BREAST CANCER PREDICTION

To predict Breast Cancer from the dataset using Machine Learning Models.

MACHINE LEARNING Models

- Logistic Regression
- Decision Tree Classifier
- Random Forest Classifier
- Support Vector Classifier

Weather the person is having BREAST CANCER of the Category:

- BENAINE
- MALIGNANT

In [130]:

```
#To hold Warning Interruption of required installations on versions.

import warnings
warnings.filterwarnings('ignore')
```

In [131]:

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

In [132]:

```
df = pd.read_csv("https://raw.githubusercontent.com/ingledarshan/AIML-B2/main/data.csv")
```

In [133]:

```
df.head()
```

Out[133]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	cor
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	

5 rows × 33 columns

In [134]:

```
df.tail()
```

Out[134]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	cor

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	con
564	926424	M	21.56	22.39	142.00	1479.0	0.11100	0.11590	
565	926682	M	20.13	28.25	131.20	1261.0	0.09780	0.10340	
566	926954	M	16.60	28.08	108.30	858.1	0.08455	0.10230	
567	927241	M	20.60	29.33	140.10	1265.0	0.11780	0.27700	
568	92751	B	7.76	24.54	47.92	181.0	0.05263	0.04362	

5 rows × 33 columns

Weather a person is having breast cancer, and if having BENAINE or MALIGNANT

In [135]:

```
df.columns
```

Out[135]:

```
Index(['id', 'diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
       '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_se',
       '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 [136]:

```
df.info()   #checking the datatype
```

```
<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 [137]:

```
#dropping unnamed column

df["Unnamed: 32"]
```

Out[137]:

```
0      NaN
1      NaN
2      NaN
3      NaN
4      NaN
        ..
564    NaN
565    NaN
566    NaN
567    NaN
568    NaN
Name: Unnamed: 32, Length: 569, dtype: float64
```

In [138]:

```
df = df.drop("Unnamed: 32" , axis = 1)
```

In [139]:

```
df.head()
```

Out[139]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	cor
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	

5 rows × 32 columns

In [140]:

```
df.drop('id', axis=1, inplace=True)

# OR df = df.drop("id" , axis = 1)
```

In [141]:

```
df.head()
```

Out[141]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mea
0	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.300
1	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.086
2	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.197
3	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.241

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mea
4	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.198

5 rows × 31 columns

In [142]:

```
type(df.columns)  #converting to list datatype
```

Out[142]:

```
pandas.core.indexes.base.Index
```

In [143]:

```
lst = list(df.columns)
print(lst)
```

```
['diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean', 'area_mean', 'smoothness_m
ean', 'compactness_mean', 'concavity_mean', 'concave points_mean', 'symmetry_mean', 'frac
tal_dimension_mean', 'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se
', 'compactness_se', 'concavity_se', 'concave points_se', 'symmetry_se', 'fractal_dimensi
on_se', 'radius_worst', 'texture_worst', 'perimeter_worst', 'area_worst', 'smoothness_wor
st', 'compactness_worst', 'concavity_worst', 'concave points_worst', 'symmetry_worst', 'f
ractal_dimension_worst']
```

In [144]:

```
features_mean = lst[1:11]

features_se = lst[11:20]

features_worst = lst[21:]
```

In [145]:

```
print(features_mean)
```

```
['radius_mean', 'texture_mean', 'perimeter_mean', 'area_mean', 'smoothness_mean', 'compac
tness_mean', 'concavity_mean', 'concave points_mean', 'symmetry_mean', 'fractal_dimension
_mean']
```

In [146]:

```
print(features_se)
```

```
['radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se', 'compactness_se',
'concavity_se', 'concave points_se', 'symmetry_se']
```

In [147]:

```
print(features_worst)
```

```
['radius_worst', 'texture_worst', 'perimeter_worst', 'area_worst', 'smoothness_worst', 'c
ompactness_worst', 'concavity_worst', 'concave points_worst', 'symmetry_worst', 'fractal_
dimension_worst']
```

In [148]:

```
df.head(2)
```

Out[148]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mea
0	M	17.99	10.38	122.8	1001.0	0.11840	0.27760	0.300
1	M	20.57	17.77	132.9	1326.0	0.08474	0.07864	0.086

2 rows × 31 columns

In [149]:

```
#to see values inside diagnosis column / unique function
#to see unique values in this function
# M = Malegnant
# B = Beanine

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

Out[149]:

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

In [150]:

```
#to know the values of M and B

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

Out[150]:

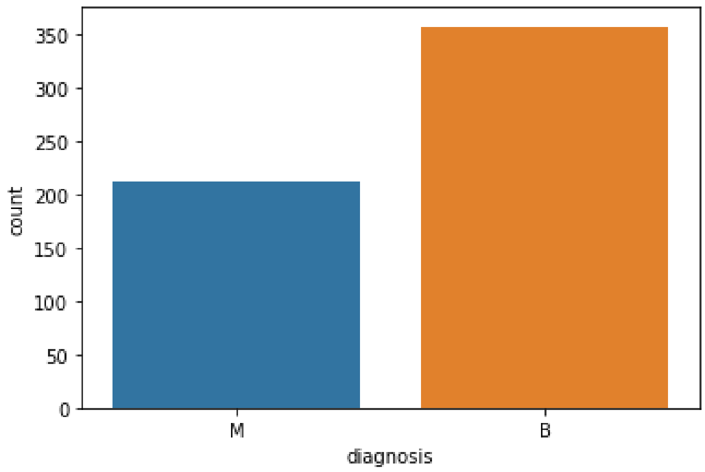
```
B    357
M    212
Name: diagnosis, dtype: int64
```

In [151]:

```
sns.countplot(df["diagnosis"])
```

Out[151]:

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



In [152]:

```
#rows and columns
# 357 + 212

df.shape
```

Out[152]:

```
(569, 31)
```

Explore the data

In [153]:

```
#summary of the numeric columns
df.describe()
```

Out[153]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	pe
count	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	5
mean	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	0.088799	
std	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	0.079720	
min	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	0.000000	
25%	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	0.029560	
50%	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630	0.061540	
75%	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400	0.130700	
max	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	0.426800	

8 rows × 30 columns

In [154]:

```
#correlation plot
```

In [155]:

```
corr = df.corr()
```

In [156]:

```
corr
```

Out[156]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	co
radius_mean	1.000000	0.323782	0.997855	0.987357	0.170581	0.506124	
texture_mean	0.323782	1.000000	0.329533	0.321086	-0.023389	0.236702	
perimeter_mean	0.997855	0.329533	1.000000	0.986507	0.207278	0.556936	
area_mean	0.987357	0.321086	0.986507	1.000000	0.177028	0.498502	
smoothness_mean	0.170581	-0.023389	0.207278	0.177028	1.000000	0.659123	
compactness_mean	0.506124	0.236702	0.556936	0.498502	0.659123	1.000000	
concavity_mean	0.676764	0.302418	0.716136	0.685983	0.521984	0.883121	
concave points_mean	0.822529	0.293464	0.850977	0.823269	0.553695	0.831135	
symmetry_mean	0.147741	0.071401	0.183027	0.151293	0.557775	0.602641	
fractal_dimension_mean	-0.311631	-0.076437	-0.261477	-0.283110	0.584792	0.565369	
radius_se	0.679090	0.275869	0.691765	0.732562	0.301467	0.497473	
texture_se	-0.097317	0.386358	-0.086761	-0.066280	0.068406	0.046205	
perimeter_se	0.674172	0.281673	0.693135	0.726628	0.296092	0.548905	
area_se	0.735864	0.259845	0.744983	0.800086	0.246552	0.455653	
smoothness_se	-0.222600	0.006614	-0.202694	-0.166777	0.332375	0.135299	
compactness_se	0.206000	0.191975	0.250744	0.212583	0.318943	0.738722	
concavity_se	0.194204	0.143293	0.228082	0.207660	0.248396	0.570517	
concave points_se	0.376169	0.163851	0.407217	0.372320	0.380676	0.642262	
symmetry_se	-0.104321	0.009127	-0.081629	-0.072497	0.200774	0.229977	
fractal_dimension_se	-0.042641	0.054458	-0.005523	-0.019887	0.283607	0.507318	
radius_worst	0.969539	0.352573	0.969476	0.962746	0.213120	0.535315	
texture_worst	0.297008	0.912045	0.303038	0.287489	0.036072	0.248133	

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	co
perimeter_worst	0.965137	0.358040	0.970387	0.959120	0.238853	0.590210	
area_worst	0.941082	0.343546	0.941550	0.959213	0.206718	0.509604	
smoothness_worst	0.119616	0.077503	0.150549	0.123523	0.805324	0.565541	
compactness_worst	0.413463	0.277830	0.455774	0.390410	0.472468	0.865809	
concavity_worst	0.526911	0.301025	0.563879	0.512606	0.434926	0.816275	
concave points_worst	0.744214	0.295316	0.771241	0.722017	0.503053	0.815573	
symmetry_worst	0.163953	0.105008	0.189115	0.143570	0.394309	0.510223	
fractal_dimension_worst	0.007066	0.119205	0.051019	0.003738	0.499316	0.687382	

30 rows × 30 columns

In [157]:

```
corr.shape
```

Out[157]:

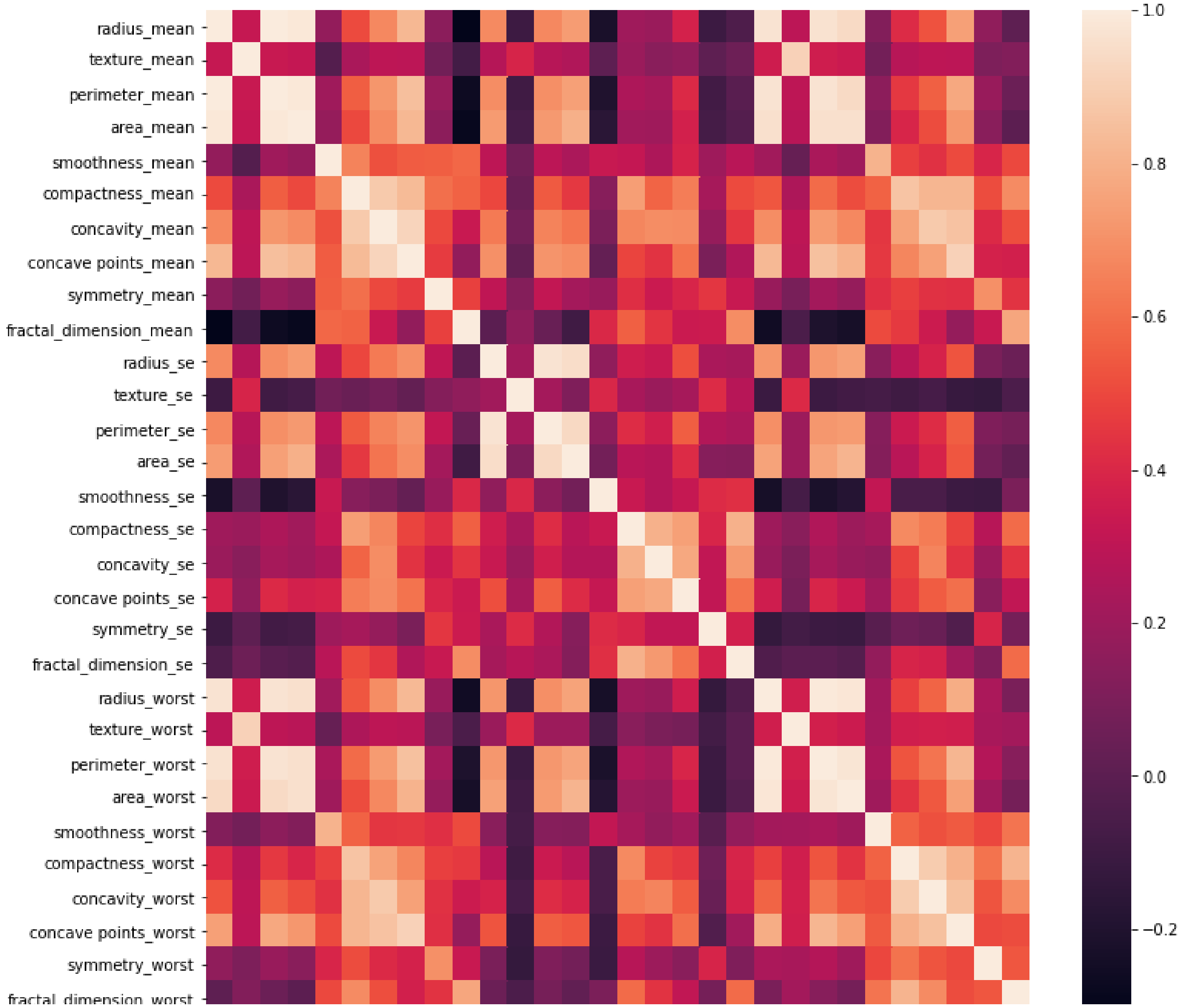
```
(30, 30)
```

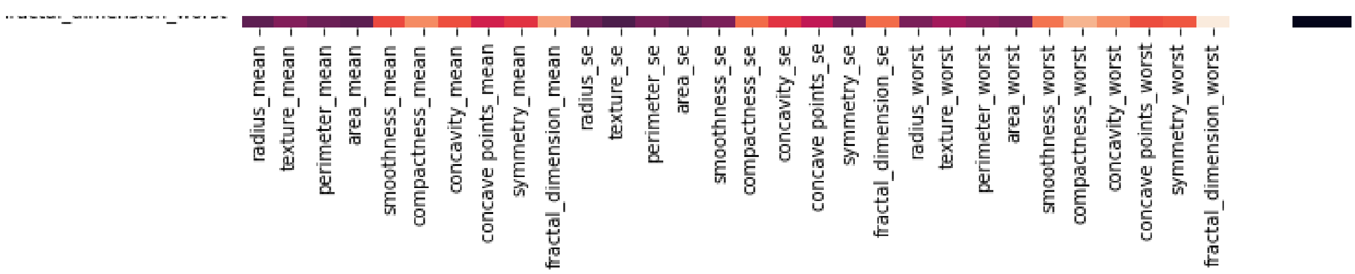
In [158]:

```
plt.figure(figsize=(12,12))
sns.heatmap(corr)
```

Out[158]:

```
<AxesSubplot:>
```





In [159]:

```
# to make M as 1 and B as 0

df["diagnosis"] = df["diagnosis"].map({"M":1,"B":0})
```

In [160]:

```
df.head()
```

Out[160]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mea
0	1	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.300
1	1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.086
2	1	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.197
3	1	11.42	20.38	77.58	386.1	0.14250	0.28390	0.241
4	1	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.198

5 rows × 31 columns

In [161]:

```
# to check values in diagnosis column
df["diagnosis"].unique()
```

Out[161]:

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

In [162]:

```
X = df.drop("diagnosis",axis = 1)
X.head()
```

Out[162]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	con points_r
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.1
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.0
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.1
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.1
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.1

5 rows × 30 columns

In [163]:

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

Out[163]:

```
0    1
1    1
2    1
3    1
4    1
Name: diagnosis, dtype: int64
```

In [164]:

```
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 [165]:

```
y_train.shape
```

Out[165]:

```
(398,)
```

In [166]:

```
y_test.shape
```

Out[166]:

```
(171,)
```

In [167]:

```
#Scaling down the model bringing the values close to 0
#bringing values close to 0

from sklearn.preprocessing import StandardScaler
ss = StandardScaler()
X_train = ss.fit_transform(X_train)
X_test = ss.transform(X_test)
```

In [168]:

```
X_train
```

Out[168]:

```
array([[ 1.51394871, -0.30322167,  1.55895129, ...,  1.63429064,
         1.15502702,  0.44207373],
       [-0.07318757, -0.88809992, -0.14030758, ..., -0.52009115,
        -1.21377856, -0.92844307],
       [-0.53479633, -0.41362038, -0.47594324, ..., -0.64795751,
        -0.55359386,  0.51624415],
       ...,
       [-0.94796466,  0.44842908, -0.87151384, ...,  0.39038983,
         0.08772842,  0.65095074],
       [-1.01635114,  0.22293385, -0.9707165 , ..., -0.46205253,
        -0.47657231,  0.15739017],
       [ 1.72765647,  0.48601162,  1.77389039, ...,  2.29629375,
         4.16515488,  0.99671582]])
```

MACHINE LEARNING Models

Logistic Regression

In [169]:

```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
lr.fit(X_train, y_train)
```

Out[169]:

```
LogisticRegression()
```

In [170]:

```
y_pred = lr.predict(X_test)
y_pred
```

Out[170]:

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

In [171]:

```
y_test
```

Out[171]:

```
205    1
500    0
433    1
480    0
91     1
      ..
168    1
310    0
254    1
187    0
360    0
Name: diagnosis, Length: 171, dtype: int64
```

In [172]:

```
#testing accuracy between y_test and y_pred

from sklearn.metrics import accuracy_score
print(accuracy_score(y_test, y_pred))
```

```
0.9824561403508771
```

In [173]:

```
lr_acc = accuracy_score(y_test, y_pred)
print(lr_acc)
```

```
0.9824561403508771
```

In [174]:

```
#created empty dataframe
results = pd.DataFrame()
```

In [175]:

```
results
```

Out[175]:

_

In [176]:

```
#create one more data frame
#Algorithm and Accuracy as the headings
```

```
#index values for both headings is logistic regression method

tempResults = pd.DataFrame({"Algorithm":["Logistic Regression method"], "Accuracy": [lr_
acc]})
results = pd.concat([results, tempResults])
results = results[["Algorithm","Accuracy"]]
results
```

Out[176]:

	Algorithm	Accuracy
0	Logistic Regression method	0.982456

Decision Tree Classifier

In [177]:

```
from sklearn.tree import DecisionTreeClassifier
dtc = DecisionTreeClassifier()
dtc.fit(X_train, y_train)
```

Out[177]:

```
DecisionTreeClassifier()
```

In [178]:

```
y_pred = dtc.predict(X_test)
y_pred
```

Out[178]:

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

In [179]:

```
#testing accuracy between y_test and y_pred

from sklearn.metrics import accuracy_score
print(accuracy_score(y_test, y_pred))
```

```
0.9298245614035088
```

In [180]:

```
dtc_acc = accuracy_score(y_test, y_pred)
print(dtc_acc)
```

```
0.9298245614035088
```

In [181]:

```
#create one more data frame
#Algorithm and Accuracy as the headings
#index values for both headings is decision tree class method

tempResults = pd.DataFrame({"Algorithm":["decision tree class"], "Accuracy": [dtc_acc]})
results = pd.concat([results, tempResults])
results = results[["Algorithm","Accuracy"]]
results
```

Out[181]:

	Algorithm	Accuracy
0	Logistic Regression method	0.982456
0	decision tree class	0.929825

Random Forest Classifier

In [182]:

```
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier()
rfc.fit(X_train, y_train)
```

Out[182]:

```
RandomForestClassifier()
```

In [183]:

```
y_pred = rfc.predict(X_test)
y_pred
```

Out[183]:

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

In [184]:

```
#testing accuracy between y_test and y_pred

from sklearn.metrics import accuracy_score
print(accuracy_score(y_test, y_pred))
```

```
0.9707602339181286
```

In [185]:

```
rfc_acc = accuracy_score(y_test, y_pred)
print(rfc_acc)
```

```
0.9707602339181286
```

In [186]:

```
#create one more data frame
#Algorithm and Accuracy as the headings
#index values for both headings is random forest classifier method

tempResults = pd.DataFrame({"Algorithm":["random forest classifier"], "Accuracy": [rfc_a
cc]})
results = pd.concat([results, tempResults])
results = results[["Algorithm","Accuracy"]]
results
```

Out[186]:

	Algorithm	Accuracy
0	Logistic Regression method	0.982456
0	decision tree class	0.929825
0	random forest classifier	0.970760

Support Vector Classifier

In [187]:

```
from sklearn import svm
svc = svm.SVC()
svc.fit(X_train, y_train)
```

Out[187]:

```
SVC()
```

In [188]:

```
y_pred = svc.predict(X_test)
y_pred
```

Out[188]:

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

In [189]:

```
#testing accuracy between y_test and y_pred

from sklearn.metrics import accuracy_score
print(accuracy_score(y_test, y_pred))
```

```
0.9590643274853801
```

In [190]:

```
svc_acc = accuracy_score(y_test, y_pred)
print(svc_acc)
```

```
0.9590643274853801
```

In [191]:

```
#create one more data frame
#Algorithm and Accuracy as the headings
#index values for both headings is Support Vector Classifier method

tempResults = pd.DataFrame({"Algorithm":["Support Vector Classifier"], "Accuracy": [svc_
acc]})
results = pd.concat([results, tempResults])
results = results[["Algorithm","Accuracy"]]
results
```

Out[191]:

	Algorithm	Accuracy
0	Logistic Regression method	0.982456
0	decision tree class	0.929825
0	random forest classifier	0.970760
0	Support Vector Classifier	0.959064

After successful execution in multiple ways,

Hence we conclude that LOGISTIC REGRESSION Model is more accurate for this Dataset.