DIABETES PREDICTION MODEL

Predictive Analytics Project

Submitted to:
Prof. Enayat Rajabi

Submitted by:
Chandrayog Yadav(20196914)
Umesh Garg(20193834)
Sakshi Jakhar(20195895)

Table of Contents

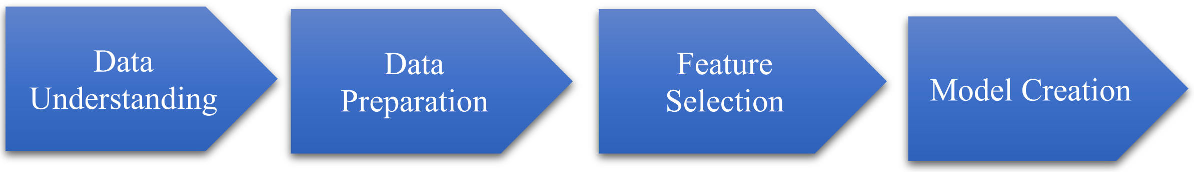
Objective:
Create a predictive model to predict the onset of the Diabetes in female patients.

Problem Type:
This is a classification problem to detect the diabetes in female based on certain predictor variables.

Dataset:
The dataset contains the data of patients which are females at least 21 years old of Pima Indian heritage. This dataset is originally from the National Institute of Diabetes and Digestive and Kidney Diseases, US.

Dataset has been taken from Kaggle:
https://www.kaggle.com/uciml/pima-indians-diabetes-database

Model Development Stages:



Exploratory Data Analysis:
Exploratory data analysis is an approach to analyzing data sets to summarize their main characteristics mostly using with visual methods like charts, plots etc.

1. Data Understanding:

To better understand our dataset and its data, we are using data understanding methods which is the initial or first step of EDA.

The datasets consist of several medical predictor variables like Glucose, Insulin, BMI, Pregnancies, Age etc. and one target variable named **Outcome**.

Steps:

Loading Libraries:

```
### Importing Python libraries
import pandas as pd
import numpy as np

### Plots
import seaborn as sns
import matplotlib.pyplot as plt
import plotly.offline as py
```

1. Details about records of the dataset:

print ("This dataset has {} samples with {} features each.".format(df.shape[0], df.shape[1]))

This dataset has 768 samples with 9 features each.

2. Statistical Analysis:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768	768	768	768	768	768	768	768	768
mean	3.845052083	120.8945313	69.10546875	20.53645833	79.79947917	31.99257812	0.471876302	33.24088542	0.348958333
std	3.369578063	31.9726182	19.35580717	15.95221757	115.2440024	7.88416032	0.331328595	11.76023154	0.476951377
min	0	0	0	0	0	0	0.078	21	0
25%	1	99	62	0	0	27.3	0.24375	24	0
50%	3	117	72	23	30.5	32	0.3725	29	0
75%	6	140.25	80	32	127.25	36.6	0.62625	41	1
max	17	199	122	99	846	67.1	2.42	81	1

Findings:

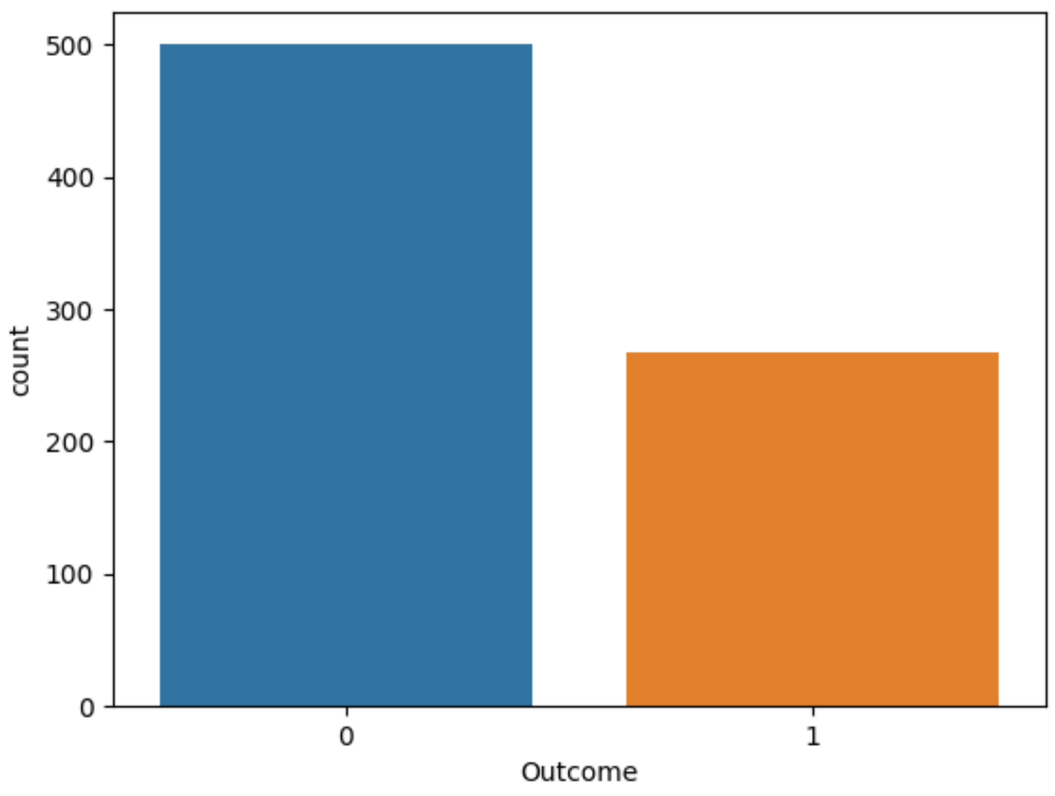
After observing the values, we identified that few features have 0 min value which is erratic, Insulin has max value quite high as compare to their std deviation and IQR values.

3. Details about Columns and their datatypes:

```
<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
None
```

4. Overview of Dataset:

```
Total number of individuals: 768
Individuals with diabetes: 268
Individuals without diabetes: 500
Percentage of individuals with diabetes: 34.90%
```



5. Distribution of Data:

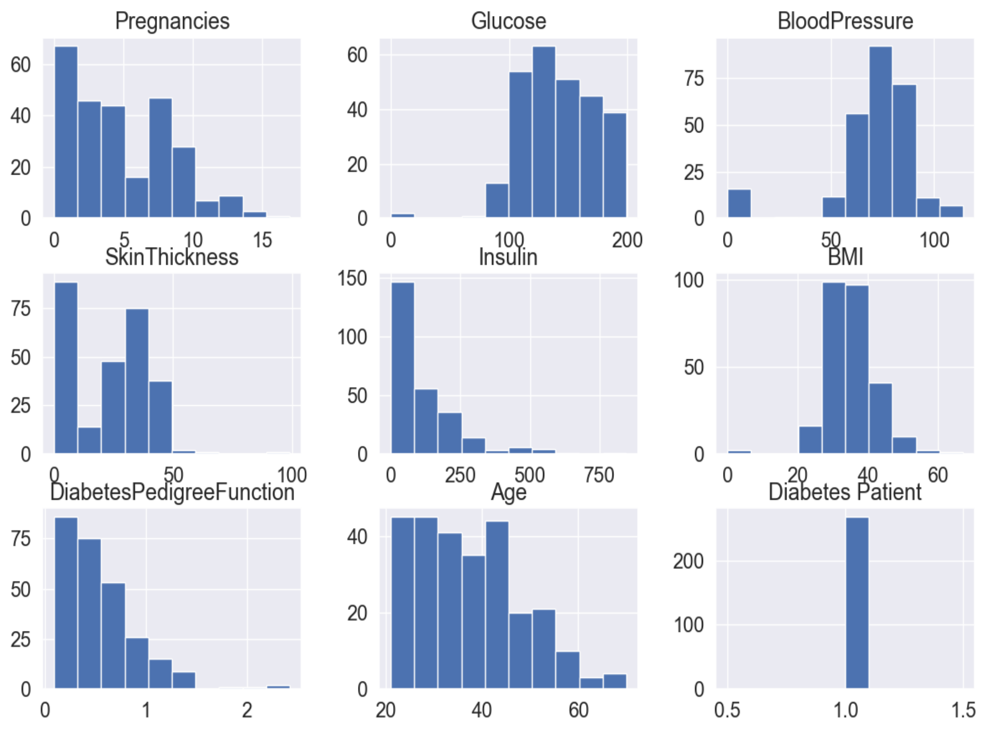
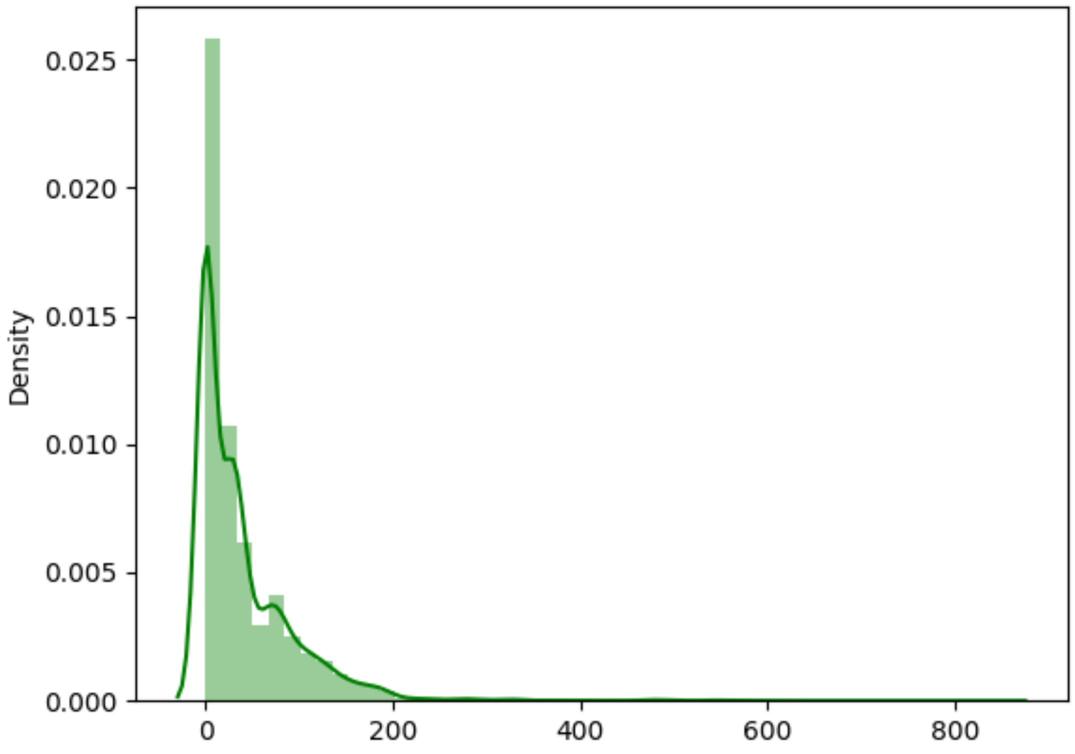


Figure 1

Overall Dataset Distribution:



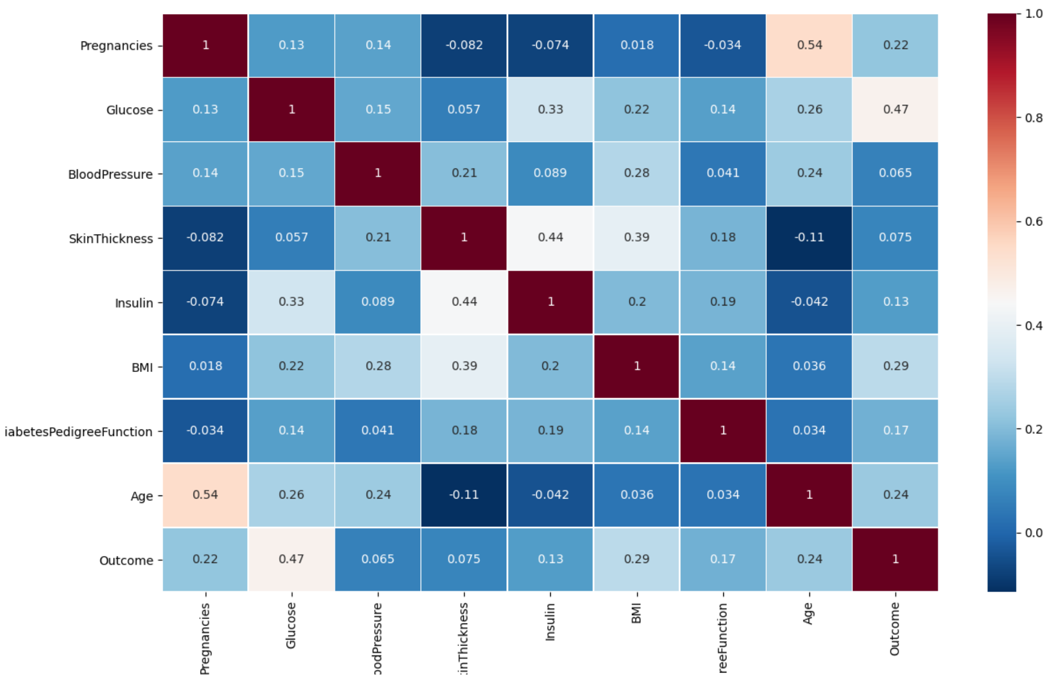
Data distribution is Positively Skewed which means skewness value is greater than 0.
In these situations, mean is greater than the median.
Also in this case we considered median to be the best representative of the central location of the data as it represents the half values are above and half below of the distribution.

Positively Skewed- Mean > Med > Mode
Normal Curve- Mean=Med=Mode

7. Correlation among Variables and with Target Variable:

By using correlation matrix in the python we can determine the strong and weak relationships of variables.

Correlation Matrix with values:



Findings:

After analyzing the relationship matrix, we can depict that **Glucose** has strong relationship with Target Variable Outcome, **BMI** has mildly strong relationship and **Age, Pregnancies** has moderate relationship with Target variable.

Important Features:

Glucose
BMI
Age
Pregnancies

Outliers:

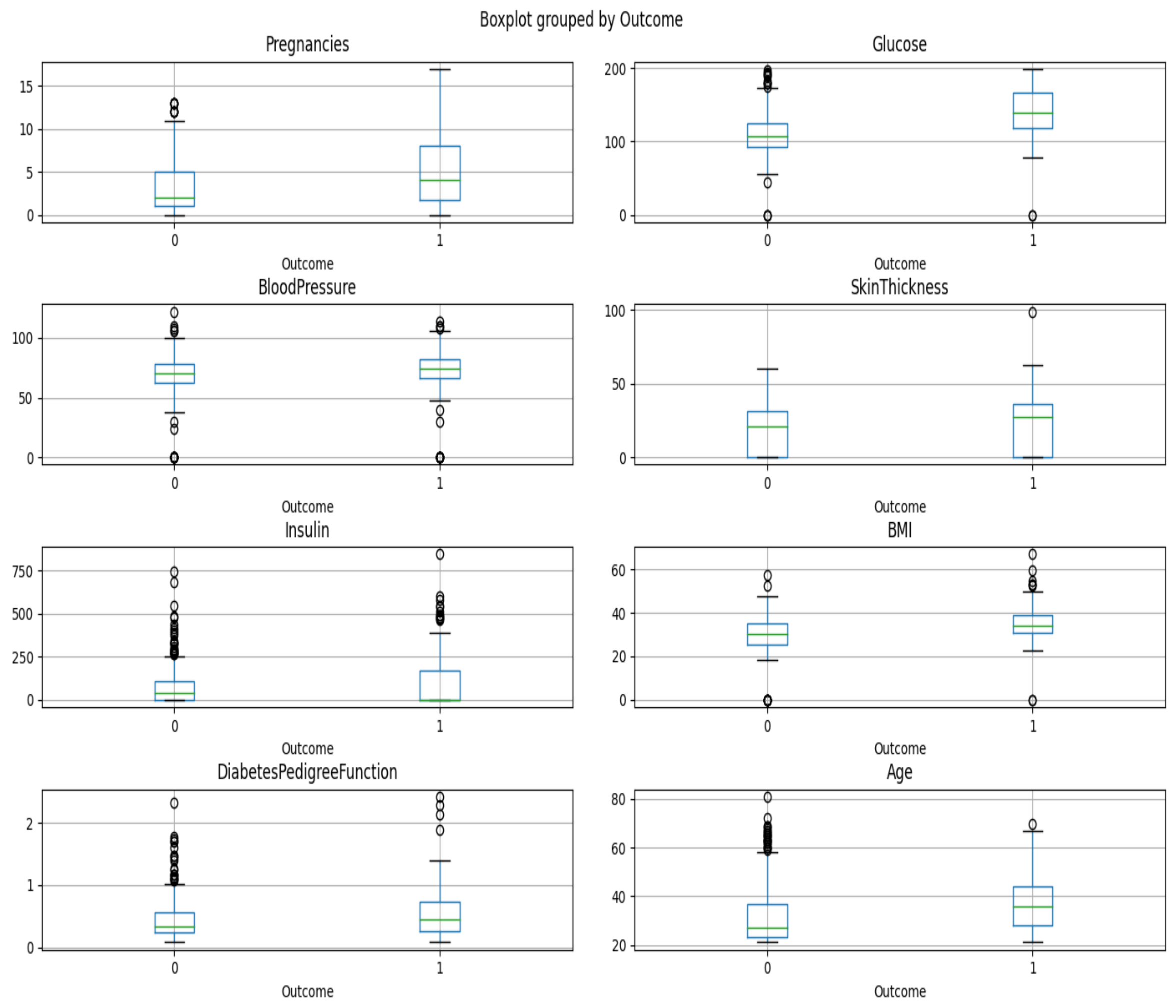
We have checked the data distribution in previous steps. Now we will explore further into the dispersion to identify and check the outliers.

Using Tukey IQR method-
Tukey's rule says that the outliers are values more than 1.5 times the interquartile range from the quartiles — either below Q1 − 1.5IQR, or above Q3 + 1.5IQR.

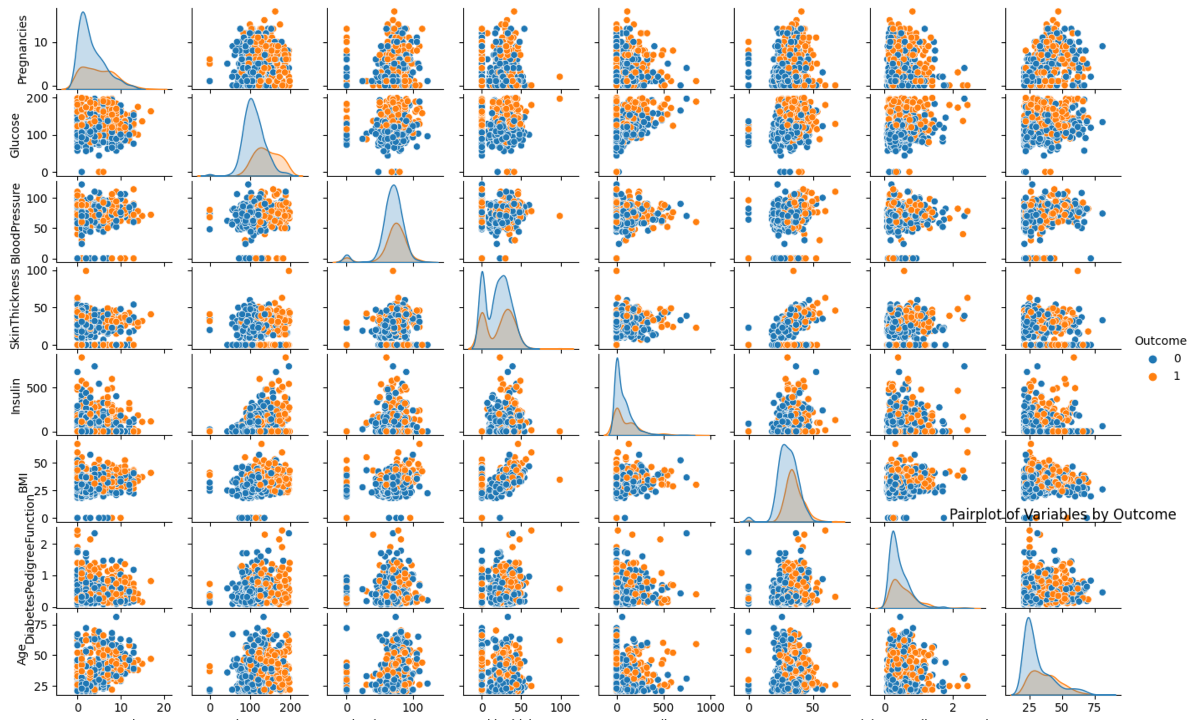
```
#### Method 2: Finding Outliers using Tukey IQR method

def find_outliers(x):
    q1 = np.percentile(x, 25)
    q3 = np.percentile(x, 75)
    iqr = q3 - q1
    lower = q1 - 1.5*iqr
    ceiling = q3 + 1.5*iqr
    ind = list(x.index[(x < lower) | (x > ceiling)])
    outlier = list(x[ind])
    return ind, outlier


out_inx, out_val = find_outliers(df['Insulin'])
print(np.sort(out_val))
```

We will use Boxplot chart to visualize required columns. Basically, a boxplot is a graph that gives you a good indication of how the values in the data are spread out.

Outlier description of all features w.r.t Target variable:



Pairplots of variables with respect to Target variable:



Inference from Pair Plot and Graphs

- The plot on the diagonal allows us to see the distribution of a single variable while the scatter plots on the upper and lower triangles show the relationship (or lack thereof) between two variables.
- From scatter plots, only BMI & SkinThickness and Pregnancies & Age seem to have positive linear relationships. Another likely suspect is Glucose and Insulin.
- When looking at the segmented histograms, a hypothesis is the as pregnancies includes, women are more likely to be diabetic
- Histogram of Glucose data(Figure1 above) slightly skewed to right. Understandably, the data set contains over 60% who are diabetic and its likely that their Glucose levels were higher. But the grand mean of Glucose is at 122.
- Clearly diabetic group has higher glucose than non-diabetic.
- After looking on Age, there is a tendency that as people age, they are likely to become diabetic. This needs statistical validation
- In BMI, there are few outliers. Few are obese in the dataset. Expected range is between 18 to 25. In general, people are obese
- Diabetic people seem to be only higher side of BMI
- For Insulin, clearly there are Outliers in the data. These Outliers are concern for us and most of them with higher insulin values are also diabetic. So, this is a suspect.
- BloodPressure, few outliers in the data. Its likely that some people have low and some have high BP. So, the association between diabetic (Outcome) and BP is a suspect and needs to be statistically validated

2. Data Preparation:

It is a pre-processing step in which data from one or more sources is cleaned and transformed to improve its quality prior to its use in business analytics.

This includes data cleaning, variable selection, and feature creation.

Data Cleaning, Missing Value or Irrelevant values:

First, let's check inspect the values of columns.

Using below command:

```
print(df.describe())
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768	768	768	768	768	768	768	768	768
mean	3.845052083	120.894531 3	69.10546875	20.53645833	79.79947917	31.99257812	0.471876302	33.24088542	0.3489583 33
std	3.369578063	31.9726182	19.35580717	15.95221757	115.24400 24	7.88416032	0.331328595	11.76023154	0.4769513 77
min	0	0	0	0	0	0	0.078	21	0
25%	1	99	62	0	0	27.3	0.24375	24	0
50%	3	117	72	23	30.5	32	0.3725	29	0
75%	6	140.25	80	32	127.25	36.6	0.62625	41	1
max	17	199	122	99	846	67.1	2.42	81	1

Here minimum values of glucose, bloodpressure, skinthickness, insulin, BMI are 0,which appears unusual.

Now lets check the all null values in all columns

```
#Checking Null Values
print(df.isnull())

# Or
df.isna()
```

Sum of all null columns

```
df.isnull().sum()
```

We can see that there no null values but some columns with few 0 values and some columns with many 0 values. Now we will replace the 0 values with NaN, not a number feature values in the required columns

```
# Replace 0 with NaN
df.replace(0, np.NaN, inplace=True)
```

Now check count of Nan Values.

```
Pregnancies              111
Glucose                    5
BloodPressure             35
SkinThickness            227
Insulin                  374
BMI                       11
DiabetesPedigreeFunction    0
Age                        0
```

We have replaced the NaN values in identified columns with their mean or median based on their data distribution or skewness.

Glucose -> Mean
BloodPressure -> Mean
SkinThickness -> Median
Insulin -> Median
BMI -> Median
(We didn't replace pregnancies as it can be 0)

```
# fill missing values with for the columns in accordance with their distribution
df['Glucose'].fillna(df['Glucose'].mean(), inplace = True)
df['BloodPressure'].fillna(df['BloodPressure'].mean(), inplace = True)
df['SkinThickness'].fillna(df['SkinThickness'].median(), inplace = True)
df['Insulin'].fillna(df['Insulin'].median(), inplace = True)
df['BMI'].fillna(df['BMI'].median(), inplace = True)
```

Rename complex/large column names to shorter names:

After observing all the columns, we found that one of the columns is having quite lengthy name which causing view issues in plots.

Therefore, renaming it to a shorter abbreviation.

```
### Renaming the column

df.rename(columns={'DiabetesPedigreeFunction':'DPF'}, inplace=True)
df.info()
```

3. Feature Selection:

There are various feature extraction techniques in data science which are categorized into 3 parts: Filter methods, Wrapper methods and Embedded methods.

We have implemented few of the above techniques to verify importance of features and selecting them based on the accuracy of various algorithms.

1. Filter Method- Variance Threshold

We can perform a simple check to see the variance of each feature by using below code:

```
### Check the Variance ###
df.var()
```

Low variance resembles less importance of feature. In the below output we can see Diabetes function and pregnancies are seeming to be lower variances.

Pregnancies	10.350962
Glucose	926.346983
BloodPressure	146.321591
SkinThickness	77.285567
Insulin	7462.033
BMI	47.268056
DPF	0.109779
Age	138.303046
Outcome	0.227483

Findings:

Above just gives us an overview of features importance with higher variance value. Insulin and Glucose are the most significant features here but we still need to be sure about the relevance and importance of features and will perform some more tests.

Important Features:
Insulin
Glucose
BloodPressure
Age

2. Wrapper Method- Recursive Feature Elimination Method

Recursive feature elimination (RFE) is a feature selection method that fits a model and removes the weakest feature (or features) until the specified number of features is reached.

RFE attempts to eliminate dependencies and collinearity that may exist in the model. This is achieved by fitting the given machine learning algorithm used in the core of the model, ranking features by importance, discarding the least important features, and re-fitting the model. This process is repeated until a specified number of features remains.

Steps:
Create base model first.
We have used logistic regression

```
### use a baseline classifier logistic regression
log_reg = LogisticRegression()
lr_model = log_reg.fit(x_train,y_train)
```

Implement RFE on base model

```
rfe = RFE(estimator=lr_model, step=1)
### fit the rfe function for ranking
rfe = rfe.fit(x_train, y_train)
```

Output:

No of feautres : 4
Selected feautres : [True True False False False True True False]
Feautres rank   : [1 1 2 3 4 1 1 5]

Feature	Ranking
Pregnancies	1
Glucose	1
BMI	1
DPF	1
BloodPressure	2
Age	3
Insulin	4
SkinThickness	5

Important Features:
Pregnancies
Glucose
BMI
DPF

3. Embedded Method- Decision Tree Regressor (CART)

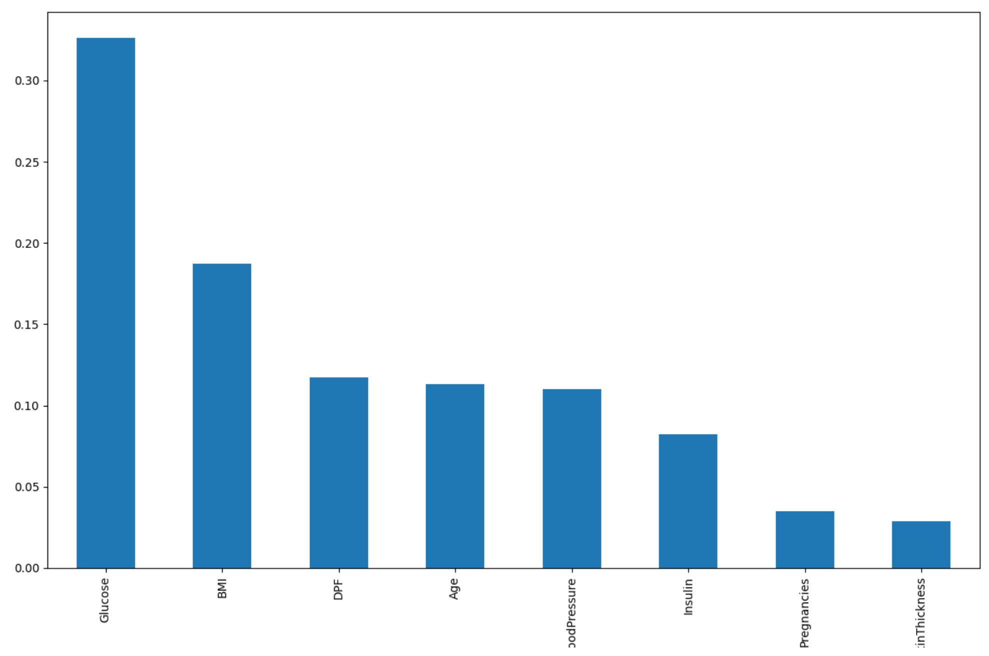
Create model first then fit the data with all features.

```
cart_model = DecisionTreeRegressor()
cart_tree = cart_model.fit(X, Y)
imp_feature = cart_tree.feature_importances_
```

Now Check the important features with higher score value

```
#check summary
feature_val = pd.Series(imp_feature, index = X.columns)

# draw chart of important feature
feature_val.nlargest(10).plot(kind='bar')
plt.show()
```

Features	feature_val
Pregnancies	0.053534
Glucose	0.327399
BloodPressure	0.102722
SkinThickness	0.032313
Insulin	0.070061
BMI	0.197875
DPF	0.110259
Age	0.105837



Important Features:
Glucose
BMI
DPF
AGE

Selected Features:

After analyzing all three methods which are filter method - Correlation and Variance, Wrapper method- RFE and embedded method- CART, we concluded that below are the important features based on their occurrences and comparison in each method:

Method Name	Feature 1 >	Feature 2>	Feature 3>	Feature 4
Pearson Correlation	Glucose	Insulin	BMI	Age
Variance	Insulin	Glucose	BloodPressure	Age
RFE	Pregnancies	Glucose	BMI	DPF
Decision Tree	Glucose	BMI	DPF	Age

🟩	Most Freq
🟨	Medium Freq
🟧	Least Freq

Selected Features	**Occurrences in all methods**
Glucose	**4**
BMI	**3**
Age	**3**
DPF	**2**
Insulin	**2**

Therefore, we will select only above identified features in our model implementation for prediction.

4. Model Creation and Validation

We have used Logistic Regression model as initial model.

- A logistic regression is used from the dependent variable is binary, ordinal or nominal and the independent variables are either continuous or discrete

Method 1- Logistic regression model from sklearn

Importing LogisticRegression from Sk.Learn linear model as stats model logit function cannot give us classification report and confusion matrix.

a. Create Test and Training parts to apply model.

```
x_train, x_test, y_train, y_test = train_test_split(X, Y, random_state=8)


### use a baseline classifier , here for example logistic regression
log_reg = LogisticRegression()
lr_model = log_reg.fit(x_train,y_train)
```

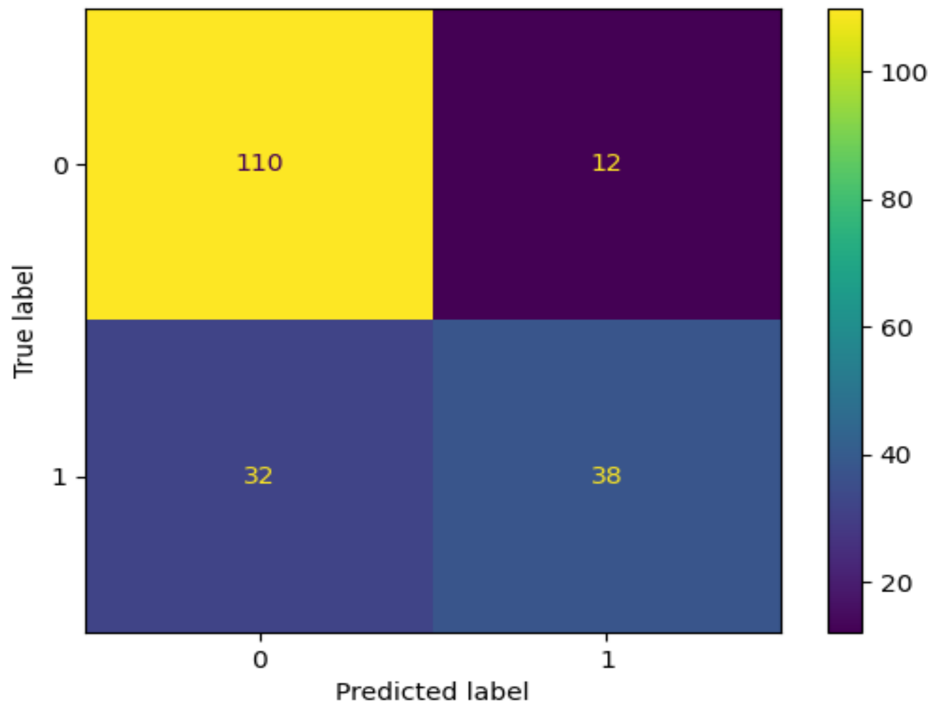
b. Then Check Accuracy and F1 Score with all the features initially:

```
### scores
ac = accuracy_score(y_test, y_pred)
f_score = f1_score(y_test, y_pred)

print(ac)
print(f_score)


### Check confusion matrix
plot_confusion_matrix(lr_model, x_test, y_test)  # doctest: +SKIP
plt.show()
```

Accuracy is used when the True Positives and True negatives are more important while F1-score is used when the False Negatives and False Positives are crucial

Baseline model accuracy score with all feature: 0.7447916666666666
Baseline model F1 score with all feature: 0.6016260162601625

Confusion Matrix:



2nd Iteration with selected feature:

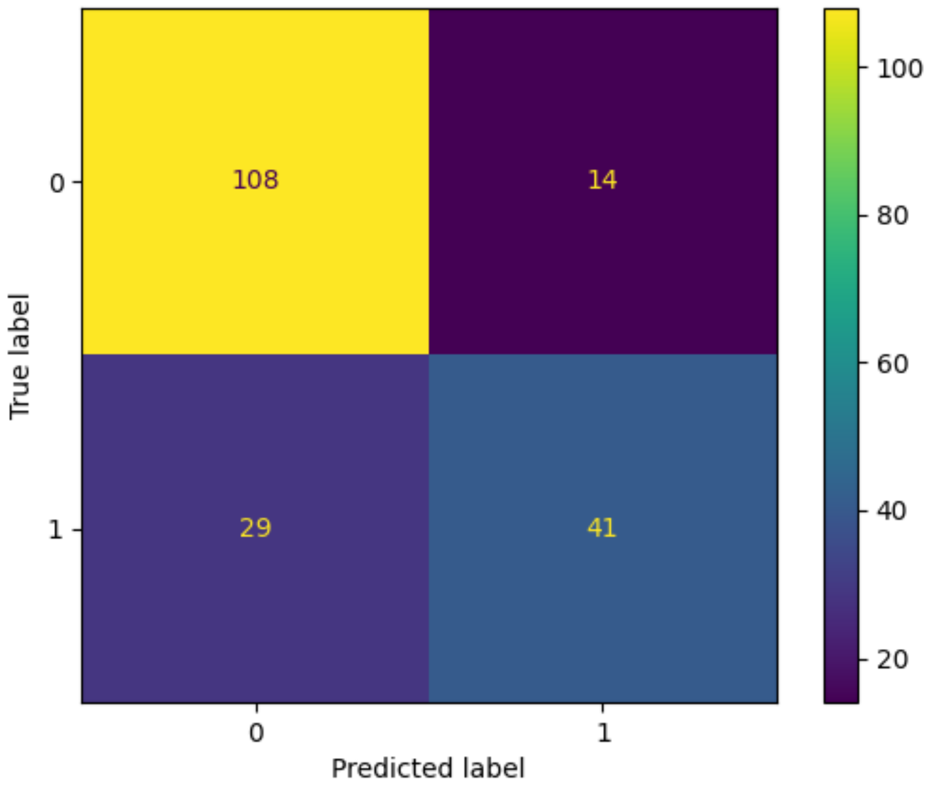
Dropping the columns/features which are not required.

```
##### Dropping not required columns  #####
df.drop(["Pregnancies", "BloodPressure", "SkinThickness"], 1 , inplace = True)
```

Now again running the model we will get improved accuracy:

Baseline model accuracy score with selected features: 0.7708333333333334
Baseline model F1 score with selected features: 0.639344262295082

Confusion Matrix:



Method 2- Decision Tree Model

We will use Random Forest classifier to re run the model in order to compare the performance of the model with the selected features.

Call Random Forest Classifier

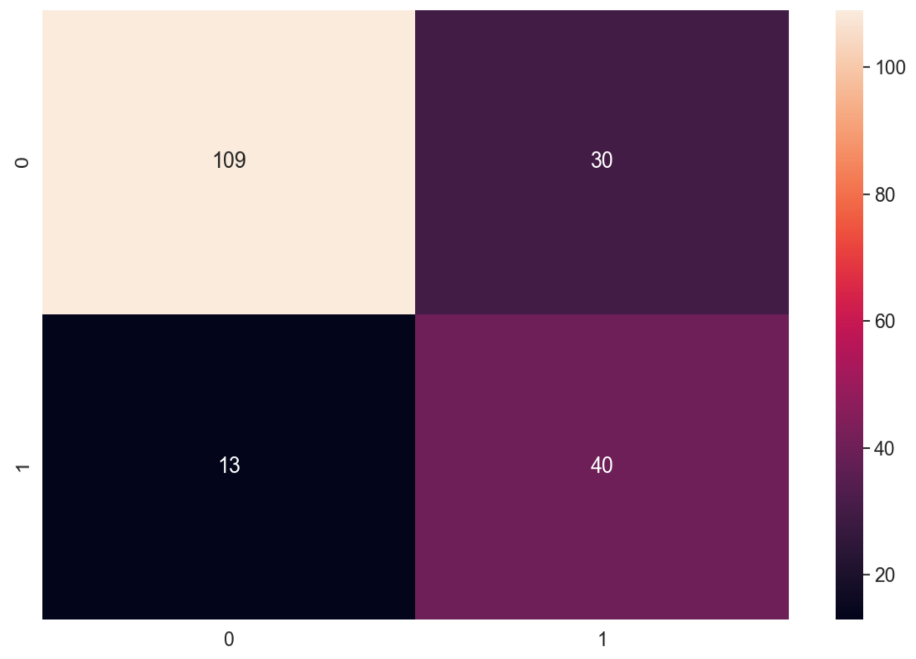
```
rfc = RandomForestClassifier(n_estimators=1000, random_state=1, n_jobs=-1)
```

Train the classier and print the accuracy of the model

```
### Train the classifier
rfc.fit(x_train, y_train)
y_pred_rfc = rfc.predict(x_test)

rfc_acc= accuracy_score(y_test, y_pred_rfc)
rfc_f_score= f1_score(y_test, y_pred_rfc)

print('Random Forest Accuracy:', rfc_acc)
print('Random Forest F score:', rfc_f_score)
```

```
Random Forest Accuracy with selected features: 0.7708333333333334
Random Forest F1 score with selected features: 0.65625
```

Confusion Matrix



Method 3 – Using KNN Classification

The KNN algorithm assumes that similar things exist in close proximity. In other words, similar things are near to each other.

Calling KNN instance

```
### Method 4 Using KNN Classification
knn = KNeighborsClassifier()
```

Transform the data

```
### Transform the data
scaler = preprocessing.StandardScaler().fit(x_train)
x_train = scaler.transform(x_train)
x_test = scaler.transform(x_test)
```

Fit the model

```
### Fit the model
knn = neighbors.KNeighborsClassifier(n_neighbors=15)
knn.fit(x_train, y_train)
y_pred = knn.predict(x_test)
```

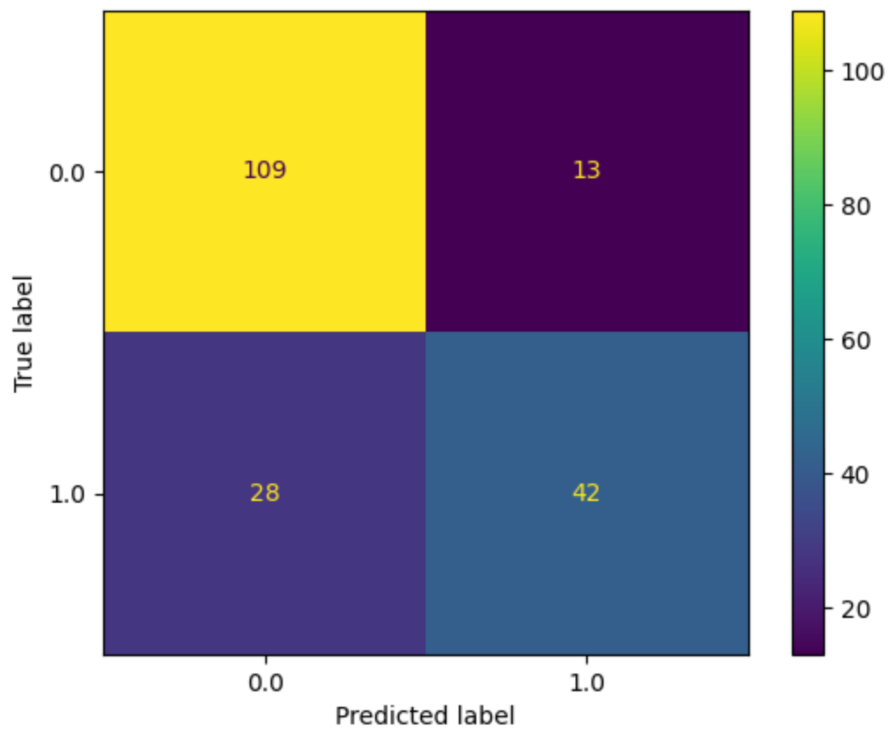
Print the scores and accuracy

```
print('KNN Accuracy with selected features:' , accuracy_score(y_test, y_pred))
print('KNN F1 Score with selected features:' , f1_score(y_test, y_pred))
```

```
### Check confusion matrix
plot_confusion_matrix(knn, x_test, y_test)
plt.show()
```

```
KNN Accuracy with selected features: 0.7864583333333334
KNN F1 Score with selected features: 0.6719999999999999
```

Confusion Matrix



Findings:

- Accuracy of the model is 78.6% with 109+42 correct values and 13+28 incorrect predicted values.
- F1 score is 67% which is the lowest among all other models.

Conclusion:

Models	Accuracy	F1 Score
Logistic Regression	77	64
Decision Tree	77	66
KNN	78	67

- After comparisons we found that KNN classifier performed the best with k=15
- Accuracy and F1 score both are high for KNN as compare to logistic and DT
- The result is telling us that we have 108+41 are correct predictions and 14+29 are incorrect predictions with Logistic Regression
- On the other hand, decision tree classifier predicted 109+40 correct and 30+13 incorrect predictions
- We can select any model based on accuracy
- However, accuracy works best if false positives and false negatives have similar cost. If the cost of false positives and false negatives are very different, it’s better to look at both Precision and Recall.

References:

- https://www.kaggle.com/uciml/pima-indians-diabetes-database/notebooks?sortBy=relevance&group=everyone&search=python&page=1&pageSize=20&datasetId=228&tagIds=13102

- https://www.kaggle.com/kaggle19/simple-data-exploration-using-python

- https://www.kaggle.com/vincentlugat/pima-indians-diabetes-eda-prediction-0-906

- https://www.kaggle.com/meeravinod/tutorial-removing-missing-values