Predicting Credit Card Approvals:

A Classification-Based Approach

PRESENTED BY: **GROUP NUMBER 2**

BHAVANS VIVEKANANDA DEGREE COLLEGE

GROUP MEMBERS:

HARSHIT KUMAR/SRIKANTH YADAV/CHAITANYA JADAV

ABSTRACT

•**Purpose**: TO Analyze applicant profiles to understand factors affecting credit approval decisions and **help Financial Institutes** make better decisions

•**Focus Attributes**: **Age, Income, Debt levels, Employment stability, Credit history.**

•**Goal**: Identify patterns and key drivers behind credit approvals for data-driven decision-making.

•**Outcome** : Support development of **fair, transparent, and equitable lending** practices.

Provide actionable insights to **improve credit scoring models**.

Help financial institutions balance **risk management** with **inclusive credit access**.

OBJECTIVES

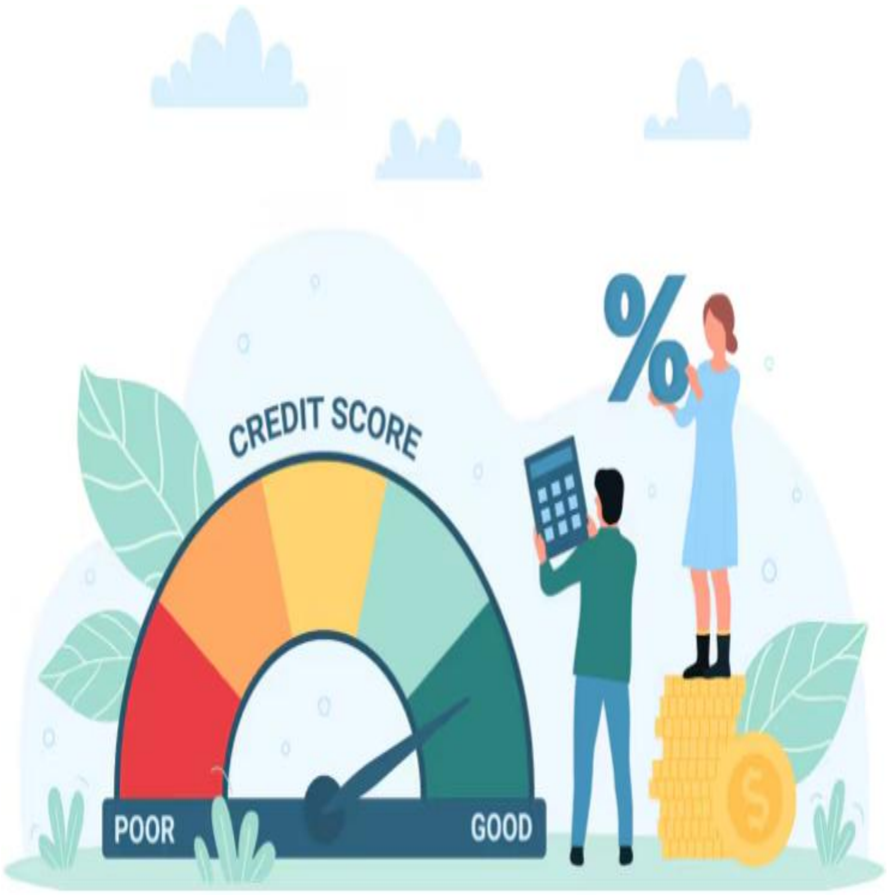
•**Identify Key Drivers**: Determine which factors (e.g., income, debt, credit history) most significantly impact credit approval decisions.

•**Support Fair Lending**: Analyze trends to ensure that credit decisions are made equitably and without bias.

•**Improve Credit Scoring Models**: Use insights from data to enhance existing credit scoring models, leading to more accurate and inclusive approval processes.

•**Enhance Transparency**: Provide data-backed insights to foster a transparent credit approval process that stakeholders can understand and trust.

•**Assist Risk Management**: Help financial institutions mitigate risk by providing a clearer understanding of applicant profiles and their creditworthiness.



CONTENT

INTRODUCTION

•**Credit Approval Process**: A vital step for financial institutions to determine an applicant's eligibility for credit.

•**Importance**: Ensures responsible lending by evaluating **applicants' financial profiles**.

•Helps in maintaining **financial stability** by reducing credit risk.

Promotes **fair access** to credit for diverse applicant backgrounds.

•**Dataset**: Comprises detailed applicant information, including **age, income, debt levels, employment stability, and credit history**.

•**Focus**: Understanding applicant characteristics that influence credit decisions.

Highlighting data-driven trends that align with fair and transparent lending practices.

LITERATURE REVIEW

LITERATURE REVIEW- 1

Peela, H.V., Gupta, T., Rathod, N., Bose, T. and Sharma, N., Prediction of Credit Card Approval. **International Journal of Soft Computing and Engineering.**

Peela et al. (2021) analyzed machine learning approaches for predicting credit card approvals, comparing models such as **logistic regression, decision trees, and random forests.** Their study focuses on identifying applicant features most influential in approval decisions and demonstrates that ensemble methods, especially **random forests**, **excels in predictive accuracy**. The research underscores the effectiveness of machine learning in **simplifying and improving the credit approval process**, emphasizing the role of feature selection in **optimizing model performance.**

LITERATURE REVIEW- II

Nandipati, V.S.S. and Boddala, L.V., 2024. Credit Card Approval Prediction: A comparative analysis between Logistic Regression, KNN, Decision Trees, Random Forest, XGBoost.

Nandipati and Boddala (2024) present a comparative analysis of machine learning models—**logistic regression, KNN, decision trees, random forests, and XGBoost**—for predicting credit card approvals. Their study assesses each model's accuracy, interpretability, and efficiency, finding that while **XGBoost offers the highest predictive accuracy**, logistic regression remains useful due to its interpretability. The authors suggest that although advanced models provide accuracy gains, simpler models like logistic regression and decision trees can be advantageous in contexts where interpretability and processing speed are prioritized for practical credit approval decisions.

DATA PREPROCESSING

DATA

Data Set : There are 16 variables and 691 records

SOURCE:https://drive.google.com/drive/folders/1vGSRCnhqSxEH53BgLqhh32F1qNKqfOim?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

VARIABLE

The variables are of two types:
continuous and categorical data, where the continuous variables are in integer and float format and categorical variables are in object format

CATEGORICAL	CONTINUOUS
Gender	Age
Married	Debt
Bankcustomer	Yearsemployed
Industry	CreditScore
Ethinicity	Income
Priordefault	
Emplyed	
Driverlicense	
Citizen	

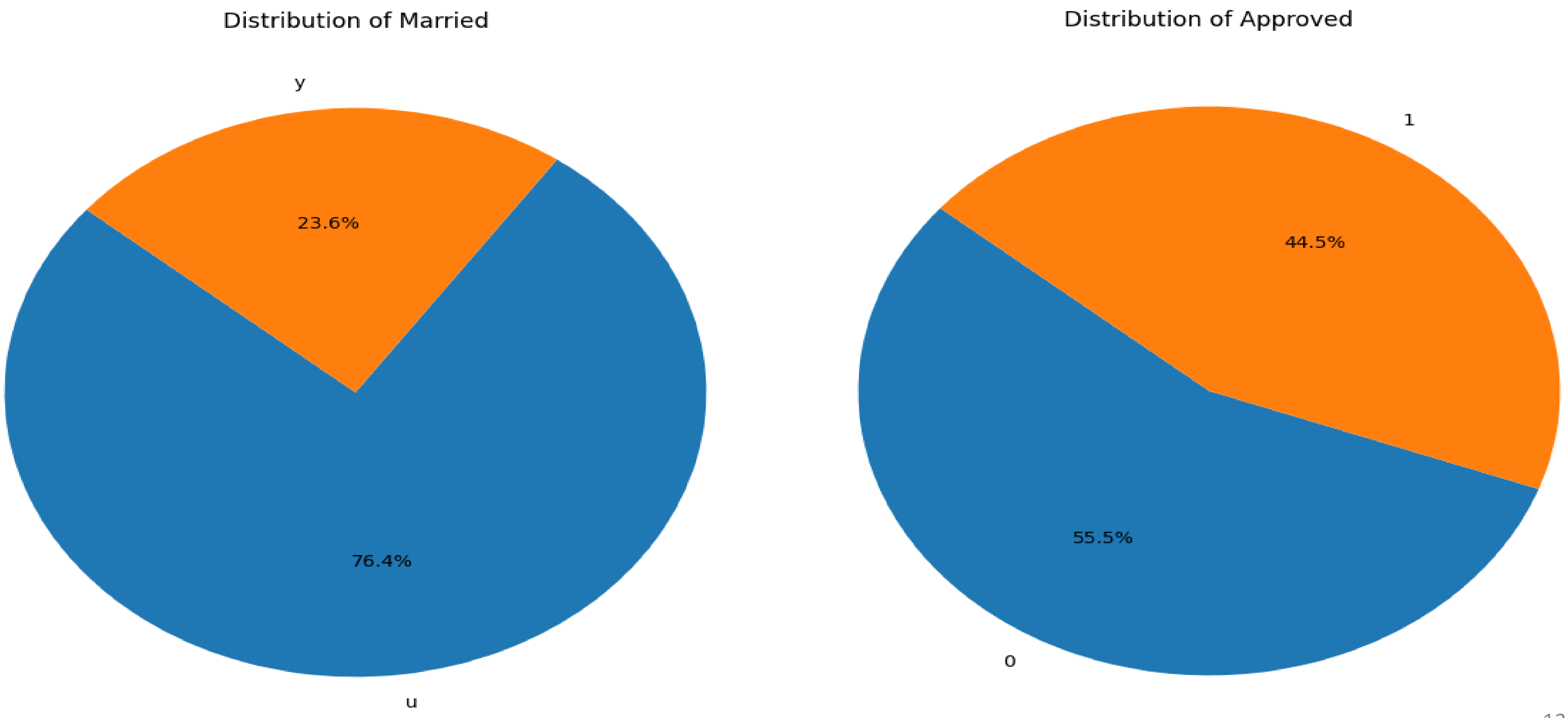
- We have checked for NULL VALUES but there are no null values present in our data.
- Then we proceeded to perform dummy variable encoding for which we divided the data into two parts, which are independent variables and dependent variables.
- The categorical variables are encoded into continuous variables using the dummy variables.
- The renamed columns after dummy variables encoding are shown below,

ORGINAL VARIABLES	RENAMED VARIABLES
Gender	Gender_b
Married	Married_y
Bankcustomer	Bankcustomer_p
Industry	Industry_c, Industry_cc,……
Ethinicity	Ethinicity_h, Ethinicity_j,…..
Priordefault	Priordefault_t
Employed	Employed_t
Driverlicense	Driverlicense_t
Citizen	Citizen_s

EXPLORATORY DATA ANALYSIS

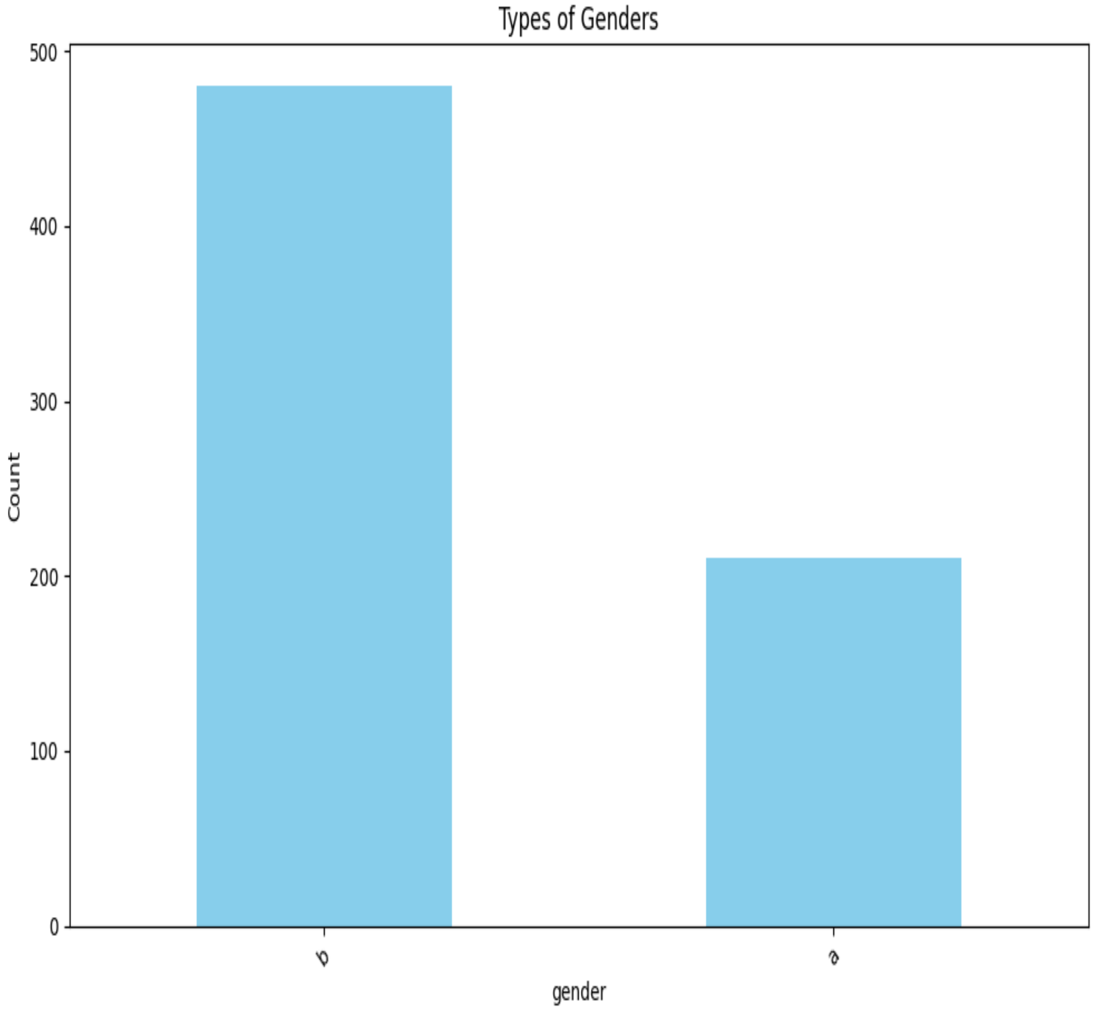
PIECHART

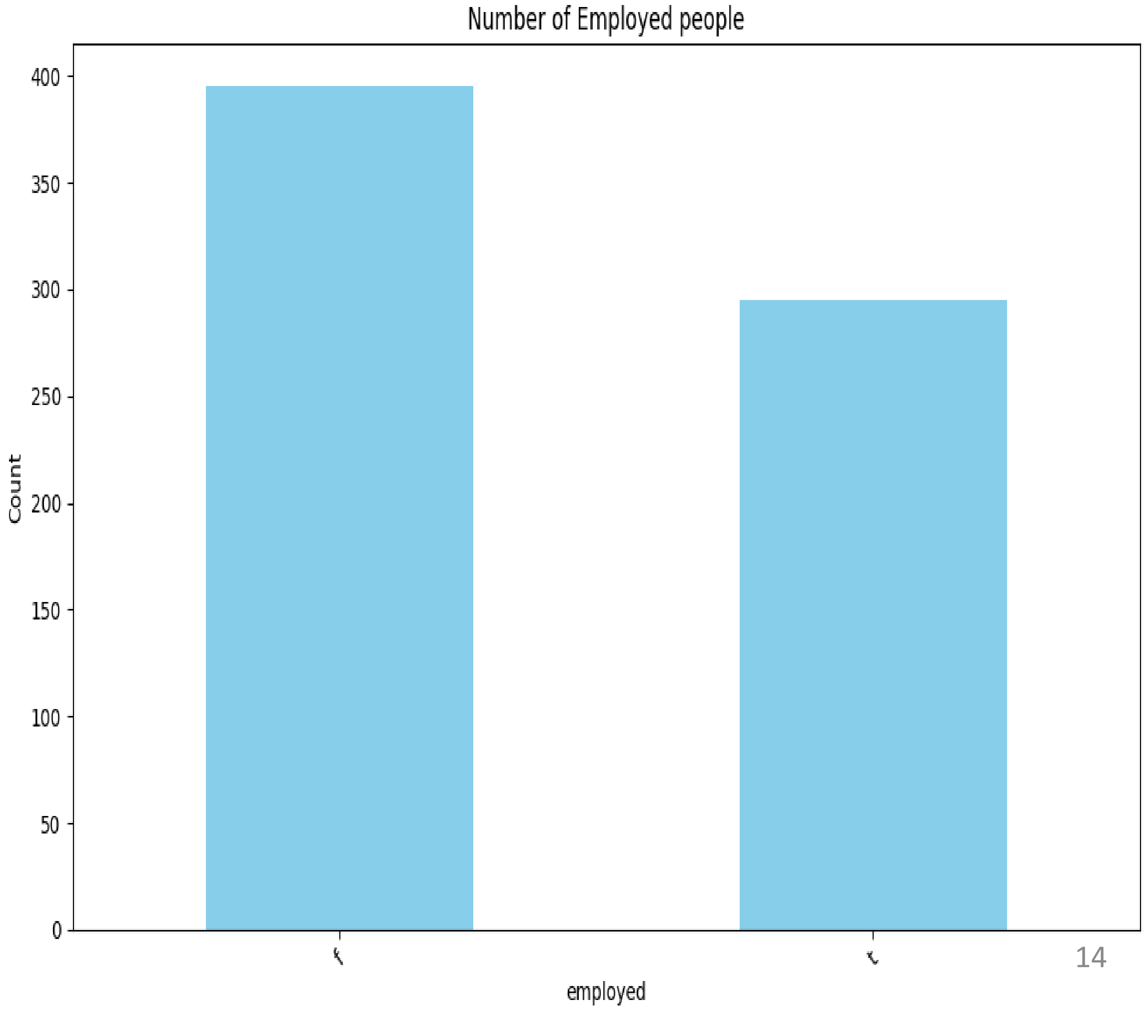
The first pie chart shows number of married people and the second pie chart shows the number of approvals



BARCHART

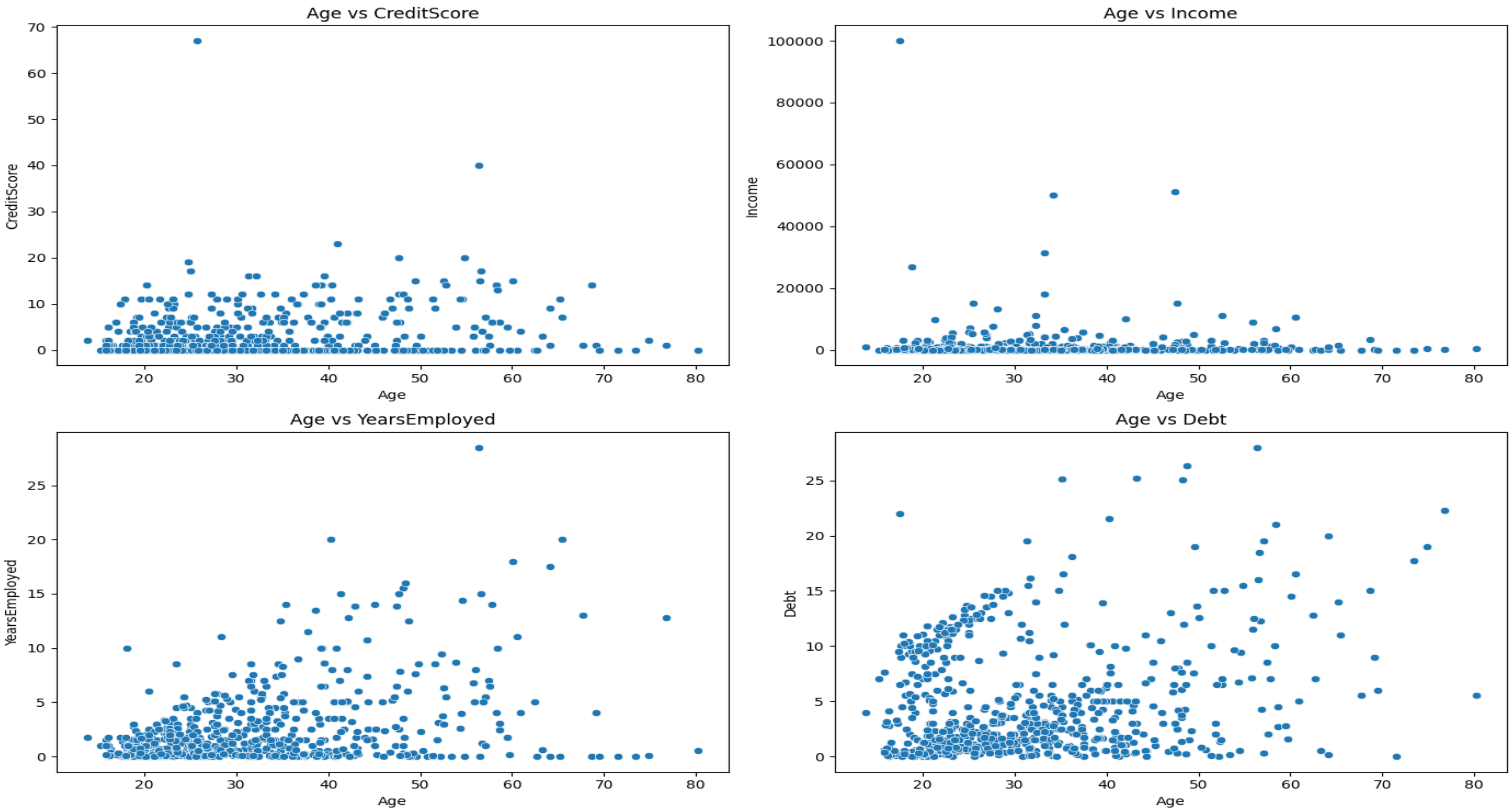
The first bar chart shows the distribution of Gender and the second one shows the distribution of employees





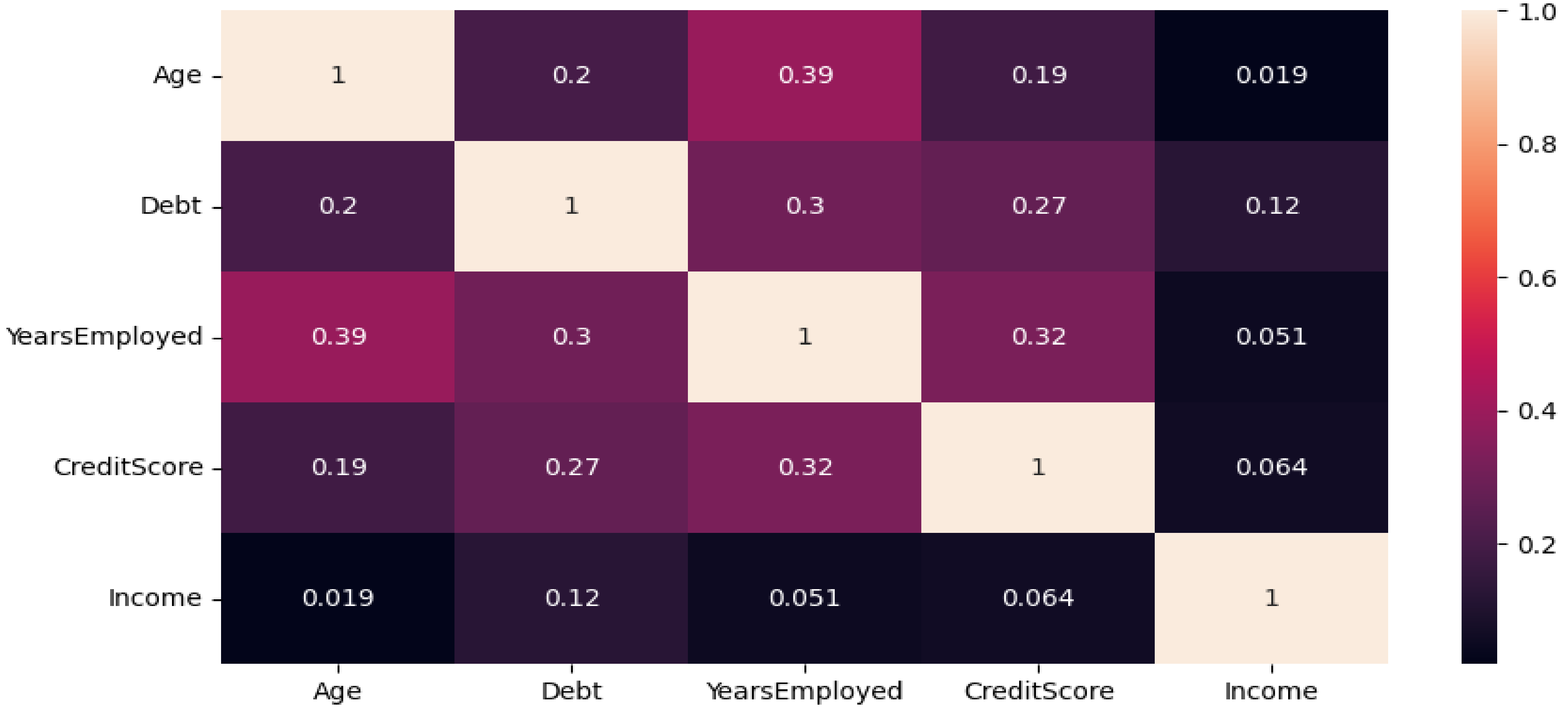
Scatterplot

The target variable “age” is plotted against the independent variables in the scatterplots as shown below



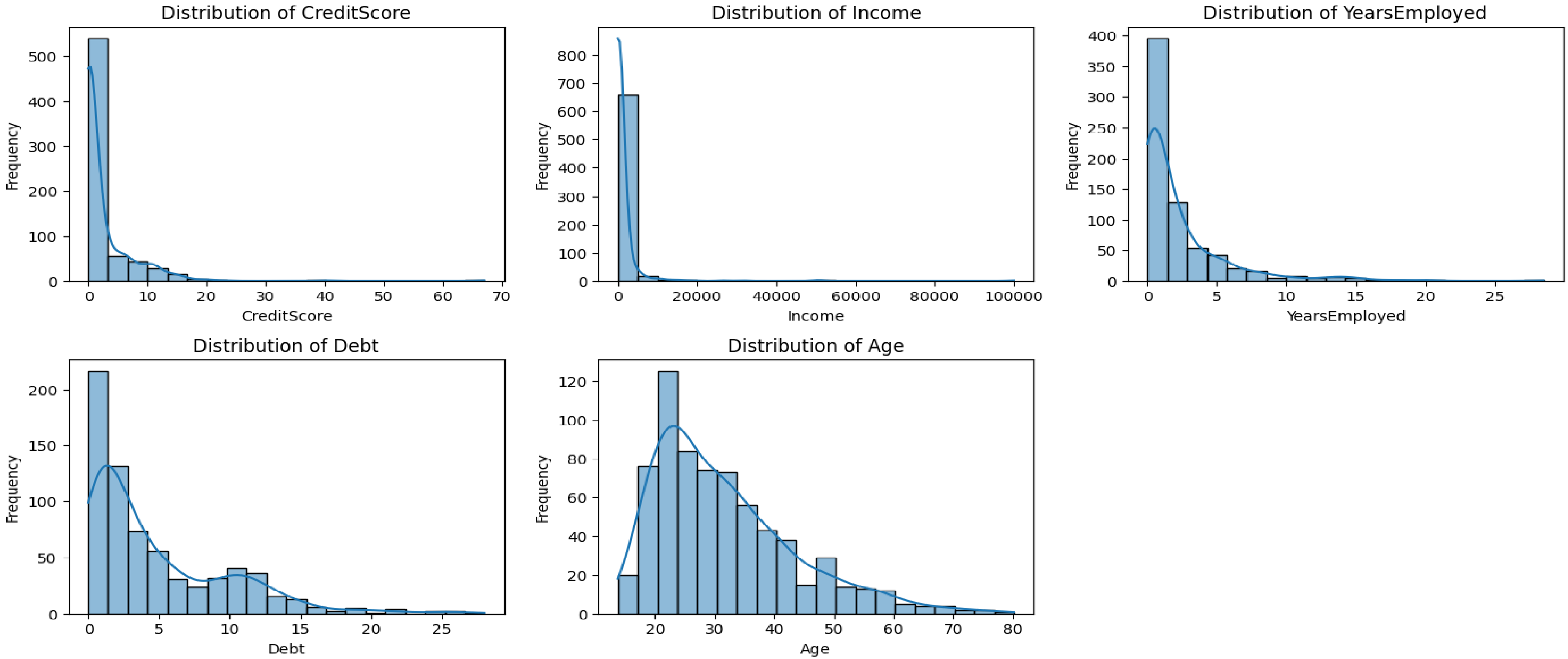
CORRELATION MATRIX

Here we can observe that Age and Years Employed have high positive correlation.
The next most positively correlated are Years Employed and Credit Score



HISTOGRAM

The histograms display the distributions of various financial and demographic features. Here’s a summary interpretation for each plot



MULTICOLLINEARITY CHECK

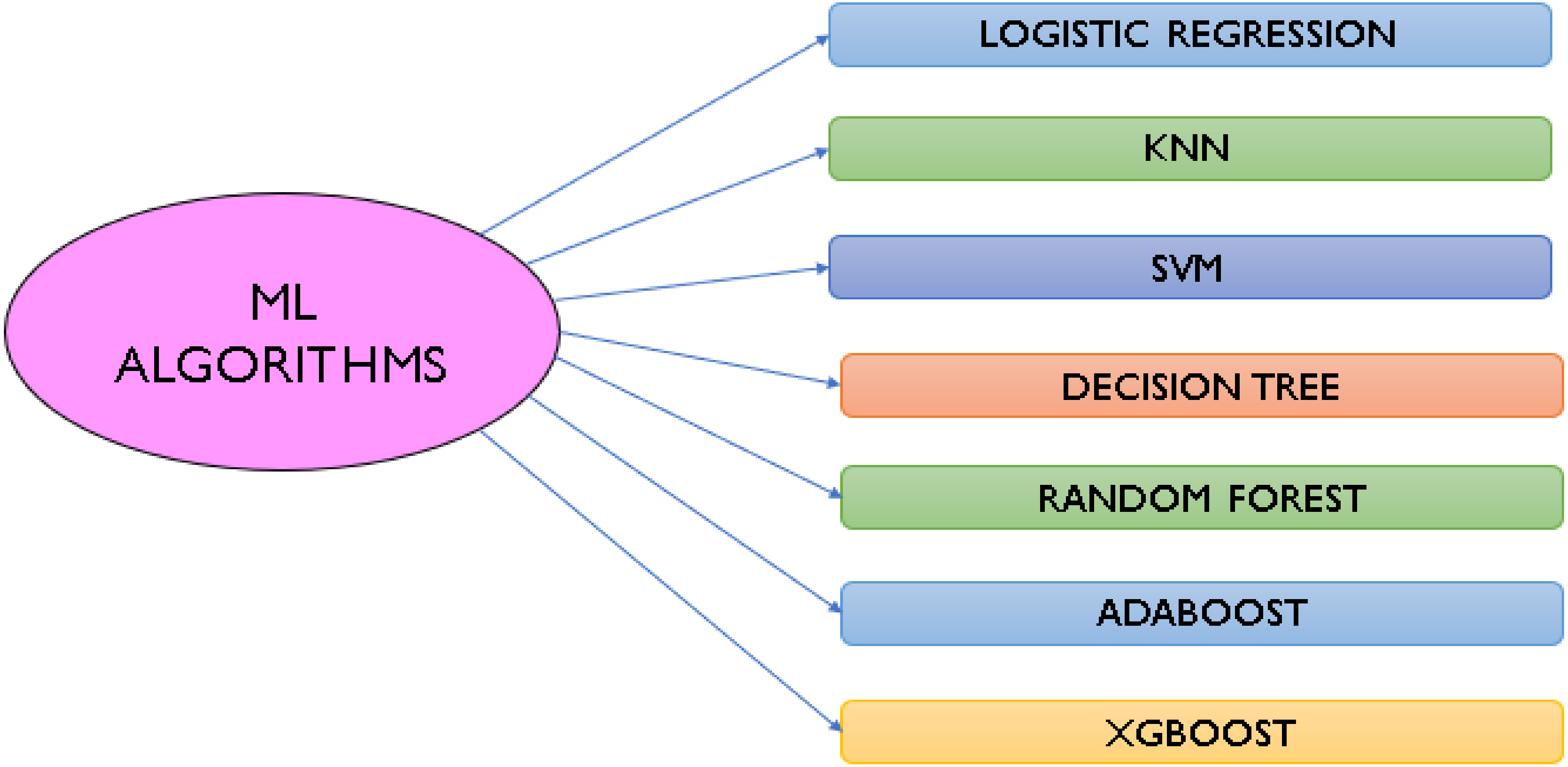
We have removed the variables which are highly correlated to each other, the variables are **'Married_y','Ethnicity_ff','Age','Industry_j','Ethnicity_v','Gender_b','Income','PriorDefault_t'**

	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
29	PriorDefault_t	3.2
30	Employed_t	3.1

	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
26	Citizen_s	1.3

	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
24	Citizen_s	1.2

MACHINE LEARNING ALGORITHM


ML ALGORITHMS
LOGISTIC REGRESSION
KNN
SVM
DECISION TREE
RANDOM FOREST
ADABOOST
XGBOOST

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

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

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

Algorithms Comparison

1. 60-40 Split Before Applying VIF

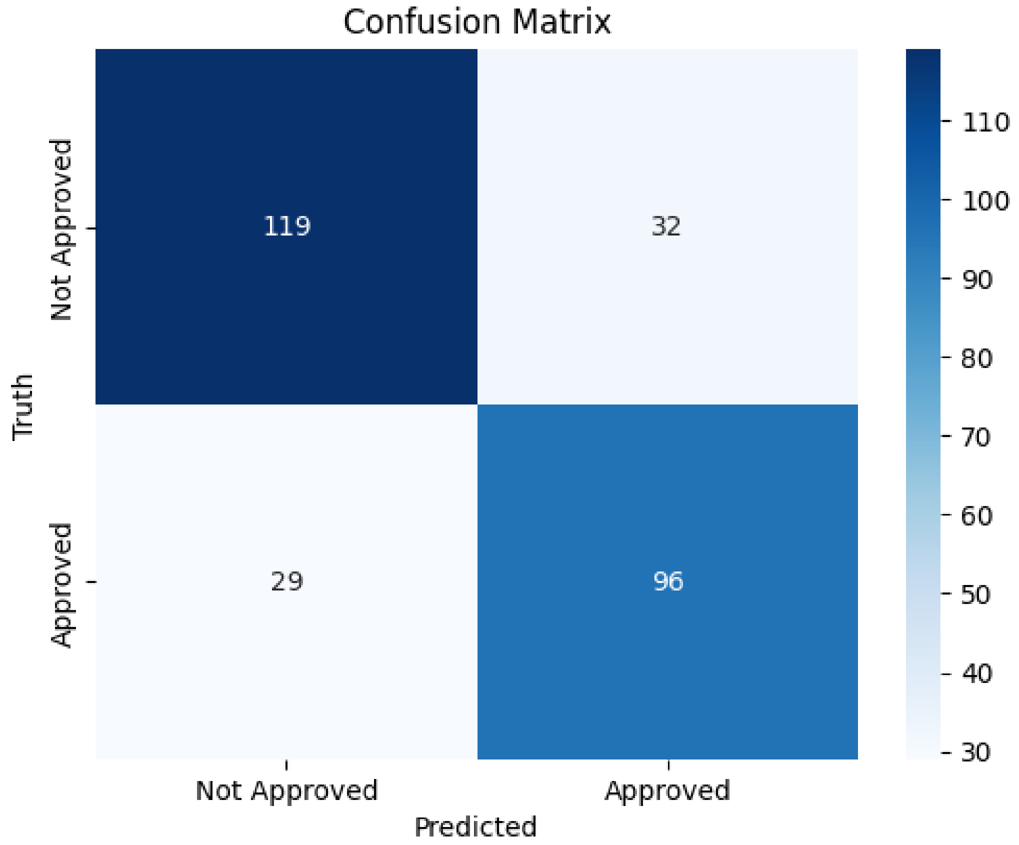
Algorithm	Accuracy
Logistic Regression	0.822
KNN	0.706
SVM	0.836
Decision Tree	0.847
Random Forest	0.865
AdaBoost	0.815
XGBoost	0.862

2. 80-20 Split After Applying VIF

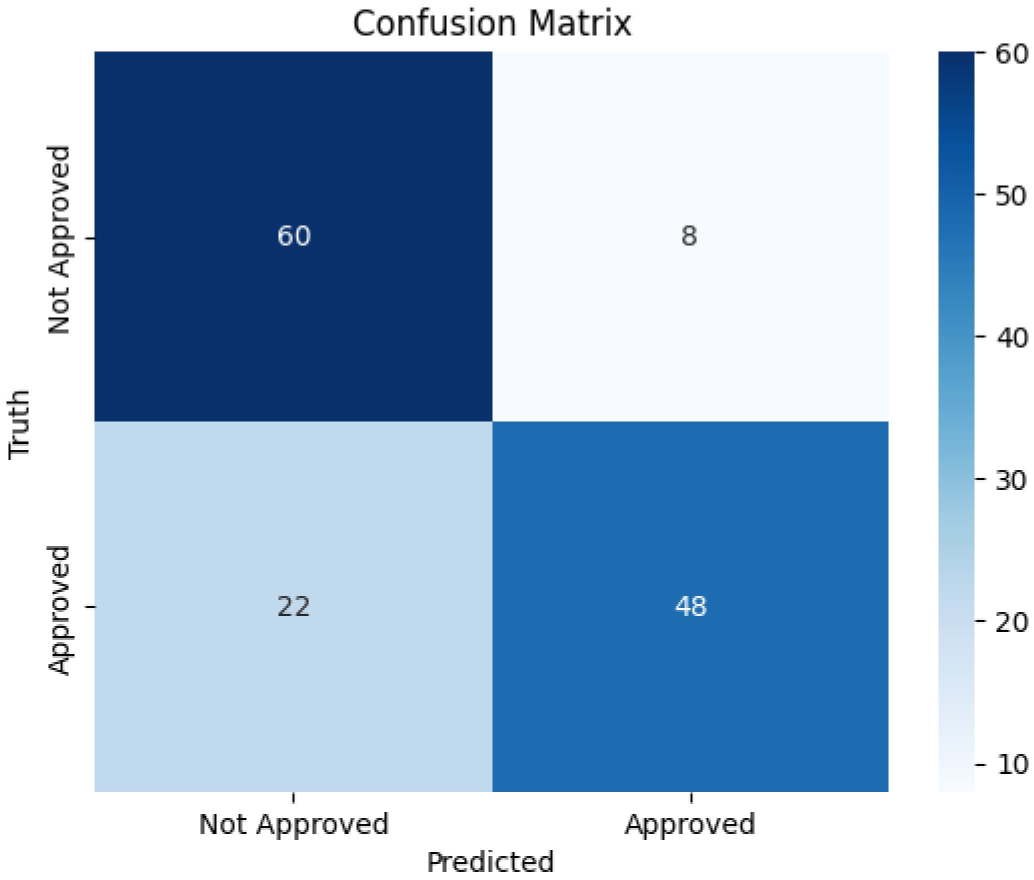
Algorithm	Accuracy
Logistic Regression	0.768
KNN	0.717
SVM	0.782
Decision Tree	0.724
Random Forest	0.673
AdaBoost	0.746
XGBoost	0.775

CONFUSION MATRIX

1. RANDOM FOREST before VIF for 60:40 split



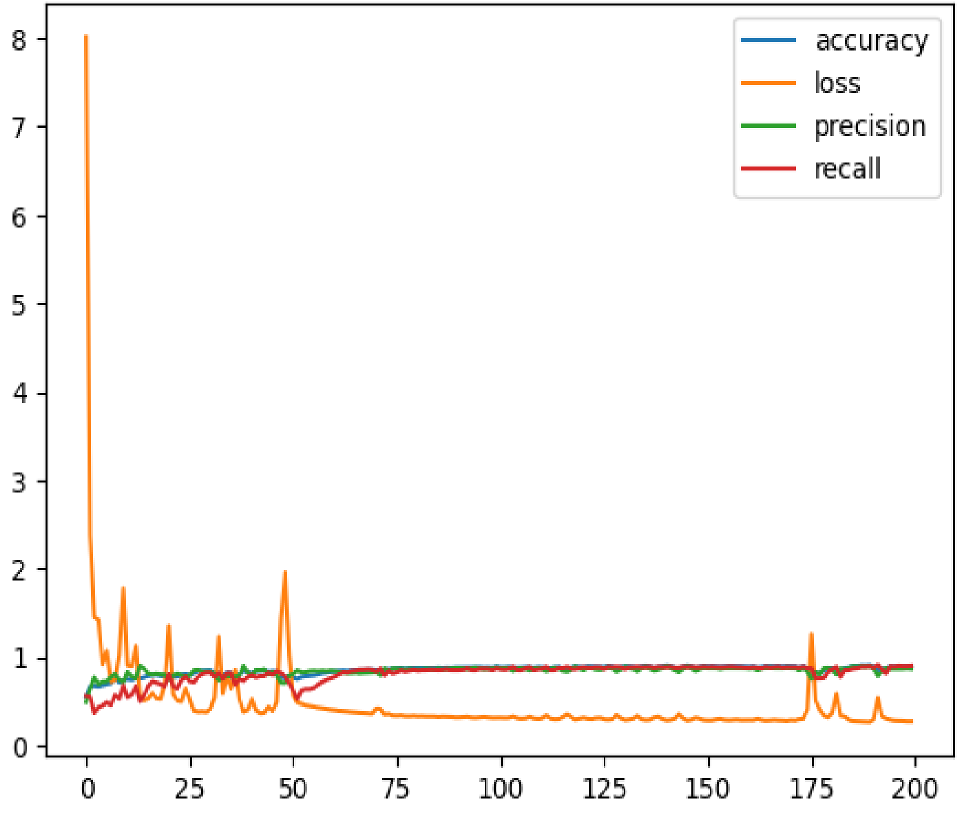
2. SVM after VIF for 80:20 spit



ANN Before VIF

SLPIT	ARCHITECTURE	OPTIMIZER	EPOCHS	ACCURACY
60-40	10-7-5-1	Adam	200	81.58
70-30	10-7-5-1	Adam	150	80.76
75-25	10-7-5-1	Adam	250	77.26
80-20	10-7-5-1	Adam	250	77.58

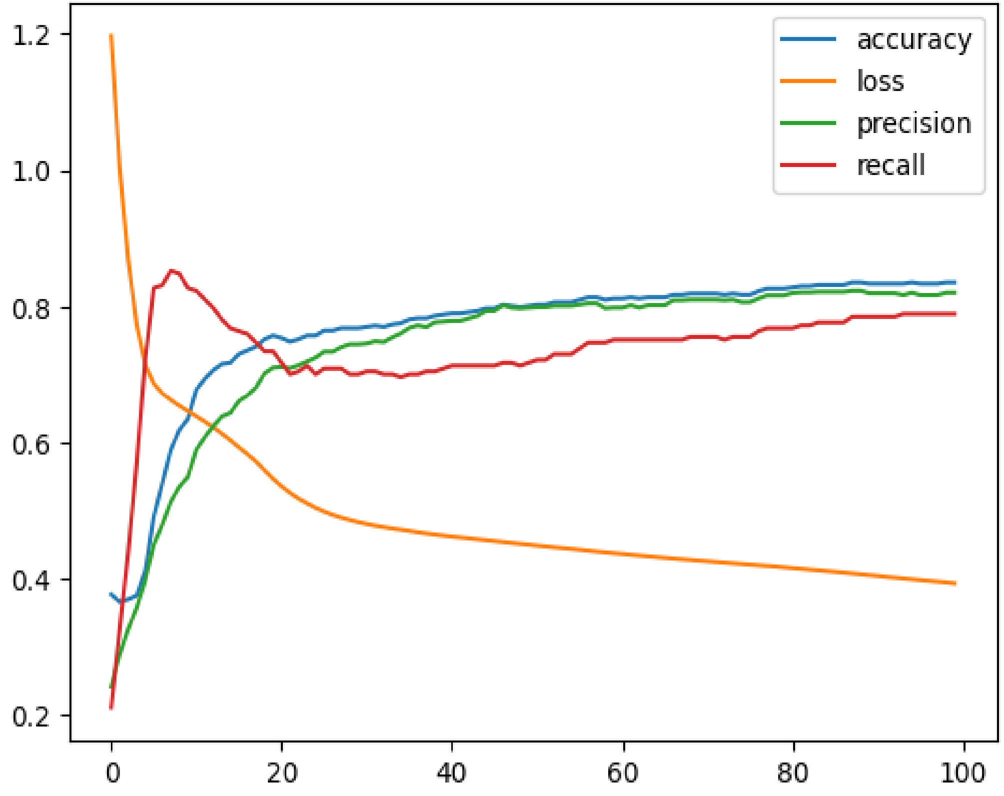
Training Progress: Accuracy, Loss, Precision, and Recall for best split 60-40



ANN After VIF

SLPIT	ARCHITECTURE	OPTIMIZER	EPOCHS	ACCURACY
60-40	10-7-5-1	Adam	250	73.91
70-30	10-7-5-1	Adam	150	76.39
75-25	10-7-5-1	Adam	100	73.77
80-20	10-7-5-1	Adam	250	77.49

Training Progress: Accuracy, Loss, Precision, and Recall for best split 80-20



SUMMARY:

- The purpose of this research is to determine the best-performing machine learning techniques to predict **Credit Card Approval**.
- So 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%**
- From our study we conclude that **random forest (Model-1)** is to be considered for this analysis since it has better accuracy score when compared to **SVM(Model-2)**
- The Artificial Neural Network (ANN) model has demonstrated reliable results (i.e. 81.58 accuracy); however, its accuracy is lower compared to the Machine Learning (ML) models. Therefore, for our dataset and study, the best-performing model is the **ML model (Random Forest)**

INSIGHTS:

•**Correlation Observations**:Variables like **age** and **income** revealed interesting non-linear relationships, hinting that credit risk doesn't increase linearly with age but stabilizes after a certain point.

•**Demographic Trends**:Distribution patterns (e.g., marital status, employment sectors) provide deeper context about the applicant pool, allowing segmentation strategies for lenders.

•**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

FUTURE SCOPE:

- **Integration of Advanced Machine Learning Model**

Future work can explore more complex algorithms, such as **ADVANCED DEEP LEARNING TECHNIQUES** other than **ANN** to further improve the Accuracy of the model especially for challenging cases

- **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

Work Distribution

Team Member 1 (Chaitanya)	Collecting Information about credit card approvals and Literature Review
Team Member 2 (Srikanth)	Data Pre-processing and EDA
Team Member 3 (Harshit)	Implementing ML Algorithms

COLAB NOTEBOOK

THANK YOU

B.HARSHIT KUMAR
SRIKANTH YADAV
CHAITANYA JADAV

APPENDIX

LOADING 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 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

REPLACING THE “?” VALUES IN THE DATA

Replacing Null Values

```python
data['Age'] = pd.to_numeric(data['Age'], errors='coerce')
mean_age = data['Age'].mean()
data['Age'] = data['Age'].replace(to_replace='?', value=str(int(mean_age)))
data['Age'].fillna(mean_age, inplace=True)
```

```
<ipython-input-329-c6897e2531a7>:4: FutureWarning: A value is trying to be set or
The behavior will change in pandas 3.0. This inplace method will never work becau

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.meth


  data['Age'].fillna(mean_age, inplace=True)
```

```python
data['Married'] = data['Married'].replace({'?':'u', 'l':'u'})
data['Gender'] = data['Gender'].replace('?', 'b')
data['BankCustomer'] = data['BankCustomer'].replace({'?':'g', 'gg':'g'})
data['Citizen'] = data['Citizen'].replace('p', 'g')
data['Industry'] = data['Industry'].replace('?', 'c')
data['Ethnicity'] = data['Ethnicity'].replace('?', 'v')
data['Approved'] = data['Approved'].replace('+','1')
data['Approved'] = data['Approved'].replace('-','0')
data.head()
```

	Gender	Age	Debt	Married	BankCustomer	Industry	Ethnicity	YearsEmployed	PriorDefault	Employed	CreditScore	DriversLicense	Citizen	Income	Appro
0	b	30.83	0.000	u	g	w	v	1.25	t	t	1	f	g	0	
1	a	58.67	4.460	u	g	q	h	3.04	t	t	6	f	g	560	
2	a	24.50	0.500	u	g	q	h	1.50	t	f	0	f	g	824	
3	b	27.83	1.540	u	g	w	v	3.75	t	t	5	t	g	3	
4	b	20.17	5.625	u	g	w	v	1.71	t	f	0	f	s	0	

CHECKING THE UNIQUE VALUES AND ITS COUNT

```python
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

```python
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
```

TRAIN –TEST-SPLIT

```python
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)
```

CREATING THE DUMMY VARAIBLES

```python
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

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

60-40

```python
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

```python
from sklearn.neighbors import KNeighborsClassifier
```

```python
model=KNeighborsClassifier(n_neighbors=25)
```

```python
model.fit(X_train1, y_train1)
```

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

```python
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', '0', '1', '0', '0', '0', '0', '0', '1', '0', '1', '0',
```

SVM BEFORE VIF

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

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

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

```
SVC
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
```

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
```

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

TRAIN-TEST-SPLIT AFTER REMOVING HIGH MULTICOLLINEARITY VARIABLS

```python
X_train1_nomulti, X_test1_nomulti, y_train1_nomulti, y_test1_nomulti = train_test_split(X_nomulti, y, test_size=0.40, random_state=42)
X_train2_nomulti, X_test2_nomulti, y_train2_nomulti, y_test2_nomulti = train_test_split(X_nomulti, y, test_size=0.30, random_state=42)
X_train3_nomulti, X_test3_nomulti, y_train3_nomulti, y_test3_nomulti = train_test_split(X_nomulti, y, test_size=0.25, random_state=42)
X_train4_nomulti, X_test4_nomulti, y_train4_nomulti, y_test4_nomulti = train_test_split(X_nomulti, y, test_size=0.20, random_state=42)
```

LOGISTIC REGRESSION AFTER VIF

60-40

```python
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']
```

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',
```

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
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
```

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,
```

DEEP LEARNING MODEL(ANN)

NN before VIF

```
import tensorflow as tf
```

60-40 Train Test Split

Epochs=100 ,optimizer=Adam

```
tf.random.set_seed(42)

# STEP 1: Creating the model
model = tf.keras.Sequential([
    tf.keras.layers.Dense(10, activation='relu'),
    tf.keras.layers.Dense(7, activation='relu'),
    tf.keras.layers.Dense(5, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

# STEP 2: Compiling the model
model.compile(
    loss=tf.keras.losses.binary_crossentropy,
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),  # Corrected here
    metrics=[
        tf.keras.metrics.BinaryAccuracy(name='accuracy'),
        tf.keras.metrics.Precision(name='precision'),
        tf.keras.metrics.Recall(name='recall')  # Removed typo 'a=recall'
    ]
)

# STEP 3: Fit the model
history = model.fit(X_train1, y_train1, epochs=100)
```

```
Epoch 1/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.4989 - loss: 65.2616 - precision: 0.4213 - recall: 0.2542
Epoch 2/100
```

NN after VIF

60-40 Train Test Split

Epochs=100 ,optimizer=Adam

```
import tensorflow as tf
```

```
tf.random.set_seed(42)

# STEP 1: Creating the model
model = tf.keras.Sequential([
    tf.keras.layers.Dense(10, activation='relu'),
    tf.keras.layers.Dense(7, activation='relu'),
    tf.keras.layers.Dense(5, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

# STEP 2: Compiling the model
model.compile(
    loss=tf.keras.losses.binary_crossentropy,
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),  # Corrected here
    metrics=[
        tf.keras.metrics.BinaryAccuracy(name='accuracy'),
        tf.keras.metrics.Precision(name='precision'),
        tf.keras.metrics.Recall(name='recall')  # Removed typo 'a=recall'
    ]
)

# STEP 3: Fit the model
history = model.fit(X_train1_nomulti, y_train1_nomulti, epochs=100)
```

```
Epoch 1/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.6204 - loss: 0.6609 - precision: 0.5773 - recall: 0.6505
```