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')
```

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):
Pregnancies                 768 non-null int64
Glucose                     768 non-null int64
BloodPressure               768 non-null int64
SkinThickness               768 non-null int64
Insulin                     768 non-null int64
BMI                         768 non-null float64
DiabetesPedigreeFunction    768 non-null float64
Age                         768 non-null int64
Outcome                     768 non-null int64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

```
# Statistical summary
dataset.describe().T
```

```
                           count        mean         std     min
25%  \
Pregnancies                768.0    3.845052    3.369578   0.000
1.00000
Glucose                    768.0  120.894531   31.972618   0.000
99.00000
BloodPressure              768.0   69.105469   19.355807   0.000
62.00000
SkinThickness              768.0   20.536458   15.952218   0.000
0.00000
Insulin                    768.0   79.799479  115.244002   0.000
0.00000
BMI                        768.0   31.992578    7.884160   0.000
27.30000
DiabetesPedigreeFunction   768.0    0.471876    0.331329   0.078
0.24375
Age                        768.0   33.240885   11.760232  21.000
24.00000
Outcome                    768.0    0.348958    0.476951   0.000
0.00000

                              50%        75%     max
```

```
Pregnancies                 3.0000    6.00000   17.00
Glucose                   117.0000  140.25000  199.00
BloodPressure              72.0000   80.00000  122.00
SkinThickness              23.0000   32.00000   99.00
Insulin                    30.5000  127.25000  846.00
BMI                        32.0000   36.60000   67.10
DiabetesPedigreeFunction    0.3725    0.62625    2.42
Age                        29.0000   41.00000   81.00
Outcome                     0.0000    1.00000    1.00
```

```
# 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
```

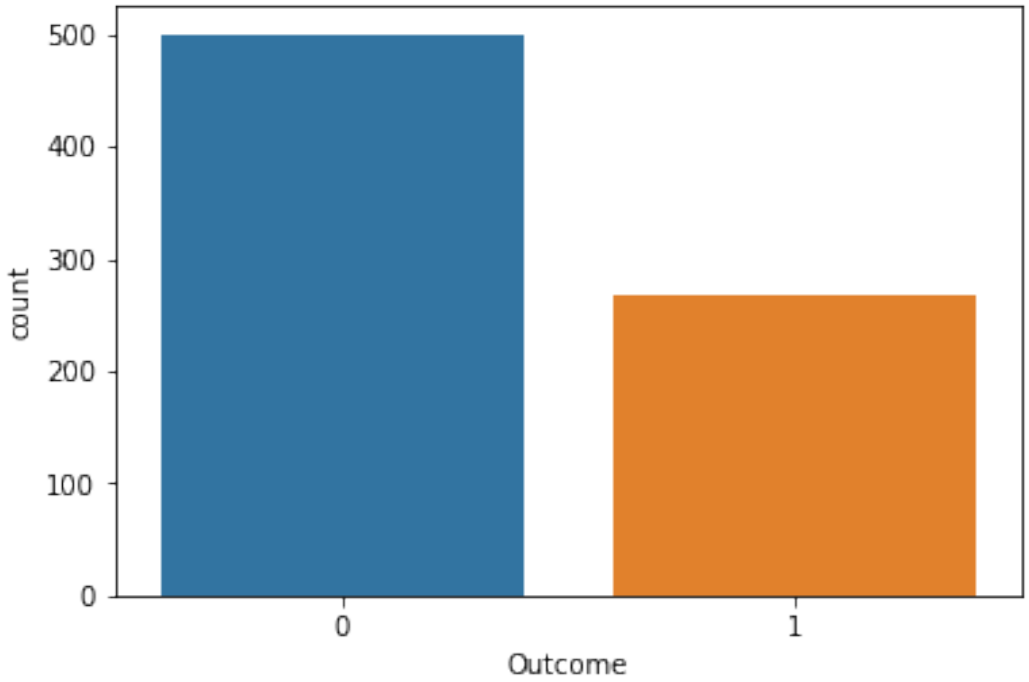
Observations:

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.

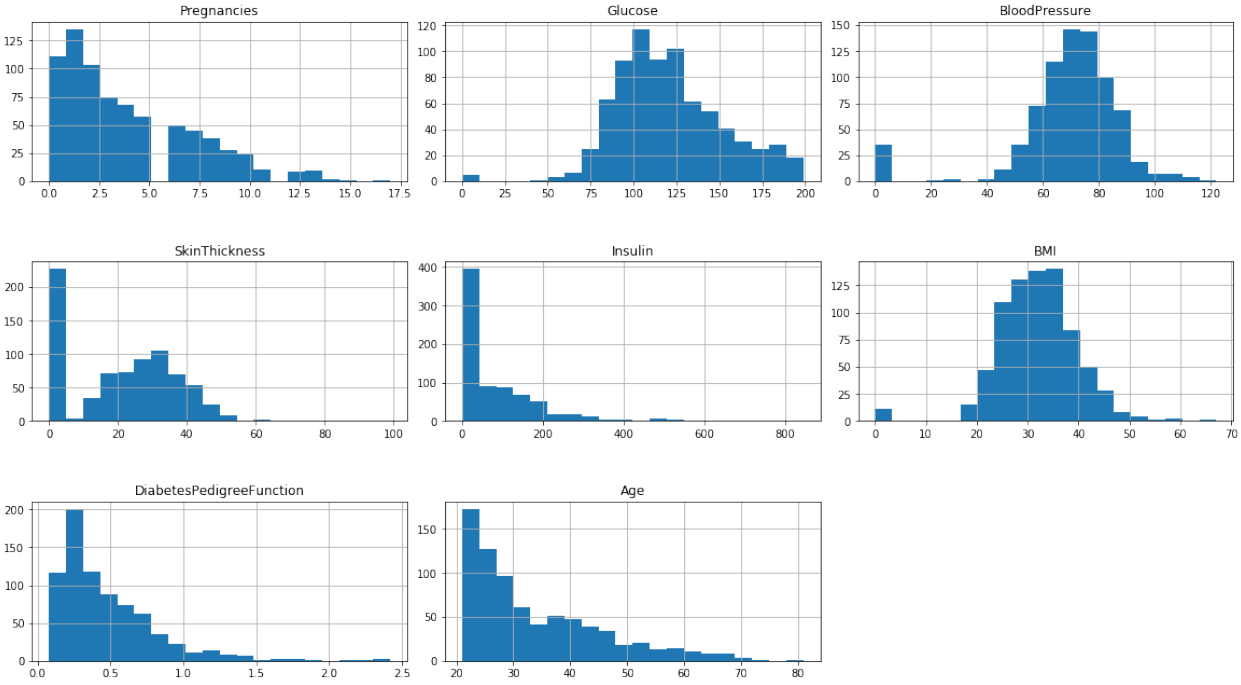
Step 2: Data Visualization

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

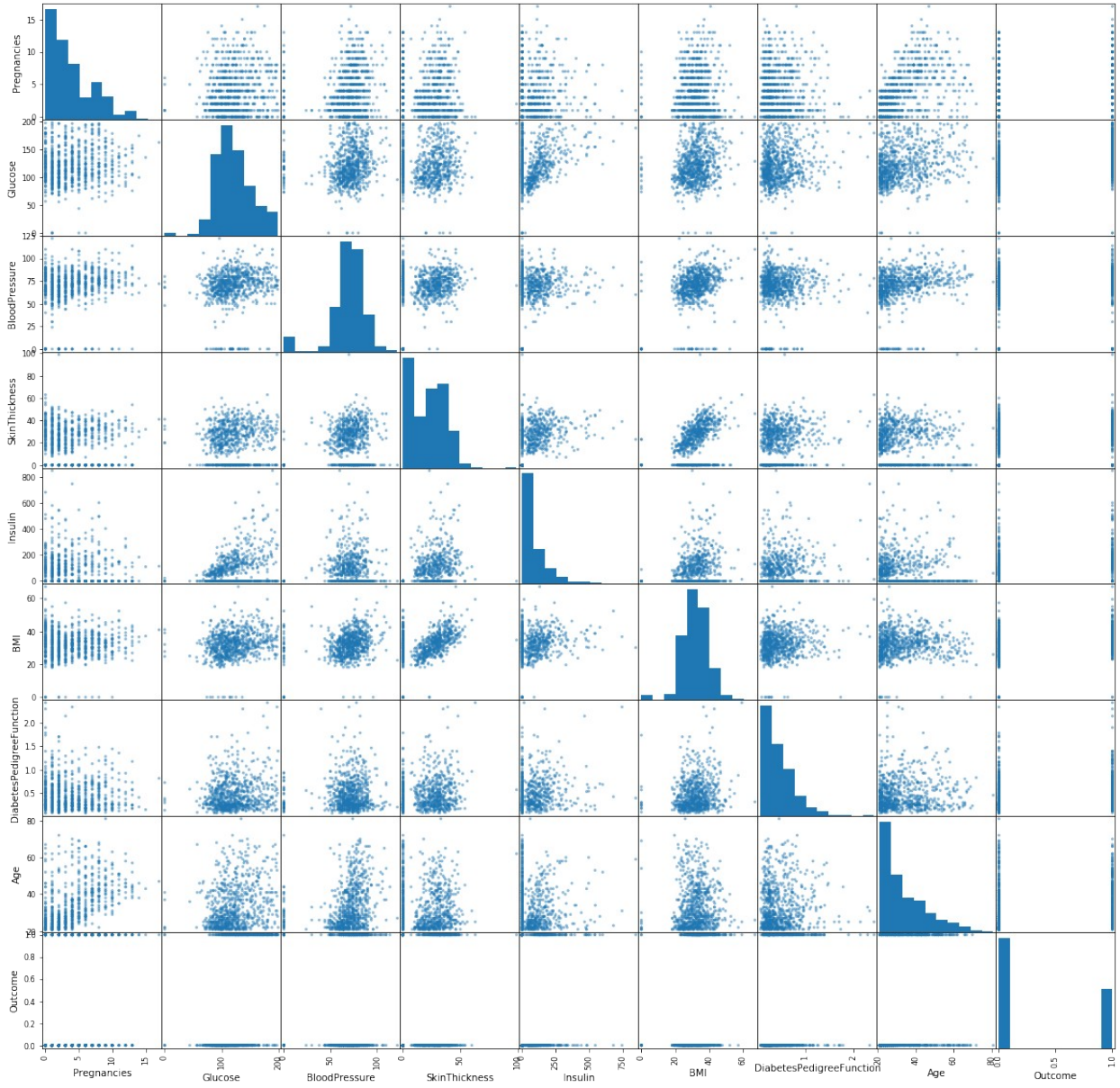
```
<matplotlib.axes._subplots.AxesSubplot at 0x1976214b128>
```



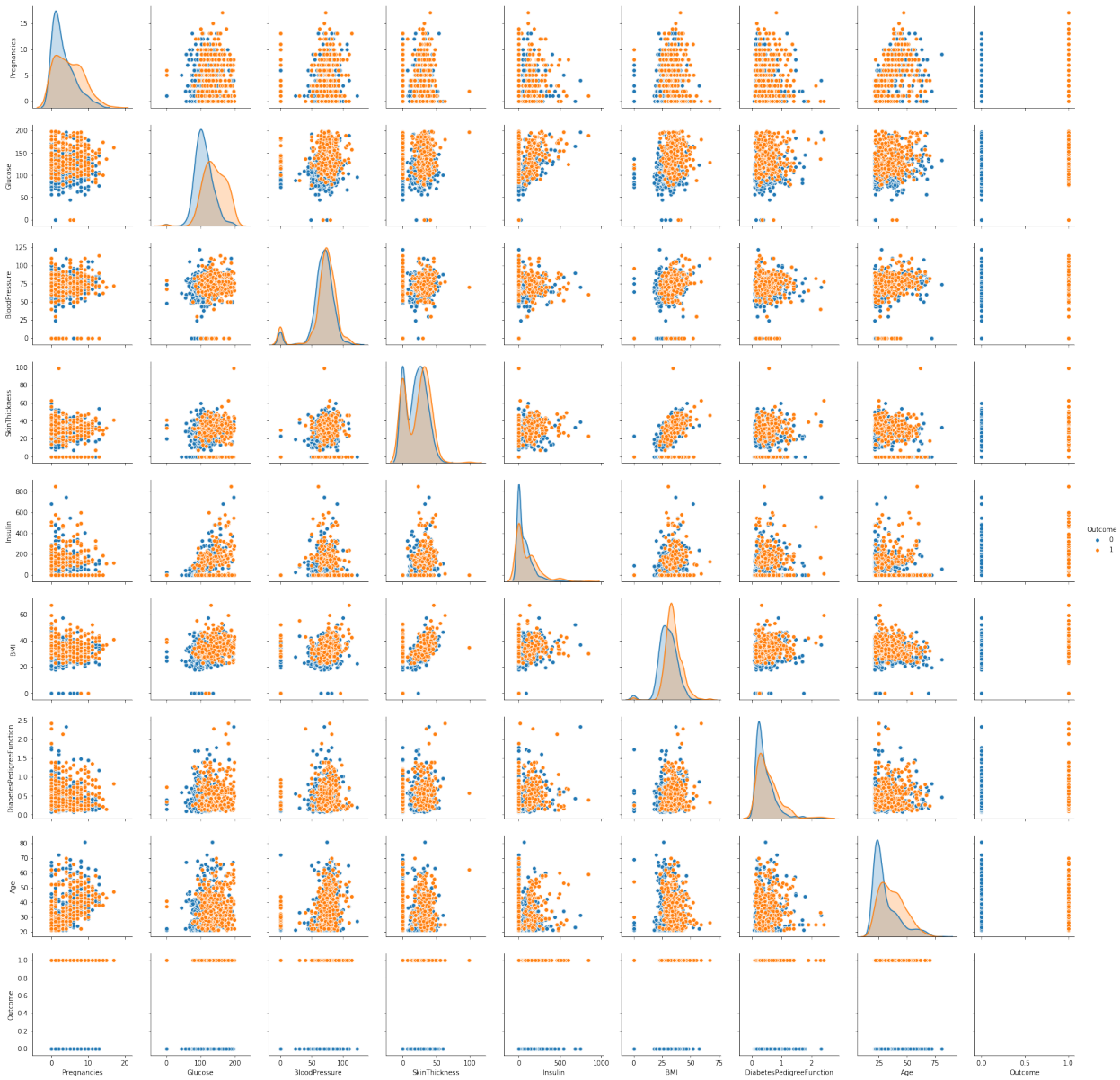

```
# Histogram of each feature
import itertools

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()
```



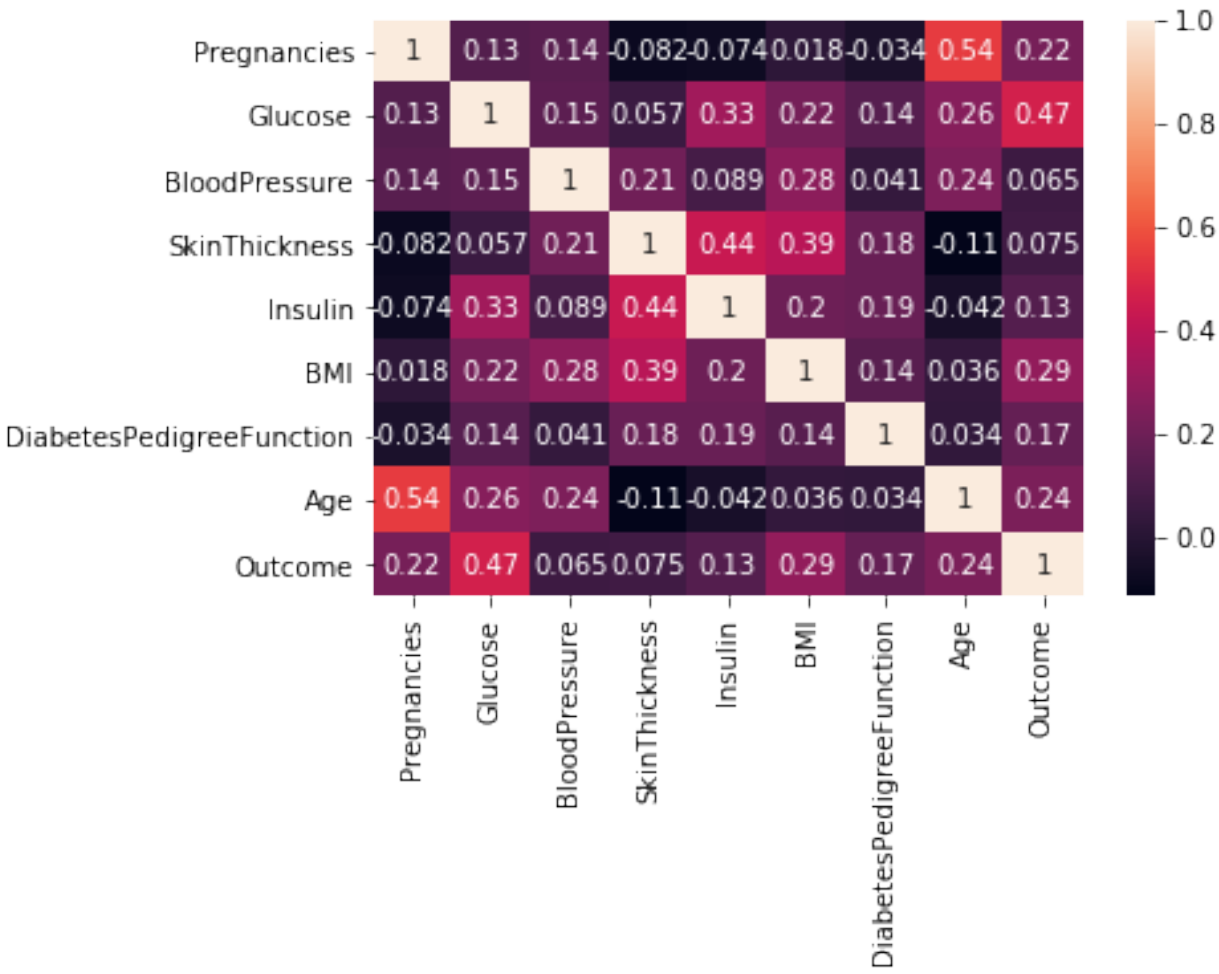

```
# Scatter plot matrix
from pandas.tools.plotting import scatter_matrix
scatter_matrix(dataset, figsize = (20, 20));
```




```
# 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().T
```

```
                          count        mean        std     min
25%  \
Pregnancies               768.0    3.845052   3.369578   0.000
1.00000
Glucose                   768.0  121.686763  30.435949  44.000
99.75000
BloodPressure             768.0   72.405184  12.096346  24.000
64.00000
SkinThickness             768.0   29.153420   8.790942   7.000
25.00000
Insulin                   768.0  155.548223  85.021108  14.000
121.50000
BMI                       768.0   32.457464   6.875151  18.200
27.50000
DiabetesPedigreeFunction  768.0    0.471876   0.331329   0.078
0.24375
Age                       768.0   33.240885  11.760232  21.000
24.00000
Outcome                   768.0    0.348958   0.476951   0.000
0.00000

                                50%         75%     max
Pregnancies                3.000000    6.000000   17.00
Glucose                  117.000000  140.250000  199.00
BloodPressure             72.202592   80.000000  122.00
```

```
SkinThickness               29.153420   32.000000   99.00
Insulin                    155.548223  155.548223  846.00
BMI                         32.400000   36.600000   67.10
DiabetesPedigreeFunction     0.372500    0.626250    2.42
Age                         29.000000   41.000000   81.00
Outcome                      0.000000    1.000000    1.00
```

```
# 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,)
```

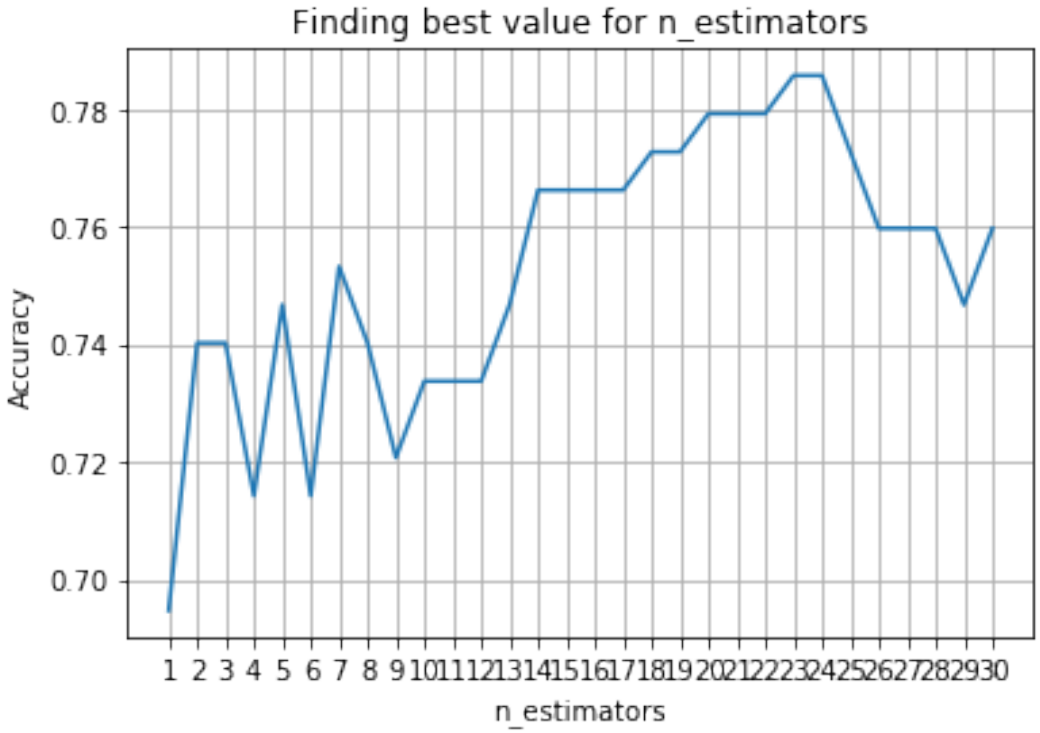
Step 4: Data Modelling

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

```
LogisticRegression(C=1.0, class_weight=None, dual=False,
fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=42, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)
```

```
# 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(algorithm='auto', leaf_size=30,
metric='minkowski',
           metric_params=None, n_jobs=None, n_neighbors=24, p=2,
           weights='uniform')

# Support Vector Classifier Algorithm
from sklearn.svm import SVC
svc = SVC(kernel = 'linear', random_state = 42)
svc.fit(X_train, Y_train)

SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
  kernel='linear', max_iter=-1, probability=False, random_state=42,
  shrinking=True, tol=0.001, verbose=False)

# Naive Bayes Algorithm
from sklearn.naive_bayes import GaussianNB
nb = GaussianNB()
nb.fit(X_train, Y_train)

GaussianNB(priors=None, var_smoothing=1e-09)

# Decision tree Algorithm
from sklearn.tree import DecisionTreeClassifier
dectree = DecisionTreeClassifier(criterion = 'entropy', random_state =
42)
dectree.fit(X_train, Y_train)

DecisionTreeClassifier(class_weight=None, criterion='entropy',
max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False,
random_state=42,
            splitter='best')

# 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(bootstrap=True, class_weight=None,
criterion='entropy',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
```

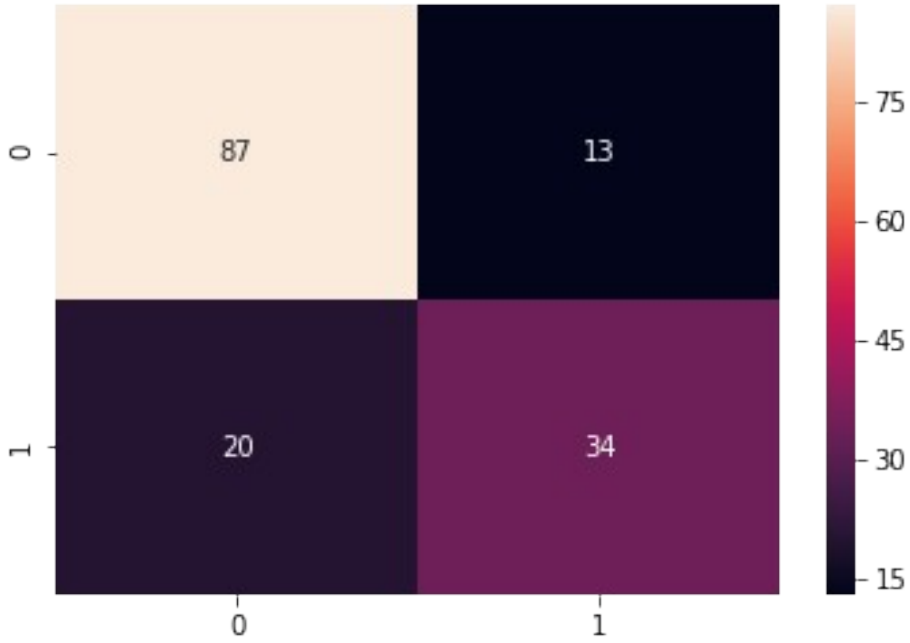
```
            min_weight_fraction_leaf=0.0, n_estimators=11,
n_jobs=None,
            oob_score=False, random_state=42, verbose=0,
warm_start=False)

# Making predictions on test dataset
Y_pred_logreg = logreg.predict(X_test)
Y_pred_knn = knn.predict(X_test)
Y_pred_svc = svc.predict(X_test)
Y_pred_nb = nb.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_svc = accuracy_score(Y_test, Y_pred_svc)
accuracy_nb = accuracy_score(Y_test, Y_pred_nb)
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("Support Vector Classifier: " + str(accuracy_svc * 100))
print("Naive Bayes: " + str(accuracy_nb * 100))
print("Decision tree: " + str(accuracy_dectree * 100))
print("Random Forest: " + str(accuracy_ranfor * 100))

Logistic Regression: 71.42857142857143
K Nearest neighbors: 78.57142857142857
Support Vector Classifier: 73.37662337662337
Naive Bayes: 71.42857142857143
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)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1976620cb70>
```




```
# 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

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