```
#import libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

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

```
Mounted at /content/drive
```

Double-click (or enter) to edit

Double-click (or enter) to edit

```
# df = cv2.imread('/content/drive/My Drive/BTP/data.csv')
df = pd.read_csv('/content/drive/My Drive/BTP/data.csv')
df.head(7)
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness
0	842302	M	17.99	10.38	122.80	1001.0	0
1	842517	M	20.57	17.77	132.90	1326.0	0.
2	84300903	M	19.69	21.25	130.00	1203.0	0.
3	84348301	M	11.42	20.38	77.58	386.1	0.
4	84358402	M	20.29	14.34	135.10	1297.0	0.
5	843786	M	12.45	15.70	82.57	477.1	0.
6	844359	M	18.25	19.98	119.60	1040.0	0.

7 rows × 33 columns

Double-click (or enter) to edit

```
#Count the number of rows and columns in the data set
df.shape
```

```
(569, 33)
```

```
#Count the empty (NaN, NAN, na) values in each column
df.isna().sum()
```

```
id                         0
diagnosis                  0
radius_mean                0
texture_mean               0
perimeter_mean             0
area_mean                  0
smoothness_mean            0
compactness_mean           0
concavity_mean             0
concave points_mean        0
symmetry_mean              0
fractal_dimension_mean     0
radius_se                  0
texture_se                 0
perimeter_se               0
area_se                    0
smoothness_se              0
compactness_se             0
concavity_se               0
concave points_se          0
symmetry_se                0
fractal_dimension_se       0
radius_worst               0
texture_worst              0
perimeter_worst            0
area_worst                 0
```

```
smoothness_worst            0
compactness_worst           0
concavity_worst             0
concave points_worst        0
symmetry_worst              0
fractal_dimension_worst     0
Unnamed: 32               569
dtype: int64
```

```
#Drop the column with all missing values (na, NAN, NaN)
#NOTE: This drops the column Unnamed
df = df.dropna(axis=1)
```

```
#Get the new count of the number of rows and cols
df.shape
```

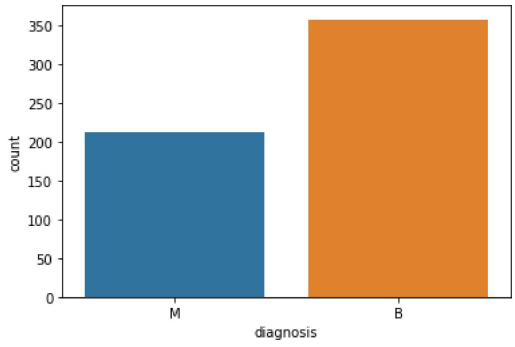
```
(569, 32)
```

```
#Get a count of the number of 'M' & 'B' cells
df['diagnosis'].value_counts()
```

```
B    357
M    212
Name: diagnosis, dtype: int64
```

```
#Visualize this count
sns.countplot(df['diagnosis'],label="Count")
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass th
  FutureWarning
<matplotlib.axes._subplots.AxesSubplot at 0x7faa688d8990>
```




```
#Look at the data types
df.dtypes
```

```
id                        int64
diagnosis                 int64
radius_mean             float64
texture_mean            float64
perimeter_mean          float64
area_mean               float64
smoothness_mean         float64
compactness_mean        float64
concavity_mean          float64
concave points_mean     float64
symmetry_mean           float64
fractal_dimension_mean  float64
radius_se               float64
texture_se              float64
perimeter_se            float64
area_se                 float64
smoothness_se           float64
compactness_se          float64
concavity_se            float64
concave points_se       float64
symmetry_se             float64
fractal_dimension_se    float64
radius_worst            float64
texture_worst           float64
perimeter_worst         float64
area_worst              float64
smoothness_worst        float64
compactness_worst       float64
```

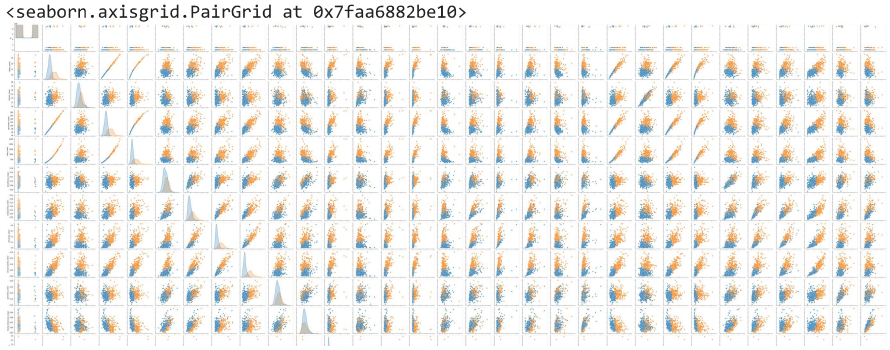
```
concavity_worst            float64
concave points_worst       float64
symmetry_worst             float64
fractal_dimension_worst    float64
dtype: object
```

```
#Encoding categorical data values (
from sklearn.preprocessing import LabelEncoder
labelencoder_Y = LabelEncoder()
df.iloc[:,1]= labelencoder_Y.fit_transform(df.iloc[:,1].values)
print(labelencoder_Y.fit_transform(df.iloc[:,1].values))
```

```
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1
 0 1 1 1 1 1 1 1 1 0 1 0 0 0 0 0 1 1 0 1 1 0 0 0 0 1 0 1 1 0 0 0 0 1 0 1 1
 0 1 0 1 1 0 0 0 1 1 0 1 1 1 0 0 0 1 0 0 1 1 0 0 0 1 1 0 0 0 0 1 0 0 1 0 0
 0 0 0 0 0 0 1 1 1 0 1 1 0 0 0 1 1 0 1 0 1 1 0 1 1 0 0 1 0 0 1 0 0 0 0 1 0
 0 0 0 0 0 0 0 0 1 0 0 0 0 1 1 0 1 0 0 1 1 0 0 1 1 0 0 0 0 1 0 0 1 1 1 0 1
 0 1 0 0 0 1 0 0 1 1 0 1 1 1 1 0 1 1 1 0 1 0 1 0 0 1 0 1 1 1 1 0 0 1 1 0 0
 0 1 0 0 0 0 0 1 1 0 0 1 0 0 1 1 0 1 0 0 0 0 1 0 0 0 0 0 0 1 0 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 0 0 0 0 0 0 1 0 1 0 0 1 0 0 1 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0
 0 1 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 1 0 0 0 0 1 1 1 0 0
 0 0 1 0 1 0 1 0 0 0 1 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 1 1 0 1 1
 1 0 1 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 1 0 0 1 1 0 0 0 0 0 0 1 0 0 0 0 0 0
 0 1 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 0 1 0 0 0 0 0 1 0 0
 1 0 1 0 0 1 0 1 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0
 0 0 0 0 0 0 1 0 1 0 0 1 0 0 0 0 0 0 1 1 0 1 0 1 0 0 0 0 0 1 0 0 1 0 1 0 1 1
 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 1 1 1 1 1 1 0]
```

```
sns.pairplot(df, hue="diagnosis")
```

```
<seaborn.axisgrid.PairGrid at 0x7faa6882be10>
```



```
df.head(5)
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness
0	842302	1	17.99	10.38	122.80	1001.0	0
1	842517	1	20.57	17.77	132.90	1326.0	0.
2	84300903	1	19.69	21.25	130.00	1203.0	0.
3	84348301	1	11.42	20.38	77.58	386.1	0.
4	84358402	1	20.29	14.34	135.10	1297.0	0.

5 rows × 32 columns

```
#Get the correlation of the columns
df.corr()
```

radius_mean	0.074626	0.730029	1.000000	0.323782	0.99785
texture_mean	0.099770	0.415185	0.323782	1.000000	0.3295
perimeter_mean	0.073159	0.742636	0.997855	0.329533	1.0000
area_mean	0.096893	0.708984	0.987357	0.321086	0.9865
smoothness_mean	-0.012968	0.358560	0.170581	-0.023389	0.2072
compactness_mean	0.000096	0.596534	0.506124	0.236702	0.5569
concavity_mean	0.050080	0.696360	0.676764	0.302418	0.7161
concave points_mean	0.044158	0.776614	0.822529	0.293464	0.8509
symmetry_mean	-0.022114	0.330499	0.147741	0.071401	0.1830
fractal_dimension_mean	-0.052511	-0.012838	-0.311631	-0.076437	-0.2614

```
plt.figure(figsize=(20,20))
sns.heatmap(df.corr(), annot=True, fmt='.0%')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7faa54a59050>
```

```
X = df.iloc[:, 2:31].values
Y = df.iloc[:, 1].values
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.25, random_state = 0)
```

```
print(f'The shape of Train data is {X_train.shape}')
print(f'The shape of Test data is {X_test.shape}')
```

```
The shape of Train data is (426, 29)
The shape of Test data is (143, 29)
```

```
#Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

K Nearest Neighbor

```
def models(X_train,Y_train):
  #Using KNeighborsClassifier
  from sklearn.neighbors import KNeighborsClassifier
  knn = KNeighborsClassifier(n_neighbors = 4, metric = 'minkowski', p = 2)
  knn.fit(X_train, Y_train)
  #print model accuracy on the training data.
  print('K Nearest Neighbor Training Accuracy:', knn.score(X_train, Y_train))
  return knn


model = models(X_train,Y_train)
```

```
K Nearest Neighbor Training Accuracy: 0.971830985915493
```

```
def models(X_train,Y_train):
  #Using KNeighborsClassifier
  from sklearn.neighbors import KNeighborsClassifier
  knn = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 2)
  knn.fit(X_train, Y_train)
  #print model accuracy on the training data.
  print('K Nearest Neighbor Training Accuracy:', knn.score(X_train, Y_train))
  return knn


model = models(X_train,Y_train)
```

```
K Nearest Neighbor Training Accuracy: 0.9765258215962441
```

```
def models(X_train,Y_train):
  #Using KNeighborsClassifier
  from sklearn.neighbors import KNeighborsClassifier
  knn = KNeighborsClassifier(n_neighbors = 6, metric = 'minkowski', p = 2)
  knn.fit(X_train, Y_train)
  #print model accuracy on the training data.
  print('K Nearest Neighbor Training Accuracy:', knn.score(X_train, Y_train))
  return knn


model = models(X_train,Y_train)
```

```
      K Nearest Neighbor Training Accuracy: 0.9765258215962441
```

```
def models(X_train,Y_train):
  #Using KNeighborsClassifier
  from sklearn.neighbors import KNeighborsClassifier
  knn = KNeighborsClassifier(n_neighbors = 7, metric = 'minkowski', p = 2)
  knn.fit(X_train, Y_train)
  #print model accuracy on the training data.
  print('K Nearest Neighbor Training Accuracy:', knn.score(X_train, Y_train))
  return knn


model = models(X_train,Y_train)
```

```
      K Nearest Neighbor Training Accuracy: 0.9694835680751174
```

```
def models(X_train,Y_train):
  #Using KNeighborsClassifier
  from sklearn.neighbors import KNeighborsClassifier
  knn = KNeighborsClassifier(n_neighbors = 8, metric = 'minkowski', p = 2)
  knn.fit(X_train, Y_train)
  #print model accuracy on the training data.
  print('K Nearest Neighbor Training Accuracy:', knn.score(X_train, Y_train))
  return knn


model = models(X_train,Y_train)
```

```
      K Nearest Neighbor Training Accuracy: 0.9765258215962441
```

```
def models(X_train,Y_train):
  #Using KNeighborsClassifier
  from sklearn.neighbors import KNeighborsClassifier
  knn = KNeighborsClassifier(n_neighbors = 9, metric = 'minkowski', p = 2)
  knn.fit(X_train, Y_train)
  #print model accuracy on the training data.
  print('K Nearest Neighbor Training Accuracy:', knn.score(X_train, Y_train))
  return knn


model = models(X_train,Y_train)
```

```
      K Nearest Neighbor Training Accuracy: 0.9765258215962441
```

```
def models(X_train,Y_train):
  #Using KNeighborsClassifier
  from sklearn.neighbors import KNeighborsClassifier
  knn = KNeighborsClassifier(n_neighbors = 10, metric = 'minkowski', p = 2)
  knn.fit(X_train, Y_train)
  #print model accuracy on the training data.
  print('K Nearest Neighbor Training Accuracy:', knn.score(X_train, Y_train))
  return knn


model = models(X_train,Y_train)
```

```
      K Nearest Neighbor Training Accuracy: 0.971830985915493
```

Confusion Matrix

```
from sklearn.metrics import confusion_matrix
#for i in range(len(model)):
i=0
cm = confusion_matrix(Y_test, model.predict(X_test))

TN = cm[0][0]
TP = cm[1][1]
FN = cm[1][0]
FP = cm[0][1]

print(cm)
print('Model[{}] Testing Accuracy = "{}!"'.format(i,(TP + TN) / (TP + TN + FN + FP)))
print()# Print a new line
```

```
[[88  2]
 [ 6 47]]
Model[0] Testing Accuracy = "0.9440559440559441!"
```

Naive Bayes

```
# M = df[df.diagnosis == "M"]


# B = df[df.diagnosis == "B"]


# plt.title("Malignant vs Benign Tumor")
# plt.xlabel("Radius Mean")
# plt.ylabel("Texture Mean")
# plt.scatter(M.radius_mean, M.texture_mean, color = "red", label = "Malignant", alpha = 0.3)
# plt.scatter(B.radius_mean, B.texture_mean, color = "lime", label = "Benign", alpha = 0.3)
# plt.legend()
# plt.show()


# # df.diagnosis = [1 if i== "M" else 0 for i in df.diagnosis]
# x = df.drop(["diagnosis"], axis = 1)
# y = df.diagnosis.values


# Normalization:
# import numpy as np
# from numpy import *
# x = (x - np.min(x)) / (np.max(x) - np.min(x))
# x = (x - min(x)) / (max(x) - min(x))


# # Test Train Split
# from sklearn.model_selection import train_test_split
# x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.3, random_state = 42)



# # Sklearn Gaussian Naive Bayes Model
# def models(x_train,y_train):
#   from sklearn.naive_bayes import GaussianNB
#   nb = GaussianNB()
#   nb.fit(x_train, y_train)
#   # return nb
#   # Accuracy
#   print("Naive Bayes score: ",nb.score(x_test, y_test))
#   return nb


# model = models(x_train,y_train)
```

```
def modelss(X_train,Y_train):
  from sklearn.naive_bayes import GaussianNB
  gauss = GaussianNB()
  gauss.fit(X_train, Y_train)
  print('Gaussian Naive Bayes Training Accuracy:', gauss.score(X_train, Y_train))
  return gauss



model = modelss(X_train,Y_train)
```

```
Gaussian Naive Bayes Training Accuracy: 0.9507042253521126
```

Confusion Matrix

```
from sklearn.metrics import confusion_matrix
# for i in range(len(model)):
i=1
cm = confusion_matrix(Y_test, model.predict(X_test))

TN = cm[0][0]
TP = cm[1][1]
FN = cm[1][0]
FP = cm[0][1]

print(cm)
print('Model[{}] Testing Accuracy = "{}!"'.format(i,  (TP + TN) / (TP + TN + FN + FP)))
print()# Print a new line
```

```
[[85  5]
 [ 6 47]]
Model[1] Testing Accuracy = "0.9230769230769231!"
```

Decision Tree

```
#Decision Tree
#Using DecisionTreeClassifier
def decision(X_train,Y_train):
  from sklearn.tree import DecisionTreeClassifier
  tree = DecisionTreeClassifier(criterion = 'entropy', max_depth = 2)
  tree.fit(X_train, Y_train)
  print('Decision Tree Classifier Training Accuracy:', tree.score(X_train, Y_train))
  return tree
model = decision(X_train,Y_train)
```

```
Decision Tree Classifier Training Accuracy: 0.9272300469483568
```

```
def decision(X_train,Y_train):
  from sklearn.tree import DecisionTreeClassifier
  tree = DecisionTreeClassifier(criterion = 'entropy', max_depth = 7)
  tree.fit(X_train, Y_train)
  print('Decision Tree Classifier Training Accuracy:', tree.score(X_train, Y_train))
  return tree
model = decision(X_train,Y_train)
```

```
Decision Tree Classifier Training Accuracy: 1.0
```

```
def decision(X_train,Y_train):
  from sklearn.tree import DecisionTreeClassifier
  tree = DecisionTreeClassifier(criterion = 'gini', max_depth = 2)
  tree.fit(X_train, Y_train)
  print('Decision Tree Classifier Training Accuracy:', tree.score(X_train, Y_train))
  return tree
model = decision(X_train,Y_train)
```

```
Decision Tree Classifier Training Accuracy: 0.9413145539906104
```

```
def decision(X_train,Y_train):
  from sklearn.tree import DecisionTreeClassifier
  tree = DecisionTreeClassifier(criterion = 'gini', max_depth = 7)
  tree.fit(X_train, Y_train)
  print('Decision Tree Classifier Training Accuracy:', tree.score(X_train, Y_train))
  return tree
model = decision(X_train,Y_train)
```

```
Decision Tree Classifier Training Accuracy: 0.9976525821596244
```

Confusion Matrix

```
from sklearn.metrics import confusion_matrix
# for i in range(len(model)):
i=2
cm = confusion_matrix(Y_test, model.predict(X_test))

TN = cm[0][0]
TP = cm[1][1]
FN = cm[1][0]
FP = cm[0][1]

print(cm)
print('Model[{}] Testing Accuracy = "{}!"'.format(i,  (TP + TN) / (TP + TN + FN + FP)))
print()# Print a new line
```

```
[[79 11]
 [ 3 50]]
Model[2] Testing Accuracy = "0.9020979020979021!"
```

Logistic Regression

```
def models(X_train,Y_train):
  #Using Logistic Regression
  from sklearn.linear_model import LogisticRegression
  log = LogisticRegression(random_state = 0)
  log.fit(X_train, Y_train)
  #print model accuracy on the training data.
  print('Logistic Regression Training Accuracy:', log.score(X_train, Y_train))
  return log
model = models(X_train,Y_train)
```

```
Logistic Regression Training Accuracy: 0.9906103286384976
```

Confusion Matrix

```
from sklearn.metrics import confusion_matrix
# for i in range(len(model)):
i=3
cm = confusion_matrix(Y_test, model.predict(X_test))
```

```
cm = confusion_matrix(Y_test, model.predict(X_test))

TN = cm[0][0]
TP = cm[1][1]
FN = cm[1][0]
FP = cm[0][1]

print(cm)
print('Model[{}] Testing Accuracy = "{}!"'.format(i,  (TP + TN) / (TP + TN + FN + FP)))
print()# Print a new line
```

```
[[86  4]
 [ 4 49]]
Model[3] Testing Accuracy = "0.9440559440559441!"
```

SVM(Support Vector Machine)

```
def models(X_train,Y_train):
  #Using SVC linear
  from sklearn.svm import SVC
  svc_lin = SVC(kernel = 'linear', random_state = 0)
  svc_lin.fit(X_train, Y_train)
  print('Support Vector Machine (Linear Classifier) Training Accuracy:', svc_lin.score(X_train, Y_train))
  return svc_lin


model = models(X_train,Y_train)
```

```
Support Vector Machine (Linear Classifier) Training Accuracy: 0.9882629107981221
```

Confusion Matrix

```
from sklearn.metrics import confusion_matrix
# for i in range(len(model)):
i=4
cm = confusion_matrix(Y_test, model.predict(X_test))

TN = cm[0][0]
TP = cm[1][1]
FN = cm[1][0]
FP = cm[0][1]

print(cm)
print('Model[{}] Testing Accuracy = "{}!"'.format(i,  (TP + TN) / (TP + TN + FN + FP)))
print()# Print a new line
```

```
[[87  3]
 [ 2 51]]
Model[4] Testing Accuracy = "0.965034965034965!"
```

Comparing All

```
def models(X_train,Y_train):

  #Using KNeighborsClassifier
  from sklearn.neighbors import KNeighborsClassifier
  knn = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 2)
  knn.fit(X_train, Y_train)

  #Using GaussianNB
  from sklearn.naive_bayes import GaussianNB
  gauss = GaussianNB()
  gauss.fit(X_train, Y_train)

  #Using DecisionTreeClassifier
  from sklearn.tree import DecisionTreeClassifier
  tree = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)
  tree.fit(X_train, Y_train)

  #Using Logistic Regression
  from sklearn.linear_model import LogisticRegression
  log = LogisticRegression(random_state = 0)
  log.fit(X_train, Y_train)

  #Using SVC linear
  from sklearn.svm import SVC
  svc_lin = SVC(kernel = 'linear', random_state = 0)
  svc_lin.fit(X_train, Y_train)

  #print model accuracy on the training data.
  print('[2]K Nearest Neighbor Training Accuracy:', knn.score(X_train, Y_train))
  print('[4]Gaussian Naive Bayes Training Accuracy:', gauss.score(X_train, Y_train))
  print('[5]Decision Tree Classifier Training Accuracy:', tree.score(X_train, Y_train))
  print('[1]Logistic Regression Training Accuracy:', log.score(X_train, Y_train))
  print('[3]Support Vector Machine (Linear Classifier) Training Accuracy:', svc_lin.score(X_train, Y_train))



  return knn, gauss, tree, log, svc_lin


model = models(X_train,Y_train)
```

```
[2]K Nearest Neighbor Training Accuracy: 0.9765258215962441
[4]Gaussian Naive Bayes Training Accuracy: 0.9507042253521126
[5]Decision Tree Classifier Training Accuracy: 1.0
[1]Logistic Regression Training Accuracy: 0.9906103286384976
[3]Support Vector Machine (Linear Classifier) Training Accuracy: 0.9882629107981221
```

Compairing Confusion Matrix

```
from sklearn.metrics import confusion_matrix
for i in range(len(model)):
  cm = confusion_matrix(Y_test, model[i].predict(X_test))

  TN = cm[0][0]
  TP = cm[1][1]
  FN = cm[1][0]
  FP = cm[0][1]

  print(cm)
  print('Model[{}] Testing Accuracy = "{}!"'.format(i,  (TP + TN) / (TP + TN + FN + FP)))
  print()# Print a new line
```

```
[[89  1]
 [ 5 48]]
Model[0] Testing Accuracy = "0.958041958041958!"

[[85  5]
 [ 6 47]]
Model[1] Testing Accuracy = "0.9230769230769231!"

[[84  6]
 [ 1 52]]
Model[2] Testing Accuracy = "0.951048951048951!"

[[86  4]
 [ 4 49]]
```

```
Model[3] Testing Accuracy = "0.9440559440559441!"

[[87  3]
 [ 2 51]]
Model[4] Testing Accuracy = "0.965034965034965!"
```

```
#Other ways to get the classification accuracy & other metrics

from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score

for i in range(len(model)):
  print('Model ',i)
  #Check precision, recall, f1-score
  print( classification_report(Y_test, model[i].predict(X_test)) )
  #Another way to get the models accuracy on the test data
  print( accuracy_score(Y_test, model[i].predict(X_test)))
  print()#Print a new line
```

```
Model  0
              precision    recall  f1-score   support

           0       0.96      0.96      0.96        90
           1       0.92      0.92      0.92        53

    accuracy                           0.94       143
   macro avg       0.94      0.94      0.94       143
weighted avg       0.94      0.94      0.94       143

0.9440559440559441

Model  1
              precision    recall  f1-score   support

           0       0.95      0.99      0.97        90
           1       0.98      0.91      0.94        53

    accuracy                           0.96       143
   macro avg       0.96      0.95      0.95       143
weighted avg       0.96      0.96      0.96       143

0.958041958041958

Model  2
              precision    recall  f1-score   support

           0       0.98      0.97      0.97        90
           1       0.94      0.96      0.95        53

    accuracy                           0.97       143
   macro avg       0.96      0.96      0.96       143
weighted avg       0.97      0.97      0.97       143

0.965034965034965

Model  3
              precision    recall  f1-score   support

           0       0.93      0.94      0.94        90
           1       0.90      0.89      0.90        53

    accuracy                           0.92       143
   macro avg       0.92      0.92      0.92       143
weighted avg       0.92      0.92      0.92       143

0.9230769230769231

Model  4
              precision    recall  f1-score   support

           0       0.99      0.93      0.96        90
           1       0.90      0.98      0.94        53

    accuracy                           0.95       143
   macro avg       0.94      0.96      0.95       143
weighted avg       0.95      0.95      0.95       143
```