Random Forest Classifier

→ This classifier produces a forest of trees where a random selection of characteristics from the total characteristics shapes each tree. The number of trees used to estimate the class can be varied. We are going to consider 1 to 500 to calculate the test scores.

rfc = RandomForestClassifier(n_estimators=500, criterion = 'entropy')

→ Creating a model RandomForestClassifier whose parameters are n_estimators and criterion.

→ n_estimators: The number of trees in the forest. Here we have taken 500 trees in the forest.

→ criterion: The function to measure the quality of a split. "entropy" for the information gain.

rfc.fit(x_train,y_train)

→ Build a forest of trees from the training set (x_train,y_train). (training a model)

rf_score = rfc.score(x_test,y_test)

→ Return the mean accuracy on the given test data and labels. (prediction of score)

print("The accuracy score achieved using Random Forest Classifier is: "+str(rfc.score(x_test,y_test)))

→ Print the accuracy score achieved using Random Forest Classifier

```
The accuracy score achieved using Random Forest Classifier is:
0.7106666666666667
```

print(dict(zip(df.columns,rfc.feature_importances_)))

→ rfc.feature_importances_: use of forests of trees to evaluate the importance of features (i.e. age, height, weight, ap_hi, ap_lo, smoke, alco, active, cardio, male, female, normal_cholesterol, above_high_cholesterol, high_cholesterol, normal_glucose, high_glucose) on an artificial classification task i.e., it show the importance of each features used in Random Forest Classifier.

→ The result will be show in dictionary form.

```
{'age': 0.32558352558347714, 'height': 0.17129304402214626, 'we
ight': 0.1897235791333796, 'ap_hi': 0.1439028957999641, 'ap_lo'
: 0.07895122039322883, 'smoke': 0.008914948584239973, 'alco': 0
.007953233175482893, 'active': 0.013673052876168589, 'cardio':
0.0072152484693539486, 'male': 0.007137714301128996, 'female':
0.012462824037261835, 'normal_cholesterol': 0.00477384164080203
5, 'above_high_cholesterol': 0.012654165453139191, 'high_choles
terol': 0.006493181917566632, 'normal_glucose': 0.004777762051
36239, 'high_glucose': 0.004489748407523682}
```

K Nearest neighbor Classifier

→ This classifier considers the nearest K neighbors' classes of a specified data point and is assigns a class to this data point based on the majority class. The number of neighbors may vary. We are going to vary them from 1 to 30 neighbors to calculate the test score. Then, we are going to plot a line graph of the number of neighbors and the test score in each case.

```
kn_scores = []
```
→ Taking an array where all the test score will be store in the form of array.
```
k_range = range(1, 31)
```
→ We are taking 1 to 30 neighbors to calculate the test score
```
for k in k_range:
```
→ loop is used here to find test score for 1 to 30 neighbors
```
    knc = KNeighborsClassifier(n_neighbors=k)
```
→ creating a model KNeighborsClassifier whose parameters is n_neighbors.

→ n_neighbors: Number of neighbors
```
    knc.fit(x_train,y_train)
```
→ fit the k-nearest neighbors classifier from the training dataset. (training a model)
```
    kn_scores.append(knc.score(x_test,y_test))
```
→ score() is use to return the mean accuracy on the given test data and labels. (prediction of score)

→ append() is use to add test scores in the array
```
print(dict(zip(k_range,kn_scores)))
```
→ Print all the test score for 1 to 30 neighbors in the form of dictionary.

```
{1: 0.6260289855072464, 2: 0.6326376811594203, 3: 0.6598840579
710145, 4: 0.664, 5: 0.6793623188405797, 6: 0.6846376811594203
, 7: 0.692, 8: 0.691304347826087, 9: 0.6958840579710145, 10: 0
.6942028985507246, 11: 0.6980869565217391, 12: 0.7006376811594
203, 13: 0.6996521739130435, 14: 0.7009855072463768, 15: 0.703
9420289855073, 16: 0.704231884057971, 17: 0.7058550724637681,
18: 0.7055072463768116, 19: 0.705391304347826, 20: 0.704811594
2028985, 21: 0.7056811594202899, 22: 0.7053333333333334, 23: 0
.7049275362318841, 24: 0.7066086956521739, 25: 0.7073623188405
798, 26: 0.7066666666666667, 27: 0.707536231884058, 28: 0.7052
173913043478, 29: 0.7066666666666667, 30: 0.7048115942028985}
```

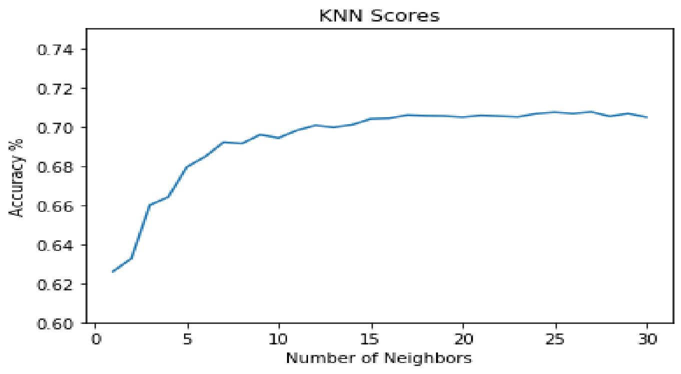
```
# Accuracy of different KNN Classifiers

plt.plot(k_range, kn_scores)
```
→ Plotting the graph of number of neighbours vs. test scores.
```
plt.ylim(0.6,0.75)
```
→ used to get or set the y-limits of the current axes.
```
plt.title('KNN Scores')
```
→ title of the graph

plt.xlabel('Number of Neighbors')

→ label in x-axis

plt.ylabel('Accuracy %')

→ label in y-axis



Logistic Regression

→ Logistic regression is a type of statistical regression analysis used to predict the outcome of a categorical dependent variable from a set of predictors or independent variables. The dependent variable is always discreet in logistic regression. Logistic regression is used to predict and calculate the likelihood of success.

lr = LogisticRegression(penalty= 'l1',solver= 'liblinear')

→ Creating a model LogisticRegression whose parameters are penality and solver.

→ penalty- this parameter is used to specify the norm used in the penalization (regularization).

→ solver- this parameter represents which algorithm to use in the optimization problem. Liblinear is a good for small database. It handles L1 penality.

lr.fit(x_train,y_train)

→ fit the model according to the given training data. (training a model)

lr_score = accuracy_score(y_test,lr.predict(x_test))

→ predict(): Predict class labels for samples in x_test

→ score(): Return the mean accuracy on the given test data and labels. (prediction of score)

print("The accuracy score achieved using Logistic Regression is: "+str(accuracy_score(y_test,lr.predict(x_test))))

→ Print the accuracy score achieved using Logistic Regression

```
The accuracy score achieved using Logistic Regression is: 0.72
67246376811595
```

Naive Bayes

→ The technique of the Naïve Bayes Classifier applies especially when the input dimensionality is high. Naive Bayes can sometimes outperform more advanced methods of classification, amid its simplicity. Naïve Bayes model identifies the features of heart disease patients. This displays the likelihood for the predictable state of each input attribute.
→ Here we are using Gaussian Naïve Bayes as we are taking categorical as well as continues data as well.

```
gnb = GaussianNB()
```
→ Making a model of GaussianNB
```
gnb.fit(x_train,y_train)
```
→ Fit Gaussian Naive Bayes according to x_test, y_test (training a model)
```
gnb_score = gnb.score(x_test, y_test)
```
→ Return the mean accuracy on the given test data and labels. (prediction of score)
```
print ("The accuracy score achieved using Naive Bayes is: "+str(gnb.score(x_test, y_test)))
```
→ Print the accuracy score achieved using Naïve Bayes.

```
The accuracy score achieved using Naive Bayes is: 0.6627246376
811594
```

Voting Classifier

→ Voting is one of the easiest ways to combine the predictions of several algorithms in machine learning. Voting classifier is not an actual classifier, but a wrapper for a set of different classifiers that are trained and rated in parallel to exploit that algorithm's different characteristics.

```
knn = KNeighborsClassifier(n_neighbors=26)
```
→ Creating a model KNeighborsClassifier
```
lr = LogisticRegression(penalty='l1',solver= 'liblinear')
```
→ Creating a model LogisticRegression
```
rfc = RandomForestClassifier(n_estimators=500, criterion = 'entropy')
```
→ Creating a model VotingClassifiers
```
vc = VotingClassifier(estimators=[('knn',knn),('lr',lr),('rfc',rfc)], voting='soft')
```
→ Creating a model VotingClassifiers whose parameters are estimators and voting.
→ Estimators: estimators is an equation for picking the best or most accurate, data model based upon there observation. [Estimators: list of (str, estimator) tuples]. Here we are taking K-neighbors classifier, Logistic Regression, Random Forest Classifier.

→ Voting: “soft” in soft voting the output class is the prediction based on the average of probability given to that class.

```
vc.fit(x_train,y_train)
```

→ Fit the estimators. (training a model)

```
vc_score = vc.score(x_test, y_test)
```

→ Return the mean accuracy on the given test data and labels. (prediction of score)

```
print ("The accuracy score achieved using Voting Classifier is: "+str(vc.score(x_test, y_test)))
```

→ Print the accuracy score achieved using Voting Classifier

```
The accuracy score achieved using Voting Classifier is: 0.72834
78260869565
```