```
Start coding or generate with AI.
```

ABOUT THE DATASET

This dataset primarily focuses on the diagnosis of breast cancer, specifically distinguishing between malignant (cancerous) and benign (non-cancerous) tumors based on various attributes derived from digitized images of fine needle aspirates (FNA) of breast masses

Double-click (or enter) to edit

IMPORTING LIBRARIES AND LOADING DATA

```
import numpy as np
import pandas as pd
df=pd.read_csv('/content/data.csv')
df
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	p
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	
...	...	...	...	...	...	...	...	...	...	
564	926424	M	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	
565	926682	M	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	
566	926954	M	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	
567	927241	M	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	
568	92751	B	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	

569 rows × 33 columns

DATA INFORMATIONS

```
df.head()
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	poi
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	

5 rows × 33 columns

```
df.tail()
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	poi
564	926424	M	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	
565	926682	M	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	
566	926954	M	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	
567	927241	M	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	
568	92751	B	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	

5 rows × 33 columns

```
df.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', 'Unnamed: 32'],
      dtype='object')
```

```
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
```

DROPPING UNWANTED COLUMNS

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

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean
0	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710
1	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017
2	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790
3	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520
4	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430
...	...	...	...	...	...	...	...	...	...
564	M	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890
565	M	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791
566	M	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302
567	M	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200
568	B	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000

569 rows × 31 columns

```
df.dtypes
```

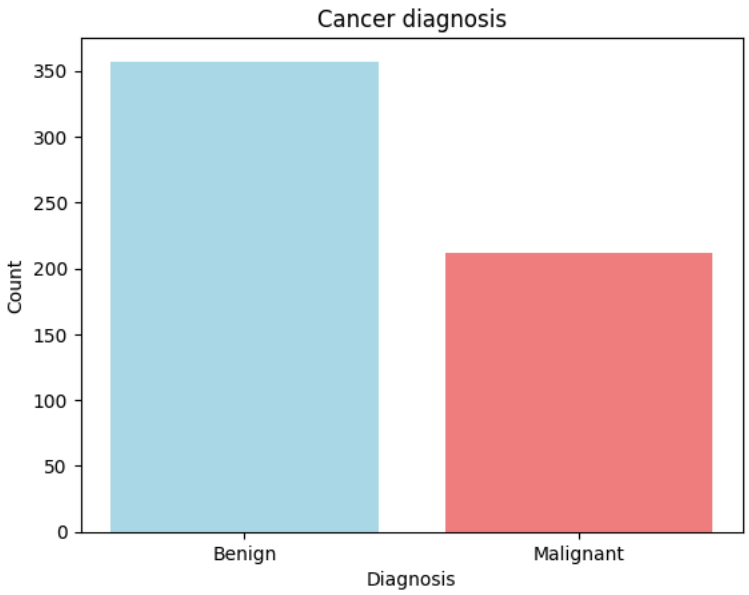
```
diagnosis                 object
radius_mean              float64
```

```
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
```

DATA VISUALIZATION

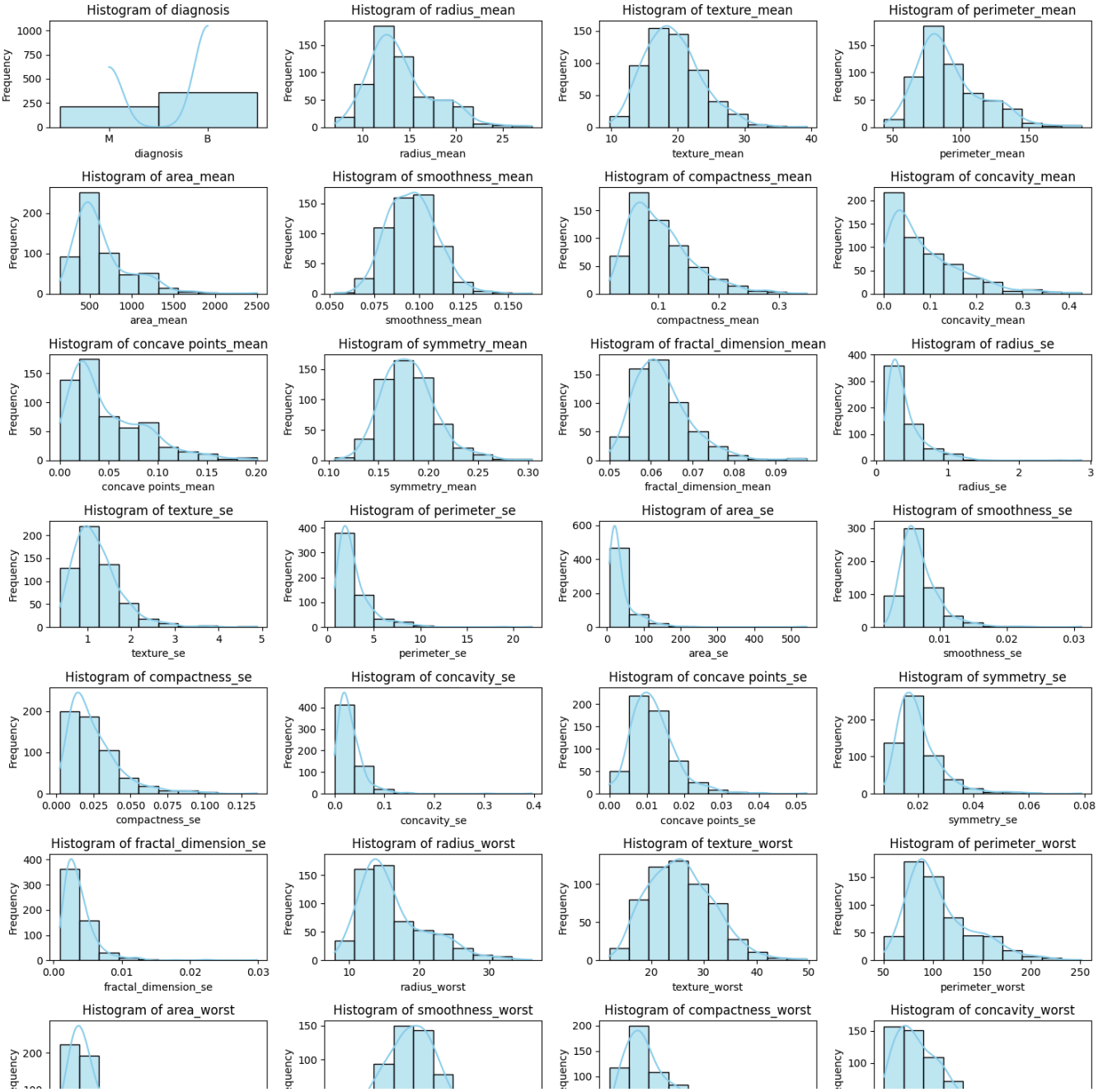
```
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.graph_objects as go
import plotly.express as px
```

```
diabetes_counts = df['diagnosis'].value_counts()
plt.bar(diabetes_counts.index, diabetes_counts.values, color=['lightblue', 'lightcoral'])
plt.xlabel('Diagnosis')
plt.ylabel('Count')
plt.title('Cancer diagnosis')
plt.xticks(['B', 'M'], ['Benign', 'Malignant'])
plt.show()
```



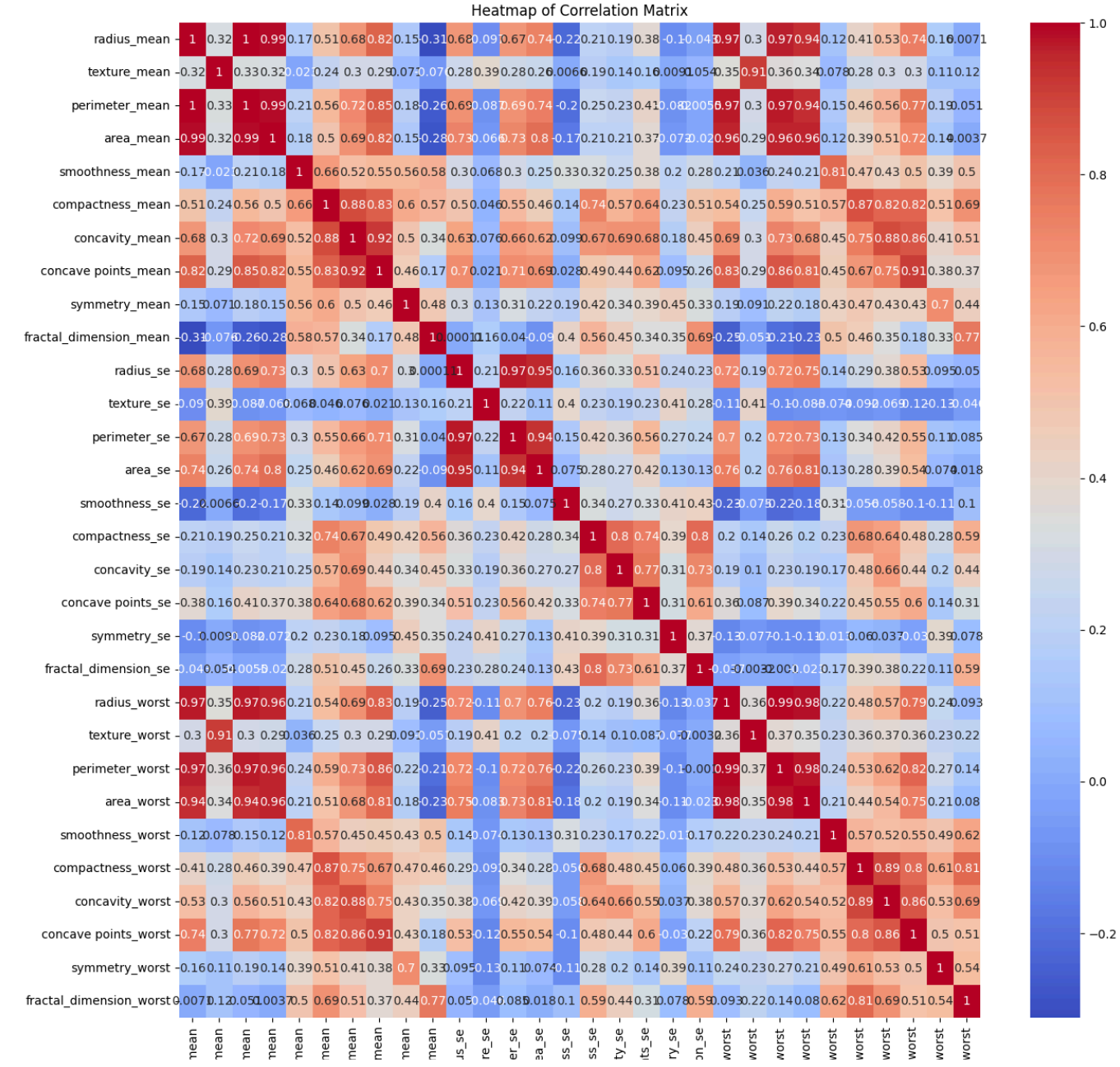
According to the plot above, we can see that the number of benign is more than malignant

```
fig,axes = plt.subplots(nrows=8,ncols=4,figsize=(15,18))

column_names=df.columns

for i,column in enumerate(column_names):
    row = i // 4  # Calculate row index
    col = i % 4
    sns.histplot(data=df[column], bins=10, color='skyblue',kde=True, edgecolor='black', ax=axes[row, col])

    # Set labels and title
    axes[row, col].set_xlabel(f'{column}')
    axes[row, col].set_ylabel('Frequency')
    axes[row, col].set_title(f'Histogram of {column}')

# Adjust layout
plt.tight_layout()

# Show the plots
plt.show()
```




```
plt.figure(figsize=(15,15))

correlation_matrix = df.corr()
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title('Heatmap of Correlation Matrix')
plt.show()
```

```
<ipython-input-53-268270cf5de5>:3: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future ver
  correlation_matrix = df.corr()
```



SEPARATING INPUT AND OUTPUT AS X AND Y

```
x=df.drop(['diagnosis'],axis=1)
x
```

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	0.
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	0.
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	0.
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	0.
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	0.
...	...	...	...	...	...	...	...	...	
564	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	0.
565	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	0.
566	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	0.
567	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	0.
568	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	0.

569 rows × 30 columns

```
y=df['diagnosis']
y
```

```
0      M
1      M
2      M
3      M
4      M
      ..
564    M
565    M
566    M
567    M
568    B
Name: diagnosis, Length: 569, dtype: object
```

TRAIN TEST SPLIT

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.30,random_state=42)
```

NORMALIZATION

```
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
scaler.fit(x_train)
x_train=scaler.transform(x_train)
x_test=scaler.transform(x_test)
x_train
```

```
array([[-0.12348985, -0.29680142, -0.17050713, ..., -0.84082156,
        -0.8563616 , -0.76574773],
       [-0.22826757, -0.65795149, -0.25377521, ..., -0.37706655,
        -1.3415819 , -0.41480748],
       [ 0.14553402, -1.23056444,  0.24583328, ..., -0.04762652,
        -0.08997059,  0.4882635 ],
       ...,
       [ 0.03226081, -0.55578404, -0.08064356, ..., -1.26179013,
        -0.6828391 , -1.27672587],
       [-0.05552593,  0.10949242, -0.04684166, ...,  1.07924018,
         0.4755842 ,  1.25530227],
       [-0.56525537,  0.32333128, -0.619825  , ..., -0.61952313,
        -0.30366032, -0.84348042]])
```

HYPER PARAMETER TUNING

```
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
knn=KNeighborsClassifier()
param={'n_neighbors':[3,5,7,9],'weights':['uniform','distance']}
gd=GridSearchCV(knn,param,cv=10,scoring='accuracy')
gd.fit(x_train,y_train)
print(gd.best_params_)
```

```
{'n_neighbors': 5, 'weights': 'uniform'}
```

MODEL CREATION AND PERFORMANCE EVALUATION

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix,accuracy_score
from sklearn.metrics import classification_report
k_model=KNeighborsClassifier(n_neighbors=7)
n_model=GaussianNB()
s_model=SVC()
r_model=RandomForestClassifier()
d_model=DecisionTreeClassifier(criterion='entropy')
lst_model=[k_model,n_model,r_model,s_model,d_model]


for i in lst_model:
  print('model is',i)
  i.fit(x_train,y_train)
  y_pred=i.predict(x_test)
  print("*"*100)
  print(confusion_matrix(y_test,y_pred))
  print("Accuracy score is",accuracy_score(y_test,y_pred))
  print(".......classification Report........")
  print(classification_report(y_test,y_pred))
```

```
model is KNeighborsClassifier(n_neighbors=7)
****************************************************************************************************
[[105   3]
 [  4  59]]
Accuracy score is 0.9590643274853801
.......classification Report........
              precision    recall  f1-score   support

           B       0.96      0.97      0.97       108
           M       0.95      0.94      0.94        63

    accuracy                           0.96       171
   macro avg       0.96      0.95      0.96       171
weighted avg       0.96      0.96      0.96       171

model is GaussianNB()
****************************************************************************************************
[[103   5]
 [  6  57]]
Accuracy score is 0.935672514619883
.......classification Report........
              precision    recall  f1-score   support

           B       0.94      0.95      0.95       108
           M       0.92      0.90      0.91        63

    accuracy                           0.94       171
   macro avg       0.93      0.93      0.93       171
weighted avg       0.94      0.94      0.94       171

model is RandomForestClassifier()
****************************************************************************************************
[[107   1]
 [  4  59]]
Accuracy score is 0.9707602339181286
.......classification Report........
              precision    recall  f1-score   support

           B       0.96      0.99      0.98       108
           M       0.98      0.94      0.96        63

    accuracy                           0.97       171
   macro avg       0.97      0.96      0.97       171
weighted avg       0.97      0.97      0.97       171
```