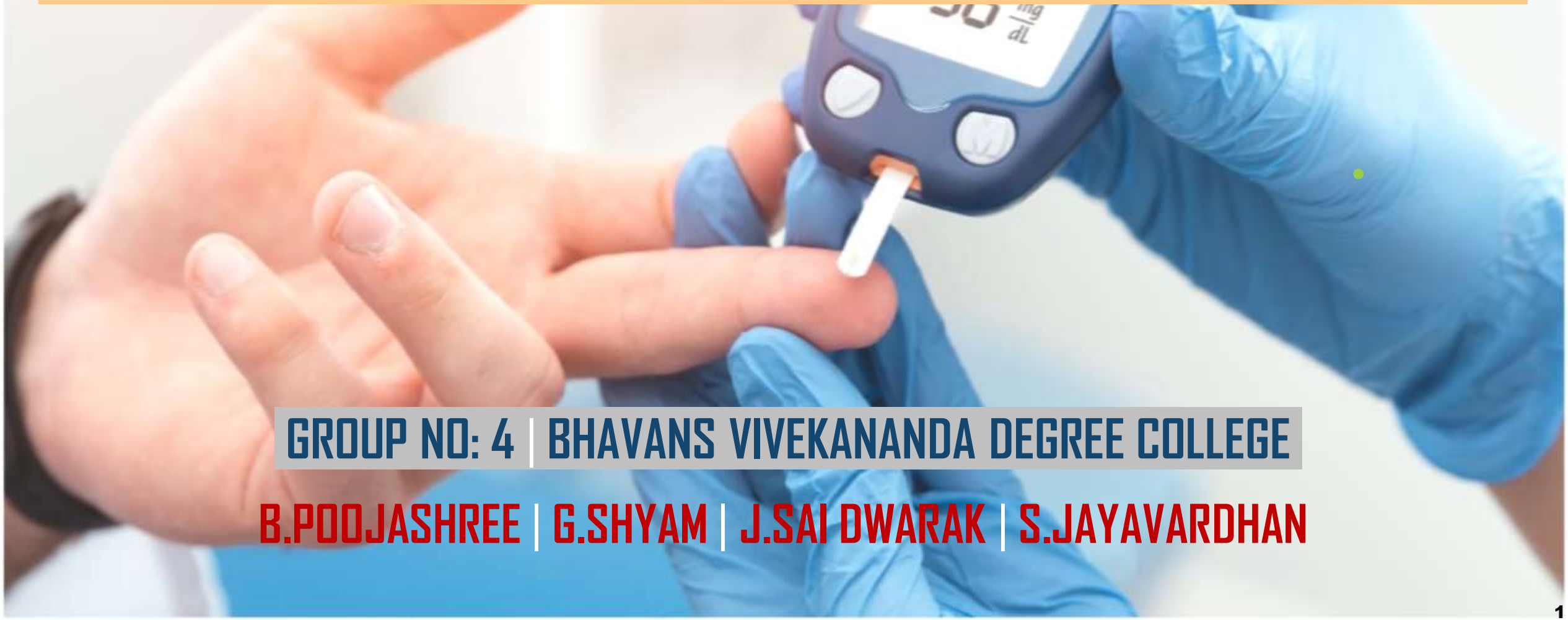
Predicting Diabetes (Classification Project) Using Machine Learning Algorithms



GROUP NO: 4 | BHAVANS VIVEKANANDA DEGREE COLLEGE

B.POOJASHREE | G.SHYAM | J.SAI DWARAK | S.JAYAVARDHAN

Abstract

The project focuses on developing a model that accurately predicts diabetes in a person, leading to early prevention and care.

The study explores machine learning algorithms like Logistic Regression, Decision Tree, Random Forest, K-Nearest Neighbors, Support Vector Machines, and Boosting to predict diabetes.

Objective

To find the suitable machine learning model to predict diabetes in a person for giving proper medication and early prevention.

Content

Introduction

- Diabetes, a growing health crisis, demands early detection for effective management.
- Traditional diagnosis methods are often time-consuming and resource-intensive.
- Machine learning offers a promising solution by analyzing patient data to predict diabetes risk.
- This project aims to leverage machine learning algorithms to develop a predictive model to assist healthcare professionals.

Literature Review

Literature Review-1

Aishwarya Mujumudar; Dr. Vaidehi V - 2nd International Conference on Recent Trends in Advanced Computing
ICRTAC -DISRUP - TIV INNOVATION

Aishwarya et al.- In November 2019 Various ML algorithms were applied to the diabetes dataset from which Logistic Regression gave 96% accuracy and Adaptive Boosting gave 98.8% accuracy.

Literature Review-2

Jobeda Jamal Khanam, Simon Y. Foo -ICT Express, Volume 7, Issue 4, 2021,

Jobeda et al.– Analysed the diabetes data using an Artificial Neural Network with three hidden layers and 400 epochs, providing an accuracy of 88.6%

Literature Review-3

Sana Maqbool; Imran Sarwar Bajwa - AI-Enabled Smart Sensing Technologies for Human-Centered Healthcare Applications

Sana et al.– 1 December 2023 Used Random forest for analyzing diabetes data which gave 98% accuracy.

Literature Review-4

12th International Conference on Computing Communication and Networking Technologies (ICCCNT). IEEE, 2021.

Md.Mehedi Hassan et al. – Utilized Unsupervised and supervised approaches which yielded 99.57% and 99.03% accuracy for random forest respectively.

Data Preprocessing

Data

Dataset: Our Dataset consists of 9 variables and 769 records

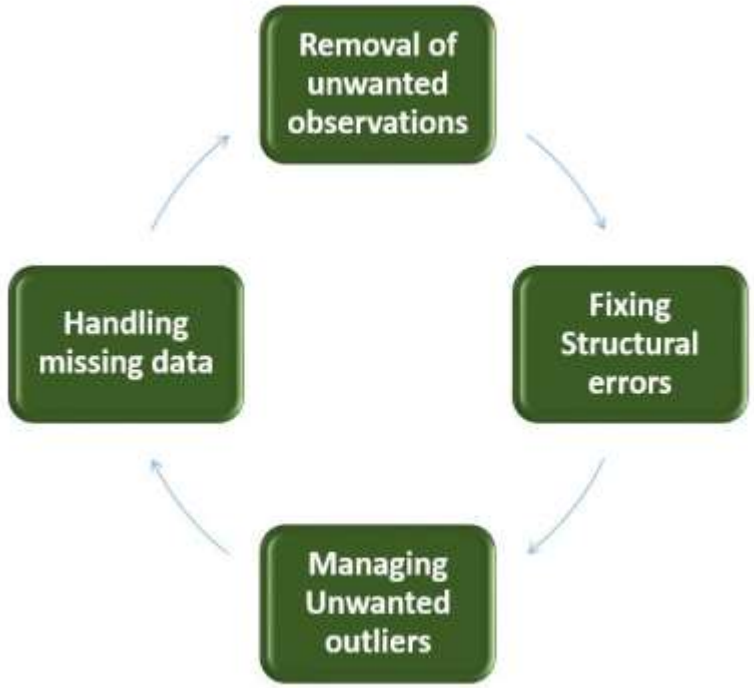
Source: https://drive.google.com/drive/folders/1vGSRCnhqSxEH53BgLqhh32F1qNKqfOim?usp=drive_link

Data View:

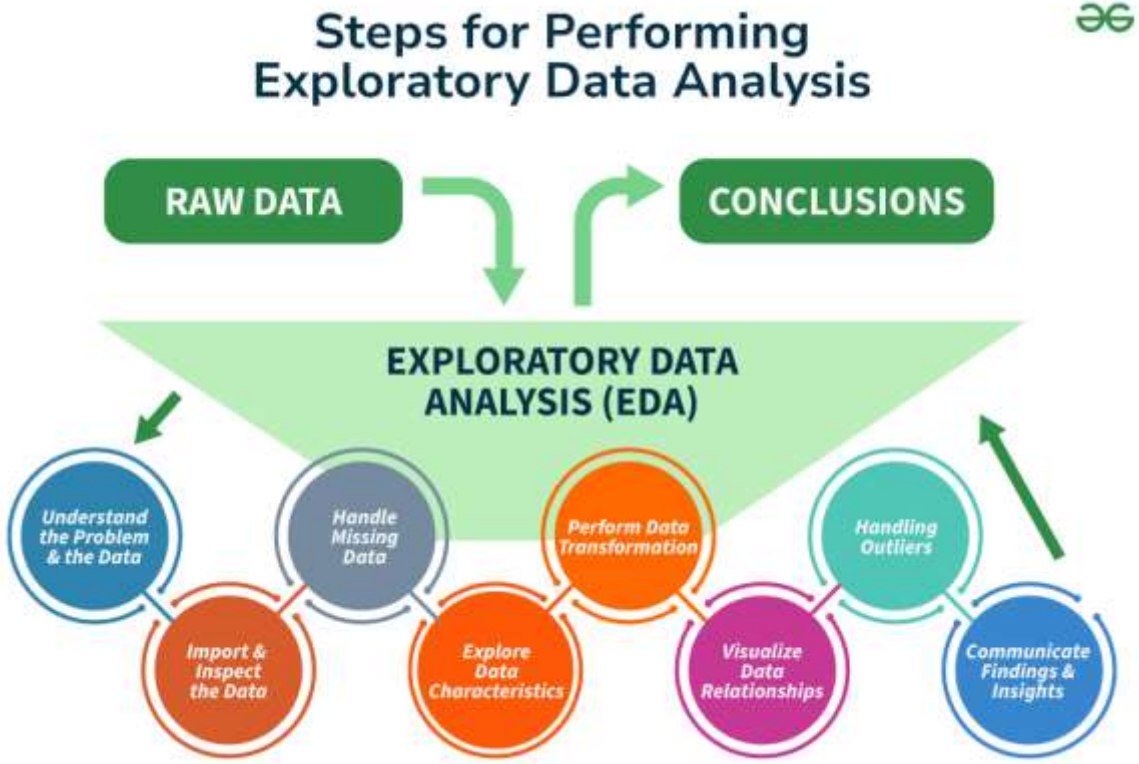
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

Data Cleaning

- Checked for missing values and unique values.
- **Only for EDA:**
  - Encoded the BMI variable: Enabled “Overweight” as 1 and “Normal Weight” as 0
  - Age has been converted into two categories: “Less than 60 years” and “Greater than 60 years”.

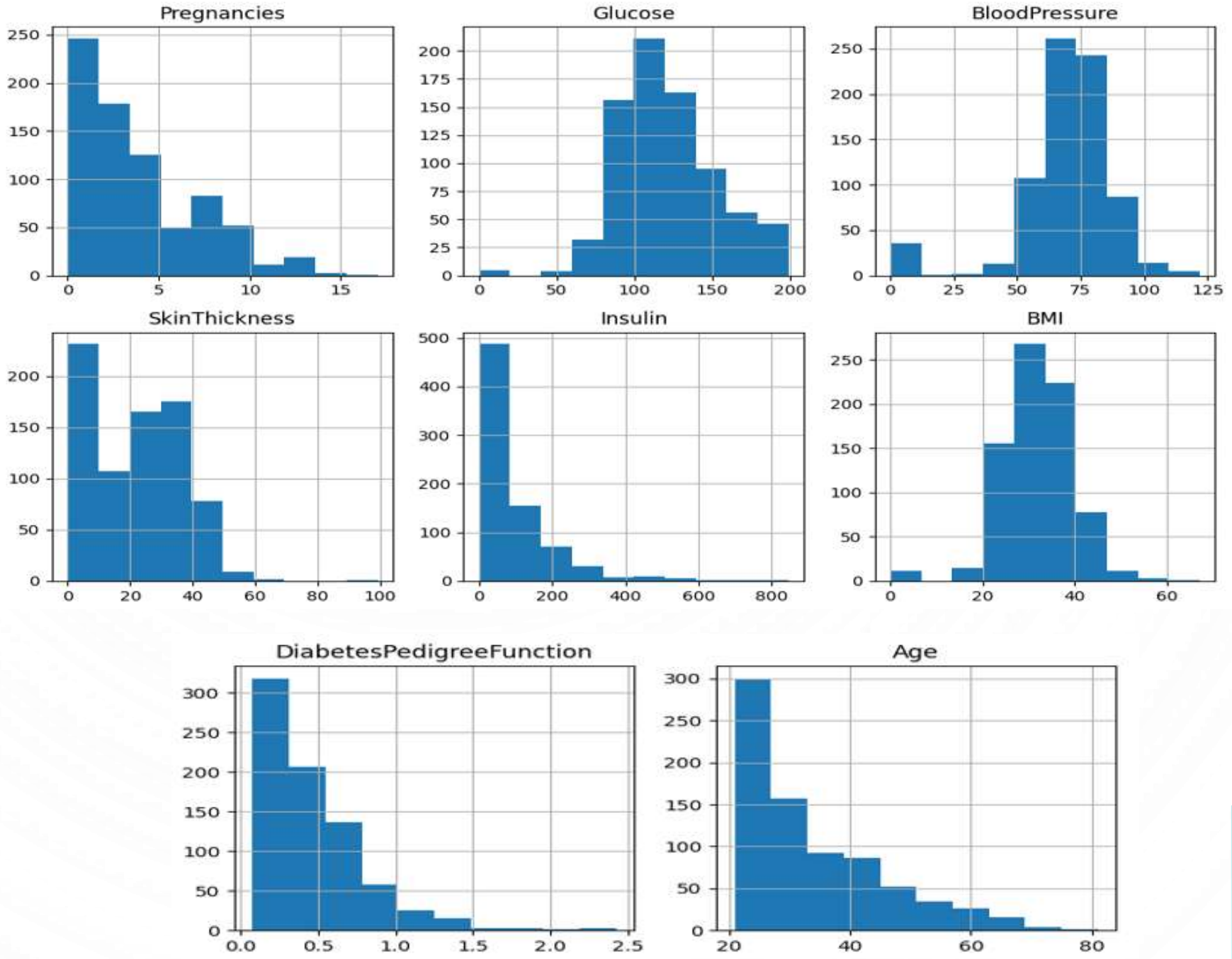


Exploratory Data Analysis



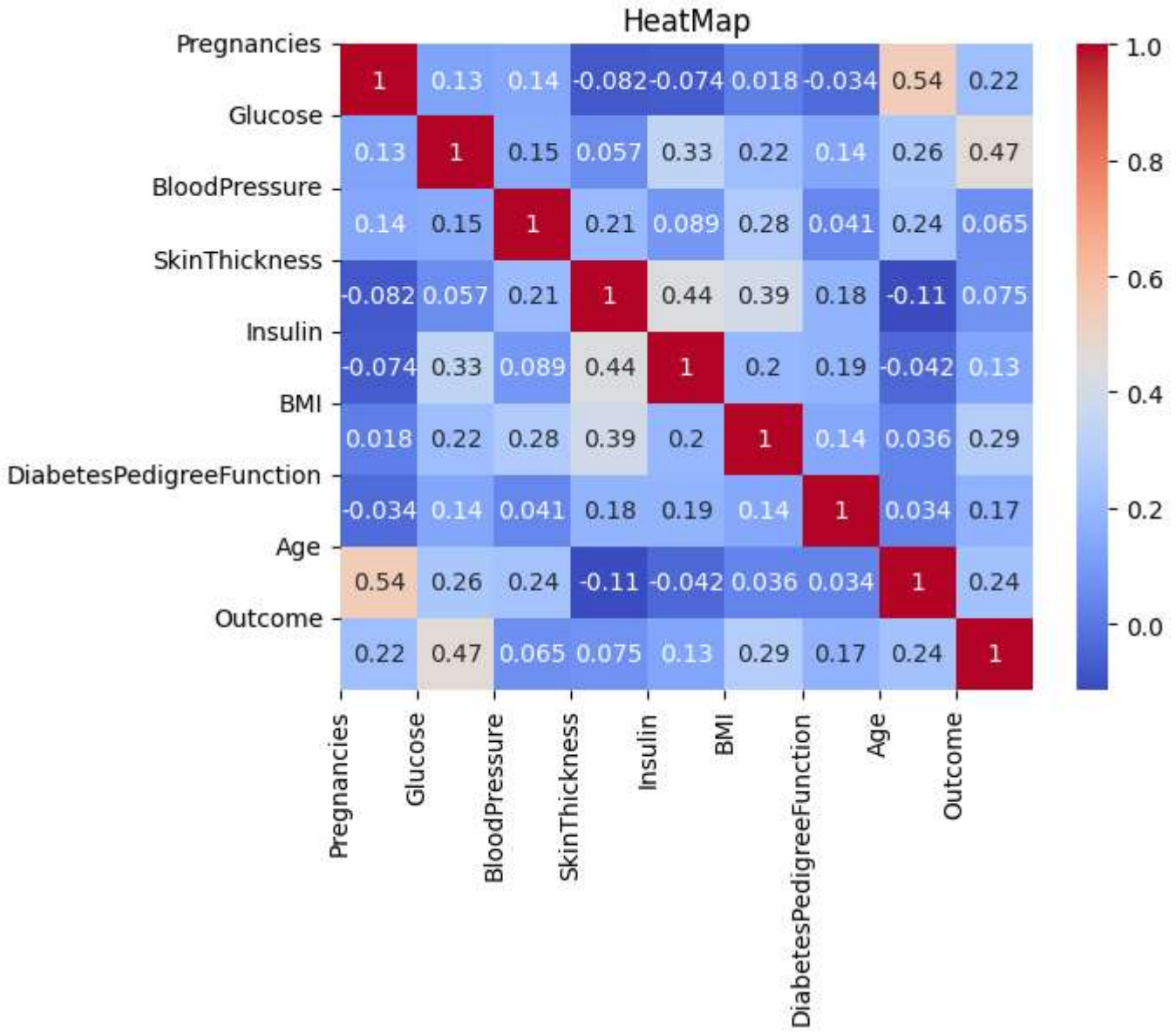
Histogram

- Histogram for numerical variables is shown below.

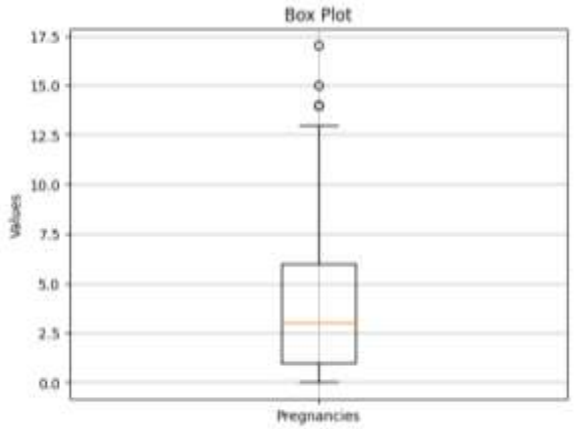


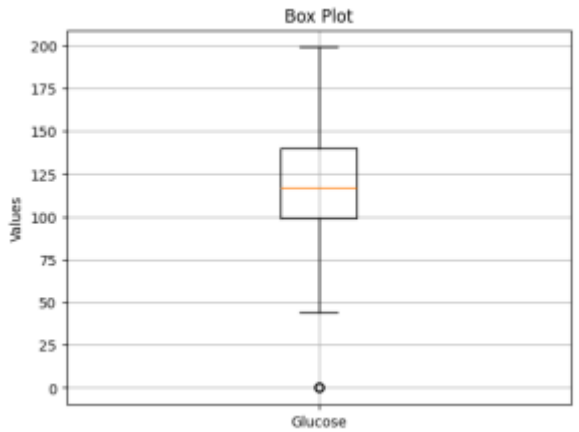
Heat Map

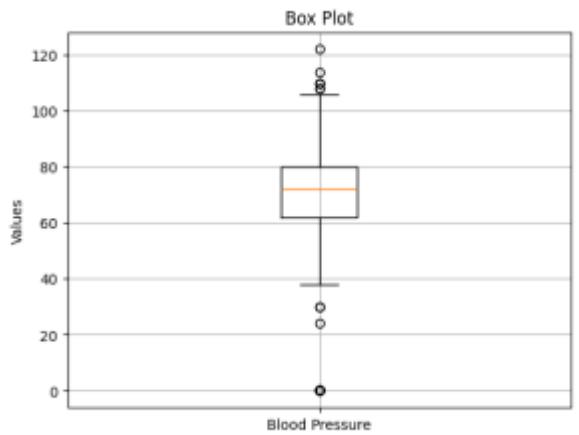
- We observe that all variables are moderately correlated.

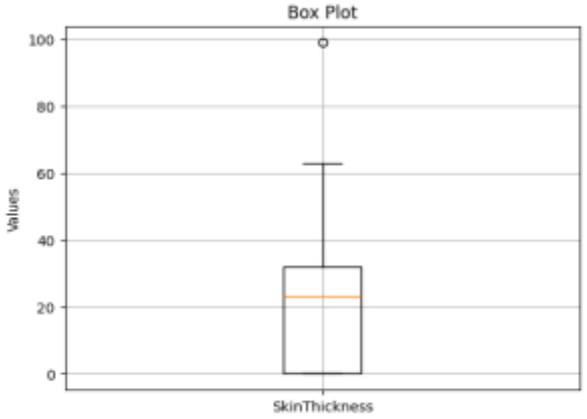


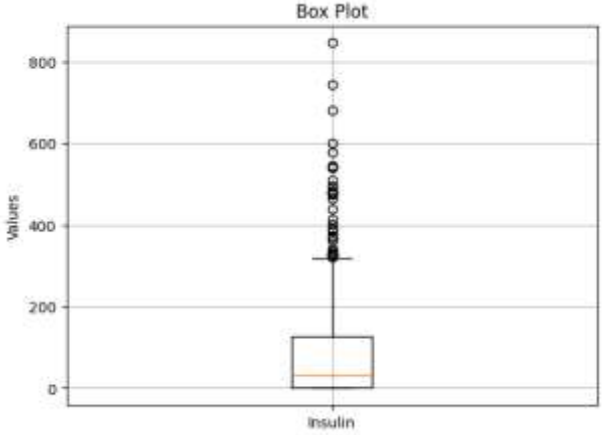
Box Plot

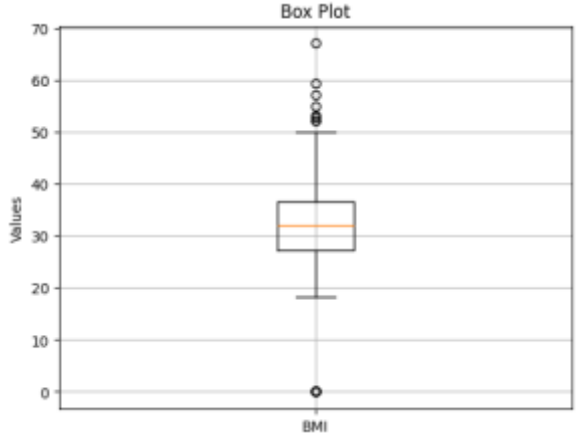

Box Plot
Values
Pregnancies

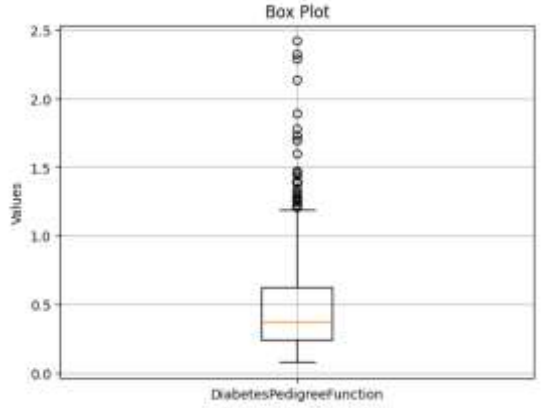

Box Plot
Values
Glucose

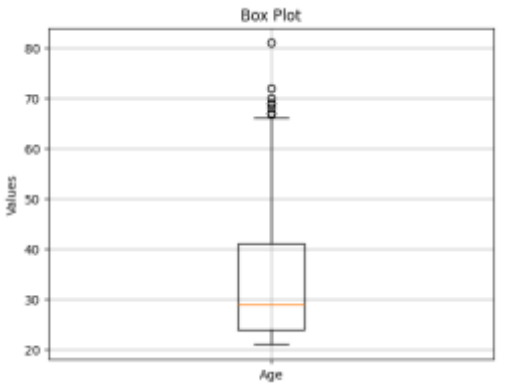

Box Plot
Values
Blood Pressure


Box Plot
Values
SkinThickness


Box Plot
Values
Insulin

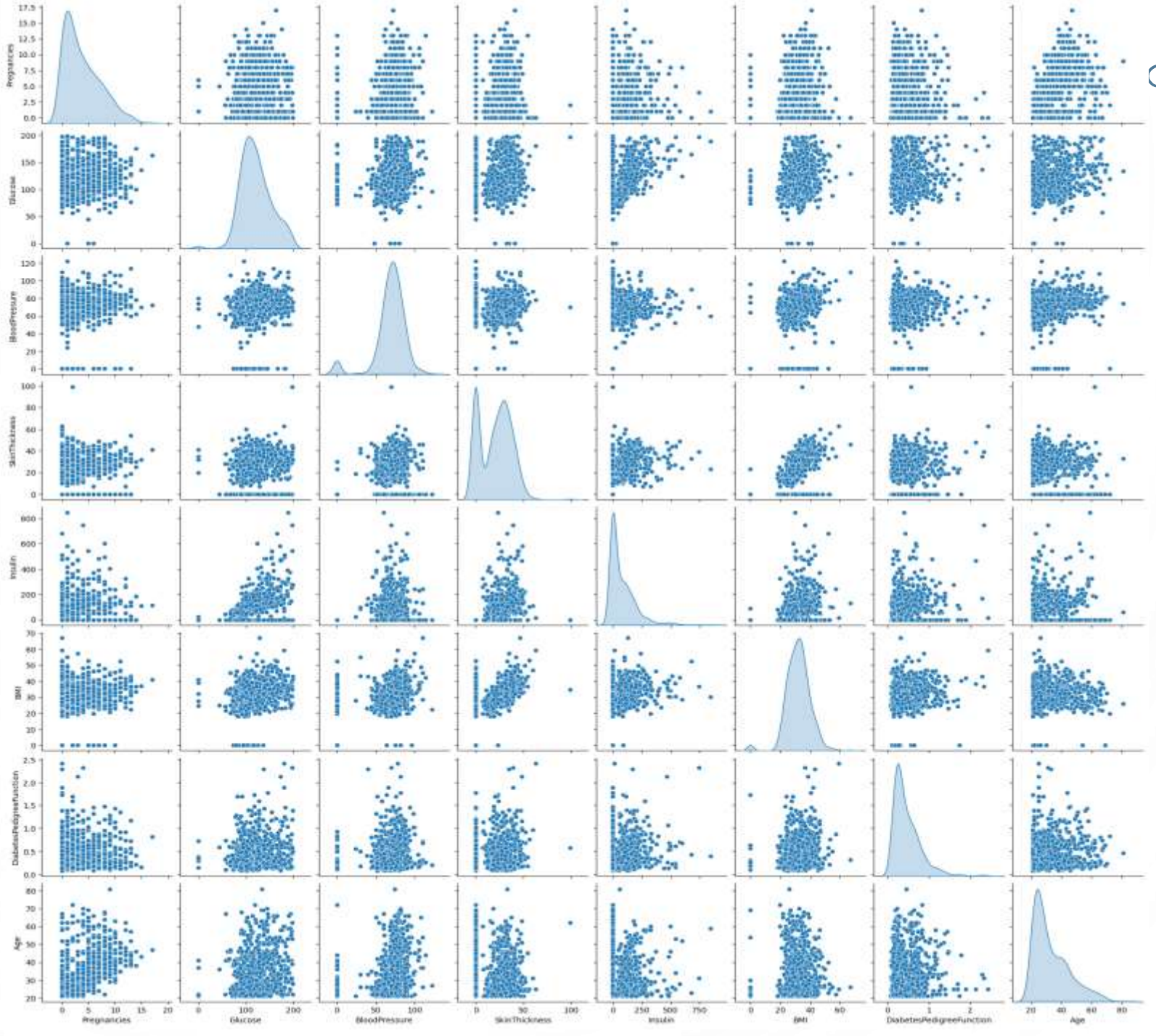

Box Plot
Values
BMI


Box Plot
Values
DiabetesPedigreeFunction


Box Plot
Values
Age

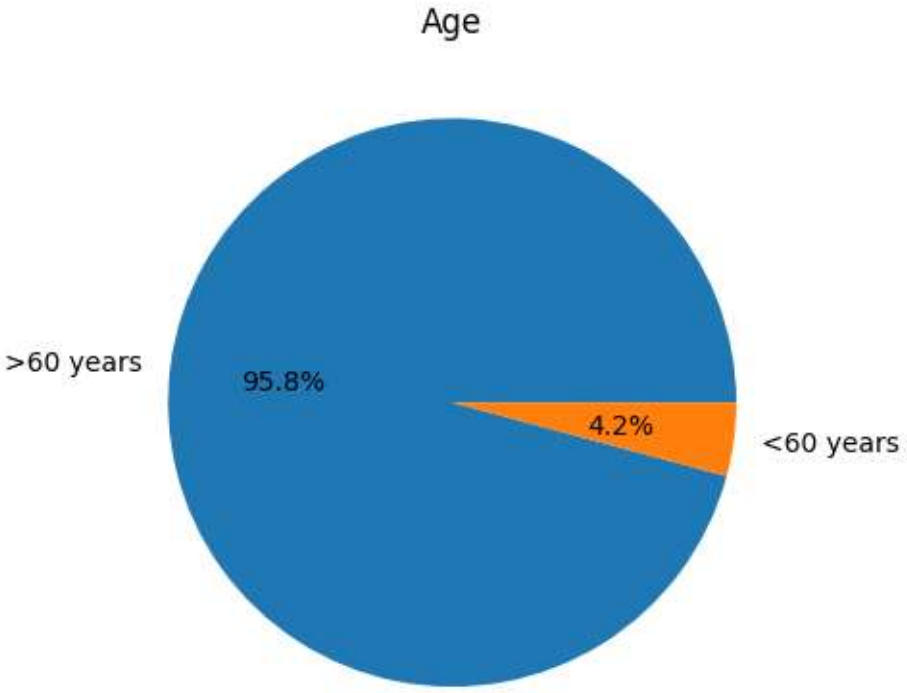
Pair Plot

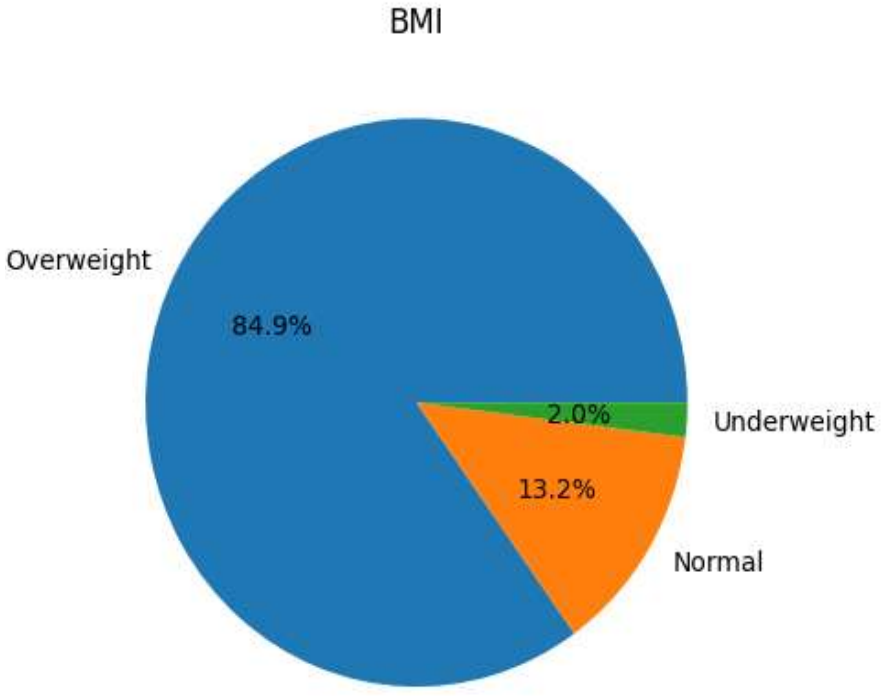
- Pair Plot enables visualization of the relationship between each pair of variables.

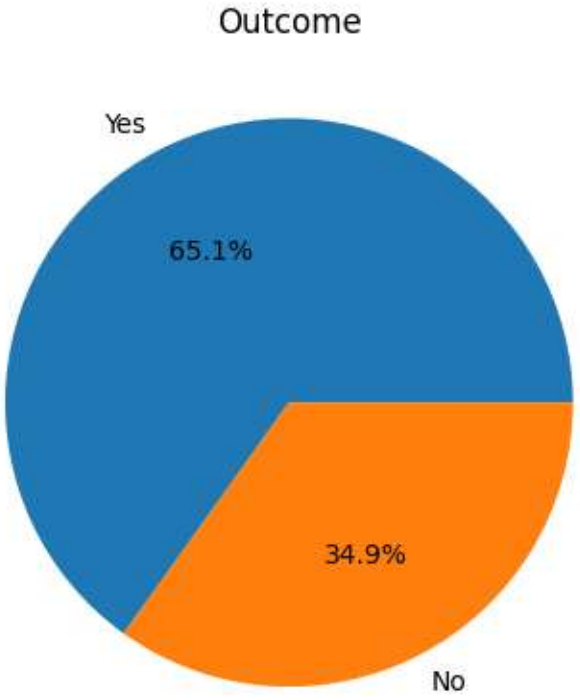


Pie Chart

- The pie chart is given for Age, BMI, and Outcome

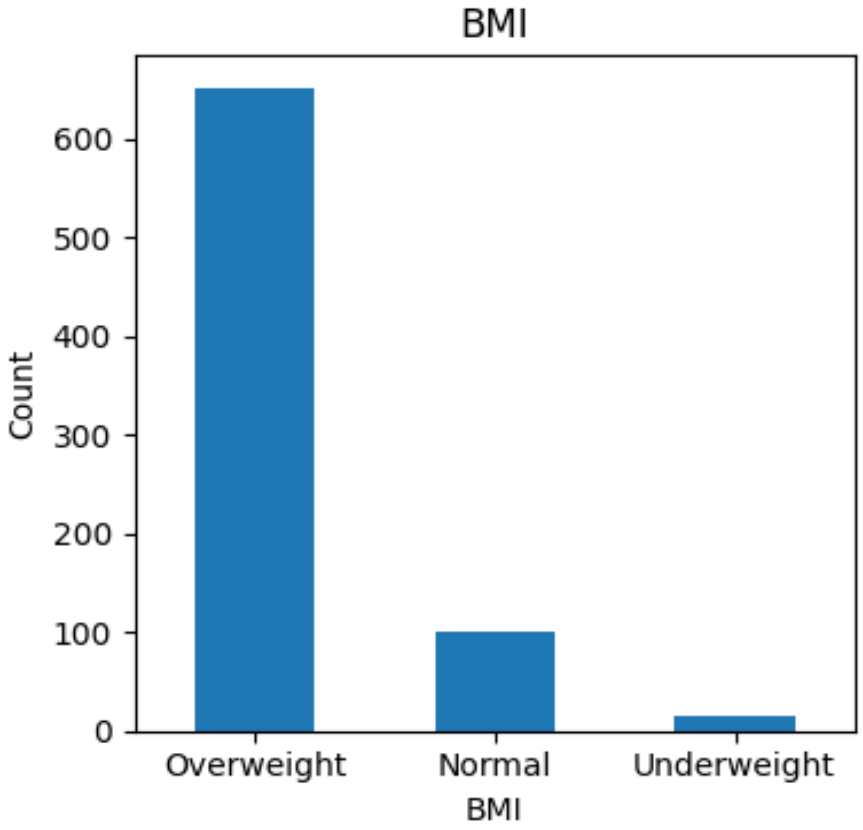


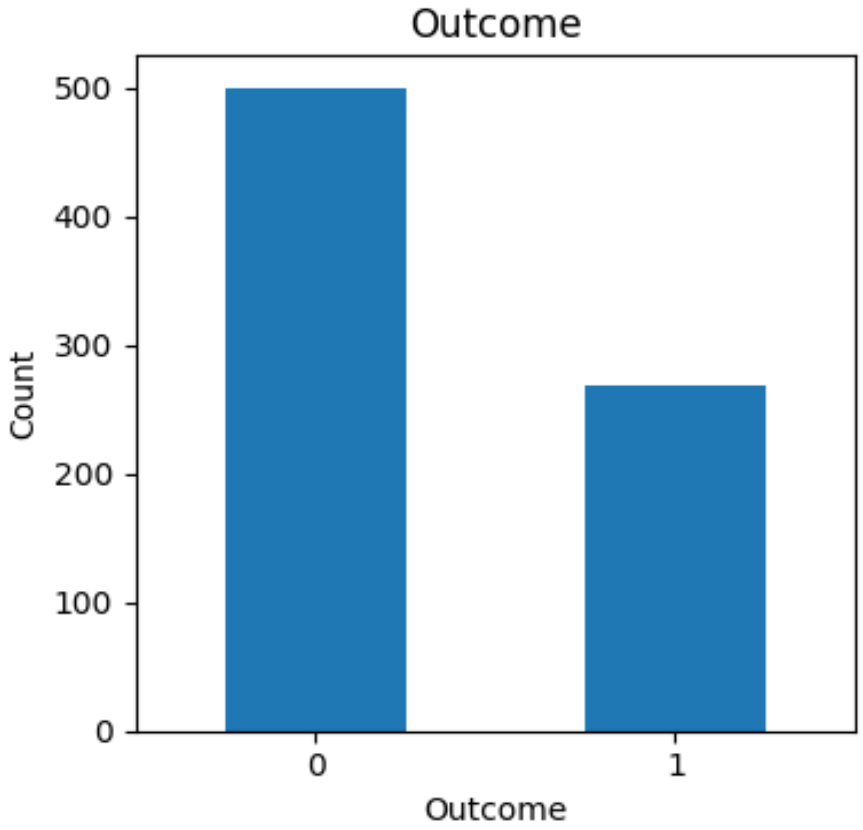




Bar Chart

- The bar chart is given for BMI, and Outcome





Multicollinearity Checking

- Variables showing VIF>3 are removed one after the other

	variables	VIF
0	Pregnancies	3.3
1	Glucose	16.7
2	BloodPressure	14.6
3	SkinThickness	4.0
4	Insulin	2.1
5	BMI	18.4
6	DiabetesPedigreeFunction	3.2
7	Age	13.5

Remove “BMI”

	variables	VIF
0	Pregnancies	3.3
1	Glucose	13.6
2	BloodPressure	12.4
3	SkinThickness	3.5
4	Insulin	2.0
5	DiabetesPedigreeFunction	3.2
6	Age	13.4

Remove “Glucose”

	variables	VIF
0	Pregnancies	3.3
1	BloodPressure	9.9
2	SkinThickness	3.5
3	Insulin	1.9
4	DiabetesPedigreeFunction	3.0
5	Age	10.9

Remove “Blood Pressure”

	variables	VIF
0	Pregnancies	3.3
1	SkinThickness	2.9
2	Insulin	1.9
3	DiabetesPedigreeFunction	2.9
4	Age	5.8

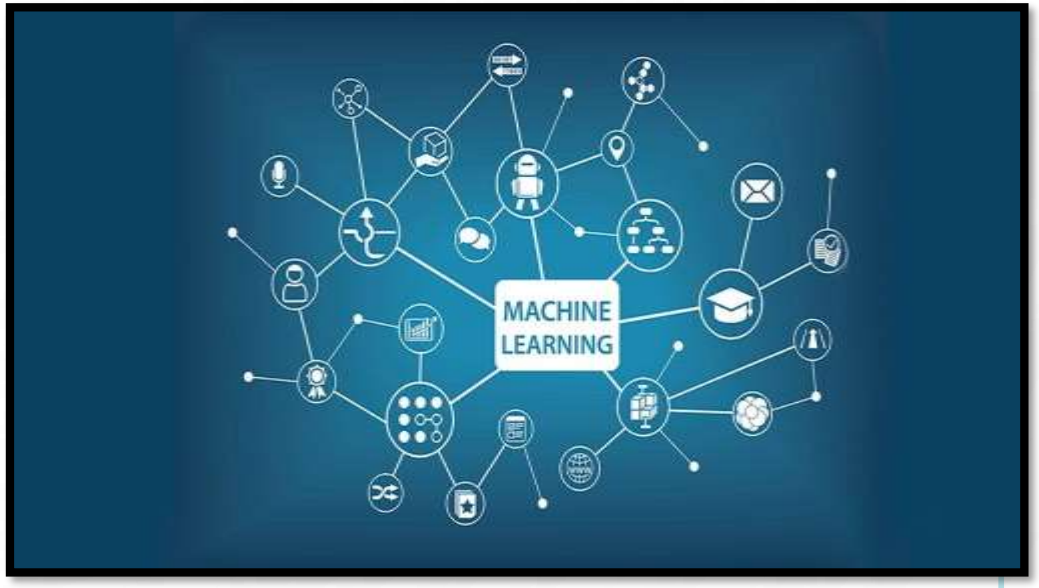
Remove “Age”

	variables	VIF
0	Pregnancies	1.7
1	SkinThickness	2.7
2	Insulin	1.9
3	DiabetesPedigreeFunction	2.4

Final List of Variables after VIF Checking

Machine Learning Algorithms

- Logistic Regression
- K Nearest Neighbour
- Support Vector Machine
- Decision Tree
- Boosting(XG Boosting, Adaptive Boosting)
- Random Forest



Logistic Regression

Logistic Regression in machine learning is a statistical method used for binary classification tasks, where the goal is to predict one of two possible outcomes(yes or no; 1 or 0; True or false).

Ratio	Model 1 (Before VIF) (Accuracy Score)	Model 2 (After VIF) (Accuracy Score)
80-20	0.779	0.656
75-25	0.776	0.656
70-30	0.784	0.654
60-40	0.750	0.666

K-Nearest Neighbour

K-Nearest Neighbors is a supervised learning algorithm which makes predictions based on the similarity between a new data point and the existing data points in the training dataset

Ratio	Model 1 (Before VIF) (Accuracy Score)	Model 2 (After VIF) (Accuracy Score)
80-20	0.753	0.617
75-25	0.771	0.625
70-30	0.771	0.632
60-40	0.740	0.620

Support Vector Machine

Support Vector Machine is a supervised classification algorithm that finds the optimal decision boundary (hyperplane) that maximizes the margin between different classes.

There are two types of SVM's - Linear SVM; Non-Linear SVM

Ratio	Model 1 (Before VIF) (Accuracy Score)	Model 2 (After VIF) (Accuracy Score)
80-20	0.779	0.688
75-25	0.781	0.677
70-30	0.792	0.688
60-40	0.773	0.662

Decision Tree

A Decision Tree is a supervised machine learning algorithm that models decisions and their possible consequences as a tree-like structure, where each node represents a decision based on a feature and each branch represents the outcome of that decision.

Ratio	Model 1 (Before VIF) (Accuracy Score)	Model 2 (After VIF) (Accuracy Score)
80-20	0.786	0.656
75-25	0.776	0.677
70-30	0.758	0.671
60-40	0.731	0.675

Boosting (XG Boost)

XG Boost (Extreme Gradient Boosting) is an advanced, efficient version of gradient boosting that builds a strong model by combining many weak decision trees and optimizes performance through techniques like regularization, parallelization, and handling missing values

Ratio	Model 1 (Before VIF) (Accuracy Score)	Model 2 (After VIF) (Accuracy Score)
80-20	0.792	0.662
75-25	0.771	0.667
70-30	0.758	0.649
60-40	0.770	0.633

Boosting (Adaptive Boosting)

Adaptive Boosting is a machine learning algorithm that improves the accuracy of weak learners by focusing on the data points that are hardest to classify. It works by sequentially training weak models, adjusting the weights of misclassified data points, and combining the predictions of all models into a stronger classifier.

Ratio	Model 1 (Before VIF) (Accuracy Score)	Model 2 (After VIF) (Accuracy Score)
80-20	0.790	0.688
75-25 .	0.760	0.698
70-30	0.753	0.675
60-40	0.744	0.698

Random Forest

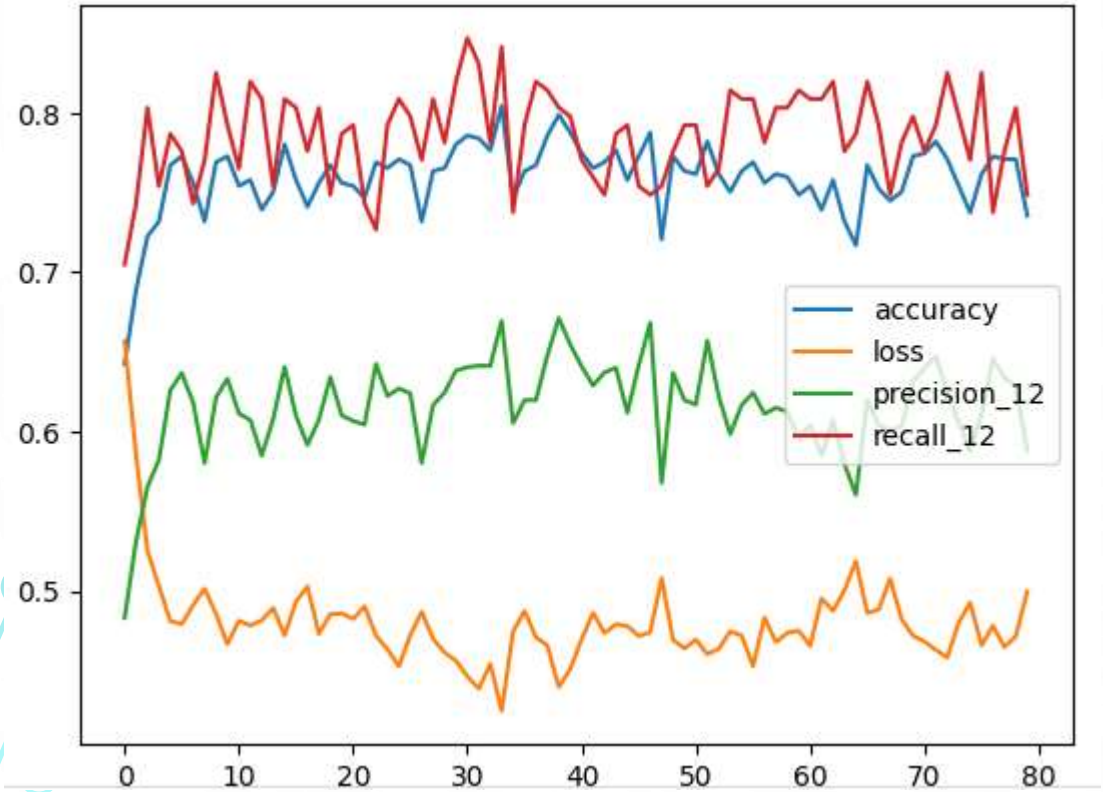
Random Forest is an ensemble learning algorithm that combines multiple decision trees to create a more accurate and stable model. It reduces overfitting by using random samples of data and features to build each tree and by averaging (or voting) across all trees for final predictions

Ratio	Model 1 (Before VIF) (Accuracy Score)	Model 2 (After VIF) (Accuracy Score)
80-20	0.799	0.656
75-25 .	0.786	0.646
70-30	0.788	0.662
60-40	0.788	0.656

ARTIFICIAL NEURAL NETWORK

SPLIT	Architecture	Optimizer	Epochs	Accuracy
80-20	128-64-32-1	SDG	500	nan
80-20	128-64-32-1	Adam	500	0.769
80-20	128-64-32-1	Adam	250	0.655
75-25	128-64-32-1	SGD	500	nan
75-25	128-64-32-1	Adam	500	0.764
75-25	128-64-32-1	Adam	500	0.672
70-30	128-64-32-1	SGD	250	nan
70-30	128-64-32-1	Adam	80	0.818
70-30	128-64-32-1	Adam	80	0.068
60-40	128-64-32-1	SGD	500	nan
60-40	128-64-32-1	Adam	250	0.695
60-40	128-64-32-1	Adam	250	0.652

Neural Network Plot for Observation



Train Test Split	70-30
Architecture	128-64-32-1
Optimizer	Adam
Epochs	80

Finding the Best Split

Algorithm	Before VIF (Highest Accuracy Score)		After VIF (Highest Accuracy Score)	
Logistic Regression	0.779	(80-20)	0.666	(60-40)
K-Nearest Neighbour	0.771	(80-20, 75-25)	0.632	(70-30)
Support Vector Machine	0.792	(70-30)	0.688	(70-30)
Decision Tree	0.786	(80-20)	0.677	(75-25)
XG Boost	0.792	(80-20)	0.667	(75-25)
Adaptive Boosting	0.790	(80-20)	0.698	(75-25)
Random Forest	0.779	(80-20)	0.662	(70-30)
Artificial Neural Network	0.818	(70-30)	0.681	(70-30)

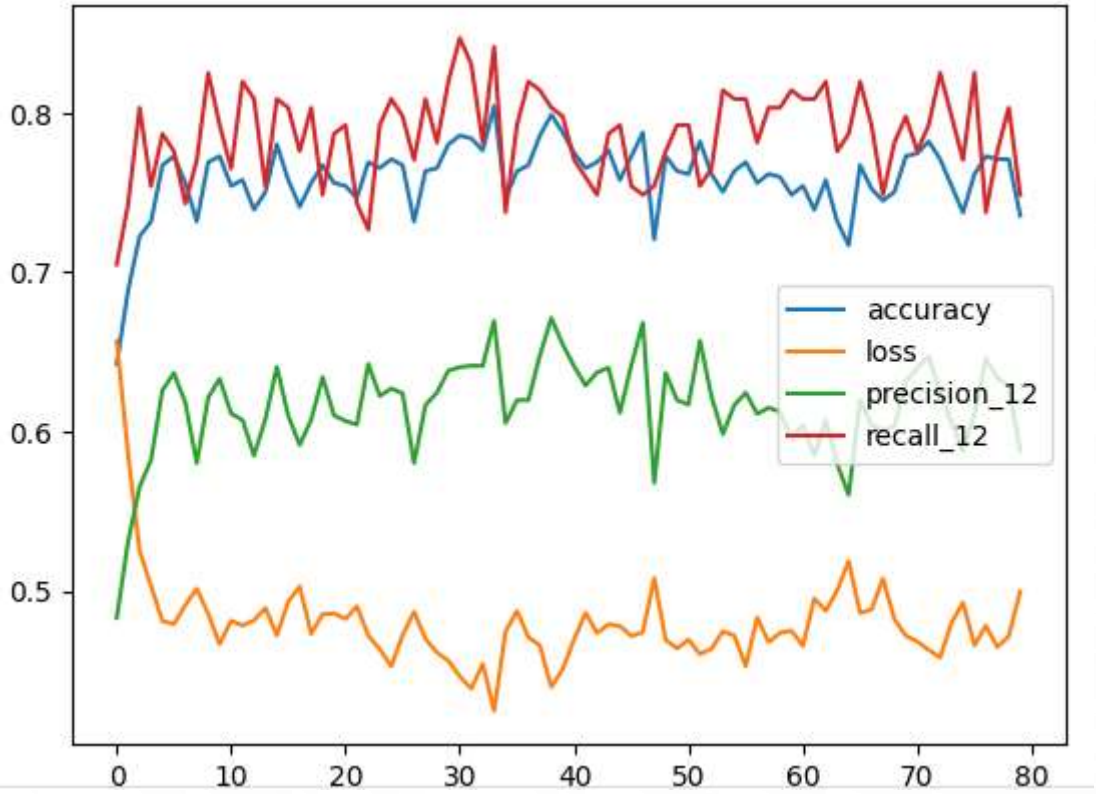
70-30 is the best split "Before VIF"

75-25 is the best split "After VIF"

Finding the Best Algorithm

Algorithm	Before VIF(Highest Accuracy Score)	After VIF (Highest Accuracy Score)
Logistic Regression	0.779	0.657
K-Nearest Neighbour	0.766	0.625
Support Vector Machine	0.779	0.680
Decision Tree	0.770	0.703
XG Boost	0.776	0.682
Adaptive Boosting	0.750	0.700
Random Forest	0.800	0.660
Artificial Neural Network	0.818	0.672

Conclusion



- From the above analysis of algorithms, it can be concluded that the **Artificial Neural Network** Algorithm has the highest accuracy

Summary

After reviewing the above comparisons it is evident that the Artificial Neural Network has the highest accuracy and Hence, can be considered the best machine learning algorithm for predicting diabetes.

Insights

The main cause of Diabetes is a sedentary lifestyle and lack of exercise, spreading awareness about these factors is essential to bring about a massive change in society. This insight can be inferred from the slide where the age distribution has been depicted in the form of a pie chart.

Enhancing Diagnosis: This analysis helps doctors to diagnose diabetes earlier thereby preventing it rather than curing it.

Future Scope

- The current model can be further improved by adding much more advanced Algorithms such as Recurrent Neural Networks or Convolutional Neural Networks as they can identify more complex patterns efficiently.
- It can be trained on a much larger and more diverse dataset.
- The model can also be implemented in real-life scenarios such as advanced medical technologies that continuously monitor the patient's health and provide insights on how to treat them.

Work Distribution

Team Member	Tasks
S.Jayavardhan	Introduction and Literature Review
J.Sai Dwarak	Data Pre-Processing and EDA
B.Poojashree	Data Modelling and Evaluation
G.Shyam	Algorithm Comparison & Summary

Thank You

Done By

G.Shyam (107222546014)

J.Sai Dwarak (107222546016)

B.Poojashree (107222546007)

S.Jayavardhan (107222546017)

Colab Notebook- Classification Project

https://colab.research.google.com/drive/1Mwcx-xliRDJcxoTm1QVuYkoaE-QtxXSS?usp=sharing

Appendix

Importing Libraries

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

Uploading File

```
[ ] 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 diabetes.csv to diabetes (2).csv
```

Data Preprocessing

```
[ ] data=pd.read_csv('/content/diabetes.csv')
    data.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

```
[ ] data.shape
```

```
(768, 9)
```

```
[ ] data.isna().sum()
```

	0
Pregnancies	0
Glucose	0
BloodPressure	0
SkinThickness	0
Insulin	0
BMI	0
DiabetesPedigreeFunction	0
Age	0
Outcome	0

dtype: int64

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

```
44      4
100     3
106     3
98      3
110     3
55      2
108     2
104     2
46      2
30      2
122     1
95      1
102     1
61      1
24      1
38      1
40      1
114     1
Name: count, dtype: int64
[35 29  0 23 32 45 19 47 38 30 41 33 26 15 36 11 31 37 42 25 18 24 39 27
 21 34 10 60 13 20 22 28 54 40 51 56 14 17 50 44 12 46 16  7 52 43 48  8
 49 63 99]
SkinThickness
0     227
32     31
30     27
27     23
23     22
```

```
[ ] data.iloc[:,8].value_counts()
```

Outcome	count
0	500
1	268

dtype: int64

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

```
Pregnancies
1     135
0     111
2     103
3      75
4      68
5      57
6      50
7      45
8      38
9      28
10     24
11     11
13     10
12      9
14      2
15      1
17      1
Name: count, dtype: int64
Glucose
99     17
100    17
111    14
129    14
125    14
       ..
191     1
177     1
44      1
62      1
190     1
Name: count, Length: 136, dtype: int64
BloodPressure
70     57
74     52
78     45
68     45
72     44
```

```
[ ] data2=data.copy(deep=True)
```

```
[ ] print(data)
```

```
     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0              6      148             72             35        0  33.6
1              1       85             66             29        0  26.6
2              8      183             64              0        0  23.3
3              1       89             66             23       94  28.1
4              0      137             40             35      168  43.1
..           ...      ...            ...            ...      ...   ...
763           10      101             76             48      180  32.9
764            2      122             70             27        0  36.8
765            5      121             72             23      112  26.2
766            1      126             60              0        0  30.1
767            1       93             70             31        0  30.4

     DiabetesPedigreeFunction  Age  Outcome
0                       0.627   50        1
1                       0.351   31        0
2                       0.672   32        1
3                       0.167   21        0
4                       2.288   33        1
..                        ...  ...      ...
763                     0.171   63        0
764                     0.340   27        0
765                     0.245   30        0
766                     0.349   47        1
767                     0.315   23        0

[768 rows x 9 columns]
```

```
[ ] data.describe()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.348958
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

Assigning Feature and Target Variables

```python
#Feature Variables
X=data.drop(['Outcome'],axis=1)
print(X)

#Target Variable
y=data['Outcome']
print(y)
```

```
     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0              6      148             72             35        0  33.6
1              1       85             66             29        0  26.6
2              8      183             64              0        0  23.3
3              1       89             66             23       94  28.1
4              0      137             40             35      168  43.1
..           ...      ...            ...            ...      ...   ...
763           10      101             76             48      180  32.9
764            2      122             70             27        0  36.8
765            5      121             72             23      112  26.2
766            1      126             60              0        0  30.1
767            1       93             70             31        0  30.4

     DiabetesPedigreeFunction  Age
0                       0.627   50
1                       0.351   31
2                       0.672   32
3                       0.167   21
4                       2.288   33
..                        ...  ...
763                     0.171   63
764                     0.340   27
765                     0.245   30
766                     0.349   47
767                     0.315   23

[768 rows x 8 columns]
0      1
1      0
2      1
3      0
4      1
      ..
763    0
764    0
765    0
766    1
767    0
Name: Outcome, Length: 768, dtype: int64
```

Exploratory Data Analysis

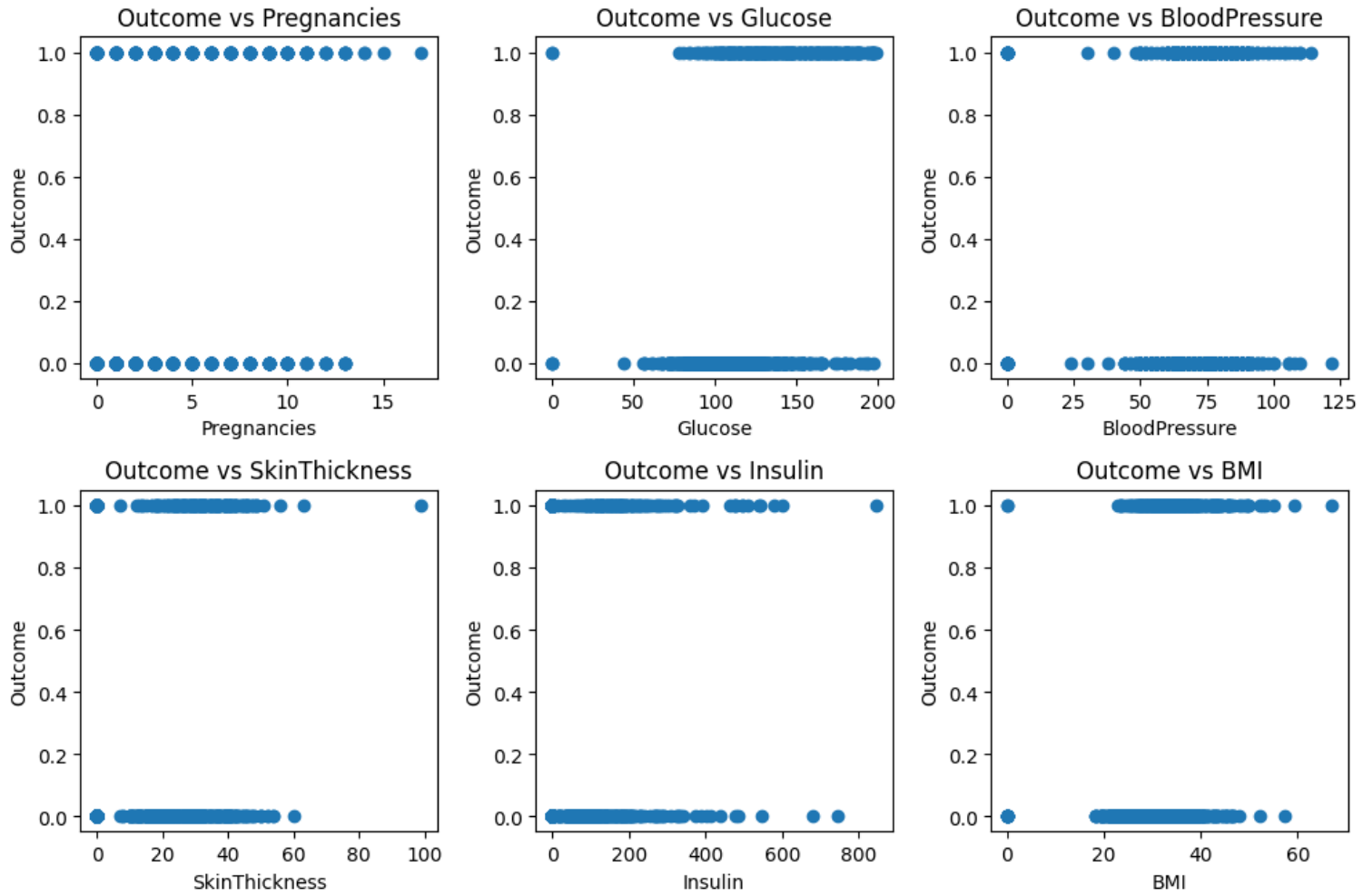
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               768 non-null    int64  
 1   Glucose                   768 non-null    int64  
 2   BloodPressure             768 non-null    int64  
 3   SkinThickness             768 non-null    int64  
 4   Insulin                   768 non-null    int64  
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64  
 8   Outcome                   768 non-null    int64  
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

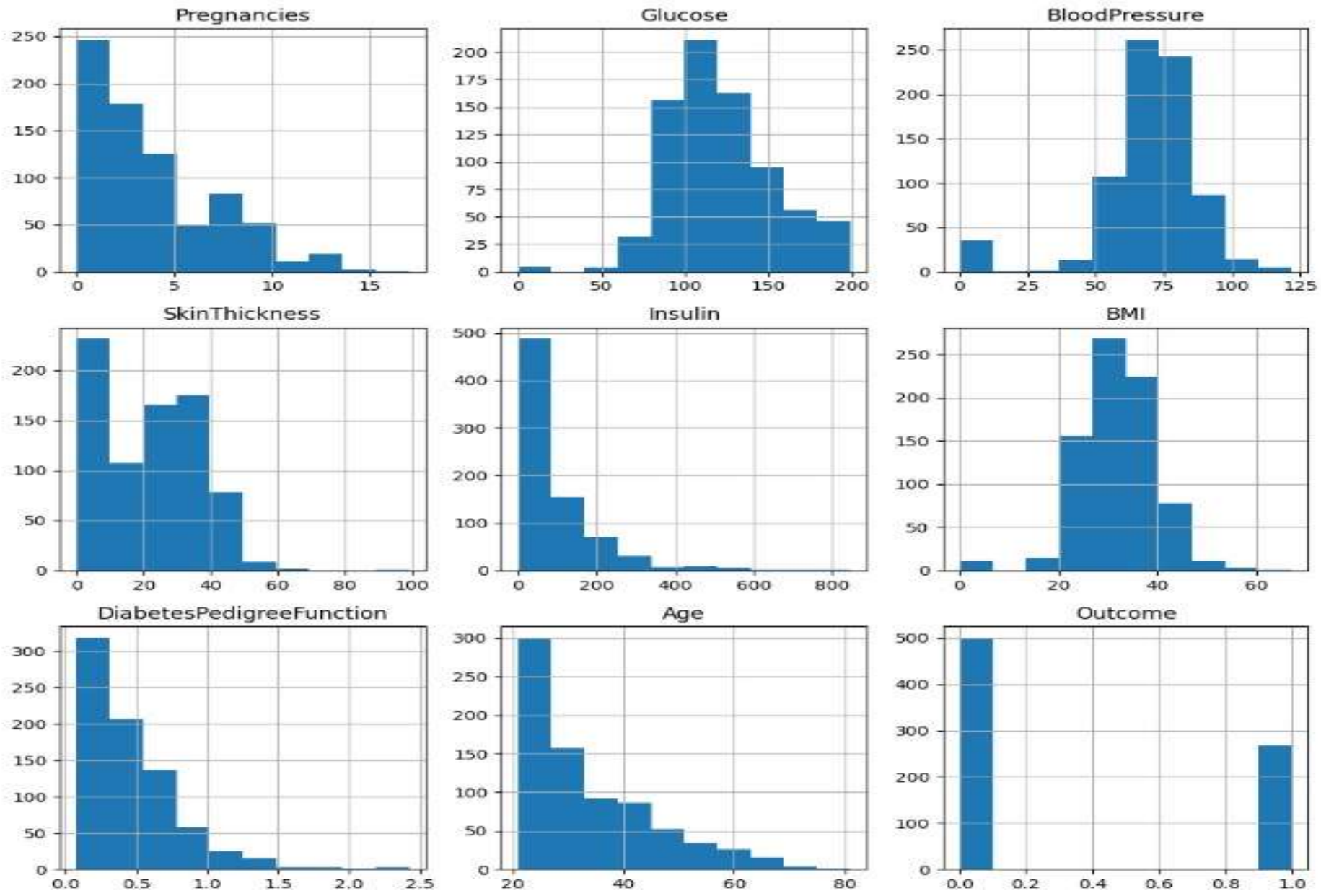
```
[ ] data.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

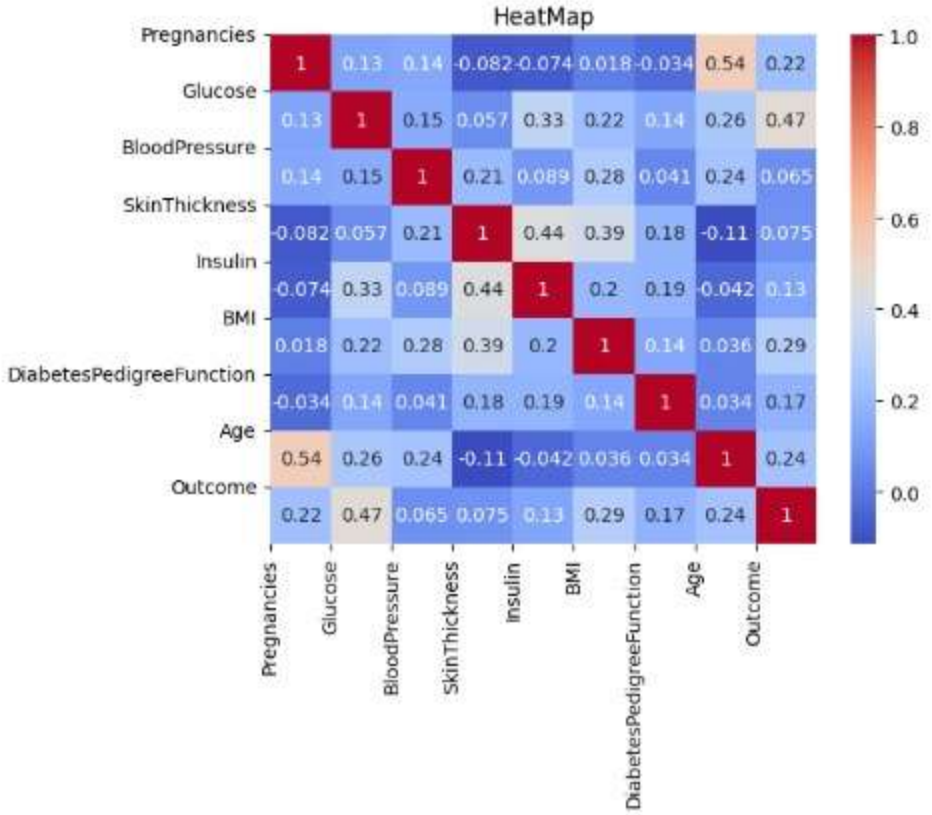
```python
# Scatter Plot

features = ["Pregnancies","Glucose","BloodPressure","SkinThickness","Insulin","BMI","DiabetesPedigreeFunction","Age"]
plt.figure(figsize=(10, 10))

for i, feature in enumerate(features):
    plt.subplot(3, 3, i+1)
    plt.scatter(data[feature], data['Outcome'])
    plt.xlabel(feature)
    plt.ylabel('Outcome')
    plt.title(f'Outcome vs {feature}')

plt.tight_layout()
plt.show()
```



```
# Histogram

data.hist(figsize=(10, 10))
plt.tight_layout()
plt.show()
```



```
# Correlation Matrix or HeatMap

Data=data[['Pregnancies','Glucose','BloodPressure','SkinThickness','Insulin','BMI','DiabetesPedigreeFunction','Age','Outcome']].corr()
sns.heatmap(Data,cmap='coolwarm',annot=True)
plt.xticks(range(len(data.columns)),data.columns,rotation=90)
plt.yticks(range(len(data.columns)),data.columns)
plt.title('HeatMap')
plt.show()
```



Logistic Regression (Before VIF)

```
[40] from sklearn.model_selection import train_test_split
     from sklearn.linear_model import LogisticRegression
     from sklearn.metrics import accuracy_score
     from sklearn.metrics import confusion_matrix
     from sklearn.metrics import classification_report
```

```
[41] X = data2.drop('Outcome', axis=1)
     y = data2['Outcome']

     # 2. Ensure X and y have the same number of samples
     print(X.shape)
     print(y.shape)

     X_train1,X_test1,y_train1,y_test1=train_test_split(X,y,test_size=0.20,train_size=0.80,random_state=1)
     X_train2,X_test2,y_train2,y_test2=train_test_split(X,y,test_size=0.25,train_size=0.75,random_state=1)
     X_train3,X_test3,y_train3,y_test3=train_test_split(X,y,test_size=0.30,train_size=0.70,random_state=1)
     X_train4,X_test4,y_train4,y_test4=train_test_split(X,y,test_size=0.40,train_size=0.60,random_state=1)
```

```
(768, 8)
(768,)
```

80-20 Train Test split

```
[42] logreg = LogisticRegression(C=1e9)
     logreg.fit(X_train1, y_train1)
     predictions = logreg.predict(X_test1)
     print(predictions)
     z=confusion_matrix(y_test1, predictions)
     print(z)
     print(accuracy_score(y_test1,predictions))
     print(classification_report(y_test1,predictions))
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 0 0 0 0 0 0 1 0 0 0 0 1 0 1 0 0 0 1 0 1 0
 0 0 1 0 0 0 0 0 0 1 1 0 0 0 0 1 0 1 0 1 0 1 0 1 0 1 0 0 0 0 0 1 1 1 1 1 0
 1 0 1 0 0 1 1 0 0 0 0 1 1 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 1 0 0 0
 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 1 0 0 1 0 0 1 1 0 0 0
 0 0 0 1 0 0]
[[89 10]
 [24 31]]
0.7792207792207793
              precision    recall  f1-score   support

           0       0.79      0.90      0.84        99
           1       0.76      0.56      0.65        55

    accuracy                           0.78       154
   macro avg       0.77      0.73      0.74       154
weighted avg       0.78      0.78      0.77       154

/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:469: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

K-Nearest Neighbor (Before VIF)

```
[46] from sklearn.neighbors import KNeighborsClassifier
     model=KNeighborsClassifier(n_neighbors=25)
     from sklearn.metrics import accuracy_score
     from sklearn.metrics import confusion_matrix
     from sklearn.metrics import classification_report
```

80-20 Train Test Split

```
[47] model.fit(X_train1, y_train1)
     y_pred1 = model.predict(X_test1)
     print(y_pred1)

     knn = pd.DataFrame({'Predicted':y_pred1,'Actual':y_test1})
     print(knn)
     print(accuracy_score(y_test1,y_pred1))
     cm = confusion_matrix(y_test1,y_pred1)
     print(cm)

     sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=["No Diabetes", "Diabetes"], yticklabels=["No Diabetes", "Diabetes"])
     plt.title('Confusion Matrix')
     plt.xlabel("Predicted")
     plt.ylabel("Truth")
     plt.show()

     classification_rep = classification_report(y_test1, y_pred1)
     print(classification_rep)
```

```
[1 0 0 0 0 0 1 0 0 0 0 0 1 1 0 1 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 1 0 1 0
 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 0 1 0 1 1 1 1 0
 1 0 1 0 0 1 0 0 0 0 0 1 0 0 1 0 1 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 1 1 0 0 1
 0 1 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 1 0 1 0 1 0 1 0 0 0 0 0 1 0 0 1 0 0 0 0
 0 0 0 1 0 0]
     Predicted  Actual
285          1       0
101          0       0
581          0       0
352          0       0
726          0       0
..         ...     ...
563          0       0
318          0       0
154          1       1
684          0       0
643          0       0

[154 rows x 2 columns]
0.7532467532467533
[[87 12]
 [26 29]]
```

Confusion Matrix

Support Vector Machine (Before VIF)

```
[51] import numpy as np
     import pandas as pd
     import seaborn as sns
     %matplotlib inline
     import matplotlib.pyplot as plt
     from sklearn.metrics import accuracy_score
     from sklearn.metrics import confusion_matrix
     from sklearn.metrics import classification_report
     from sklearn.svm import SVC
     model = SVC(kernel='linear')
```

80-20 Train Test Split

```
[52] model.fit(X_train1, y_train1)
     y_pred1 = model.predict(X_test1)
     print(y_pred1)

     svm = pd.DataFrame({'Predicted':y_pred1,'Actual':y_test1})
     print(svm)
     print(accuracy_score(y_test1,y_pred1))
     cm = confusion_matrix(y_test1,y_pred1)
     print(cm)

     sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=["No Diabetes", "Diabetes"], yticklabels=["No Diabetes", "Diabetes"])
     plt.title('Confusion Matrix')
     plt.xlabel("Predicted")
     plt.ylabel("Truth")
     plt.show()
```

```
[0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 0 0 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0 1 0
 0 0 1 0 0 0 0 0 1 1 0 0 0 1 0 1 0 1 0 0 0 1 0 1 0 0 0 0 0 1 1 1 1 1 0
 1 0 1 0 0 1 1 0 0 0 0 1 1 0 0 0 0 0 0 0 1 1 0 0 1 0 0 0 1 0 0 0 1 1 0 0 0
 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 1 0 0 1 0 0 1 1 0 0 0
 0 0 0 1 0 0]
     Predicted  Actual
285          0       0
101          0       0
581          0       0
352          0       0
726          0       0
..         ...     ...
563          0       0
318          0       0
154          1       1
684          0       0
643          0       0

[154 rows x 2 columns]
0.7792207792207793
[[89 10]
 [24 31]]
```

Multicollinearity

```
[83] X = data2.drop('Outcome', axis=1)
     y = data2["Outcome"]

     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

```
[84] from statsmodels.stats.outliers_influence import variance_inflation_factor
     import pandas as pd
     import numpy as np

     def calc_vif(X):
         # Convert all columns to numeric if possible, otherwise handle errors
         X = X.apply(pd.to_numeric, errors='coerce').fillna(0)

         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)
```

	variables	VIF
0	Pregnancies	3.3
1	Glucose	16.7
2	BloodPressure	14.6
3	SkinThickness	4.0
4	Insulin	2.1
5	BMI	18.4
6	DiabetesPedigreeFunction	3.2

Logistic Regression (After VIF)

```
[91] from sklearn.linear_model import LogisticRegression
```

80-20 Train Test Split

```
[92] logreg = LogisticRegression(C=1e9)
     logreg.fit(X_train1_nomulti, y_train1_nomulti,)
     predictions = logreg.predict(X_test1_nomulti)
     print(predictions)
     z=confusion_matrix(y_test1_nomulti,predictions)
     print(z)
     print(accuracy_score(y_test1_nomulti,predictions))
     print(classification_report(y_test1_nomulti,predictions))
```

```
[0 0 0 0 1 1 0 1 0 1 0 0 0 1 1 1 0 0 0 0 0 0 0 1 1 0 0 1 1 0 1 0 1 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 1 1 0 0 0 0 0 1 0
 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 1 0 1 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0
 0 1 0 0 0 0 1 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1
 0 0 0 0 0 1]
[[85 16]
 [37 16]]
0.6558441558441559
              precision    recall  f1-score   support

           0       0.70      0.84      0.76       101
           1       0.50      0.30      0.38        53

    accuracy                           0.66       154
   macro avg       0.60      0.57      0.57       154
weighted avg       0.63      0.66      0.63       154
```

K-Nearest Neighbor (After VIF)

```
[96] from sklearn.neighbors import KNeighborsClassifier
     from sklearn.metrics import accuracy_score
     from sklearn.metrics import confusion_matrix
     from sklearn.metrics import classification_report
     model=KNeighborsClassifier(n_neighbors=25)
```

80-20 Train Test Split

```
[97] model.fit(X_train1_nomulti, y_train1_nomulti)
     y_pred1_nomulti = model.predict(X_test1_nomulti)
     print(y_pred1_nomulti)

     knn = pd.DataFrame({'Predicted':y_pred1_nomulti,'Actual':y_test1_nomulti})
     print(knn)
     print(accuracy_score(y_test1_nomulti,y_pred1_nomulti))
     cm = confusion_matrix(y_test1_nomulti,y_pred1_nomulti)
     print(cm)

     sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=["No Diabetes", "Diabetes"], yticklabels=["No Diabetes", "Diabetes"])
     plt.title('Confusion Matrix')
     plt.xlabel("Predicted")
     plt.ylabel("Truth")
     plt.show()

     classification_rep = classification_report(y_test1_nomulti, y_pred1_nomulti)
     print(classification_rep)
```

```
[0 0 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 1 0 1 0 0 1 0
 0 0 0 0 0 0 1 0 0 1 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0
 1 0 0 0 0 0 0 0 1 0 0 0 0 1 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
 0 1 0 0 0 0 0 0 0 0 1 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 0 0 0 0 1]
     Predicted  Actual
723          0       0
234          0       0
703          0       0
105          1       0
638          0       1
..         ...     ...
699          0       0
484          0       1
122          0       0
716          0       1
314          1       1

[154 rows x 2 columns]
0.6168831168831169
[[85 16]
 [43 10]]
```

Support Vector Machine (After VIF)

```
[101] import numpy as np
      import pandas as pd
      import seaborn as sns
      %matplotlib inline
      import matplotlib.pyplot as plt
      from sklearn.svm import SVC
      from sklearn.model_selection import train_test_split
      from sklearn.metrics import classification_report, confusion_matrix
```

```
[102] model = SVC(kernel='linear')
```

80-20 Train Test Split

```
[103] model.fit(X_train1_nomulti, y_train1_nomulti)
      y_pred1_nomulti = model.predict(X_test1_nomulti)
      print(y_pred1_nomulti)

      svm = pd.DataFrame({'Predicted':y_pred1_nomulti,'Actual':y_test1_nomulti})
      print(svm)
      print(accuracy_score(y_test1_nomulti,y_pred1_nomulti))
      cm = confusion_matrix(y_test1_nomulti,y_pred1_nomulti)
      print(cm)

      sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=["No Diabetes", "Diabetes"], yticklabels=["No Diabetes", "Diabetes"])
      plt.title('Confusion Matrix')
      plt.xlabel("Predicted")
      plt.ylabel("Truth")
```

```
[0 0 0 0 0 1 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 1 1 0 0 0 0 0 1 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0
 0 0 0 0 0 0]
     Predicted  Actual
723          0       0
234          0       0
703          0       0
105          0       0
638          0       1
..         ...     ...
699          0       0
484          0       1
122          0       0
716          0       1
314          0       1

[154 rows x 2 columns]
0.6883116883116883
[[94  7]
 [41 12]]
Text(50.72222222222221, 0.5, 'Truth')
```