```
# importing necessary libraries
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
```

```
# reading dataset
data = pd.read_csv('/content/Brest cancer.csv')
```

```
#reading first five rows from our dataset

data.head()
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_m
0	842302	M	17.99	10.38	122.80	1001.0	0.11
1	842517	M	20.57	17.77	132.90	1326.0	0.08
2	84300903	M	19.69	21.25	130.00	1203.0	0.10
			11.42	20.38	77.58	386.1	0.14
			20.29	14.34	135.10	1297.0	0.10

Saving...

5 rows × 32 columns

```
# finding the columns/features in our dataset
data.columns
```

```
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'],
      dtype='object')
```

```
#finding the length of the features dataset
len(data.columns)
```

```
32
```

found 32 columns in our dataset

```
#shape of the dataset
data.shape
```

```
(569, 32)
```

findings: Here we have 569 rows and 32 columns in out dataset

```
data.isnull().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
dtype: int64
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
                                    68
                                    :
                                    on-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
dtypes: float64(30), int64(1), object(1)
memory usage: 142.4+ KB
```

```
data.describe()
```

	id	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean
count	5.690000e+02	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.0000
mean	3.037183e+07	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	0.088799	0.0489
std	1.250206e+08	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	0.079720	0.0388
min	8.670000e+03	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	0.000000	0.0000
25%	8.692180e+05	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	0.029560	0.0203
75%	8.813129e+06	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400	0.130700	0.0740

```
data = data.drop('id',axis=1)
```

```
data.head()
```

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

5 rows × 31 columns

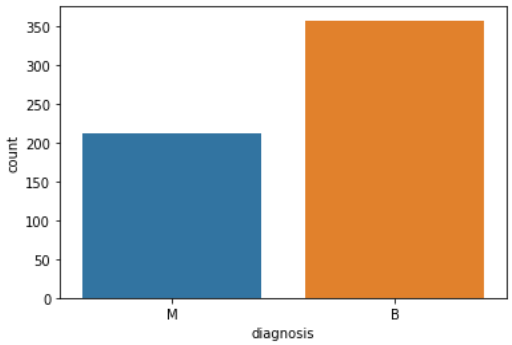
Saving...

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

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

```
#graphical representation of diagnosis
sns.countplot(data['diagnosis'], label = 'count')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0cebb28880>
```




```
#importing label encoder for converting the categorical variables in to numerical variables
from sklearn.preprocessing import LabelEncoder
```

```
le = LabelEncoder()
```

```
le_diagnosis = le.fit_transform(data.iloc[:,0].values)
```

```
data.iloc[:,0].values
```

```
array(['M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M',
       'M', 'M', 'M', 'M', 'M', 'M', 'B', 'B', 'B', 'M', 'M', 'M', 'M',
       'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'B', 'M',
       'M', 'M', 'M', 'M', 'M', 'M', 'M', 'B', 'M', 'B', 'B', 'B', 'B',
       'B', 'M', 'M', 'B', 'M', 'M', 'B', 'B', 'B', 'B', 'M', 'B', 'M',
       'M', 'B', 'B', 'B', 'B', 'M', 'B', 'M', 'M', 'B', 'M', 'B', 'M',
       'M', 'B', 'B', 'B', 'M', 'M', 'B', 'M', 'M', 'M', 'B', 'B', 'B',
```

```
'M', 'B', 'B', 'M', 'M', 'B', 'B', 'B', 'M', 'M', 'B', 'B', 'B',
'B', 'M', 'B', 'B', 'M', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B',
'M', 'M', 'M', 'B', 'M', 'M', 'B', 'B', 'B', 'M', 'M', 'B', 'M',
'B', 'M', 'M', 'B', 'M', 'M', 'B', 'B', 'M', 'B', 'B', 'M', 'B',
'B', 'B', 'B', 'M', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B',
'M', 'B', 'B', 'B', 'B', 'M', 'M', 'B', 'M', 'B', 'B', 'M', 'M',
'B', 'B', 'M', 'M', 'B', 'B', 'B', 'B', 'M', 'B', 'B', 'M', 'M',
'M', 'B', 'M', 'B', 'M', 'B', 'B', 'B', 'M', 'B', 'B', 'M', 'M',
'B', 'M', 'M', 'M', 'M', 'B', 'M', 'M', 'M', 'B', 'M', 'B', 'M',
'B', 'B', 'M', 'B', 'M', 'M', 'M', 'M', 'B', 'B', 'M', 'M', 'B',
'B', 'B', 'M', 'B', 'B', 'B', 'B', 'B', 'M', 'M', 'B', 'B', 'M',
'B', 'B', 'M', 'M', 'B', 'M', 'B', 'B', 'B', 'M', 'B', 'B',
'B', 'B', 'B', 'M', 'B', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M',
'M', 'M', 'M', 'M', 'M', 'M', 'B', 'B', 'B', 'B', 'B', 'B', 'M',
'B', 'M', 'B', 'B', 'M', 'B', 'B', 'M', 'B', 'M', 'M', 'B', 'B',
'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'M', 'B',
'B', 'M', 'B', 'M', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B',
'B', 'B', 'B', 'B', 'B', 'M', 'B', 'B', 'B', 'M', 'B', 'M',
'B', 'B', 'M', 'M', 'B', 'B', 'B', 'B', 'B', 'B', 'M', 'B', 'B',
'B', 'B', 'B', 'B', 'B', 'M', 'B', 'B', 'B', 'B', 'B', 'M', 'B',
'B', 'M', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B',
'B', 'M', 'B', 'M', 'M', 'B', 'M', 'B', 'B', 'B', 'B', 'B', 'M',
'B', 'B', 'M', 'B', 'M', 'B', 'B', 'M', 'B', 'M', 'B', 'B', 'B',
'B', 'B', 'B', 'B', 'B', 'M', 'M', 'B', 'B', 'B', 'B', 'B', 'B',
'M', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'M', 'B',
'B', 'B', 'B', 'B', 'B', 'B', 'M', 'B', 'M', 'B', 'B', 'M', 'B',
                                'M', 'B', 'M', 'B', 'M', 'B', 'B', 'B',
                                'M', 'B', 'M', 'B', 'M', 'M', 'B', 'B',
                                'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B',
'M', 'B', 'M', 'M', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B',
'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B',
'B', 'B', 'B', 'M', 'M', 'M', 'M', 'M', 'M', 'B'], dtype=object)
```

Saving...

```
le_diagnosis
```

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

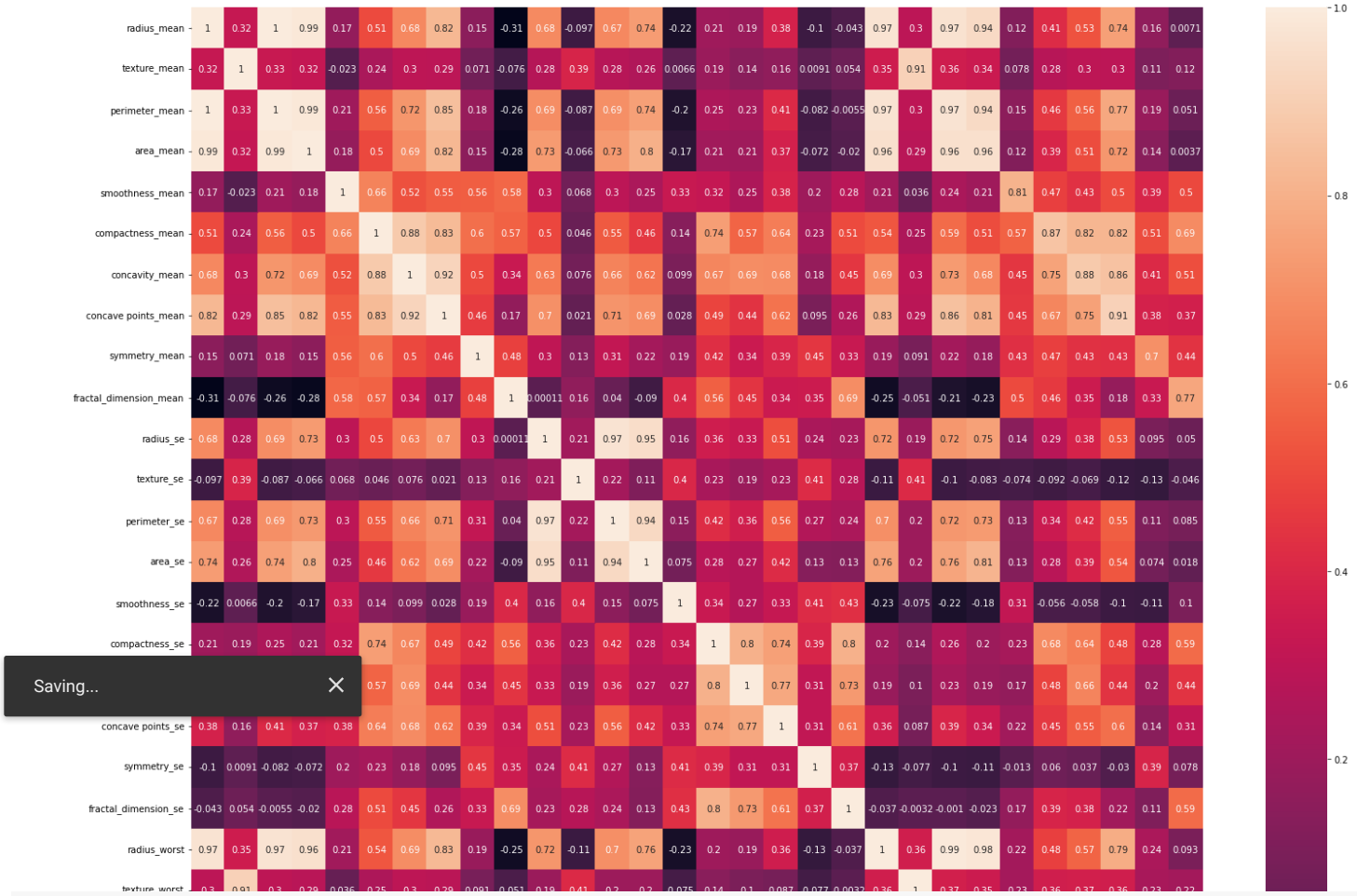
```
# finding the correlation
data.corr().T
```

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean
radius_mean	1.000000	0.323782	0.997855	0.987357	0.170581	0.506124	0.676764	0.8225
texture_mean	0.323782	1.000000	0.329533	0.321086	-0.023389	0.236702	0.302418	0.2934
perimeter_mean	0.997855	0.329533	1.000000	0.986507	0.207278	0.556936	0.716136	0.8509
area_mean	0.987357	0.321086	0.986507	1.000000	0.177028	0.498502	0.685983	0.8232
smoothness_mean	0.170581	-0.023389	0.207278	0.177028	1.000000	0.659123	0.521984	0.5536
compactness_mean	0.506124	0.236702	0.556936	0.498502	0.659123	1.000000	0.883121	0.8311
concavity_mean	0.676764	0.302418	0.716136	0.685983	0.521984	0.883121	1.000000	0.9213
concave points_mean	0.822529	0.293464	0.850977	0.823269	0.553695	0.831135	0.921391	1.0000
symmetry_mean	0.147741	0.071401	0.183027	0.151293	0.557775	0.602641	0.500667	0.4624
fractal_dimension_mean	-0.311631	-0.076437	-0.261477	-0.283110	0.584792	0.565369	0.336783	0.1669
radius_se	0.679090	0.275869	0.691765	0.732562	0.301467	0.497473	0.631925	0.6980
texture_se	-0.097317	0.386358	-0.086761	-0.066280	0.068406	0.046205	0.076218	0.0214
perimeter_se	0.674172	0.281673	0.693135	0.726628	0.296092	0.548905	0.660391	0.7106
area_se	0.735864	0.259845	0.744983	0.800086	0.246552	0.455653	0.617427	0.6902
smoothness_se	-0.222600	0.006614	-0.202694	-0.166777	0.332375	0.135299	0.098564	0.0276
	000	0.191975	0.250744	0.212583	0.318943	0.738722	0.670279	0.4904
	204	0.143293	0.228082	0.207660	0.248396	0.570517	0.691270	0.4391
concave points_se	0.376169	0.163851	0.407217	0.372320	0.380676	0.642262	0.683260	0.6156
symmetry_se	-0.104321	0.009127	-0.081629	-0.072497	0.200774	0.229977	0.178009	0.0953
fractal_dimension_se	-0.042641	0.054458	-0.005523	-0.019887	0.283607	0.507318	0.449301	0.2575
radius_worst	0.969539	0.352573	0.969476	0.962746	0.213120	0.535315	0.688236	0.8303
texture_worst	0.297008	0.912045	0.303038	0.287489	0.036072	0.248133	0.299879	0.2927
perimeter_worst	0.965137	0.358040	0.970387	0.959120	0.238853	0.590210	0.729565	0.8559
area_worst	0.941082	0.343546	0.941550	0.959213	0.206718	0.509604	0.675987	0.8096
smoothness_worst	0.119616	0.077503	0.150549	0.123523	0.805324	0.565541	0.448822	0.4527
compactness_worst	0.413463	0.277830	0.455774	0.390410	0.472468	0.865809	0.754968	0.6674

Saving...

```
#correlation matrix for all the features (Heat Map)

plt.figure(figsize=(25,25))
sns.heatmap(data.corr(),annot=True)
plt.savefig('correlation-matrix.png')
```



Saving...

```
# Remove the labels which is having high correlation co-efficient

#split the data into features and target variable
x = data.iloc[:,1:32].values
y = data.iloc[:,0].values
```

```
x
```

```
array([[1.799e+01, 1.038e+01, 1.228e+02, ..., 2.654e-01, 4.601e-01,
        1.189e-01],
       [2.057e+01, 1.777e+01, 1.329e+02, ..., 1.860e-01, 2.750e-01,
        8.902e-02],
       [1.969e+01, 2.125e+01, 1.300e+02, ..., 2.430e-01, 3.613e-01,
        8.758e-02],
       ...,
       [1.660e+01, 2.808e+01, 1.083e+02, ..., 1.418e-01, 2.218e-01,
        7.820e-02],
       [2.060e+01, 2.933e+01, 1.401e+02, ..., 2.650e-01, 4.087e-01,
        1.240e-01],
       [7.760e+00, 2.454e+01, 4.792e+01, ..., 0.000e+00, 2.871e-01,
        7.039e-02]])
```

```
y
```

```
array(['M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M',
       'M', 'M', 'M', 'M', 'M', 'M', 'B', 'B', 'B', 'M', 'M', 'M', 'M',
       'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'B', 'M',
       'M', 'M', 'M', 'M', 'M', 'M', 'M', 'B', 'M', 'B', 'B', 'B', 'B',
       'B', 'M', 'M', 'B', 'M', 'M', 'B', 'B', 'B', 'B', 'M', 'B', 'M',
       'M', 'B', 'B', 'B', 'B', 'M', 'B', 'M', 'M', 'B', 'M', 'B', 'M',
       'M', 'B', 'B', 'B', 'M', 'M', 'B', 'M', 'M', 'M', 'B', 'B', 'B',
       'M', 'B', 'B', 'M', 'M', 'B', 'B', 'B', 'M', 'M', 'B', 'B', 'B',
       'B', 'M', 'B', 'B', 'M', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B',
       'M', 'M', 'M', 'B', 'M', 'M', 'B', 'B', 'B', 'M', 'M', 'B', 'M',
       'B', 'M', 'M', 'B', 'M', 'B', 'B', 'M', 'B', 'B', 'M', 'B', 'B',
       'B', 'B', 'B', 'M', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B',
       'M', 'B', 'B', 'B', 'B', 'M', 'M', 'B', 'M', 'B', 'B', 'M', 'M',
       'B', 'B', 'M', 'M', 'B', 'B', 'B', 'B', 'M', 'B', 'B', 'M', 'M',
       'M', 'B', 'M', 'B', 'M', 'B', 'B', 'B', 'M', 'B', 'B', 'M', 'M',
```

```
       'B', 'M', 'M', 'M', 'M', 'B', 'M', 'M', 'M', 'B', 'M', 'B', 'M',
       'B', 'B', 'M', 'B', 'M', 'M', 'M', 'M', 'B', 'B', 'M', 'M', 'B',
       'B', 'B', 'M', 'B', 'B', 'B', 'B', 'B', 'M', 'M', 'B', 'B', 'M',
       'B', 'B', 'M', 'M', 'B', 'M', 'B', 'B', 'B', 'B', 'M', 'B', 'B',
       'B', 'B', 'B', 'M', 'B', 'M', 'M', 'M', 'M', 'M', 'M', 'M', 'M',
       'M', 'M', 'M', 'M', 'M', 'M', 'B', 'B', 'B', 'B', 'B', 'B', 'M',
       'B', 'M', 'B', 'B', 'M', 'B', 'B', 'M', 'B', 'M', 'M', 'B', 'B',
       'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'M', 'B',
       'B', 'M', 'B', 'M', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B',
       'B', 'B', 'B', 'B', 'B', 'M', 'B', 'B', 'B', 'M', 'B', 'M', 'B',
       'B', 'B', 'B', 'M', 'M', 'M', 'B', 'B', 'B', 'B', 'M', 'B', 'M',
       'B', 'M', 'B', 'B', 'B', 'M', 'B', 'B', 'B', 'B', 'B', 'B', 'B',
       'M', 'M', 'M', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B',
       'B', 'M', 'M', 'B', 'M', 'M', 'M', 'B', 'M', 'M', 'B', 'B', 'B',
       'B', 'B', 'M', 'B', 'B', 'B', 'B', 'B', 'M', 'B', 'B', 'B', 'M',
       'B', 'B', 'M', 'M', 'B', 'B', 'B', 'B', 'B', 'B', 'M', 'B', 'B',
       'B', 'B', 'B', 'B', 'B', 'M', 'B', 'B', 'B', 'B', 'B', 'M', 'B',
       'B', 'M', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B',
       'B', 'M', 'B', 'M', 'M', 'B', 'M', 'B', 'B', 'B', 'B', 'B', 'M',
       'B', 'B', 'M', 'B', 'M', 'B', 'B', 'M', 'B', 'M', 'B', 'B', 'B',
       'B', 'B', 'B', 'B', 'B', 'M', 'M', 'B', 'B', 'B', 'B', 'B', 'B',
       'M', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'M', 'B',
       'B', 'B', 'B', 'B', 'B', 'B', 'M', 'B', 'M', 'B', 'B', 'M', 'B',
       'B', 'B', 'B', 'B', 'M', 'M', 'B', 'M', 'B', 'M', 'B', 'B', 'B',
       'B', 'B', 'M', 'B', 'B', 'M', 'B', 'M', 'B', 'M', 'M', 'B', 'B',
       'B', 'M', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B',
       'M', 'B', 'M', 'M', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B',
       'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B',
       'B', 'B', 'B', 'M', 'M', 'M', 'M', 'M', 'M', 'B'], dtype=object)
```

Saving...

```
train_test_split
```

```
x_train, x_test, y_train, y_test = train_test_split(
 x, y, test_size=0.2, random_state=42)
```

```
# importing standarscaler
from sklearn.preprocessing import StandardScaler
```

```
sc = StandardScaler()
```

```
x_train = sc.fit_transform(x_train)
```

```
x_train
```

```
array([[-1.44075296, -0.43531947, -1.36208497, ...,  0.9320124 ,
         2.09724217,  1.88645014],
       [ 1.97409619,  1.73302577,  2.09167167, ...,  2.6989469 ,
         1.89116053,  2.49783848],
       [-1.39998202, -1.24962228, -1.34520926, ..., -0.97023893,
         0.59760192,  0.0578942 ],
       ...,
       [ 0.04880192, -0.55500086, -0.06512547, ..., -1.23903365,
        -0.70863864, -1.27145475],
       [-0.03896885,  0.10207345, -0.03137406, ...,  1.05001236,
         0.43432185,  1.21336207],
       [-0.54860557,  0.31327591, -0.60350155, ..., -0.61102866,
        -0.3345212 , -0.84628745]])
```

```
x_test = sc.transform(x_test)
```

```
x_test
```

```
array([[-0.46649743, -0.13728933, -0.44421138, ..., -0.19435087,
         0.17275669,  0.20372995],
       [ 1.36536344,  0.49866473,  1.30551088, ...,  0.99177862,
        -0.561211  , -1.00838949],
       [ 0.38006578,  0.06921974,  0.40410139, ...,  0.57035018,
        -0.10783139, -0.20629287],
       ...,
       [-0.73547237, -0.99852603, -0.74138839, ..., -0.27741059,
        -0.3820785 , -0.32408328],
       [ 0.02898271,  2.0334026 ,  0.0274851 , ..., -0.49027026,
```

```
             -1.60905688, -0.33137507],
            [ 1.87216885,  2.80077153,  1.80354992, ...,  0.7925579 ,
             -0.05868885, -0.09467243]])
```

```
#defining all the necessary models

def models(x_train, y_train):
 # train using Logistic Regression
  from sklearn.linear_model import LogisticRegression
  model1 = LogisticRegression()
  model1.fit(x_train,y_train)

  #train using KNN
  from sklearn.neighbors import KNeighborsClassifier

  model2 = KNeighborsClassifier()
  model2.fit(x_train, y_train)
  #train using GaussianNB
  from sklearn.naive_bayes import GaussianNB
  model3 = GaussianNB()
  model3.fit(x_train, y_train)

  #train using Decision Tree
  from sklearn.tree import DecisionTreeClassifier
  model4 = DecisionTreeClassifier()
  model4.fit(x_train,y_train)

  # train using RandomForest
  mForestClassifier
  model5.fit(x_train,y_train)

  #train using SVC
  from sklearn.svm import SVC
  model6 = SVC()
  model6.fit(x_train,y_train)

  print('[0] Logistic Regression Training Accuracy', model1.score(x_train, y_train))
  print('[1] KNN training accuracy', model2.score(x_train,y_train))
  print('[2] Guassian NB training accuracy',model3.score(x_train,y_train))
  print('[3] decision tree training accuracy', model4.score(x_train,y_train))
  print('[4] random forest training accuracy',model5.score(x_train,y_train))
  print('[5] svc training accuracy', model6.score(x_train,y_train))

  return model1,model2,model3,model4,model5,model6
```

Saving...

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

```
[0] Logistic Regression Training Accuracy 0.9868131868131869
[1] KNN training accuracy 0.9802197802197802
[2] Guassian NB training accuracy 0.9362637362637363
[3] decision tree training accuracy 1.0
[4] random forest training accuracy 1.0
[5] svc training accuracy 0.989010989010989
```

```
# confusion Matrix for all algorithms that we used in models
# importing 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+FP+FN)))
  print()
```

```
[[70  1]
 [ 2 41]]
model[0] testing accuracy = "0.9736842105263158!"

[[68  3]
```

```
 [ 3 40]]
model[1] testing accuracy = "0.9473684210526315!"

[[70  1]
 [ 3 40]]
model[2] testing accuracy = "0.9649122807017544!"

[[67  4]
 [ 3 40]]
model[3] testing accuracy = "0.9385964912280702!"

[[70  1]
 [ 3 40]]
model[4] testing accuracy = "0.9649122807017544!"

[[71  0]
 [ 2 41]]
model[5] testing accuracy = "0.9824561403508771!"
```

```
# importing classificaiton_report and accuracy_score

from sklearn.metrics import classification_report, accuracy_score
```

```
for i in range(len(model)):
  print('model',i)
  #print classification report
  print(classification_report(y_test,model[i].predict(x_test)))
  print(accuracy_score(y_test,model[i].predict(x_test)))
  print()
```

Saving...

```
              B       0.96      0.96      0.96        71
              M       0.93      0.93      0.93        43

       accuracy                           0.95       114
      macro avg       0.94      0.94      0.94       114
   weighted avg       0.95      0.95      0.95       114

   0.9473684210526315

   model 2
                 precision    recall  f1-score   support

              B       0.96      0.99      0.97        71
              M       0.98      0.93      0.95        43

       accuracy                           0.96       114
      macro avg       0.97      0.96      0.96       114
   weighted avg       0.97      0.96      0.96       114

   0.9649122807017544

   model 3
                 precision    recall  f1-score   support

              B       0.96      0.94      0.95        71
              M       0.91      0.93      0.92        43

       accuracy                           0.94       114
      macro avg       0.93      0.94      0.93       114
   weighted avg       0.94      0.94      0.94       114

   0.9385964912280702
```

```
              B       0.97      1.00      0.99        71
              M       1.00      0.95      0.98        43

       accuracy                           0.98       114
      macro avg       0.99      0.98      0.98       114
   weighted avg       0.98      0.98      0.98       114

   0.9824561403508771
```

```
#predictions of every model
for i in range(len(model)):
  y_pred = model[i].predict(x_test)
  print(y_pred)
  print()
print(y_test)
```

```
['B' 'M' 'M' 'B' 'B' 'M' 'M' 'M' 'B' 'B' 'B' 'M' 'B' 'M' 'B' 'M' 'B' 'B'
 'B' 'M' 'B' 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'B'
 'M' 'B' 'M' 'B' 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'M' 'M' 'B' 'B'
 'B' 'B' 'B' 'M' 'M' 'B' 'B' 'M' 'M' 'B' 'B' 'B' 'M' 'M' 'B' 'B' 'M' 'M'
 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'B' 'M' 'B' 'M' 'M' 'M' 'M' 'M' 'M' 'B' 'B'
 'B' 'B' 'B' 'B' 'B' 'B' 'M' 'M' 'B' 'M' 'M' 'B' 'M' 'M' 'B' 'B' 'B' 'M'
 'B' 'B' 'M' 'B' 'M' 'M']

['B' 'M' 'M' 'B' 'B' 'M' 'M' 'M' 'M' 'B' 'B' 'M' 'B' 'B' 'B' 'M' 'B' 'B'
 'B' 'M' 'B' 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'B'
 'M' 'B' 'M' 'B' 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'M' 'M' 'B' 'B'
 'B' 'B' 'B' 'M' 'M' 'B' 'B' 'M' 'M' 'B' 'B' 'B' 'M' 'M' 'B' 'B' 'M' 'M'
 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'B' 'M' 'B' 'M' 'M' 'M' 'M' 'M' 'M' 'B' 'B'
 'B' 'B' 'B' 'B' 'B' 'B' 'M' 'M' 'B' 'M' 'M' 'B' 'M' 'M' 'B' 'B' 'B' 'M'
 'B' 'B' 'M' 'B' 'B' 'M']

['B' 'M' 'M' 'B' 'B' 'M' 'M' 'M' 'M' 'B' 'B' 'M' 'B' 'M' 'B' 'M' 'B' 'B'
 'B' 'M' 'B' 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'B'
 'M' 'B' 'M' 'B' 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'M' 'M' 'B' 'B'
 'B' 'B' 'B' 'M' 'M' 'B' 'B' 'M' 'M' 'B' 'B' 'B' 'M' 'M' 'B' 'B' 'M' 'M'
 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'B' 'M' 'B' 'B' 'M' 'M' 'M' 'M' 'M' 'B' 'B'
 'B' 'B' 'B' 'B' 'B' 'B' 'M' 'M' 'B' 'M' 'M' 'B' 'M' 'M' 'B' 'B' 'B' 'M'
 'B' 'B' 'M' 'B' 'B' 'M']

['B' 'M' 'M' 'B' 'B' 'M' 'M' 'M' 'M' 'B' 'B' 'M' 'B' 'M' 'B' 'M' 'B' 'B'
 'B' 'M' 'B' 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'B' 'M' 'B' 'B' 'M' 'B' 'B' 'B'
 'M' 'B' 'M' 'B' 'B' 'M' 'B' 'B' 'B' 'B' 'M' 'B' 'B' 'B' 'M' 'M' 'B' 'B'
 'B' 'B' 'B' 'M' 'M' 'B' 'B' 'M' 'M' 'B' 'B' 'B' 'M' 'M' 'B' 'B' 'M' 'M'
 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'B' 'M' 'B' 'M' 'M' 'M' 'M' 'M' 'M' 'B' 'B'
 'B' 'B' 'B' 'B' 'B' 'B' 'M' 'M' 'B' 'M' 'M' 'B' 'M' 'M' 'B' 'B' 'B' 'M'
 'M' 'B' 'M' 'B' 'B' 'M']

['B' 'M' 'M' 'B' 'B' 'M' 'M' 'M' 'M' 'B' 'B' 'M' 'B' 'M' 'B' 'M' 'B' 'B'
 'B' 'M' 'B' 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'B'
 'M' 'B' 'M' 'B' 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'M' 'M' 'B' 'B'
 'B' 'B' 'B' 'M' 'M' 'B' 'B' 'M' 'M' 'B' 'B' 'B' 'M' 'M' 'B' 'B' 'M' 'M'
 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'B' 'M' 'B' 'B' 'M' 'M' 'M' 'M' 'M' 'B' 'B'
 'B' 'B' 'B' 'B' 'B' 'B' 'M' 'M' 'B' 'M' 'M' 'B' 'M' 'M' 'B' 'B' 'B' 'M'
 'B' 'B' 'M' 'B' 'B' 'M']

['B' 'M' 'M' 'B' 'B' 'M' 'M' 'M' 'B' 'B' 'B' 'M' 'B' 'M' 'B' 'M' 'B' 'B'
 'B' 'M' 'M' 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'B'
 'M' 'B' 'M' 'B' 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'M' 'M' 'B' 'B'
 'B' 'B' 'B' 'M' 'M' 'B' 'B' 'M' 'M' 'B' 'B' 'B' 'M' 'M' 'B' 'B' 'M' 'M'
 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'B' 'M' 'B' 'M' 'M' 'M' 'M' 'M' 'M' 'B' 'B'
 'B' 'B' 'B' 'B' 'B' 'B' 'M' 'M' 'B' 'M' 'M' 'B' 'M' 'M' 'B' 'B' 'B' 'M'
 'B' 'B' 'M' 'B' 'B' 'M']

['B' 'M' 'M' 'B' 'B' 'M' 'M' 'M' 'B' 'B' 'B' 'M' 'B' 'M' 'B' 'M' 'B' 'B'
 'B' 'M' 'M' 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'B'
 'M' 'B' 'M' 'B' 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'M' 'M' 'B' 'B'
 'B' 'B' 'B' 'M' 'M' 'B' 'B' 'M' 'M' 'B' 'B' 'B' 'M' 'M' 'B' 'B' 'M' 'M'
 'B' 'M' 'B' 'B' 'B' 'M' 'B' 'B' 'M' 'B' 'M' 'M' 'M' 'M' 'M' 'M' 'B' 'B'
 'B' 'B' 'B' 'B' 'B' 'B' 'M' 'M' 'B' 'M' 'M' 'B' 'M' 'M' 'B' 'B' 'B' 'M'
 'B' 'B' 'M' 'B' 'B' 'M']
```

✓ 0s completed at 6:02 PM

Saving...