Data Mining
Lab 03

Names:

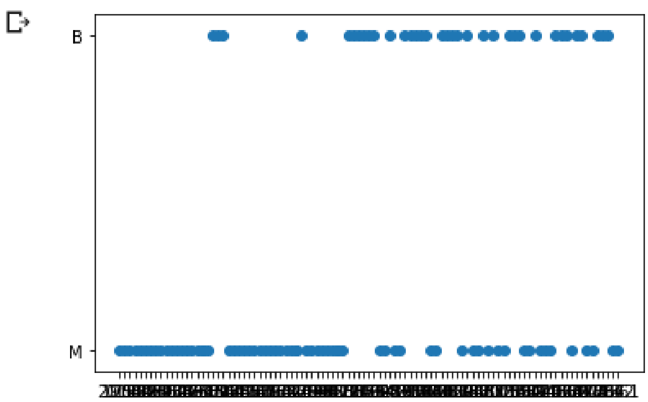
1) Ahmed Mohamed EL-Bawab (08)

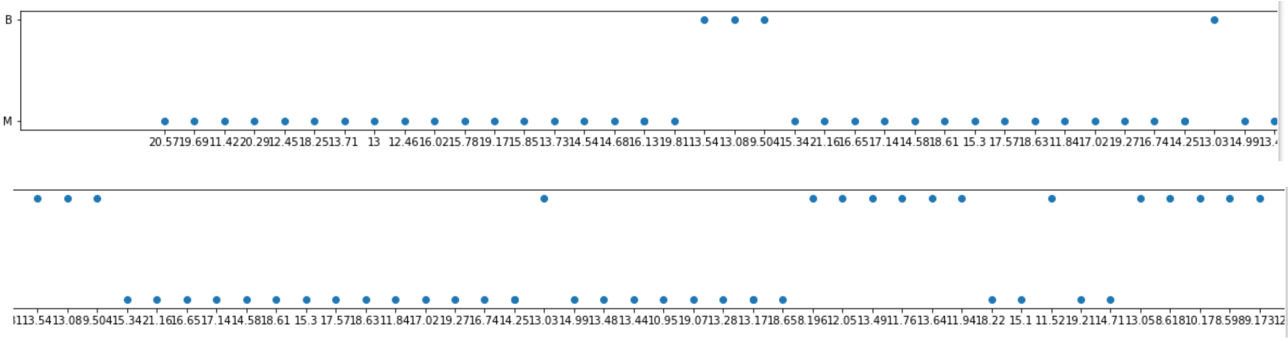
2) Khalil Ismail Khalil (23)

1) Visualization:

a) scatter plot:

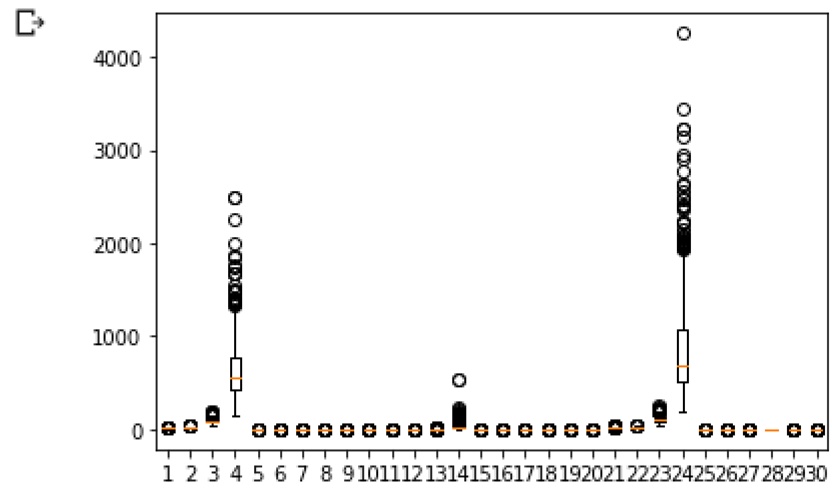
```
data1 = dataFrame.loc[1:100,2]
Class1 = dataFrame.loc[1:100,1]
plt.scatter(data1,Class1)
plt.show()
```

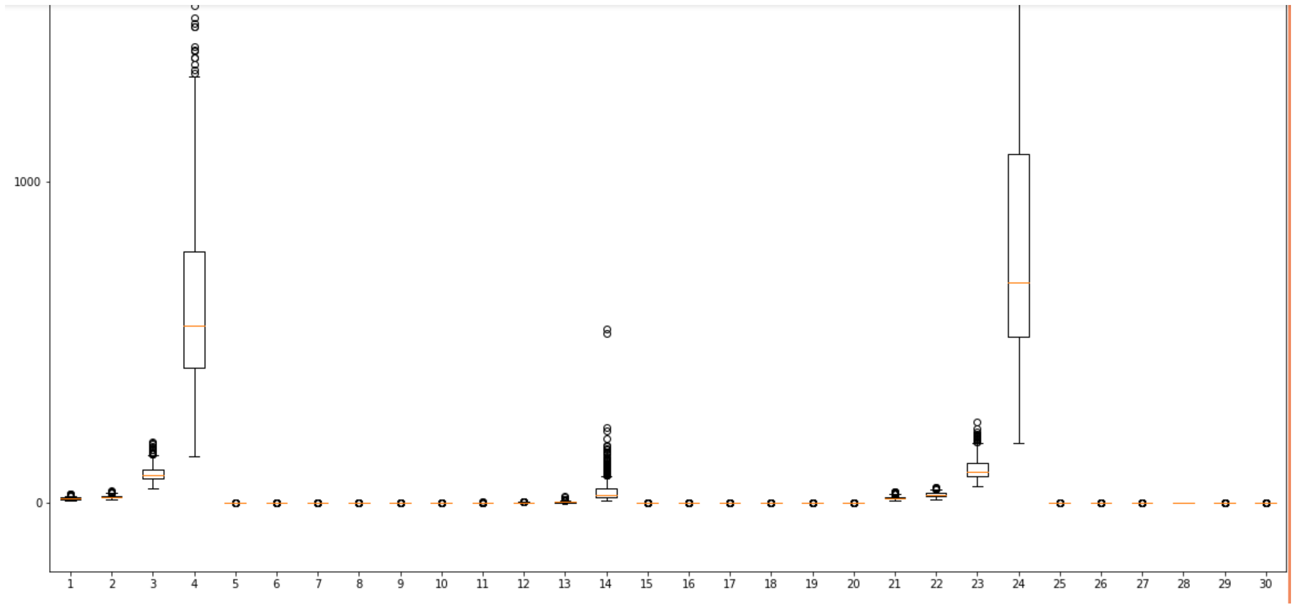




b) box plot:

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

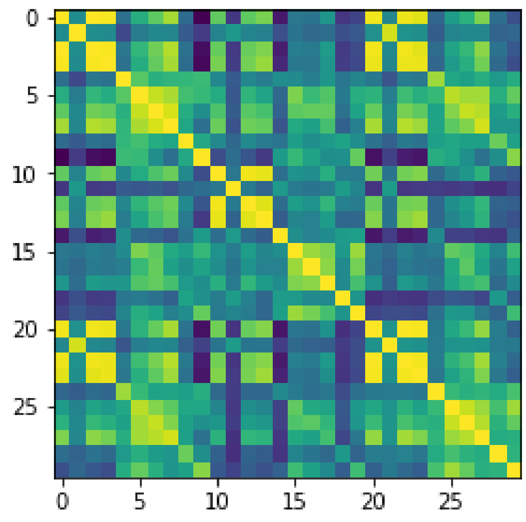




c) correlation matrix imshow:

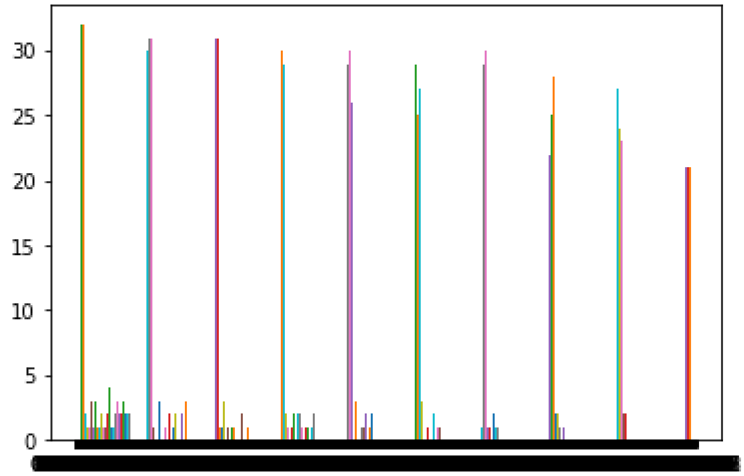
```
#Visualize correlationMatrix using imshow
plt.imshow(correlationMatrix)
```

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

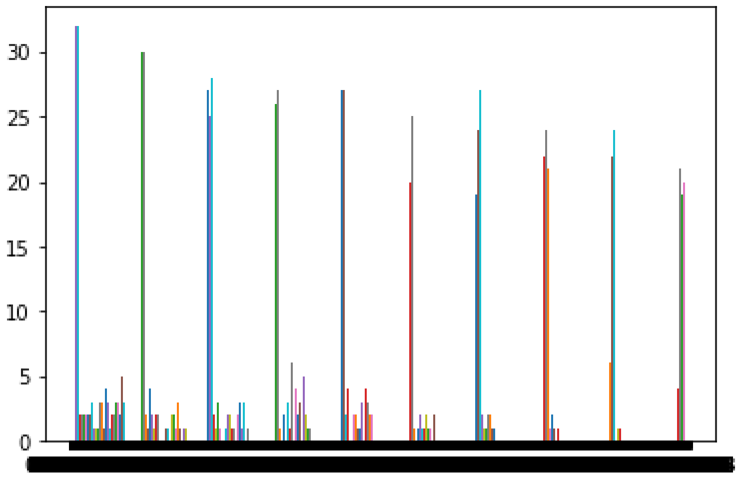


<u>d) histogram plot:</u>

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

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



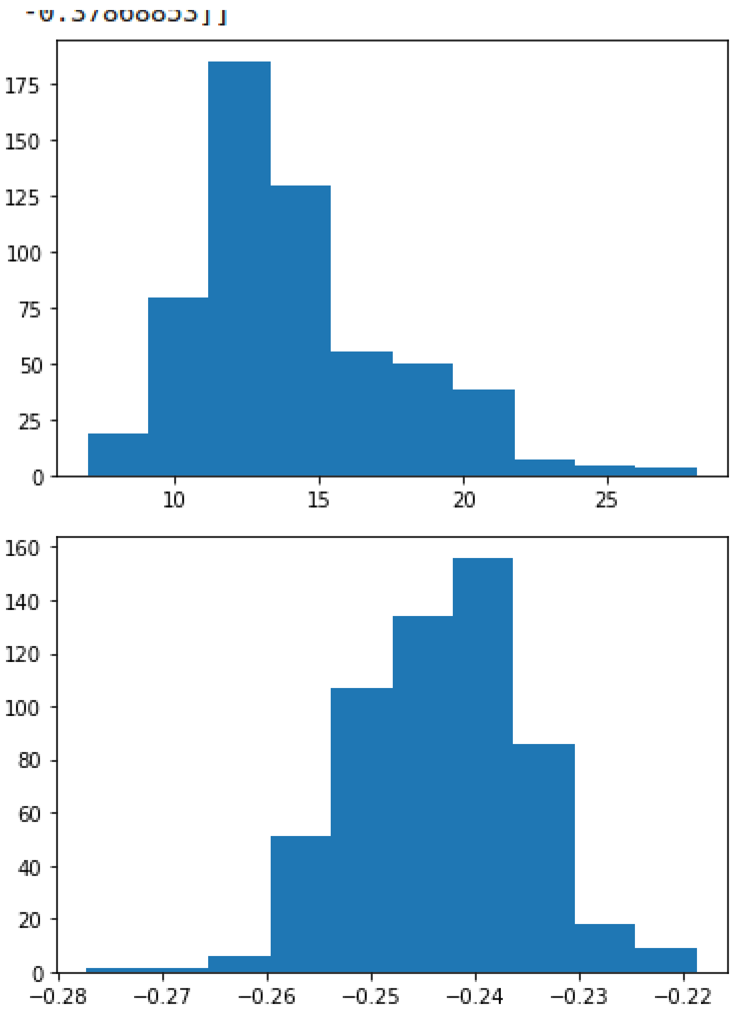

```
tmp = dataFrame.loc[dataFrame[1] == 'B',:]

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



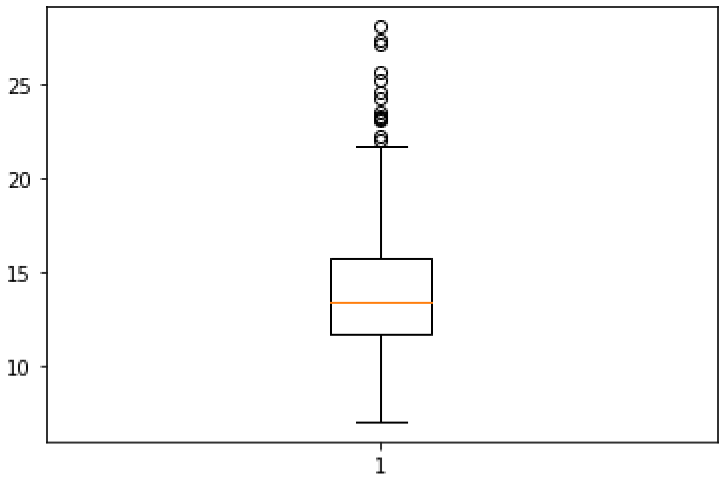
2) Preprocessing:

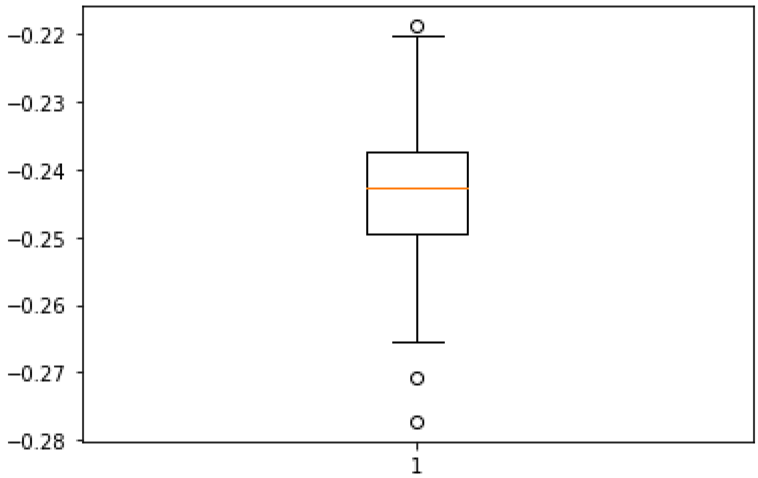
a) z-score normalization:

- histogram plot before and after normalization:



- box plot before and after normalization:





b) feature projection:

- using 2 principal components:

	principal component 1	principal component 2
0	1.175780	0.163033
1	0.473196	-0.174642
2	0.371859	-0.172990
3	0.267926	0.317152
4	-0.026473	-0.332718
...	...	...
564	0.312665	-0.257312
565	0.287031	-0.200130
566	0.112614	-0.126077
567	0.382942	-0.106956
568	0.160593	0.705579

569 rows × 2 columns

3) Dataset Splitting:

- train data:

```
398
==========
[[-2.52367402e-01  1.31362161e-01]
 [ 1.74684380e-01  1.18320053e-01]
 [-3.39207919e-01  3.19777214e-02]
 [-2.66675149e-01 -3.55008044e-01]
 [-9.89672226e-02  1.06157215e-01]
 [-2.73711164e-01  1.51105680e-01]
 [-3.29125593e-01 -9.34484617e-02]
 [-3.16477990e-01 -9.19519647e-02]
 [-1.71499929e-01 -2.86985231e-02]
 [-7.72540681e-02 -9.52354586e-02]
 [ 5.49623504e-01  2.14200286e-02]
 [-4.48583694e-02 -2.74671709e-01]
 [-2.44015977e-01 -1.85859386e-01]
 [-1.71192559e-03 -1.22399715e-01]
 [-1.51908677e-01  4.26214378e-02]
 [-4.64962257e-01 -1.26579438e-01]
 [ 9.26545770e-02 -2.28527210e-01]
 [ 1.43952423e-01 -3.14979050e-02]
 [-1.70240668e-01  3.24300178e-01]
 [-1.23862510e-01 -4.18755740e-02]
```

- test data:

```
171
==========
[[ 4.27930874e-01 -2.77828688e-01]
 [-2.30242091e-01 -4.34038291e-02]
 [ 2.42563390e-02  1.44676543e-01]
 [ 4.73196201e-01 -1.74642230e-01]
 [ 7.12221831e-02  6.75675837e-02]
 [ 1.19517543e+00  1.17461818e-01]
 [-4.53451229e-01 -5.13930923e-02]
 [ 3.09581722e-01 -7.36074752e-02]
 [ 1.05234413e-01 -8.44659161e-02]
 [ 2.00576244e-01  1.42477781e-01]
 [ 8.67042118e-02 -8.60133729e-02]
 [-2.25184083e-01 -4.52836192e-02]
 [-3.75463555e-02  3.67588105e-01]
 [-3.27640110e-01 -7.40292039e-02]
 [-1.45527629e-01 -5.77327912e-02]
 [-3.52102757e-01  3.46809300e-02]
 [-2.90977558e-01  1.40487472e-01]
 [-5.14694435e-02 -5.53814537e-02]
 [-8.21152163e-02  1.21642084e-01]
 [-9.32464748e-02  2.19603535e-01]
```

- train class:

```
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']
-----------------------------
```

- test class:

```
-----------------------------
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']
```

4) Classification:

a) KN-model:

- before using cross validation (best parameters):

```
#create K-neighbours model
from sklearn import metrics
from sklearn.neighbors import KNeighborsClassifier
KN_model = KNeighborsClassifier(n_neighbors=3)
# Train the model using the training sets
KN_model.fit(data_train,Class_train)#Predict Output
KN_predicted= KN_model.predict(data_test)

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

```
Accuracy: 0.8888888888888888
[[99  8]
 [11 53]]
              precision    recall  f1-score   support

           M       0.90      0.93      0.91       107
           B       0.87      0.83      0.85        64

    accuracy                           0.89       171
   macro avg       0.88      0.88      0.88       171
weighted avg       0.89      0.89      0.89       171
```

- after using cross validation (best parameters):

```
#create K-neighbours model
from sklearn import metrics
from sklearn.neighbors import KNeighborsClassifier
KN_model = KNeighborsClassifier(n_neighbors=8)
# Train the model using the training sets
KN_model.fit(data_train,Class_train)
#Predict Output
KN_predicted= KN_model.predict(data_test)

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

```
Accuracy: 0.9122807017543859
[[103   4]
 [ 11  53]]
              precision    recall  f1-score   support

           M       0.90      0.96      0.93       107
           B       0.93      0.83      0.88        64

    accuracy                           0.91       171
   macro avg       0.92      0.90      0.90       171
weighted avg       0.91      0.91      0.91       171
```

b) SVM-model:

- before using cross validation (best parameters):

```
#create SVM model
#Import svm model
from sklearn import svm
#Create a svm Classifier
SVM_model = svm.SVC(kernel='linear',C=1) # Linear Kernel
#Train the model using the training sets
SVM_model.fit(data_train, Class_train)
#Predict the response for test dataset
SVM_pred = SVM_model.predict(data_test)

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

```
Accuracy: 0.8947368421052632
[[103   4]
 [ 14  50]]
              precision    recall  f1-score   support

           M       0.88      0.96      0.92       107
           B       0.93      0.78      0.85        64

    accuracy                           0.89       171
   macro avg       0.90      0.87      0.88       171
weighted avg       0.90      0.89      0.89       171
```

- after using cross validation (best parameters):

```
#create SVM model
#Import svm model
from sklearn import svm
#Create a svm Classifier
SVM_model = svm.SVC(kernel='linear',C=1) # Linear Kernel
#Train the model using the training sets
SVM_model.fit(data_train, Class_train)
#Predict the response for test dataset
SVM_pred = SVM_model.predict(data_test)

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

```
Accuracy: 0.8947368421052632
[[103   4]
 [ 14  50]]
              precision    recall  f1-score   support

           M       0.88      0.96      0.92       107
           B       0.93      0.78      0.85        64

    accuracy                           0.89       171
   macro avg       0.90      0.87      0.88       171
weighted avg       0.90      0.89      0.89       171
```

c) SVC-model:

- before using cross validation (best parameters):

```
#create SVC model

# Create a SVC classifier using an RBF kernel
SVC_model = svm.SVC(kernel='rbf', random_state=0, gamma=.01, C=1)
# Train the classifier
SVC_model.fit(data_train, Class_train)# Visualize the decision boundaries
#Predict the response for test dataset
SVC_pred = SVC_model.predict(data_test)

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

```
Accuracy: 0.7017543859649122
[[107   0]
 [ 51  13]]
              precision    recall  f1-score   support

           M       0.68      1.00      0.81       107
           B       1.00      0.20      0.34        64

    accuracy                           0.70       171
   macro avg       0.84      0.60      0.57       171
weighted avg       0.80      0.70      0.63       171
```

- after using cross validation (best parameters):

```
#create SVC model

# Create a SVC classifier using an RBF kernel
SVC_model = svm.SVC(kernel='rbf', random_state=0, gamma=1, C=10)
# Train the classifier
SVC_model.fit(data_train, Class_train)# Visualize the decision boundaries
#Predict the response for test dataset
SVC_pred = SVC_model.predict(data_test)

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

```
Accuracy: 0.9239766081871345
[[103   4]
 [  9  55]]
              precision    recall  f1-score   support

           M       0.92      0.96      0.94       107
           B       0.93      0.86      0.89        64

    accuracy                           0.92       171
   macro avg       0.93      0.91      0.92       171
weighted avg       0.92      0.92      0.92       171
```

d) Logistic Regression-model:

- before using cross validation (best parameters):

```
#create logistic regression model
# import the class
from sklearn.linear_model import LogisticRegression
# instantiate the model (using the default parameters)
LogReg_model = LogisticRegression(C=1)
# fit the model with data
LogReg_model.fit(data_train,Class_train)#
LogReg_pred=LogReg_model.predict(data_test)

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

```
Accuracy: 0.8947368421052632
[[103   4]
 [ 14  50]]
              precision    recall  f1-score   support

           M       0.88      0.96      0.92       107
           B       0.93      0.78      0.85        64

    accuracy                           0.89       171
   macro avg       0.90      0.87      0.88       171
weighted avg       0.90      0.89      0.89       171
```

- after using cross validation (best parameters):

```
#create logistic regression model
# import the class
from sklearn.linear_model import LogisticRegression
# instantiate the model (using the default parameters)
LogReg_model = LogisticRegression(C=10000)
# fit the model with data
LogReg_model.fit(data_train,Class_train)#
LogReg_pred=LogReg_model.predict(data_test)

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

```
Accuracy: 0.9064327485380117
[[102   5]
 [ 11  53]]
              precision    recall  f1-score   support

           M       0.90      0.95      0.93       107
           B       0.91      0.83      0.87        64

    accuracy                           0.91       171
   macro avg       0.91      0.89      0.90       171
weighted avg       0.91      0.91      0.91       171
```

5) Cross Validation (best parameters of models):

```
KN_best_parameters {'n_neighbors': 8}
SVM_best_parameters {'C': 1}
SVC_best_parameters {'C': 10, 'gamma': 1}
LogReg_best_parameters {'C': 10000}
```