Develop ML Model to Predict Diabetes

Name: Sujata Mallik

Batch: PG DS June Cohort 1

DESCRIPTION

NIDDK (National Institute of Diabetes and Digestive and Kidney Diseases) research creates knowledge about and treatments for the most chronic, costly, and consequential diseases. The dataset contains details of the female patients from Pima Indian heritage who are at least 21 years old.

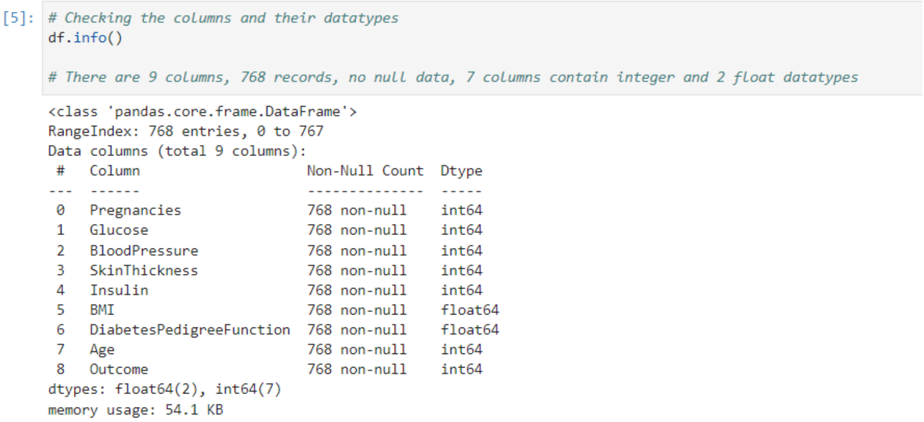
OBJECTIVE

Building a model to predict whether or not a patient has diabetes, based on certain diagnostic measurements included in the dataset.

DATA PREPROCESSING

The dataset contains total of 768 observations and 9 variables. There are 8 medical predictor variables and one target variable (**Outcome**).

Variables	**Description**
Pregnancies	Number of times of pregnancy
Glucose	Plasma glucose concentration in an oral glucose tolerance test
BloodPressure	Diastolic blood pressure (mm Hg)
SkinThickness	Triceps skinfold thickness (mm)
Insulin	Two hour serum insulin (mu U/ml)
BMI	Body Mass Index (kg/m squared)
DiabetesPedigreeFunction	Determines type2 diabetes risk in the family. Larger values indicates higher risk for type2 diabetes
Age	Age in years
Outcome	Whether the patient is diagnosed with type2 diabetes. Class variable (0 = No , 1 = Yes)



```
[5]: # Checking the columns and their datatypes
     df.info()

     # There are 9 columns, 768 records, no null data, 7 columns contain integer and 2 float 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
```

There are no null values in the dataset. 7 variables are integer datatypes and 2 variables are float datatype.

Analysis

Pregnancies: Value range: 0 - 17, with a median of 3. Max value is too high for number of pregnancies. Minimum value 0 indicates the patient never got pregnant. Here the data is not missing.

Glucose: Value range: 0 - 199. Glucose level 0 is not realistic. Will treat 0 as missing data. Max value can be possible.

BloodPressure: Value range: 0 - 122. Blood Pressure 0 is not realistic. Will treat 0 as missing data. Maximum value can be possible.

SkinThickness: Value range: 0 to 846. Skin thickness 0 is not realistic. Will treat 0 as missing data.

BMI: Value range: 0 to 67.1. Body mass index cannot be 0. Will treat 0 as missing data.

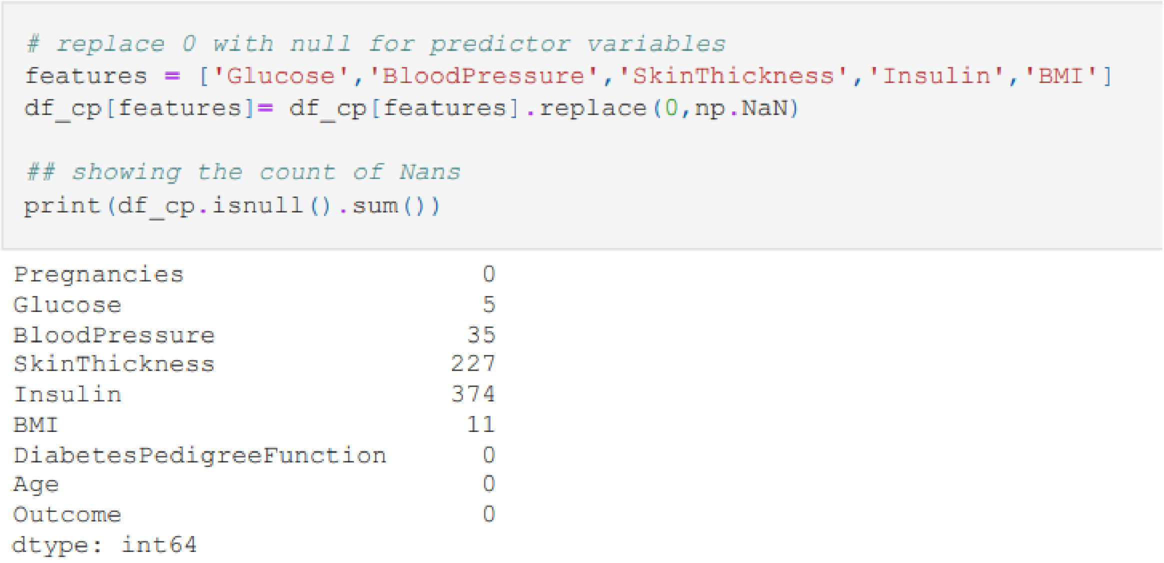
DiabetesPedigreeFunction: Value range: .078 - 2.42. Age: Value range: 21 - 81. Replacing zeros with NaN for easy count of actual values. Will replace missing values later with appropriate values feature and target variables

Insulin: Value range: 0 to 846. Insulin value 0 is not realistic. Will treat 0 as missing data.

Target variable: Outcome

Predictor variables: Pregnancies, Glucose, BloodPressure, SkinThickness, Insulin, BMI, DiabetesPedigreeFunction and Age

Missing values: By replacing 0 values for the predictor variables with Null values it helps to identify the missing data



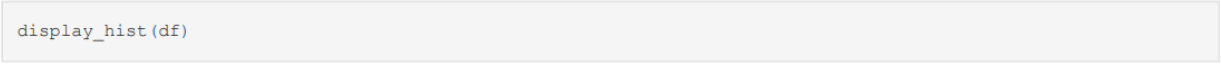
```
# replace 0 with null for predictor variables
features = ['Glucose','BloodPressure','SkinThickness','Insulin','BMI']
df_cp[features]= df_cp[features].replace(0,np.NaN)

## showing the count of Nans
print(df_cp.isnull().sum())
```

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

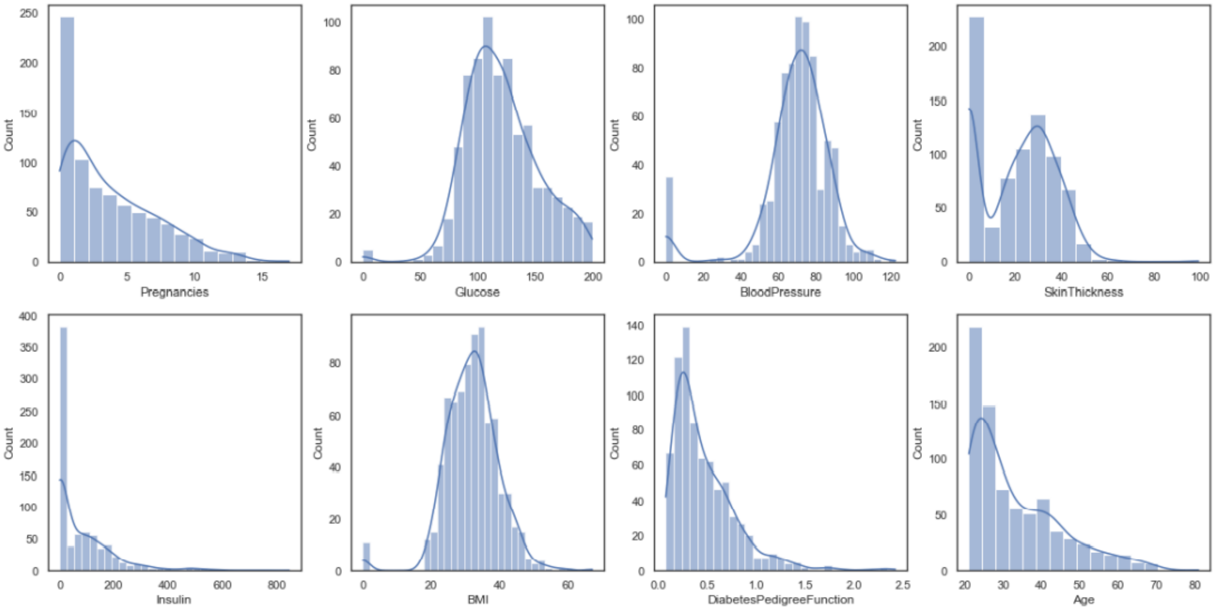
Glucose, BMI and BloodPressure columns got a few missing records, but SkinThickness and Insulin got quite a bit of missing records.

Data distribution

The data distribution for all the predictor variables:



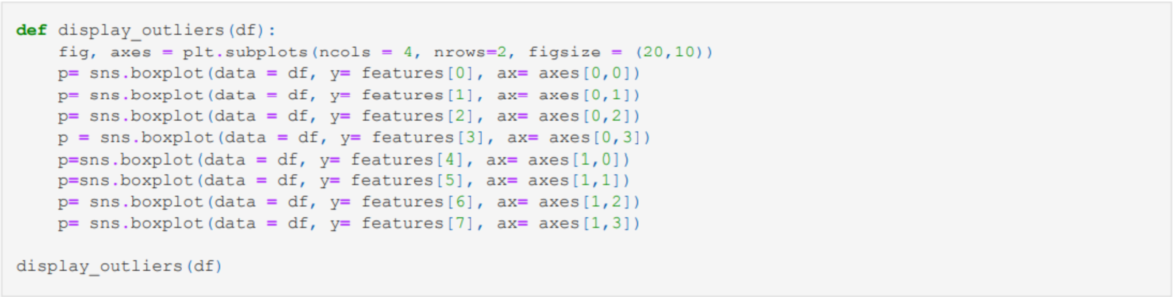
```
display_hist(df)
```



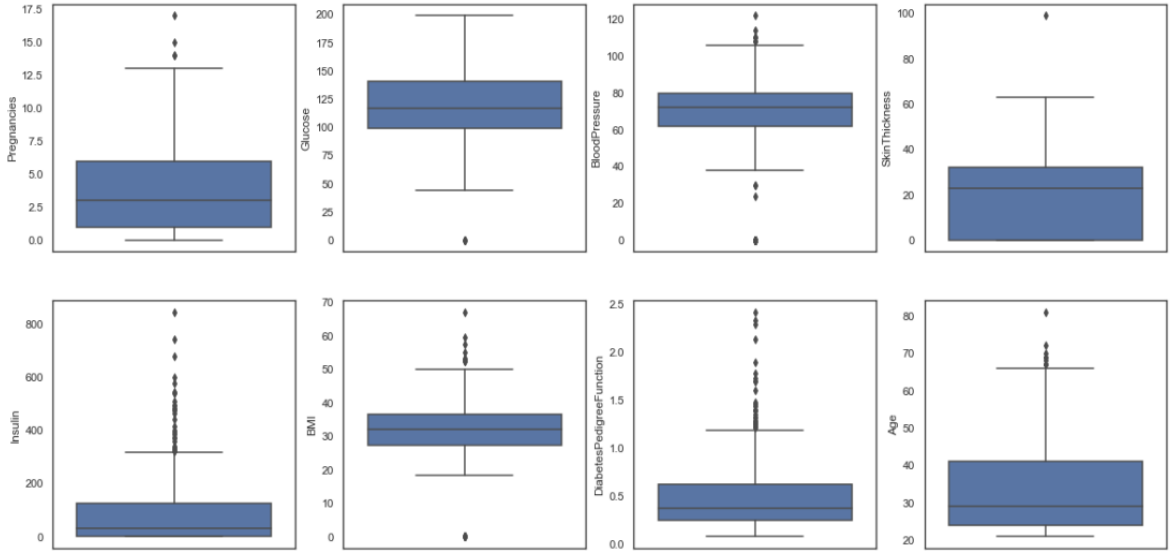
Observation

Glucose, BloodPressure, SkinThickness and BMI data are more or less normally distributed. Insulin, Pregnancies, DiabetesPedigreeFunction and Age data are all right skewed.

Outliers



```
def display_outliers(df):
    fig, axes = plt.subplots(ncols = 4, nrows=2, figsize = (20,10))
    p= sns.boxplot(data = df, y= features[0], ax= axes[0,0])
    p= sns.boxplot(data = df, y= features[1], ax= axes[0,1])
    p= sns.boxplot(data = df, y= features[2], ax= axes[0,2])
    p = sns.boxplot(data = df, y= features[3], ax= axes[0,3])
    p=sns.boxplot(data = df, y= features[4], ax= axes[1,0])
    p=sns.boxplot(data = df, y= features[5], ax= axes[1,1])
    p= sns.boxplot(data = df, y= features[6], ax= axes[1,2])
    p= sns.boxplot(data = df, y= features[7], ax= axes[1,3])

display_outliers(df)
```



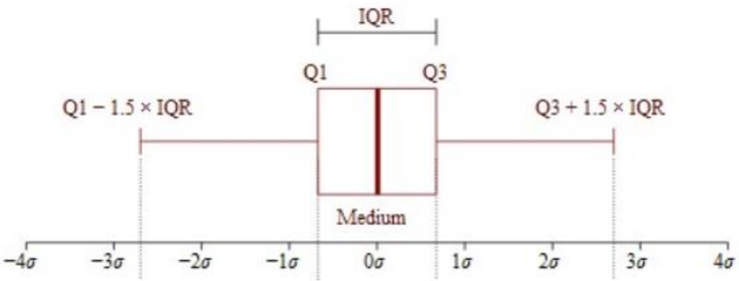
Observations:

All the predictor variables got outliers. As classifier are sensitive to the data range, it is important to remove outliers as far as possible.

Each of these variables needed to be analyzed separately to figure out how to handle missing values and outliers.

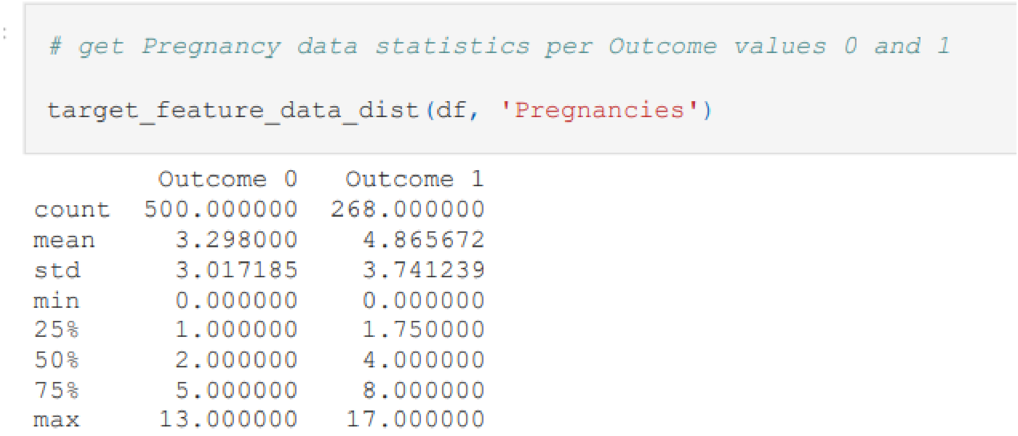
IQR Formula

I have used either Interquartile values or median values to replace the outliers or missing data.



The interquartile range is the middle 50% of a data set. Box and whiskers image by Jhguch at en.wikipedia

Outliers for Pregnancies:



```
# get Pregnancy data statistics per Outcome values 0 and 1

target_feature_data_dist(df, 'Pregnancies')
```

```
         Outcome 0   Outcome 1
count   500.000000  268.000000
mean      3.298000    4.865672
std       3.017185    3.741239
min       0.000000    0.000000
25%       1.000000    1.750000
50%       2.000000    4.000000
75%       5.000000    8.000000
max      13.000000   17.000000
```

For non-diabetic patients, mean is 3.2 and median is 2. Maximum value 13 seems a bit high

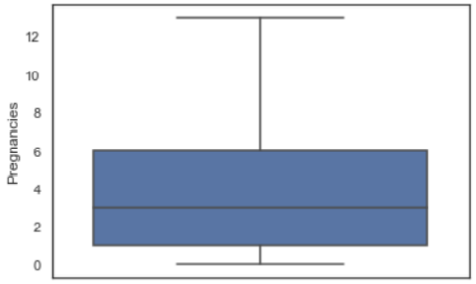
Similarly, for diabetic patients, mean is 4.8 and median is 4. The maximum pregnancies of 17 seems quite high.

This suggests that patients with higher number of pregnancies have the tendency to be diabetic.

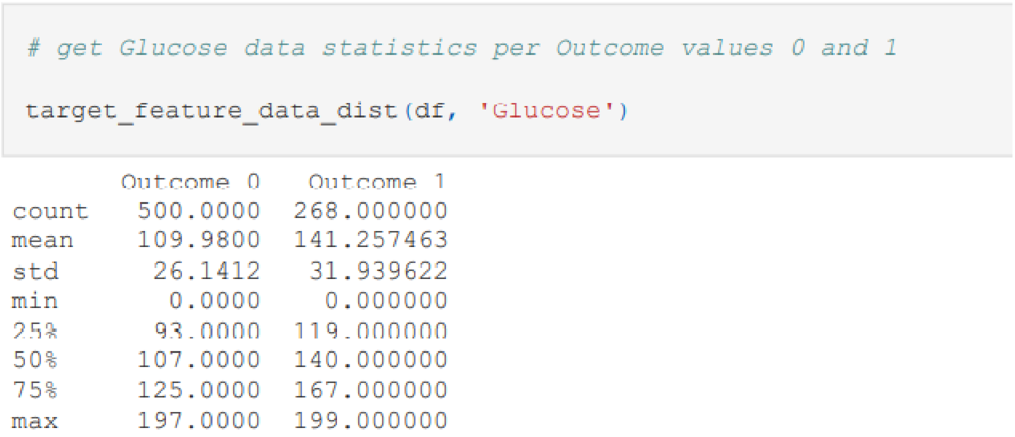
Since 0 pregnancies are possible, only values higher than 3rd quartile range needed to be changed. For Outcome 0, replacing values higher than 3rd quartile with the median value 2 and for outcome 1, replacing outlier values with 4. Since 3rd quartile values for both Outcome 0 and 1 seems little more than regular cases, median values were chosen.

x > 3rd Quartile = Median

Here is the data distribution after the value replacement.



Outliers from Glucose:



```
# get Glucose data statistics per Outcome values 0 and 1

target_feature_data_dist(df, 'Glucose')
```

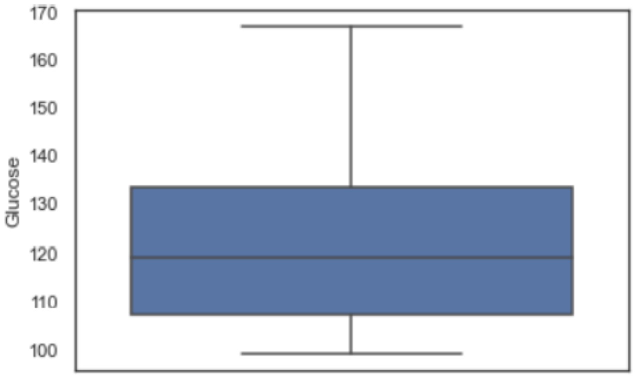
```
       Outcome 0   Outcome 1
count   500.0000  268.000000
mean    109.9800  141.257463
std      26.1412   31.939622
min       0.0000    0.000000
25%      93.0000  119.000000
50%     107.0000  140.000000
75%     125.0000  167.000000
max     197.0000  199.000000
```

The minimum value 0 is impossible. Replacing the minimum values with the individual median values. This also removes 0's from the data. For higher range of 197 for non -diabetic and 199 for diabetic seems bad data. Replacing the higher outliers with the 3rd quartile data.

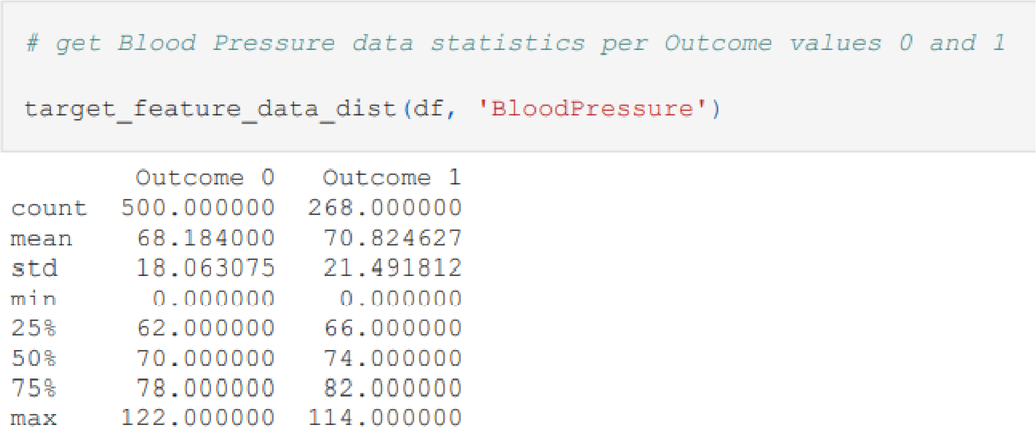
x < 1st Quartile = Median

x> 3rd Quartile = 3rd Quartile

Here is the data distribution afterwards.



Outliers for BloodPressure:



```
# get Blood Pressure data statistics per Outcome values 0 and 1

target_feature_data_dist(df, 'BloodPressure')
```

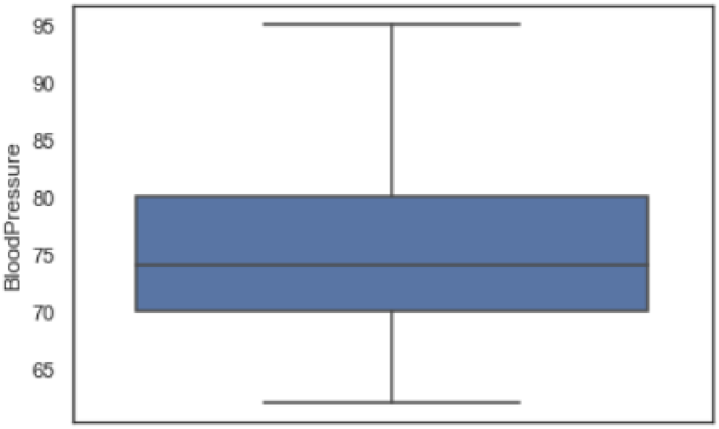
```
        Outcome 0   Outcome 1
count  500.000000  268.000000
mean    68.184000   70.824627
std     18.063075   21.491812
min      0.000000    0.000000
25%     62.000000   66.000000
50%     70.000000   74.000000
75%     78.000000   82.000000
max    122.000000  114.000000
```

Again, the minimum Diastolic blood pressure 0 is clinically impossible. Replacing the minimum values with individual medeian values for each class. Also, the maximum values got too high compared to the medians. Normal BP range is 80 – 89. So, took an acceptable 95 to replace the values greater than 3rd quartile values for individual Outcome.

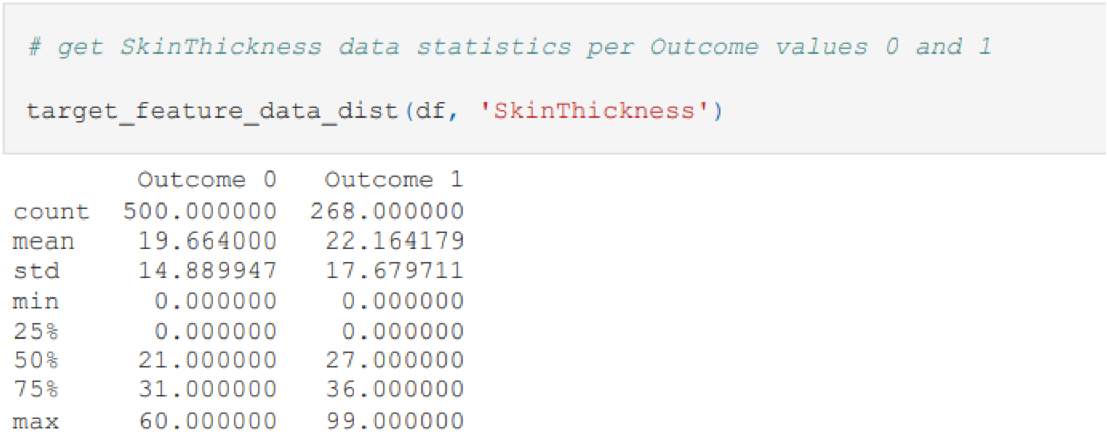
x < 1st Quartile = Median for Outcome

x> 95 = 3rd Quartile for Outcome

Here is the data distribution afterwards



Outliers for SkinThickness

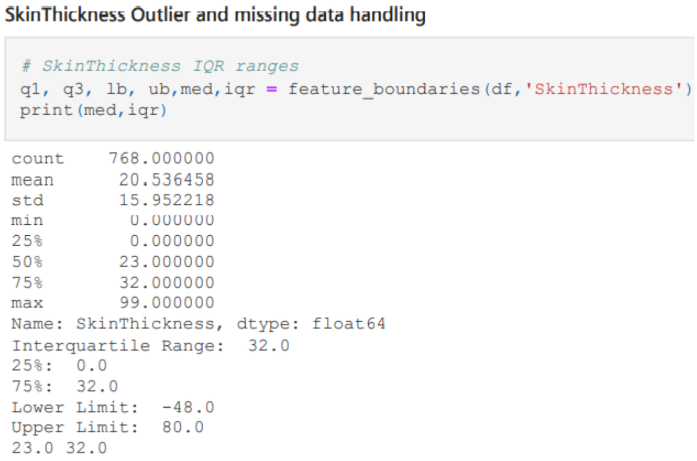


```
# get SkinThickness data statistics per Outcome values 0 and 1

target_feature_data_dist(df, 'SkinThickness')
```

```
        Outcome 0   Outcome 1
count  500.000000  268.000000
mean    19.664000   22.164179
std     14.889947   17.679711
min      0.000000    0.000000
25%      0.000000    0.000000
50%     21.000000   27.000000
75%     31.000000   36.000000
max     60.000000   99.000000
```

SkinThickness of 0 is clinically not possible. Even 1st Quartile data values are 0. So, replacing any values lower than 1st quartile with the median for the entire dataset, which is 23

Here is the for the entire column SkinThickness details:



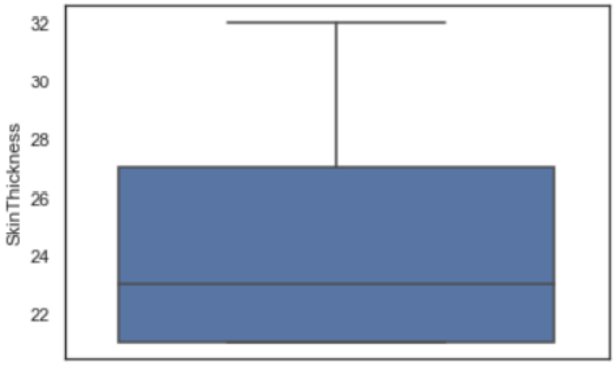
SkinThickness Outlier and missing data handling

```
# SkinThickness IQR ranges
q1, q3, lb, ub,med,iqr = feature_boundaries(df,'SkinThickness')
print(med,iqr)
```

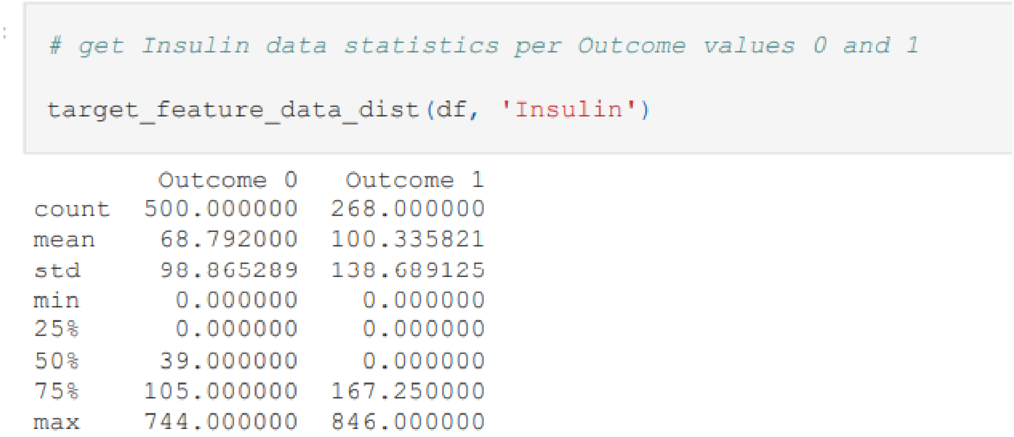
```
count    768.000000
mean      20.536458
std       15.952218
min        0.000000
25%        0.000000
50%       23.000000
75%       32.000000
max       99.000000
Name: SkinThickness, dtype: float64
Interquartile Range:  32.0
25%:  0.0
75%:  32.0
Lower Limit:  -48.0
Upper Limit:  80.0
23.0 32.0
```

x < 1st Quartile = median for the entire SkinThickness column

For higher outliers, replacing them with the median for the entire column. This is data after changing the outlier values.



Outliers for Insulin



```
# get Insulin data statistics per Outcome values 0 and 1

target_feature_data_dist(df, 'Insulin')
```

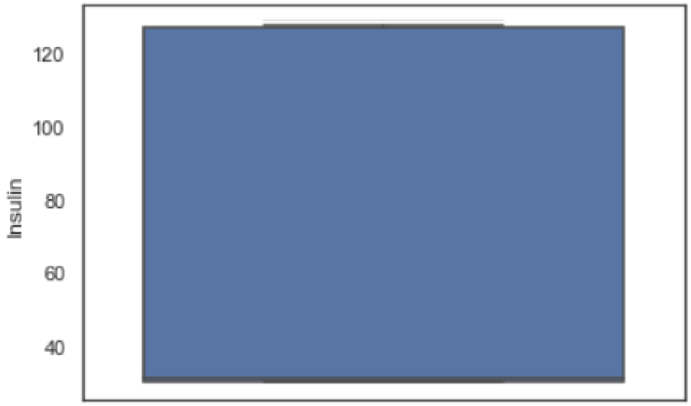
```
        Outcome 0   Outcome 1
count  500.000000  268.000000
mean    68.792000  100.335821
std     98.865289  138.689125
min      0.000000    0.000000
25%      0.000000    0.000000
50%     39.000000    0.000000
75%    105.000000  167.250000
max    744.000000  846.000000
```

Both the minimum and maximum data range for Insulin column for both Outcome 0 and 1 are clinically not possible. Also 1st quartile range is 0. For Outcome 1, even the median is 0. This is definitely bad data. For values that less than median for both Outcome 01 and 1, replacing them with the overall column median. For values more than 3rd quartile, replacing them with the 3rd quartile value for the entire column.

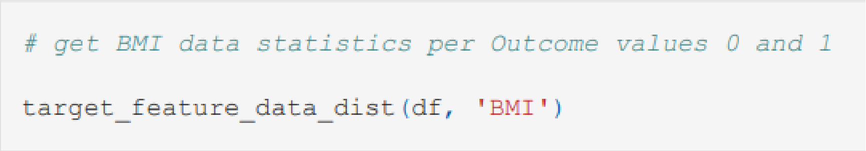
x < 1st Quartile = Median for the entire column

x> 3rd Quartile = 3rd Quartile for the entire column

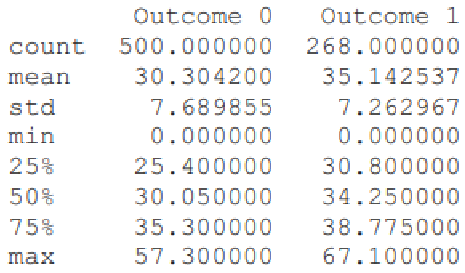
Here is the data after outlier handling



Outliers for BMI



```
# get BMI data statistics per Outcome values 0 and 1

target_feature_data_dist(df, 'BMI')
```



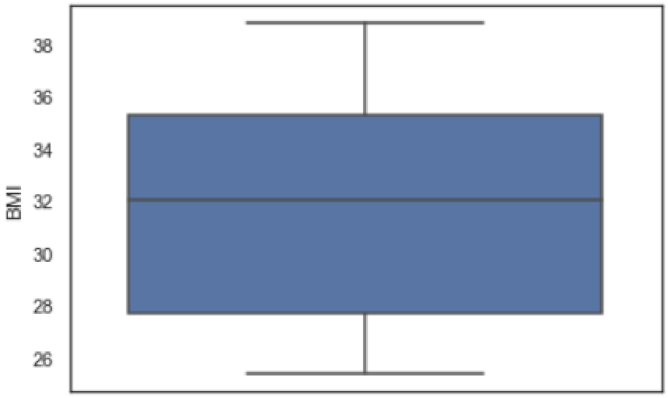
```
         Outcome 0   Outcome 1
count   500.000000  268.000000
mean     30.304200   35.142537
std       7.689855    7.262967
min       0.000000    0.000000
25%      25.400000   30.800000
50%      30.050000   34.250000
75%      35.300000   38.775000
max      57.300000   67.100000
```

BMI 0 is not possible. Replacing values lower than 1st Quartile range with the median values for each Outcome 0 and 1. For values higher than 3rd Quartile range, replacing them with individual median values.

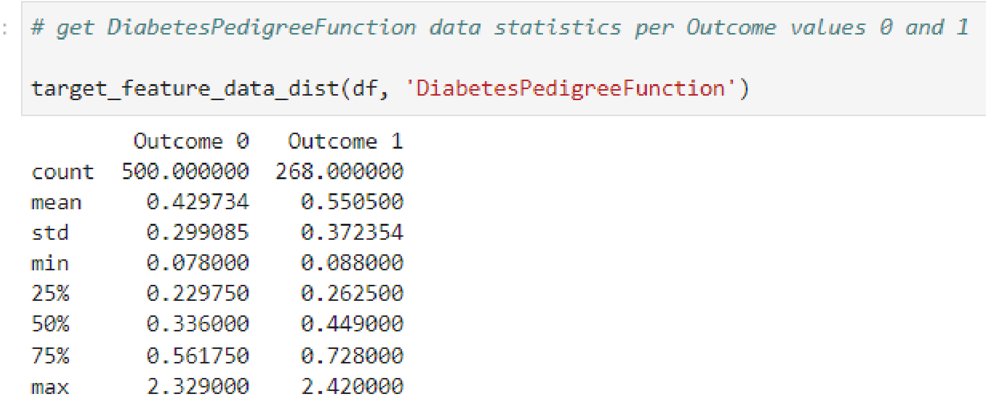
x< 1st Quartile = median

x>3rd Quartile = 3rd quartile value for each Outcome

Here is the data after the change



Outliers for DiabetesPedigreeFunction



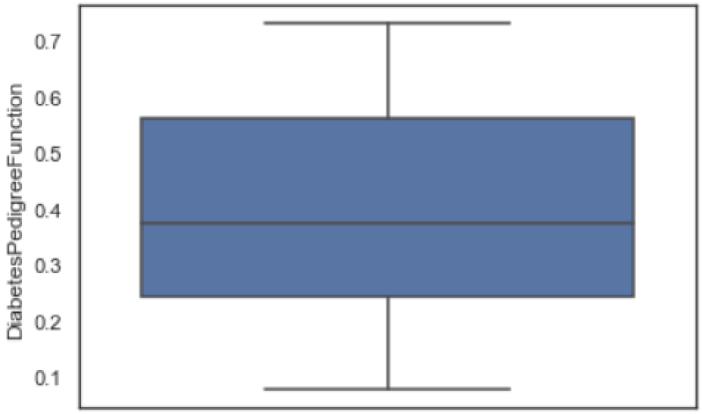
```
# get DiabetesPedigreeFunction data statistics per Outcome values 0 and 1

target_feature_data_dist(df, 'DiabetesPedigreeFunction')
```

```
        Outcome 0   Outcome 1
count  500.000000  268.000000
mean     0.429734    0.550500
std      0.299085    0.372354
min      0.078000    0.088000
25%      0.229750    0.262500
50%      0.336000    0.449000
75%      0.561750    0.728000
max      2.329000    2.420000
```

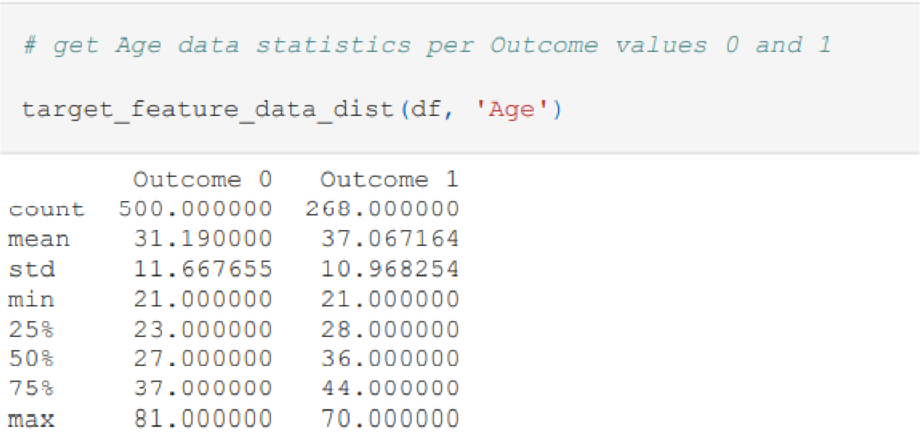
Replacing any values higher than 3rd quartile with 3rd Quartile for each Outcome to remove the outliers.

x>3rd Quartile = 3rd quartile value for each Outcome

This is the data after replacements:



Outliers for Age:



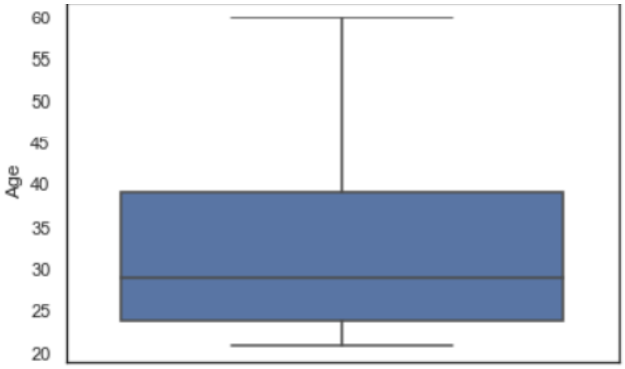
```
# get Age data statistics per Outcome values 0 and 1

target_feature_data_dist(df, 'Age')
```

```
         Outcome 0   Outcome 1
count  500.000000  268.000000
mean    31.190000   37.067164
std     11.667655   10.968254
min     21.000000   21.000000
25%     23.000000   28.000000
50%     27.000000   36.000000
75%     37.000000   44.000000
max     81.000000   70.000000
```

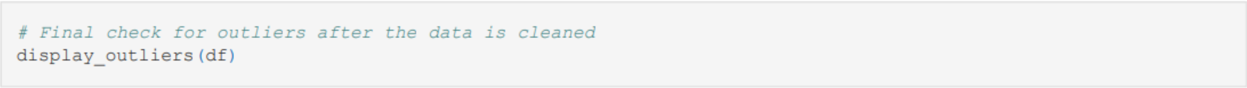
The minimum age is 21. The 3rd quartile data is around 30-40, but the maximum age is in 70-80s, causing them outliers. So, replacing all values higher than 60s with the 3rd quartile data to remove the outliers.

x> 60 = 3rd quartile value

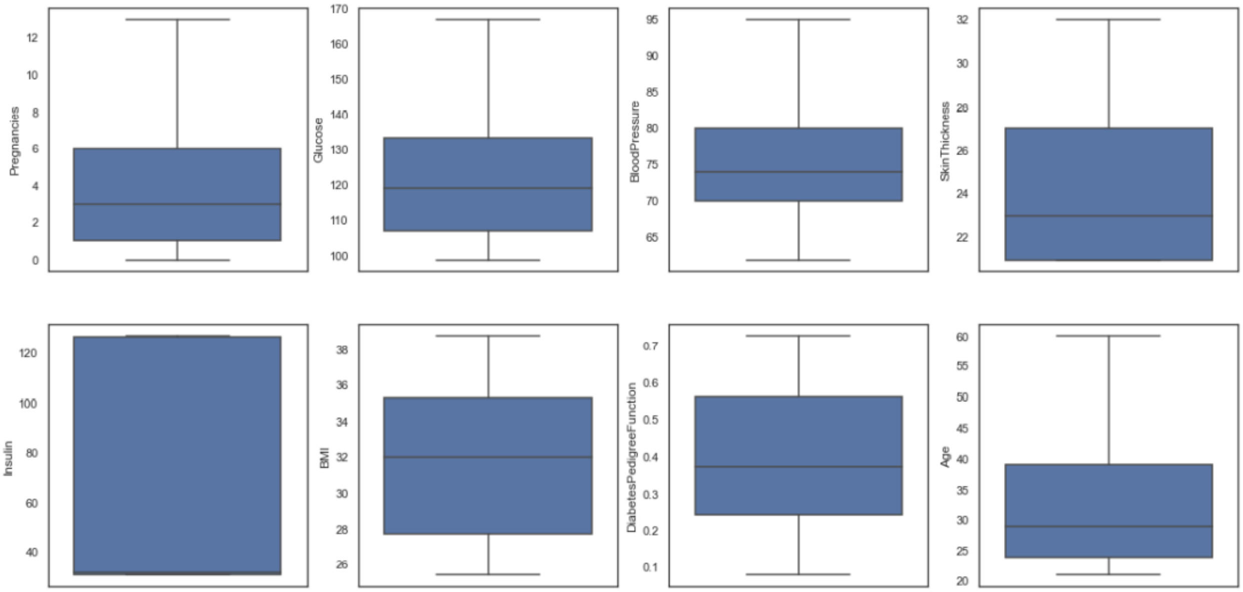
Here is the data after changes



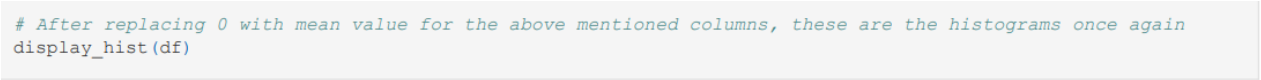
After removing outliers or missing values from all variables, here is the data distribution



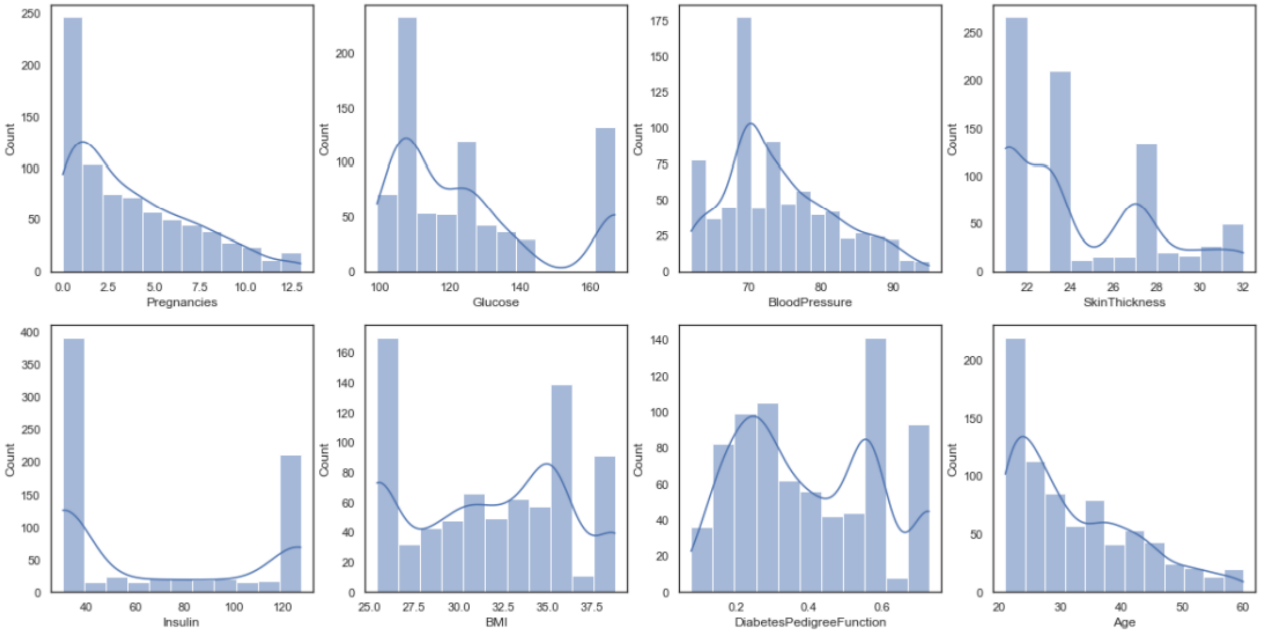
```
# Final check for outliers after the data is cleaned
display_outliers(df)
```



The histogram for all the variables after cleaning up the data. It shows that there is improvement in the data value ranges for each variable and the data is much more distributed than before.

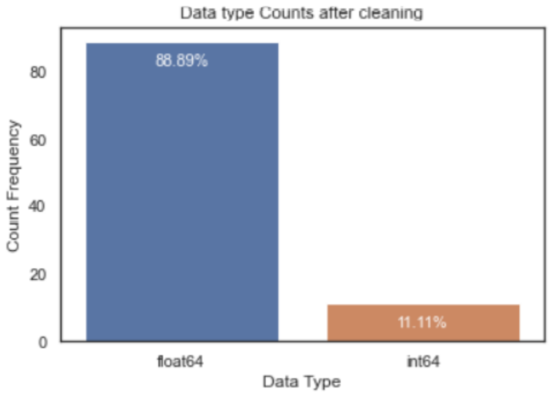


```
# After replacing 0 with mean value for the above mentioned columns, these are the histograms once again
display_hist(df)
```

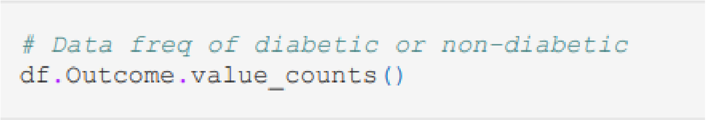


Variable Datatype

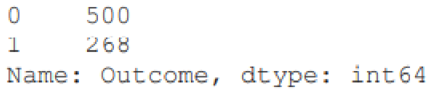
After handling all the missing values and outliers, here are the datatype distribution for the all the variables:



Data distribution for the dependent variable- Outcome:

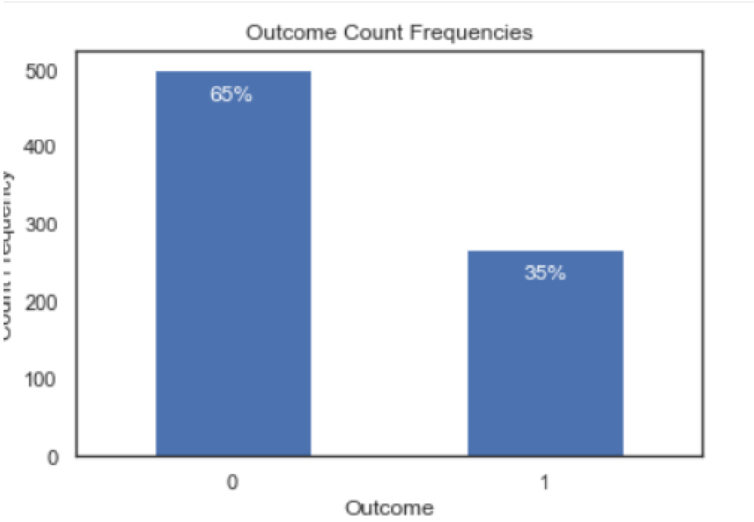


```
# Data freq of diabetic or non-diabetic
df.Outcome.value_counts()
```

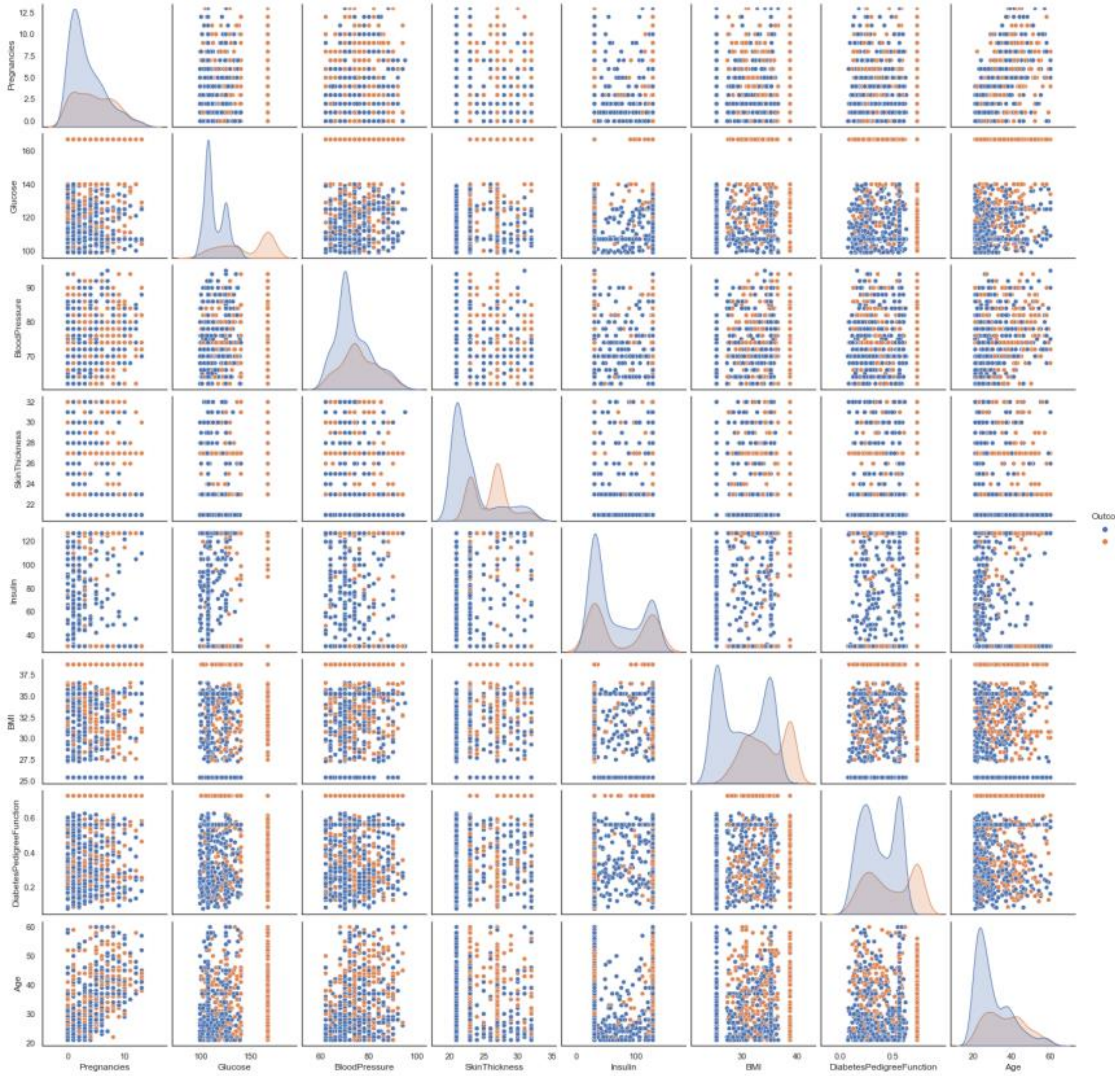


```
0    500
1    268
Name: Outcome, dtype: int64
```

Number of cases marked as diabetic are just 50% less than the cases marked as unbalanced. This is **highly unbalanced** data.

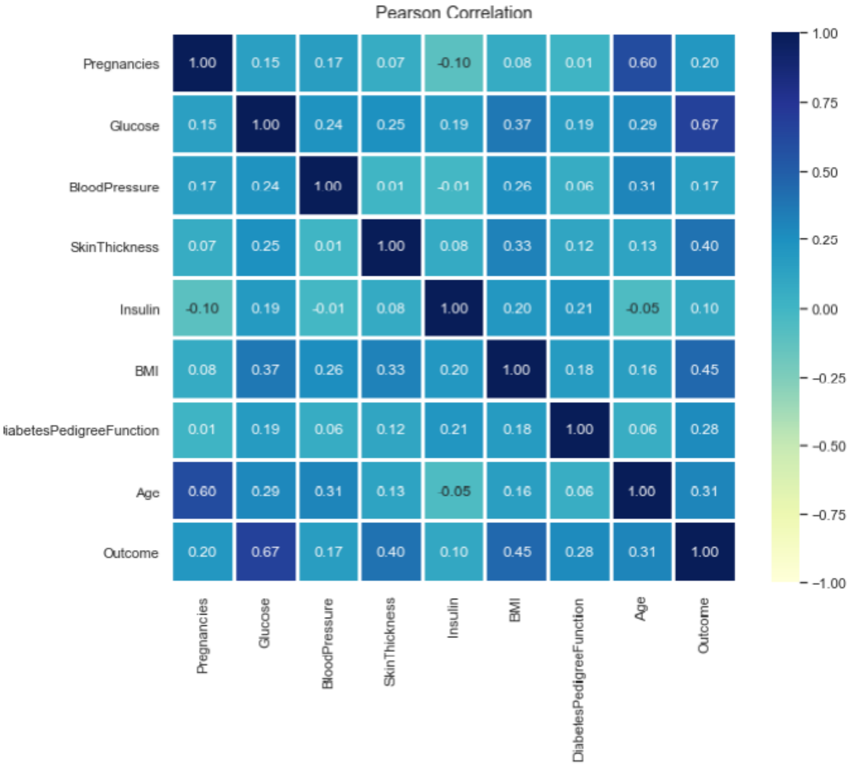


Correlation between the variables



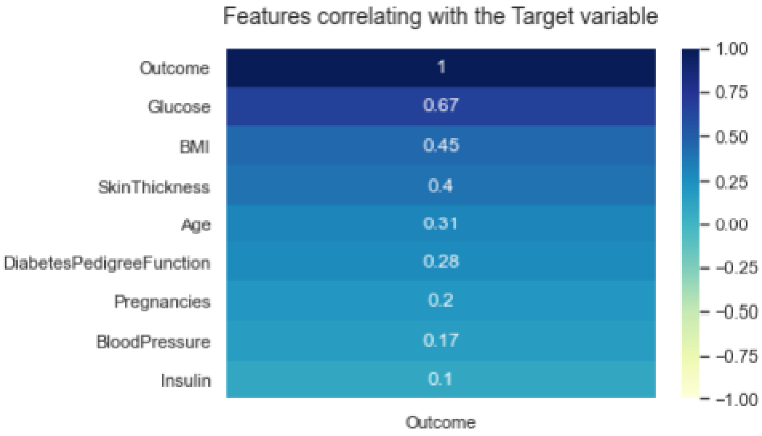
For Pregnancies and Age there is some positive relationship. Distribution of Glucose by Outcome (Orange – 1, Blue -0) shows increase in Glucose level increases the risk of diabetes. Distribution of SkinThickness and Outcome indicates increase in SkinThickness elevates the diabetes. Individual distribution graphs for BMI, DiabeticsPedigreeFunction and Age and Outcome shows increased level causing diabetes.

HeatMap showing the Pearson correlation coeeficient among all variables including the Outcome



Looks like all the independent variables are positively co related with the dependent variable Outcome. Age and Pregnancies variables have moderately positive co relationship among themselves.

Here is the sorted list of how each variable is related to Outcome



Glucose is the most important column that is influencing the Outcome. Insulin values are the least. But all of the 8 independent variables influence the Outcome.

MODEL BUILDING

Here are the steps followed:

- Identify predictor and target variables
- Split dataset into Training and Test
- Feature Scaling
- Correct un balanced target data using SMOTE
- Identify models to include
- Collect the best possible parameters for each model
- Run and compare the model accuracies
- Get the model with the best accuracy score
- Perform KFold Cross Validation and pick the model with the best accuracy score

For classification problem, the independent variables play a big role to predict the outcome.

I am keeping all the independent variables for model building dataset: Glucose, BMI, SkinThickness, Age, DiabetesPedigreeFunction, Pregnancies, BloodPressure and Insulin.

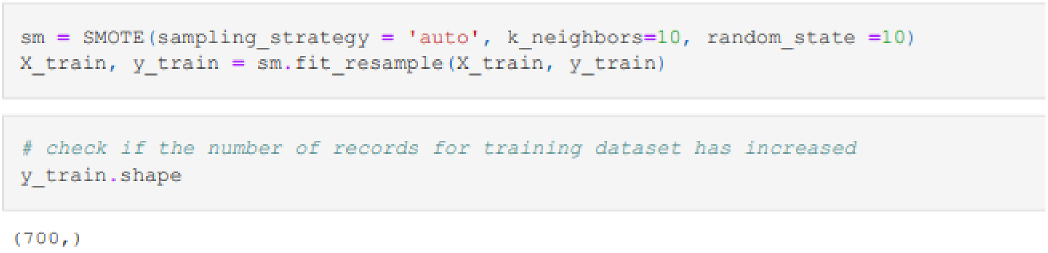
In order to build the model, we need to train the dataset first and then test the accuracy of the model. For this, I am splitting the model into train and test. As the dataset is small, I am splitting the dataset into 70:30 ratio.

Balancing the train dataset

As this is an unbalanced dataset, some machine learning models have built in learning functionality to compensate this and some does not. Here the target variable Outcome is a binary variable. The balanced plot for Outcome variable showed that the number of diabetic patients is 268 while the number of non-diabetic patients is 500. That means, the Outcome is biased. That means the model will have poor predictive performance, specifically for the minority class (i.e., Diabetic prediction). To resolve this class imbalance, resampling technique is used which will do random oversampling and undersampling. As the dataset is small, resampling needs to be done only for the training dataset. Oversampling means to randomly duplicate the records from minority class and undersampling means removing or deleting records from the majority class.

SMOTE (Synthetic Minority Oversampling Technique) is one way to balance the dataset.

After applying the SMOTE, training dataset got more data.



```
sm = SMOTE(sampling_strategy = 'auto', k_neighbors=10, random_state =10)
X_train, y_train = sm.fit_resample(X_train, y_train)
```

```
# check if the number of records for training dataset has increased
y_train.shape
```

```
(700,)
```

Feature Scaling

As all input features are numerical values, will perform Standardization i.e., Z-Score normalization

Formula

$$Z = \frac{x - \mu}{\sigma}$$

Z = standard score
x = observed value
μ = mean of the sample
σ = standard deviation of the sample

This is done to put scales for all predictor variables in the same scale.

Classification Algorithms

Approach

Build the model with K-Nearest Neighbor (KNN)

Check its accuracy

Use hyper parameter tuning with GridSearchCV to get the best possible score

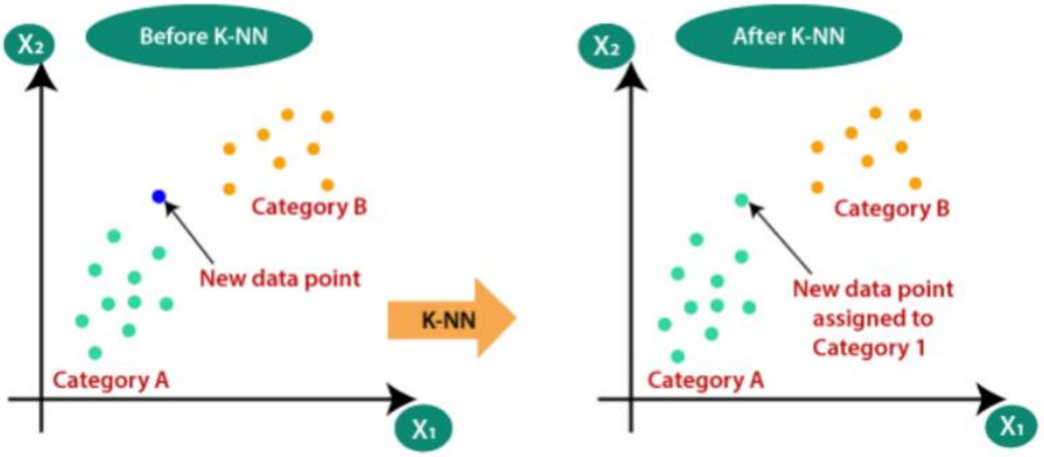
Build other models and check their accuracy scores

Compare which model gives the better accuracy

Models used in this project

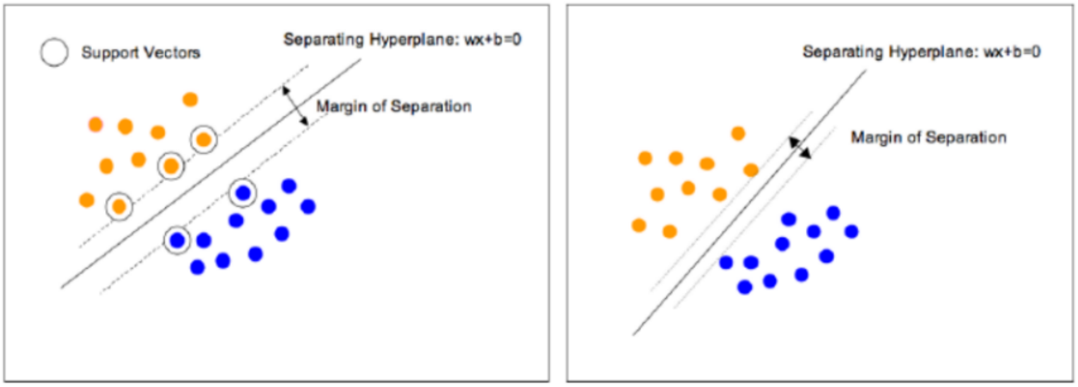
K-Nearest Neighbor (KNN)

The k-nearest neighbor algorithm is a type of supervised machine learning algorithm. It is used for classification and regression. It estimates the likelihood of grouping a data point based on commonality of k nearest neighbors. Weights are assigned to the neighbors based on the distance from the data point. That means closed data points will have more weights than others. Normally k is a positive integer. Smaller k value means a very few nearest neighbors will be picked for classification. This usually results in model overfitting. Also, a very big k indicates a lot of nearest neighbors will be picked. In that case model will be underfitting. So, a proper k value needs to be identified. As this method is a supervised ML algorithm, it relies on labelled input data to predict classes for unlabeled data points.



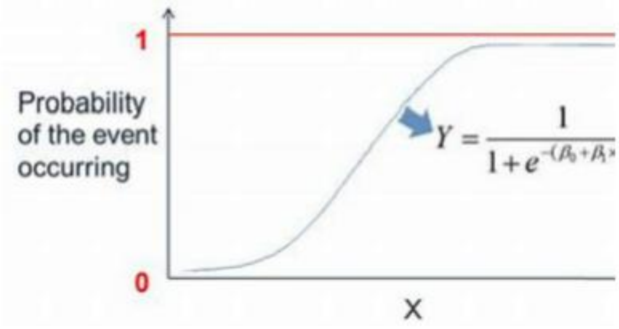
Support Vector Machine SVM

Support Vector Machine (SVM) is a supervised machine learning algorithm that is used to identify a hyperplane that got the maximum distance between the two classes in a training dataset. Using that information, it can classify any new datapoints.



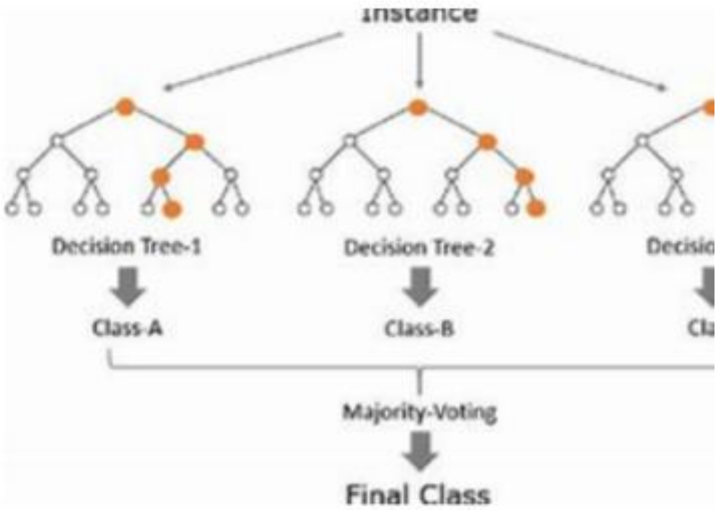
Logistic Regression

Logistic Regression Logistic regression is a supervised classification algorithm. It utilizes a linear combination of an input datapoint to solve a binary classification problem (i.e., there are only two possible classes).



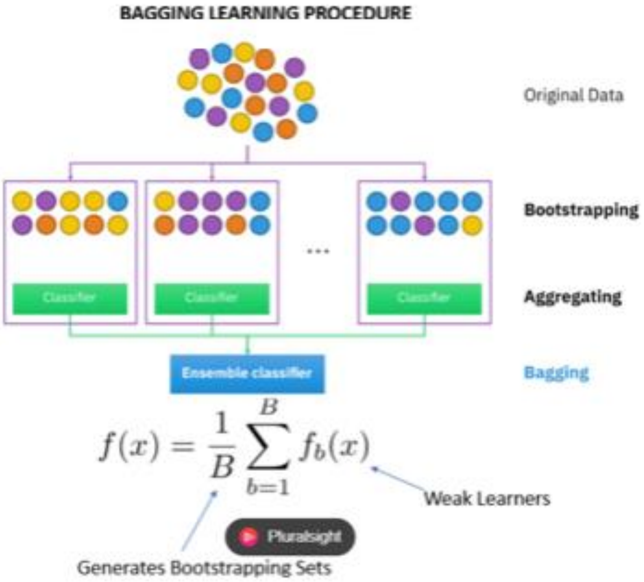
Random Forest

The bootstrapping Random Forest algorithm combines ensemble learning methods with the decision tree framework to create multiple randomly drawn decision trees from the data, averaging the results to output strong predictions/classifications.



Bagging Classifier

Bagging Classifier technique is also called bootstrap aggregation. It is a data sampling technique where data is sampled with replacement. Bootstrap aggregation is a machine learning ensemble meta-algorithm for reducing the variance of an estimate produced by bagging, which reduces its stability and enhances its bias.



Confusion Matrix

Confusion Matrix is used to evaluate the performance of a classifier.

TP: actual value is Positive and predicted is also Positive

TN: actual value is Negative and prediction is also Negative

FP: actual is negative but prediction is Positive

FN: actual is Positive but the prediction is Negative

Precision = Gives how many predictions are actually positive out of all the total positive predicted.

Recall/ Sensitivity = Out of all the positive classes, how much we have predicted correctly. In medical prediction, it is important to predict actual positive cases! So, recall score is highly important in this project

F1-Score = Harmonic mean of precision and recall

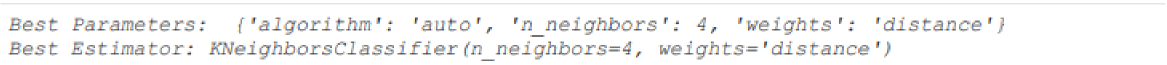
AUC ROC Curve = ROC is a probability curve and AUC is the measure of separability. It tells how much the model is capable of distinguishing between classes. Higher AUC indicates better predicting 0 classes as 0 and 1 classes as 1. In this project, we would like to see higher AUC, which means how well the model can distinguish between patients with diabetes and not.

Formula:

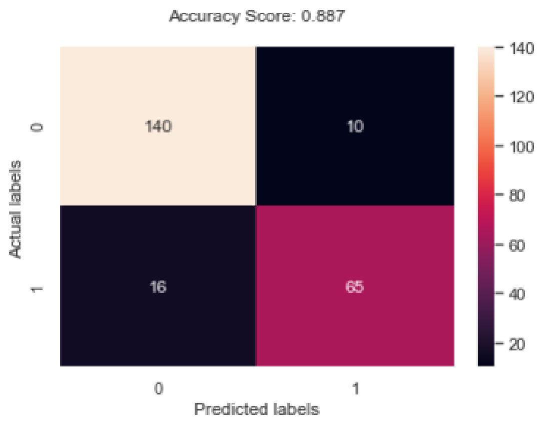
Accuracy = (TP +TN) / (TP + TN + FP +FN)
Precision = TP / (TP + FP)
Recall (Sensitivity)=TP / (TP + FN)
F1 score = 2 x ((Precision x Recall) / (Precision + Recall))
True Negative Rate (TNR) = TN/(TN+FN)
False Positive Rate (FPR) = 1 –Specificity

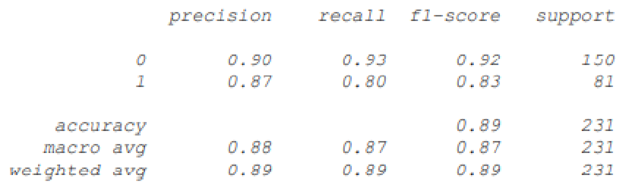
KNN Model Outcome

I utilized the GridsearchCV to obtain the best parameters to get best result possible for KNN model. These are the best parameters I got:

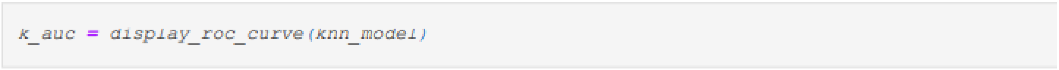


```
Best Parameters:  {'algorithm': 'auto', 'n_neighbors': 4, 'weights': 'distance'}
Best Estimator: KNeighborsClassifier(n_neighbors=4, weights='distance')
```

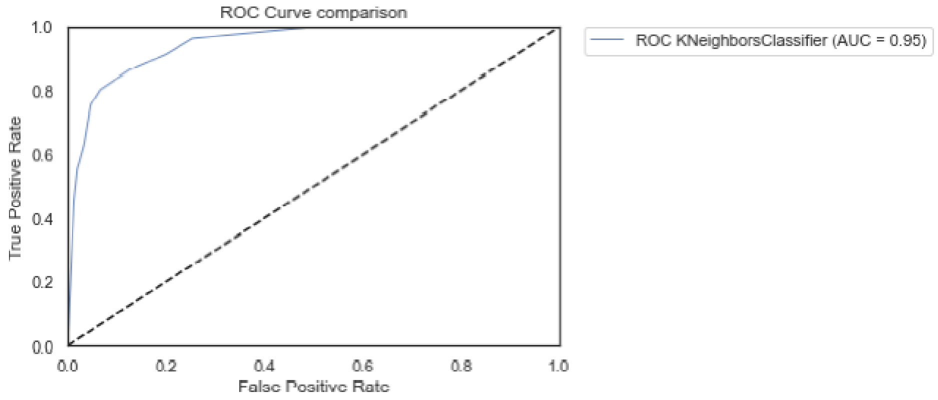




```
              precision    recall  f1-score   support

           0       0.90      0.93      0.92       150
           1       0.87      0.80      0.83        81

    accuracy                           0.89       231
   macro avg       0.88      0.87      0.87       231
weighted avg       0.89      0.89      0.89       231
```



```
k_auc = display_roc_curve(knn_model)
```



Observations

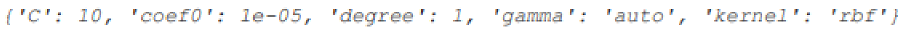
KNN model is able to detect 80% of the diabetic cases(recall) and 87% of its prediction are correct (precision). F1-Score is 83%. Test Accuracy is 89%. AUC score is 95%.

Overall, a very good prediction.

Support Vector Machine (SVM) Model Outcome

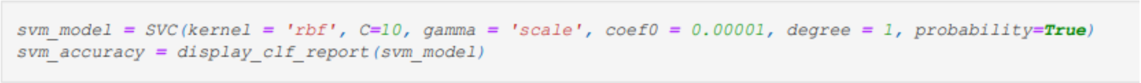
Utilized GridSearchCV to get the best parameters to pass for the SVM model

These are the best parameters returned:

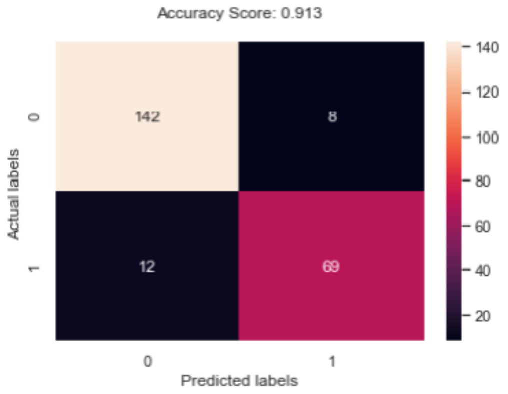


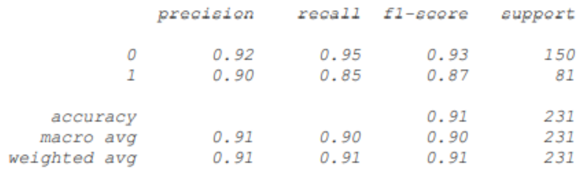
```
{'C': 10, 'coef0': 1e-05, 'degree': 1, 'gamma': 'auto', 'kernel': 'rbf'}
```

Here is the confusion matrix

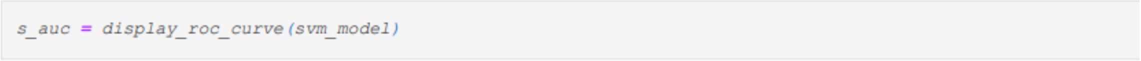


```
svm_model = SVC(kernel = 'rbf', C=10, gamma = 'scale', coef0 = 0.00001, degree = 1, probability=True)
svm_accuracy = display_clf_report(svm_model)
```

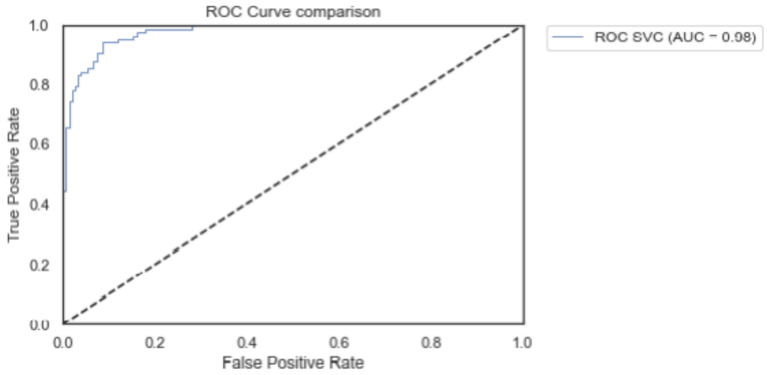




```
              precision    recall  f1-score   support

           0       0.92      0.95      0.93       150
           1       0.90      0.85      0.87        81

    accuracy                           0.91       231
   macro avg       0.91      0.90      0.90       231
weighted avg       0.91      0.91      0.91       231
```



```
s_auc = display_roc_curve(svm_model)
```

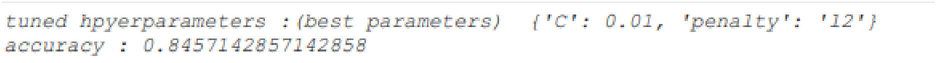


Observations:

SVM model is able to detect 85% of the diabetic cases(recall) and 90% of its prediction are correct (precision). The test accuracy score is 91%. AUC score is 98%. Improved accuracy than KNN

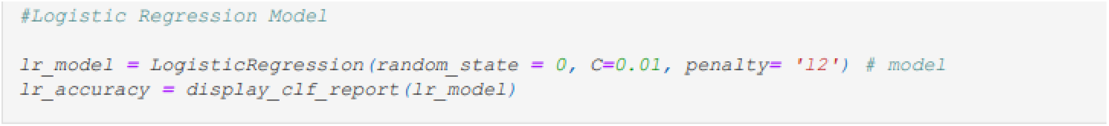
Logistic Regression Model Outcome

The best parameters returned through GridSearchCV are

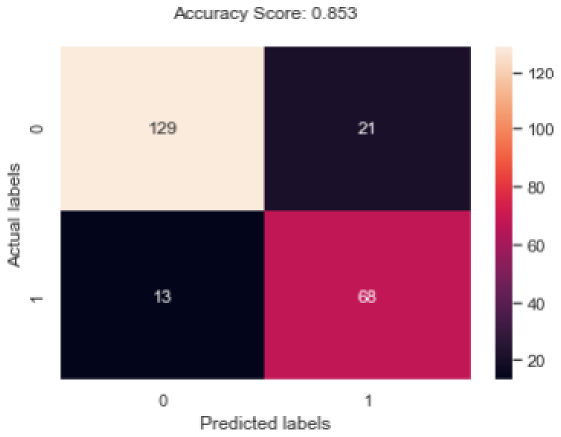


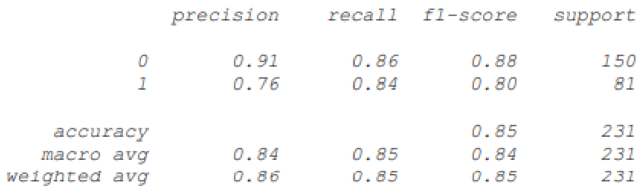
```
tuned hpyerparameters :(best parameters)  {'C': 0.01, 'penalty': 'l2'}
accuracy : 0.8457142857142858
```

Using these parameter values, this is the confusion matrix and ROC AUC curve I got:

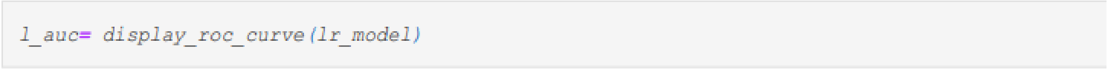


```
#Logistic Regression Model

lr_model = LogisticRegression(random_state = 0, C=0.01, penalty= 'l2') # model
lr_accuracy = display_clf_report(lr_model)
```

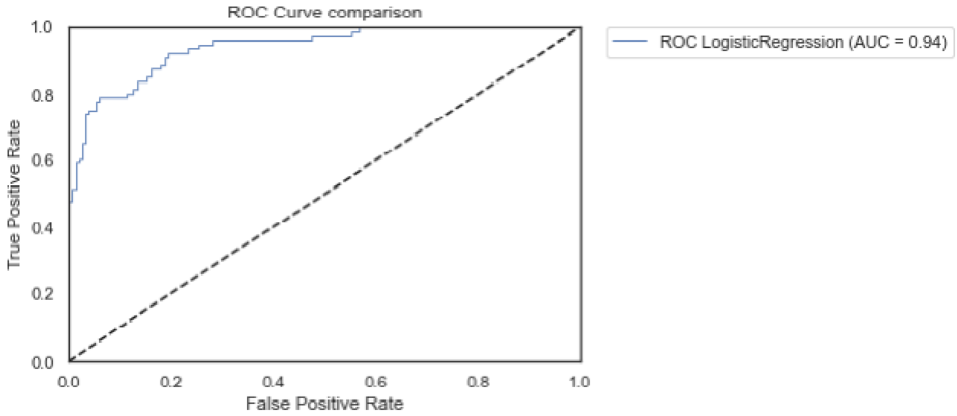




```
              precision    recall  f1-score   support

           0       0.91      0.86      0.88       150
           1       0.76      0.84      0.80        81

    accuracy                           0.85       231
   macro avg       0.84      0.85      0.84       231
weighted avg       0.86      0.85      0.85       231
```



```
l_auc= display_roc_curve(lr_model)
```

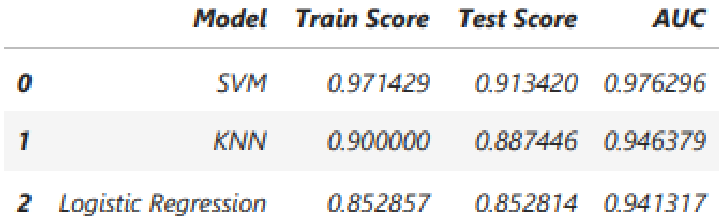


Observations:

Logistic Regression model is able to detect 84% of the diabetic cases(recall) and 76% of its prediction are correct (precision). The test accuracy score is 85%. AUC score is 94%

Model Comparison

Comparing 3 models: KNN, SVM and Logistic Regression



	Model	Train Score	Test Score	AUC
0	SVM	0.971429	0.913420	0.976296
1	KNN	0.900000	0.887446	0.946379
2	Logistic Regression	0.852857	0.852814	0.941317

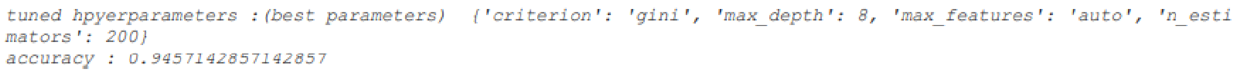
Clearly, SVM got out to be the best among the three. All three models perform well in Train and Test datasets. The AUC score suggests all three can predict diabetes pretty well.

Now I check these models along with two ensemble models to see which one gives better prediction.

ENSEMBLE MODEL APPLICATION

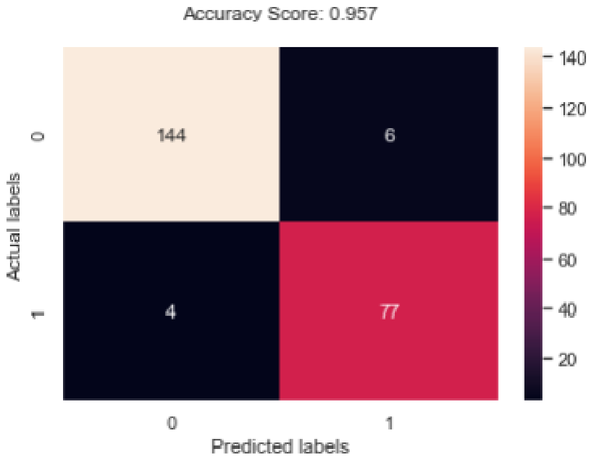
Random Forest

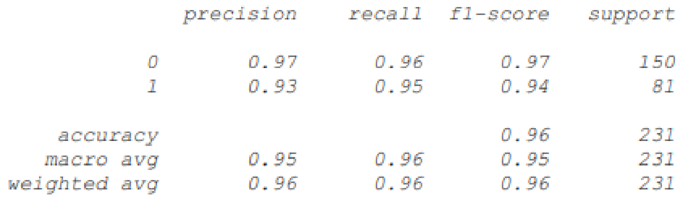
These are the best estimators returned through GridSearchCV



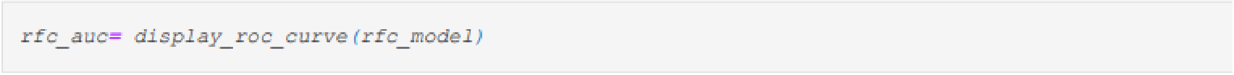
```
tuned hpyerparameters :(best parameters)  {'criterion': 'gini', 'max_depth': 8, 'max_features': 'auto', 'n_esti
mators': 200}
accuracy : 0.9457142857142857
```

Here is the confusion matrix and ROC AUC curve

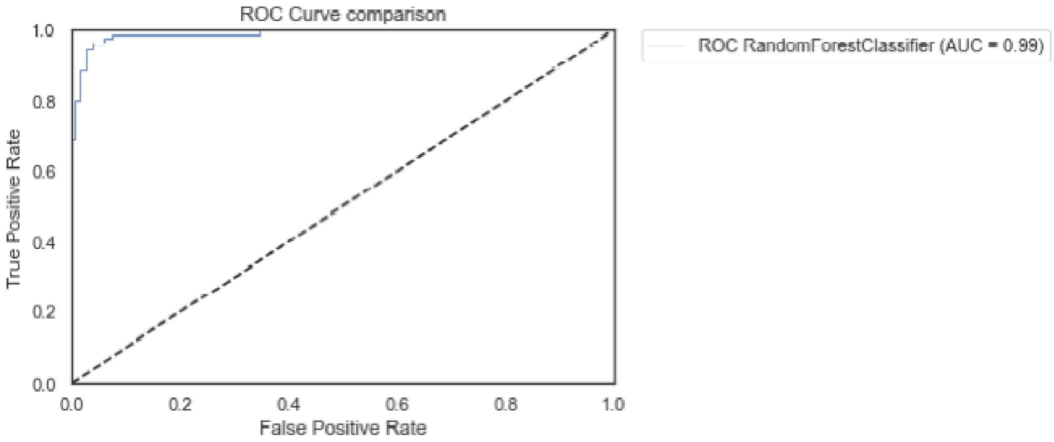




```
              precision    recall  f1-score   support

           0       0.97      0.96      0.97       150
           1       0.93      0.95      0.94        81

    accuracy                           0.96       231
   macro avg       0.95      0.96      0.95       231
weighted avg       0.96      0.96      0.96       231
```



```
rfc_auc= display_roc_curve(rfc_model)
```

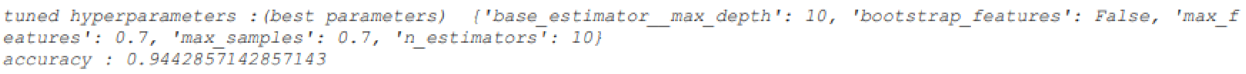


Observations

Random Forest model is able to detect 95% of the diabetic cases(recall) and 93% of its prediction are correct (precision). The test accuracy score is 96%. The AUC score is 99%

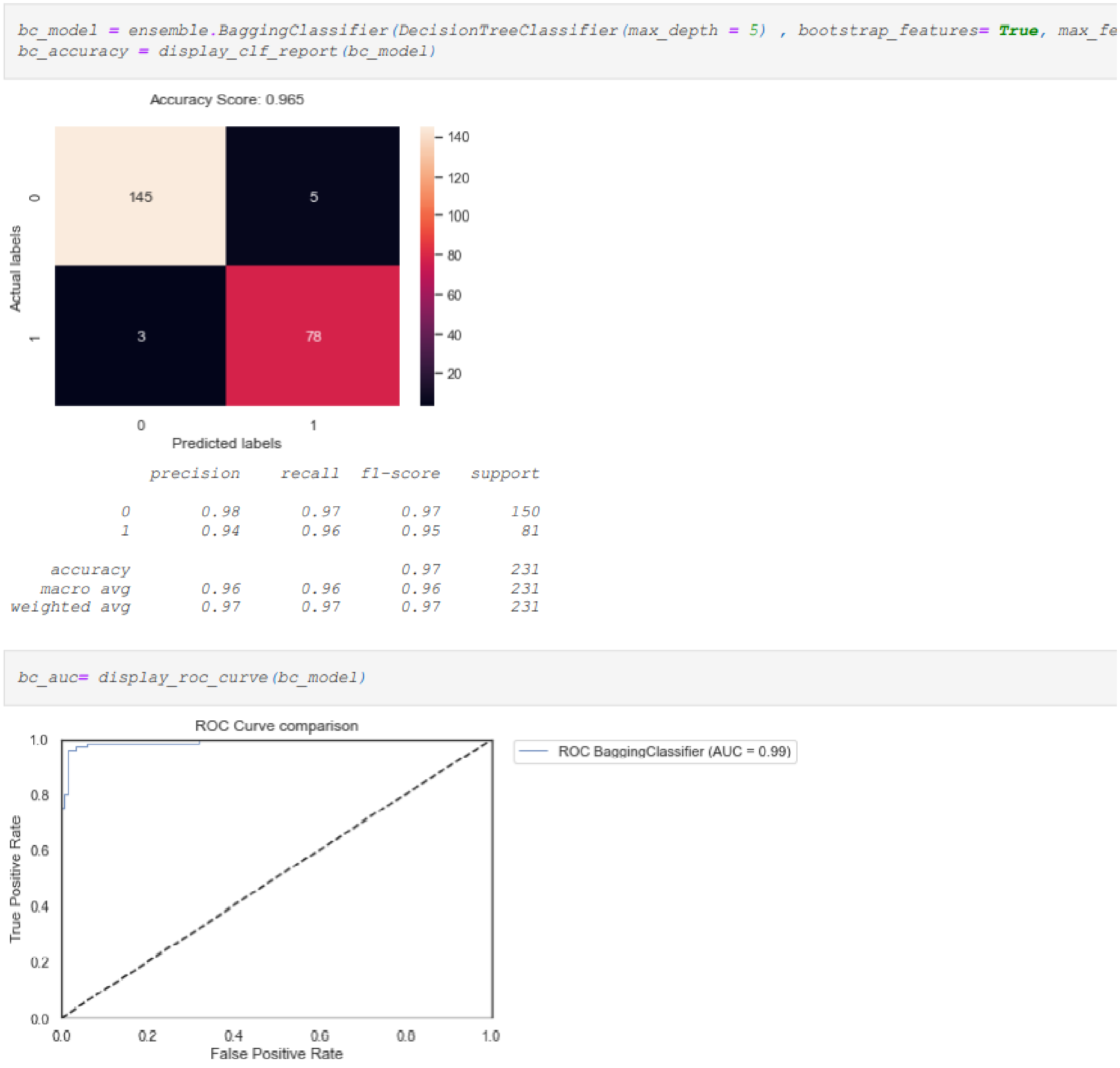
Bagging Classifier

Here are the best parameters returned through GridSearchCV



```
tuned hyperparameters :(best parameters)  {'base_estimator__max_depth': 10, 'bootstrap_features': False, 'max_f
eatures': 0.7, 'max_samples': 0.7, 'n_estimators': 10}
accuracy : 0.9442857142857143
```

Here is the confusion matrix and ROC AUC Curve



```
bc_model = ensemble.BaggingClassifier(DecisionTreeClassifier(max_depth = 5) , bootstrap_features= True, max_fe
bc_accuracy = display_clf_report(bc_model)
```

Accuracy Score: 0.965

```
              precision    recall  f1-score   support

           0       0.98      0.97      0.97       150
           1       0.94      0.96      0.95        81

    accuracy                           0.97       231
   macro avg       0.96      0.96      0.96       231
weighted avg       0.97      0.97      0.97       231
```

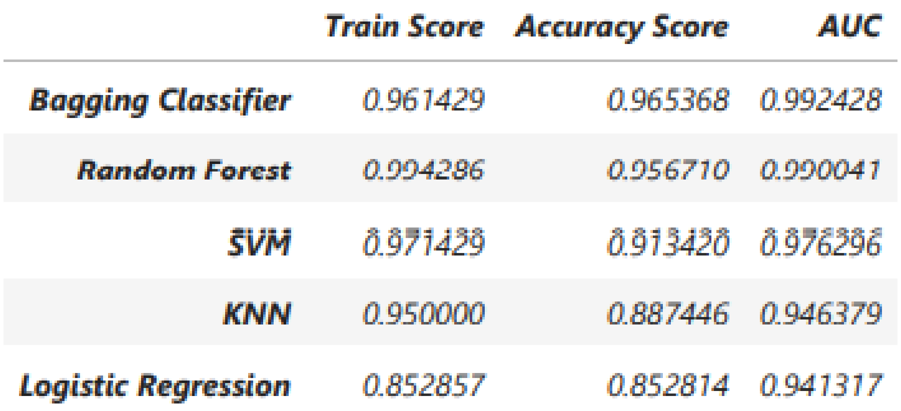
```
bc_auc= display_roc_curve(bc_model)
```

Observations:

Bagging Classifier model is able to detect 96% of the diabetic cases(recall) and 94% of its prediction are correct (precision). The test accuracy score is 97%. AUC score is 98%

Comparing all Models

Comparing all models KNN, SVM, Logistic Regression, Random Forest and Bagging Classifier



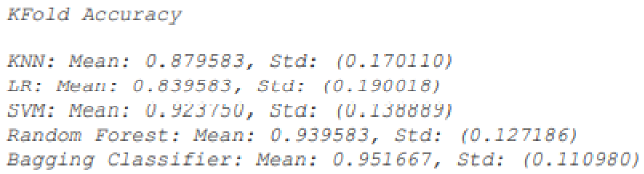
	Train Score	Accuracy Score	AUC
Bagging Classifier	0.961429	0.965368	0.992428
Random Forest	0.994286	0.956710	0.990041
SVM	0.971429	0.913420	0.976296
KNN	0.950000	0.887446	0.946379
Logistic Regression	0.852857	0.852814	0.941317

Clearly, **Bagging Classifier** model is predicting the best out of all. It's train and test accuracy score are more or less similar, and AUC score is 99%.

Perform K Fold Cross Validation

Just to make sure this model will perform well compared to others with unseen data, I did K Fold Cross Validation. The performance on unseen data will see if this model is Under-fitting/Over-fitting/Well generalized. Cross validation (CV) is used to test the effectiveness of a machine learning model by re-sampling data to evaluate a model with limited data.

Here is the result

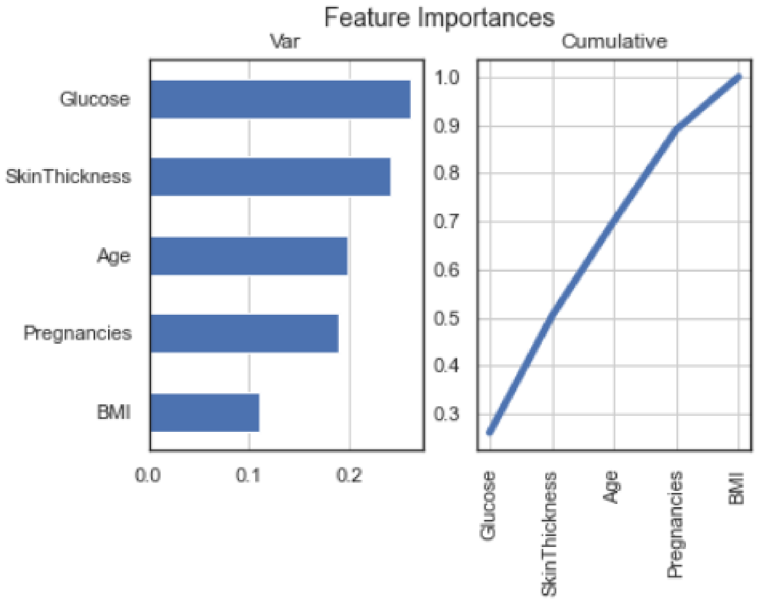


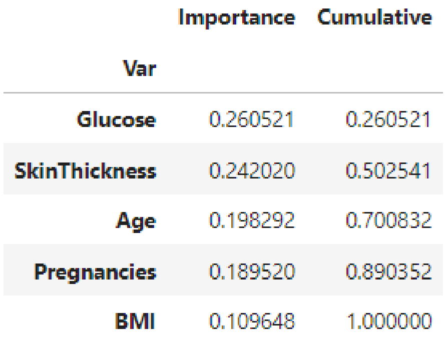
```
KFold Accuracy

KNN: Mean: 0.879583, Std: (0.170110)
LR: Mean: 0.839583, Std: (0.190018)
SVM: Mean: 0.923750, Std: (0.138889)
Random Forest: Mean: 0.939583, Std: (0.127186)
Bagging Classifier: Mean: 0.951667, Std: (0.110980)
```

The Mean score is the average accuracy score of all the accuracy scores performed with different combination of train and test datasets. Clearly, Bagging Classifier got the highest score with the lowest Standard Deviation.

FEATURE IMPORTANCE

These are the top five features getting used by Bagging Classifiers





	Importance	Cumulative
Var		
Glucose	0.260521	0.260521
SkinThickness	0.242020	0.502541
Age	0.198292	0.700832
Pregnancies	0.189520	0.890352
BMI	0.109648	1.000000

Bagging Classifier model is the one I pick to predict diabetes.