CURNEU MEDTECH INNOVATIONS PRIVATE LIMITED

SD03Q03

- **BAALA VIDHYA SREE K C (1832011)**

PROBLEM STATEMENT – 2

PREDICTION OF DIABETES USING SUITABLE ML MODEL

ABSTARCT:

This document deals with finding a suitable model to predict the results for the given Diabetes dataset. The model that found suitable here for this dataset is Random Forest model. For the given dataset, data preprocessing, removal outliers (if any), and Exploratory Data Analysis (EDA), are done and then we compare built – in classifier functions of different classification algorithms and build a scratch code for the most suitable algorithm and finally predict results of new data from the created classifier.

INTRODUCTION:

Diabetes is a metabolic disease that causes high blood sugar. We can find if a person is Diabetic or not with certain parameters whose values (level) in the body decides if a person is diabetic or not. These values can be fed to a machine as data for which a model can be developed and can be made to identify the accurate status of that particular individual.

ABOUT THE DATASET:

The given 'Diabetes Databse.csv' contains the following attributes:

1. Pregnancies
2. Glucose
3. BloodPressure
4. SkinThickness
5. Insulin
6. BMI
7. DiabetesPedigreeFunction
8. Age
9. Outcome

About the attributes:

⇒ Pregnancies: No. of times pregnant
⇒ Glucose: Plasma glucose concentration (2-Hour Oral Glucose Tolerance Test)
⇒ BloodPressure: Diastolic blood pressure (mm Hg)
⇒ SkinThickness: Triceps skin fold thickness (mm)
⇒ Insulin: 2-Hour serum insulin (mu U/ml)
⇒ BMI: Body Mass Index (weight in kg / (height in m) ^ 2)

⇒ DiabetesPedigreeFunction: It provides a synthesis of the diabetes mellitus history in relatives and the genetic relationship of those relatives to the subject.
⇒ Age: Age of the people (in years)
⇒ Outcome: Whether the person is Diabetic (1) or Not diabetic (0).

A	B	C	D	E	F	G	H	I
Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
6	148	72	35	0	33.6	0.627	50	1
1	85	66	29	0	26.6	0.351	31	0
8	183	64	0	0	23.3	0.672	32	1
1	89	66	23	94	28.1	0.167	21	0
0	137	40	35	168	43.1	2.288	33	1
5	116	74	0	0	25.6	0.201	30	0
3	78	50	32	88	31	0.248	26	1
10	115	0	0	0	35.3	0.134	29	0
2	197	70	45	543	30.5	0.158	53	1
8	125	96	0	0	0	0.232	54	1

CODE EXPLANATION:

Importing Required Packages:

⇒ First, we import the required libraries for the code to run effectively.

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split as TTS
from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler as SS
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import zscore
from sklearn.svm import SVC
from sklearn.preprocessing import MinMaxScaler as MMS
from sklearn.preprocessing import LabelEncoder as LE
from sklearn.metrics import accuracy_score as a_s, precision_score a
s p_s, confusion_matrix as c_m, classification_report as c_r
from sklearn.linear_model import LogisticRegression as LR
from sklearn.neighbors import KNeighborsClassifier as KNN
from sklearn.model_selection import cross_val_score as CVS
from sklearn.metrics import roc_auc_score as RAS
from sklearn.naive_bayes import GaussianNB as NB
from sklearn import tree
from sklearn.ensemble import RandomForestClassifier as RFC
import warnings
warnings.filterwarnings("ignore")
```

Data Pre – processing:

⇒ The Diabetes dataset is imported.
⇒ A dictionary is created for the Outcome so that it will be helpful to display the desired output.

```
data = pd.read_csv("/content/Diabetes Database.csv")
pred = dict([(1, 'has Diabetes'), (0, 'is Free from Diabetes')])
pred
```

⇒ To know about the dataset's format, we can use the following code lines:

```
data.describe()
data.info()
```

⇒ To visualize all attributes, we can use histogram:

```
data.hist(figsize=(10,10))
plt.show()
```

⇒ Z Score can be used to detect outliers in the dataset.

```
z_scores = zscore(data.iloc[:,:-1])
abs_z_scores = np.abs(z_scores)
filtered_entries = (abs_z_scores < 3).all(axis=1)
data_outlir = data[filtered_entries]
data_outlir.shape
data_outlir['Outcome'].value_counts()
```

⇒ Feature Scaling:

```
scaler = MMS()
data_std = scaler.fit_transform(data_outlir.iloc[:,:-1])
data_std = pd.DataFrame(data_std)
data_std.shape
```

Exploratory data analysis (EDA):

EDA involves exploring the datasets by means of visualizations. With the help of EDA, we can able to identify the methods that can be used for further analysis and classification.

```
data_va = data_std.var(axis= 0)
data_vas = data_va.sort_values(ascending=False)
y = data_vas.values
x = range(len(y))
```

Checking for Mutli – collinearity:

```
plt.figure(figsize = (5,5))
plt.plot(x, y)
plt.title("MULTI - COLLINEARITY CHECK")
plt.xlabel("ATTRIBUTES")
plt.ylabel("VARIANCES")
plt.show()
```

Splitting Data into Train and Test sets:

```
X = data_outlir.iloc[:, :-1].values
Y = data_outlir['Outcome'].values

X_train, X_test, y_train, y_test = TTS(X, Y, test_size=0.33, random_sta
te=42)
```

To Check for Suitable ML Models:

We try to find a suitable model by comparing the accuracies of the following models:

1. Logistic Regression

```
lr = LR(C = 0.2)
clf1 = lr.fit(X_train, y_train)
```

2. Naïve – Bayes

```
nb = NB()
clf2 = nb.fit(X_train, y_train)
```

3. Decision Tree

```
dt = tree.DecisionTreeClassifier(criterion='entropy')
clf3 = dt.fit(X_train, y_train)
```

4. Random Forest

```
rf = RFC(max_depth=5, random_state=0, n_estimators=100)
clf4 = rf.fit(X_train, y_train)
```

5. Support Vector Machine (SVM)

```
svm = SVC(probability=True)
clf5 = svm.fit(X_train, y_train)
```

6. KNN

```
knn = KNN(n_neighbors = 3, metric = 'euclidean', p = 2)
clf6 = knn.fit(X_train, y_train)
```

ROC – AUC Curve:

AUC - ROC curve is a performance measurement for the classification problems at various threshold settings. ROC is a probability curve and AUC represents the degree or measure of separability. It tells how much the model is capable of distinguishing between classes.

```
pred_prob1 = clf1.predict_proba(X_test)
pred_prob2 = clf2.predict_proba(X_test)
pred_prob3 = clf3.predict_proba(X_test)
pred_prob4 = clf4.predict_proba(X_test)
pred_prob5 = clf5.predict_proba(X_test)
pred_prob6 = clf6.predict_proba(X_test)

from sklearn.metrics import roc_curve

fpr1, tpr1, thresh1 = roc_curve(y_test, pred_prob1[:,1], pos_label=1)
fpr2, tpr2, thresh2 = roc_curve(y_test, pred_prob2[:,1], pos_label=1)
fpr3, tpr3, thresh3 = roc_curve(y_test, pred_prob3[:,1], pos_label=1)
fpr4, tpr4, thresh4 = roc_curve(y_test, pred_prob4[:,1], pos_label=1)
fpr5, tpr5, thresh5 = roc_curve(y_test, pred_prob5[:,1], pos_label=1)
fpr6, tpr6, thresh6 = roc_curve(y_test, pred_prob6[:,1], pos_label=1)

random_probs = [0 for i in range(len(y_test))]
p_fpr, p_tpr, _ = roc_curve(y_test, random_probs, pos_label=1)

import matplotlib.pyplot as plt
plt.style.use('seaborn')
plt.figure(figsize = (9, 6))

plt.plot(fpr1, tpr1, linestyle='--',color='orange', label='LR')
plt.plot(fpr2, tpr2, linestyle='--',color='green', label='NB')
plt.plot(fpr3, tpr3, linestyle='--',color='red', label='DT')
plt.plot(fpr4, tpr4, linestyle='--',color='purple', label='RF')
plt.plot(fpr5, tpr5, linestyle='--',color='black', label='SVM')
plt.plot(fpr6, tpr6, linestyle='--',color='grey', label='KNN')
plt.plot(p_fpr, p_tpr, linestyle='--', color='blue')

plt.title('ROC - AUC Curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive rate')
plt.legend(loc = 'lower right')
plt.savefig('ROC')
plt.show()
```

```
from sklearn import model_selection
models = []
models.append(('LR', lr))
models.append(('NB', nb))
models.append(('DT', dt))
models.append(('RF', rf))
models.append(('SVM', svm))
models.append(('KNN', knn))

results = []
names = []
scoring = 'accuracy'
for name, model in models:
  kfold = model_selection.KFold(n_splits=5)
  cv_results = model_selection.cross_val_score(model, X_train, y_train,
 cv=kfold, scoring=scoring)
  results.append(cv_results)
  names.append(name)
  msg = "%s   : %.3f (%.3f)" % (name, cv_results.mean(), cv_results.std
())
  print(msg)
fig = plt.figure()
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
plt.boxplot(results)
plt.xlabel("CLASSIFICATION MODEL")
plt.ylabel("CROSS VALIDATION SCORES")
ax.set_xticklabels(names)
plt.show()
```

From the results obtained, we infer that:

“BEST MODEL: RANDOM FOREST”

Random Forest Classifier Function:

About RF:

The random forest is a classification algorithm consisting of many decision trees. It uses bagging and feature randomness when building each individual tree to try to create an uncorrelated forest of trees whose prediction by committee is more accurate than that of any individual tree.

```
import numpy as np
from collections import Counter
def entropy(y):
    hist = np.bincount(y)
    ps = hist / len(y)
```

```
    return -np.sum([p * np.log2(p) for p in ps if p > 0])
class Node:
    def __init__(self, feature=None, threshold=None, left=None, right =
                                                    None, *, value=None):
        self.feature = feature
        self.threshold = threshold
        self.left = left
        self.right = right
        self.value = value
    def is_leaf_node(self):
        return self.value is not None

class DecisionTree:
    def __init__(self, min_samples_split=2,max_depth=100,n_feats=None):
        self.min_samples_split = min_samples_split
        self.max_depth = max_depth
        self.n_feats = n_feats
        self.root = None
    def fit(self, X, y):
        self.n_feats = X.shape[1] if not self.n_feats else
                                          min(self.n_feats, X.shape[1])
        self.root = self._grow_tree(X, y)
    def predict(self, X):
        return np.array([self._traverse_tree(x, self.root) for x in X])
    def _grow_tree(self, X, y, depth=0):
        n_samples, n_features = X.shape
        n_labels = len(np.unique(y))
        if (depth >= self.max_depth
                or n_labels == 1
                or n_samples < self.min_samples_split):
            leaf_value = self._most_common_label(y)
            return Node(value=leaf_value)
        feat_idxs = np.random.choice(n_features, self.n_feats,
                                                    replace = False)
        best_feat, best_thresh = self._best_criteria(X, y, feat_idxs)
        left_idxs, right_idxs = self._split(X[:, best_feat],
                                                    best_thresh)
        left = self._grow_tree(X[left_idxs, :], y[left_idxs], depth+1)
        right = self._grow_tree(X[right_idxs, :],y[right_idxs],depth+1)
        return Node(best_feat, best_thresh, left, right)
    def _best_criteria(self, X, y, feat_idxs):
        best_gain = -1
        split_idx, split_thresh = None, None
        for feat_idx in feat_idxs:
            X_column = X[:, feat_idx]
            thresholds = np.unique(X_column)
            for threshold in thresholds:
                gain = self._information_gain(y, X_column, threshold)
```

```
                if gain > best_gain:
                    best_gain = gain
                    split_idx = feat_idx
                    split_thresh = threshold
        return split_idx, split_thresh
    def _information_gain(self, y, X_column, split_thresh):
        parent_entropy = entropy(y)
        left_idxs, right_idxs = self._split(X_column, split_thresh)
        if len(left_idxs) == 0 or len(right_idxs) == 0:
            return 0
        n = len(y)
        n_l, n_r = len(left_idxs), len(right_idxs)
        e_l, e_r = entropy(y[left_idxs]), entropy(y[right_idxs])
        child_entropy = (n_l / n) * e_l + (n_r / n) * e_r
        ig = parent_entropy - child_entropy
        return ig
    def _split(self, X_column, split_thresh):
        left_idxs = np.argwhere(X_column <= split_thresh).flatten()
        right_idxs = np.argwhere(X_column > split_thresh).flatten()
        return left_idxs, right_idxs
    def _traverse_tree(self, x, node):
        if node.is_leaf_node():
            return node.value
        if x[node.feature] <= node.threshold:
            return self._traverse_tree(x, node.left)
        return self._traverse_tree(x, node.right)
    def _most_common_label(self, y):
        counter = Counter(y)
        most_common = counter.most_common(1)[0][0]
        return most_common

def bootstrap_sample(X, y):
    n_samples = X.shape[0]
    idxs = np.random.choice(n_samples, n_samples, replace=True)
    return X[idxs], y[idxs]

def most_common_label(y):
    counter = Counter(y)
    most_common = counter.most_common(1)[0][0]
    return most_common

class RandomForest:
    def __init__(self,n_trees=10,min_samples_split=2,max_depth=100,
                                                    n_feats = None):
        self.n_trees = n_trees
        self.min_samples_split = min_samples_split
        self.max_depth = max_depth
        self.n_feats = n_feats
```

```
        self.trees = []
    def fit(self, X, y):
        self.trees = []
        for _ in range(self.n_trees):
            tree=DecisionTree(min_samples_split=self.min_samples_split,
                 max_depth = self.max_depth, n_feats=self.n_feats)
            X_samp, y_samp = bootstrap_sample(X, y)
            tree.fit(X_samp, y_samp)
            self.trees.append(tree)
    def predict(self, X):
        tree_preds = np.array([tree.predict(X) for tree in self.trees])
        tree_preds = np.swapaxes(tree_preds, 0, 1)
        y_pred=[most_common_label(tree_pred) for tree_pred in
                                                        tree_preds]
        return np.array(y_pred)
```

Prediction using the Classifier:

The classifier is fit to the train set which then will be implemented in the test set prediction.

```
import pandas as pd
from sklearn import datasets
from sklearn.model_selection import train_test_split

def accuracy(y_true, y_pred):
    accuracy = np.sum(y_true == y_pred) / len(y_true)
    return accuracy

clf = RandomForest(n_trees=50, max_depth=5)

clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
acc = accuracy(y_test, y_pred)

print ("Accuracy:", acc)
```

Classification Report:

A Classification report is used to measure the quality of predictions from a classification algorithm.

The classification report displays the following:

⇒ Precision (percent of predictions that were correct)
⇒ Recall (percent of positive instances found)
⇒ F1 (percent of positive predictions that were correct)
⇒ Support scores (number of actual occurrences of the class in the specified dataset)

```
print('CLASSIFICATION REPORT\n\n', c_r(y_test, y_pred))
```

Confusion Matrix:

The Confusion Matrix is used to describe the classification model performance on a test dataset for which the true values are known.

```
cm1 = c_m(y_test, y_pred)
print("CONFUSION MATRIX : \n\n",cm1)
```

Test Cases:

The test cases can be used to verify the performance of the classifier on new data.

```
test_case_1 = clf.predict([[6, 148, 72, 35, 0, 33.6, 0.627, 50]])
print('The Patient', ''.join(map(str, pred[test_case_1[0]])))
```

```
test_case_2 = clf.predict([[1, 85, 66, 29, 0, 26.6, 0.351, 31, 0]])
print('The Patient', ''.join(map(str, pred[test_case_2[0]])))
```

OUTPUT AND INFERENCE:

⇒ The dictionary created for each fruit_name using fruit_label respectively.

```
{0: 'is Free from Diabetes', 1: 'has Diabetes'}
```

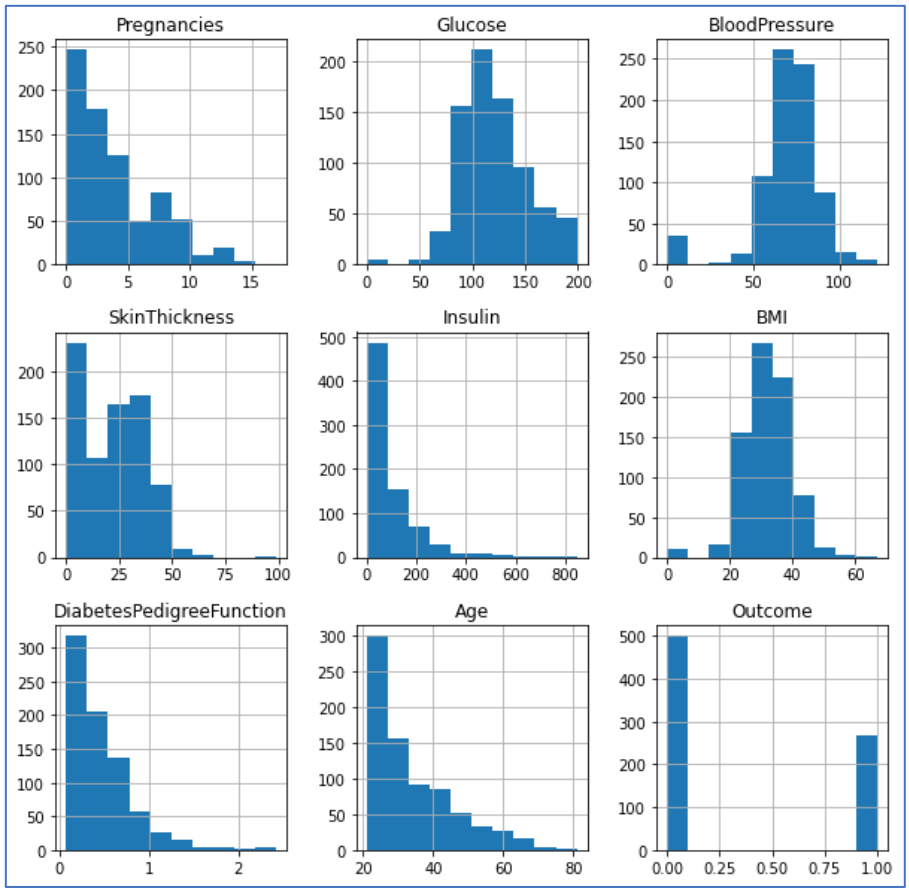
⇒ Results of data.describe() give details about the summary statistics like mean, median, standard deviation, quantile values, maximum and minimum values and count of each attribute.

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.348958
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

⇒ Results of data.info() tell us that the data has 9 attributes, all free of null values and of datatypes as shown below:

```
<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
```

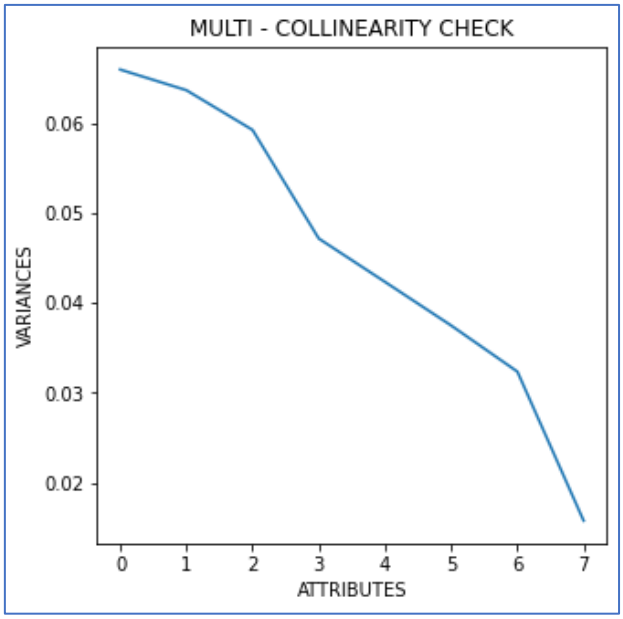
⇒ The Histogram plots shows how each attribute is distributed, range of attributes.



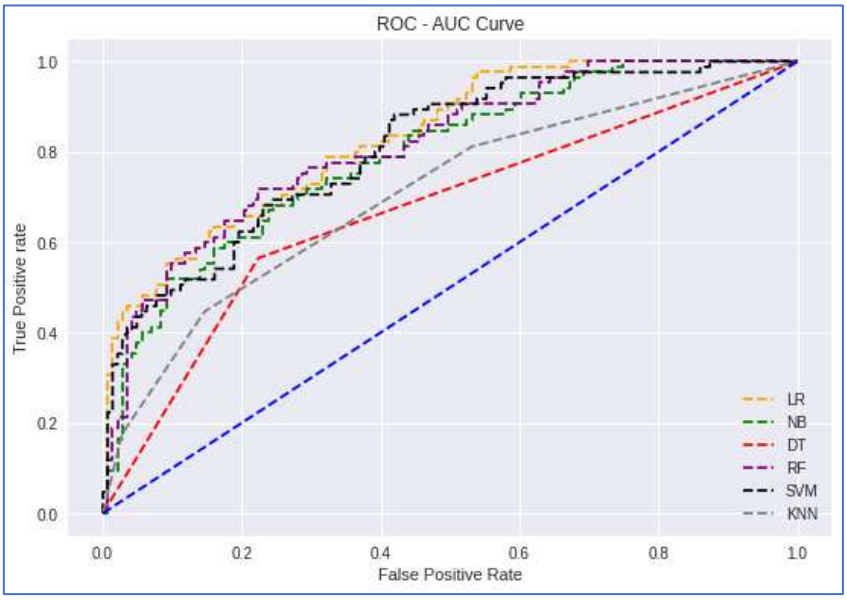
⇒ The below output represents the filtered data count after removing the outliers.

```
0    461
1    227
Name: Outcome, dtype: int64
```

⇒ From the below graph, it is clear that there is no strong collinearity among the attributes and so all the attributes can be used.



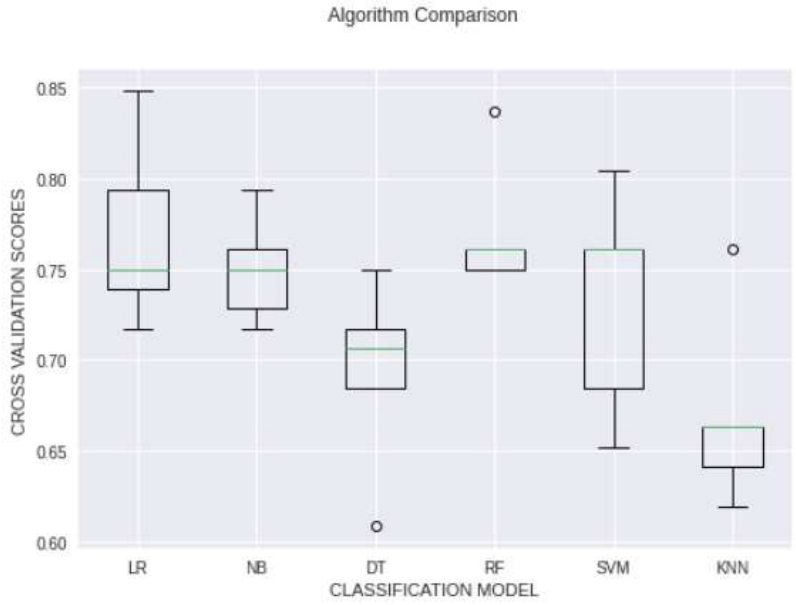
⇒ The resulted ROC – AUC Curve is shown below:



⇒ By comparing the accuracies of all the classifier functions, it's clear that the most suitable model for the diabetes prediction is **'Random Forest'** classification.

Hence, a Random forest model from scratch is developed and the prediction results are verified.

```
LR   : 0.770 (0.046)
NB   : 0.750 (0.027)
DT   : 0.693 (0.047)
RF   : 0.772 (0.033)
SVM  : 0.733 (0.056)
KNN  : 0.670 (0.048)
```



Random Forest Classifier Accuracy: `0.7456140350877193`

Classification Report Results:

- ∅ Accuracy for the model is approximately 75%, meaning it can give good results for the values of the dataset already present, but the model can't adapt and accurately predict for new values.
- ∅ Precision Score for class 0 is 79% and that of class 1 is 67%.
- ∅ Recall value for class 0 is 81% and that of class 1 is 64%.
- ∅ F1 score accuracy is 75%, which is used to compare with other classifier models.

```
CLASSIFICATION REPORT

              precision    recall  f1-score   support

           0       0.79      0.81      0.80       143
           1       0.67      0.64      0.65        85

    accuracy                           0.75       228
   macro avg       0.73      0.72      0.73       228
weighted avg       0.74      0.75      0.74       228
```

⇒ Confusion Matrix:

From the below result, it is clear that the model predicted:

- True Positive : 116
- True Negative : 54
- False Positive : 27
- False Negative : 31

```
CONFUSION MATRIX :

 [[116  27]
 [ 31  54]]
```

⇒ Prediction of New Results (Test Cases):

The following are the results for the test cases:

```
test_case_1 = clf.predict([[6, 148, 72, 35, 0, 33.6, 0.627, 50]])

print('The Patient', ''.join(map(str, pred[test_case_1[0]])))
```

```
The Patient has Diabetes
```

```
test_case_2 = clf.predict([[1, 85, 66, 29, 0, 26.6, 0.351, 31, 0]])

print('The Patient', ''.join(map(str, pred[test_case_2[0]])))
```

```
The Patient is Free from Diabetes
```

CONCLUSION:

Though this model can predict accurately to some extent, still it is not a perfect model that will be suitable to predict any test cases/new values. But, if nearby values for the attributes are given as new values, then this model is suitable for prediction.