
686     a 22 67 0 75 u g c v 2 11 02 f g 00200 394 -
687     a 25 25 13 5 y p ff 2 11 01 f g 00200 1 -
688     b 17 92 0 205 u g aa v 0 04 11 01 f g 00280 750 -
689     b 35 00 3 375 u g c h 8 26 11 01 f g 00000 0 -

690 rows x 1 columns

In [9]: # Described specific row data
headerRow = ['Gender', 'Age', 'Debt', 'Married', 'Bank Customer', 'Education', 'Ethnicity', 'Years Employed', 'Prior Default', 'Employed', 'Credit Score', 'Driving License', 'Citizenship', 'Zip Code', 'Income', 'Approved']
df = pd.read_csv('crx.data', names = headerRow)
display(df)

```

Describing Data

```

In [9]: # Described specific row data
headerRow = ['Gender', 'Age', 'Debt', 'Married', 'Bank Customer', 'Education', 'Ethnicity', 'Years Employed', 'Prior Default', 'Employed', 'Credit Score', 'Driving License', 'Citizenship', 'Zip Code', 'Income', 'Approved']
df = pd.read_csv('crx.data', names = headerRow)
display(df)

+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| Gender | Age | Debt | Married | Bank Customer | Education | Ethnicity | Years Employed | Prior Default | Employed | Credit Score | Driving License | Citizenship | Zip Code | Income | Approved |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 0      | b   | 30 83 | u       | g             | w         | v         | 1.25           | t             | t        | 1            | f           | g          | 00202    | 0       | +         |
| 1      | a   | 58 67 | 4 46    | u             | g         | q         | 3.04           | t             | t        | 6            | f           | g          | 00043    | 560     | +         |
| 2      | a   | 24 50 | 0 500   | u             | g         | q         | 1.50           | t             | f        | 0            | f           | g          | 00280    | 824     | +         |
| 3      | b   | 27 83 | 1 54    | u             | g         | w         | 3.75           | t             | t        | 5            | t           | s          | 00100    | 3       | +         |
| 4      | b   | 20 17 | 5 625   | u             | g         | w         | 1.71           | t             | f        | 0            | f           | s          | 00120    | 0       | +         |
| ...    | ... | ...   | ...     | ...           | ...       | ...       | ...            | ...           | ...     | ...          | ...         | ...       | ...      | ...     | ...       |
| 685    | b   | 21 06 | 10 085  | y             | p         | e         | 1.25           | f             | f        | 0            | f           | g          | 00260    | 0       | -         |
| 686    | a   | 22 67 | 0 750   | u             | g         | c         | 2.00           | f             | t        | 2            | t           | g          | 00200    | 394     | -         |
| 687    | a   | 25 25 | 13 500  | y             | p         | ff        | 2.00           | f             | t        | 1            | t           | g          | 00200    | 1       | -         |
| 688    | b   | 17 92 | 0 205   | u             | g         | aa        | 0.04           | f             | f        | 0            | f           | g          | 00280    | 750     | -         |
| 689    | b   | 35 00 | 3 375   | u             | g         | c         | 8.29           | f             | f        | 0            | t           | g          | 00000    | 0       | -         |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+

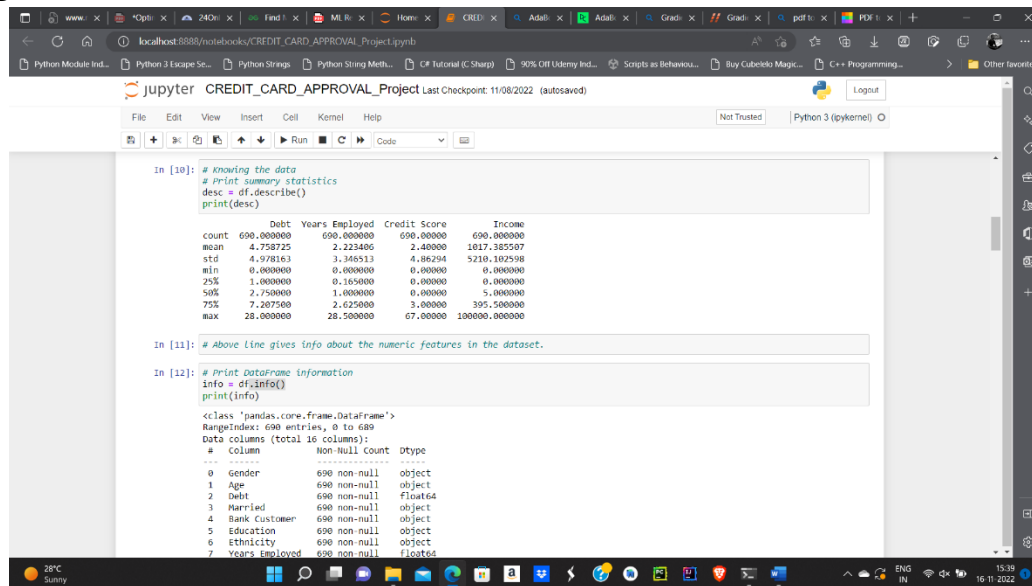
690 rows x 16 columns

In [10]: # Knowing the data
# Print summary statistics
desc = df.describe()
print(desc)

Debt    Years Employed    Credit Score    Income

```

Knowing Data



The screenshot shows a Jupyter Notebook interface with the following code and output:

```
In [10]: # knowing the data
# Print summary statistics
desc = df.describe()
print(desc)
```

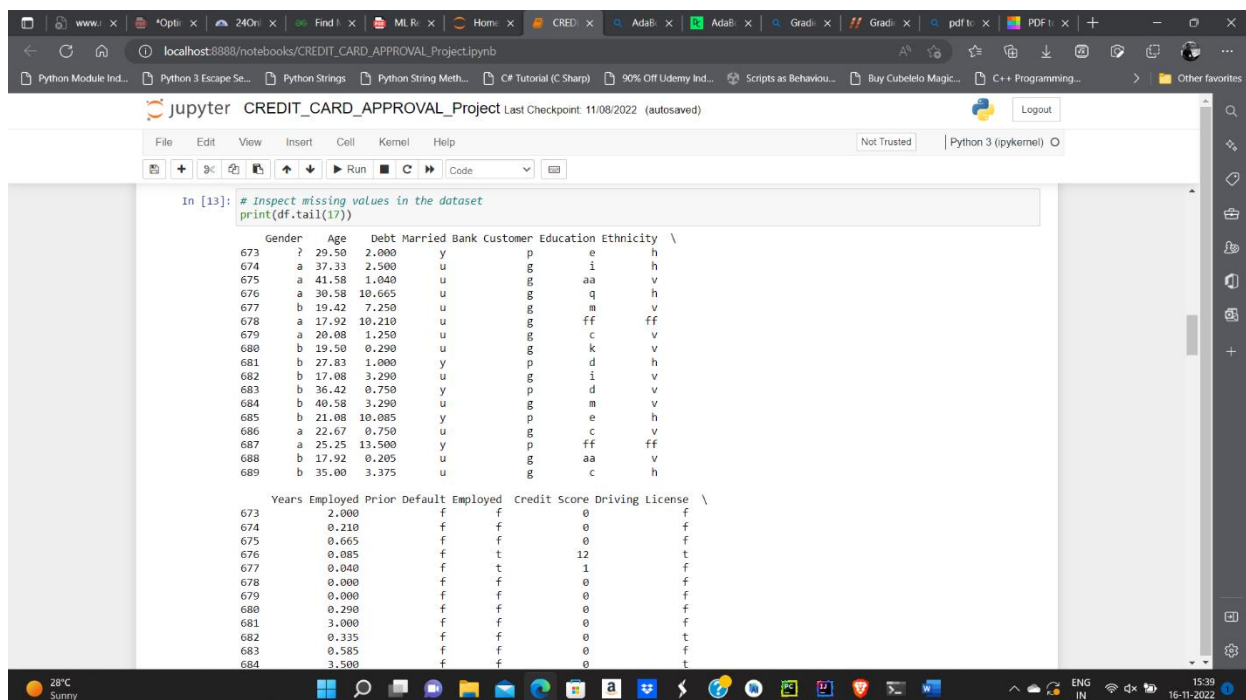
	Debt	Years Employed	Credit Score	Income
count	600.000000	600.000000	600.000000	600.000000
mean	4.758725	2.223406	2.400000	1017.382507
std	4.978163	3.346513	4.86294	5210.102598
min	0.000000	0.000000	0.000000	0.000000
25%	1.000000	0.165000	0.000000	0.000000
50%	2.750000	1.000000	0.000000	5.000000
75%	7.207500	2.625000	3.000000	395.500000
max	26.000000	26.500000	67.000000	100000.000000

```
In [11]: # Above line gives info about the numeric features in the dataset.

In [12]: # Print DataFrame information
info = df.info()
print(info)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 600 entries, 0 to 599
Data columns (total 16 columns):
 #   Column              Non-Null Count  Dtype
---  -
 0   Gender              600 non-null   object
 1   Age                 600 non-null   object
 2   Debt                600 non-null   float64
 3   Married             600 non-null   object
 4   Bank Customer       600 non-null   object
 5   Education           600 non-null   object
 6   Ethnicity           600 non-null   object
 7   Years Employed      600 non-null   float64
```

Inspecting data and finding null values



The screenshot shows a Jupyter Notebook interface with the following code and output:

```
In [13]: # Inspect missing values in the dataset
print(df.tail(17))
```

```
Gender      Age      Debt  Married  Bank Customer  Education  Ethnicity  \
673  ?  29.50  2.000      y      p      e      h
674  a  37.33  2.500      u      g      i      h
675  a  41.58  1.040      u      g      aa     v
676  a  30.58  10.665     u      g      q      h
677  b  19.42  7.250      u      g      m      v
678  a  17.92  10.210     u      g      ff     ff
679  a  20.08  1.250      u      g      c      v
680  b  19.50  0.290      u      g      k      v
681  b  27.83  1.000      y      p      d      h
682  b  17.08  3.290      u      g      i      v
683  b  36.42  0.750      y      p      d      v
684  b  40.58  3.290      u      g      m      v
685  b  21.08  10.085     y      p      e      h
686  a  22.67  0.750      u      g      c      v
687  a  25.25  13.500     y      p      ff     ff
688  b  17.92  0.205      u      g      aa     v
689  b  35.00  3.375      u      g      c      h

Years Employed  Prior Default  Employed  Credit Score  Driving License  \
673      2.000      f      f      0      f
674      0.210      f      f      0      f
675      0.665      f      f      0      f
676      0.085      f      t      12     t
677      0.040      f      t      1      f
678      0.000      f      f      0      f
679      0.000      f      f      0      f
680      0.290      f      f      0      f
681      3.000      f      f      0      f
682      0.335      f      f      0      t
683      0.585      f      f      0      f
684      3.500      f      f      0      t
```

localhost:8888/notebooks/CREDIT_CARD_APPROVAL_ProjectIpyb

jupyter CREDIT_CARD_APPROVAL_Project Last Checkpoint: 11/08/2022 (autosaved)

File Edit View Insert Cell Kernel Help Not Trusted Python 3 (pykernel)

```
681 3.000 f f 0 t
682 0.335 f f 0 t
683 0.585 f f 0 f
684 3.500 f f 0 t
685 1.250 f f 0 f
686 2.000 f t 2 t
687 2.000 f t 1 t
688 0.040 f f 0 f
689 8.290 f f 0 t
```

	Citizenship	Zip Code	Income	Approved
673	g	00256	17	-
674	g	00260	246	-
675	g	00240	237	-
676	g	00129	3	-
677	g	00100	1	-
678	g	00000	50	-
679	g	00000	0	-
680	g	00280	364	-
681	g	00176	537	-
682	g	00140	2	-
683	g	00240	3	-
684	s	00400	0	-
685	g	00260	0	-
686	g	00200	394	-
687	g	00200	1	-
688	g	00280	750	-
689	g	00000	0	-

In [14]: # Handling the null Values

```
import numpy as np
# Replace the '?'s with NaN
df = df.replace('?', np.nan)
```

28°C Sunny 15:40 16-11-2022

Handling Missing Values

localhost:8888/notebooks/CREDIT_CARD_APPROVAL_ProjectIpyb

jupyter CREDIT_CARD_APPROVAL_Project Last Checkpoint: 11/08/2022 (autosaved)

File Edit View Insert Cell Kernel Help Not Trusted Python 3 (pykernel)

```
In [16]: def handleMissingNumeric(df, colNames):
for col in colNames:
    df[col] = pd.to_numeric(df[col], errors='coerce')
    df[col] = df[col].fillna(df[col].mean())
def filterDf(df, colNames):
for cols in colNames:
    d = {}
    for i in df[cols]:
        if i not in d:
            d[i] = len(d)
    df[cols] = df[cols].map(d)
handleMissingNumeric(df, ['Age', 'Debt', 'Years Employed', 'Credit Score', 'Zip Code', 'Income'])
filterDf(df, ['Gender', 'Married', 'Bank Customer', 'Education', 'Ethnicity', 'Prior Default', 'Employed', 'Driving License', 'C
```

```
In [17]: import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score as score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import normalize as normalizeSk

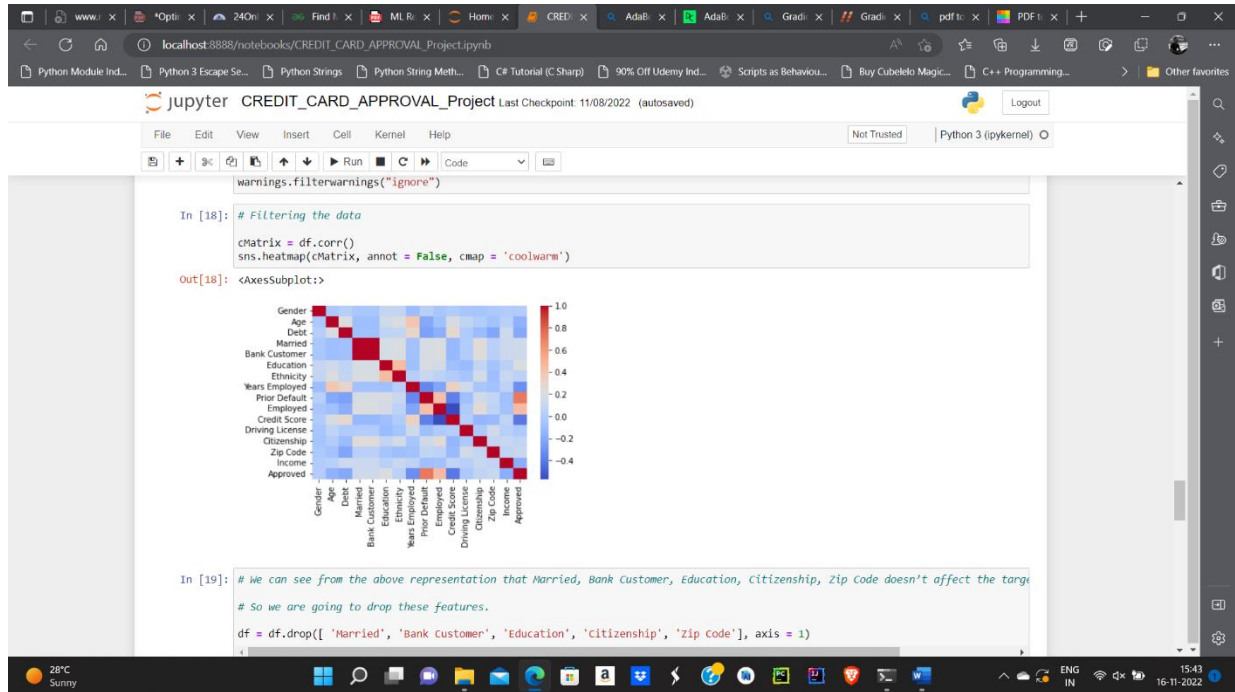
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
```

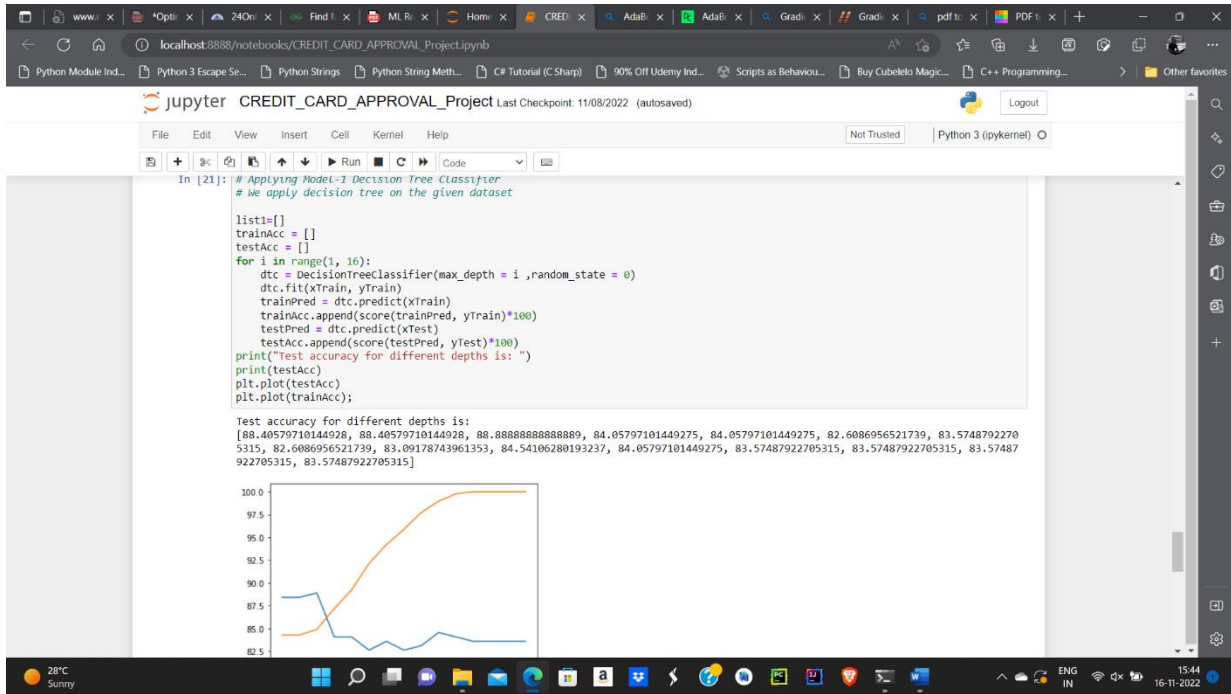
In [18]: # Filtering the data

```
cMatrix = df.corr()
sns.heatmap(cMatrix, annot=True, cmap='coolwarm')
```

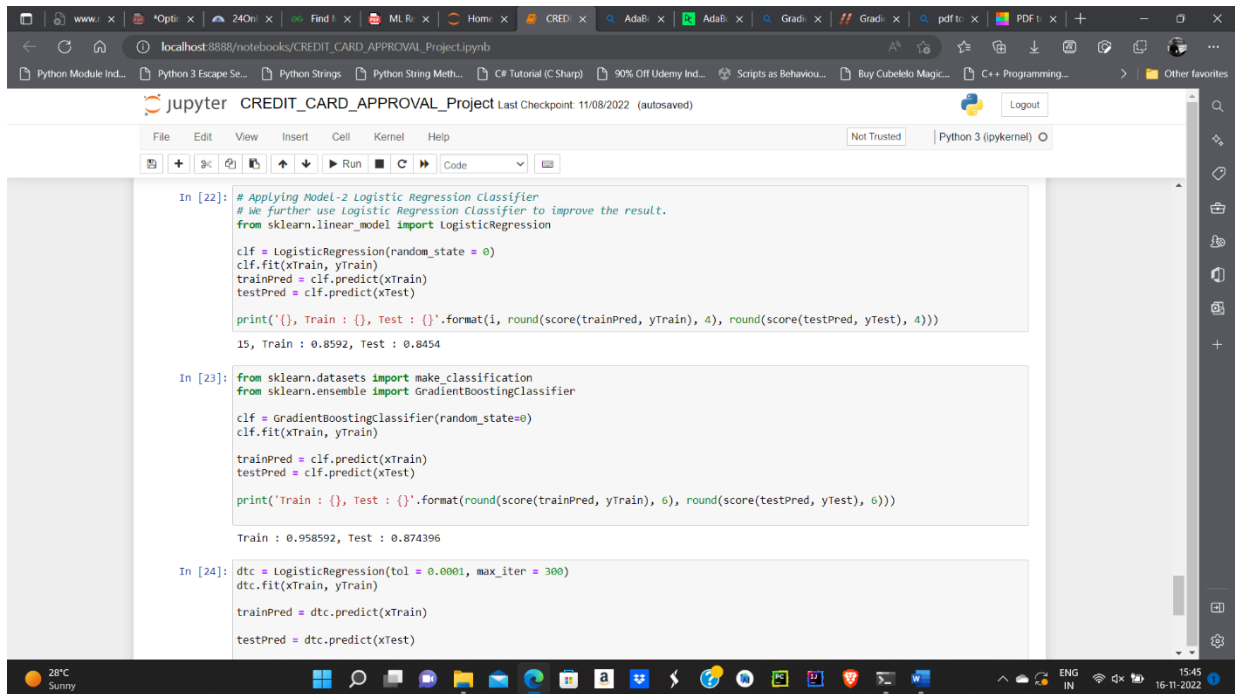
28°C Sunny 15:42 16-11-2022

Correlation Matrix and heatmap

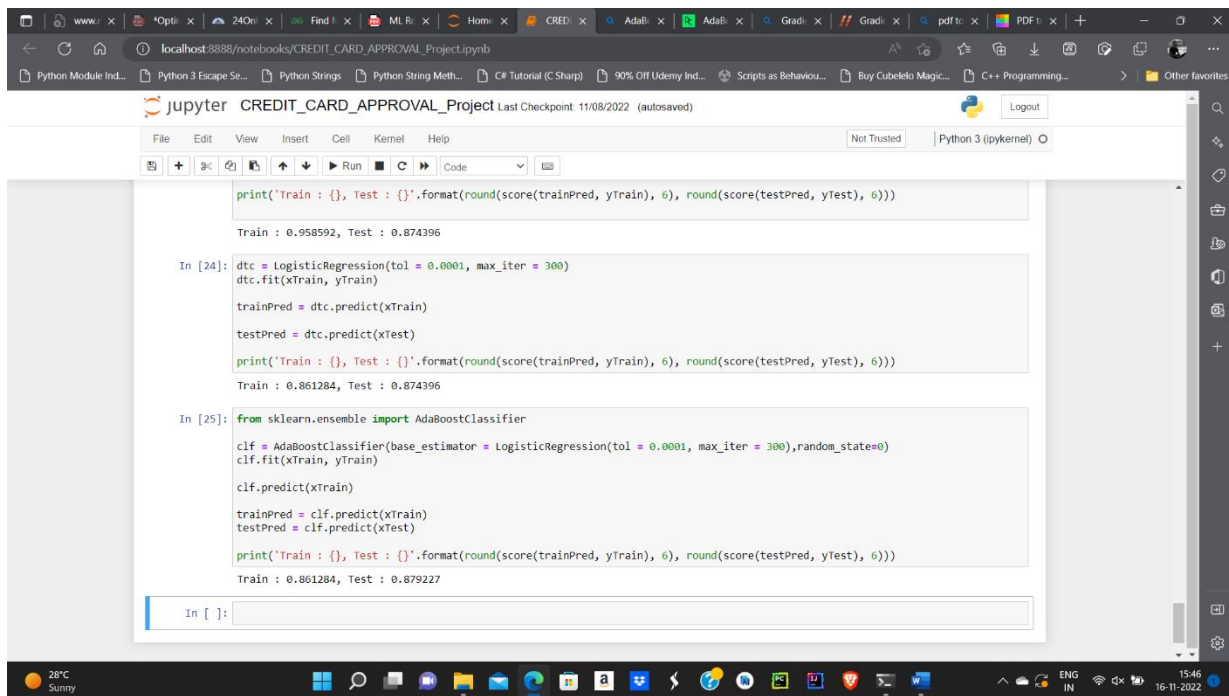




Decision tree Classifier



Logistic regression and Gradient Boosting classifier



```
print('Train : {}, Test : {}'.format(round(score(trainPred, yTrain), 6), round(score(testPred, yTest), 6)))
Train : 0.958592, Test : 0.874396

In [24]: dtc = LogisticRegression(tol = 0.0001, max_iter = 300)
dtc.fit(xTrain, yTrain)
trainPred = dtc.predict(xTrain)
testPred = dtc.predict(xTest)
print('Train : {}, Test : {}'.format(round(score(trainPred, yTrain), 6), round(score(testPred, yTest), 6)))
Train : 0.861284, Test : 0.874396

In [25]: from sklearn.ensemble import AdaBoostClassifier
clf = AdaBoostClassifier(base_estimator = LogisticRegression(tol = 0.0001, max_iter = 300), random_state=0)
clf.fit(xTrain, yTrain)
trainPred = clf.predict(xTrain)
testPred = clf.predict(xTest)
print('Train : {}, Test : {}'.format(round(score(trainPred, yTrain), 6), round(score(testPred, yTest), 6)))
Train : 0.861284, Test : 0.879227

In [ ]:
```

Algorithms used

Logistic regression:

It is the appropriate regression analysis to conduct when the dependent variable is dichotomous (binary). Like all regression analyses, the logistic regression is a predictive analysis. Logistic regression is used to describe data and to explain the relationship between one dependent binary variable and one or more nominal, ordinal, interval, or ratio-level independent variables.

Decision tree classifier:

Decision Tree is a Supervised learning technique that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules, and each leaf node represents the outcome.

Gradient Boosting classifier:

Gradient boosting is a method standing out for its prediction speed and accuracy, particularly with large and complex datasets. From Kaggle competitions to machine learning solutions for business, this algorithm has produced the best results. We already know that errors play a major role in any machine learning algorithm. There are mainly two types of errors, bias and variance error. Gradient boost algorithm helps us minimise bias error of the model.

Adaboost classifier:

An Adaboost classifier is a meta-estimator that begins by fitting a classifier on the original dataset and then fits additional copies of the classifier on the same data set but where the weight of in correctly classified instances is registered such that subsequent classifiers focus more on difficult cases

CONCLUSION

With ever increasing number of people who are actively using credit cards in today's world and focussing on the sheer exponential difference in the number of human employees that check and grant credit cards to customers, it is the need of the moment to introduce more reliable and sustainable means of technology that could take over this work efficiently

In these hard times, this Machine Learning project could be used to judge if multiple people make the appropriate cut to receive the credit card helping in not only reducing the human burden and mental agony but also making this process exponentially quicker.

Logistic regression:

Train Accuracy: 0.8592

Test Accuracy : 0.8454

Gradient Boosting Classifier:

Train Accuracy : 0.958592

Test Accuracy : 0.874396

AdaBoost Classifier:

Train Accuracy: 0.861284%

Test Accuracy

BIBLIOGRAPHY

- <https://www.kaggle.com/datasets/rikdifos/credit-card-approval-prediction>
- <https://medium.datadriveninvestor.com/predicting-credit-card-approvals-using-mltechniques9cd8eaeb5b8c>
- <https://www.youtube.com/watch?v=kO0dnOucoWc>

PROJECT GITHUB LINK

https://github.com/PratyushPriyam/CreditCardApproval_Project.git

