RAMAIAH UNIVERSITY OF APPLIED SCIENCES

ASSIGNMENT

Subject Code	CSE402A
Subject Name	Data Mining
Programme/Course	B. Tech
Department	CSE
Faculty	FET

Name of the Student	Ashish Kumar
Reg. No	16ETCS002144
Semester/Year	7th sem/ 4th year
Subject Leader/s	Prof. N. D. Gangadhar

Ramaiah University of Applied Sciences

University House, Gnanagangothri Campus, New BEL Road,
M S R Nagar, Bangalore, Karnataka, INDIA - 560 054.

Declaration Sheet			
Student Name	Ashish Kumar		
Reg. No	16ETCS002144		
Programme/Course	B.Tech	Semester/Year	7th sem/ 4th year
Subject Code	CSE402A		
Subject Title	Data Mining		
Subject Date		to	
Subject Leader	Prof. N. D. Gangadhar		

Declaration

The assignment submitted herewith is a result of my own investigations and that I have conformed to the guidelines against plagiarism as laid out in the Student Handbook. All sections of the text and results, which have been obtained from other sources, are fully referenced. I understand that cheating and plagiarism constitute a breach of University regulations and will be dealt with accordingly.

Signature of the Student		Date	
Submission date stamp (by Examination & Assessment Section)			

Signature of the Subject Leader and date	Signature of the Reviewer and date

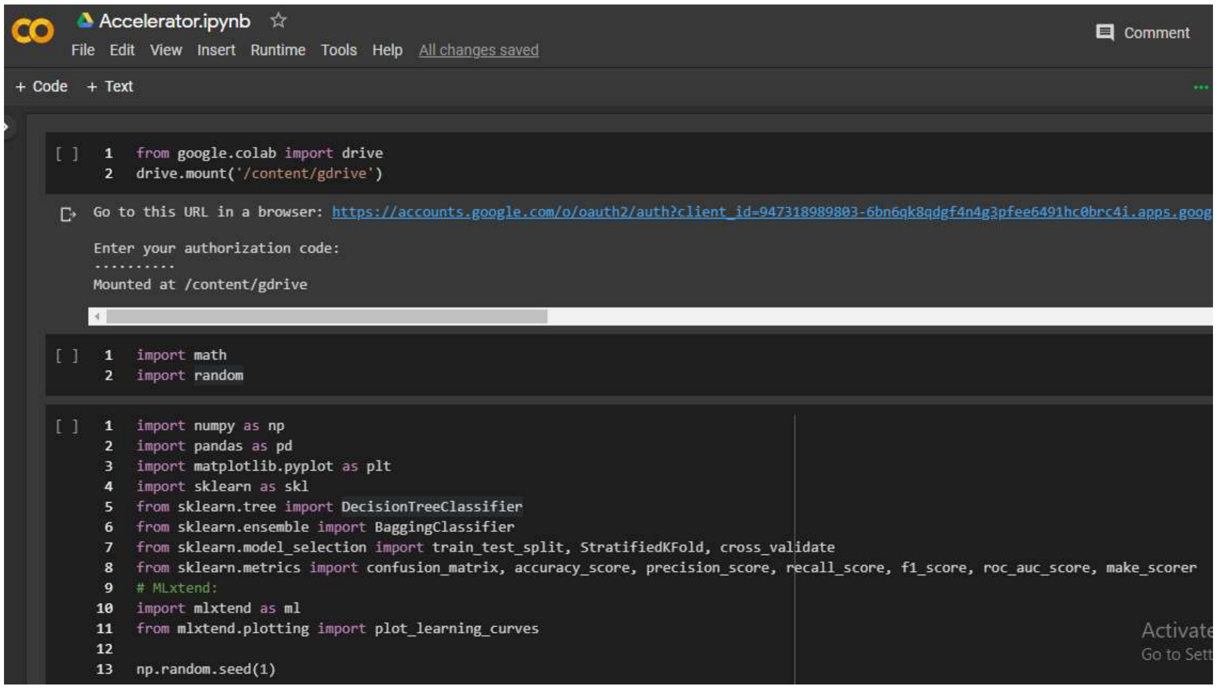
Solution to Question No. 1:

1.1 Introduction:

Classification is a method of identifying an object's class by learning from a set of known data and their classes, and use that information to classify the class of object whose class is unknown. In a particle accelerator we try to detect presence of fundamental particles like Higgs. To detect these particle, we can use classifier with two labels, particle is present and particle is not present. The presence of a particle depends on the traces left by the experiment. The attributes for classification consists of eight kinematic values measured by the particle detectors and ten derived values from these eight values. We can use different classifier algorithms to classify the above data like SVM, Neural Network and Decision tree etc. In this assignment we will use decision tree as base classifier and random ensemble decision tree because SVM is very complex classifier and neural network has very high computation expense so for such huge data set like in this problem neural network is also not helpful. Decision tree are the best choice as they are very good for tabular data and are simpler to implement and computationally simple too hence we use decision tree classifier. Ensemble of decision tree is used to improve the performance of the algorithm by selecting multiple decision tree to work at a time hence the confidence of the ensemble with be higher compared to single decision tree and will help take more accurate decisions.

1.2 Decision Tree Classifier:

The decision tree classifier implementation:



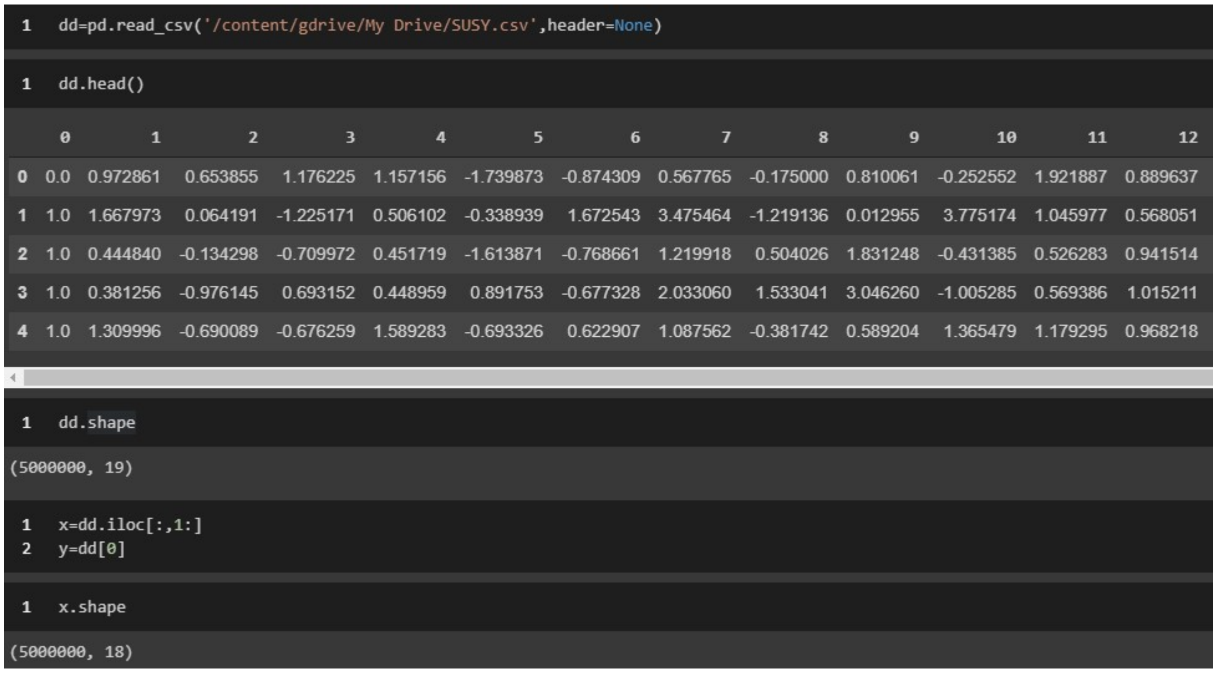
```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.goog

Enter your authorization code:
··········
Mounted at /content/gdrive
```

```
import math
import random
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import sklearn as skl
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.model_selection import train_test_split, StratifiedKFold, cross_validate
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score, f1_score, roc_auc_score, make_scorer
# MLxtend:
import mlxtend as ml
from mlxtend.plotting import plot_learning_curves

np.random.seed(1)
```

The above figure shows the mounting of the google drive to the google colaboratory notebook and the necessary libraries imported.



```
dd=pd.read_csv('/content/gdrive/My Drive/SUSY.csv',header=None)
```

```
dd.head()
```

```
     0         1         2         3         4         5         6         7         8         9        10        11        12
0  0.0  0.972861  0.653855  1.176225  1.157156 -1.739873 -0.874309  0.567765 -0.175000  0.810061 -0.252552  1.921887  0.889637
1  1.0  1.667973  0.064191 -1.225171  0.506102 -0.338939  1.672543  3.475464 -1.219136  0.012955  3.775174  1.045977  0.568051
2  1.0  0.444840 -0.134298 -0.709972  0.451719 -1.613871 -0.768661  1.219918  0.504026  1.831248 -0.431385  0.526283  0.941514
3  1.0  0.381256 -0.976145  0.693152  0.448959  0.891753 -0.677328  2.033060  1.533041  3.046260 -1.005285  0.569386  1.015211
4  1.0  1.309996 -0.690089 -0.676259  1.589283 -0.693326  0.622907  1.087562 -0.381742  0.589204  1.365479  1.179295  0.968218
```

```
dd.shape
```

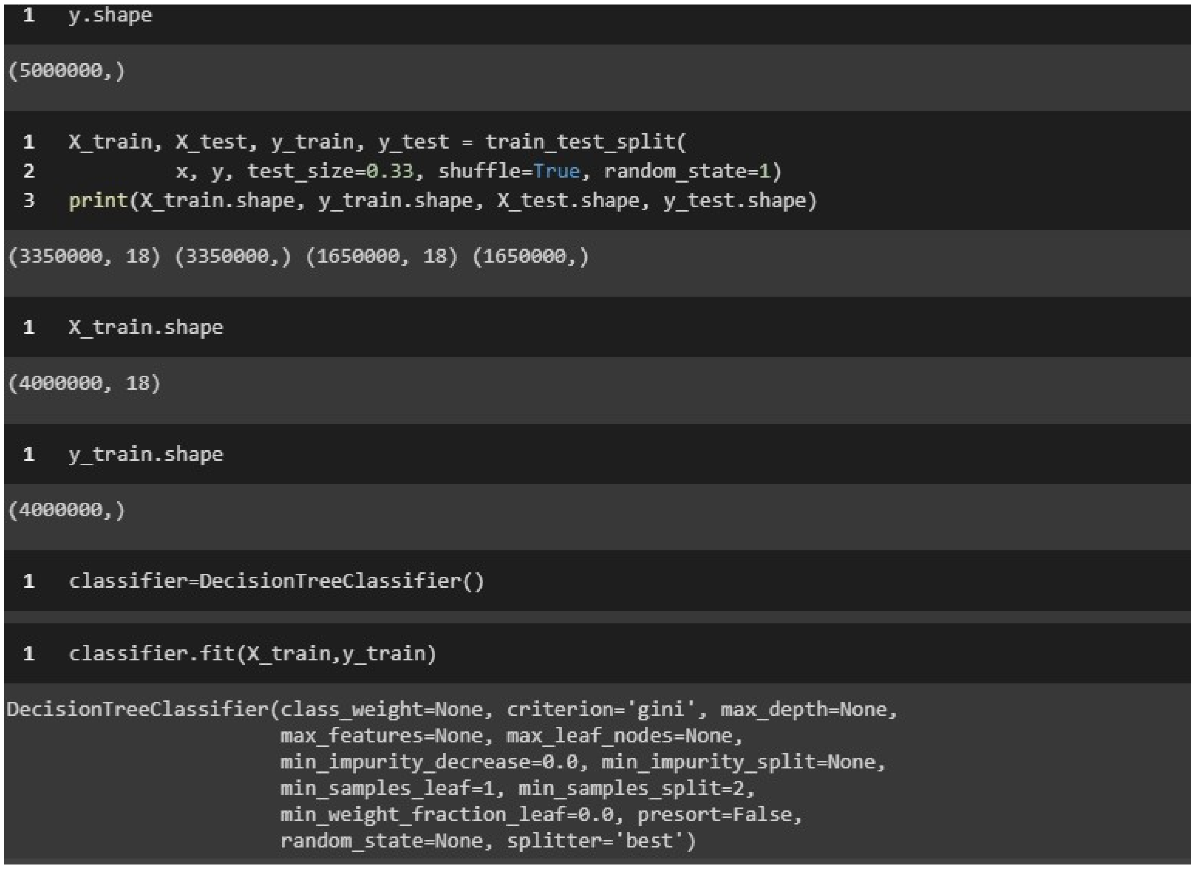
```
(5000000, 19)
```

```
x=dd.iloc[:,1:]
y=dd[0]
```

```
x.shape
```

```
(5000000, 18)
```

The above figure shows the reading of dataset from drive to colab and extracting the data and labels into x and y variables respectively.



```
y.shape
```

```
(5000000,)
```

```
X_train, X_test, y_train, y_test = train_test_split(
          x, y, test_size=0.33, shuffle=True, random_state=1)
print(X_train.shape, y_train.shape, X_test.shape, y_test.shape)
```

```
(3350000, 18) (3350000,) (1650000, 18) (1650000,)
```

```
X_train.shape
```

```
(4000000, 18)
```

```
y_train.shape
```

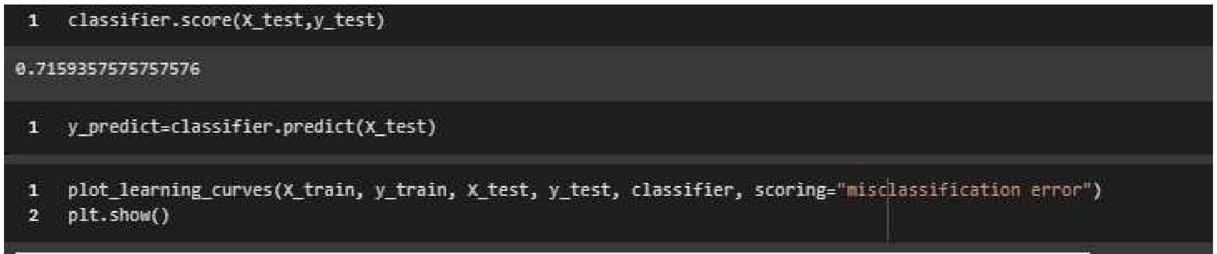
```
(4000000,)
```

```
classifier=DecisionTreeClassifier()
```

```
classifier.fit(X_train,y_train)
```

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=None, splitter='best')
```

The above figure shows the split of random shuffled data into train and test set in a ratio of 2:1 and the decision tree classifier fitting to the data.

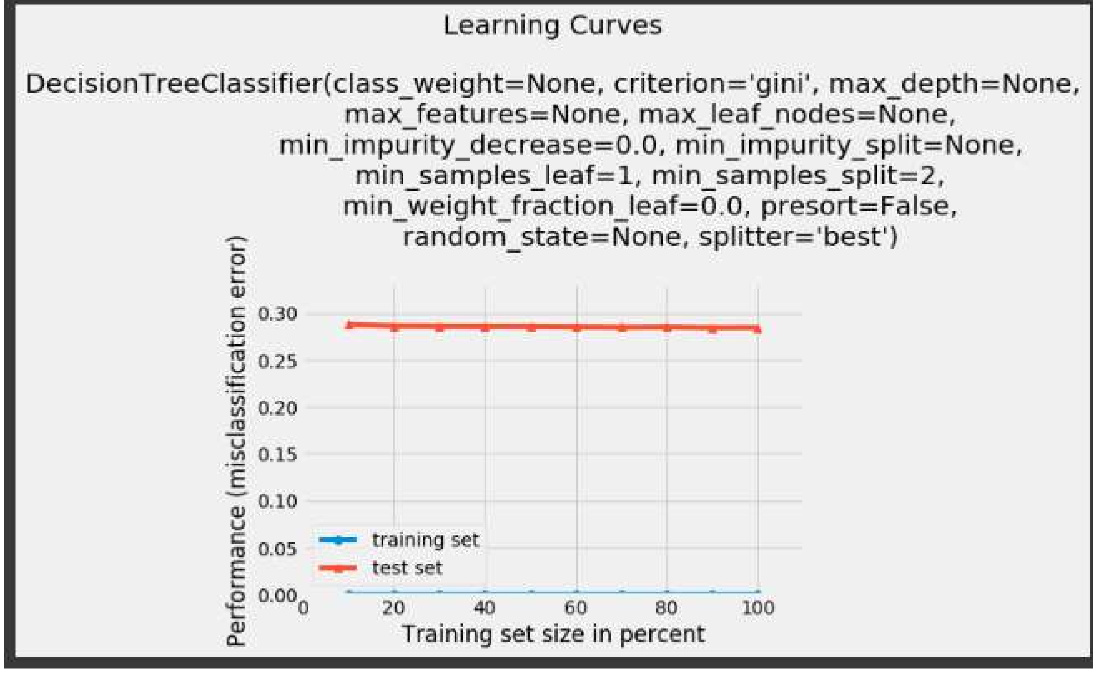


```
classifier.score(X_test,y_test)
```

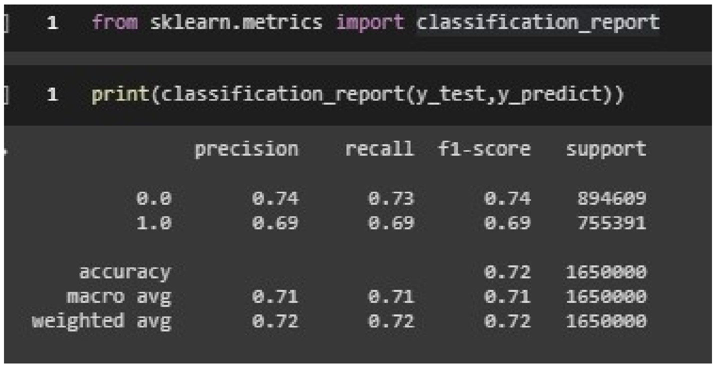
```
0.7159357575757576
```

```
y_predict=classifier.predict(X_test)
```

```
plot_learning_curves(X_train, y_train, X_test, y_test, classifier, scoring="misclassification error")
plt.show()
```



The above figure shows the accuracy score of the classifier (0.7159) and the learning curve. The learning curve can be seen decreasing slowly as the training data is feed to the algorithm, the error being close to less than 0.28.



```
from sklearn.metrics import classification_report
```

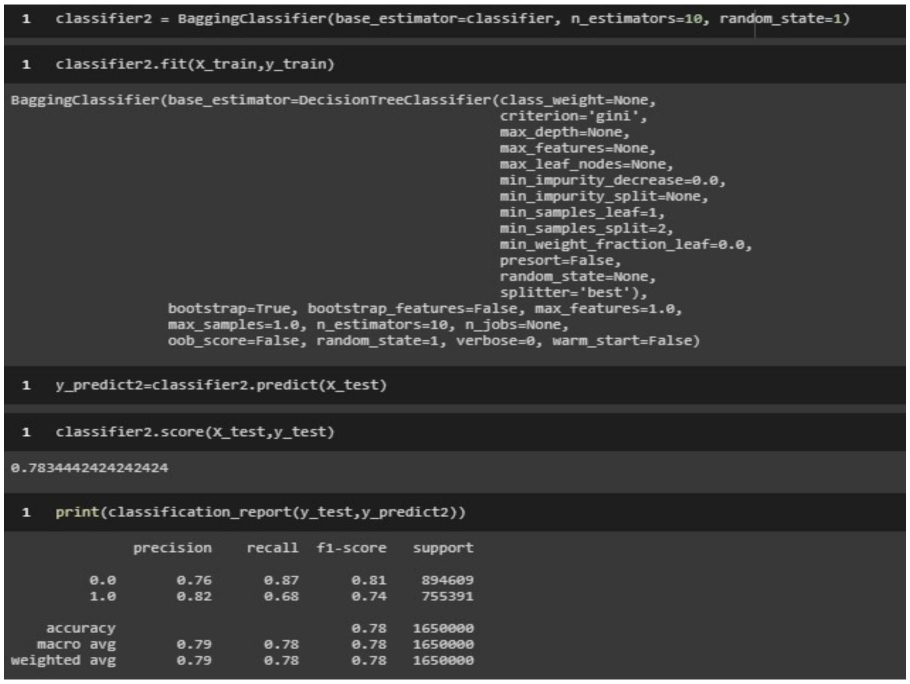
```
print(classification_report(y_test,y_predict))
```

```
              precision    recall  f1-score   support

         0.0       0.74      0.73      0.74    894609
         1.0       0.69      0.69      0.69    755391

    accuracy                           0.72   1650000
   macro avg       0.71      0.71      0.71   1650000
weighted avg       0.72      0.72      0.72   1650000
```

The above figure shows the report of the classifier, the precision, recall, f1-score, support and accuracy. The accuracy of the algorithm can be seen to be 0.72 (OR 72%).

1.3 Ensemble of Decision Tree Classifiers

Ensemble of Decision Tree classifier:



```
classifier2 = BaggingClassifier(base_estimator=classifier, n_estimators=10, random_state=1)
```

```
classifier2.fit(X_train,y_train)
```

```
BaggingClassifier(base_estimator=DecisionTreeClassifier(class_weight=None,
                                                        criterion='gini',
                                                        max_depth=None,
                                                        max_features=None,
                                                        max_leaf_nodes=None,
                                                        min_impurity_decrease=0.0,
                                                        min_impurity_split=None,
                                                        min_samples_leaf=1,
                                                        min_samples_split=2,
                                                        min_weight_fraction_leaf=0.0,
                                                        presort=False,
                                                        random_state=None,
                                                        splitter='best'),
                  bootstrap=True, bootstrap_features=False, max_features=1.0,
                  max_samples=1.0, n_estimators=10, n_jobs=None,
                  oob_score=False, random_state=1, verbose=0, warm_start=False)
```

```
y_predict2=classifier2.predict(X_test)
```

```
classifier2.score(X_test,y_test)
```

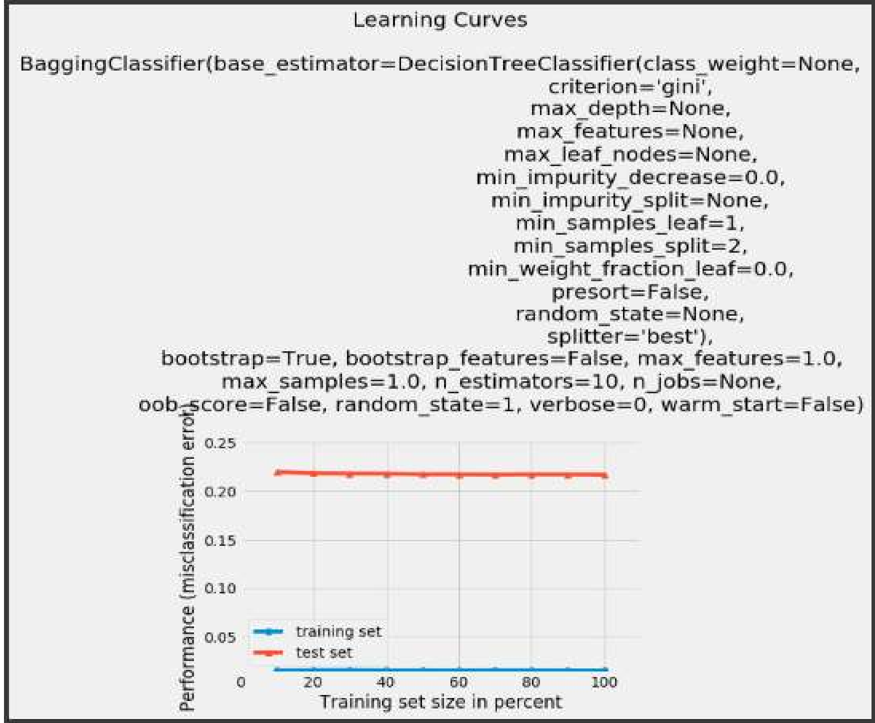
```
0.7834442424242424
```

```
print(classification_report(y_test,y_predict2))
```

```
              precision    recall  f1-score   support

         0.0       0.76      0.87      0.81    894609
         1.0       0.82      0.68      0.74    755391

    accuracy                           0.78   1650000
   macro avg       0.79      0.78      0.78   1650000
weighted avg       0.79      0.78      0.78   1650000
```

The above figure shows the ensemble of decision tree using bagging classifier with base classifier being the decision tree classifier developed above and no of estimator (no of decision trees) used is 10. We can also see the fit of the bagging algorithm and the accuracy score bring 0.78344 which is higher as compared to the 0.7159 that was achieved with a single decision tree. The classification report shows the precision, recall, f1-score, support and accuracy of the classifier.

```
# Learning curves:
plot_learning_curves(X_train, y_train, X_test, y_test, classifier2, scoring="misclassification error")
plt.show()
```



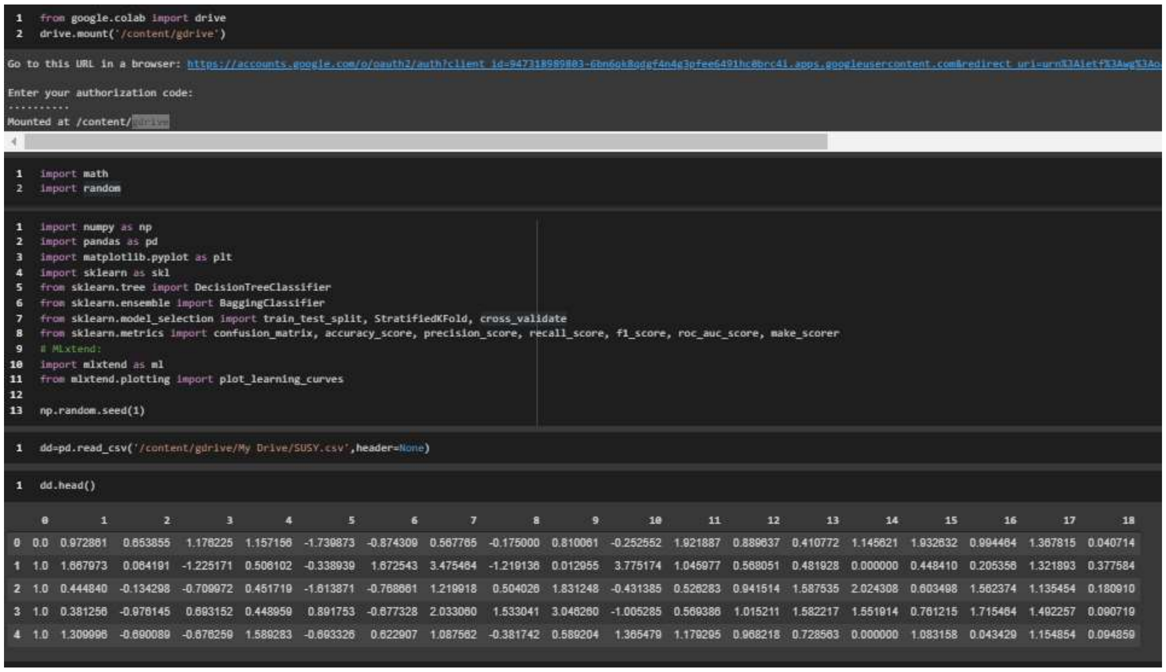
The above figure shows the learning curve of the bagging classifier where we can see the gradual decrease in the error as the no of data feed to the algorithm increases, the error being close to 0.22 which is less compared to 0.28 obtained for the single decision tree.

1.4 Decision Tree Classifier with Selected Data Attributes

Decision tree classifier with selected data attributes

Selected attributes are attribute no: 1,2,3,4,5,6,7,8,9,12,13,14,16,18

Decision tree for the selected 14 attributes:



```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc41.apps.googleusercontent.com&redirect_uri=urn%3Aietf%3Awg%3Ao

Enter your authorization code:
··········
Mounted at /content/gdrive
```

```
import math
import random
```

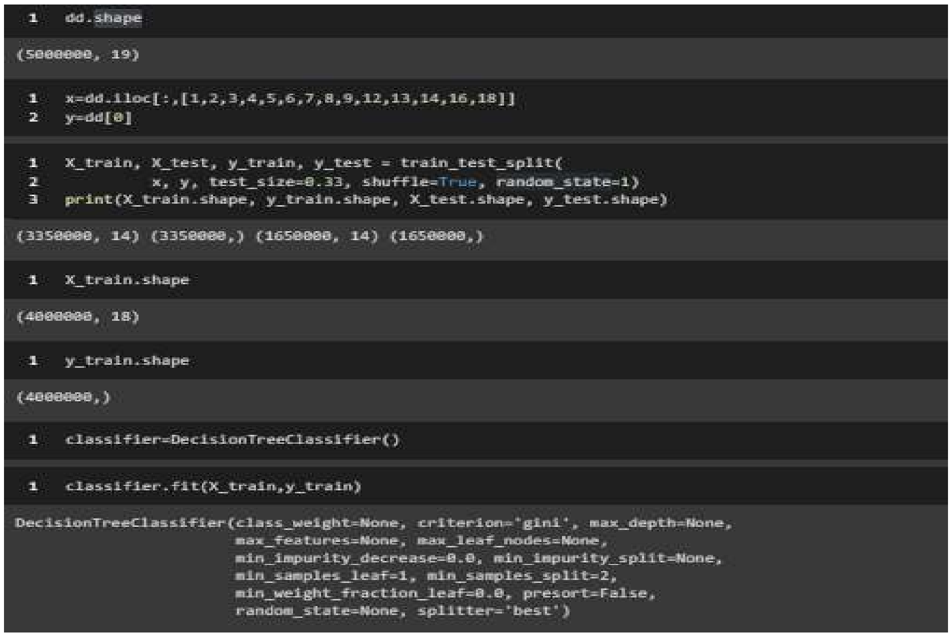
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import sklearn as skl
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.model_selection import train_test_split, StratifiedKFold, cross_validate
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score, f1_score, roc_auc_score, make_scorer
# MLxtend:
import mlxtend as ml
from mlxtend.plotting import plot_learning_curves

np.random.seed(1)
```

```
dd=pd.read_csv('/content/gdrive/My Drive/SUSY.csv',header=None)
```

```
dd.head()
```

```
     0         1         2         3         4         5         6         7         8         9        10        11        12        13        14        15        16        17        18
0  0.0  0.972861  0.653855  1.176225  1.157156 -1.739873 -0.874309  0.567765 -0.175000  0.810061 -0.252552  1.921887  0.889637  0.410772  1.145621  1.932632  0.994464  1.367815  0.040714
1  1.0  1.667973  0.064191 -1.225171  0.506102 -0.338939  1.672543  3.475464 -1.219136  0.012955  3.775174  1.045977  0.568051  0.481928  0.000000  0.448410  0.205356  1.321893  0.377584
2  1.0  0.444840 -0.134298 -0.709972  0.451719 -1.613871 -0.768661  1.219918  0.504026  1.831248 -0.431385  0.526283  0.941514  1.587535  2.024308  0.603498  1.562374  1.135454  0.180910
3  1.0  0.381256 -0.976145  0.693152  0.448959  0.891753 -0.677328  2.033060  1.533041  3.046260 -1.005285  0.569386  1.015211  1.582217  1.551914  0.761215  1.715464  1.492257  0.090719
4  1.0  1.309996 -0.690089 -0.676259  1.589283 -0.693326  0.622907  1.087562 -0.381742  0.589204  1.365479  1.179295  0.968218  0.728563  0.000000  1.083158  0.043429  1.154854  0.094859
```

The above figure shows the google drive mounted to the google colab and the necessary libraries imported and the data read from the drive.



```
dd.shape
```

```
(5000000, 19)
```

```
x=dd.iloc[:,[1,2,3,4,5,6,7,8,9,12,13,14,16,18]]
y=dd[0]
```

```
X_train, X_test, y_train, y_test = train_test_split(
            x, y, test_size=0.33, shuffle=True, random_state=1)
print(X_train.shape, y_train.shape, X_test.shape, y_test.shape)
```

```
(3350000, 14) (3350000,) (1650000, 14) (1650000,)
```

```
X_train.shape
```

```
(4000000, 18)
```

```
y_train.shape
```

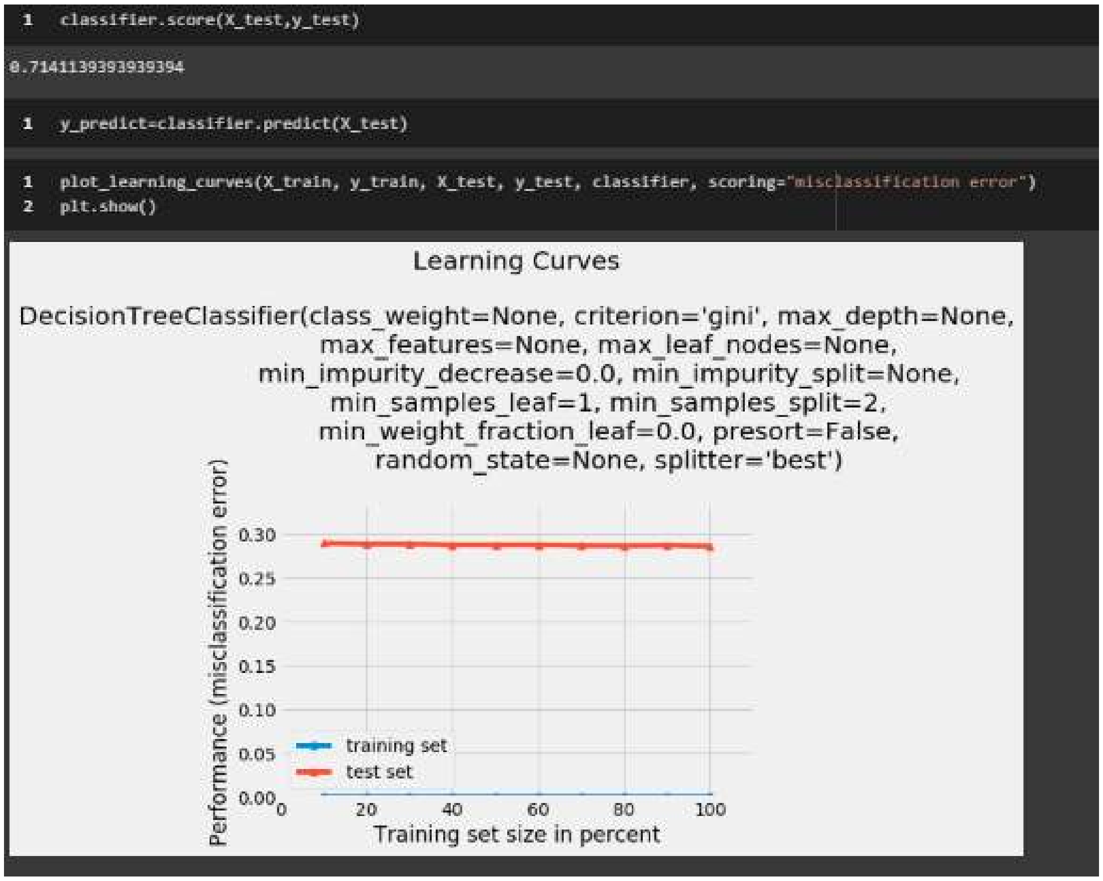
```
(4000000,)
```

```
classifier=DecisionTreeClassifier()
```

```
classifier.fit(X_train,y_train)
```

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=None, splitter='best')
```

The above figure shows the selected attributes selected for training in x variable and the split of random shuffled dataset into train and test set in ratio 2:1. Then the decision tree classifier is fit to the selected data.



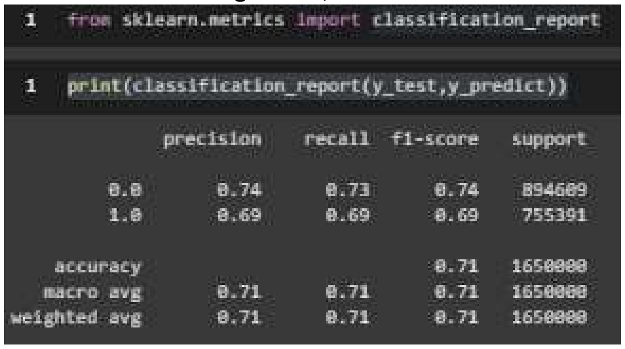
```
classifier.score(X_test,y_test)
```

```
0.7141139393939394
```

```
y_predict=classifier.predict(X_test)
```

```
plot_learning_curves(X_train, y_train, X_test, y_test, classifier, scoring="misclassification error")
plt.show()
```

The above figure shows the accuracy score of the classifier 0.71411 which is less than the accuracy score of the decision tree classifier with all the attributes due to loss of information by not selecting some attributes. The figure also shows the learning curve of the classifier which is decreasing as more training data is feed to the algorithm, the error is closed to less than 0.28.



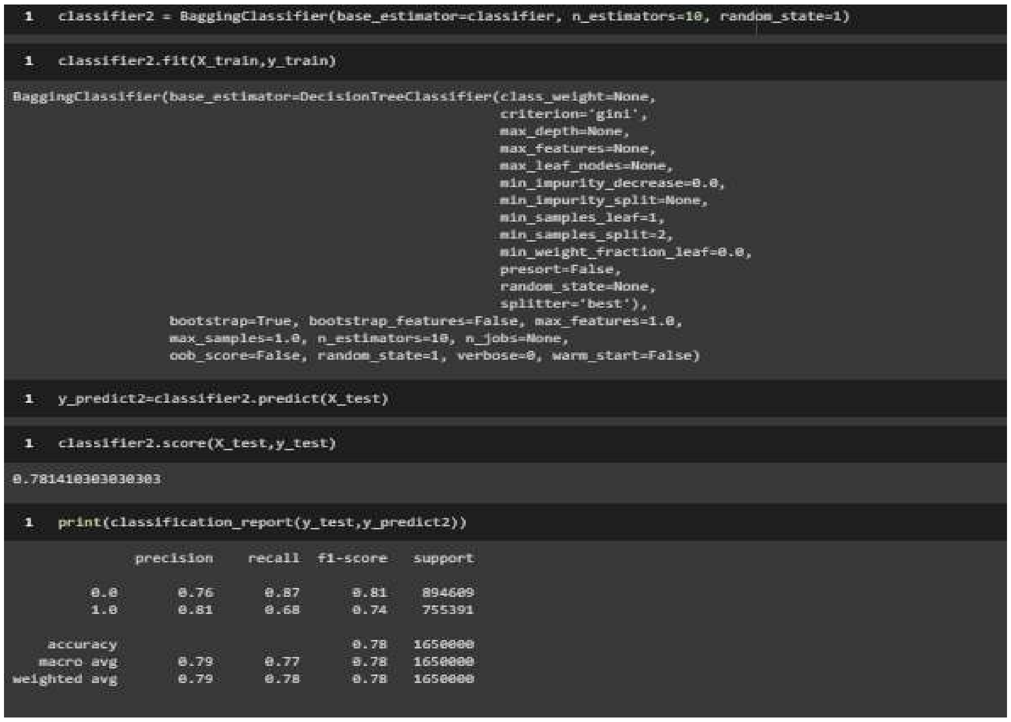
```
from sklearn.metrics import classification_report
```

```
print(classification_report(y_test,y_predict))
```

```
              precision    recall  f1-score   support

         0.0       0.74      0.73      0.74    894609
         1.0       0.69      0.69      0.69    755391

    accuracy                           0.71   1650000
   macro avg       0.71      0.71      0.71   1650000
weighted avg       0.71      0.71      0.71   1650000
```

The above figure shows the classification report of the classifier, the precision, recall, f1-score, support and accuracy.

1.5 Ensemble of Decision Tree Classifiers with Selected Attributes:
Ensemble of decision tree classifier with selected attributes:



```
classifier2 = BaggingClassifier(base_estimator=classifier, n_estimators=10, random_state=1)
```

```
classifier2.fit(X_train,y_train)
```

```
BaggingClassifier(base_estimator=DecisionTreeClassifier(class_weight=None,
                                                        criterion='gini',
                                                        max_depth=None,
                                                        max_features=None,
                                                        max_leaf_nodes=None,
                                                        min_impurity_decrease=0.0,
                                                        min_impurity_split=None,
                                                        min_samples_leaf=1,
                                                        min_samples_split=2,
                                                        min_weight_fraction_leaf=0.0,
                                                        presort=False,
                                                        random_state=None,
                                                        splitter='best'),
                  bootstrap=True, bootstrap_features=False, max_features=1.0,
                  max_samples=1.0, n_estimators=10, n_jobs=None,
                  oob_score=False, random_state=1, verbose=0, warm_start=False)
```

```
y_predict2=classifier2.predict(X_test)
```

```
classifier2.score(X_test,y_test)
```

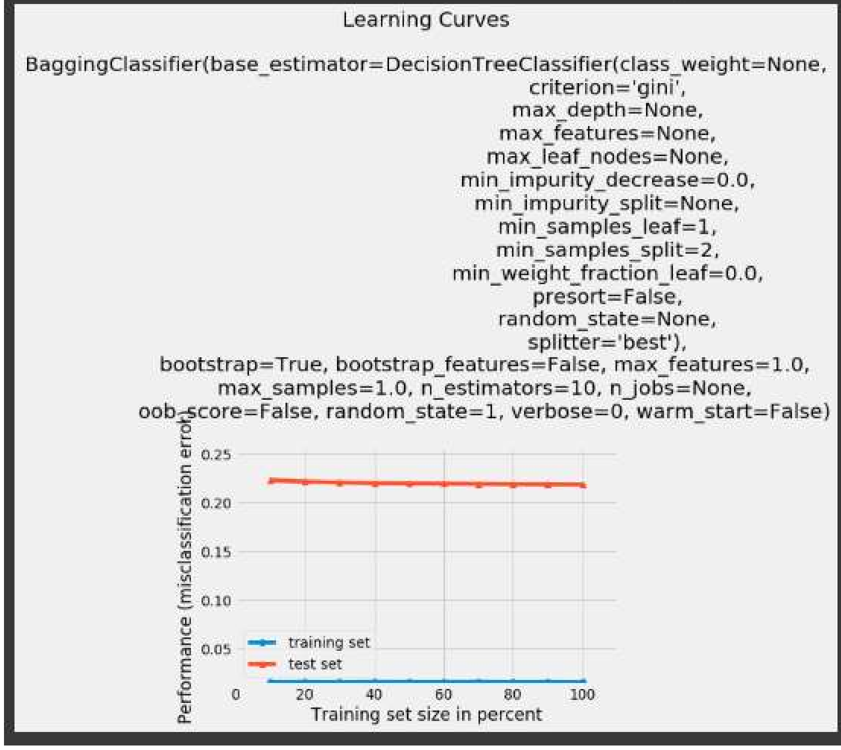
```
0.781410303030303
```

```
print(classification_report(y_test,y_predict2))
```

```
              precision    recall  f1-score   support

         0.0       0.76      0.87      0.81    894609
         1.0       0.81      0.68      0.74    755391

    accuracy                           0.78   1650000
   macro avg       0.79      0.77      0.78   1650000
weighted avg       0.79      0.78      0.78   1650000
```

The above figure shows the fit of the bagging classifier with selected attributes and the accuracy score 0.7814 which is more than the accuracy of decision tree with selected attributes but less than the bagging classifier with all the attributes due to loss of information with loss of attributes. The classifier report shows the precision, recall, f1-score, support and accuracy of the algorithm.

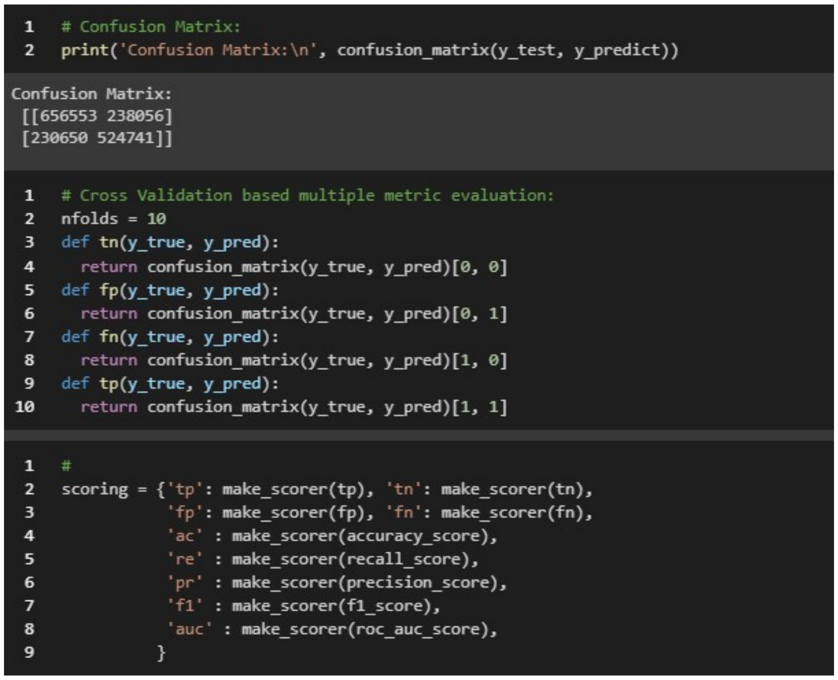
```
# Learning curves:
plot_learning_curves(X_train, y_train, X_test, y_test, classifier2, scoring="misclassification error")
plt.show()
```



The above figure shows the learning curve of the ensemble of decision tree with selected attributes. It can be seen that the error gradually decreases as more data is feed to the algorithm, the error being close to less than 0.22 which is less than decision tree with selected attributes and almost equal to the error of ensemble decision tree classifier with all the attributes.

1.6 Comparative evaluation of the Classifiers

Comparative evaluation of the classifiers:
Cross validation for decision tree with all attributes



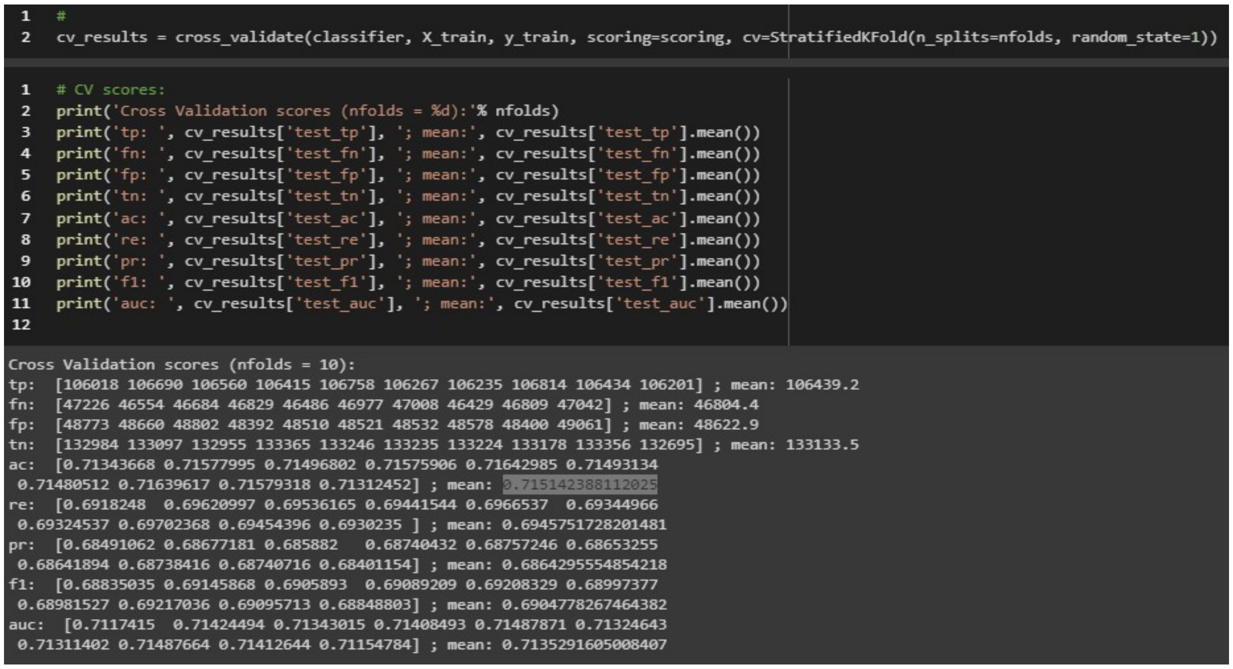
```
# Confusion Matrix:
print('Confusion Matrix:\n', confusion_matrix(y_test, y_predict))
```

```
Confusion Matrix:
 [[656553 238056]
 [230650 524741]]
```

```
# Cross Validation based multiple metric evaluation:
nfolds = 10
def tn(y_true, y_pred):
  return confusion_matrix(y_true, y_pred)[0, 0]
def fp(y_true, y_pred):
  return confusion_matrix(y_true, y_pred)[0, 1]
def fn(y_true, y_pred):
  return confusion_matrix(y_true, y_pred)[1, 0]
def tp(y_true, y_pred):
  return confusion_matrix(y_true, y_pred)[1, 1]
```

```
#
scoring = {'tp': make_scorer(tp), 'tn': make_scorer(tn),
           'fp': make_scorer(fp), 'fn': make_scorer(fn),
           'ac' : make_scorer(accuracy_score),
           're' : make_scorer(recall_score),
           'pr' : make_scorer(precision_score),
           'f1' : make_scorer(f1_score),
           'auc' : make_scorer(roc_auc_score),
          }
```

The above figure shows the confusion matrix of the decision tree algorithm.



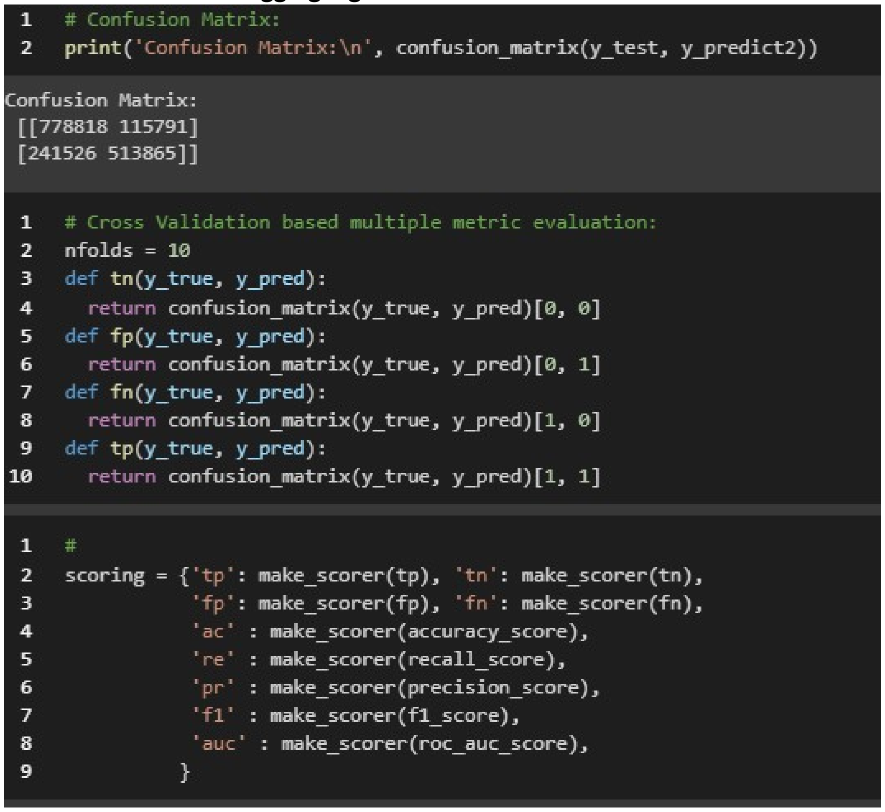
```
#
cv_results = cross_validate(classifier, X_train, y_train, scoring=scoring, cv=StratifiedKFold(n_splits=nfolds, random_state=1))
```

```
# CV scores:
print('Cross Validation scores (nfolds = %d):'% nfolds)
print('tp: ', cv_results['test_tp'], '; mean:', cv_results['test_tp'].mean())
print('fn: ', cv_results['test_fn'], '; mean:', cv_results['test_fn'].mean())
print('fp: ', cv_results['test_fp'], '; mean:', cv_results['test_fp'].mean())
print('tn: ', cv_results['test_tn'], '; mean:', cv_results['test_tn'].mean())
print('ac: ', cv_results['test_ac'], '; mean:', cv_results['test_ac'].mean())
print('re: ', cv_results['test_re'], '; mean:', cv_results['test_re'].mean())
print('pr: ', cv_results['test_pr'], '; mean:', cv_results['test_pr'].mean())
print('f1: ', cv_results['test_f1'], '; mean:', cv_results['test_f1'].mean())
print('auc: ', cv_results['test_auc'], '; mean:', cv_results['test_auc'].mean())

```

```
Cross Validation scores (nfolds = 10):
tp:  [106018 106690 106560 106415 106758 106267 106235 106814 106434 106201] ; mean: 106439.2
fn:  [47226 46554 46684 46829 46486 46977 47008 46429 46809 47042] ; mean: 46804.4
fp:  [48773 48660 48802 48392 48510 48521 48532 48578 48400 49061] ; mean: 48622.9
tn:  [132984 133097 132955 133365 133246 133235 133224 133178 133356 132695] ; mean: 133133.5
ac:  [0.71343668 0.71577995 0.71496802 0.71575906 0.71642985 0.71493134
 0.71480512 0.71639617 0.71579318 0.71312452] ; mean: 0.715142388112025
re:  [0.6918248  0.69620997 0.69536165 0.69441544 0.6966537  0.69344966
 0.69324537 0.69702368 0.69454396 0.6930235 ] ; mean: 0.6945751728201481
pr:  [0.68491062 0.68677181 0.685882   0.68740432 0.68757246 0.68653255
 0.68641894 0.68738416 0.68740716 0.68401154] ; mean: 0.6864295554854218
f1:  [0.68835035 0.69145868 0.6905893  0.69089209 0.69208329 0.68997377
 0.68981527 0.69217036 0.69095713 0.68848803] ; mean: 0.6904778267464382
auc:  [0.7117415  0.71424494 0.71343015 0.71408493 0.71487871 0.71324643
 0.71311402 0.71487664 0.71412644 0.71154784] ; mean: 0.7135291605008407
```

The above figure shows the cross validation for 10 fold for the decision tree algorithm. We can see the values of 10 folds true positive, false positive, true negative, false negative, accuracy, recall, precision, f1-score and AUC (area under the curve). The accuracy and auc is above 0.5 hence the algorithm is working correctly.

Cross validation for bagging algorithm with all attributes:



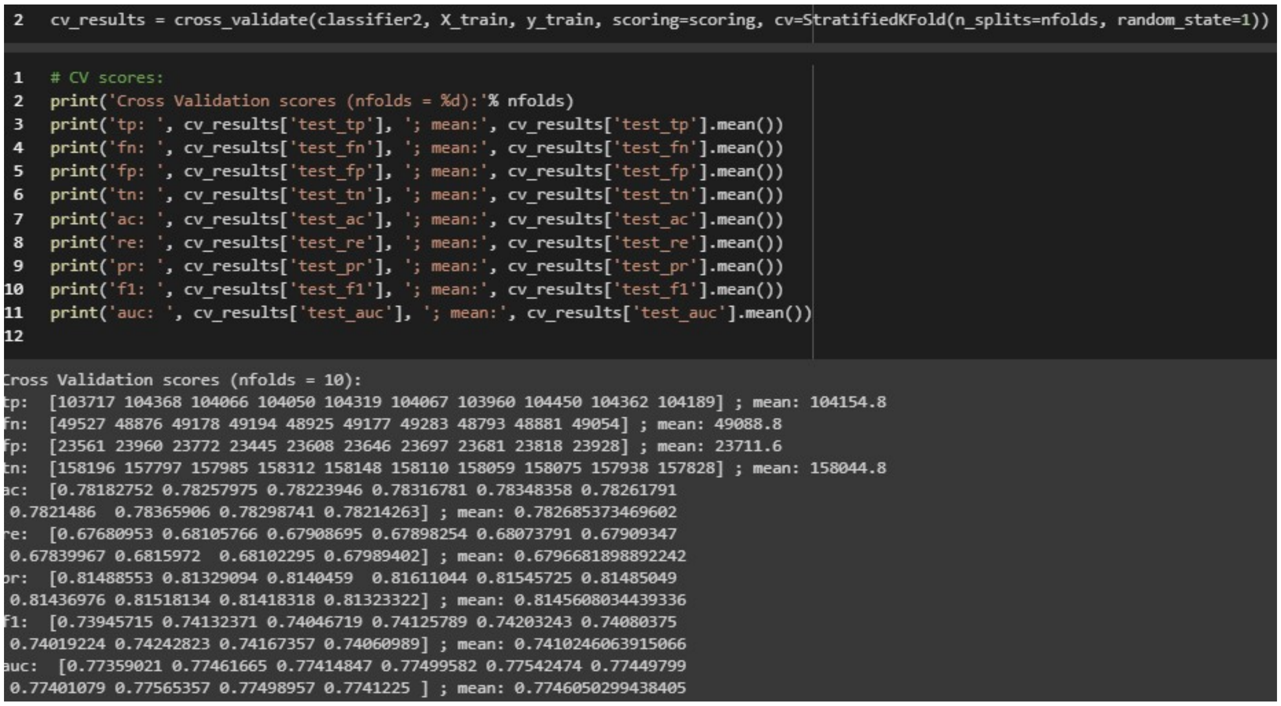
```
# Confusion Matrix:
print('Confusion Matrix:\n', confusion_matrix(y_test, y_predict2))
```

```
Confusion Matrix:
 [[778818 115791]
 [241526 513865]]
```

```
# Cross Validation based multiple metric evaluation:
nfolds = 10
def tn(y_true, y_pred):
  return confusion_matrix(y_true, y_pred)[0, 0]
def fp(y_true, y_pred):
  return confusion_matrix(y_true, y_pred)[0, 1]
def fn(y_true, y_pred):
  return confusion_matrix(y_true, y_pred)[1, 0]
def tp(y_true, y_pred):
  return confusion_matrix(y_true, y_pred)[1, 1]
```

```
#
scoring = {'tp': make_scorer(tp), 'tn': make_scorer(tn),
           'fp': make_scorer(fp), 'fn': make_scorer(fn),
           'ac' : make_scorer(accuracy_score),
           're' : make_scorer(recall_score),
           'pr' : make_scorer(precision_score),
           'f1' : make_scorer(f1_score),
           'auc' : make_scorer(roc_auc_score),
          }
```

The above figure shows the confusion matrix of bagging algorithm.



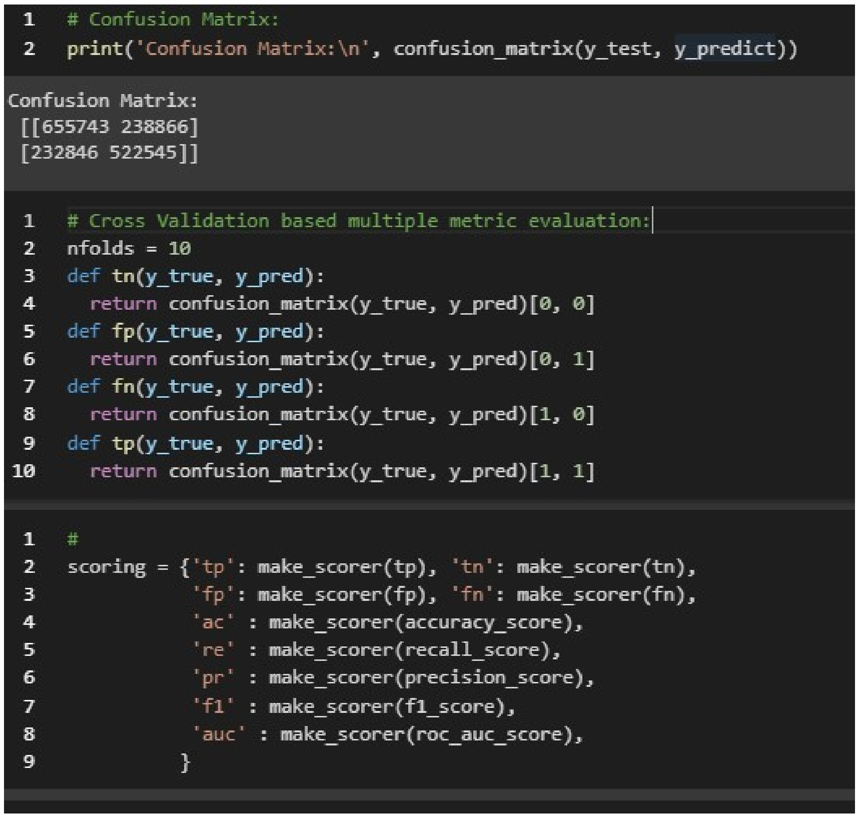
```
cv_results = cross_validate(classifier2, X_train, y_train, scoring=scoring, cv=StratifiedKFold(n_splits=nfolds, random_state=1))
```

```
# CV scores:
print('Cross Validation scores (nfolds = %d):'% nfolds)
print('tp: ', cv_results['test_tp'], '; mean:', cv_results['test_tp'].mean())
print('fn: ', cv_results['test_fn'], '; mean:', cv_results['test_fn'].mean())
print('fp: ', cv_results['test_fp'], '; mean:', cv_results['test_fp'].mean())
print('tn: ', cv_results['test_tn'], '; mean:', cv_results['test_tn'].mean())
print('ac: ', cv_results['test_ac'], '; mean:', cv_results['test_ac'].mean())
print('re: ', cv_results['test_re'], '; mean:', cv_results['test_re'].mean())
print('pr: ', cv_results['test_pr'], '; mean:', cv_results['test_pr'].mean())
print('f1: ', cv_results['test_f1'], '; mean:', cv_results['test_f1'].mean())
print('auc: ', cv_results['test_auc'], '; mean:', cv_results['test_auc'].mean())
```

```
Cross Validation scores (nfolds = 10):
tp:  [103717 104368 104066 104050 104319 104067 103960 104450 104362 104189] ; mean: 104154.8
fn:  [49527 48876 49178 49194 48925 49177 49283 48793 48881 49054] ; mean: 49088.8
fp:  [23561 23960 23772 23445 23608 23646 23697 23681 23818 23928] ; mean: 23711.6
tn:  [158196 157797 157985 158312 158148 158110 158059 158075 157938 157828] ; mean: 158044.8
ac:  [0.78182752 0.78257975 0.78223946 0.78316781 0.78348358 0.78261791
 0.7821486  0.78365906 0.78298741 0.78214263] ; mean: 0.782685373469602
re:  [0.67680953 0.68105766 0.67908695 0.67898254 0.68073791 0.67909347
 0.67839967 0.6815972  0.68102295 0.67989402] ; mean: 0.6796681898892242
pr:  [0.81488553 0.81329094 0.8140459  0.81611044 0.81545725 0.81485049
 0.81436976 0.81518134 0.81418318 0.81323322] ; mean: 0.8145608034439336
f1:  [0.73945715 0.74132371 0.74046719 0.74125789 0.74203243 0.74080375
 0.74019224 0.74242823 0.74167357 0.74060989] ; mean: 0.7410246063915066
auc:  [0.77359021 0.77461665 0.77414847 0.77499582 0.77542474 0.77449799
 0.77401079 0.77565357 0.77498957 0.7741225 ] ; mean: 0.7746050299438405
```

The above figure shows the cross validation for 10 fold for the bagging algorithm. We can see the values of 10 folds true positive, false positive, true negative, false negative, accuracy, recall, precision, f1-score and AUC (area under the curve). The accuracy and auc is above 0.5 hence the algorithm is working correctly.

Cross validation for decision tree with selected attributes:



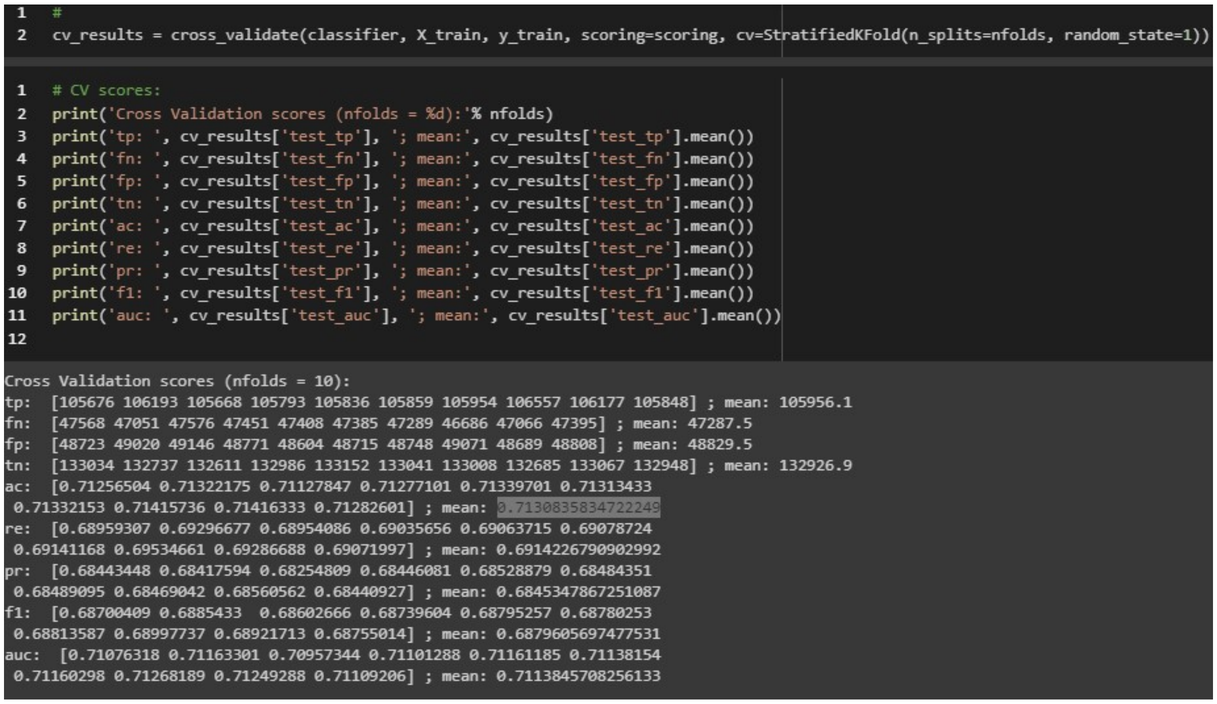
```
# Confusion Matrix:
print('Confusion Matrix:\n', confusion_matrix(y_test, y_predict))
```

```
Confusion Matrix:
 [[655743 238866]
 [232846 522545]]
```

```
# Cross Validation based multiple metric evaluation:
nfolds = 10
def tn(y_true, y_pred):
  return confusion_matrix(y_true, y_pred)[0, 0]
def fp(y_true, y_pred):
  return confusion_matrix(y_true, y_pred)[0, 1]
def fn(y_true, y_pred):
  return confusion_matrix(y_true, y_pred)[1, 0]
def tp(y_true, y_pred):
  return confusion_matrix(y_true, y_pred)[1, 1]
```

```
#
scoring = {'tp': make_scorer(tp), 'tn': make_scorer(tn),
           'fp': make_scorer(fp), 'fn': make_scorer(fn),
           'ac' : make_scorer(accuracy_score),
           're' : make_scorer(recall_score),
           'pr' : make_scorer(precision_score),
           'f1' : make_scorer(f1_score),
           'auc' : make_scorer(roc_auc_score),
          }
```

The above figure shows the confusion matrix of the decision tree algorithm with selected attributes.



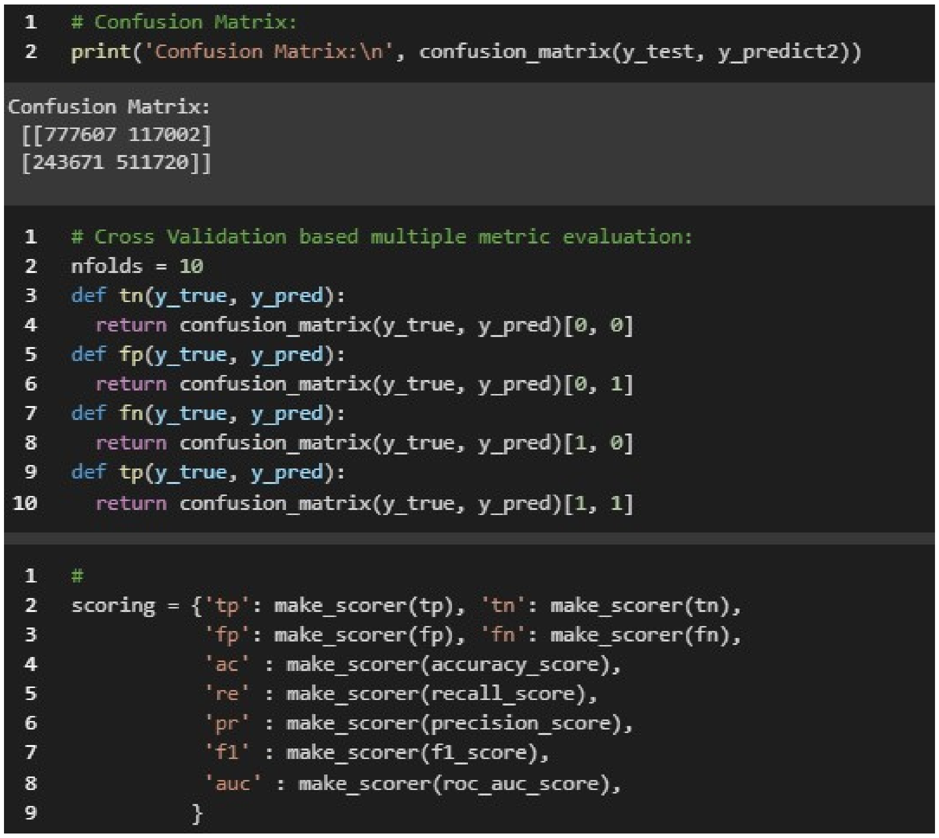
```
#
cv_results = cross_validate(classifier, X_train, y_train, scoring=scoring, cv=StratifiedKFold(n_splits=nfolds, random_state=1))
```

```
# CV scores:
print('Cross Validation scores (nfolds = %d):'% nfolds)
print('tp: ', cv_results['test_tp'], '; mean:', cv_results['test_tp'].mean())
print('fn: ', cv_results['test_fn'], '; mean:', cv_results['test_fn'].mean())
print('fp: ', cv_results['test_fp'], '; mean:', cv_results['test_fp'].mean())
print('tn: ', cv_results['test_tn'], '; mean:', cv_results['test_tn'].mean())
print('ac: ', cv_results['test_ac'], '; mean:', cv_results['test_ac'].mean())
print('re: ', cv_results['test_re'], '; mean:', cv_results['test_re'].mean())
print('pr: ', cv_results['test_pr'], '; mean:', cv_results['test_pr'].mean())
print('f1: ', cv_results['test_f1'], '; mean:', cv_results['test_f1'].mean())
print('auc: ', cv_results['test_auc'], '; mean:', cv_results['test_auc'].mean())
```

```
Cross Validation scores (nfolds = 10):
tp:  [105676 106193 105668 105793 105836 105859 105954 106557 106177 105848] ; mean: 105956.1
fn:  [47568 47051 47576 47451 47408 47385 47289 46686 47066 47395] ; mean: 47287.5
fp:  [48723 49020 49146 48771 48604 48715 48748 49071 48689 48808] ; mean: 48829.5
tn:  [133034 132737 132611 132986 133152 133041 133008 132685 133067 132948] ; mean: 132926.9
ac:  [0.71256504 0.71322175 0.71127847 0.71277101 0.71339701 0.71313433
 0.71332153 0.71415736 0.71416333 0.71282601] ; mean: 0.7130835834722249
re:  [0.68959307 0.69296677 0.68954086 0.69035656 0.69063715 0.69078724
 0.69141168 0.69534661 0.69286688 0.69071997] ; mean: 0.6914226790902992
pr:  [0.68443448 0.68417594 0.68254809 0.68446081 0.68528879 0.68484351
 0.68489095 0.68469042 0.68560562 0.68440927] ; mean: 0.6845347867251087
f1:  [0.68700409 0.6885433  0.68602666 0.68739604 0.68795257 0.68780253
 0.68813587 0.68997737 0.68921713 0.68755014] ; mean: 0.6879605697477531
auc:  [0.71076318 0.71163301 0.70957344 0.71101288 0.71161185 0.71138154
 0.71160298 0.71268189 0.71249288 0.71109206] ; mean: 0.7113845708256133
```

The above figure shows the cross validation for 10 fold for the decision tree algorithm. We can see the values of 10 folds true positive, false positive, true negative, false negative, accuracy, recall, precision, f1-score and AUC (area under the curve). The accuracy and auc is above 0.5 hence the algorithm is working correctly.

Cross validation for bagging algorithm with selected attributes:



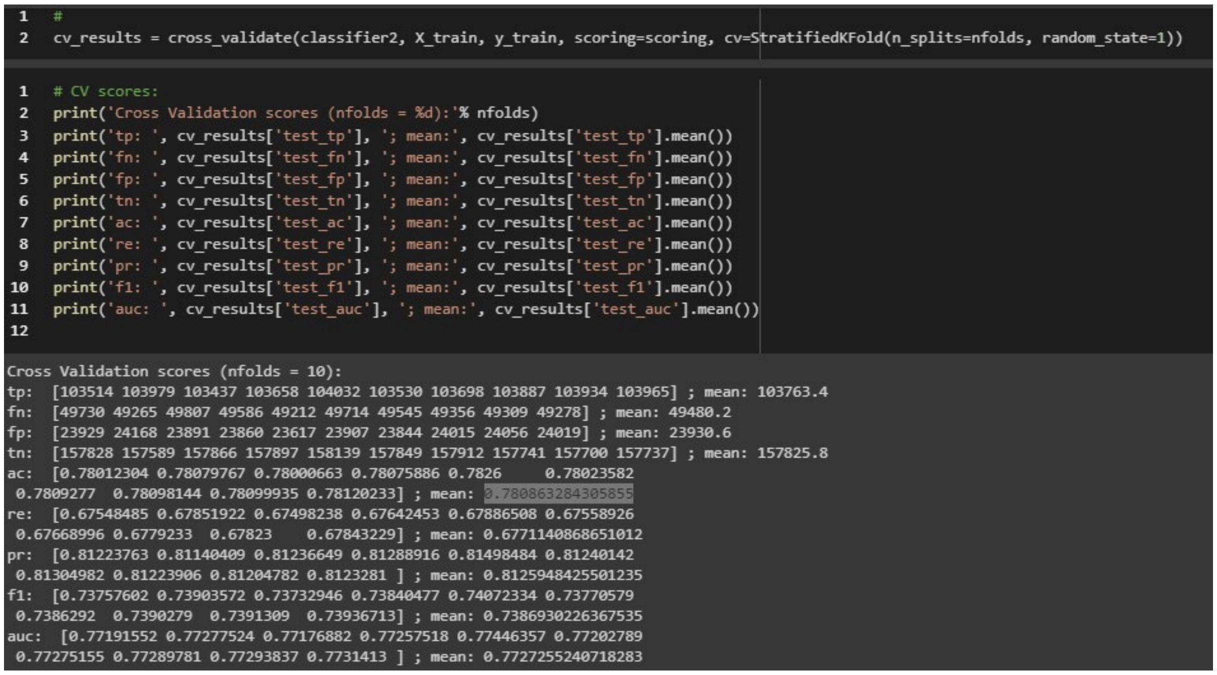
```
# Confusion Matrix:
print('Confusion Matrix:\n', confusion_matrix(y_test, y_predict2))
```

```
Confusion Matrix:
 [[777607 117002]
 [243671 511720]]
```

```
# Cross Validation based multiple metric evaluation:
nfolds = 10
def tn(y_true, y_pred):
  return confusion_matrix(y_true, y_pred)[0, 0]
def fp(y_true, y_pred):
  return confusion_matrix(y_true, y_pred)[0, 1]
def fn(y_true, y_pred):
  return confusion_matrix(y_true, y_pred)[1, 0]
def tp(y_true, y_pred):
  return confusion_matrix(y_true, y_pred)[1, 1]
```

```
#
scoring = {'tp': make_scorer(tp), 'tn': make_scorer(tn),
           'fp': make_scorer(fp), 'fn': make_scorer(fn),
           'ac' : make_scorer(accuracy_score),
           're' : make_scorer(recall_score),
           'pr' : make_scorer(precision_score),
           'f1' : make_scorer(f1_score),
           'auc' : make_scorer(roc_auc_score),
          }
```

The above figure shows the confusion matrix for bagging algorithm with selected attributes.



```
#
cv_results = cross_validate(classifier2, X_train, y_train, scoring=scoring, cv=StratifiedKFold(n_splits=nfolds, random_state=1))
```

```
# CV scores:
print('Cross Validation scores (nfolds = %d):'% nfolds)
print('tp: ', cv_results['test_tp'], '; mean:', cv_results['test_tp'].mean())
print('fn: ', cv_results['test_fn'], '; mean:', cv_results['test_fn'].mean())
print('fp: ', cv_results['test_fp'], '; mean:', cv_results['test_fp'].mean())
print('tn: ', cv_results['test_tn'], '; mean:', cv_results['test_tn'].mean())
print('ac: ', cv_results['test_ac'], '; mean:', cv_results['test_ac'].mean())
print('re: ', cv_results['test_re'], '; mean:', cv_results['test_re'].mean())
print('pr: ', cv_results['test_pr'], '; mean:', cv_results['test_pr'].mean())
print('f1: ', cv_results['test_f1'], '; mean:', cv_results['test_f1'].mean())
print('auc: ', cv_results['test_auc'], '; mean:', cv_results['test_auc'].mean())

```

```
Cross Validation scores (nfolds = 10):
tp:  [103514 103979 103437 103658 104032 103530 103698 103887 103934 103965] ; mean: 103763.4
fn:  [49730 49265 49807 49586 49212 49714 49545 49356 49309 49278] ; mean: 49480.2
fp:  [23929 24168 23891 23860 23617 23907 23844 24015 24056 24019] ; mean: 23930.6
tn:  [157828 157589 157866 157897 158139 157849 157912 157741 157700 157737] ; mean: 157825.8
ac:  [0.78012304 0.78079767 0.78000663 0.78075886 0.7826     0.78023582
 0.7809277  0.78098144 0.78099935 0.78120233] ; mean: 0.780863284305855
re:  [0.67548485 0.67851922 0.67498238 0.67642453 0.67886508 0.67558926
 0.67668996 0.6779233  0.67823    0.67843229] ; mean: 0.6771140868651012
pr:  [0.81223763 0.81140409 0.81236649 0.81288916 0.81498484 0.81240142
 0.81304982 0.81223906 0.81204782 0.8123281 ] ; mean: 0.8125948425501235
f1:  [0.73757602 0.73903572 0.73732946 0.73840477 0.74072334 0.73770579
 0.7386292  0.7390279  0.7391309  0.73936713] ; mean: 0.7386930226367535
auc:  [0.77191552 0.77277524 0.77176882 0.77257518 0.77446357 0.77202789
 0.77275155 0.77289781 0.77293837 0.7731413 ] ; mean: 0.7727255240718283
```

The above figure shows the cross validation for 10 fold for the decision tree algorithm. We can see the values of 10 folds true positive, false positive, true negative, false negative, accuracy, recall, precision, f1-score and AUC (area under the curve). The accuracy and auc is above 0.5 hence the algorithm is working correctly.

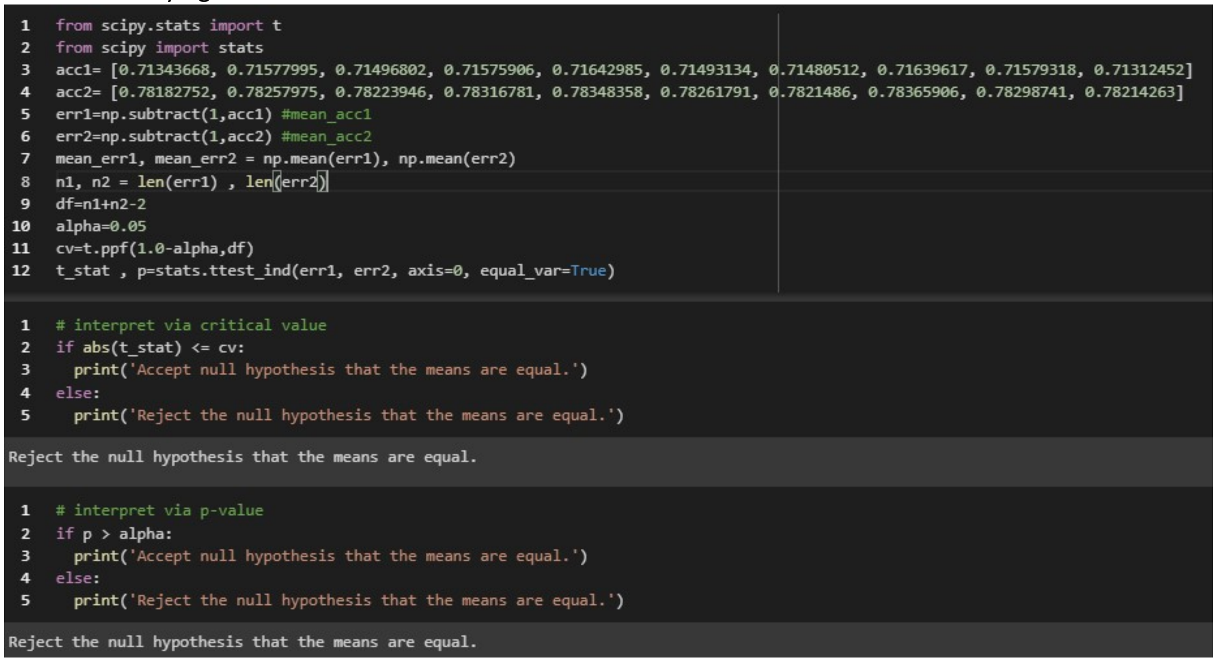
Comparison between decision tree algorithm with all attributes and bagging algorithm with all attributes:

We can observe from the value of accuracy and AUC of both the algorithms below:

	Decision tree algorithm	Bagging of decision tree algorithm
Accuracy	0.715142388112025	0.782685373469602
AUC	0.7135291605008407	0.7746050299438405

The value of accuracy and AUC of both the algorithm is more than 0.5 hence the algorithm is able to predict correctly for more than 50% of times. The accuracy of decision tree is 71.51% whereas accuracy of bagging algorithm is 78.26% which is higher than the accuracy of decision tree. This is due to ensemble of the decision tree in bagging which improves the performance of the algorithm by using different decision tree for different cases, in other word the ensemble contains different decision trees which has better performance for different types of input hence a single decision tree may miss classify some of the inputs but ensemble is more likely to correctly classify the inputs.

T-test for verifying the skill scores:



```
from scipy.stats import t
from scipy import stats
acc1= [0.71343668, 0.71577995, 0.71496802, 0.71575906, 0.71642985, 0.71493134, 0.71480512, 0.71639617, 0.71579318, 0.71312452]
acc2= [0.78182752, 0.78257975, 0.78223946, 0.78316781, 0.78348358, 0.78261791, 0.7821486, 0.78365906, 0.78298741, 0.78214263]
err1=np.subtract(1,acc1) #mean_acc1
err2=np.subtract(1,acc2) #mean_acc2
mean_err1, mean_err2 = np.mean(err1), np.mean(err2)
n1, n2 = len(err1) , len(err2)
df=n1+n2-2
alpha=0.05
cv=t.ppf(1.0-alpha,df)
t_stat , p=stats.ttest_ind(err1, err2, axis=0, equal_var=True)
```

```
# interpret via critical value
if abs(t_stat) <= cv:
  print('Accept null hypothesis that the means are equal.')
else:
  print('Reject the null hypothesis that the means are equal.')
```

```
Reject the null hypothesis that the means are equal.
```

```
# interpret via p-value
if p > alpha:
  print('Accept null hypothesis that the means are equal.')
else:
  print('Reject the null hypothesis that the means are equal.')
```

```
Reject the null hypothesis that the means are equal.
```

The above figure shows the t-test for verifying the skill scores and since the hypothesis is rejected hence we can conclude that the skill scores is statistically significant and the above conclusion holds correctly.

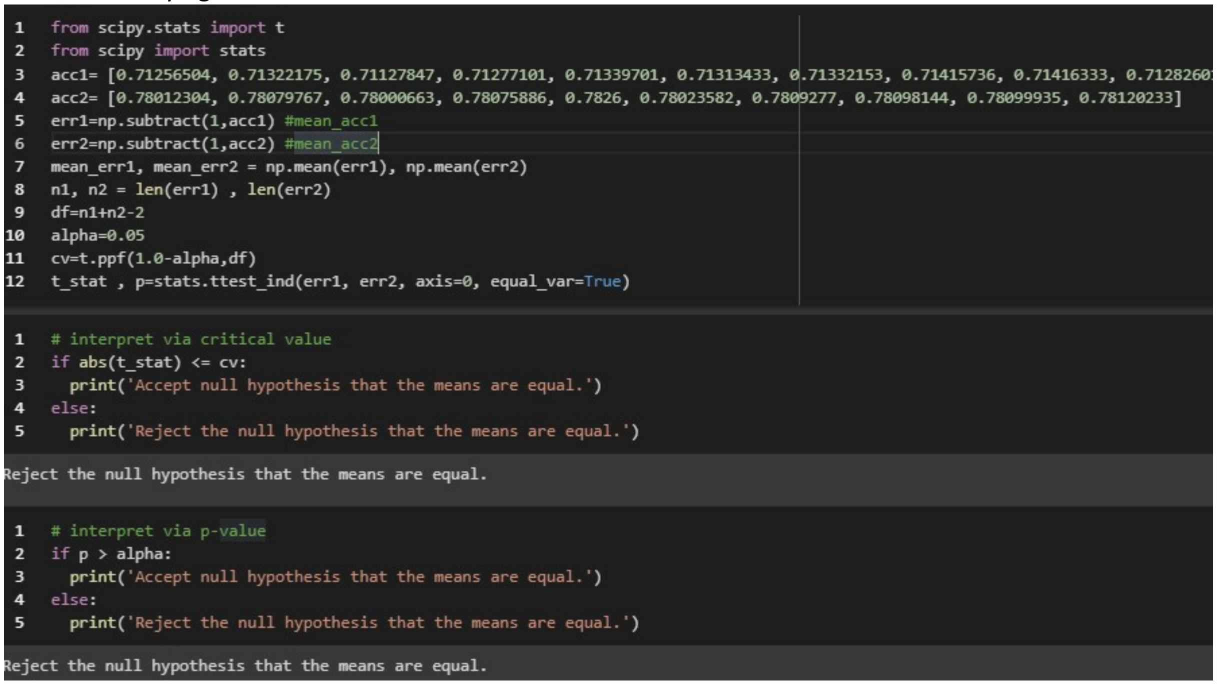
Comparison between decision tree algorithm with selected attributes and bagging algorithm with selected attributes:

We can observe from the value of accuracy and AUC of both the algorithms below:

	Decision tree algorithm	Bagging of decision tree algorithm
Accuracy	0.7130835834722249	0.780863284305855
AUC	0.7113845708256133	0.7727255240718283

The value of accuracy and AUC of both the algorithm is more than 0.5 hence the algorithm is able to predict correctly for more than 50% of times. The accuracy of decision tree is 71.30% whereas accuracy of bagging algorithm is 78.08% which is higher than the accuracy of decision tree. This is due to ensemble of the decision tree in bagging which improves the performance of the algorithm by using different decision tree for different cases, in other word the ensemble contains different decision trees which has better performance for different types of input hence a single decision tree may miss classify some of the inputs but ensemble is more likely to correctly classify the inputs.

T-test for verifying the skill scores:



```
from scipy.stats import t
from scipy import stats
acc1= [0.71256504, 0.71322175, 0.71127847, 0.71277101, 0.71339701, 0.71313433, 0.71332153, 0.71415736, 0.71416333, 0.7128260
acc2= [0.78012304, 0.78079767, 0.78000663, 0.78075886, 0.7826, 0.78023582, 0.7809277, 0.78098144, 0.78099935, 0.78120233]
err1=np.subtract(1,acc1) #mean_acc1
err2=np.subtract(1,acc2) #mean_acc2
mean_err1, mean_err2 = np.mean(err1), np.mean(err2)
n1, n2 = len(err1) , len(err2)
df=n1+n2-2
alpha=0.05
cv=t.ppf(1.0-alpha,df)
t_stat , p=stats.ttest_ind(err1, err2, axis=0, equal_var=True)
```

```
# interpret via critical value
if abs(t_stat) <= cv:
  print('Accept null hypothesis that the means are equal.')
else:
  print('Reject the null hypothesis that the means are equal.')
```

```
Reject the null hypothesis that the means are equal.
```

```
# interpret via p-value
if p > alpha:
  print('Accept null hypothesis that the means are equal.')
else:
  print('Reject the null hypothesis that the means are equal.')
```

```
Reject the null hypothesis that the means are equal.
```

The above figure shows the t-test for verifying the skill scores and since the hypothesis is rejected hence we can conclude that the skill scores is statistically significant and the above conclusion holds correctly.

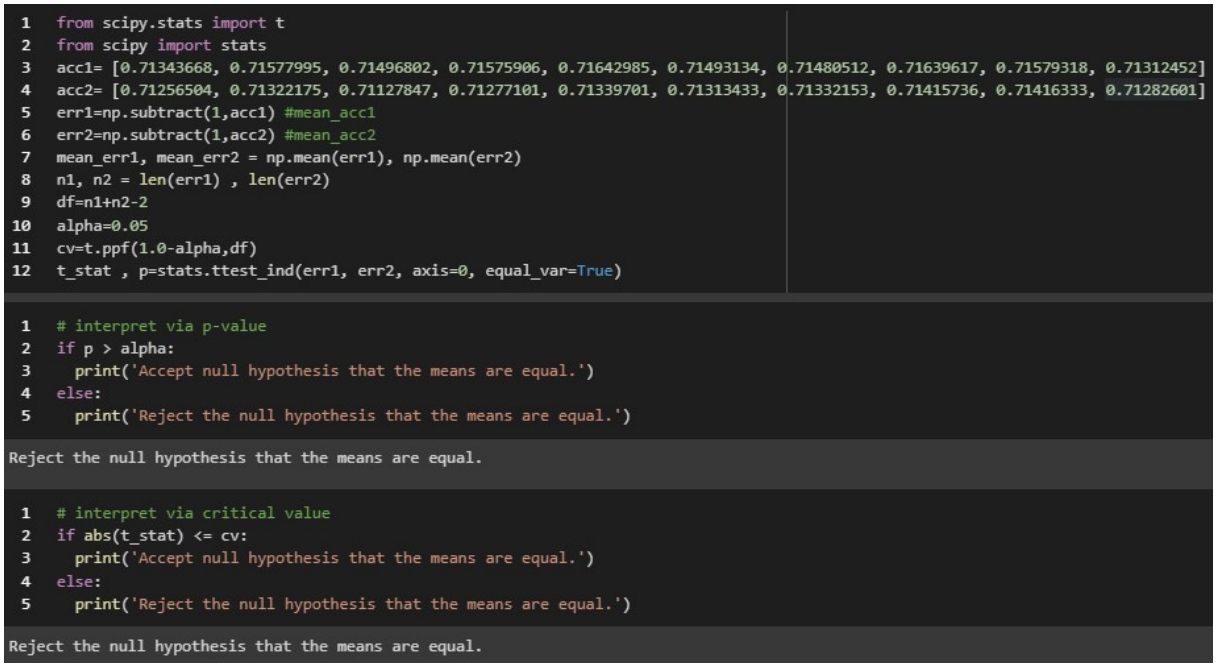
Comparison between decision tree algorithm with all attributes and decision tree with selected attributes:

We can observe from the value of accuracy and AUC of both the algorithms below:

	Decision tree with selected attributes	Decision tree with all attributes
Accuracy	0.7130835834722249	0.715142388112025
AUC	0.7113845708256133	0.7135291605008407

The value of accuracy and AUC of both the algorithm is more than 0.5 hence the algorithm is able to predict correctly for more than 50% of times. The accuracy of decision tree with selected attributes is 71.30% whereas accuracy of decision tree with all attributes is 71.51% which is higher than the accuracy of decision tree. This is due to loss of information in the decision tree with selected attributes, more data means more information which can help build better classifier with better performance of the algorithm. By using all attributes the classifier uses all the information available hence makes better prediction hence decision tree with all attributes has higher accuracy than decision tree with selected attributes.

T-test for verifying the skill scores:



```python
from scipy.stats import t
from scipy import stats
acc1= [0.71343668, 0.71577995, 0.71496802, 0.71575906, 0.71642985, 0.71493134, 0.71480512, 0.71639617, 0.71579318, 0.71312452]
acc2= [0.71256504, 0.71322175, 0.71127847, 0.71277101, 0.71339701, 0.71313433, 0.71332153, 0.71415736, 0.71416333, 0.71282601]
err1=np.subtract(1,acc1) #mean_acc1
err2=np.subtract(1,acc2) #mean_acc2
mean_err1, mean_err2 = np.mean(err1), np.mean(err2)
n1, n2 = len(err1) , len(err2)
df=n1+n2-2
alpha=0.05
cv=t.ppf(1.0-alpha,df)
t_stat , p=stats.ttest_ind(err1, err2, axis=0, equal_var=True)
```

```python
# interpret via p-value
if p > alpha:
  print('Accept null hypothesis that the means are equal.')
else:
  print('Reject the null hypothesis that the means are equal.')
```

```
Reject the null hypothesis that the means are equal.
```

```python
# interpret via critical value
if abs(t_stat) <= cv:
  print('Accept null hypothesis that the means are equal.')
else:
  print('Reject the null hypothesis that the means are equal.')
```

```
Reject the null hypothesis that the means are equal.
```

The above figure shows the t-test for verifying the skill scores and since the hypothesis is rejected hence we can conclude that the skill scores is statistically significant and the above conclusion holds correctly.

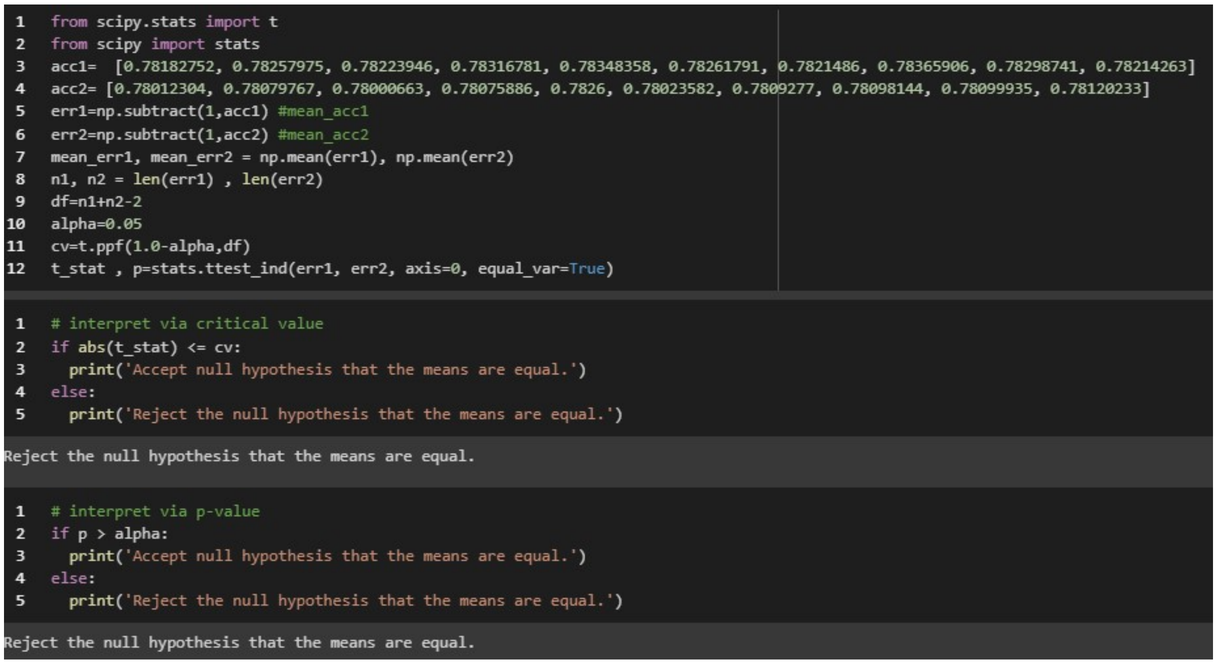
Comparison between ensemble of decision tree algorithm with all attributes and ensemble of decision tree with selected attributes:

We can observe from the value of accuracy and AUC of both the algorithms below:

	Ensemble with selected attributes	Ensemble with all attributes
Accuracy	0.780863284305855	0.782685373469602
AUC	0.7727255240718283	0.7746050299438405

The value of accuracy and AUC of both the algorithm is more than 0.5 hence the algorithm is able to predict correctly for more than 50% of times. The accuracy of ensemble of decision tree with selected attributes is 78.08% whereas accuracy of ensemble of decision tree with all attributes is 78.26% which is higher than the accuracy of decision tree. This is due to loss of information in the ensemble of decision tree with selected attributes, more data means more information which can help build better classifier with better performance of the algorithm. By using all attributes the classifier uses all the information available hence makes better prediction hence ensemble of decision tree with all attributes has higher accuracy than ensemble of decision tree with selected attributes.

T-test for verifying the skill scores:



```
from scipy.stats import t
from scipy import stats
acc1=  [0.78182752, 0.78257975, 0.78223946, 0.78316781, 0.78348358, 0.78261791, 0.7821486, 0.78365906, 0.78298741, 0.78214263]
acc2= [0.78012304, 0.78079767, 0.78000663, 0.78075886, 0.7826, 0.78023582, 0.7809277, 0.78098144, 0.78099935, 0.78120233]
err1=np.subtract(1,acc1) #mean_acc1
err2=np.subtract(1,acc2) #mean_acc2
mean_err1, mean_err2 = np.mean(err1), np.mean(err2)
n1, n2 = len(err1) , len(err2)
df=n1+n2-2
alpha=0.05
cv=t.ppf(1.0-alpha,df)
t_stat , p=stats.ttest_ind(err1, err2, axis=0, equal_var=True)
```

```
# interpret via critical value
if abs(t_stat) <= cv:
  print('Accept null hypothesis that the means are equal.')
else:
  print('Reject the null hypothesis that the means are equal.')
```

```
Reject the null hypothesis that the means are equal.
```

```
# interpret via p-value
if p > alpha:
  print('Accept null hypothesis that the means are equal.')
else:
  print('Reject the null hypothesis that the means are equal.')
```

```
Reject the null hypothesis that the means are equal.
```

The above figure shows the t-test for verifying the skill scores and since the hypothesis is rejected hence we can conclude that the skill scores is statistically significant and the above conclusion holds correctly.

1.7 Conclusions

In this assignment 4 classifiers were developed, decision tree algorithm, ensemble of decision tree, decision tree with selected attributes and ensemble of decision tree with selected attributes. All the different algorithm had different performance for the same input and test data. The ensemble of decision tree performed better than the single decision tree for both the cases due to ensemble which helped the algorithm perform better by using multiple decision trees. For ensemble 10 estimators are used with base classifier being the decision tree itself. The performance skill score verified by the T-test which rejects the hypothesis hence concluding that the skill scores are statistically significant. It is also seen that the algorithms with selected attributes has less accuracy than the algorithms that use all the attributes which is due to loss of information in selecting attributes and leaving some attributes. Therefore, ensemble of decision tree with attributes has best performance among all the 4 algorithms and the decision tree with selected attributes has least performance. Hence it can be concluded that best algorithm to use for this problem is ensemble algorithm with all attributes unless there is a problem of overfitting in which case few attributes can be dropped.