Data Science Project Report

Predict the onset of diabetes based on diagnostic data

Prepared by Sukanya Sadhu (sadh8004@gmail.com)

Motivation behind the Problem:

Diabetes is a global health concern with significant societal and economic implications. Early prediction can identify individuals at risk of developing diabetes and allows for early intervention and lifestyle modifications. Predictive models enable targeted screening and preventive measures for high-risk individuals, leading to more efficient resource allocation in healthcare settings. It can also inform public health policies and interventions aimed at reducing the prevalence of diabetes and its associated complications on a population level.

The Approach:

This project explores the application of machine learning techniques to predict the onset of diabetes based on diagnostic data. The dataset contains various medical attributes such as,

- **Pregnancies:** Number of times a woman has been pregnant
- **Glucose:** Blood glucose level
- **BloodPressure:** Blood pressure measurement in mm Hg
- **SkinThickness:** Skin thickness (mm)
- **Insulin:** Blood insulin level (μU/mL)
- **BMI:** Body mass index (calculated from weight and height)
- **DiabetesPedigreeFunction:** Function score based on family history of diabetes
- **Age:** Patient's age (years)
- **Outcome:** Indicates presence of diabetes (1 - Yes, 0 - No)

I employed various supervised machine learning algorithms e.g. KNN, decision tree, random forest, logistic regression etc to develop predictive models. Through rigorous evaluation and comparison of multiple algorithms, I achieved promising results in identifying individuals at risks of developing diabetes. This findings can highlight the potential of machine learning in assisting healthcare professionals in early detection and intervention strategies for diabetes prevention.

Tools and technologies used:

Python: Programming language used for model development, data preprocessing.

Scikit-learn: Machine learning library used for model training, evaluation and prediction.

Pandas: Data manipulation library used for data preprocessing and analysis.

Numpy: Library for numerical computing used for handling arrays and mathematical operations.

Seaborn: For plotting and visualization.

Jupyter Notebook: Coding environment for Python.

Installation and Usage:

Download and unzip the project folder.

Open Jupyter Notebook and run 'EDA and Model building-diabetes.ipynb'

For any queries, please contact me at sadh8004@gmail.com & 7980722619.

Key Features:

Input Data Collection: The dataset is downloaded from Kaggle.com(Pima Indians Diabetes Database) and then imported to Jupyter Notebook via pandas library using 'read_csv' command.

Data Preprocessing: The input data is preprocessed to check for missing values or duplicates. However the data was already clean.

Data Normalization: Normalization using MinMaxScaler and StandardScaler has been done from sklearn library.

Machine Learning Models: Various machine learning algorithms are employed, including decision trees, random forests, KNN, and logistic regression, to build predictive models.

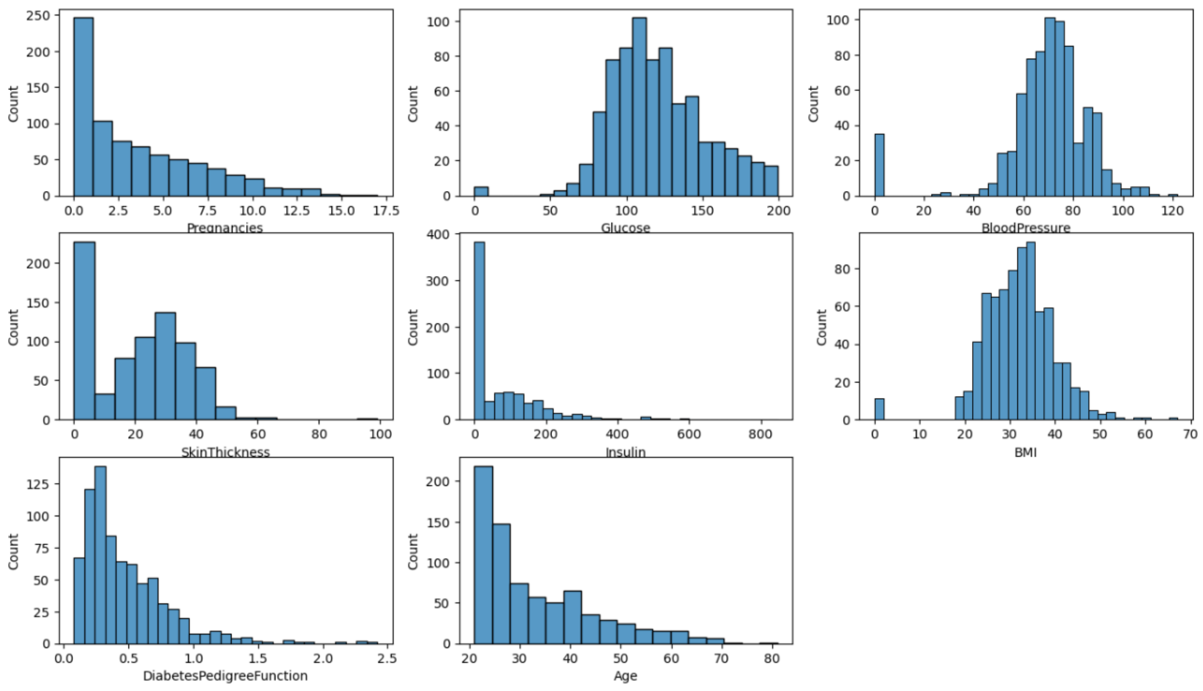
Model Training and Evaluation: The models are trained on historical data and evaluated using appropriate performance metrics to ensure accuracy and reliability.

Prediction of diabetes: Based on the trained models, the system recommends which patients are more likely to have diabetes with ~80% accuracy.

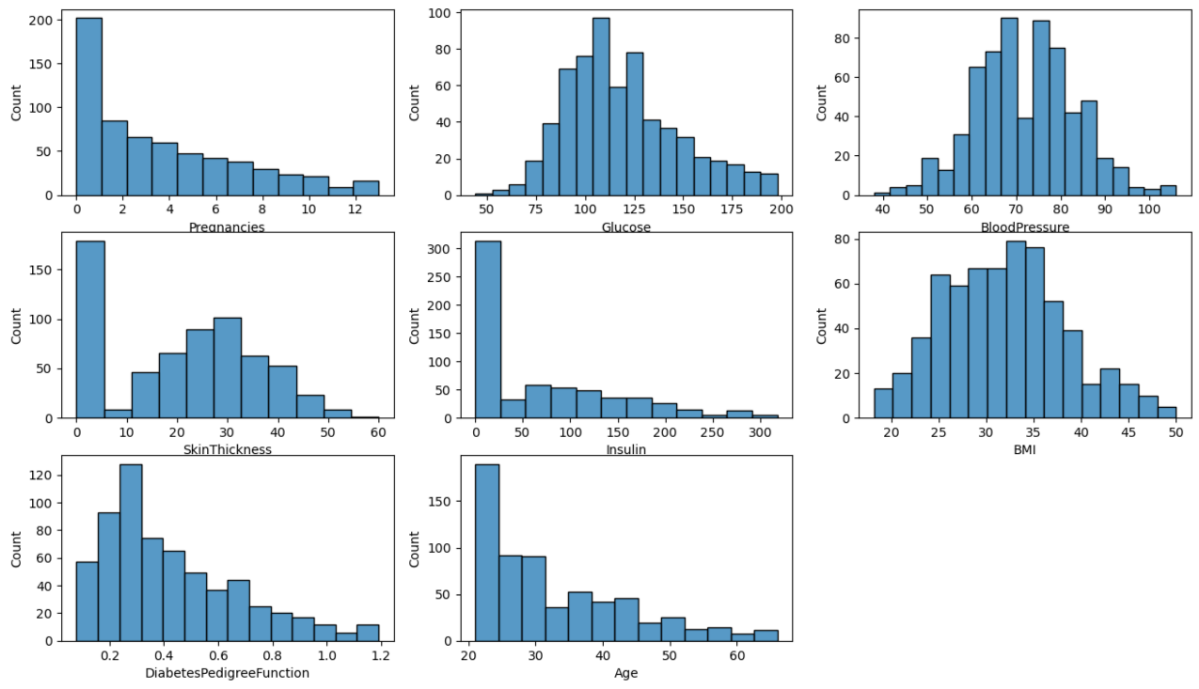
Outlier Treatment:

The features of the dataset have been checked for outliers using histograms and boxplots and corresponding outlier treatment has been done using IQR method.

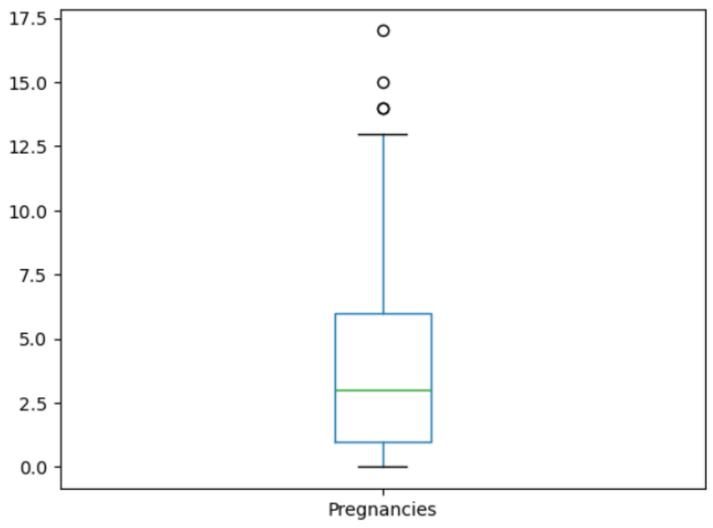
Before outlier treatment:



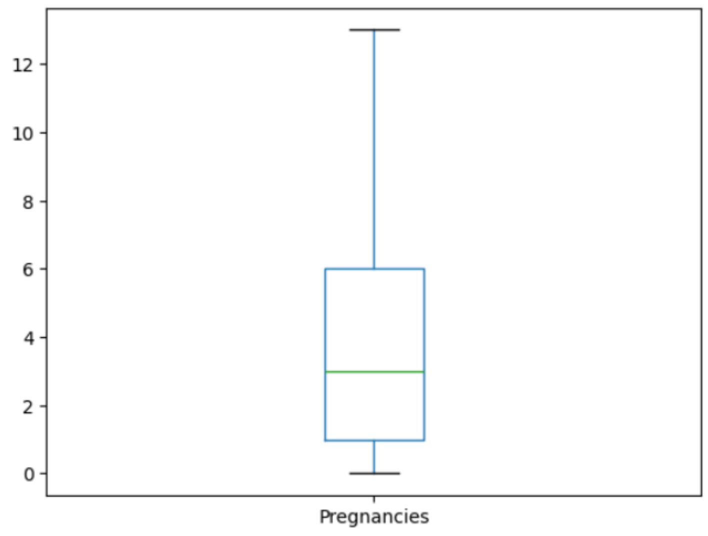
After outlier treatment:



Before outlier treatment:

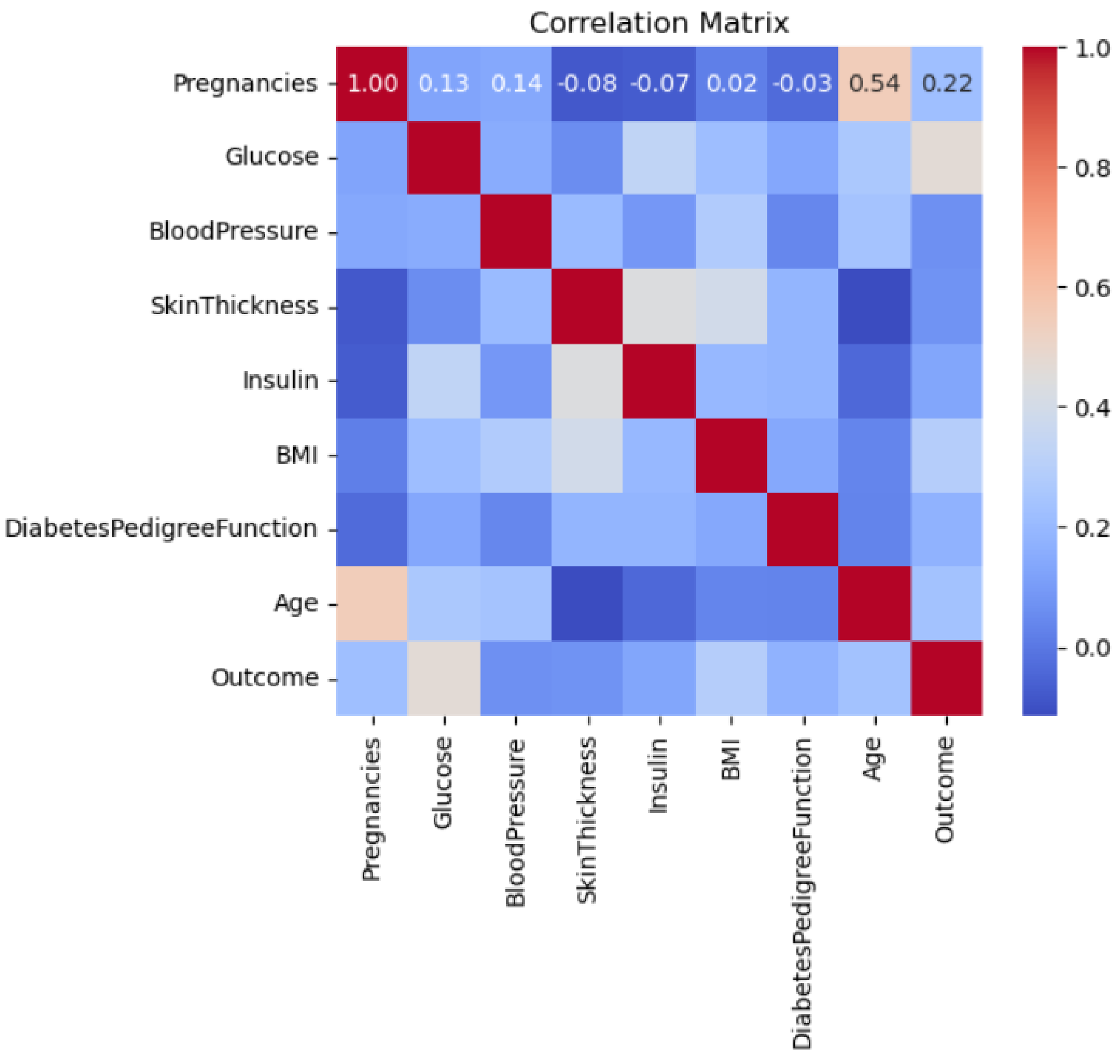


After outlier treatment:



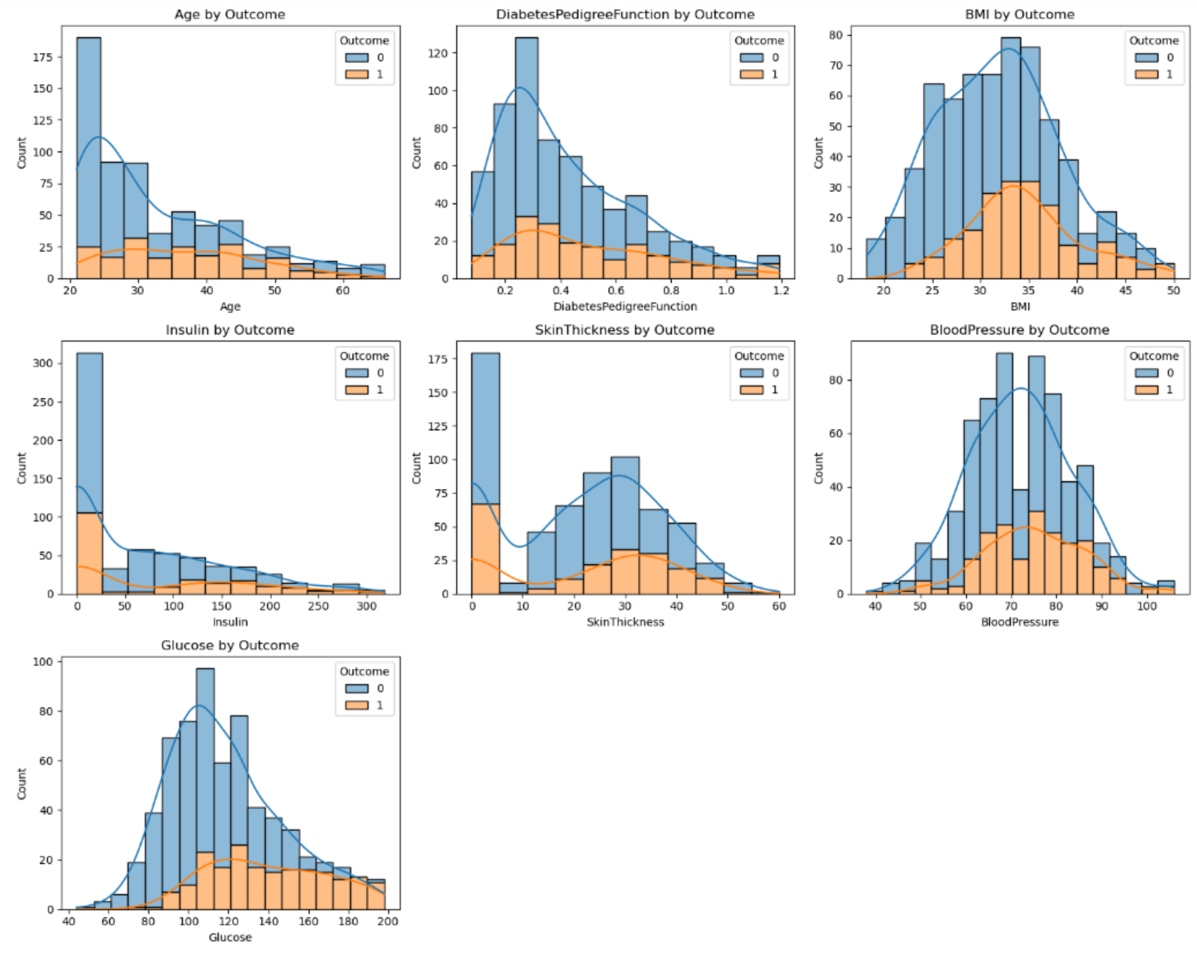
Data Visualization:

From the correlation heatmap plot by seaborn, we can see how the variables are correlated to each other. One important thing to note is that the outcome of whether a patient has diabetes or not is highly dependent on the Glucose levels.



I have divided the dataset based on the outcome (whether it is 0 or 1) and plotted multiple stacked histograms of all the features. Here blue mean outcome = 0 i.e. not diabetic and yellow histograms are for outcome = 1, i.e. diabetic.

By analyzing these stacked histograms, we can observe how the distribution of each feature (e.g., 'Age', 'BMI') varies across the two outcome groups. This visualization technique allows us to identify potential patterns or relationships between feature values and the presence or absence of diabetes.



Observations:

The key thing to note here is that the 'Glucose' histogram is skewed to the higher side for outcome 1 patients.

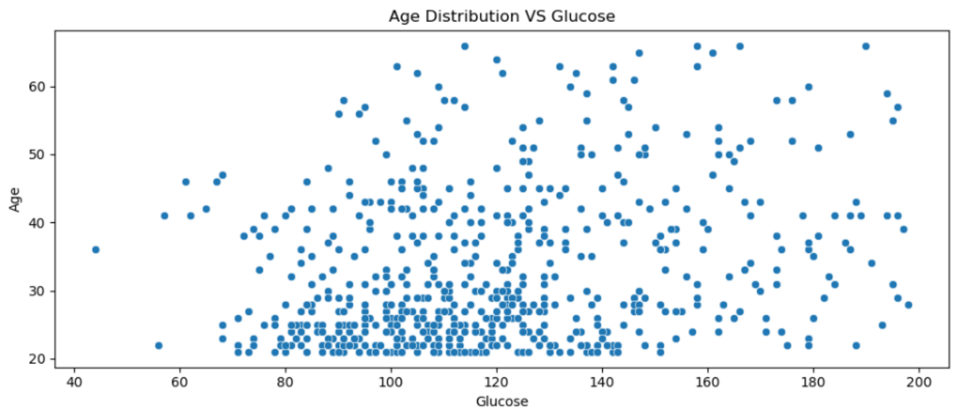
From the 'Age' histograms, we clearly see that the outcome 0 population (healthy) is skewed towards the younger side that means young people are less likely to have diabetes, but for the outcome 1 plot (diabetic), the age is well distributed over a range of 20-60 with highest probability around 30 years.

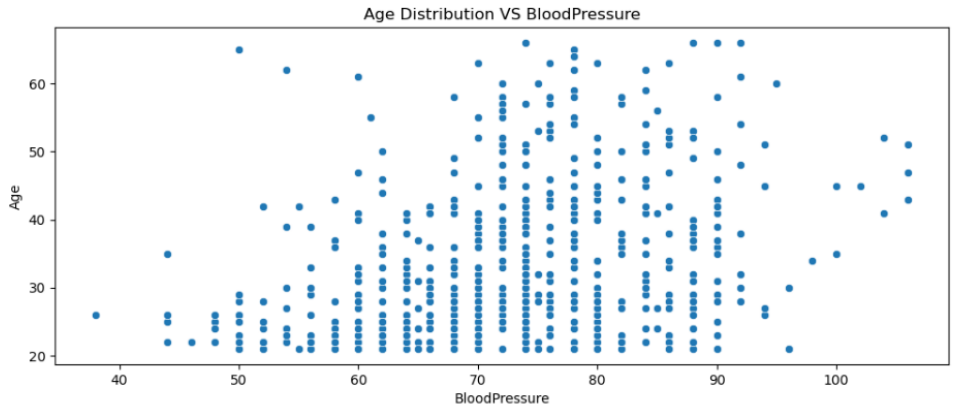
Also the mean of the 'BMI', 'pressure', 'SkinThickness' and 'Insulin' are also little skewed towards the right for the outcome 1 (diabetic) patients. Which imply that these are the important factors to monitor for the early-stage detection.

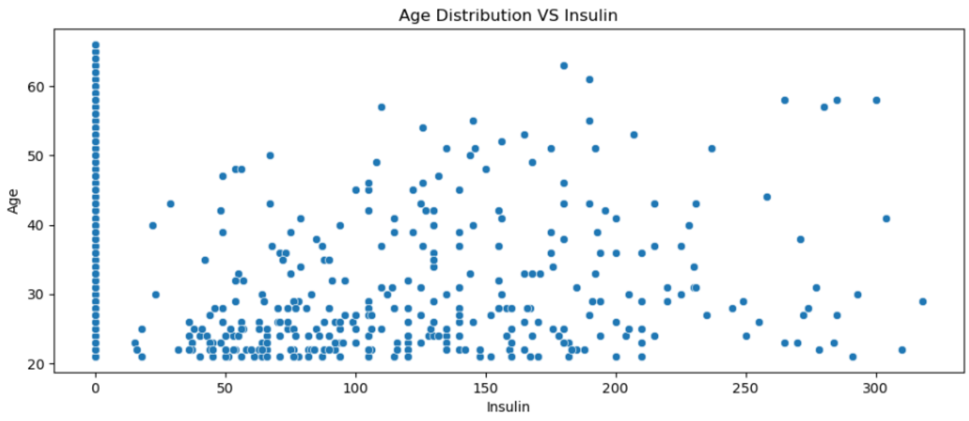
The number of total patients with outcome 1 is very less to do a proper statistical analysis. A dataset with more number of patients would be useful to discover clearer trends.

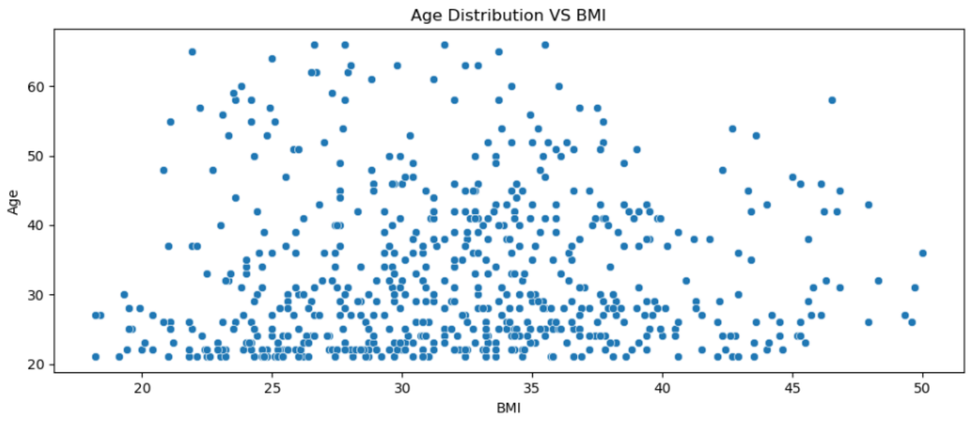
Scatter Plots:

From the scatter plots of Age distribution of Glucose, Blood pressure, Insulin and BMI we notice that the values tend to go higher with age, although there is a significant amount of spread for lower ages also.


Age Distribution VS Glucose
Age
Glucose
20
30
40
50
60
40
60
80
100
120
140
160
180
200


Age Distribution VS BloodPressure
Age
BloodPressure
20
30
40
50
60
40
50
60
70
80
90
100


Age Distribution VS Insulin
Age
Insulin
20
30
40
50
60
0
50
100
150
200
250
300

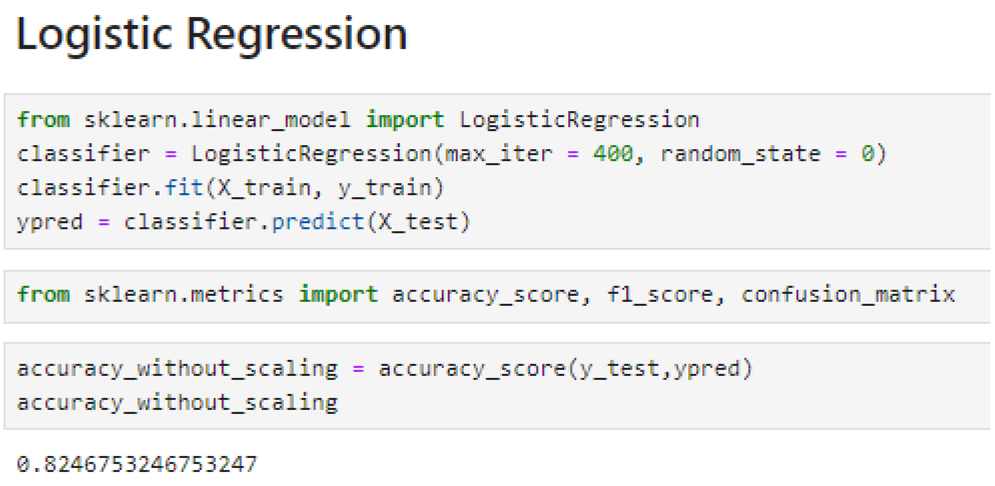

Age Distribution VS BMI
Age
BMI
20
30
40
50
60
20
25
30
35
40
45
50

Modelling the data and prediction:

I have used several machine learning models from the open source scikit-learn library package e.g. logistic regression, KNN, random forest model & decision tree. I have split the data into 80%-20% for training and testing.

1. Logistic Regression

Logistic regression was chosen as the first model due to its simplicity for predicting categorical data.



Logistic Regression

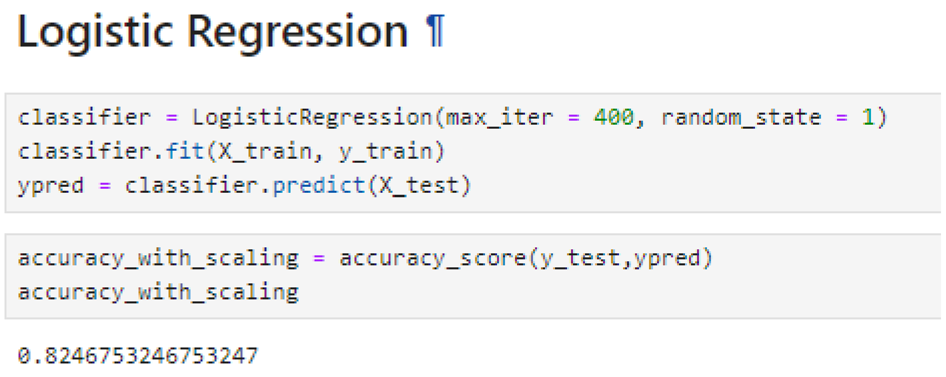
```
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(max_iter = 400, random_state = 0)
classifier.fit(X_train, y_train)
ypred = classifier.predict(X_test)
```

```
from sklearn.metrics import accuracy_score, f1_score, confusion_matrix
```

```
accuracy_without_scaling = accuracy_score(y_test,ypred)
accuracy_without_scaling
```

```
0.8246753246753247
```

First the model was implemented without feature scaling and surprisingly it yielded 82% accuracy. After feature scaling also the accuracy remained unaffected.



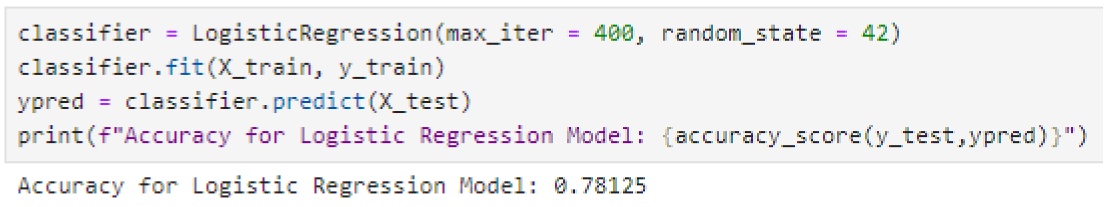
Logistic Regression ¶

```
classifier = LogisticRegression(max_iter = 400, random_state = 1)
classifier.fit(X_train, y_train)
ypred = classifier.predict(X_test)
```

```
accuracy_with_scaling = accuracy_score(y_test,ypred)
accuracy_with_scaling
```

```
0.8246753246753247
```

However, after both outlier treatment and feature scaling, the accuracy came down to 78%. This might have happened due to less number of training data in the dataset.

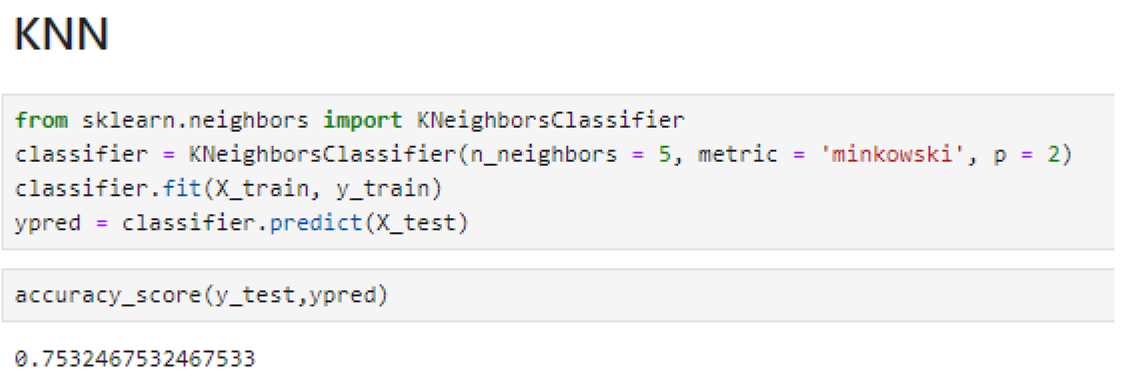


```
classifier = LogisticRegression(max_iter = 400, random_state = 42)
classifier.fit(X_train, y_train)
ypred = classifier.predict(X_test)
print(f"Accuracy for Logistic Regression Model: {accuracy_score(y_test,ypred)}")
```

```
Accuracy for Logistic Regression Model: 0.78125
```

2. KNN Classification

K nearest neighbour or KNN is another classification model which is very simple and popular to use.



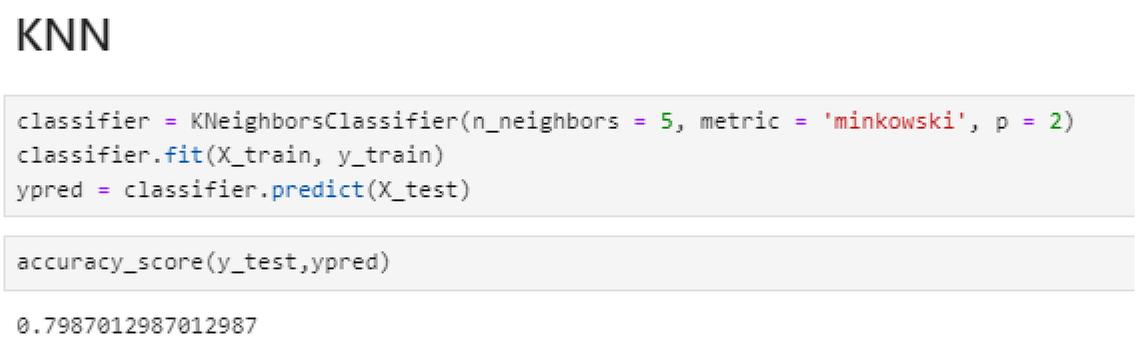
KNN

```
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 2)
classifier.fit(X_train, y_train)
ypred = classifier.predict(X_test)
```

```
accuracy_score(y_test,ypred)
```

```
0.7532467532467533
```

KNN model without feature scaling incurred 75% accuracy. However after feature scaling the accuracy improved to 79.9%.



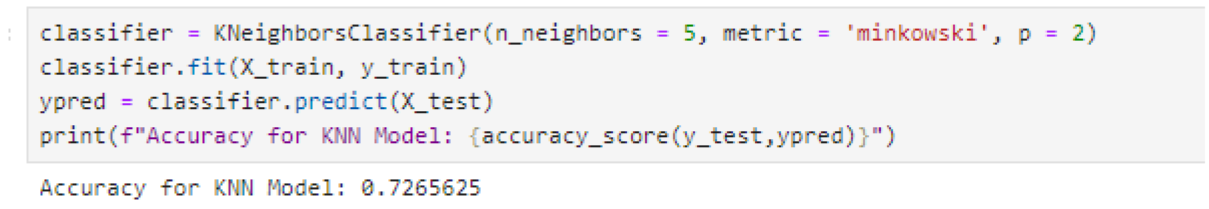
KNN

```
classifier = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 2)
classifier.fit(X_train, y_train)
ypred = classifier.predict(X_test)
```

```
accuracy_score(y_test,ypred)
```

```
0.7987012987012987
```

After both feature scaling and outlier treatment the accuracy decreased to 72.65%.

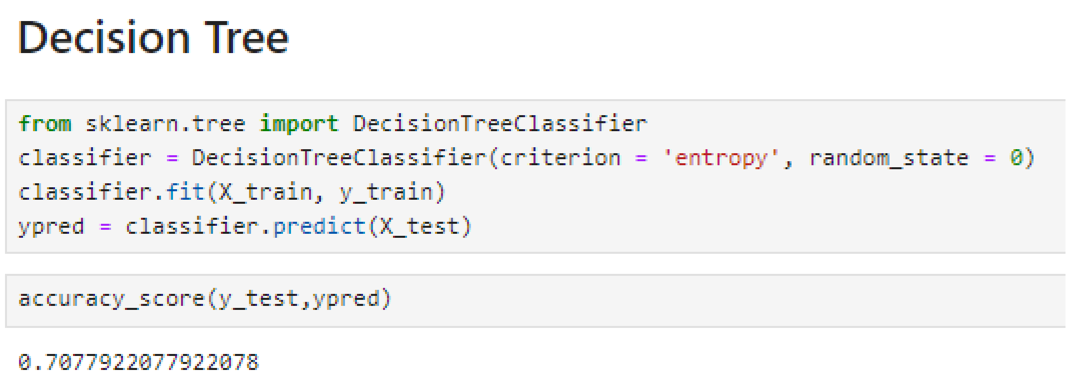


```
classifier = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 2)
classifier.fit(X_train, y_train)
ypred = classifier.predict(X_test)
print(f"Accuracy for KNN Model: {accuracy_score(y_test,ypred)}")
```

```
Accuracy for KNN Model: 0.7265625
```

3. Decision Tree Classifier:

A decision tree classifier is a predictive modeling algorithm that recursively splits the dataset into subsets based on the values of input features, constructing a tree-like structure where each internal node represents a decision based on a feature, leading to leaf nodes representing the predicted outcome.



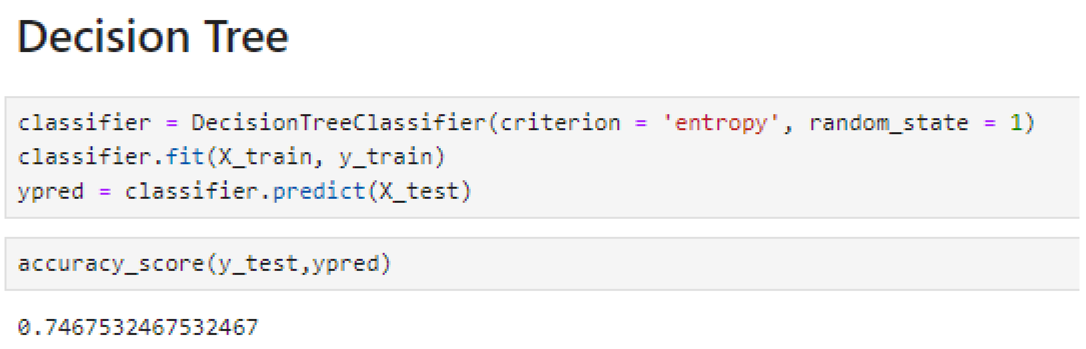
Decision Tree

```
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)
classifier.fit(X_train, y_train)
ypred = classifier.predict(X_test)
```

```
accuracy_score(y_test,ypred)
```

```
0.7077922077922078
```

Without any feature scaling the decision tree classifier yielded an accuracy of ~71%, and after feature scaling the accuracy improved to ~75%.

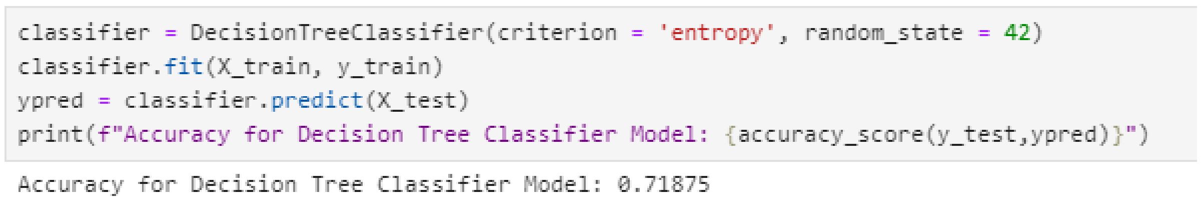


Decision Tree

```
classifier = DecisionTreeClassifier(criterion = 'entropy', random_state = 1)
classifier.fit(X_train, y_train)
ypred = classifier.predict(X_test)
```

```
accuracy_score(y_test,ypred)
```

```
0.7467532467532467
```



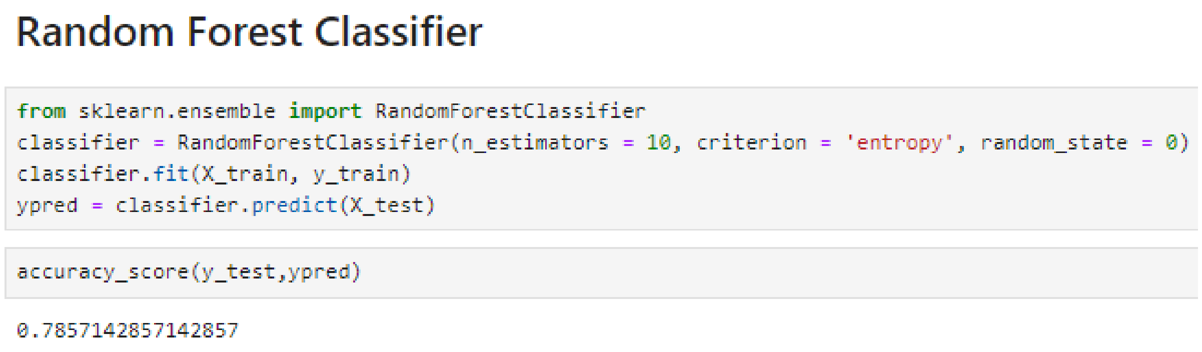
```
classifier = DecisionTreeClassifier(criterion = 'entropy', random_state = 42)
classifier.fit(X_train, y_train)
ypred = classifier.predict(X_test)
print(f"Accuracy for Decision Tree Classifier Model: {accuracy_score(y_test,ypred)}")
```

```
Accuracy for Decision Tree Classifier Model: 0.71875
```

However, after outlier treatment the accuracy again fell back to ~72%.

4. Random Forest Classification:

Random Forest is an ensemble learning method that constructs multiple decision trees during training and outputs the mode of the classes of the individual trees. It is popular because of its improved accuracy and robustness by reducing overfitting and variance.



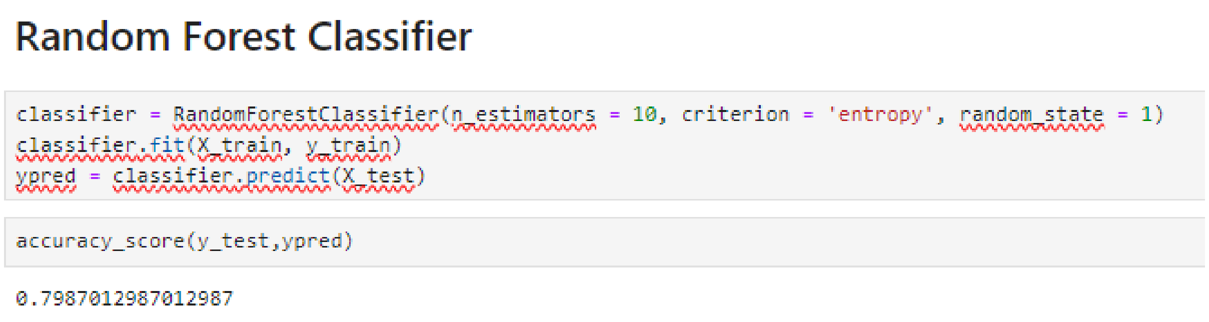
Random Forest Classifier

```
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(n_estimators = 10, criterion = 'entropy', random_state = 0)
classifier.fit(X_train, y_train)
ypred = classifier.predict(X_test)
```

```
accuracy_score(y_test,ypred)
```

```
0.7857142857142857
```

The accuracy without feature scaling for random forest classification is 78.6%, after feature scaling it has slightly improved to 79.9%.



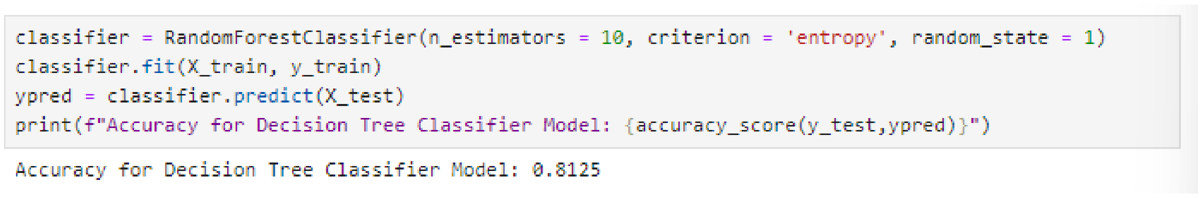
Random Forest Classifier

```
classifier = RandomForestClassifier(n_estimators = 10, criterion = 'entropy', random_state = 1)
classifier.fit(X_train, y_train)
ypred = classifier.predict(X_test)
```

```
accuracy_score(y_test,ypred)
```

```
0.7987012987012987
```

After both feature scaling and outlier treatment the accuracy has further increased to 81%.



```
classifier = RandomForestClassifier(n_estimators = 10, criterion = 'entropy', random_state = 1)
classifier.fit(X_train, y_train)
ypred = classifier.predict(X_test)
print(f"Accuracy for Decision Tree Classifier Model: {accuracy_score(y_test,ypred)}")
```

```
Accuracy for Decision Tree Classifier Model: 0.8125
```

Conclusion:

I have applied logistic regression, KNN, decision tree classifier and random forest classification for model building. Also I have done outlier treatment of the dataset to eliminate outliers. However the treatment has procured mixed results and not always have improved the accuracy. In case of logistic regression, KNN and decision tree, the accuracies have decreased after outlier treatment, but for Random forest model it has improved. This can be due to reduced number of training data which can in turn result in poor accuracy.

Also after feature scaling the accuracy has improved for three classes: KNN, Decision tree and Random forest but accuracy has decreased for logistic regression. In general, it is expected that feature scaling enhances the accuracy by standardizing the data.