Classification Analysis on Credit Risk Prediction Dataset

GROUP - 1 / BHAVANS VIVEKANANDA COLLEGE

B. Sruthi | | Amreen | B. Saketh

107222546004 | 107222546002 | 107222546005

Abstract

The main purpose is to Analyze applicant profiles to understand factors affecting credit approval decisions and help Financial Institutes make better decisions

Objective

The main objective is to identify key drivers and assist risk management .The support development of fair,transparent provide insights to improve credit scoring models.

CONTENT

Introduction

The credit card approval process involves a complex evaluation of your financial history and creditworthiness. Understanding the science behind credit card approval can help you make more informed decisions and improve your chances of getting the card you want.

This dataset has the details which are age,income,debt levels,employement stability

Literature Review

Literarture Review 1

Author:Archana Gahlaut

University of Delhi, New Delhi, Delhi, IN

This paper explores the use of data mining techniques to predict and classify customers' credit scores as good or bad, helping banks assess the risk of lending. By using historical data from a bank, predictive models are developed to aid in better decision-making regarding loans. The accuracy of these models can significantly improve banks' credit risk management.

Published in: 2017 8th International Conference on Computing, Communication and Networking Technologies (ICCCNT)

Literature Review 2

Author:Trilok Nath Pandey

This paper surveys various techniques for credit risk analysis in the banking industry, focusing on evaluating customer datasets to decide loan approvals or rejections. It highlights the challenges of analyzing credit risk data for informed decision-making. The accuracy of these techniques is crucial for effective risk management and lending decisions.

Published in: 2017 International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS)

Data Preprocessing

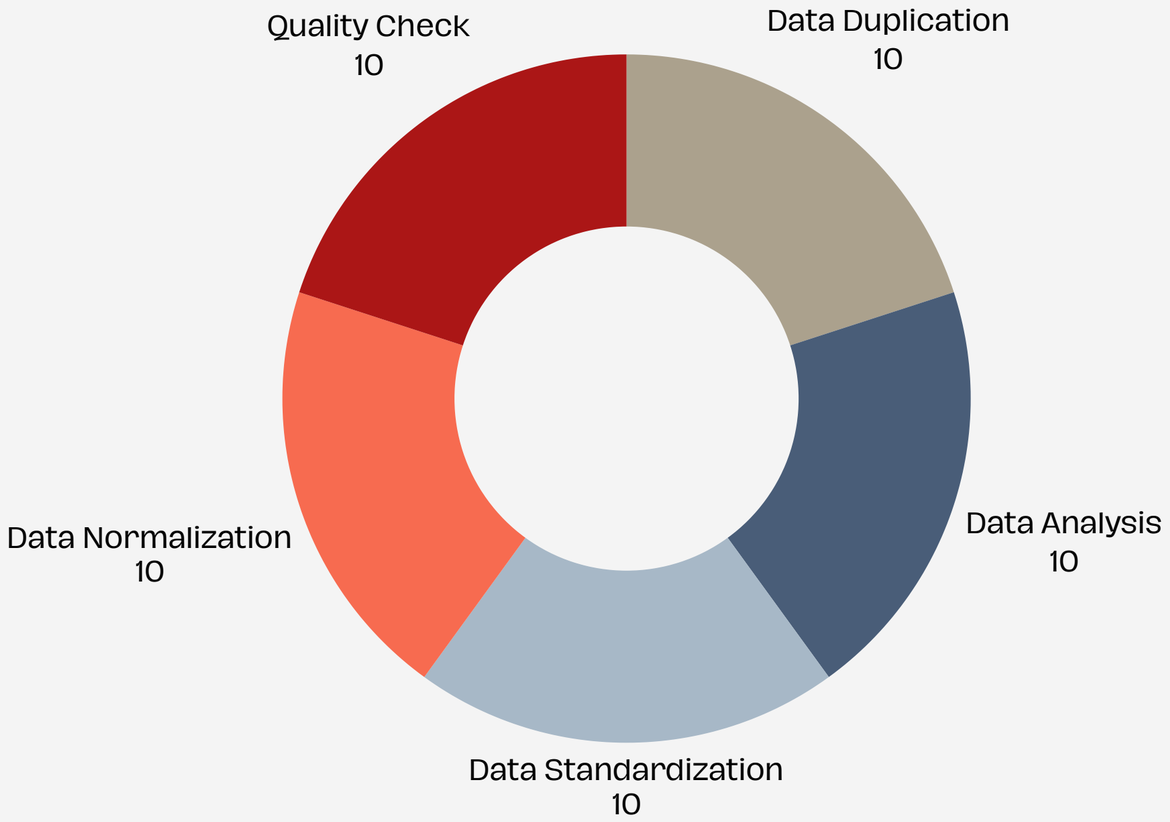
Data

Dataset: Our dataset consists of 16 variables and 690 records

Source: https://drive.google.com/drive/folders/103h2OzYaGAVmcWAfgGsBYHfdKAdJMFn-?usp=drive_link

Gender	Age	Debt	Married	BankCusto	Industry	Ethnicity	YearsEmpl	PriorDefau	Employed	CreditScor	DriversLice	Citizen	ZipCode	Income	Approved
1	30.83	0	1	1	Industrials	White	1.25	1	1	1	0	ByBirth	202	0	1
0	58.67	4.46	1	1	Materials	Black	3.04	1	1	6	0	ByBirth	43	560	1
0	24.5	0.5	1	1	Materials	Black	1.5	1	0	0	0	ByBirth	280	824	1
1	27.83	1.54	1	1	Industrials	White	3.75	1	1	5	1	ByBirth	100	3	1
1	20.17	5.625	1	1	Industrials	White	1.71	1	0	0	0	ByOtherM	120	0	1
1	32.08	4	1	1	Communic	White	2.5	1	0	0	1	ByBirth	360	0	1
1	33.17	1.04	1	1	Transport	Black	6.5	1	0	0	1	ByBirth	164	31285	1
0	22.92	11.585	1	1	Informatic	White	0.04	1	0	0	0	ByBirth	80	1349	1
1	54.42	0.5	0	0	Financials	Black	3.96	1	0	0	0	ByBirth	180	314	1
1	42.5	4.915	0	0	Industrials	White	3.165	1	0	0	1	ByBirth	52	1442	1
1	22.08	0.83	1	1	Energy	Black	2.165	0	0	0	1	ByBirth	128	0	1
1	29.92	1.835	1	1	Energy	Black	4.335	1	0	0	0	ByBirth	260	200	1
0	38.25	6	1	1	Financials	White	1	1	0	0	1	ByBirth	0	0	1
1	48.08	6.04	1	1	Financials	White	0.04	0	0	0	0	ByBirth	0	2690	1
0	45.83	10.5	1	1	Materials	White	5	1	1	7	1	ByBirth	0	0	1
1	36.67	4.415	0	0	Financials	White	0.25	1	1	10	1	ByBirth	320	0	1
1	28.25	0.875	1	1	Communic	White	0.96	1	1	3	1	ByBirth	396	0	1
0	23.25	5.875	1	1	Materials	White	3.17	1	1	10	0	ByBirth	120	245	1
1	21.83	0.25	1	1	Real Estat	Black	0.665	1	0	0	1	ByBirth	0	0	1
0	19.17	8.585	1	1	Informatic	Black	0.75	1	1	7	0	ByBirth	96	0	1
1	25	11.25	1	1	Energy	White	2.5	1	1	17	0	ByBirth	200	1208	1
1	23.25	1	1	1	Energy	White	0.835	1	0	0	0	ByOtherM	300	0	1

Data Cleaning



Checked for null /NaN values : We replaced the Nan values with mean and mode of the column based on the attribute type.

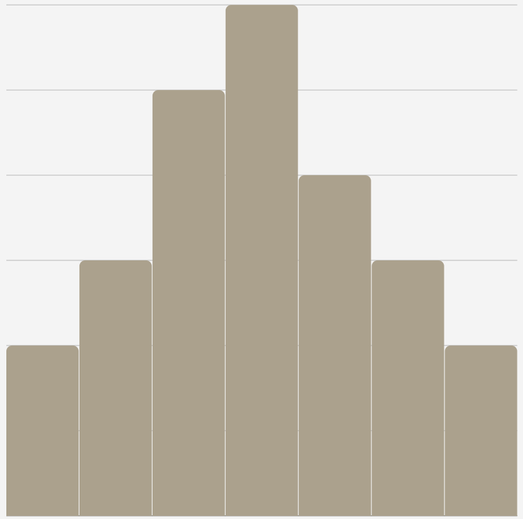
Garbage values : We replaced the garbage values of the column based on the domain knowledge of the dataset.

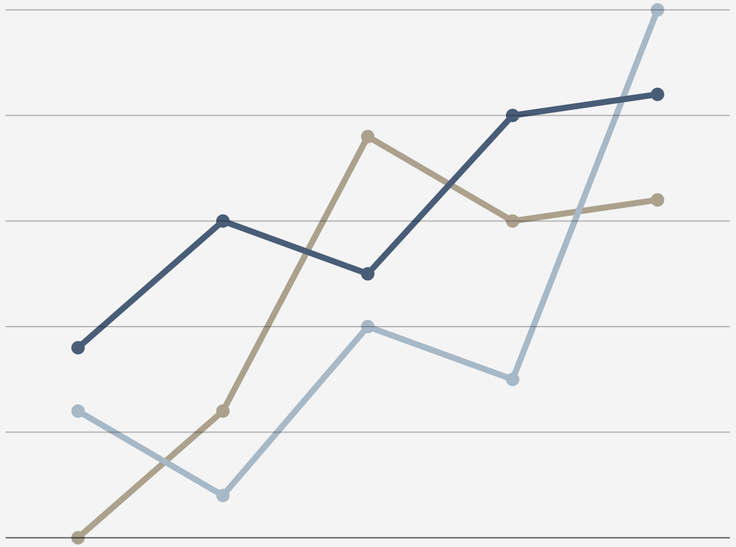
Dummification: enabled and dummified the categorical variables.

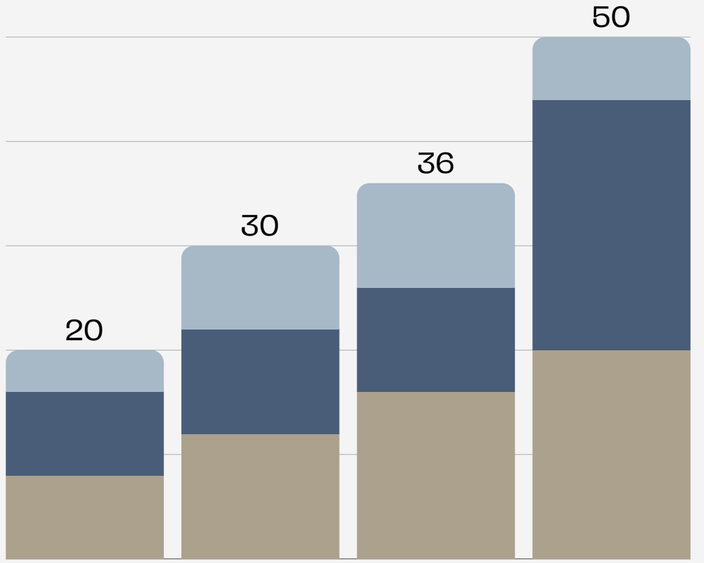
variables:

categorical variables: Gender, Married, BankCustomer, Industry, Ethnicity, PriorDefault, Employed, DriversLicense, Citizen, Zipcode, Approved

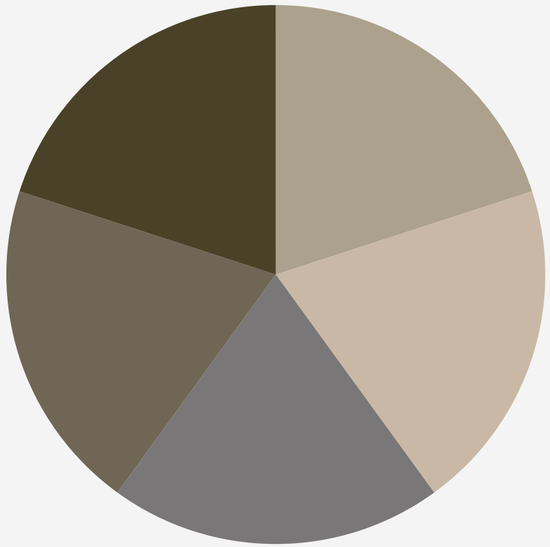
continuous variables: Age, Debt, YearsEmployed, CreditScore,Income

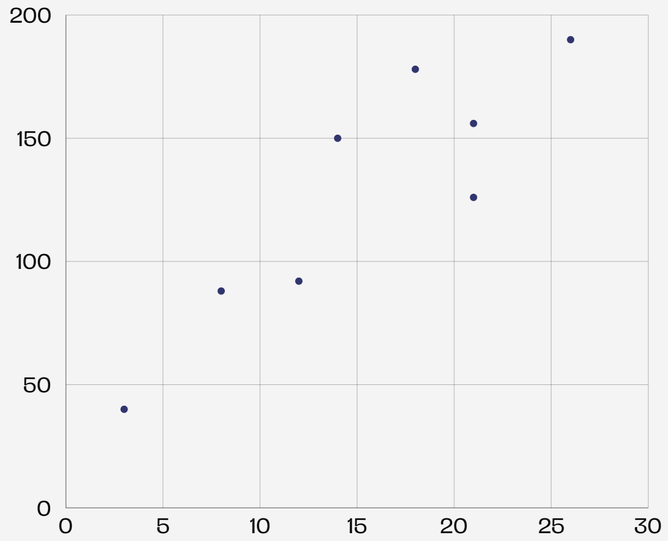


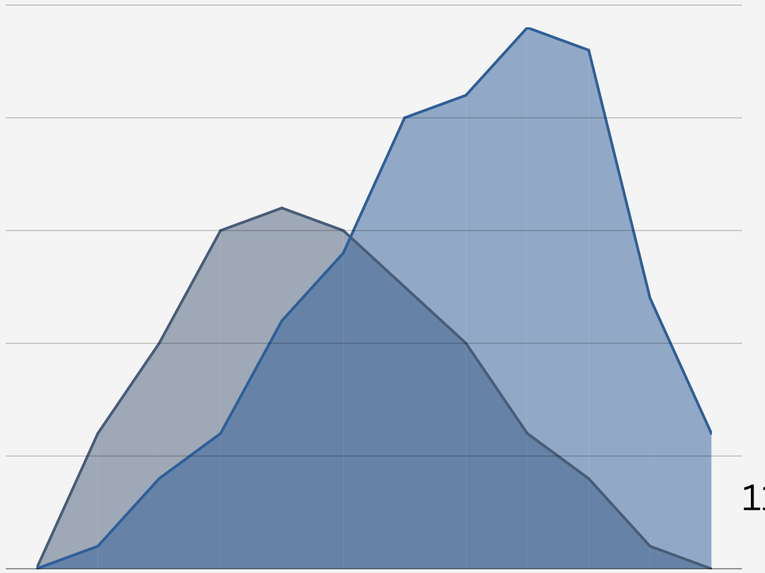




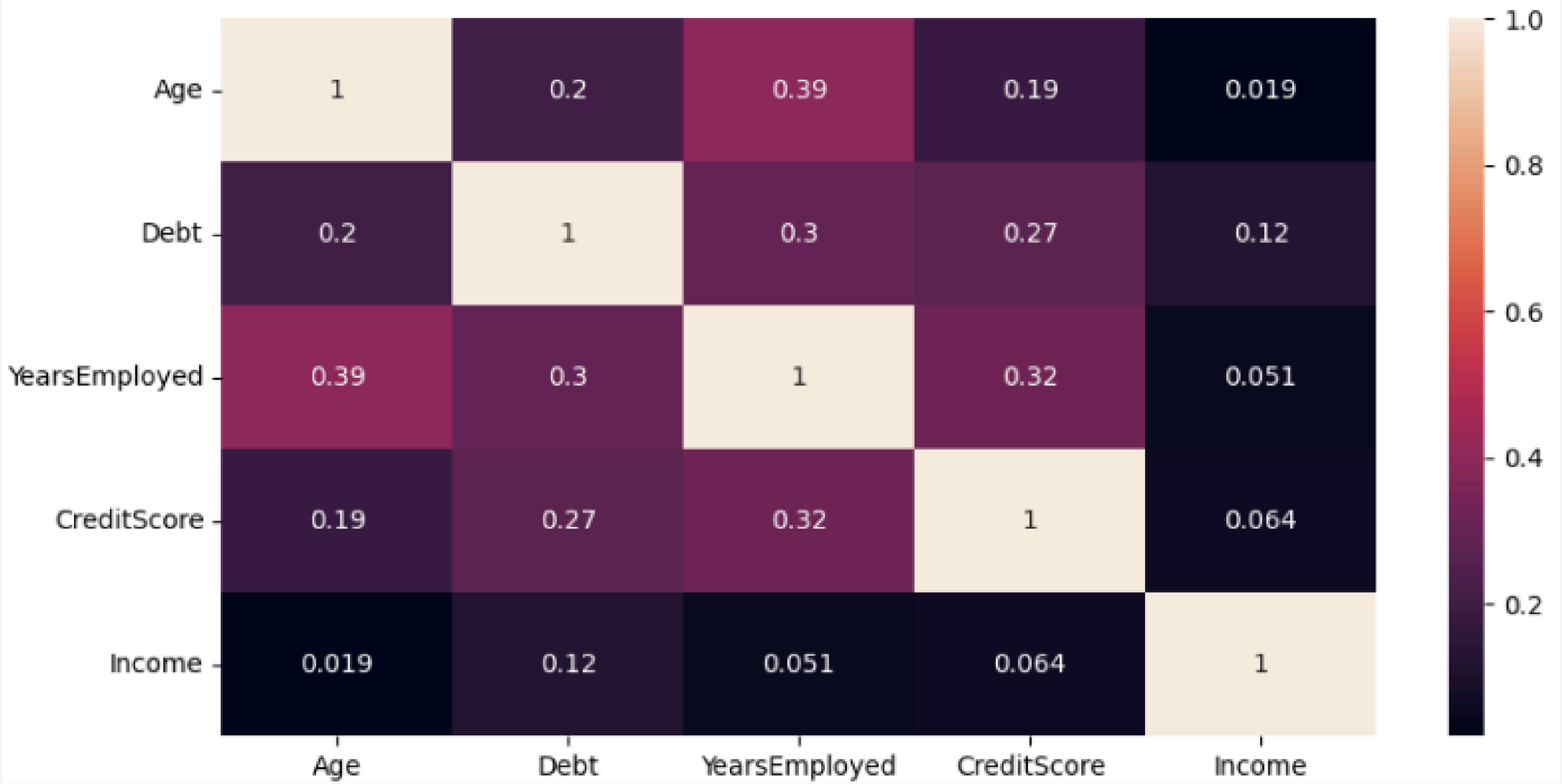
Exploratory Data Analysis



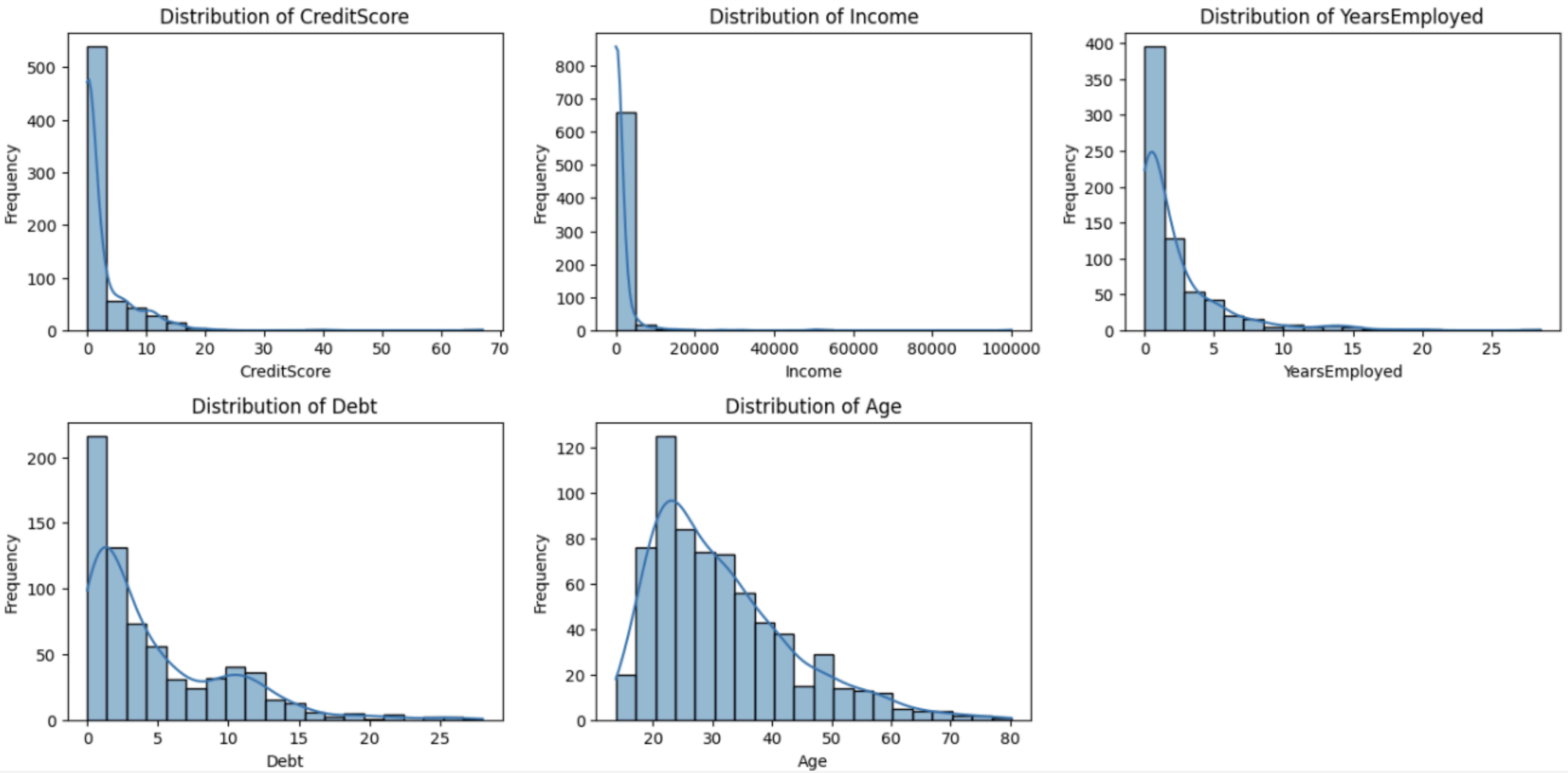




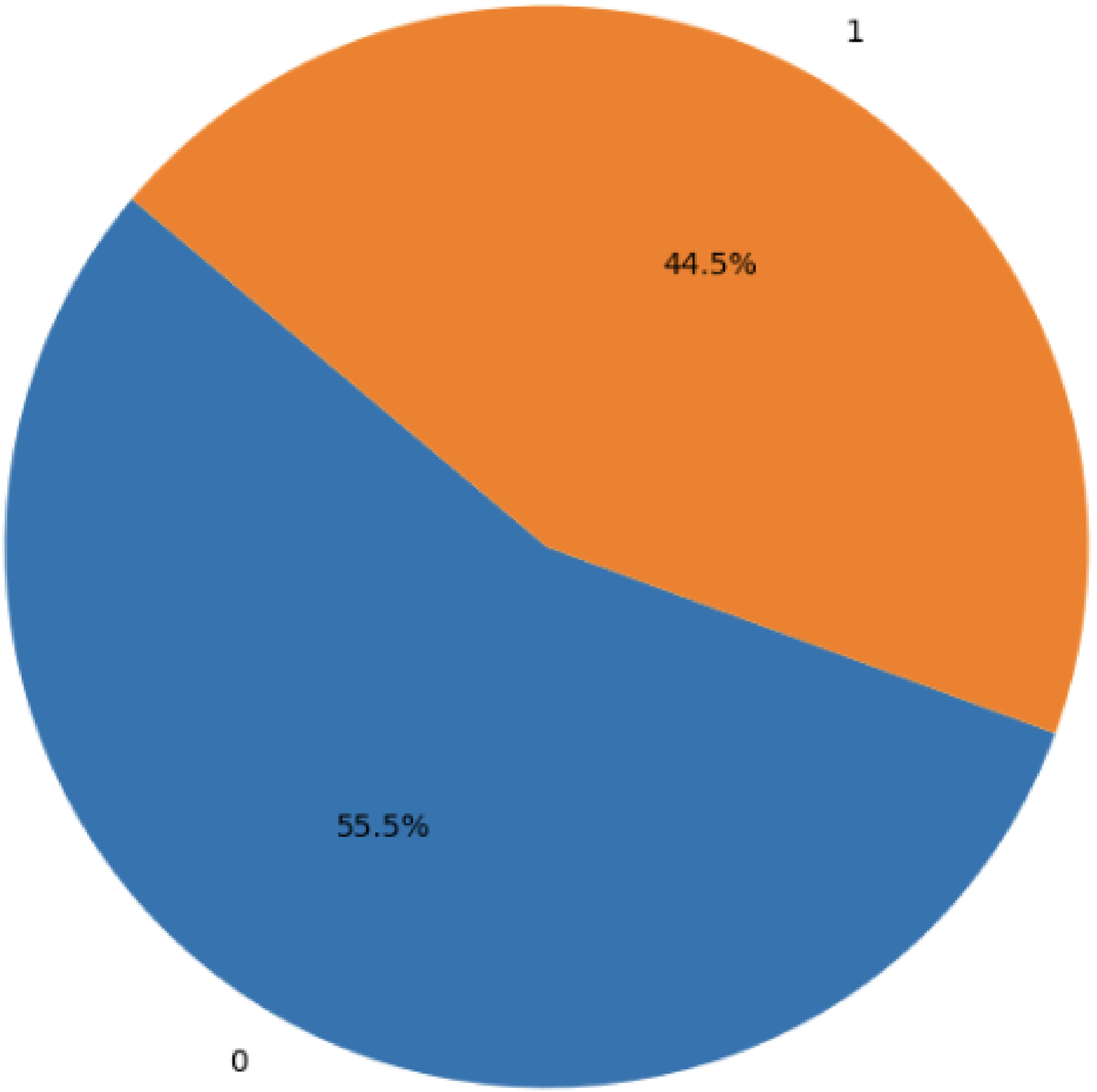
Correlation Matrix



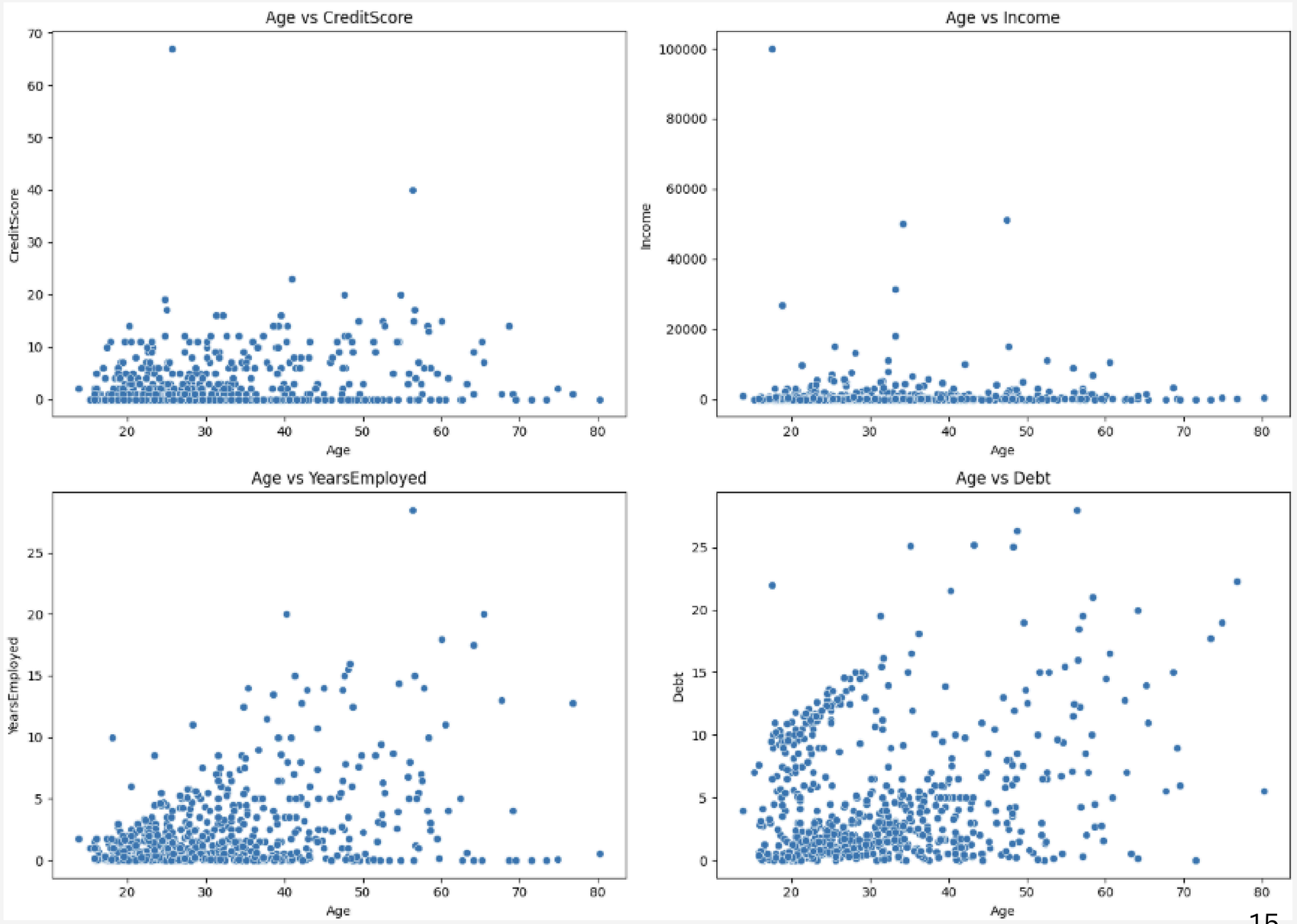
Histogram


Distribution of CreditScore
Frequency
CreditScore
Distribution of Income
Frequency
Income
Distribution of YearsEmployed
Frequency
YearsEmployed
Distribution of Debt
Frequency
Debt
Distribution of Age
Frequency
Age

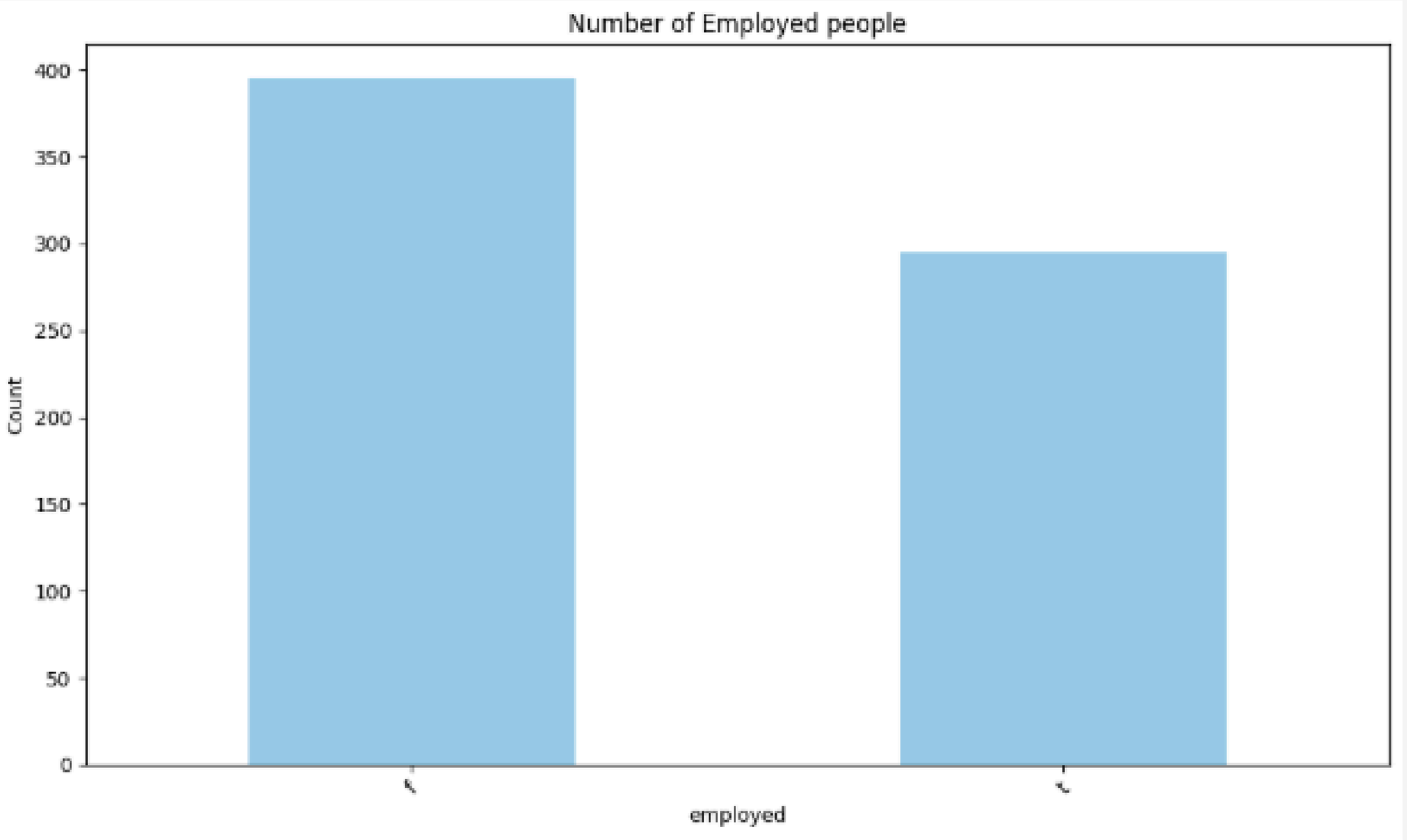
Pie chart



Scatter plot



Bar plot


Number of Employed people
Count
400
350
300
250
200
150
100
50
0
employed

MULTICOLLINEARITY CHECK

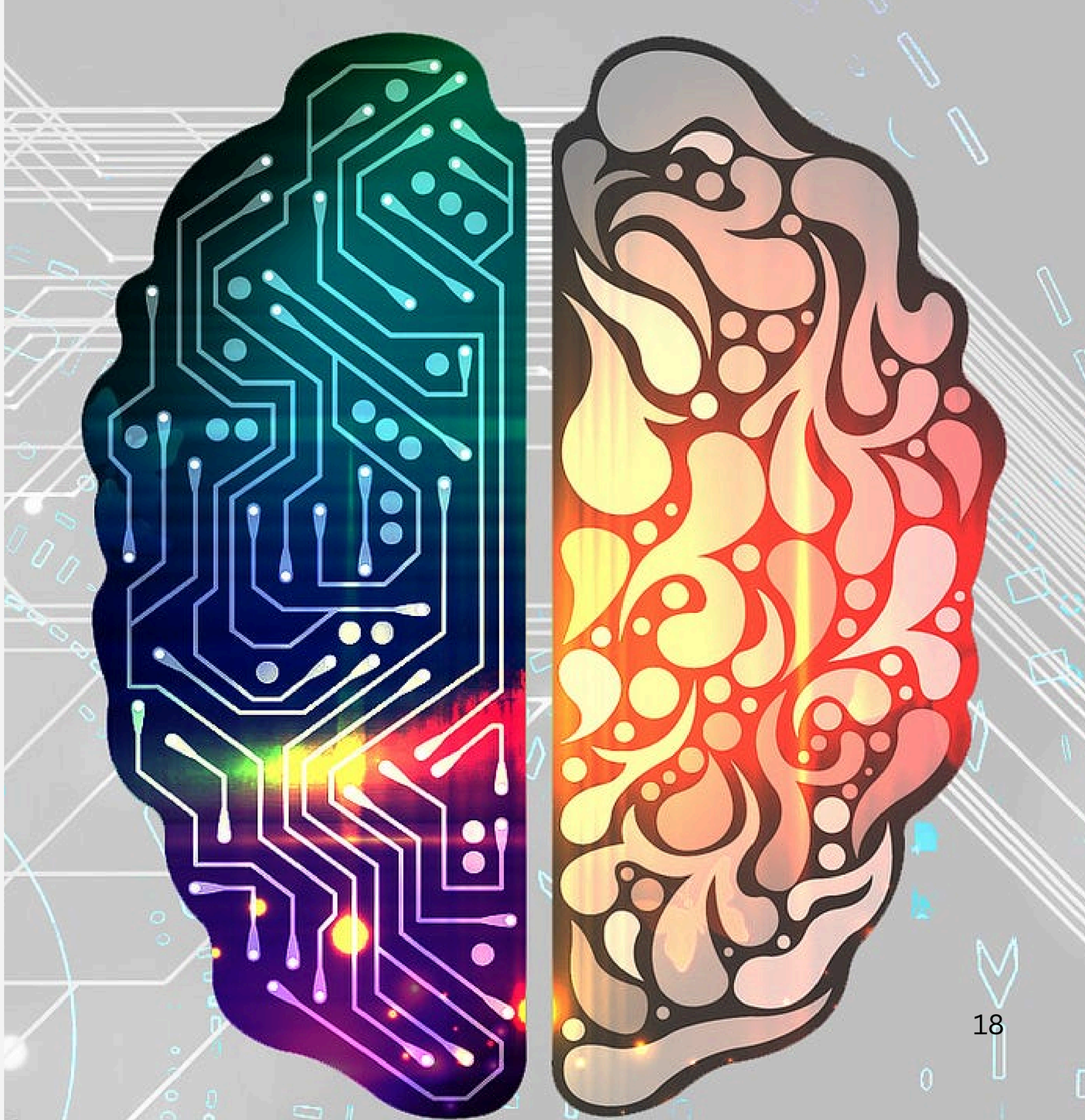
We have removed the variables which are highly correlated to each other, the variables are
'Married_y','Ethnicity_ff','Age','Industry_j','Ethnicity_v','Gender_b','Income','PriorDefault_t'

	variables	VIF
0	Age	9.2
1	Debt	2.4
2	YearsEmployed	2.1
3	CreditScore	2.1
4	Income	1.6
5	Gender_b	3.7
6	Married_y	inf
7	BankCustomer_p	inf
8	Industry_c	3.1
9	Industry_cc	1.7
10	Industry_d	1.5
11	Industry_e	2.8
12	Industry_ff	14.9
13	Industry_i	1.7
14	Industry_j	3.5
15	Industry_k	1.8
16	Industry_m	1.7
17	Industry_q	2.2
18	Industry_r	1.6
19	Industry_w	2.0
20	Industry_x	1.7
21	Ethnicity_dd	1.6
22	Ethnicity_ff	14.6
23	Ethnicity_h	3.1
24	Ethnicity_j	3.1
25	Ethnicity_n	1.6
26	Ethnicity_o	1.5
27	Ethnicity_v	6.0
28	Ethnicity_z	1.9

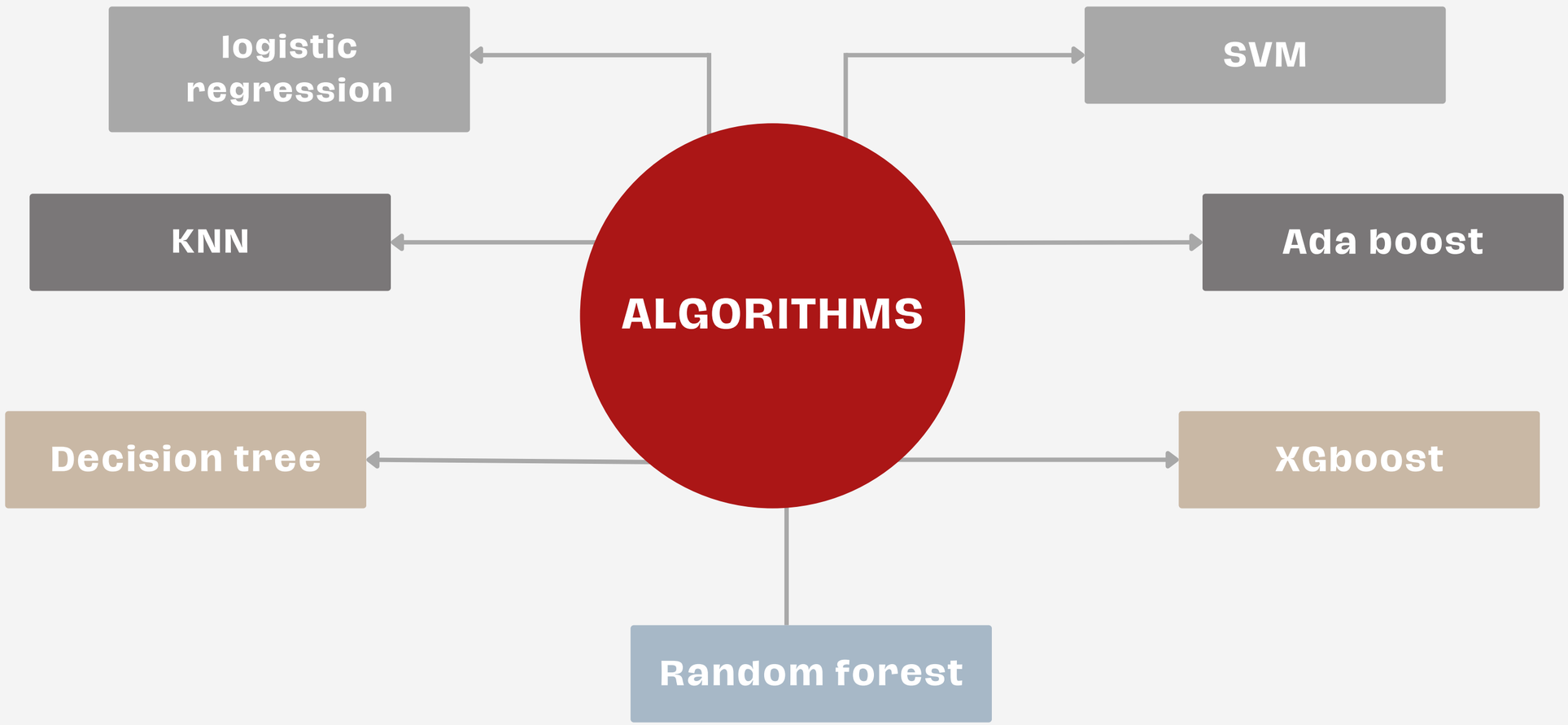
	variables	VIF
0	Debt	2.2
1	YearsEmployed	1.9
2	CreditScore	2.1
3	Income	1.5
4	BankCustomer_p	1.3
5	Industry_c	1.6
6	Industry_cc	1.3
7	Industry_d	1.2
8	Industry_e	2.5
9	Industry_ff	1.2
10	Industry_i	1.2
11	Industry_k	1.2
12	Industry_m	1.3
13	Industry_q	1.5
14	Industry_r	1.6
15	Industry_w	1.3
16	Industry_x	1.4
17	Ethnicity_dd	1.6
18	Ethnicity_h	1.5
19	Ethnicity_j	1.0
20	Ethnicity_n	1.6
21	Ethnicity_o	1.4
22	Ethnicity_z	1.8
23	PriorDefault_t	3.0
24	Employed_t	3.1
25	DriversLicense_t	1.9

	variables	VIF
0	Debt	2.2
1	YearsEmployed	1.9
2	CreditScore	2.1
3	BankCustomer_p	1.3
4	Industry_c	1.6
5	Industry_cc	1.2
6	Industry_d	1.2
7	Industry_e	2.4
8	Industry_ff	1.2
9	Industry_i	1.2
10	Industry_k	1.2
11	Industry_m	1.3
12	Industry_q	1.5
13	Industry_r	1.5
14	Industry_w	1.3
15	Industry_x	1.3
16	Ethnicity_dd	1.6
17	Ethnicity_h	1.5
18	Ethnicity_j	1.0
19	Ethnicity_n	1.5
20	Ethnicity_o	1.0
21	Ethnicity_z	1.8
22	Employed_t	2.9
23	DriversLicense_t	1.9

Machine learning algorithms



MACHINE LEARNING ALGORITHMS USED



70-30 Train-test Split

Algorithm	MODEL-1(Before VIF) Accuracy	MODEL-2(After VIF) Accuracy
Logistic Regression	0.826	0.743
KNN	0.681	0.743
SVM	0.830	0.768
Decision Tree	0.840	0.748
Random Forest	0.864	0.700
AdaBoost	0.821	0.758
XGBoost	0.850	0.748

60-40 Train test split

Algorithms	MODEL-1(Before VIF) Accuracy	MODEL-2(After VIF) Accuracy
Logistic Regression	0.822	0.775
KNN	0.706	0.760
SVM	0.836	0.753
Decision Tree	0.847	0.750
Random Forest	0.865	0.721
AdaBoost	0.815	0.713
XGBoost	0.862	0.779

75-25 Train test split

Algorithm	MODEL-1(Before VIF) Accuracy	MODEL-2(After VIF) Accuracy
Logistic Regression	0.826	0.757
KNN	0.664	0.745
SVM	0.820	0.774
Decision Tree	0.843	0.745
Random Forest	0.849	0.716
AdaBoost	0.803	0.734
XGBoost	0.855	0.774

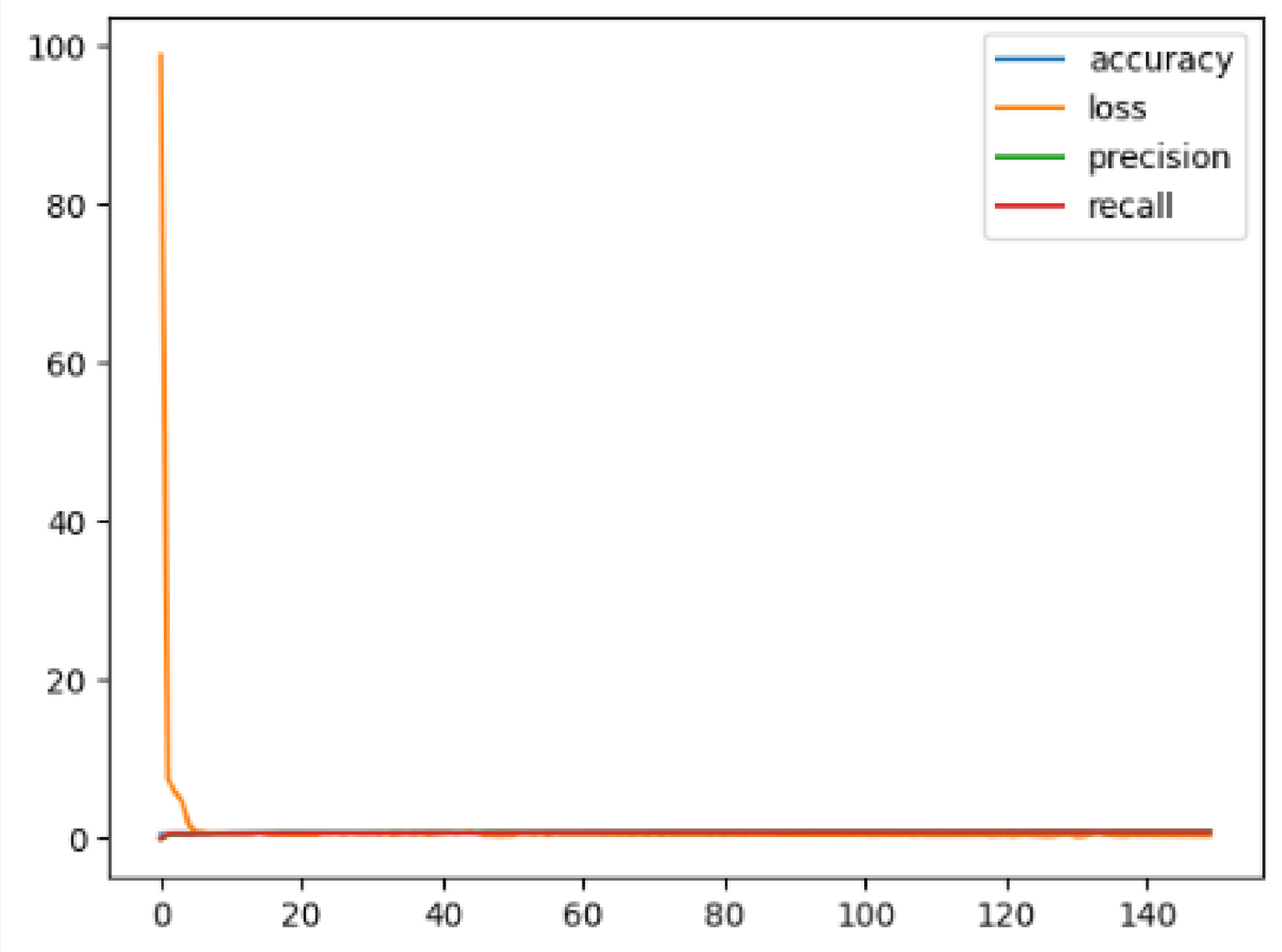
80-20 Train test split

Algorithm	MODEL-1(Before VIF) Accuracy	MODEL-2(After VIF) Accuracy
Logistic Regression	0.811	0.768
KNN	0.644	0.717
SVM	0.804	0.782
Decision Tree	0.826	0.724
Random Forest	0.840	0.673
AdaBoost	0.782	0.746
XGBoost	0.833	0.775

Neural Networks

Train- Test	Architecture	Optimizer	Epochs	Accuracy	Loss	Precision	Recall
60-40	30-20-10-1	Adam	150	0.7698	0.51	0.7766	0.6994
60-40	30-20-10-1	Adam	150	0.7204	1.4583	0.6986	0.6772
70-30	30-20-10-1	Adam	150	0.7852	0.7827	0.7753	0.7583
70-30	30-20-10-1	Adam	150	0.7318	1.5817	0.7167	0.6919
75-25	30-20-10-1	Adam	150	0.7676	0.8678	0.8062	0.676
75-25	30-20-10-1	Adam	150	0.6827	1.5758	0.6665	0.6329
80-20	30-20-10-1	Adam	150	0.7777	1.0747	0.7898	0.7304
80-20	30-20-10-1	Adam	150	0.7309	2.5906	0.7213	0.5885

Neural Network plot for observation



Train-Test-Split	80-20
Architecture	30-20-10-1
Optimizer	Adam
Epochs	150

Comparison of Algorithms

1. 60-40 Split Before Applying VIF

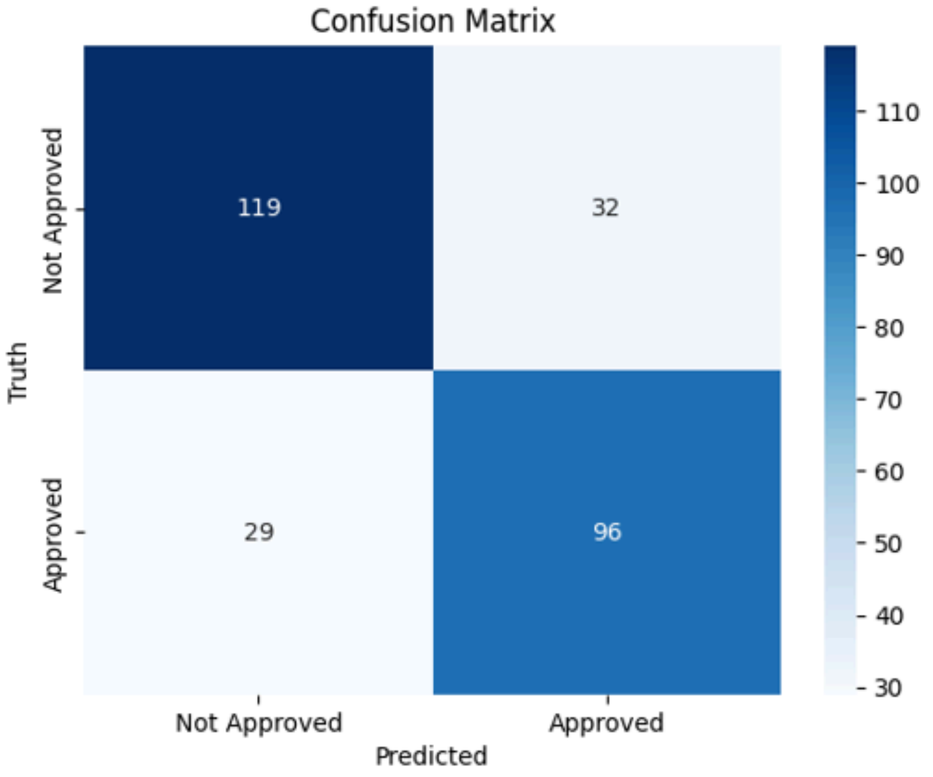
Algorithm	Accuracy
Logistic Regression	0.822
KNN	0.706
SVM	0.836
Decision Tree	0.847
Random Forest	0.865
AdaBoost	0.815
XGBoost	0.862

2. 80-20 Split After Applying VIF

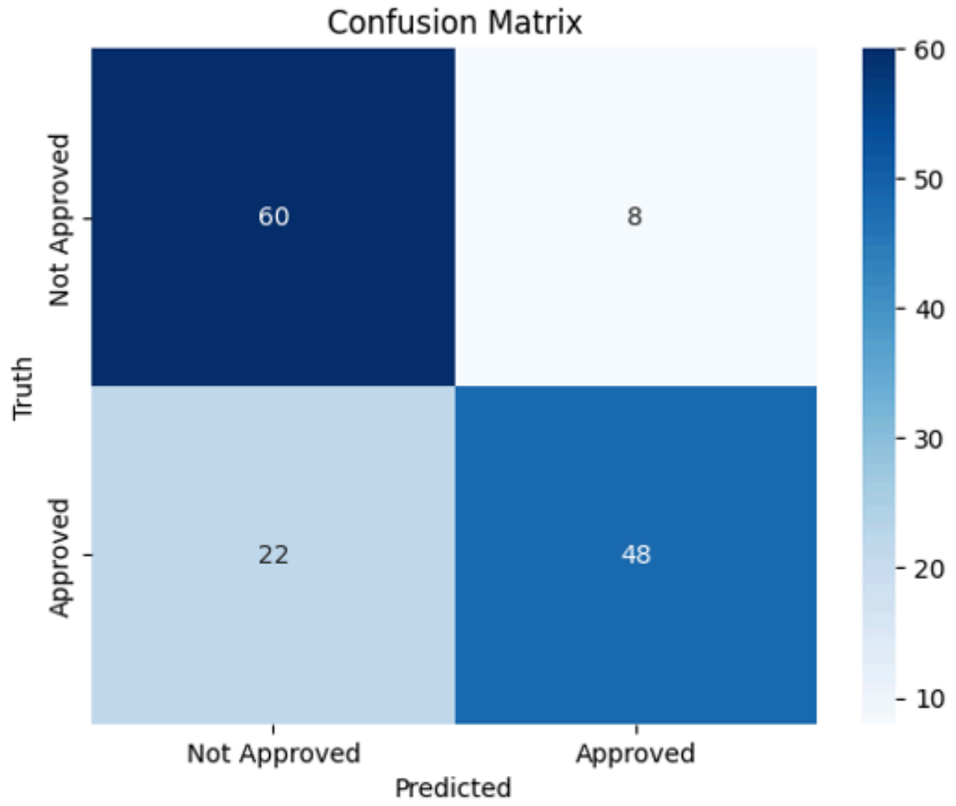
Algorithm	Accuracy
Logistic Regression	0.768
KNN	0.717
SVM	0.782
Decision Tree	0.724
Random Forest	0.673
AdaBoost	0.746
XGBoost	0.775

Confusion Matrix

1. RANDOM FOREST before VIF for 60:40 split



2. SVM after VIF for 80:20 spit



Summary

- The main aim is to predict the best model for credit card approval.
- for the actual data we have Random Forest showing the best results with an Accuracy of 86.5% and XGBoost Algorithm showing next best Accuracy of 85.5%
- And after removing the High Multicollinearity variables, the SVM Algorithm is showing the Best results with an Accuracy of 78.2%

Future Directions

-Real Time Credit Scoring

Implementing models that can process and assess application in real-time, integrating up to date financial data which could lead to faster and more responsive credit card approval systems

Insights

Transparent and Equitable Approval Process: The analysis supports transparency by revealing decision patterns, fostering trust, and promoting fair access to credit, reducing biases in approvals which helps financial organizations to avoid credit risk and help practice fair lending to the eligible applicable

Work Distribution

Amreen 107222546002	Collecting Information about credi t card approvals and Literature Re view
B.sruthi 107222546004	Data Pre-processing and EDA
B,saketh 107222546005	Implementing ML Algorithms

colab notebook

Thank you

B.sruthi

B.saketh

Amreen

Appendix

Uploading the dataset

```
from google.colab import files
uploaded = files.upload()
```

Choose Files | No file chosen | Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving credit-card.data to credit-card (1).data
```

+ Code + Text

```
data=pd.read_csv('/content/credit-card.data',header=None)
data
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	b	30.83	0.000	u	g	w	v	1.25	t	t	1	f	g	00202	0	+
1	a	58.67	4.460	u	g	q	h	3.04	t	t	6	f	g	00043	560	+
2	a	24.50	0.500	u	g	q	h	1.50	t	f	0	f	g	00280	824	+
3	b	27.83	1.540	u	g	w	v	3.75	t	t	5	t	g	00100	3	+
4	b	20.17	5.625	u	g	w	v	1.71	t	f	0	f	s	00120	0	+
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
685	b	21.08	10.085	y	p	e	h	1.25	f	f	0	f	g	00260	0	-
686	a	22.67	0.750	u	g	c	v	2.00	f	t	2	t	g	00200	394	-
687	a	25.25	13.500	y	p	ff	ff	2.00	f	t	1	t	g	00200	1	-

CHECKING THE UNIQUE VALUES AND ITS COUNT

```
for i in range(data.shape[1]):
  print(data.iloc[:,i].unique())
  print(data.iloc[:,i].value_counts())
```

```
['b' 'a']
Gender
b    480
a    210
Name: count, dtype: int64
[30.83       58.67       24.5        27.83       20.17       32.08
 33.17       22.92       54.42       42.5        22.08       29.92
 38.25       48.08       45.83       36.67       28.25       23.25
 21.83       19.17       25.         47.75       27.42       41.17
 15.83       47.         56.58       57.42       42.08       29.25
 42.         49.5        36.75       22.58       27.25       23.
 27.75       54.58       34.17       28.92       29.67       39.58
 56.42       54.33       41.         31.92       41.5        23.92
 25.75       26.         37.42       34.92       34.25       23.33
 23.17       44.33       35.17       43.25       56.75       31.67
 23.42       20.42       26.67       36.         25.5        19.42
 32.33       34.83       38.58       44.25       44.83       20.67
 34.08       21.67       21.5        49.58       27.67       39.83
 31.56817109 37.17       25.67       34.         49.         62.5
 31.42       52.33       28.75       28.58       22.5        28.5
 37.5        35.25       18.67       54.83       40.92       19.75
 29.17       24.58       33.75       25.42       37.75       52.5
 57.83       20.75       39.92       24.75       44.17       23.5
 47.67       22.75       34.42       28.42       67.75       47.42
```

DIVIDING THE DATA

```
X=data.drop(["Approved"],axis=1)
print(X)
y=data["Approved"]
print(y)
```

```
    Gender    Age    Debt Married BankCustomer Industry Ethnicity  \
0        b  30.83   0.000       u            g        w         v
1        a  58.67   4.460       u            g        q         h
2        a  24.50   0.500       u            g        q         h
3        b  27.83   1.540       u            g        w         v
4        b  20.17   5.625       u            g        w         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

     YearsEmployed PriorDefault Employed  CreditScore DriversLicense Citizen  \
0             1.25            t        t            1              f       g
1             3.04            t        t            6              f       g
2             1.50            t        f            0              f       g
3             3.75            t        t            5              t       g
4             1.71            t        f            0              f       s
..             ...          ...      ...          ...            ...     ...
685           1.25            f        f            0              f       g
686           2.00            f        t            2              t       g
```

CHECKING DATA TYPES

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 690 entries, 0 to 689
Data columns (total 16 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   Gender          690 non-null    object 
 1   Age             690 non-null    object 
 2   Debt            690 non-null    float64
 3   Married         690 non-null    object 
 4   BankCustomer    690 non-null    object 
 5   Industry        690 non-null    object 
 6   Ethnicity       690 non-null    object 
 7   YearsEmployed   690 non-null    float64
 8   PriorDefault    690 non-null    object 
 9   Employed        690 non-null    object 
 10  CreditScore     690 non-null    int64  
 11  DriversLicense  690 non-null    object 
 12  Citizen         690 non-null    object 
 13  ZipCode         690 non-null    object 
 14  Income          690 non-null    int64  
 15  Approved        690 non-null    object 
dtypes: float64(2), int64(2), object(12)
memory usage: 86.4+ KB
```

CHECKING FOR NULL VALUES

```
data.isna().sum()
```

	0
Gender	0
Age	0
Debt	0
Married	0
BankCustomer	0
Industry	0
Ethnicity	0
YearsEmployed	0
PriorDefault	0
Employed	0
CreditScore	0
DriversLicense	0
Citizen	0
ZipCode	0

CREATING THE DUMMY VARAIBLES

```
X=pd.get_dummies(X,dtype='int',drop_first=True)
print(X)
```

```
          Age     Debt   YearsEmployed   CreditScore   Income   Gender_b   Married_y   \
0     30.83    0.000              1.25             1        0          1           0
1     58.67    4.460              3.04             6      560          0           0
2     24.50    0.500              1.50             0      824          0           0
3     27.83    1.540              3.75             5        3          1           0
4     20.17    5.625              1.71             0        0          1           0
..      ...      ...               ...           ...      ...        ...         ...
685   21.08   10.085              1.25             0        0          1           1
686   22.67    0.750              2.00             2      394          0           0
687   25.25   13.500              2.00             1        1          0           1
688   17.92    0.205              0.04             0      750          1           0
689   35.00    3.375              8.29             0        0          1           0

          BankCustomer_p   Industry_c   Industry_cc   ...   Ethnicity_h   Ethnicity_j   \
0                      0            0             0   ...             0             0
1                      0            0             0   ...             1             0
2                      0            0             0   ...             1             0
3                      0            0             0   ...             0             0
4                      0            0             0   ...             0             0
..                   ...          ...           ...   ...           ...           ...
685                    1            0             0   ...             1             0
686                    0            1             0   ...             0             0
687                    1            0             0   ...             0             0
688                    0            0             0   ...             0             0
689                    0            1             0   ...             1             0
```

LOGISTIC REGRESSION BEFORE VIF

```
[ ]  from sklearn.linear_model import LogisticRegression
     logreg = LogisticRegression(C=1e9)
```

60-40

```
logreg.fit(X_train1, y_train1)
predictions = logreg.predict(X_test1)
print(predictions)
```

```
['0' '1' '0' '0' '0' '0' '0' '0' '0' '0' '0' '1' '0' '0' '1' '1' '0' '0'
 '0' '0' '0' '0' '0' '0' '0' '0' '1' '1' '0' '0' '1' '1' '0' '1' '0' '1'
 '1' '1' '1' '1' '1' '1' '0' '0' '0' '1' '0' '0' '1' '0' '0' '0' '0' '0'
 '1' '0' '1' '0' '1' '0' '1' '1' '1' '0' '0' '1' '1' '0' '1' '0' '0' '1'
 '0' '1' '0' '0' '0' '1' '1' '0' '1' '1' '0' '0' '1' '0' '1' '1' '1' '1'
 '1' '1' '1' '1' '1' '1' '1' '1' '0' '0' '0' '0' '1' '1' '1' '1' '0' '1'
 '0' '1' '0' '1' '1' '0' '1' '0' '1' '1' '1' '1' '0' '0' '1' '0' '0' '0'
 '0' '0' '1' '1' '1' '0' '1' '0' '1' '1' '1' '1' '0' '1' '1' '1' '1' '1'
 '0' '0' '0' '1' '1' '0' '0' '0' '0' '1' '0' '1' '1' '1' '0' '1' '1' '0'
 '0' '0' '0' '0' '0' '1' '0' '1' '0' '0' '1' '0' '0' '0' '0' '0' '0' '0'
 '0' '0' '1' '0' '1' '0' '0' '1' '1' '1' '0' '1' '0' '1' '0' '1' '1' '1'
 '1' '1' '1' '1' '0' '1' '1' '0' '1' '1' '0' '0' '1' '0' '1' '1' '0' '0'
 '1' '0' '0' '0' '1' '1' '0' '0' '1' '0' '0' '0' '0' '1' '0' '0' '1' '0'
 '0' '0' '1' '0' '1' '0' '0' '1' '1' '0' '0' '1' '0' '1' '0' '0' '1' '0'
 '1' '0' '0' '0' '0' '1' '1' '0' '1' '1' '0' '1' '0' '1' '1' '1' '0' '1'
```

KNN BEFORE VIF

60-40

```
[ ]  from sklearn.neighbors import KNeighborsClassifier
```

```
[ ]  model=KNeighborsClassifier(n_neighbors=25)
```

```
[ ]  model.fit(X_train1, y_train1)
```

KNeighborsClassifier

KNeighborsClassifier(n_neighbors=25)

```
[ ]  y_pred1 = model.predict(X_test1)
     y_pred1
```

```
array(['0', '1', '0', '1', '0', '1', '0', '0', '0', '0', '0', '1', '0',
       '0', '1', '0', '1', '0', '1', '0', '0', '0', '0', '0', '0', '0',
       '1', '0', '0', '1', '1', '1', '1', '0', '0', '1', '1', '0', '1',
       '1', '0', '1', '0', '0', '0', '1', '0', '0', '0', '0', '0', '0',
       '0', '0', '0', '1', '1', '1', '1', '0', '1', '0', '1', '0', '0',
       '1', '1', '1', '0', '0', '0', '1', '0', '0', '0', '0', '0', '0',
       '1', '0', '1', '0', '0', '0', '0', '0', '1', '0', '1', '0', '0',
```

TRAIN –TEST-SPLIT

```
[ ]  from sklearn.model_selection import train_test_split
     X_train1, X_test1, y_train1, y_test1 = train_test_split(X, y, test_size=0.40, random_state=42)
     X_train2, X_test2, y_train2, y_test2 = train_test_split(X, y, test_size=0.30, random_state=42)
     X_train3, X_test3, y_train3, y_test3 = train_test_split(X, y, test_size=0.25, random_state=42)
     X_train4 ,X_test4, y_train4, y_test4 = train_test_split(X, y, test_size=0.20, random_state=42)
```

KNN AFTER VIF

60-40

```
model.fit(X_train1_nomulti, y_train1_nomulti)
```

KNeighborsClassifier

```
KNeighborsClassifier(n_neighbors=25)
```

```
[ ]  y_pred1_nomulti = model.predict(X_test1_nomulti)
     y_pred1_nomulti
```

```
array(['0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0',
       '0', '1', '1', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0',
       '1', '1', '0', '0', '1', '1', '0', '1', '0', '1', '1', '0', '0',
       '1', '1', '1', '0', '1', '0', '1', '0', '0', '0', '0', '0', '0',
       '0', '0', '0', '0', '1', '0', '1', '0', '1', '1', '1', '0', '0',
       '0', '1', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0',
       '0', '0', '1', '1', '0', '0', '0', '0', '1', '1', '1', '0', '1',
       '0', '1', '0', '0', '0', '1', '0', '0', '0', '0', '1', '1', '0',
       '0', '0', '0', '1', '0', '1', '0', '0', '0', '0', '1', '1', '1',
       '1', '0', '1', '0', '0', '1', '0', '0', '0', '0', '1', '1', '1',
       '1', '0', '0', '0', '1', '1', '1', '1', '0', '1', '0', '0', '1',
       '1', '0', '0', '0', '0', '0', '0', '0', '0', '0', '1', '0', '1',
       '1', '0', '0', '0', '1', '0', '0', '0', '0', '0', '0', '1', '0',
       '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '1', '0',
       '0', '0', '1', '0', '1', '0', '0', '0', '0', '1', '0', '0', '0',
```

SVM AFTER VIF

60-40

```
[ ]  model1 = SVC(kernel='linear')
```

```
model1.fit(X_train1_nomulti, y_train1_nomulti)
```

SVC

```
SVC(kernel='linear')
```

```
[ ]  y_pred1_nomulti = model1.predict(X_test1_nomulti)
```

```
y_pred1_nomulti
```

```
array(['0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0',
       '0', '1', '1', '0', '0', '0', '1', '0', '0', '0', '1', '0', '0',
       '1', '1', '0', '0', '1', '1', '0', '1', '0', '1', '1', '1', '0',
       '1', '1', '1', '0', '1', '0', '1', '0', '0', '0', '0', '0', '1',
       '0', '0', '0', '0', '1', '0', '1', '0', '1', '1', '1', '0', '0',
       '1', '1', '0', '1', '0', '1', '0', '0', '0', '0', '0', '0', '0',
       '0', '0', '1', '1', '1', '0', '0', '0', '1', '1', '1', '1', '1',
       '0', '1', '0', '0', '1', '1', '0', '0', '0', '0', '1', '1', '0',
```

ADABOOST AFTER VIF

60-40

```
adaboost.fit(X_train1_nomulti, y_train1_nomulti)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_weight_
  warnings.warn(
```

AdaBoostClassifier

estimator: DecisionTreeClassifier

DecisionTreeClassifier

```
y_pred1_nomulti = adaboost.predict(X_test1_nomulti)
print("Accuracy:",accuracy_score(y_test1_nomulti,y_pred1_nomulti))
print(classification_report(y_test1_nomulti, y_pred1_nomulti))
print(confusion_matrix(y_test1_nomulti, y_pred1_nomulti))
```

```
Accuracy: 0.7137681159420289
              precision    recall  f1-score   support

           0       0.73      0.76      0.74       151
           1       0.69      0.66      0.67       125
```

XGBOOST AFTER VIF

```
y_train1_nomulti = y_train1_nomulti.astype('int')
y_train2_nomulti = y_train2_nomulti.astype('int')
y_train3_nomulti = y_train3_nomulti.astype('int')
y_train4_nomulti = y_train4_nomulti.astype('int')
y_test1_nomulti = y_test1_nomulti.astype('int')
y_test2_nomulti = y_test2_nomulti.astype('int')
y_test3_nomulti = y_test3_nomulti.astype('int')
y_test4_nomulti = y_test4_nomulti.astype('int')
```

60-40

```
model1.fit(X_train1_nomulti, y_train1_nomulti)
model2.fit(X_train1_nomulti,y_train1_nomulti)
```

XGBClassifier

```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=0.1, max_bin=None
```

DECISION TREE AFTER VIF

60-40

```
[ ]  clf = DecisionTreeClassifier()
     clf = clf.fit(X_train1_nomulti,y_train1_nomulti)

[ ]  y_pred1_nomulti = clf.predict(X_test1_nomulti)

[ ]  print("Accuracy:",metrics.accuracy_score(y_test1_nomulti, y_pred1_nomulti))
```

```
Accuracy: 0.7210144927536232
```

```
# Create Decision Tree classifer object
clf = DecisionTreeClassifier(criterion="entropy", max_depth=3)

# Train Decision Tree Classifer
clf = clf.fit(X_train1_nomulti,y_train1_nomulti)

#Predict the response for test dataset
y_pred1_nomulti = clf.predict(X_test1_nomulti)

# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test1_nomulti, y_pred1_nomulti))
```

RANDOM FOREST AFTER VIF

60-40

```
[ ]  rf.fit(X_train1_nomulti,y_train1_nomulti)
```

```
RandomForestClassifier ❶ ❷
RandomForestClassifier()
```

```
y_pred_train1_nomulti=rf.predict(X_test1_nomulti)
print("Accuracy:",accuracy_score(y_test1_nomulti,y_pred1_nomulti))
print(classification_report(y_test1_nomulti, y_pred1_nomulti))
print(confusion_matrix(y_test1_nomulti, y_pred1_nomulti))
```

```
Accuracy: 0.7210144927536232
              precision    recall  f1-score   support

           0       0.73      0.77      0.75       151
           1       0.71      0.66      0.68       125

    accuracy                           0.72       276
   macro avg       0.72      0.72      0.72       276
weighted avg       0.72      0.72      0.72       276
```

LOGISTIC REGRESSION AFTER VIF

60-40

```
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression(C=1e9)
logreg.fit(X_train1_nomulti, y_train1_nomulti)
predictions1 = logreg.predict(X_test1_nomulti)
print(predictions1)
```

```
['0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '1' '0' '0' '1' '1' '0' '0'
 '0' '1' '0' '0' '0' '1' '0' '0' '1' '1' '0' '0' '1' '1' '0' '1' '0' '1'
 '1' '1' '0' '1' '1' '1' '0' '1' '0' '1' '0' '0' '0' '0' '0' '1' '0' '0'
 '1' '0' '1' '0' '1' '0' '1' '1' '1' '0' '0' '1' '1' '0' '1' '0' '1' '0'
 '0' '0' '0' '0' '0' '0' '1' '1' '1' '1' '1' '0' '1' '0' '1' '1' '1' '1'
 '1' '0' '1' '0' '0' '1' '1' '0' '0' '0' '0' '1' '1' '0' '1' '0' '0' '1'
 '0' '1' '0' '1' '1' '0' '0' '0' '1' '1' '1' '1' '0' '0' '1' '1' '0' '0'
 '0' '1' '1' '1' '1' '0' '0' '0' '1' '1' '1' '1' '0' '1' '0' '0' '1' '1'
 '0' '0' '0' '1' '0' '0' '0' '0' '0' '1' '1' '1' '1' '0' '0' '1' '1' '0'
 '0' '0' '0' '0' '0' '1' '0' '0' '0' '0' '1' '0' '0' '0' '0' '0' '0' '0'
 '1' '1' '1' '1' '1' '0' '0' '1' '0' '1' '0' '1' '0' '0' '0' '1' '1' '1'
 '1' '1' '0' '1' '0' '1' '1' '0' '1' '1' '0' '0' '1' '1' '1' '0' '0' '1'
 '0' '0' '1' '0' '0' '1' '1' '0' '1' '0' '0' '0' '0' '1' '0' '0' '0' '0'
 '0' '0' '1' '0' '1' '0' '0' '1' '0' '0' '0' '1' '0' '1' '1' '1' '0' '0'
 '1' '0' '1' '1' '0' '1' '1' '0' '1' '1' '0' '1' '0' '1' '0' '0' '0' '1'
 '1' '0' '0' '1' '0' '1']
```

XGBOOST BEFORE VIF

```
[ ] import xgboost as xgb
```

```
[ ] model1 = xgb.XGBClassifier()
    model2 = xgb.XGBClassifier(n_estimators=100, max_depth=8, learning_rate=0.1, subsample=0.5)
```

```
    y_train1 = y_train1.astype('int')
    y_train2 = y_train2.astype('int')
    y_train3 = y_train3.astype('int')
    y_train4 = y_train4.astype('int')
    y_test1= y_test1.astype('int')
    y_test2= y_test2.astype('int')
    y_test3= y_test3.astype('int')
    y_test4= y_test4.astype('int')
```

60-40

```
[ ] model1.fit(X_train1, y_train1)
    model2.fit(X_train1,y_train1)
```

XGBClassifier

CHECKING FOR MULTI COLLINEARITY (VIF)

```
[ ] from statsmodels.stats.outliers_influence import variance_inflation_factor

    def calc_vif(X):

        # Calculating VIF
        vif = pd.DataFrame()
        vif["variables"] = X.columns
        vif["VIF"] = [variance_inflation_factor(X.values, i).round(1) for i in range(X.shape[1])]

        return(vif)

    calc_vif(X)
```

```
/usr/local/lib/python3.10/dist-packages/statsmodels/stats/outliers_influence.py:197: RuntimeWarning: di
  vif = 1. / (1. - r_squared_i)
```

	variables	VIF
0	Age	9.2
1	Debt	2.4

RANDOM FOREST BEFORE VIF

60-40

```
[ ]  from sklearn.ensemble import RandomForestClassifier
     from sklearn.tree import plot_tree

[ ]  rf = RandomForestClassifier()

[ ]  rf.fit(X_train1,y_train1)
```

RandomForestClassifier

RandomForestClassifier()

```
[ ]  y_pred1=rf.predict(X_test1)
     print("Accuracy:",accuracy_score(y_test1,y_pred1))
```

```
Accuracy: 0.8586956521739131
```

```
[ ]  print(classification_report(y_test1, y_pred1))
     print(confusion_matrix(y_test1, y_pred1))
```

ADABOOST BEFORE VIF

```
[ ]  from sklearn.ensemble import AdaBoostClassifier
     from sklearn.ensemble import AdaBoostClassifier
     from sklearn.tree import DecisionTreeClassifier

[ ]  # Replace 'base_estimator' with 'estimator'
     base_estimator = DecisionTreeClassifier(max_depth=3, random_state=0)
     adaboost = AdaBoostClassifier(estimator=base_estimator, # Changed argument name here
                                   n_estimators=3,random_state=0)
```

60-40

```
[ ]  adaboost.fit(X_train1, y_train1)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_weight_boosting.py:527: Fut
  warnings.warn(
```

AdaBoostClassifier

estimator: DecisionTreeClassifier

DecisionTreeClassifier

SVM BEFORE VIF

```
from sklearn.svm import SVC
```

```
model1 = SVC(kernel='linear')
```

```
model1.fit(X_train1, y_train1)
```

```
SVC(kernel='linear')
```

```
y_pred1 = model1.predict(X_test1)
```

```
y_pred1
```

```
array(['0', '1', '0', '0', '0', '0', '0', '0', '0', '0', '0', '1', '0',
       '0', '0', '1', '0', '0', '0', '0', '0', '0', '0', '1', '0', '0',
       '1', '1', '0', '0', '1', '1', '1', '1', '1', '1', '1', '1', '1',
       '1', '1', '1', '0', '1', '0', '1', '0', '0', '1', '0', '0', '1',
       '0', '0', '1', '0', '1', '0', '1', '0', '1', '1', '1', '1', '0',
       '1', '1', '1', '1', '0', '0', '1', '0', '0', '0', '0', '0', '1',
       '1', '0', '1', '1', '0', '0', '1', '0', '1', '1', '1', '1', '1',
```

DECISION TREE BEFORE VIF

60-40

```
from sklearn.tree import DecisionTreeClassifier
```

```
clf = DecisionTreeClassifier()
clf = clf.fit(X_train1,y_train1)
```

```
y_pred1 = clf.predict(X_test1)
```

```
print("Accuracy:",metrics.accuracy_score(y_test1, y_pred1))
```

```
Accuracy: 0.8115942028985508
```

```
clf = DecisionTreeClassifier(criterion="entropy", max_depth=3)
clf = clf.fit(X_train1,y_train1)
y_pred1 = clf.predict(X_test1)
print("Accuracy:",metrics.accuracy_score(y_test1, y_pred1))
```

```
Accuracy: 0.8405797101449275
```

NEURAL NETWORKS

60-40

Before VIF

```
tf.random.set_seed(42)

# STEP1: Creating the model
model1= tf.keras.Sequential([
                             tf.keras.layers.Dense(30, activation='relu'),
                             tf.keras.layers.Dense(20, activation='relu'),
                             tf.keras.layers.Dense(10, activation='relu'),
                             tf.keras.layers.Dense(1, activation='sigmoid')
])

# STEP2: Compiling the model

model1.compile(loss= tf.keras.losses.binary_crossentropy,
               optimizer= tf.keras.optimizers.Adam(learning_rate=0.001),
               metrics= [tf.keras.metrics.BinaryAccuracy(name='accuracy'),
                         tf.keras.metrics.Precision(name='precision'),
                         tf.keras.metrics.Recall(name='recall')
               ]
               )

# STEP3: Fit the model

history1= model1.fit(X_train1, y_train1, epochs= 150)
```

```
Epoch 1/150
13/13 ━━━━━━━━━━━━━━━━━━━━ 5s 16ms/step - accuracy: 0.5671 - loss: 17.4130 - precision: 0.3923 - recall: 0.0807
Epoch 2/150
13/13 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - accuracy: 0.6010 - loss: 1.6047 - precision: 0.5515 - recall: 0.6752
Epoch 3/150
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.6846 - loss: 1.5115 - precision: 0.6705 - recall: 0.6057
Epoch 4/150
13/13 ━━━━━━━━━━━━━━━━━━━━ 1s 17ms/step - accuracy: 0.6348 - loss: 0.8097 - precision: 0.7741 - recall: 0.2792
Epoch 5/150
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.6729 - loss: 0.8368 - precision: 0.7608 - recall: 0.4172
Epoch 6/150
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.6489 - loss: 0.7469 - precision: 0.7825 - recall: 0.3215
Epoch 7/150
```