The results when the train/test split is set to the default 67/33 cut:

a. Random Forest Classifier

In [14]:
```
# training the model using the training data
model.fit(X_train,Y_train)
```

Out[14]:
```
RandomForestClassifier()
```

In [15]:
```
# test the model using the test input only
prediction = model.predict(x_test)
```

In [16]:
```
# calcualting and printing the accuracy
model_acc = accuracy_score(y_test, prediction)*100
print(model_acc)
```

```
100.0
```

In [17]:
```
# generate classification report
model_cl_rep = metrics.classification_report(y_test, prediction)
print(model_cl_rep)
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00        17

    accuracy                           1.00        27
   macro avg       1.00      1.00      1.00        27
weighted avg       1.00      1.00      1.00        27
```

In [18]:
```
# generate confusion matrix
model_cm = metrics.confusion_matrix(y_test, prediction)
print(model_cm)
```

```
[[10  0]
 [ 0 17]]
```

b. Support Vector Machines

In [20]:
```
# training the model using the training data
model.fit(X_train,Y_train)
```

Out[20]:
```
SVC()
```

In [21]:
```
# test the model using the test input only
prediction_SVC = model.predict(x_test)
```

In [22]:
```
# calcualting and printing the accuracy
model_acc_SVC = accuracy_score(y_test, prediction_SVC)*100
print(model_acc_SVC)
```

```
77.77777777777779
```

In [23]:
```
# generate classification report
model_cl_rep_SVC = metrics.classification_report(y_test, prediction_SVC)
print(model_cl_rep_SVC)
```

```
              precision    recall  f1-score   support

           0       0.70      0.70      0.70        10
           1       0.82      0.82      0.82        17

    accuracy                           0.78        27
   macro avg       0.76      0.76      0.76        27
weighted avg       0.78      0.78      0.78        27
```

In [24]:
```
# generate confusion matrix
model_cm_SVC = metrics.confusion_matrix(y_test, prediction_SVC)
print(model_cm_SVC)
```

```
[[ 7  3]
 [ 3 14]]
```

c. Multilayer Perceptron Classifier


```
C:\Users\arsal\anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_
astic Optimizer: Maximum iterations (200) reached and the optimization hasn't
  warnings.warn(
```

Out[28]:
```
MLPClassifier()
```

In [29]:
```
# test the model using the test input only
prediction_MLPC = model_MLPC.predict(x_test)
```

In [30]:
```
# calcualting and printing the accuracy
model_acc_MLPC = accuracy_score(y_test, prediction_MLPC)*100
print(model_acc_MLPC)
```

```
88.88888888888889
```

In [31]:
```
# generate classification report
model_cl_rep_MLPC = metrics.classification_report(y_test, prediction_MLPC)
print(model_cl_rep_MLPC)
```

```
              precision    recall  f1-score   support

           0       0.82      0.90      0.86        10
           1       0.94      0.88      0.91        17

    accuracy                           0.89        27
   macro avg       0.88      0.89      0.88        27
weighted avg       0.89      0.89      0.89        27
```

In [32]:
```
# generate confusion matrix
model_cm_MLPC = metrics.confusion_matrix(y_test, prediction_MLPC)
print(model_cm_MLPC)
```

```
[[ 9  1]
 [ 2 15]]
```

The results when the train/test split is set to 80/20 cut:

a. Random Forest Classifier

In [7]:
```
# training the model using the training data
model.fit(X_train,Y_train)
```

Out[7]:
```
RandomForestClassifier()
```

In [8]:
```
# test the model using the test input only
prediction = model.predict(x_test)
```

In [9]:
```
# calcualting and printing the accuracy
model_acc = accuracy_score(y_test, prediction)*100
print(model_acc)
```

```
100.0
```

In [10]:
```
# generate classification report
model_cl_rep = metrics.classification_report(y_test, prediction)
print(model_cl_rep)
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         6
           1       1.00      1.00      1.00        10

    accuracy                           1.00        16
   macro avg       1.00      1.00      1.00        16
weighted avg       1.00      1.00      1.00        16
```

In [11]:
```
# generate confusion matrix
model_cm = metrics.confusion_matrix(y_test, prediction)
print(model_cm)
```

```
[[ 6  0]
 [ 0 10]]
```

b. Support Vector Machines

In [13]:
```
# training the model using the training data
model.fit(X_train,Y_train)
```

Out[13]:
```
SVC()
```

In [14]:
```
# test the model using the test input only
prediction_SVC = model.predict(x_test)
```

In [15]:
```
# calcualting and printing the accuracy
model_acc_SVC = accuracy_score(y_test, prediction_SVC)*100
print(model_acc_SVC)
```

```
81.25
```

In [16]:
```
# generate classification report
model_cl_rep_SVC = metrics.classification_report(y_test, prediction_SVC)
print(model_cl_rep_SVC)
```

```
              precision    recall  f1-score   support

           0       0.80      0.67      0.73         6
           1       0.82      0.90      0.86        10

    accuracy                           0.81        16
   macro avg       0.81      0.78      0.79        16
weighted avg       0.81      0.81      0.81        16
```

In [17]:
```
# generate confusion matrix
model_cm_SVC = metrics.confusion_matrix(y_test, prediction_SVC)
print(model_cm_SVC)
```

```
[[4 2]
 [1 9]]
```

c. Multilayer Perceptron Classifier

```
astic Optimizer: Maximum iterations (200) reached and the optimization hasn't
  warnings.warn(
```

Out[19]:
```
MLPClassifier()
```

In [20]:
```
# test the model using the test input only
prediction_MLPC = model_MLPC.predict(x_test)
```

In [21]:
```
# calcualting and printing the accuracy
model_acc_MLPC = accuracy_score(y_test, prediction_MLPC)*100
print(model_acc_MLPC)
```

```
100.0
```

In [22]:
```
# generate classification report
model_cl_rep_MLPC = metrics.classification_report(y_test, prediction_MLPC)
print(model_cl_rep_MLPC)
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         6
           1       1.00      1.00      1.00        10

    accuracy                           1.00        16
   macro avg       1.00      1.00      1.00        16
weighted avg       1.00      1.00      1.00        16
```

In [23]:
```
# generate confusion matrix
model_cm_MLPC = metrics.confusion_matrix(y_test, prediction_MLPC)
print(model_cm_MLPC)
```

```
[[ 6  0]
 [ 0 10]]
```

Q 2.4 Results after removing height and hair length

a. Random Forest Classifier

```
model.fit(X_train,Y_train)
```

Out[27]:
```
RandomForestClassifier()
```

In [28]:
```
# test the model using the test input only
prediction = model.predict(x_test)
```

In [29]:
```
# calcualting and printing the accuracy
model_acc = accuracy_score(y_test, prediction)*100
print(model_acc)
```

```
100.0
```

In [30]:
```
# generate classification report
model_cl_rep = metrics.classification_report(y_test, prediction)
print(model_cl_rep)
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         6
           1       1.00      1.00      1.00        10

    accuracy                           1.00        16
   macro avg       1.00      1.00      1.00        16
weighted avg       1.00      1.00      1.00        16
```

In [31]:
```
# generate confusion matrix
model_cm = metrics.confusion_matrix(y_test, prediction)
print(model_cm)
```

```
[[ 6  0]
 [ 0 10]]
```

b. Support Vector Machines

In [33]:
```
# training the model using the training data
model.fit(X_train,Y_train)
```

Out[33]:
```
SVC()
```

In [34]:
```
# test the model using the test input only
prediction_SVC = model.predict(x_test)
```

In [35]:
```
# calcualting and printing the accuracy
model_acc_SVC = accuracy_score(y_test, prediction_SVC)*100
print(model_acc_SVC)
```

```
81.25
```

In [36]:
```
# generate classification report
model_cl_rep_SVC = metrics.classification_report(y_test, prediction_SVC)
print(model_cl_rep_SVC)
```

```
              precision    recall  f1-score   support

           0       0.80      0.67      0.73         6
           1       0.82      0.90      0.86        10

    accuracy                           0.81        16
   macro avg       0.81      0.78      0.79        16
weighted avg       0.81      0.81      0.81        16
```

In [37]:
```
# generate confusion matrix
model_cm_SVC = metrics.confusion_matrix(y_test, prediction_SVC)
print(model_cm_SVC)
```

```
[[4 2]
 [1 9]]
```

c. Multilayer Perceptron Classifier

In [38]:
```
# changing the classifier as required
model_MLPC = MLPClassifier()
```

In [39]:
```
# training the model using the training data
model_MLPC.fit(X_train,Y_train)
```

Out[39]:
```
MLPClassifier()
```

In [40]:
```
# test the model using the test input only
prediction_MLPC = model_MLPC.predict(x_test)
```

In [41]:
```
# calcualting and printing the accuracy
model_acc_MLPC = accuracy_score(y_test, prediction_MLPC)*100
print(model_acc_MLPC)
```

```
62.5
```

In [42]:
```
# generate classification report
model_cl_rep_MLPC = metrics.classification_report(y_test, prediction_MLPC)
print(model_cl_rep_MLPC)
```

In [43]:
```
# generate confusion matrix
model_cm_MLPC = metrics.confusion_matrix(y_test, prediction_MLPC)
print(model_cm_MLPC)
```

```
[[ 0  6]
 [ 0 10]]
```

QUESTION 3

a. Monte Carlo Cross Validation

In [4]:
```
# selecting a classifier
# model = RandomForestClassifier()

# create a Gaussian classifier
# model = GaussianNB()

# create a Decision tree classifier
model = DecisionTreeClassifier()
```

In [5]:
```
# separating independent and dependent variables from the data frame
x = list(zip(df_gender.height, df_gender.weight, beard_encd, hair_length_encd, df_gender.shoe_size, scarf_encd, eye_color_er
y = gender_encd
```

In [9]:
```
# for shuffle split model
mc = ShuffleSplit(n_splits = 5,test_size = 0.33,random_state = 7)
```

In [10]:
```
print("Cross validation scores with Monte Carlo Cross Validation")
cross_val_score(model, x, y, cv = mc).mean()
```

```
Cross validation scores with Monte Carlo Cross Validation
```

Out[10]:
```
0.9517241379310345
```

b. Leave P-Out Cross Validation

In [3]:
```
# converting the string based inputs into integer as the model only understands integers
labels = preprocessing.LabelEncoder()
beard_encd = labels.fit_transform(df_gender.beard)
hair_length_encd = labels.fit_transform(df_gender.hair_length)
scarf_encd = labels.fit_transform(df_gender.scarf)
eye_color_encd = labels.fit_transform(df_gender.eye_color)
gender_encd = labels.fit_transform(df_gender.gender)
```

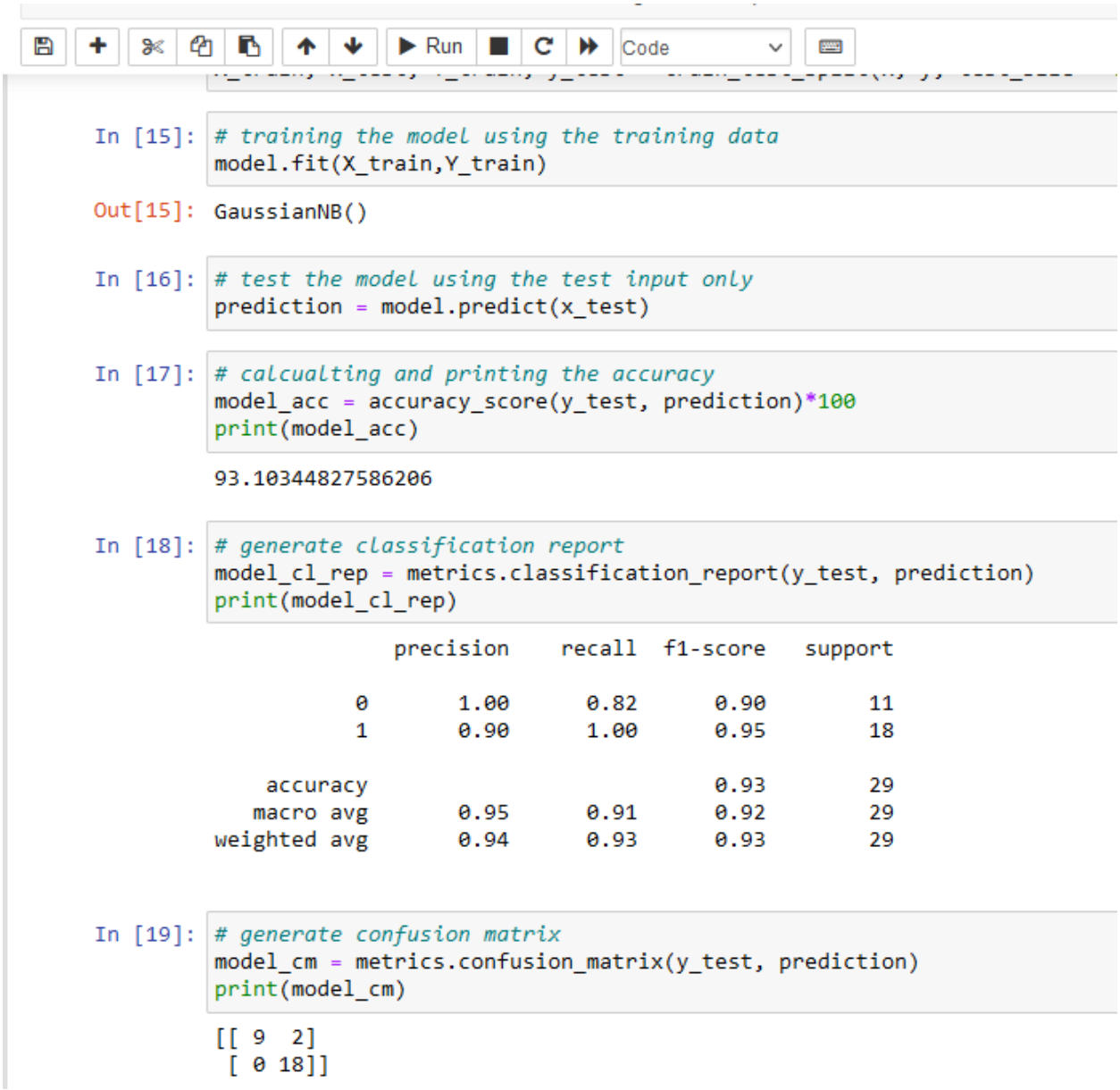
In [4]:
```
# selecting a classifier
# model = RandomForestClassifier()

# create a Gaussian classifier
# model = GaussianNB()

# create a Decision tree classifier
model = DecisionTreeClassifier()
```

In [5]:
```
# separating independent and dependent variables from the data frame
x = list(zip(df_gender.height, df_gender.weight, beard_encd, hair_length_encd, df_gender.shoe_size, scarf_encd, eye_color_er
y = gender_encd
```

In [6]:
```
# Leave POut stuff
lpo = LeavePOut(p=2)
lpo.get_n_splits(x)
tree = RandomForestClassifier(n_estimators = 10, max_depth = 5, n_jobs= -1)
score = cross_val_score(tree, x, y, cv = lpo)
print("Cross Validation Scores are {}".format(score))
print("Average Cross Validation score :{}".format(score.mean()))
```

```
Cross Validation Scores are [1. 1. 1. ... 1. 1. 1.]
Average Cross Validation score :0.9683473389355742
```

QUESTION 4



In [15]:
```
# training the model using the training data
model.fit(X_train,Y_train)
```

Out[15]:
```
GaussianNB()
```

In [16]:
```
# test the model using the test input only
prediction = model.predict(x_test)
```

In [17]:
```
# calcualting and printing the accuracy
model_acc = accuracy_score(y_test, prediction)*100
print(model_acc)
```

```
93.10344827586206
```

In [18]:
```
# generate classification report
model_cl_rep = metrics.classification_report(y_test, prediction)
print(model_cl_rep)
```

```
              precision    recall  f1-score   support

           0       1.00      0.82      0.90        11
           1       0.90      1.00      0.95        18

    accuracy                           0.93        29
   macro avg       0.95      0.91      0.92        29
weighted avg       0.94      0.93      0.93        29
```

In [19]:
```
# generate confusion matrix
model_cm = metrics.confusion_matrix(y_test, prediction)
print(model_cm)
```

```
[[ 9  2]
 [ 0 18]]
```

[End of File]