I examine the Breast Cancer dataset in this notebook and created models with the aim of identifying questionable cells as Benign or Malignan

1. RandomForest

   * Confsion Matrix
     * Testing Accuracy
     * Training Accuracy
   * ROC curve

2. k-nearest neighbors (KNN)

   * Confsion Matrix
     * Testing Accuracy
     * Training Accuracy
   * ROC curve

3. DecIsionTree

   * Confsion Matrix
     * Testing Accuracy
     * Training Accuracy
   - ROC curve

4. Logistic Regression

   * Confsion Matrix
     * Testing Accuracy
     * Training Accuracy
   * ROC curve

```
# import dependencies
# data cleaning and manipulation
import pandas as pd
import numpy as np

# data visualization
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.preprocessing import LabelEncoder
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from sklearn.metrics import roc_curve, auc
import sklearn.metrics as metrics
```

Reading the data and checking the first 5 rows

```
df = pd.read_csv('/content/data.csv')
```

```
df.head()
```

mean	smoothness_mean	compactness_mean	concavity_mean	poin
)01.0	0.11840	0.27760	0.3001	

```
df=df.loc[:,['radius_mean','texture_mean','perimeter_mean','area_mean','smoothness_mean','compactness_mean','concavity_mean','concave poi
```

```
#df.drop_duplicates(inplace =True)
```

386.1	0.14250	0.28390	0.2414

```
df.head()
```

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symm
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	

general summary of the dataframe

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 11 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   radius_mean             569 non-null    float64
 1   texture_mean            569 non-null    float64
 2   perimeter_mean          569 non-null    float64
 3   area_mean               569 non-null    float64
 4   smoothness_mean         569 non-null    float64
 5   compactness_mean        569 non-null    float64
 6   concavity_mean          569 non-null    float64
 7   concave points_mean     569 non-null    float64
 8   symmetry_mean           569 non-null    float64
 9   fractal_dimension_mean  569 non-null    float64
 10  diagnosis               569 non-null    object 
dtypes: float64(10), object(1)
memory usage: 49.0+ KB
```

remove the 'Unnamed: 32' column

```
#df = df.drop(['Unnamed: 32','id'], axis=1)
```

```
df.dtypes
```

```
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
diagnosis                  object
dtype: object
```

```
df.isnull().sum()
```

```
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
diagnosis                 0
dtype: int64
```

```
df['diagnosis'].unique()
```

```
array(['M', 'B'], dtype=object)
```

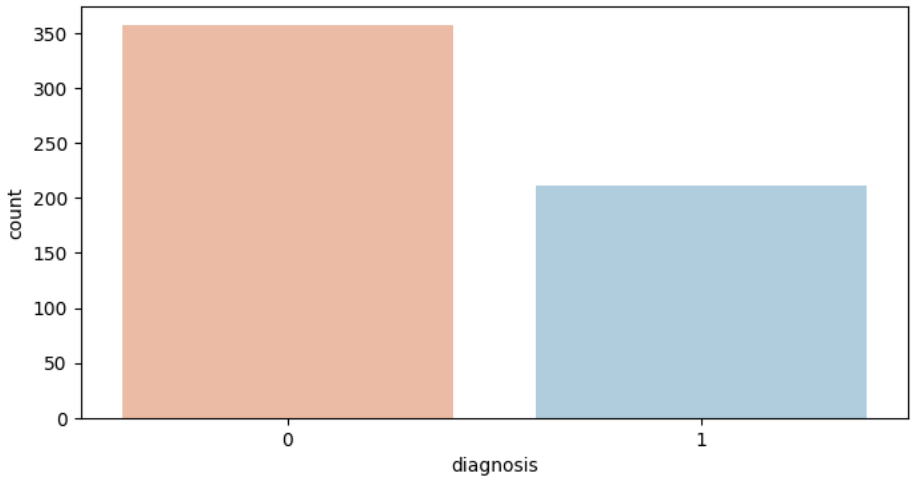
```
#Converting M, B to 0, 1
le = LabelEncoder()
df['diagnosis']=le.fit_transform(df['diagnosis'])
```

```
df['diagnosis'].unique()
```

```
array([1, 0])
```

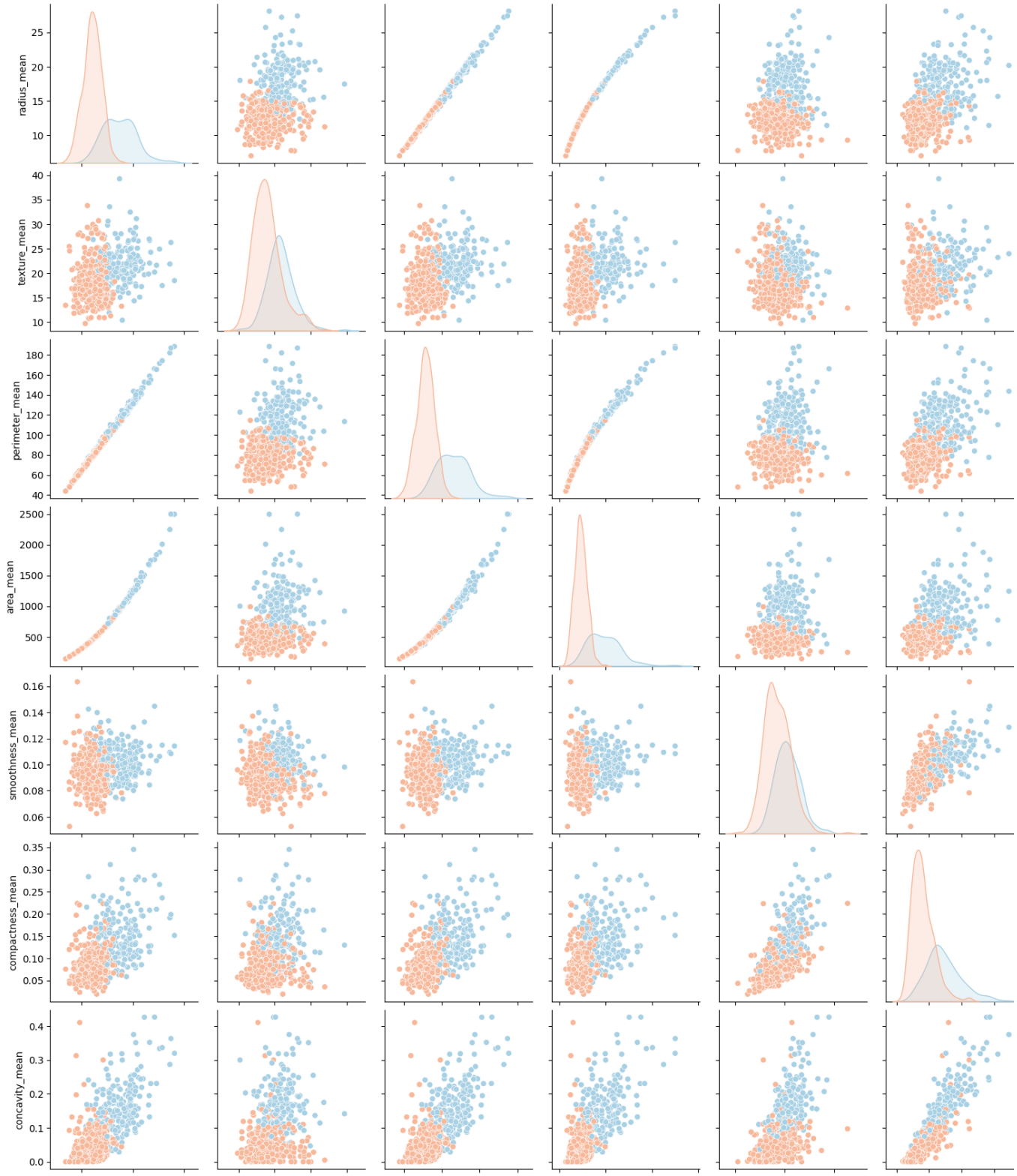
```
plt.figure(figsize=(8, 4))
sns.countplot(x='diagnosis', data=df, palette='RdBu')
```

```
<Axes: xlabel='diagnosis', ylabel='count'>
```




```
  # generate a scatter plot matrix with the "mean" columns
cols = ['diagnosis',
        'radius_mean',
        'texture_mean',
        'perimeter_mean',
        'area_mean',
        'smoothness_mean',
        'compactness_mean',
        'concavity_mean',
        'concave points_mean',
        'symmetry_mean',
        'fractal_dimension_mean']

sns.pairplot(data=df[cols], hue='diagnosis', palette='RdBu')
```

```
<seaborn.axisgrid.PairGrid at 0x7e3026b94250>
```




```
#X and Y split
X = df.drop('diagnosis',axis=1)
y = df['diagnosis']
```

```
#Standardaization
scaler = StandardScaler()
X_std = scaler.fit_transform(X)
```

```
#Feature Selection Using Principle Component Analysis(PCA)
"""n_components = 20
pca = PCA(n_components=n_components)
pca.fit(X_std)"""
```

```
'n_components = 20\npca = PCA(n_components=n_components)\npca.fit(X_std)'
```

```
#X_pca = pca.transform(X_std)
```

```
"""print(f"Original data shape: {X_std.shape}")
print(f"Transformed data shape: {X_pca.shape}")
print("Explained variance ratios:", pca.explained_variance_ratio_)"""
```

```
'print(f"Original data shape: {X_std.shape}")\nprint(f"Transformed data shape: {X_pca.shape}")\nprint("Explained variance ratios:",
```


```
#train test Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

```
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

```
((455, 10), (114, 10), (455,), (114,))
```

RandomForest

```
RF_model=clf = RandomForestClassifier(max_depth=2, random_state=43)
RF_model.fit(X_train, y_train)
```

```
RandomForestClassifier
RandomForestClassifier(max_depth=2, random_state=43)
```

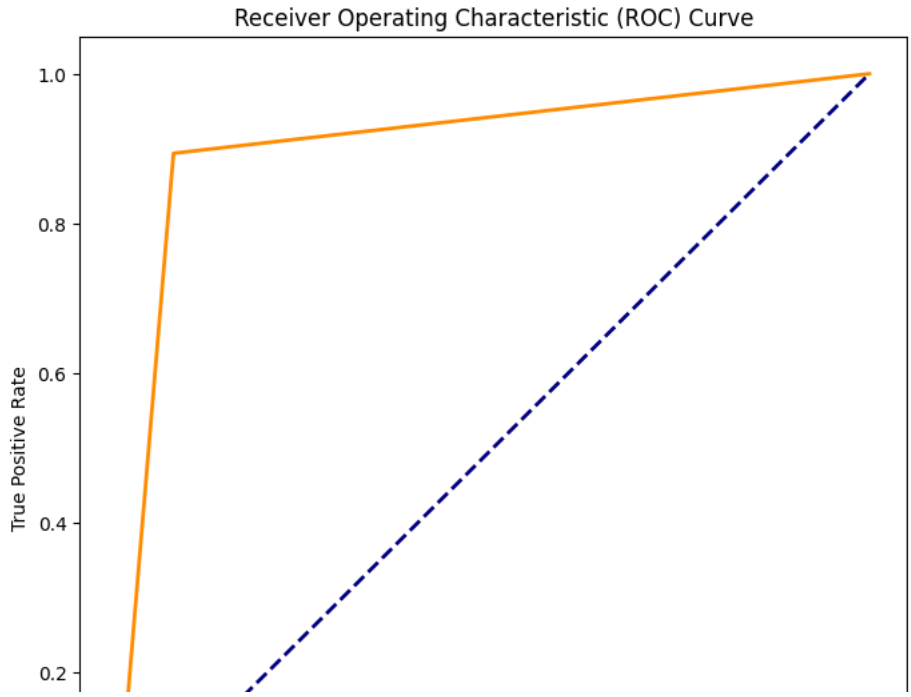
```
y_pred= RF_model.predict(X_test)
y_predt= RF_model.predict(X_train)
```

```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
result=confusion_matrix(y_test,y_pred)
print("Confsion Matrix")
print(result)
result1=classification_report(y_test,y_pred)
print("Classification Report")
print(result1)
result2= accuracy_score(y_test,y_pred)
print("Testing Accuracy : ", result2)
result3= accuracy_score(y_train,y_predt)
print(" Training Accuracy : ", result3)
```

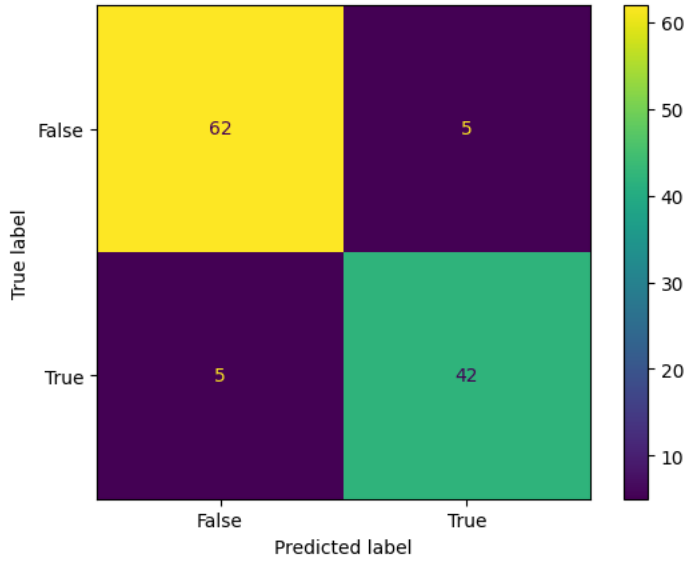
```
Confsion Matrix
[[62  5]
 [ 5 42]]
Classification Report
              precision    recall  f1-score   support

           0       0.93      0.93      0.93        67
           1       0.89      0.89      0.89        47

    accuracy                           0.91       114
   macro avg       0.91      0.91      0.91       114
weighted avg       0.91      0.91      0.91       114

Testing Accuracy :  0.9122807017543859
 Training Accuracy :  0.9428571428571428
```

```
# Compute ROC curve and ROC area
fpr, tpr, _ = roc_curve(y_test,y_pred )
roc_auc = auc(fpr, tpr)
```

```
# Plot ROC curve
plt.figure(figsize=(8, 8))
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = {:.2f})'.format(roc_auc))
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```




```
confusion_matrix = metrics.confusion_matrix(y_test,y_pred)

cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix, display_labels = [False, True])

cm_display.plot()
plt.show()
```



k-nearest neighbors (KNN)

```
neigh = KNeighborsClassifier(n_neighbors=7)
neigh.fit(X_train, y_train)
```

```
KNeighborsClassifier
KNeighborsClassifier(n_neighbors=7)
```

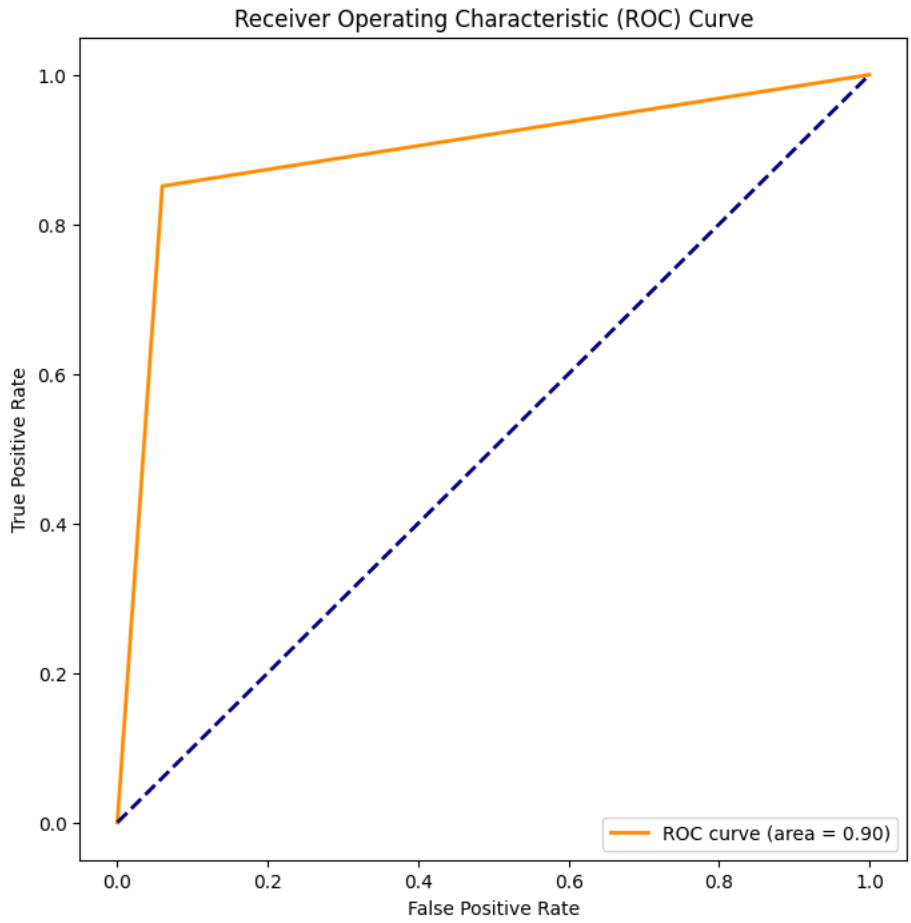
```
y_pred_1= neigh.predict(X_test)
y_predt1= neigh.predict(X_train)
```

```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
result=confusion_matrix(y_test,y_pred_1)
print("Confsion Matrix")
print(result)
result1=classification_report(y_test,y_pred_1)
print("Classification Report")
print(result1)
```

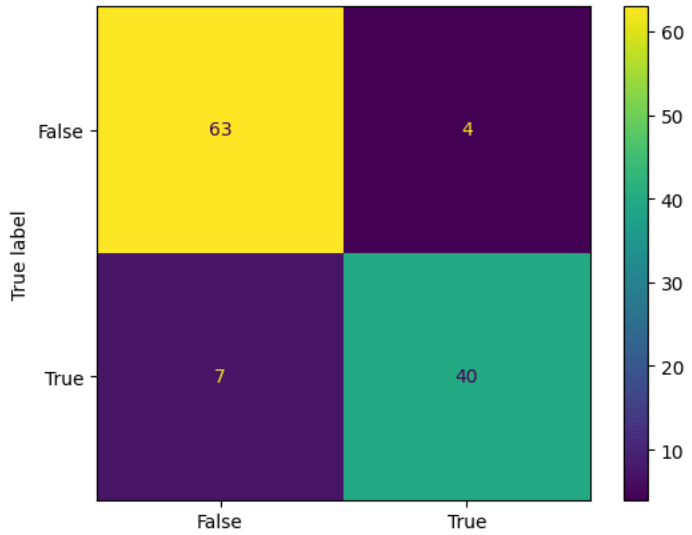
```
result3= accuracy_score(y_train,y_predt1)
print(" Training Accuracy : ", result3)
result2= accuracy_score(y_test,y_pred_1)
print("Testing Accuracy : ", result2)
```

```
Confsion Matrix
[[63  4]
 [ 7 40]]
Classification Report
              precision    recall  f1-score   support

           0       0.90      0.94      0.92        67
           1       0.91      0.85      0.88        47

    accuracy                           0.90       114
   macro avg       0.90      0.90      0.90       114
weighted avg       0.90      0.90      0.90       114

 Training Accuracy :  0.9010989010989011
Testing Accuracy :  0.9035087719298246
```

```
# Compute ROC curve and ROC area
fpr1, tpr1, _ = roc_curve(y_test, y_pred_1)
roc_auc_1 = auc(fpr1, tpr1)

# Plot ROC curve
plt.figure(figsize=(8, 8))
plt.plot(fpr1, tpr1, color='darkorange', lw=2, label='ROC curve (area = {:.2f})'.format(roc_auc_1))
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```




```
confusion_matrix = metrics.confusion_matrix(y_test,y_pred_1)

cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix, display_labels = [False, True])

cm_display.plot()
plt.show()
```



DecisionTree

```
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier( random_state=0)
classifier.fit(X_train,y_train)
```

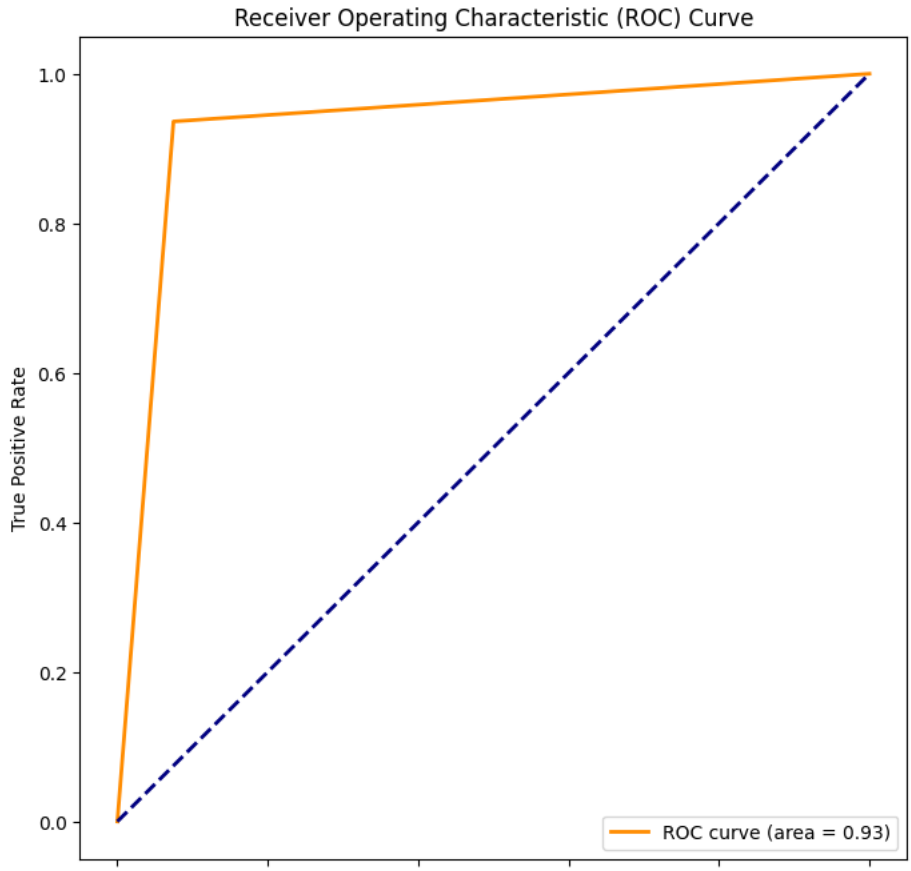
```
DecisionTreeClassifier
DecisionTreeClassifier(random_state=0)
```

```
y_pred_2= classifier.predict(X_test)
y_predt2= classifier.predict(X_train)
```

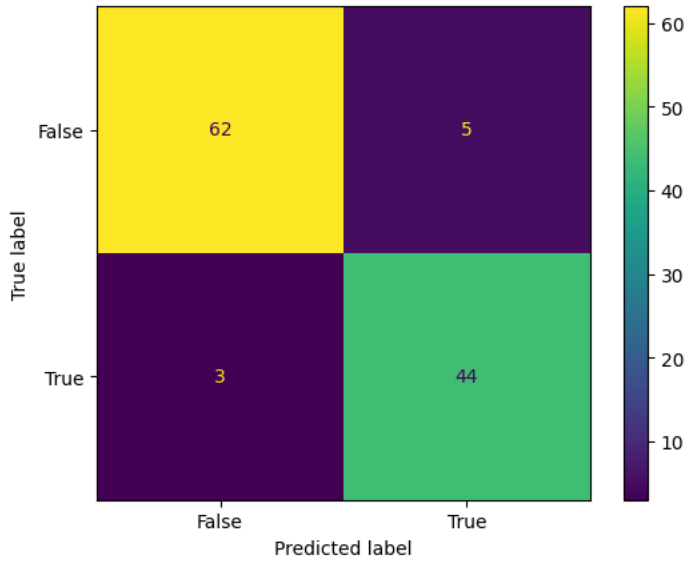
```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
result=confusion_matrix(y_test,y_pred_2)
print("Confsion Matrix")
print(result)
result1=classification_report(y_test,y_pred_2)
print("Classification Report")
print(result1)
result2= accuracy_score(y_test,y_pred_2)
print("Testing Accuracy : ", result2)
result3= accuracy_score(y_train,y_predt2)
print(" Training Accuracy : ", result3)
```

```
Confsion Matrix
[[62  5]
 [ 3 44]]
Classification Report
              precision    recall  f1-score   support

           0       0.95      0.93      0.94        67
           1       0.90      0.94      0.92        47

    accuracy                           0.93       114
   macro avg       0.93      0.93      0.93       114
weighted avg       0.93      0.93      0.93       114

Testing Accuracy :  0.9298245614035088
 Training Accuracy :  1.0
```

```
fpr1, tpr1, _ = roc_curve(y_test, y_pred_2)
roc_auc_1 = auc(fpr1, tpr1)

# Plot ROC curve
plt.figure(figsize=(8, 8))
plt.plot(fpr1, tpr1, color='darkorange', lw=2, label='ROC curve (area = {:.2f})'.format(roc_auc_1))
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```




```
confusion_matrix = metrics.confusion_matrix(y_test,y_pred_2)

cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix, display_labels = [False, True])

cm_display.plot()
plt.show()
```



Double-click (or enter) to edit

Logistic Regression

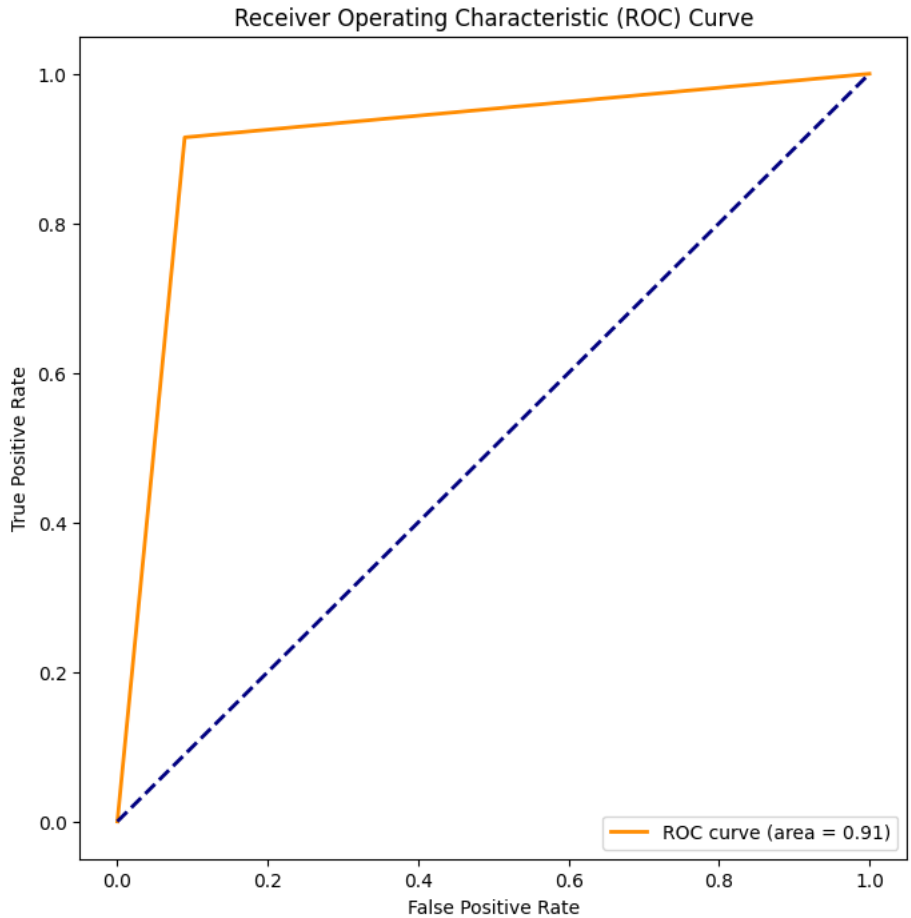
```
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(max_iter=1000)
classifier.fit(X_train,y_train)
```

```
LogisticRegression
LogisticRegression(max_iter=1000)
```

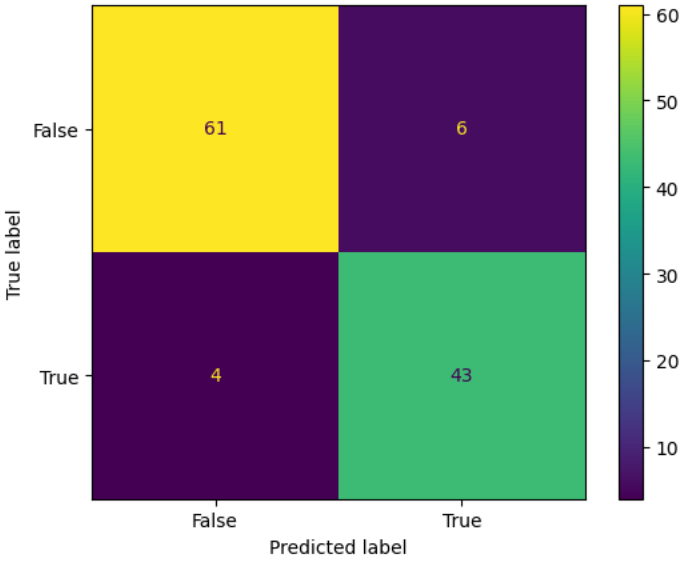
```
# Make predictions using the testing data
y_pred_3= classifier.predict(X_test)
y_predt3= classifier.predict(X_train)


from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
result=confusion_matrix(y_test,y_pred_3)
print("Confsion Matrix")
print(result)
result1=classification_report(y_test,y_pred_3)
print("Classification Report")
print(result1)
result2= accuracy_score(y_test,y_pred_3)
print("Testing Accuracy : ", result2)
result3= accuracy_score(y_train,y_predt3)
print(" Training Accuracy : ", result3)
```

```
Confsion Matrix
[[61  6]
 [ 4 43]]
Classification Report
              precision    recall  f1-score   support

           0       0.94      0.91      0.92        67
           1       0.88      0.91      0.90        47

    accuracy                           0.91       114
   macro avg       0.91      0.91      0.91       114
weighted avg       0.91      0.91      0.91       114

Testing Accuracy :  0.9122807017543859
 Training Accuracy :  0.9164835164835164
```

```
fpr1, tpr1, _ = roc_curve(y_test, y_pred_3)
roc_auc_1 = auc(fpr1, tpr1)

# Plot ROC curve
plt.figure(figsize=(8, 8))
plt.plot(fpr1, tpr1, color='darkorange', lw=2, label='ROC curve (area = {:.2f})'.format(roc_auc_1))
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```



```
confusion_matrix = metrics.confusion_matrix(y_test,y_pred_3)

cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix, display_labels = [False, True])

cm_display.plot()
plt.show()
```



Using AutoLogging_ML algo

```
!pip install AutoLogging_ML

from AutoLogging_ML import AutoLogger

a,model=AutoLogger.train_and_log_classification(X_train,y_train,X_test,y_test)
```

```
750:    learn: 0.0366986        total: 1.62s    remaining: 536ms
751:    learn: 0.0366461        total: 1.62s    remaining: 534ms
752:    learn: 0.0366035        total: 1.62s    remaining: 532ms
753:    learn: 0.0365271        total: 1.62s    remaining: 530ms
754:    learn: 0.0364673        total: 1.63s    remaining: 528ms
755:    learn: 0.0364047        total: 1.63s    remaining: 525ms
756:    learn: 0.0363009        total: 1.63s    remaining: 523ms
757:    learn: 0.0362647        total: 1.63s    remaining: 521ms
758:    learn: 0.0362069        total: 1.63s    remaining: 519ms
759:    learn: 0.0361235        total: 1.64s    remaining: 517ms
760:    learn: 0.0360542        total: 1.64s    remaining: 514ms
761:    learn: 0.0359815        total: 1.64s    remaining: 512ms
762:    learn: 0.0359136        total: 1.64s    remaining: 510ms
763:    learn: 0.0358665        total: 1.64s    remaining: 508ms
```

```
a
```

	model	training-accuracy	training-precision	training-recall	training-f1	training-confusion matrix	validation-accuracy	validation-precision
1	svm-rbf	87.032967	89.322139	83.035528	84.929123	[[283, 7], [52, 113]]	87.719298	90.343580
2	svm-poly	87.692308	89.774953	83.944619	85.776170	[[283, 7], [49, 116]]	88.596491	90.918803
4	naive bayes	91.648352	91.483871	90.313480	90.842161	[[276, 14], [24, 141]]	88.596491	88.331202
0	svm-linear	91.868132	91.666667	90.616510	91.095843	[[276, 14], [23, 142]]	91.228070	90.800628
3	knn	90.329670	90.700779	88.234065	89.236559	[[278, 12], [32, 133]]	91.228070	91.582050
10	logistic regression	91.648352	91.483871	90.313480	90.842161	[[276, 14], [24, 141]]	91.228070	90.800628
5	decision tree	100.000000	100.000000	100.000000	100.000000	[[290, 0], [0, 165]]	92.105263	91.761364
9	xgboost	100.000000	100.000000	100.000000	100.000000	[[290, 0], [0, 165]]	92.982456	93.043478
11	bagging classifier	99.780220	99.828179	99.696970	99.761966	[[290, 0], [1, 164]]	93.859649	93.797954
12	extra trees classifier	100.000000	100.000000	100.000000	100.000000	[[290, 0], [0, 165]]	93.859649	93.797954
13	linear discriminant analysis	93.846154	94.565553	92.168234	93.171683	[[285, 5], [23, 142]]	93.859649	94.155844
15	cat boost classifier	100.000000	100.000000	100.000000	100.000000	[[290, 0], [0, 165]]	93.859649	93.560606
7	adaboost	100.000000	100.000000	100.000000	100.000000	[[290, 0], [0, 165]]	94.736842	94.569705
14	quadratic discriminant analysis	93.626374	94.060109	92.126437	92.959851	[[283, 7], [22, 143]]	94.736842	95.316083
8	gradient boost	100.000000	100.000000	100.000000	100.000000	[[290, 0], [0, 165]]	95.614035	95.359848
6	random forest	100.000000	100.000000	100.000000	100.000000	[[290, 0], [0, 165]]	96.491228	96.169545

```
model
```

```
RandomForestClassifier
RandomForestClassifier()
```

17.99 10.38 122.80 1001.0 0.11840 0.27760 0.3001 0.14710 0.2419 0.07871

Performing the objective of our project to predict Malignant or Bening

```
#17.99  10.38   122.80  1001.0  0.11840 0.27760 0.3001  0.14710 0.2419  0.07871
x1=float(input())
x2=float(input())
x3=float(input())
x4=float(input())
x5=float(input())
x6=float(input())
x7=float(input())
x8=float(input())
x9=float(input())
x10=float(input())
inp=[[x1,x2,x3,x4,x5,x6,x7,x8,x9,x10]]
for x in inp:
    x=x-np.max(x)/(np.max(x)-np.min(x))
out=model.predict(inp)
a=out[0]
print(a)
if a>0.5:
```

```
    print('Malignant')
else:
    print('Benign')
```

```
 17.99
 10.38
 122.80
 1001.0
 0.11840
 0.27760
 0.3001
 0.14710
 0.2419
 0.07871
 1
 Malignant
 /usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but RandomForestClass
   warnings.warn(
```

The Prediction

I've successfully created RandomForest, k-nearest neighbors (KNN),Logistic Regression & DecIsionTree model in the earlier part. These models can effectively assign each observation a probability between 0 and 1 using some unlabeled data. However, the predictions must be encoded in order for each occurrence to be directly compared with the labels in the test data in order for us to determine whether the predictions are accurate. In other words, the forecasts should display "M" or "B"—which stand for malignant and benign, respectively—instead of integers between 0 and 1. In our paradigm, the "Benign" class corresponds to a probability of 1, while the "Malignant" class corresponds to a probability of 0. We can therefore use a threshold value of 0.5 to our predictions, assigning all values closer to 0 a label of "M" and assigniing all values closer to 1 a label of "B".