Breast Cancer Prediction

In [1]:
```
# to ignore the warnings given by commands (if any).
i mport warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
# importing libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn
```

In [3]:
```
# importing dataset and converting it into DataFrame
df = pd.read_csv("https://raw.githubusercontent.com/ingledarshan/AIML-B2/main/data.csv")

# to show only first 5 rows of the DataFrame
df.head()
```

Out[3]:

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

5 rows × 33 columns

In [4]:
```
# to get all the columns present in dataset
df.columns
```

Out[4]:
```
Index(['id', 'diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mea
n',
       'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mea
n',
       'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
       'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_
se',
       'compactness_se', 'concavity_se', 'concave points_se', 'symmetry_s
e',
       'fractal_dimension_se', 'radius_worst', 'texture_worst',
       'perimeter_worst', 'area_worst', 'smoothness_worst',
       'compactness_worst', 'concavity_worst', 'concave points_worst',
       'symmetry_worst', 'fractal_dimension_worst', 'Unnamed: 32'],
      dtype='object')
```

In [5]:
```
# to get the detailed info of columns in dataset
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 33 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   id                       569 non-null    int64  
 1   diagnosis                569 non-null    object 
 2   radius_mean              569 non-null    float64
 3   texture_mean             569 non-null    float64
 4   perimeter_mean           569 non-null    float64
 5   area_mean                569 non-null    float64
 6   smoothness_mean          569 non-null    float64
 7   compactness_mean         569 non-null    float64
 8   concavity_mean           569 non-null    float64
 9   concave points_mean      569 non-null    float64
 10  symmetry_mean            569 non-null    float64
 11  fractal_dimension_mean   569 non-null    float64
 12  radius_se                569 non-null    float64
 13  texture_se               569 non-null    float64
 14  perimeter_se             569 non-null    float64
 15  area_se                  569 non-null    float64
 16  smoothness_se            569 non-null    float64
 17  compactness_se           569 non-null    float64
 18  concavity_se             569 non-null    float64
 19  concave points_se        569 non-null    float64
 20  symmetry_se              569 non-null    float64
 21  fractal_dimension_se     569 non-null    float64
 22  radius_worst             569 non-null    float64
 23  texture_worst            569 non-null    float64
 24  perimeter_worst          569 non-null    float64
 25  area_worst               569 non-null    float64
 26  smoothness_worst         569 non-null    float64
 27  compactness_worst        569 non-null    float64
 28  concavity_worst          569 non-null    float64
 29  concave points_worst     569 non-null    float64
 30  symmetry_worst           569 non-null    float64
 31  fractal_dimension_worst  569 non-null    float64
 32  Unnamed: 32              0 non-null      float64
dtypes: float64(31), int64(1), object(1)
memory usage: 146.8+ KB
```

In [6]:
```
# to check the unnamed column
df["Unnamed: 32"].head()
```

Out[6]:
```
0   NaN
1   NaN
2   NaN
3   NaN
4   NaN
Name: Unnamed: 32, dtype: float64
```

In [7]:
```
# to remove the column as it is not required (unnamed)
df.drop("Unnamed: 32",axis=1, inplace=True)
```

In [8]:
```
# to remove the id column
df.drop("id",axis=1,inplace=True)
```

In [9]:
```
# to check if columns are removed
df.columns
```

Out[9]:
```
Index(['diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mea
n',
       'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
       'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_
se',
       'compactness_se', 'concavity_se', 'concave points_se', 'symmetry_s
e',
       'fractal_dimension_se', 'radius_worst', 'texture_worst',
       'perimeter_worst', 'area_worst', 'smoothness_worst',
       'compactness_worst', 'concavity_worst', 'concave points_worst',
       'symmetry_worst', 'fractal_dimension_worst'],
      dtype='object')
```

In [10]:
```
# to check the datatype of columns
type(df.columns)
```

Out[10]:
```
pandas.core.indexes.base.Index
```

In [11]:
```
# to segregate the columns (mean, se, worst)
l = list(df.columns)
print(l)
```

```
['diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean', 'area_mea
n', 'smoothness_mean', 'compactness_mean', 'concavity_mean', 'concave poi
nts_mean', 'symmetry_mean', 'fractal_dimension_mean', 'radius_se', 'textu
re_se', 'perimeter_se', 'area_se', 'smoothness_se', 'compactness_se', 'co
ncavity_se', 'concave points_se', 'symmetry_se', 'fractal_dimension_se',
'radius_worst', 'texture_worst', 'perimeter_worst', 'area_worst', 'smooth
ness_worst', 'compactness_worst', 'concavity_worst', 'concave points_wors
t', 'symmetry_worst', 'fractal_dimension_worst']
```

In [12]:
```
# to store columns in a variable ( just for understanding)
features_mean = l[1:11]
features_se = l[12:23]
features_worst = l[23:]
```

In [13]:
```
print(features_mean)
```

```
['radius_mean', 'texture_mean', 'perimeter_mean', 'area_mean', 'smoothnes
s_mean', 'compactness_mean', 'concavity_mean', 'concave points_mean', 'sy
mmetry_mean', 'fractal_dimension_mean']
```

In [14]:
```
print(features_se)
```

```
['texture_se', 'perimeter_se', 'area_se', 'smoothness_se', 'compactness_s
e', 'concavity_se', 'concave points_se', 'symmetry_se', 'fractal_dimensio
n_se', 'radius_worst', 'texture_worst']
```

In [15]:
```
print(features_worst)
```

```
['perimeter_worst', 'area_worst', 'smoothness_worst', 'compactness_wors
t', 'concavity_worst', 'concave points_worst', 'symmetry_worst', 'fractal
_dimension_worst']
```

In [16]:
```
# to get first 5 rews of DataFrame
df.head()
```

Out[16]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness
0	M	17.99	10.38	122.80	1001.0	(
1	M	20.57	17.77	132.90	1326.0	(
2	M	19.69	21.25	130.00	1203.0	(
3	M	11.42	20.38	77.58	386.1	(
4	M	20.29	14.34	135.10	1297.0	(

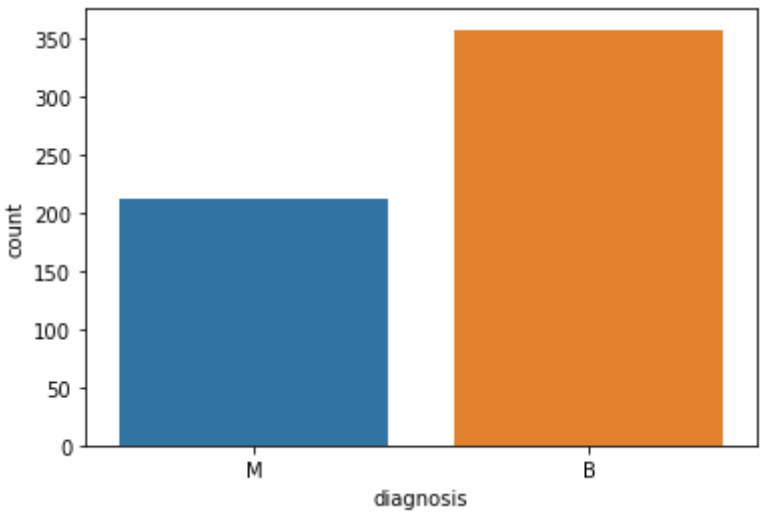
5 rows × 31 columns

In [17]:
```
# to check what values(type of cancer) present in diagnosis column  # M=
 Malignant, B= Benign
df["diagnosis"].unique()
```

Out[17]:
```
array(['M', 'B'], dtype=object)
```

In [18]:
```
# to visualise the number of patients of  Malignant and Benign cancer.
sns.countplot("diagnosis",data=df,label="Count")
```

Out[18]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1f60d1e0670>
```



In [19]:
```
# to count numbet of values (patients)
df["diagnosis"].value_counts()
```

Out[19]:
```
B    357
M    212
Name: diagnosis, dtype: int64
```

In [20]:
```
# to check the shape (rows and columns) in the DataFrame
df.shape
```

Out[20]:
```
(569, 31)
```

Explore the Data

In [21]:
```
# to describe the data (max, min, std ..etc)
df.describe()
```

Out[21]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mea
count	569.000000	569.000000	569.000000	569.000000	569.00000
mean	14.127292	19.289649	91.969033	654.889104	0.09636
std	3.524049	4.301036	24.298981	351.914129	0.01406
min	6.981000	9.710000	43.790000	143.500000	0.05263
25%	11.700000	16.170000	75.170000	420.300000	0.08637
50%	13.370000	18.840000	86.240000	551.100000	0.09587
75%	15.780000	21.800000	104.100000	782.700000	0.10530
max	28.110000	39.280000	188.500000	2501.000000	0.16340

8 rows × 30 columns

In [22]:
```
# len function to check the length of columns
len(df.columns)
```

Out[22]:
```
31
```

Co-relation Plot

In [23]:
```
# to find the co-relation among the data features
corr = df.corr()
corr
```

Out[23]:

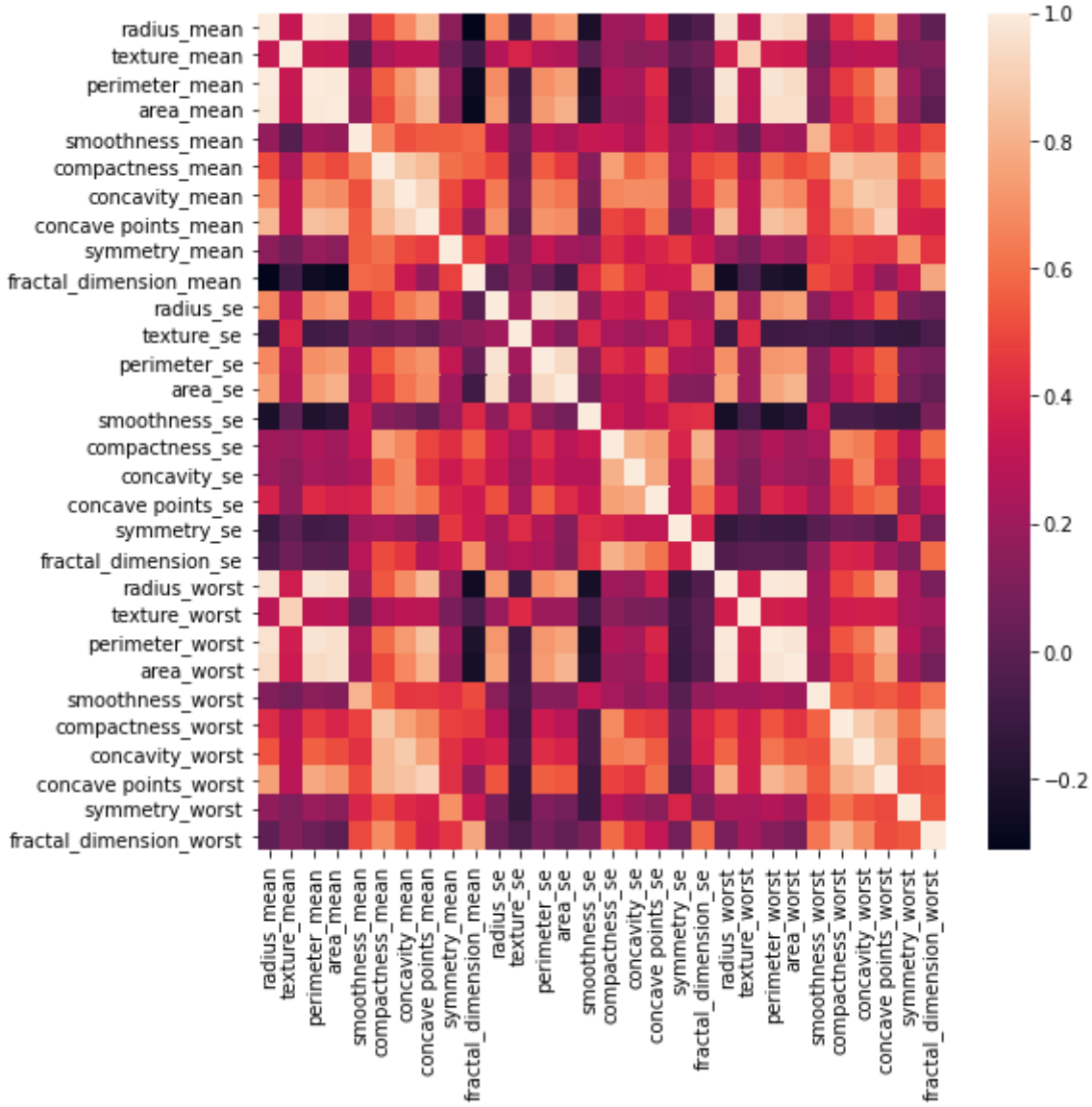
	radius_mean	texture_mean	perimeter_mean	area_mean
radius_mean	1.000000	0.323782	0.997855	0.987357
texture_mean	0.323782	1.000000	0.329533	0.321086
perimeter_mean	0.997855	0.329533	1.000000	0.986507
area_mean	0.987357	0.321086	0.986507	1.000000
smoothness_mean	0.170581	-0.023389	0.207278	0.177028
compactness_mean	0.506124	0.236702	0.556936	0.498502
concavity_mean	0.676764	0.302418	0.716136	0.685983
concave points_mean	0.822529	0.293464	0.850977	0.823269
symmetry_mean	0.147741	0.071401	0.183027	0.151293
fractal_dimension_mean	-0.311631	-0.076437	-0.261477	-0.283110
radius_se	0.679090	0.275869	0.691765	0.732562
texture_se	-0.097317	0.386358	-0.086761	-0.066280
perimeter_se	0.674172	0.281673	0.693135	0.726628
area_se	0.735864	0.259845	0.744983	0.800086
smoothness_se	-0.222600	0.006614	-0.202694	-0.166777
compactness_se	0.206000	0.191975	0.250744	0.212583
concavity_se	0.194204	0.143293	0.228082	0.207660
concave points_se	0.376169	0.163851	0.407217	0.372320
symmetry_se	-0.104321	0.009127	-0.081629	-0.072497
fractal_dimension_se	-0.042641	0.054458	-0.005523	-0.019887
radius_worst	0.969539	0.352573	0.969476	0.962746
texture_worst	0.297008	0.912045	0.303038	0.287489
perimeter_worst	0.965137	0.358040	0.970387	0.959120
area_worst	0.941082	0.343546	0.941550	0.959213
smoothness_worst	0.119616	0.077503	0.150549	0.123523
compactness_worst	0.413463	0.277830	0.455774	0.390410
concavity_worst	0.526911	0.301025	0.563879	0.512606
concave points_worst	0.744214	0.295316	0.771241	0.722017
symmetry_worst	0.163953	0.105008	0.189115	0.143570
fractal_dimension_worst	0.007066	0.119205	0.051019	0.003738

30 rows × 30 columns

In [24]:
```
# to give the size of figure req
plt.figure(figsize=(8,8))

# to plot the graph of co-related features ( white spots are co-related f
eatures )
sns.heatmap(corr)
```

Out[24]: <matplotlib.axes._subplots.AxesSubplot at 0x1f60d2e7c10>



In [25]:
```
# to give numeric vales to the categorical variables (M= Malignant, B= Be
nign)
df["diagnosis"] = df["diagnosis"].map({"M":1,"B":0})
```

In [26]:
```
df.head()
```

Out[26]:

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

5 rows × 31 columns

In [27]:
```
df["diagnosis"].unique()
```

Out[27]:
```
array([1, 0], dtype=int64)
```

In [28]:
```
# to drop diagnosis  column
X = df.drop("diagnosis",axis=1)
X.head()
```

Out[28]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	co
0	17.99	10.38	122.80	1001.0	0.11840	
1	20.57	17.77	132.90	1326.0	0.08474	
2	19.69	21.25	130.00	1203.0	0.10960	
3	11.42	20.38	77.58	386.1	0.14250	
4	20.29	14.34	135.10	1297.0	0.10030	

5 rows × 30 columns

In [29]:
```
y = df["diagnosis"]
y.head()
```

Out[29]:
```
0    1
1    1
2    1
3    1
4    1
Name: diagnosis, dtype: int64
```

In [30]:
```
# to segregate the dataset into train and test models in ratio (training
 model = 70% testing model = 30%).
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)
```

In [31]:
```
df.shape
```

Out[31]: (569, 31)

In [32]:
```
X_train.shape
```

Out[32]: (398, 30)

In [33]:
```
X_test.shape
```

Out[33]: (171, 30)

In [34]:
```
y_train.shape
```

Out[34]: (398,)

In [35]:
```
y_test.shape
```

Out[35]: (171,)

In [36]:
```
X_train.head(1)
```

Out[36]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean
184	15.28	22.41	98.92	710.6	0.09057

1 rows × 30 columns

In [ ]:
```
StandardScaler : It transforms the data in such a manner that it has mean
as 0 and standard deviation as 1. In short, it standardizes the data. Sta
ndardization is useful for data which has negative values.
```

In [37]:
```
# to standardize the features of dataset
from sklearn.preprocessing import StandardScaler
ss = StandardScaler()
```

In [38]:
```
X_train = ss.fit_transform(X_train)
X_test = ss.transform(X_test)
```

In [39]:
```
X_train
```

Out[39]:
```
array([[ 0.30958074,  0.75972699,  0.27109927, ...,  0.10105866,
         0.44466773,  0.77376994],
       [ 1.60436741,  0.24845846,  1.56250143, ...,  0.68294841,
        -0.5748044 ,  0.41763432],
       [ 1.46674366, -0.24616412,  1.42755441, ...,  1.6437431 ,
         1.12051587, -0.33775396],
       ...,
       [ 0.94433516,  0.0249271 ,  0.93274864, ...,  1.17311909,
         0.04241595, -0.29295703],
       [ 0.54269851, -0.27707803,  0.52381825, ...,  0.08602275,
        -0.17906682, -0.13000818],
       [-0.50211849, -0.41262364, -0.47642548, ..., -0.26822331,
        -0.13509598,  1.56219596]])
```

In [40]:
```
X_test
```

Out[40]:
```
array([[ 3.10137672,  1.35422528,  3.23911602, ...,  2.56845161,
        -0.89074305,  1.18702164],
       [ 1.35439775, -0.07257062,  1.27216086, ...,  0.08902993,
        -0.09926788, -1.03546619],
       [ 2.11554132, -0.46731748,  1.98778904, ...,  0.99419176,
        -0.63831784, -1.06178439],
       ...,
       [-0.5611001 , -1.22114131, -0.61505288, ..., -1.46387893,
        -0.99171314, -1.31768687],
       [ 1.0819589 , -0.56243721,  1.03089193, ...,  0.5656683 ,
        -0.09763933, -0.09249075],
       [-0.64816818,  0.55284158, -0.63222796, ..., -0.42670181,
        -0.97217054,  0.87456304]])
```

ML Models

Logistic Regression

In [41]:
```
# to import import Logistic Regression
from sklearn.linear_model import LogisticRegression
```

In [42]:
```
# to train (fit and transform) the model using Logistic Regression.
lr = LogisticRegression()
lr.fit(X_train,y_train)
```

Out[42]:
```
LogisticRegression()
```

In [43]:
```
# to predict the segregated test dataset.
y_predict = lr.predict(X_test)
```

In [44]:
```
# predicted values
y_predict
```

Out[44]:
```
array([1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0,
       1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0,
       0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0,
       0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0,
       1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0,
       0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0], dtype=int64)
```

In [45]:
```
# original vales to compare and check
y_test
```

Out[45]:
```
82     1
127    1
368    1
84     0
160    0
      ..
90     0
181    1
316    0
444    1
450    0
Name: diagnosis, Length: 171, dtype: int64
```

In [46]:
```
# since we cannot check each and every value we use accuracy score to pre
dict accuracy of model.
from sklearn.metrics import accuracy_score
print(accuracy_score(y_test,y_predict))
```

```
0.9883040935672515
```

In [47]:
```
lr_acc = accuracy_score(y_test,y_predict)
print(lr_acc)
```

```
0.9883040935672515
```

In [48]:
```
results = pd.DataFrame()
results
```

Out[48]:
—

In [49]:
```
tempResults = pd.DataFrame({'Algorithm':['Logistic Regression Method'],
'Accuracy':[lr_acc]})
results = pd.concat( [results, tempResults] )
results = results[['Algorithm','Accuracy']]
results
```

Out[49]:

	Algorithm	Accuracy
0	Logistic Regression Method	0.988304

In [ ]:

Decision Tree Classifier

In [50]:
```
# to train (fit and transform) the model using Decision Tree Classifier Algorithm.
from sklearn.tree import DecisionTreeClassifier
```

In [51]:
```
dtc = DecisionTreeClassifier()
dtc.fit(X_train,y_train)
```

Out[51]:
```
DecisionTreeClassifier()
```

In [52]:
```
y_predict = dtc.predict(X_test)
y_predict
```

Out[52]:
```
array([1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0,
       0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0,
       1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0,
       0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0,
       0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0,
       1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0,
       0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0], dtype=int64)
```

In [53]:
```
y_test
```

Out[53]:
```
82     1
127    1
368    1
84     0
160    0
      ..
90     0
181    1
316    0
444    1
450    0
Name: diagnosis, Length: 171, dtype: int64
```

In [54]:
```
from sklearn.metrics import accuracy_score
print(accuracy_score(y_test,y_predict))
```

```
0.9415204678362573
```

In [55]:
```
dtc_acc = accuracy_score(y_test,y_predict)
print(dtc_acc)
```

```
0.9415204678362573
```

In [56]:
```
tempResults = pd.DataFrame({'Algorithm':['Decision tree Classifier Metho
d'], 'Accuracy':[dtc_acc]})
results = pd.concat( [results, tempResults] )
results = results[['Algorithm','Accuracy']]
results
```

Out[56]:

	Algorithm	Accuracy
0	Logistic Regression Method	0.988304
0	Decision tree Classifier Method	0.941520

Random Forest Classifier

In [58]:
```
# to train (fit and transform) the model using Random Forest Classifier A
lgorithm
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier()
rfc.fit(X_train, y_train)
```

Out[58]:
```
RandomForestClassifier()
```

In [59]:
```
y_pred = rfc.predict(X_test)
y_pred
```

Out[59]:
```
array([1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0,
       1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0,
       0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0,
       0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0,
       1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0], dtype=int64)
```

In [60]:
```
from sklearn.metrics import accuracy_score
print(accuracy_score(y_test, y_pred))
```

```
0.9649122807017544
```

In [61]:
```
rfc_acc = accuracy_score(y_test, y_pred)
print(rfc_acc)
```

```
0.9649122807017544
```

In [62]:
```
tempResults = pd.DataFrame({'Algorithm':['Random Forest Classifier Metho
d'], 'Accuracy':[rfc_acc]})
results = pd.concat( [results, tempResults] )
results = results[['Algorithm','Accuracy']]
results
```

Out[62]:

	Algorithm	Accuracy
0	Logistic Regression Method	0.988304
0	Decision tree Classifier Method	0.941520
0	Random Forest Classifier Method	0.964912

Support Vector Classifier

In [63]:
```
# to train (fit and transform) the model using Support Vector Classifier
 Algorithm
from sklearn import svm
svc = svm.SVC()
svc.fit(X_train,y_train)
```

Out[63]:
```
SVC()
```

In [64]:
```
y_pred = svc.predict(X_test)
y_pred
```

Out[64]:
```
array([1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0,
       1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0,
       0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0,
       0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0,
       1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0], dtype=int64)
```

In [65]:
```
from sklearn.metrics import accuracy_score
print(accuracy_score(y_test, y_pred))
```

```
0.9707602339181286
```

In [66]:
```
svc_acc = accuracy_score(y_test, y_pred)
print(svc_acc)
```

```
0.9707602339181286
```

In [67]:
```
tempResults = pd.DataFrame({'Algorithm':['Support Vector Classifier Method'], 'Accuracy':[svc_acc]})
results = pd.concat( [results, tempResults] )
results = results[['Algorithm','Accuracy']]
results
```

Out[67]:

	Algorithm	Accuracy
0	Logistic Regression Method	0.988304
0	Decision tree Classifier Method	0.941520
0	Random Forest Classifier Method	0.964912
0	Support Vector Classifier Method	0.970760

In [ ]:
```
Hence , as the results shows Logistic Regression Method gives the best accuray.
```