Description:

The objective of the dataset is to predict whether or not a patient has diabetes, based on certain diagnostic measurements included in the dataset. The datasets consists of several medical predictor variables and one target variable, Outcome. Predictor variables includes the number of pregnancies the patient has had, their BMI, insulin level, age, and so on.

Dataset url : https://www.kaggle.com/uciml/pima-indians-diabetes-database

Step 0: Import libraries and Dataset

```
# Importing libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.filterwarnings('ignore')

# Importing dataset
dataset = pd.read_csv('diabetes.csv')
dataset
```

```
     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]
```

Step 1: Descriptive Statistics

```
# Preview data
dataset.head()
```

```
   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


   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
```

```
# Dataset dimensions - (rows, columns)
dataset.shape
```

```
(768, 9)
```

```
# Features data-type
dataset.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

# Statistical summary
dataset.describe()

       Pregnancies     Glucose  BloodPressure  SkinThickness
Insulin  \
count   768.000000  768.000000     768.000000     768.000000
768.000000
mean      3.845052  120.894531      69.105469      20.536458
79.799479
std       3.369578   31.972618      19.355807      15.952218
115.244002
min       0.000000    0.000000       0.000000       0.000000
0.000000
25%       1.000000   99.000000      62.000000       0.000000
0.000000
50%       3.000000  117.000000      72.000000      23.000000
30.500000
75%       6.000000  140.250000      80.000000      32.000000
127.250000
max      17.000000  199.000000     122.000000      99.000000
846.000000

              BMI  DiabetesPedigreeFunction         Age     Outcome
count  768.000000                768.000000  768.000000  768.000000
mean    31.992578                  0.471876   33.240885    0.348958
std      7.884160                  0.331329   11.760232    0.476951
min      0.000000                  0.078000   21.000000    0.000000
25%     27.300000                  0.243750   24.000000    0.000000
50%     32.000000                  0.372500   29.000000    0.000000
75%     36.600000                  0.626250   41.000000    1.000000
max     67.100000                  2.420000   81.000000    1.000000
```

```
# Count of null values
dataset.isnull().sum()
```

```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

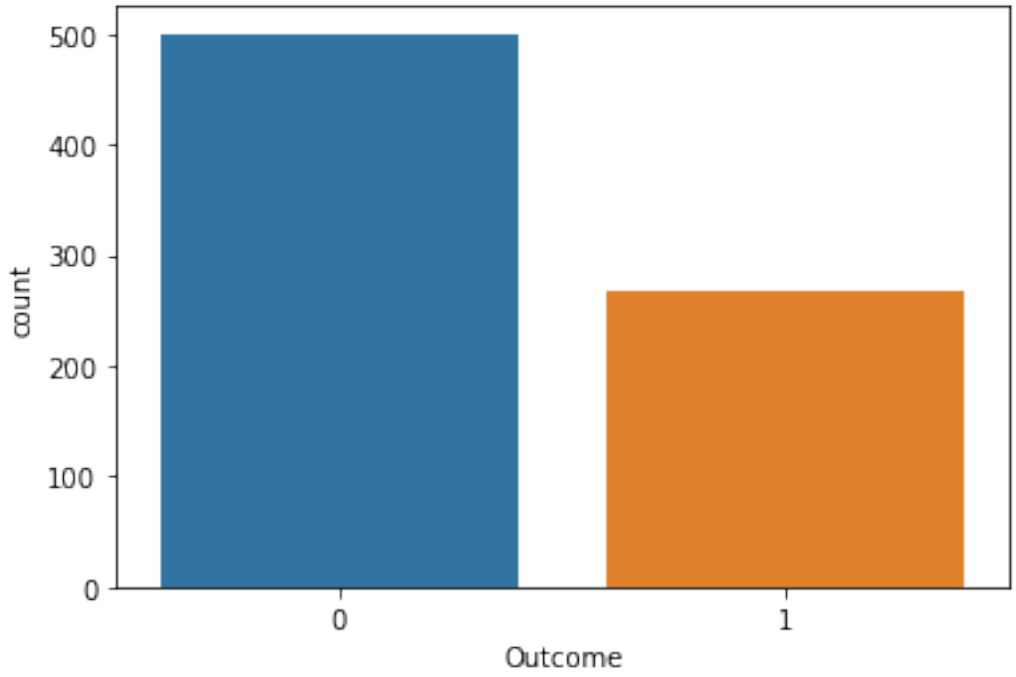
Observations:

1. There are a total of 768 records and 9 features in the dataset.
2. Each feature can be either of integer or float datatype.
3. Some features like Glucose, Blood pressure , Insulin, BMI have zero values which represent missing data.
4. There are zero NaN values in the dataset.
5. In the outcome column, 1 represents diabetes positive and 0 represents diabetes negative.

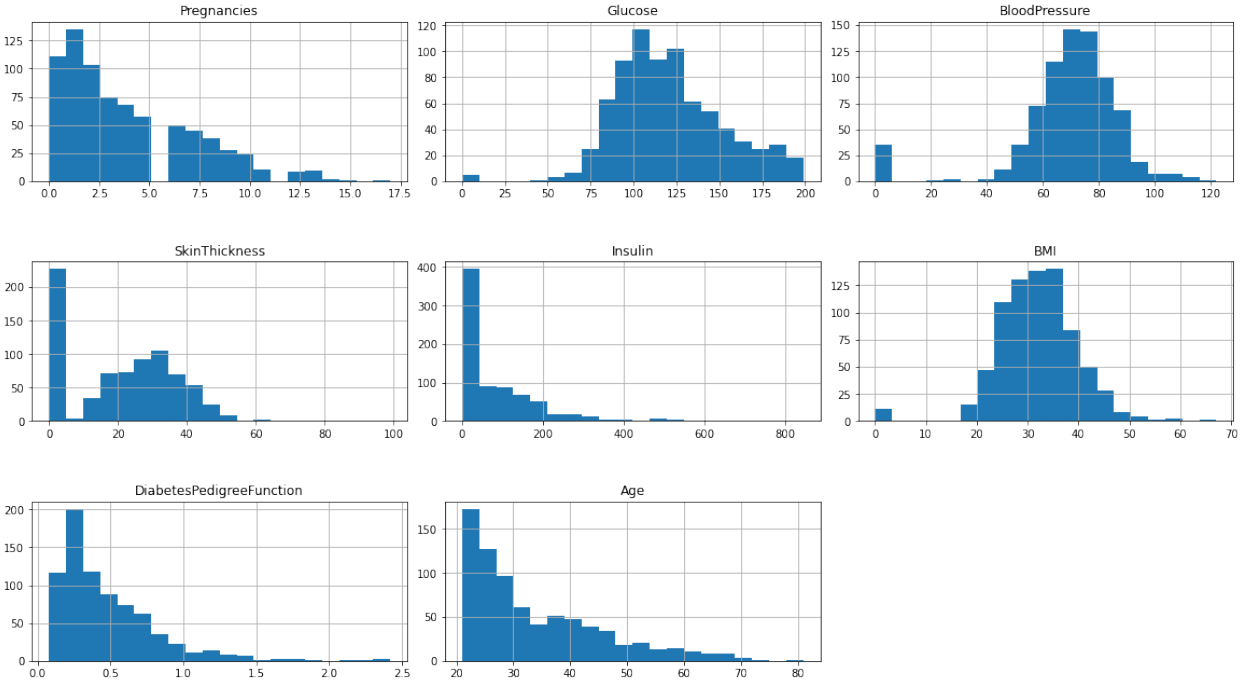
Step 2: Data Visualization

```
# Outcome countplot
sns.countplot(x = 'Outcome',data = dataset)
```

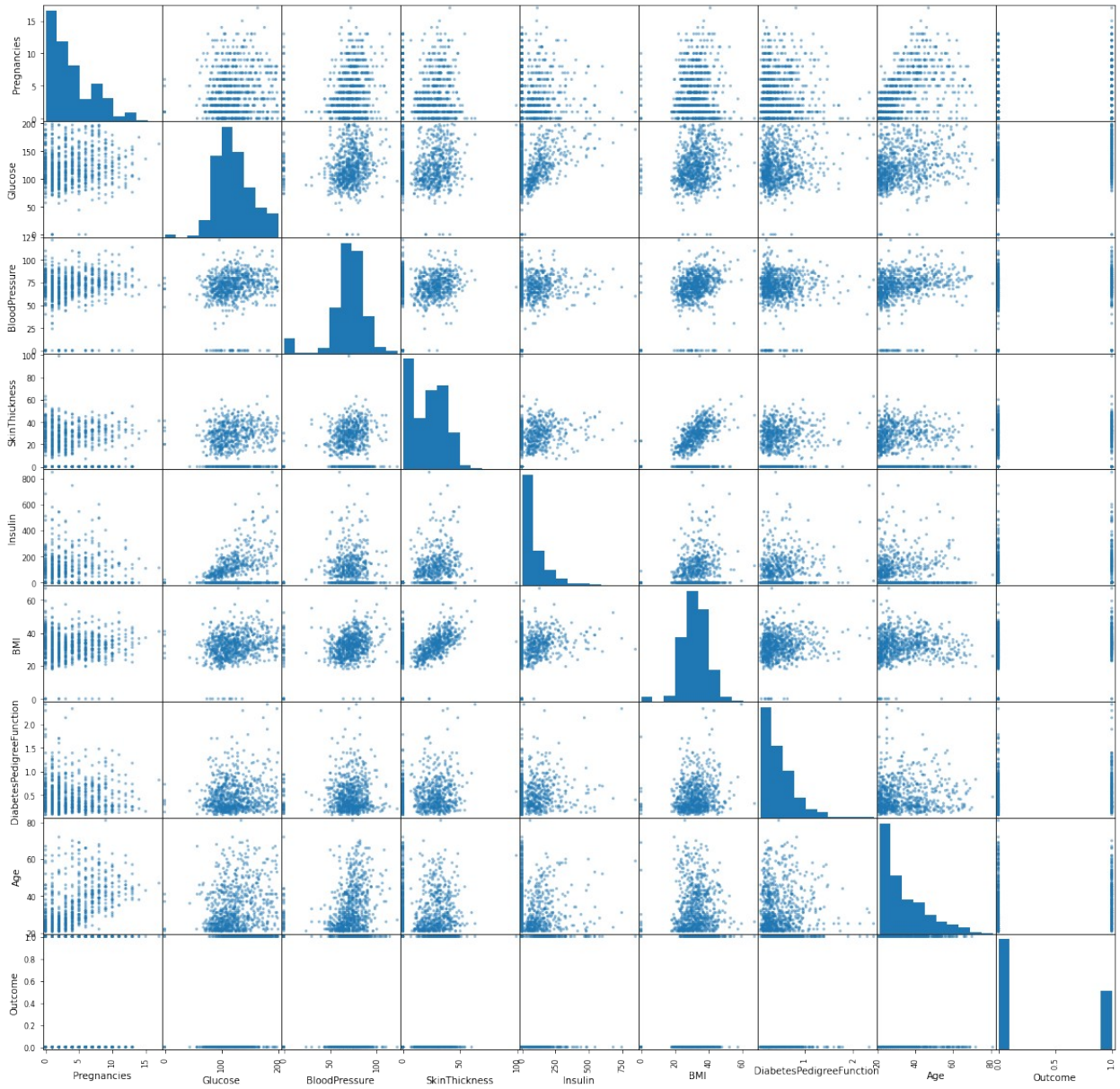
```
<AxesSubplot:xlabel='Outcome', ylabel='count'>
```



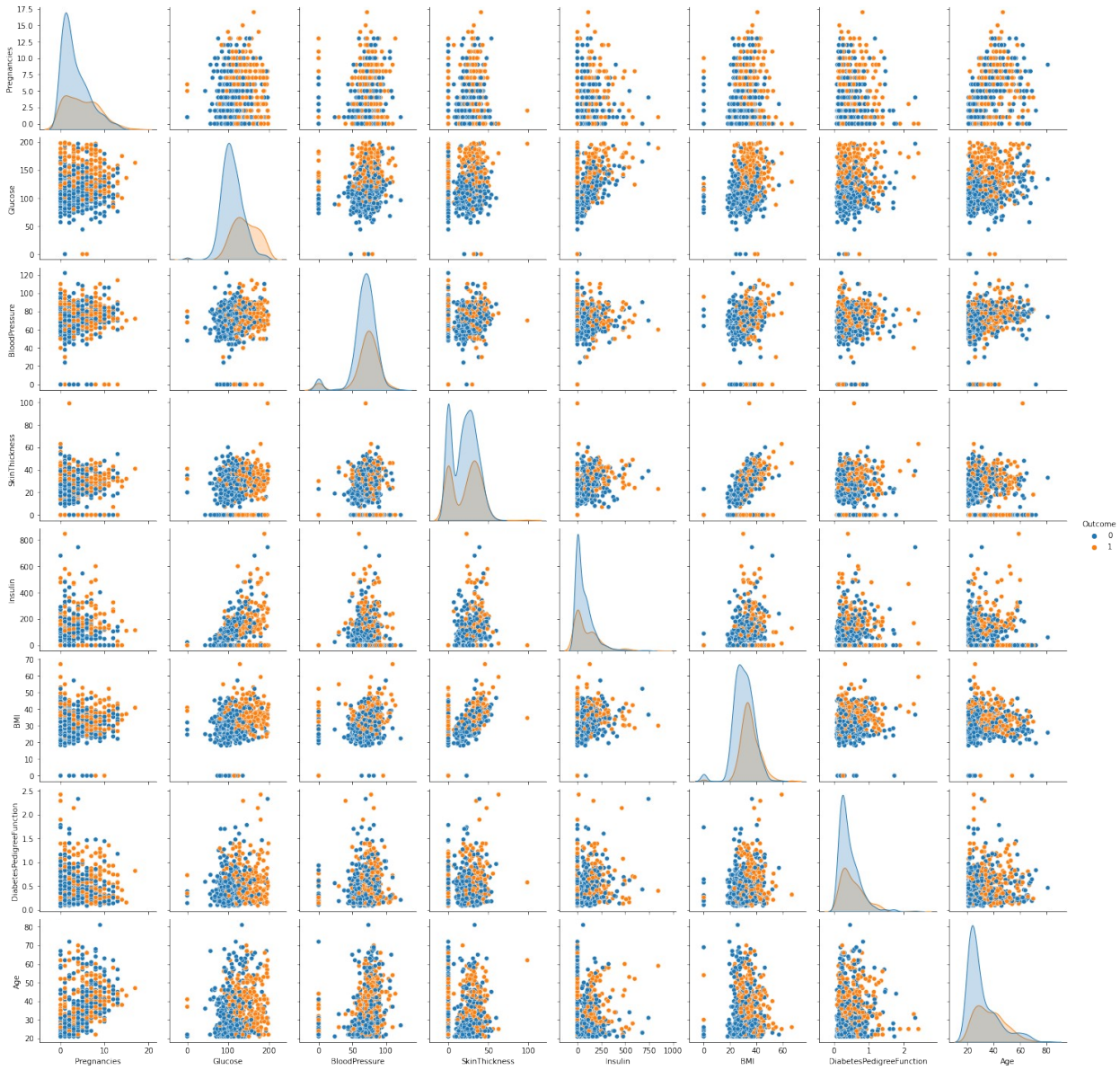

```
# Histogram of each feature
import itertools

col = dataset.columns[:8]
plt.subplots(figsize = (20, 15))
length = len(col)

for i, j in itertools.zip_longest(col, range(length)):
    plt.subplot((length//2), 3, j + 1)
    plt.subplots_adjust(wspace = 0.1,hspace = 0.5)
    dataset[i].hist(bins = 20)
    plt.title(i)
plt.show()
```



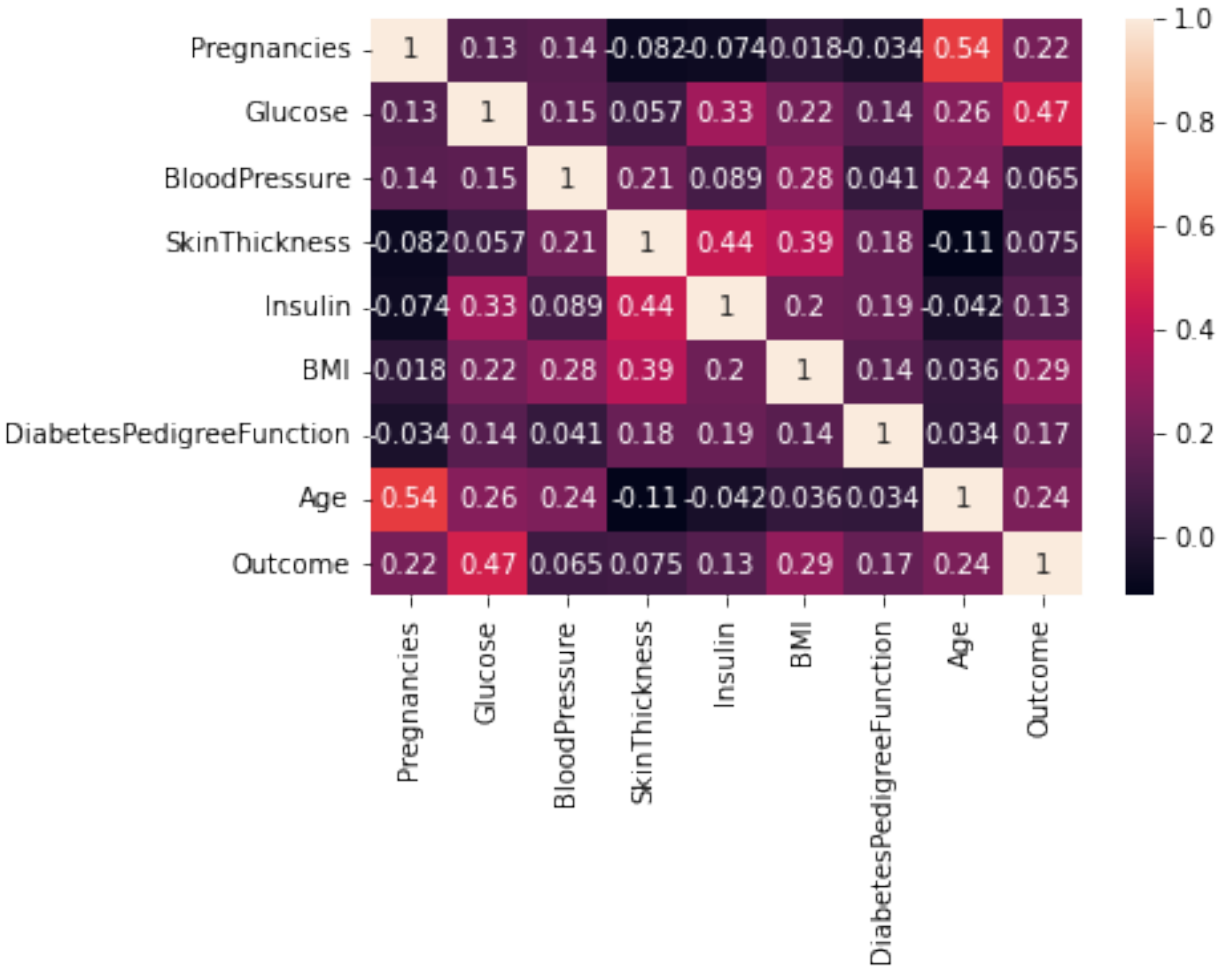

```
# Scatter plot matrix
from pandas.plotting import scatter_matrix
scatter_matrix(dataset, figsize = (20, 20));
```




```
# Pairplot
sns.pairplot(data = dataset, hue = 'Outcome')
plt.show()
```




```
# Heatmap
sns.heatmap(dataset.corr(), annot = True)
plt.show()
```



Observations:

1. The countplot tells us that the dataset is imbalanced, as number of patients who don't have diabetes is more than those who do.
2. From the correaltion heatmap, we can see that there is a high correlation between Outcome and [Glucose,BMI,Age,Insulin]. We can select these features to accept input from the user and predict the outcome.

Step 3: Data Preprocessing

```
dataset_new = dataset

# Replacing zero values with NaN
dataset_new[["Glucose", "BloodPressure", "SkinThickness", "Insulin",
"BMI"]] = dataset_new[["Glucose", "BloodPressure", "SkinThickness",
"Insulin", "BMI"]].replace(0, np.NaN)

# Count of NaN
dataset_new.isnull().sum()
```

```
Pregnancies                   0
Glucose                       5
BloodPressure                35
SkinThickness               227
Insulin                     374
BMI                          11
DiabetesPedigreeFunction      0
Age                           0
Outcome                       0
dtype: int64
```

```
# Replacing NaN with mean values
dataset_new["Glucose"].fillna(dataset_new["Glucose"].mean(), inplace =
True)
dataset_new["BloodPressure"].fillna(dataset_new["BloodPressure"].mean(
), inplace = True)
dataset_new["SkinThickness"].fillna(dataset_new["SkinThickness"].mean(
), inplace = True)
dataset_new["Insulin"].fillna(dataset_new["Insulin"].mean(), inplace =
True)
dataset_new["BMI"].fillna(dataset_new["BMI"].mean(), inplace = True)
```

```
# Statistical summary
dataset_new.describe()
```

```
       Pregnancies     Glucose  BloodPressure  SkinThickness
Insulin  \
count   768.000000  768.000000     768.000000     768.000000
768.000000
mean      3.845052  121.686763      72.405184      29.153420
155.548223
std       3.369578   30.435949      12.096346       8.790942
85.021108
min       0.000000   44.000000      24.000000       7.000000
14.000000
25%       1.000000   99.750000      64.000000      25.000000
121.500000
50%       3.000000  117.000000      72.202592      29.153420
155.548223
75%       6.000000  140.250000      80.000000      32.000000
155.548223
max      17.000000  199.000000     122.000000      99.000000
846.000000

              BMI  DiabetesPedigreeFunction         Age     Outcome
count  768.000000                768.000000  768.000000  768.000000
mean    32.457464                  0.471876   33.240885    0.348958
std      6.875151                  0.331329   11.760232    0.476951
min     18.200000                  0.078000   21.000000    0.000000
25%     27.500000                  0.243750   24.000000    0.000000
```

```
50%     32.400000                   0.372500   29.000000    0.000000
75%     36.600000                   0.626250   41.000000    1.000000
max     67.100000                   2.420000   81.000000    1.000000
```

```
# Feature scaling using MinMaxScaler
from sklearn.preprocessing import MinMaxScaler
sc = MinMaxScaler(feature_range = (0, 1))
dataset_scaled = sc.fit_transform(dataset_new)

dataset_scaled = pd.DataFrame(dataset_scaled)

# Selecting features - [Glucose, Insulin, BMI, Age]
X = dataset_scaled.iloc[:, [1, 4, 5, 7]].values
Y = dataset_scaled.iloc[:, 8].values

# Splitting X and Y
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size =
0.20, random_state = 42, stratify = dataset_new['Outcome'] )

# Checking dimensions
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("Y_train shape:", Y_train.shape)
print("Y_test shape:", Y_test.shape)
```

```
X_train shape: (614, 4)
X_test shape: (154, 4)
Y_train shape: (614,)
Y_test shape: (154,)
```

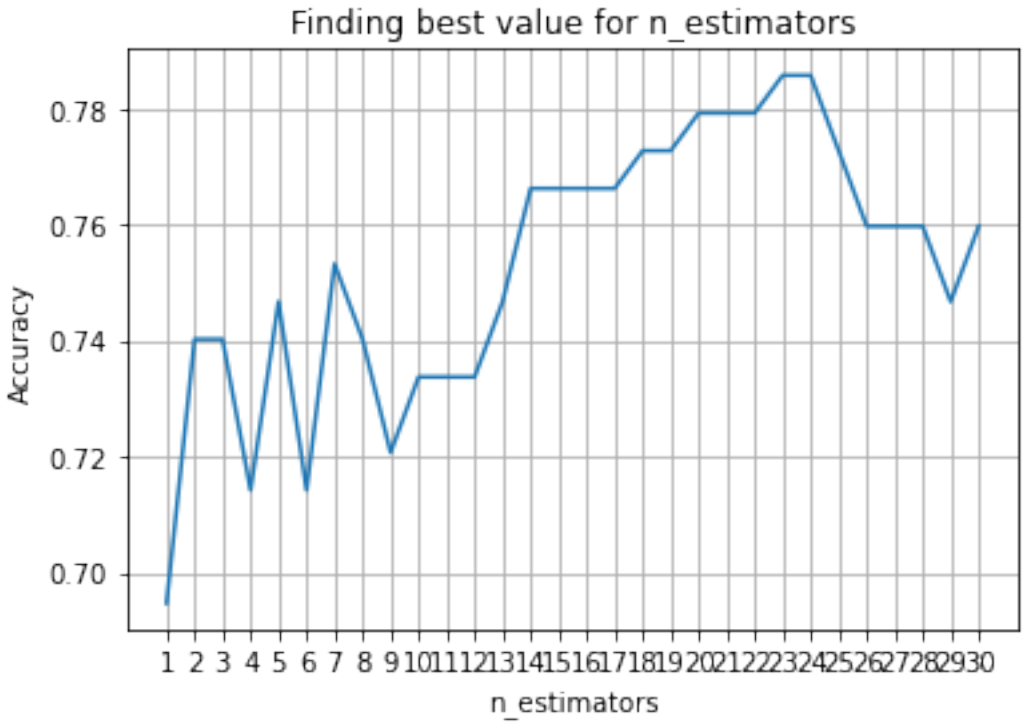
Step 4: Data Modelling

```
# Logistic Regression Algorithm
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression(random_state = 42)
logreg.fit(X_train, Y_train)
```

```
LogisticRegression(random_state=42)
```

```
# Plotting a graph for n_neighbors
from sklearn import metrics
from sklearn.neighbors import KNeighborsClassifier

X_axis = list(range(1, 31))
acc = pd.Series()
x = range(1,31)

for i in list(range(1, 31)):
```

```
    knn_model = KNeighborsClassifier(n_neighbors = i)
    knn_model.fit(X_train, Y_train)
    prediction = knn_model.predict(X_test)
    acc = acc.append(pd.Series(metrics.accuracy_score(prediction,
Y_test)))
plt.plot(X_axis, acc)
plt.xticks(x)
plt.title("Finding best value for n_estimators")
plt.xlabel("n_estimators")
plt.ylabel("Accuracy")
plt.grid()
plt.show()
print('Highest value: ',acc.values.max())
```




```
Highest value:  0.7857142857142857
```

```
# K nearest neighbors Algorithm
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors = 24, metric = 'minkowski', p =
2)
knn.fit(X_train, Y_train)
```

```
KNeighborsClassifier(n_neighbors=24)
```

```
# Decision tree Algorithm
from sklearn.tree import DecisionTreeClassifier
dectree = DecisionTreeClassifier(criterion = 'entropy', random_state =
```

```
42)
dectree.fit(X_train, Y_train)
```

```
DecisionTreeClassifier(criterion='entropy', random_state=42)
```

```
# Random forest Algorithm
from sklearn.ensemble import RandomForestClassifier
ranfor = RandomForestClassifier(n_estimators = 11, criterion =
'entropy', random_state = 42)
ranfor.fit(X_train, Y_train)
```

```
RandomForestClassifier(criterion='entropy', n_estimators=11,
random_state=42)
```

```
# Making predictions on test dataset
Y_pred_logreg = logreg.predict(X_test)
Y_pred_knn = knn.predict(X_test)
Y_pred_dectree = dectree.predict(X_test)
Y_pred_ranfor = ranfor.predict(X_test)
```

Step 5: Model Evaluation

```
# Evaluating using accuracy_score metric
from sklearn.metrics import accuracy_score
accuracy_logreg = accuracy_score(Y_test, Y_pred_logreg)
accuracy_knn = accuracy_score(Y_test, Y_pred_knn)
accuracy_dectree = accuracy_score(Y_test, Y_pred_dectree)
accuracy_ranfor = accuracy_score(Y_test, Y_pred_ranfor)

# Accuracy on test set
print("Logistic Regression: " + str(accuracy_logreg * 100))
print("K Nearest neighbors: " + str(accuracy_knn * 100))
print("Decision tree: " + str(accuracy_dectree * 100))
print("Random Forest: " + str(accuracy_ranfor * 100))
```

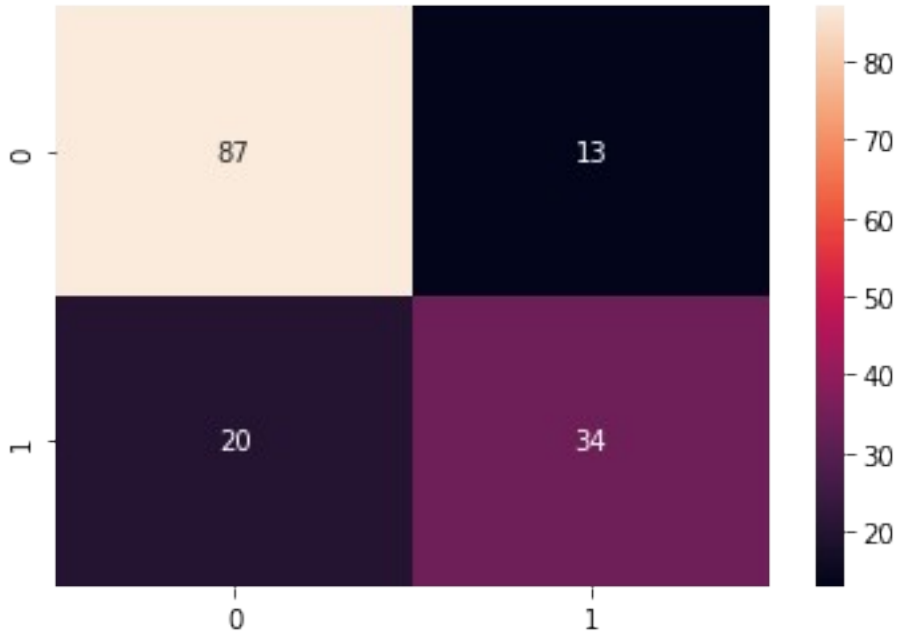
```
Logistic Regression: 72.07792207792207
K Nearest neighbors: 78.57142857142857
Decision tree: 68.18181818181817
Random Forest: 75.97402597402598
```

```
#From the above comparison, we can observe that K Nearest neighbors
gets the highest accuracy of 78.57 %

# Confusion matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(Y_test, Y_pred_knn)
cm
```

```
array([[87, 13],
       [20, 34]], dtype=int64)
```

```
# Heatmap of Confusion matrix
sns.heatmap(pd.DataFrame(cm), annot=True)
```

```
<AxesSubplot:>
```




```
# Classification report
from sklearn.metrics import classification_report
print(classification_report(Y_test, Y_pred_knn))
```

```
              precision    recall  f1-score   support

         0.0       0.81      0.87      0.84       100
         1.0       0.72      0.63      0.67        54

    accuracy                           0.79       154
   macro avg       0.77      0.75      0.76       154
weighted avg       0.78      0.79      0.78       154
```