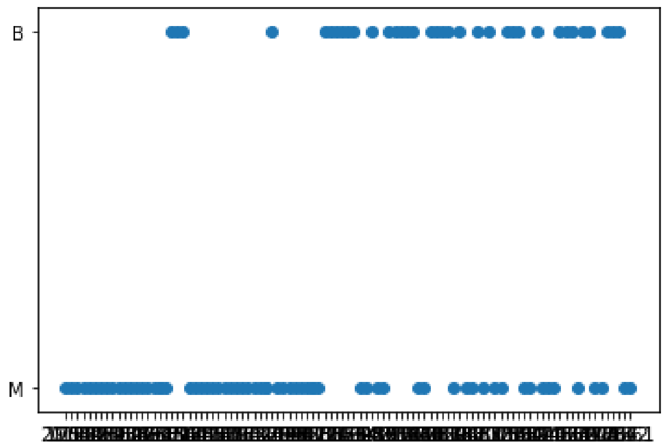
Data Mining
Lab 02

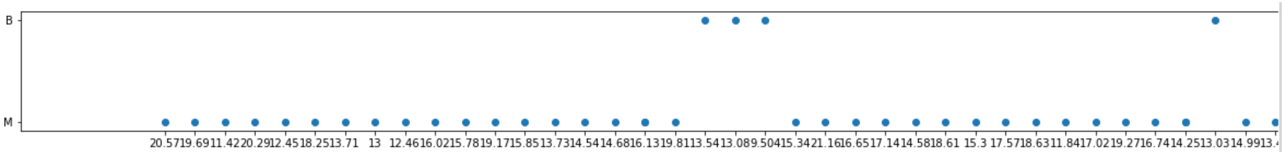
Names:

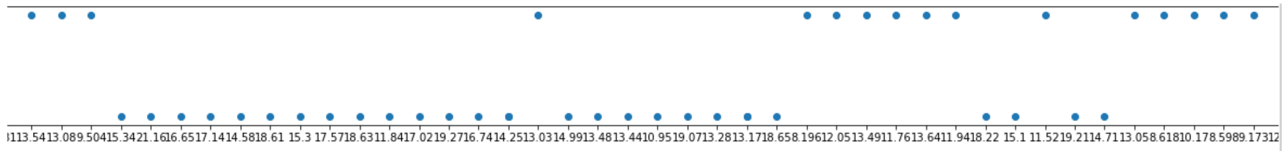
1) Khalil Ismail Khalil (23)
2) Ahmed Mohamed EL-Bawab (08)

1)Visualization:

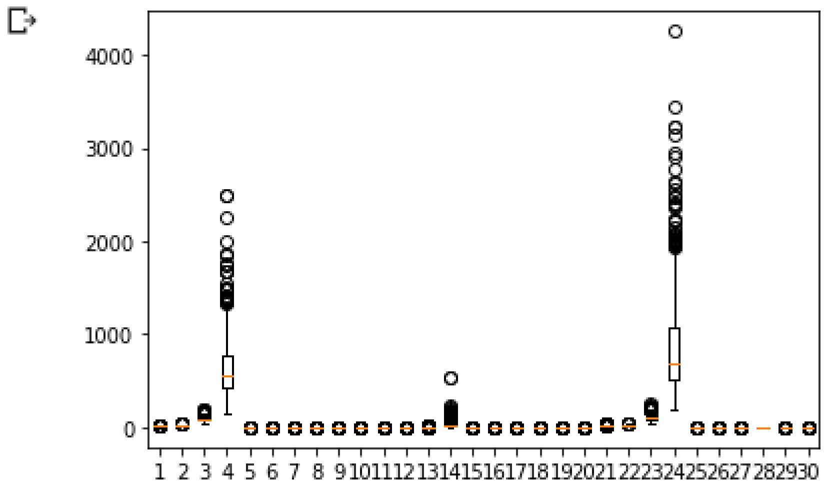
```
data1 = dataFrame.loc[1:100,2]
Class1 = dataFrame.loc[1:100,1]
plt.scatter(data1,Class1)
plt.show()
```

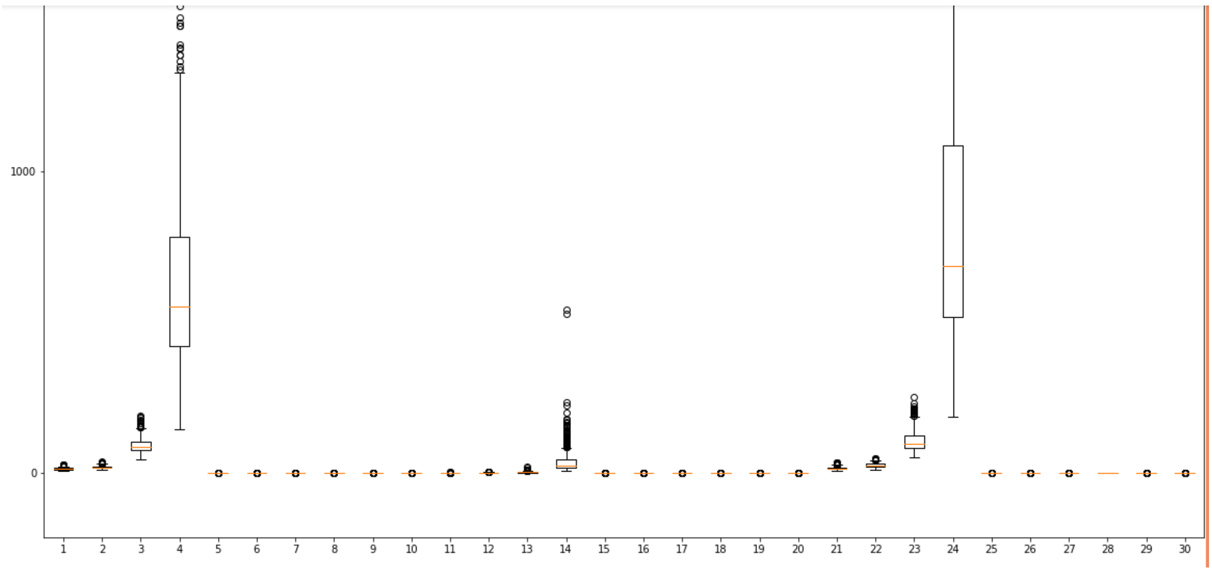






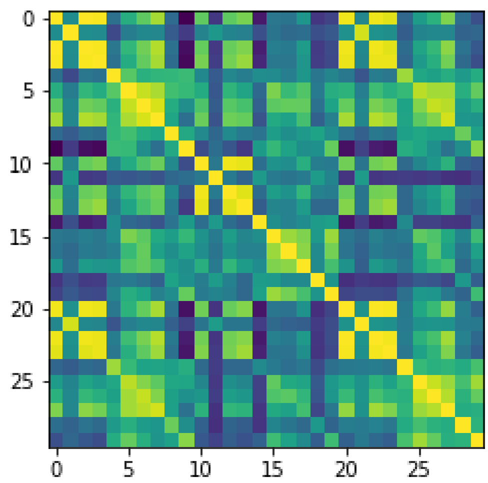
```python
# Create boxplot
plt.boxplot(attributesList)
plt.show()
```





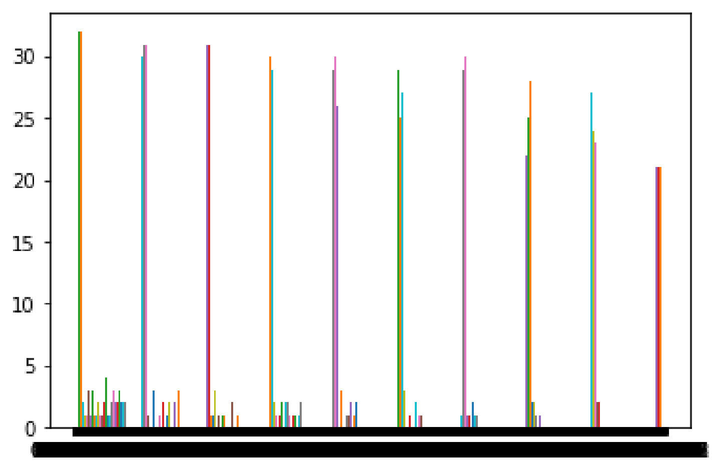

```python
#Visualize correlationMatrix using imshow
plt.imshow(correlationMatrix)
```

```
<matplotlib.image.AxesImage at 0x7f49604fd588>
```



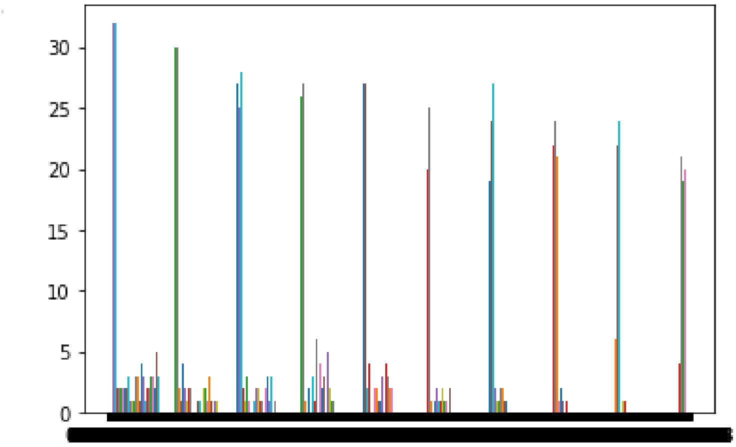

```python
tmp = dataFrame.loc[dataFrame[1] == 'M',:]

plt.hist(tmp)
plt.show()
```



```
tmp = dataFrame.loc[dataFrame[1] == 'B',:]

plt.hist(tmp)
plt.show()
```



2)Dataset Splitting:

```
----------------------------
398
==========
[['11.69' '24.44' '76.37' ... '0.1308' '0.2803' '0.0997\n']
 ['12.23' '19.56' '78.54' ... '0.108' '0.2668' '0.08174\n']
 ['10.8' '9.71' '68.77' ... '0.04603' '0.209' '0.07699\n']
 ...
 ['12.19' '13.29' '79.08' ... '0.08187' '0.3469' '0.09241\n']
 ['6.981' '13.43' '43.79' ... '0' '0.2932' '0.09382\n']
 ['12.06' '18.9' '76.66' ... '0.05093' '0.288' '0.08083\n']]
----------------------------
171
==========
[['27.22' '21.87' '182.1' ... '0.2688' '0.2856' '0.08082\n']
 ['12.96' '18.29' '84.18' ... '0.06608' '0.3207' '0.07247\n']
 ['11.67' '20.02' '75.21' ... '0.0812' '0.3206' '0.0895\n']
 ...
 ['15.27' '12.91' '98.17' ... '0.1035' '0.232' '0.07474\n']
 ['14.4' '26.99' '92.25' ... '0.05563' '0.2345' '0.06464\n']
 ['12.2' '15.21' '78.01' ... '0.05556' '0.2661' '0.07961\n']]
----------------------------
```

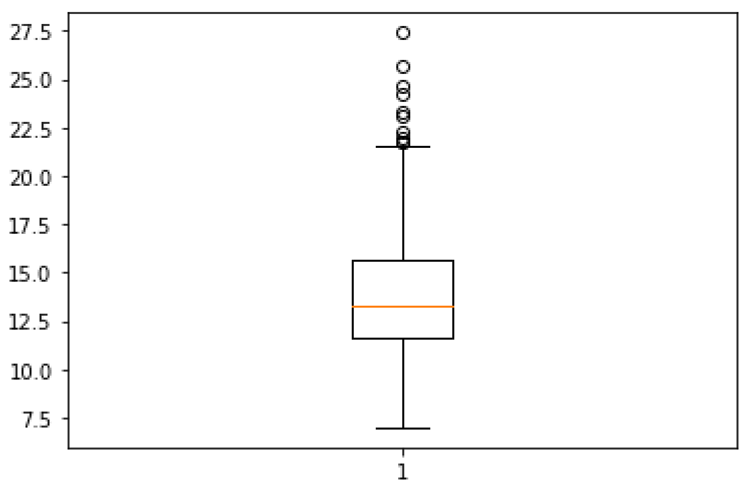
```
398
==========
['B' 'B' 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'M' 'B'
 'B' 'B' 'B' 'B' 'B' 'B' 'M' 'B' 'M' 'M' 'M' 'B' 'B' 'M' 'M' 'B' 'M' 'M'
 'B' 'M' 'M' 'M' 'M' 'B' 'B' 'B' 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'B' 'M' 'B'
 'B' 'B' 'M' 'M' 'B' 'B' 'B' 'M' 'M' 'B' 'B' 'B' 'M' 'M' 'B' 'B' 'M' 'B'
 'B' 'B' 'B' 'M' 'B' 'B' 'B' 'M' 'M' 'M' 'M' 'B' 'B' 'B' 'B' 'B' 'B' 'B'
 'B' 'B' 'B' 'B' 'M' 'B' 'M' 'B' 'M' 'B' 'B' 'M' 'B' 'B' 'B' 'B' 'M' 'B'
 'B' 'M' 'B' 'M' 'M' 'B' 'B' 'B' 'M' 'B' 'M' 'B' 'B' 'B' 'B' 'M' 'B' 'B'
 'B' 'B' 'M' 'M' 'M' 'B' 'B' 'B' 'M' 'B' 'M' 'B' 'B' 'M' 'B' 'B' 'M' 'B'
 'B' 'M' 'M' 'B' 'B' 'B' 'B' 'B' 'B' 'M' 'M' 'B' 'M' 'M' 'B' 'M' 'M' 'M'
 'M' 'B' 'B' 'B' 'M' 'M' 'M' 'M' 'M' 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'B' 'B'
 'M' 'B' 'M' 'B' 'B' 'B' 'M' 'M' 'B' 'M' 'B' 'B' 'M' 'B' 'B' 'B' 'M' 'M'
 'B' 'M' 'B' 'B' 'M' 'B' 'M' 'B' 'B' 'B' 'B' 'M' 'B' 'B' 'M' 'M' 'M' 'M'
 'M' 'B' 'B' 'B' 'M' 'B' 'B' 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'M' 'B' 'B' 'B'
 'B' 'B' 'M' 'B' 'M' 'B' 'M' 'B' 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'B' 'M' 'M'
 'B' 'B' 'B' 'B' 'M' 'B' 'B' 'M' 'M' 'M' 'B' 'B' 'M' 'M' 'B' 'B' 'M' 'B'
 'B' 'B' 'B' 'M' 'B' 'M' 'B' 'B' 'B' 'M' 'B' 'B' 'M' 'B' 'M' 'B' 'B' 'B'
 'M' 'B' 'M' 'M' 'M' 'M' 'M' 'M' 'B' 'B' 'B' 'M' 'B' 'B' 'B' 'B' 'M' 'B'
 'B' 'B' 'B' 'M' 'B' 'M' 'M' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'M' 'M'
 'M' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'M' 'M' 'M' 'M' 'B' 'B' 'M' 'B'
 'B' 'M' 'B' 'B' 'M' 'M' 'M' 'M' 'M' 'M' 'B' 'M' 'M' 'B' 'B' 'B' 'B' 'M'
 'M' 'B' 'B' 'M' 'M' 'B' 'M' 'B' 'B' 'M' 'B' 'B' 'M' 'B' 'B' 'M' 'M' 'M'
 'M' 'B' 'M' 'M' 'M' 'M' 'B' 'B' 'B' 'M' 'B' 'B' 'B' 'M' 'B' 'B' 'M' 'B'
 'B' 'B']
-----------------------------
```

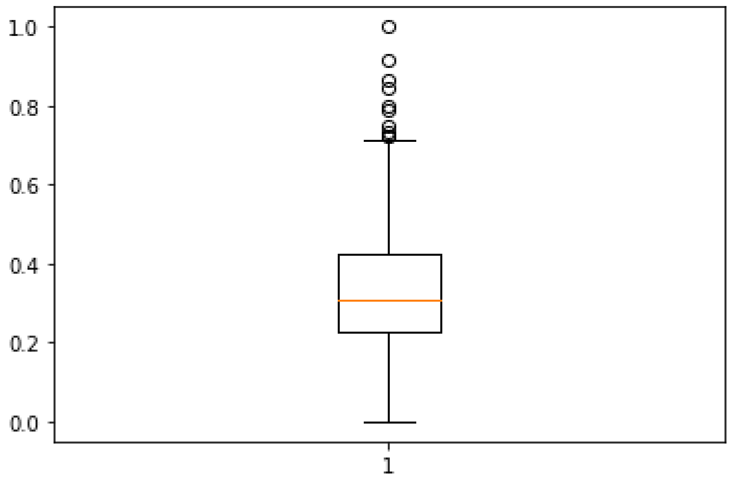
```
171
==========
['M' 'B' 'B' 'M' 'B' 'M' 'B' 'M' 'B' 'M' 'B' 'B' 'B' 'B' 'M' 'B' 'B' 'B'
 'B' 'B' 'M' 'M' 'B' 'B' 'B' 'B' 'M' 'B' 'B' 'M' 'M' 'B' 'M' 'B' 'M' 'M'
 'B' 'M' 'B' 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'M' 'B' 'B' 'M' 'M' 'B' 'B' 'B'
 'B' 'M' 'M' 'B' 'M' 'M' 'B' 'B' 'M' 'B' 'B' 'B' 'M' 'B' 'B' 'B' 'M' 'M'
 'M' 'B' 'M' 'B' 'B' 'B' 'M' 'B' 'B' 'B' 'B' 'M' 'B' 'M' 'B' 'M' 'B' 'B'
 'B' 'B' 'M' 'M' 'M' 'B' 'B' 'B' 'B' 'B' 'M' 'M' 'B' 'B' 'B' 'M' 'B' 'B'
 'B' 'M' 'B' 'M' 'M' 'B' 'B' 'M' 'B' 'B' 'B' 'B' 'M' 'B' 'M' 'B' 'M' 'M'
 'M' 'M' 'B' 'B' 'B' 'B' 'M' 'B' 'B' 'B' 'M' 'B' 'M' 'B' 'B' 'B' 'B' 'M'
 'M' 'B' 'M' 'B' 'M' 'B' 'M' 'B' 'B' 'M' 'B' 'B' 'B' 'M' 'B' 'M' 'B' 'B'
 'M' 'B' 'M' 'M' 'B' 'M' 'B' 'B' 'B']
```

3)Preprocessing:

1-train_data normalization:

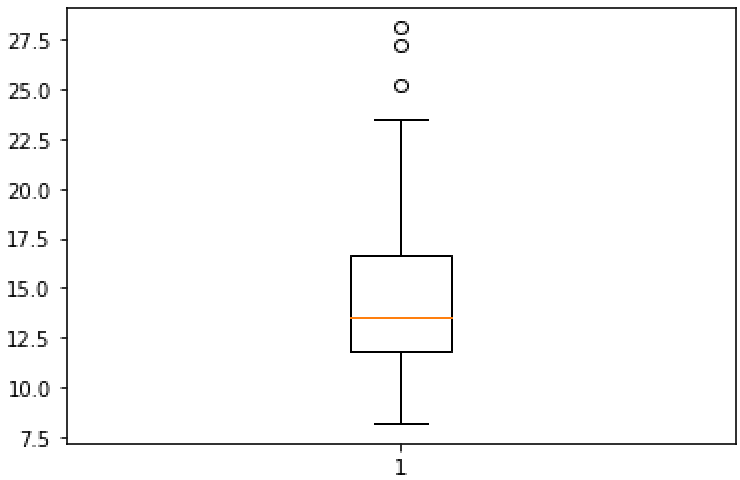
```
[[2.39579978e-02 1.91521396e-02 2.60819693e-02 ... 2.23513382e-02
  3.76943844e-02 2.14313989e-02]
 [5.01014308e-02 3.06340292e-02 2.34491056e-02 ... 2.43689618e-02
  7.25161987e-02 3.35892970e-02]
 [1.56582095e-01 1.23021730e-01 1.66106841e-01 ... 1.45041192e-01
  2.36447084e-01 1.36255992e-01]
 ...
 [2.56258955e-04 1.63945093e-04 1.06063345e-04 ... 1.42567114e-04
  0.00000000e+00 8.56314181e-05]
 [5.62803522e-04 4.12693261e-04 4.99712704e-04 ... 6.28685995e-04
  1.58315335e-03 5.07016346e-04]
 [1.92489483e-04 1.22810793e-04 1.80846337e-04 ... 1.61899616e-04
  5.06587473e-04 1.38777786e-04]]
```

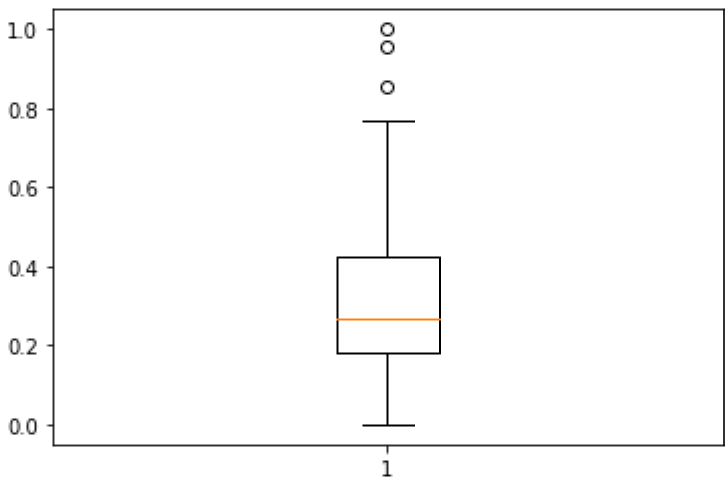




2-test_data normalization:

```
[[8.46308341e-03 2.08336495e-02 2.11903117e-02 ... 1.63700773e-02
  1.95999797e-02 2.09191874e-02]
 [6.79952478e-03 2.94042092e-02 3.63556595e-02 ... 1.38397844e-02
  3.67386027e-02 2.60812706e-02]
 [5.66223285e-02 1.35354337e-01 1.36592252e-01 ... 1.05251976e-01
  1.25576294e-01 1.33781875e-01]
 ...
 [8.27279935e-05 1.00420268e-04 1.42625102e-04 ... 1.09192860e-04
  7.31529246e-05 9.17427361e-05]
 [8.79518786e-05 5.09845394e-04 5.77425613e-04 ... 2.46965165e-04
  3.16646609e-04 4.52814157e-04]
 [2.42764511e-05 1.10695291e-04 1.57699639e-04 ... 7.83575960e-05
  8.54181345e-05 1.32987952e-04]]
```





4)Classification:

1-classification models and evaluation:

a)Decision Tree:

```
from sklearn import metrics

clf = DecisionTreeClassifier()

clf = clf.fit(data_train,Class_train)

Class_pred = clf.predict(data_test)

#print(clf.get_depth)

print("Accuracy:",metrics.accuracy_score(Class_test, Class_pred))
```

```
Accuracy: 0.9298245614035088
```

```
from sklearn.tree import DecisionTreeClassifier
from sklearn import metrics

clf = DecisionTreeClassifier(max_depth=5)

clf = clf.fit(data_train,Class_train)

Class_pred = clf.predict(data_test)

print("Accuracy:",metrics.accuracy_score(Class_test, Class_pred))
#print(metrics.confusion_matrix(Class_test, Class_pred))
#print(metrics.f1_score(Class_test, Class_pred))
```

```
Accuracy: 0.9707602339181286
```

```
clf = DecisionTreeClassifier()

clf = clf.fit(data_train,Class_train)

Class_pred = clf.predict(data_test)

#print("Accuracy:",metrics.accuracy_score(Class_test, Class_pred))
print(metrics.confusion_matrix(Class_test, Class_pred))
```

```
[[94 13]
 [ 6 58]]
```

```python
from sklearn import metrics

clf = DecisionTreeClassifier(max_depth=5)

clf = clf.fit(data_train,Class_train)

Class_pred = clf.predict(data_test)

#print("Accuracy:",metrics.accuracy_score(Class_test, Class_pred))
print(metrics.confusion_matrix(Class_test, Class_pred))
#print(metrics.f1_score(Class_test, Class_pred))
```

```
[[101   6]
 [  6  58]]
```

```python
clf = DecisionTreeClassifier(max_depth=5)

clf = clf.fit(data_train,Class_train)

Class_pred = clf.predict(data_test)

#print("Accuracy:",metrics.accuracy_score(Class_test, Class_pred))
#print(metrics.confusion_matrix(Class_test, Class_pred))
print(metrics.classification_report(Class_test, Class_pred,target_names=['M','B']))
```

```
              precision    recall  f1-score   support

           M       0.95      0.98      0.96       107
           B       0.97      0.91      0.94        64

    accuracy                           0.95       171
   macro avg       0.96      0.94      0.95       171
weighted avg       0.95      0.95      0.95       171
```

b)AdaBoost:

```python
# Create adaboost classifer object
abc = AdaBoostClassifier(n_estimators=50,learning_rate=1)
# Train Adaboost Classifer
model = abc.fit(data_train, Class_train)

#Predict the response for test dataset
Class_pred = model.predict(data_test)

# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(Class_test, Class_pred))
```

```
Accuracy: 0.9649122807017544
```

```
abc = AdaBoostClassifier(n_estimators=50,learning_rate=1)
# Train Adaboost Classifer
model = abc.fit(data_train, Class_train)

#Predict the response for test dataset
Class_pred = model.predict(data_test)

# Model Accuracy, how often is the classifier correct?
#print("Accuracy:",metrics.accuracy_score(Class_test, Class_pred))
print(metrics.confusion_matrix(Class_test, Class_pred))
```

```
[[104   3]
 [  3  61]]
```

```
from sklearn.ensemble import AdaBoostClassifier
# Create adaboost classifer object
abc = AdaBoostClassifier(n_estimators=50,learning_rate=1)
# Train Adaboost Classifer
model = abc.fit(data_train, Class_train)

#Predict the response for test dataset
Class_pred = model.predict(data_test)

# Model Accuracy, how often is the classifier correct?
#print("Accuracy:",metrics.accuracy_score(Class_test, Class_pred))
print(metrics.confusion_matrix(Class_test, Class_pred))
print(metrics.classification_report(Class_test, Class_pred,target_names=['M','B']))
```

```
[[104   3]
 [  3  61]]
              precision    recall  f1-score   support

           M       0.97      0.97      0.97       107
           B       0.95      0.95      0.95        64

    accuracy                           0.96       171
   macro avg       0.96      0.96      0.96       171
weighted avg       0.96      0.96      0.96       171
```

- **base_estimator:** It is a weak learner used to train the model. It uses DecisionTreeClassifier as default weak learner for training purpose. You can also specify different machine learning algorithms.
- **n_estimators:** Number of weak learners to train iteratively.
- **learning_rate:** It contributes to the weights of weak learners. It uses 1 as a default value.

c)Random Forest:

```
from sklearn.ensemble import RandomForestClassifier

clf=RandomForestClassifier(n_estimators=100)

clf.fit(data_train, Class_train)

Class_pred=clf.predict(data_test)

print("Accuracy:",metrics.accuracy_score(Class_test, Class_pred))
```

```
Accuracy: 0.9707602339181286
```

```
clf=RandomForestClassifier(n_estimators=100)

clf.fit(data_train, Class_train)

Class_pred=clf.predict(data_test)

#print("Accuracy:",metrics.accuracy_score(Class_test, Class_pred))
print(metrics.confusion_matrix(Class_test, Class_pred))
```

```
[[105   2]
 [  4  60]]
```

```
clf=RandomForestClassifier(n_estimators=100)

clf.fit(data_train, Class_train)

Class_pred=clf.predict(data_test)

#print("Accuracy:",metrics.accuracy_score(Class_test, Class_pred))
print(metrics.confusion_matrix(Class_test, Class_pred))
print(metrics.classification_report(Class_test, Class_pred,target_names=['M','B']))
```

```
[[107   0]
 [  4  60]]
              precision    recall  f1-score   support

           M       0.96      1.00      0.98       107
           B       1.00      0.94      0.97        64

    accuracy                           0.98       171
   macro avg       0.98      0.97      0.97       171
weighted avg       0.98      0.98      0.98       171
```

```python
from sklearn.ensemble import RandomForestClassifier

clf=RandomForestClassifier(n_estimators=100,max_depth=5)

clf.fit(data_train, Class_train)

Class_pred=clf.predict(data_test)

print("Accuracy:",metrics.accuracy_score(Class_test, Class_pred))
print(metrics.confusion_matrix(Class_test, Class_pred))
print(metrics.classification_report(Class_test, Class_pred,target_names=['M','B']))
```

```
Accuracy: 0.9707602339181286
[[106   1]
 [  4  60]]
              precision    recall  f1-score   support

           M       0.96      0.99      0.98       107
           B       0.98      0.94      0.96        64

    accuracy                           0.97       171
   macro avg       0.97      0.96      0.97       171
weighted avg       0.97      0.97      0.97       171
```

* some equations:

$$\text{Precision} = \frac{\textit{True Positive}}{\textit{True Positive} + \textit{False Positive}}$$

$$\text{Recall} = \frac{\textit{True Positive}}{\textit{True Positive} + \textit{False Negative}}$$

$$\text{F1} = 2 \times \frac{\textit{Precision} * \textit{Recall}}{\textit{Precision} + \textit{Recall}}$$

1-Hyper-parameter tuning:

```
DTC_params = {
        'max_depth' : [4,5,6,7,8,9,10]
}

ABC_params = {
    'n_estimators': [10, 50, 80, 100]
}

RFC_params = {
    'n_estimators': [10, 50, 80, 100],
    'max_depth' : [4,5,6,7,8,9,10]
}

CV_DTC = GridSearchCV(estimator=DTC_clf, param_grid=DTC_params, cv= 10)
CV_DTC.fit(data_train, Class_train)
print("DTC_best_parameters",CV_DTC.best_params_ )

CV_ABC = GridSearchCV(estimator=ABC_clf, param_grid=ABC_params, cv= 10)
CV_ABC.fit(data_train, Class_train)
print("ABC_best_parameters",CV_ABC.best_params_ )

CV_RFC = GridSearchCV(estimator=RFC_clf, param_grid=RFC_params, cv= 10)
CV_RFC.fit(data_train, Class_train)
print("RFC_best_parameters",CV_RFC.best_params_ )
```

```
DTC_best_parameters {'max_depth': 10}
ABC_best_parameters {'n_estimators': 100}
RFC_best_parameters {'max_depth': 6, 'n_estimators': 100}
```