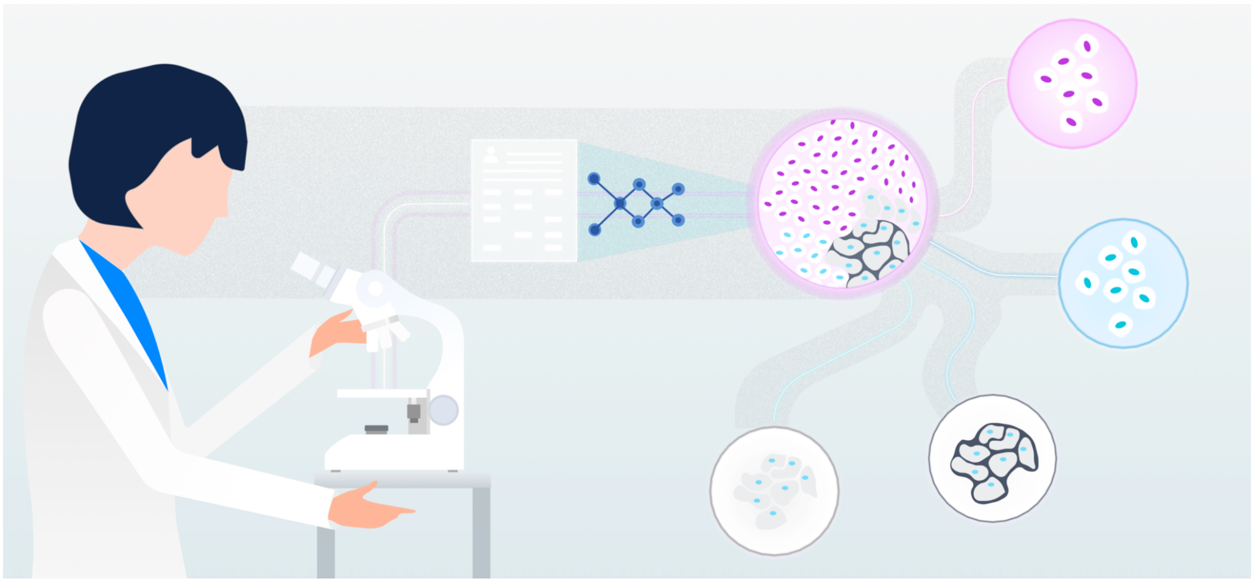
Machine learning

BREAST CANCER PREDICTION



BY:

-GUNJAN KHATRI

-BHAKTI KHOKAD

-PRERNA KOTHARI

-PRANJAL MOGDUL

PROBLEM STATEMENT

The objective of this project is investigating the probability of predicting the chances of breast cancer from the given characteristics of breast mass computed from the dataset.

INTRODUCTION

Breast cancer is the most common cancer amongst women in the world. It accounts for 25% of all cancer cases, and affected over 2.1 Million people in 2015 alone. It starts when cells in the breast begin to grow out of control. These cells usually form tumors that can be seen via X-ray or felt as lumps in the breast area.

The key challenge against it’s detection is how to classify tumors into malignant (cancerous) or benign(non cancerous).

DATASET INFORMATION

https://www.kaggle.com/datasets/jainilcoder/breast-cancer-dataset

The Dataset contains 32 Columns and 570 rows consisting all the parameters used for detecting a Breast Cancer.

The Breast Cancer Wisconsin (Diagnostic) DataSet, obtained from Kaggle, contains features computed from a digitized image of a fine needle aspirate (FNA) of a breast mass and describe characteristics of the cell nuclei present in the image.

Attribute information

- ID number

- Diagnosis (M = malignant, B = benign)
- Ten real-valued features are computed for each cell nucleus:
    - radius (mean of distances from center to points on the perimeter)
    - texture (standard deviation of gray-scale values)
    - perimeter
    - area
    - smoothness (local variation in radius lengths)
    - compactness (perimeter^2 / area - 1.0)
    - concavity (severity of concave portions of the contour)
    - concave points (number of concave portions of the contour)
    - symmetry
    - fractal dimension (“coastline approximation” - 1)

Input Code :

```
# importing libraries

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

import seaborn as sns

import sklearn.datasets

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import accuracy_score

# reading data from the file

df=pd.read_csv("data.csv")

df.head() #The head() method returns a specified numberof rows,
string from the top

df.info() #info() function is used to get a concise summary of the
dataframe

<class 'pandas.core.frame.DataFrame'>
```

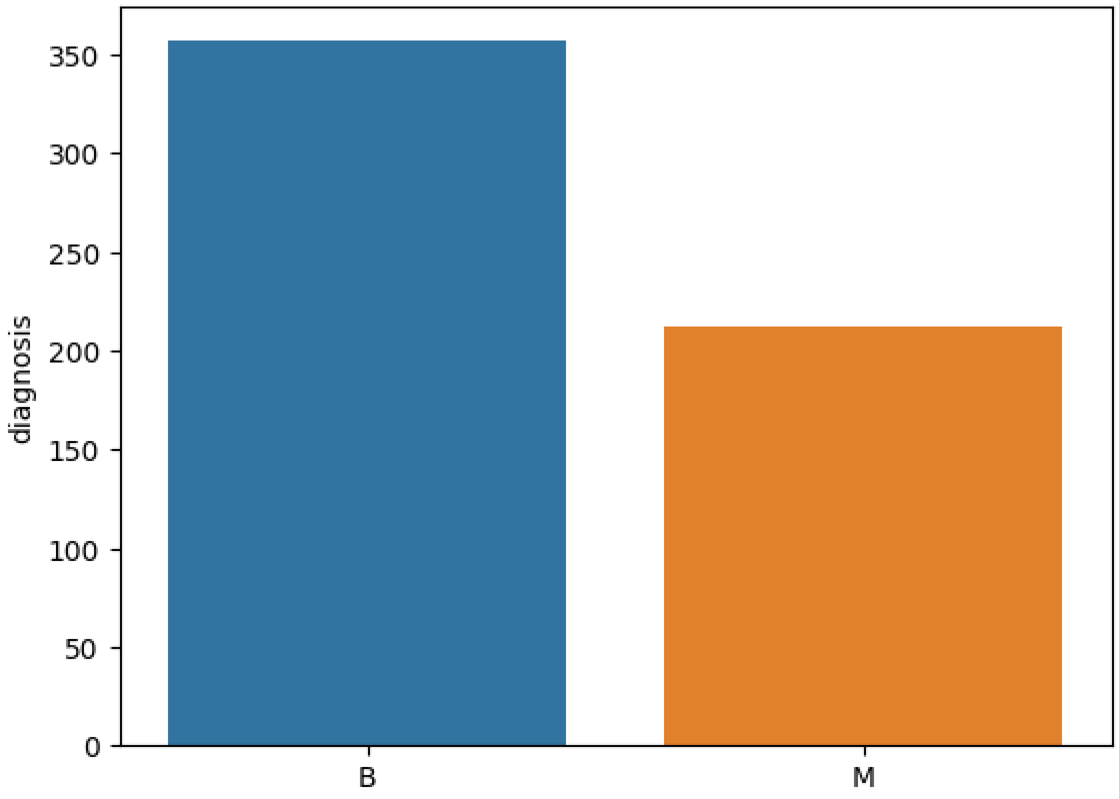
```
RangeIndex: 569 entries, 0 to 568

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
```

```
# return all the columns with null values count
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

# remove the column

df=df.dropna(axis=1)

# return the size of dataset after dropping null value

df.shape

(569, 32)

# statistical measures about the data

df.describe()

# checking the distribution of Target Varibale

df['diagnosis'].value_counts()


B    357

M    212

Name: diagnosis, dtype: int64

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


# Plot the counts using a bar plot

sns.barplot(x=value_counts.index, y=value_counts)


# Show the plot

plt.show()
```



```
df.groupby('diagnosis').mean()

  #easier to spot differences & patterns between groups

X = df.drop(columns='diagnosis', axis=1)

Y = df['diagnosis']

from sklearn.preprocessing import StandardScaler



scaler = StandardScaler()



scaler.fit(X)



standardized_data = scaler.transform(X)
```

```
print(standardized_data)
```

```
[[-0.23640517  1.09706398 -2.07333501 ...  2.29607613  2.75062224
   1.93701461]
 [-0.23640344  1.82982061 -0.35363241 ...  1.0870843  -0.24388967
   0.28118999]
 [ 0.43174109  1.57988811  0.45618695 ...  1.95500035  1.152255
   0.20139121]
 ...
 [-0.23572747  0.70228425  2.0455738  ...  0.41406869 -1.10454895
  -0.31840916]
 [-0.23572517  1.83834103  2.33645719 ...  2.28998549  1.91908301
   2.21963528]
 [-0.24240586 -1.80840125  1.22179204 ... -1.74506282 -0.04813821
  -0.75120669]]
```

```
print(X)
```

```
           id  radius_mean  texture_mean  perimeter_mean  area_mean  \
0      842302        17.99         10.38          122.80     1001.0
1      842517        20.57         17.77          132.90     1326.0
2    84300903        19.69         21.25          130.00     1203.0
3    84348301        11.42         20.38           77.58      386.1
4    84358402        20.29         14.34          135.10     1297.0
..        ...          ...           ...             ...        ...
564    926424        21.56         22.39          142.00     1479.0
565    926682        20.13         28.25          131.20     1261.0
```

```
566    926954        16.60         28.08          108.30      858.1

567    927241        20.60         29.33          140.10     1265.0

568     92751         7.76         24.54           47.92      181.0


     smoothness_mean  compactness_mean  concavity_mean  concave
points_mean  \

0            0.11840           0.27760         0.30010
0.14710

1            0.08474           0.07864         0.08690
0.07017

2            0.10960           0.15990         0.19740
0.12790

3            0.14250           0.28390         0.24140
0.10520

4            0.10030           0.13280         0.19800
0.10430

..               ...               ...             ...
...

564          0.11100           0.11590         0.24390
0.13890

565          0.09780           0.10340         0.14400
0.09791

566          0.08455           0.10230         0.09251
0.05302

567          0.11780           0.27700         0.35140
0.15200

568          0.05263           0.04362         0.00000
0.00000
```

```
     symmetry_mean  ...  radius_worst  texture_worst
perimeter_worst  \

0           0.2419  ...        25.380          17.33
184.60

1           0.1812  ...        24.990          23.41
158.80

2           0.2069  ...        23.570          25.53
152.50

3           0.2597  ...        14.910          26.50
98.87

4           0.1809  ...        22.540          16.67
152.20

..             ...  ...           ...            ...
...

564         0.1726  ...        25.450          26.40
166.10

565         0.1752  ...        23.690          38.25
155.00

566         0.1590  ...        18.980          34.12
126.70

567         0.2397  ...        25.740          39.42
184.60

568         0.1587  ...         9.456          30.37
59.16


     area_worst  smoothness_worst  compactness_worst
concavity_worst  \

0        2019.0           0.16220            0.66560
0.7119
```

```
1        1956.0           0.12380            0.18660
0.2416

2        1709.0           0.14440            0.42450
0.4504

3         567.7           0.20980            0.86630
0.6869

4        1575.0           0.13740            0.20500
0.4000

..          ...               ...                ...
...

564      2027.0           0.14100            0.21130
0.4107

565      1731.0           0.11660            0.19220
0.3215

566      1124.0           0.11390            0.30940
0.3403

567      1821.0           0.16500            0.86810
0.9387

568       268.6           0.08996            0.06444
0.0000


     concave points_worst  symmetry_worst  fractal_dimension_worst

0                  0.2654          0.4601                  0.11890

1                  0.1860          0.2750                  0.08902

2                  0.2430          0.3613                  0.08758

3                  0.2575          0.6638                  0.17300

4                  0.1625          0.2364                  0.07678

..                    ...             ...                      ...
```

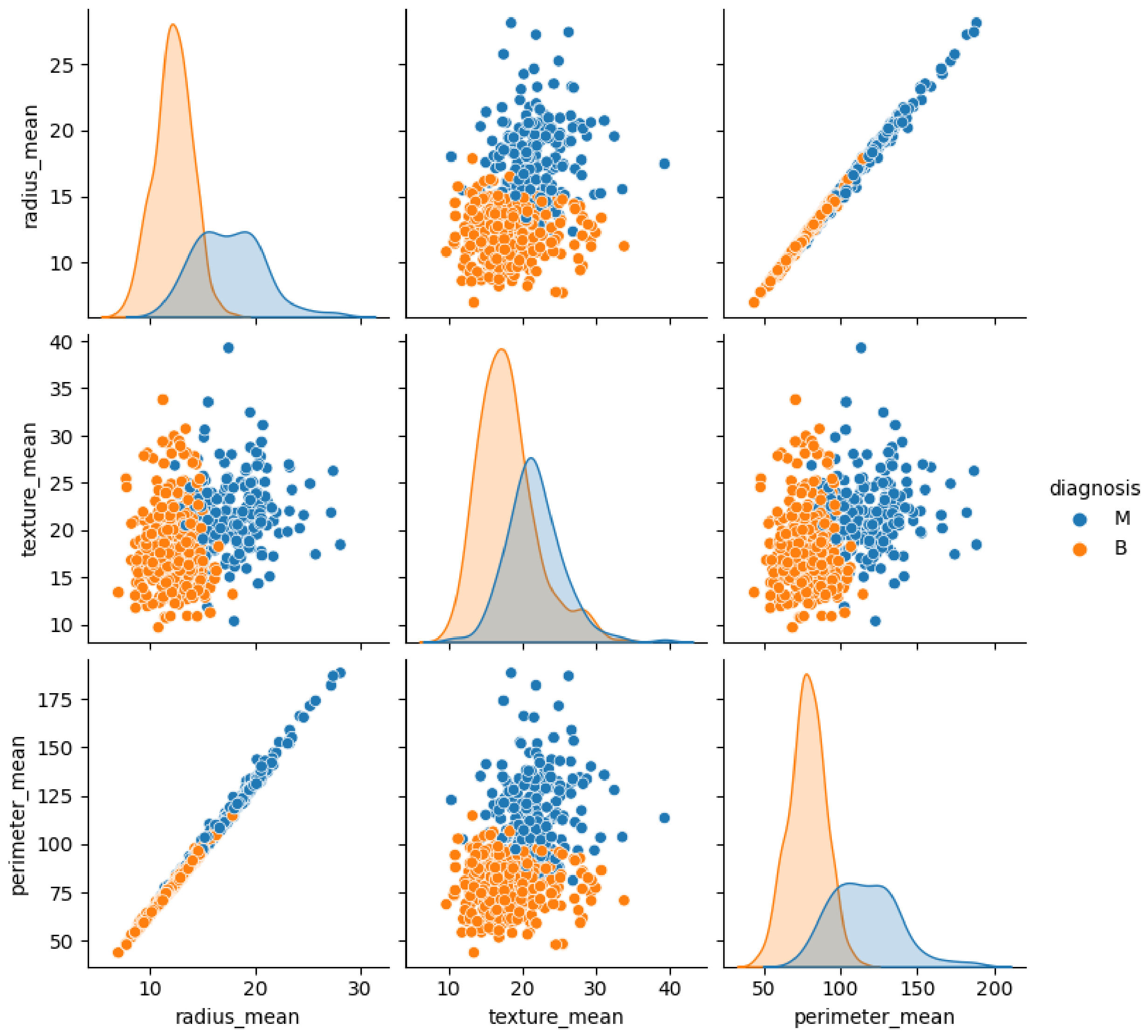
```
564                 0.2216          0.2060                  0.07115
565                 0.1628          0.2572                  0.06637
566                 0.1418          0.2218                  0.07820
567                 0.2650          0.4087                  0.12400
568                 0.0000          0.2871                  0.07039

[569 rows x 31 columns]
```

```
print(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
```

```
sns.pairplot(df.iloc[:,1:5],hue="diagnosis")
```

<seaborn.axisgrid.PairGrid at 0x7f80ff430040>

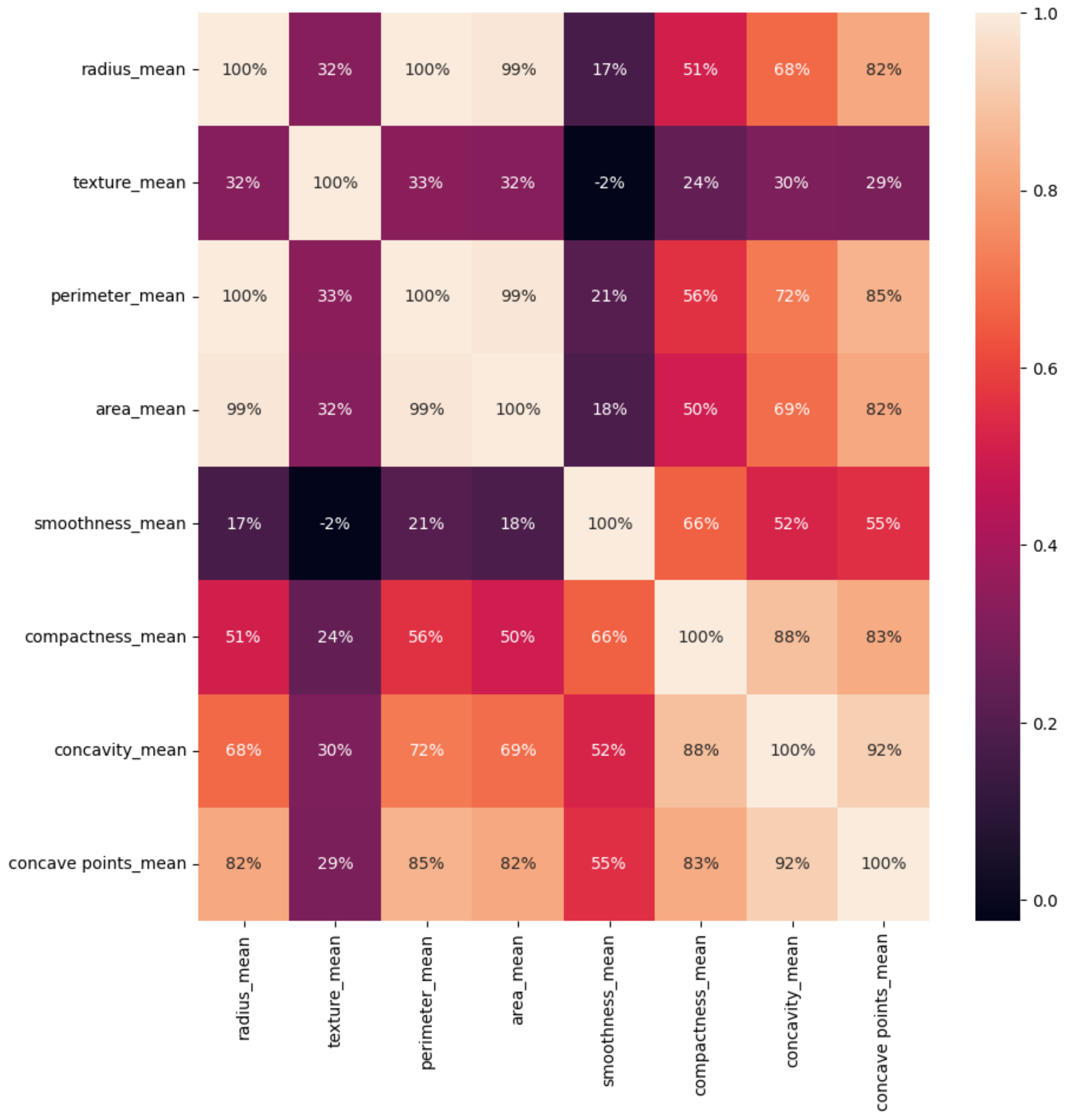


```
# get the correlation

df.iloc[:,1:32].corr()

# visualize the correlation

plt.figure(figsize=(10,10))

sns.heatmap(df.iloc[:,1:10].corr(),annot=True,fmt=".0%")
```

<ipython-input-18-a2668557c8b3>:3: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future version, it will default to False. Select only valid columns or specify the value of numeric_only to silence this warning.

```
sns.heatmap(df.iloc[:,1:10].corr(),annot=True,fmt=".0%")
```

```
<Axes: >
```



```
# spliting the data into trainning and test dateset

from sklearn.model_selection import train_test_split

X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.20,ra
ndom_state=0)

# feature scaling

from sklearn.preprocessing import StandardScaler

X_train=StandardScaler().fit_transform(X_train)

X_test=StandardScaler().fit_transform(X_test)

# LOGISTIC REGRESSION


from sklearn.linear_model import LogisticRegression

from sklearn.metrics import accuracy_score

from sklearn.metrics import classification_report


# create logistic regression object


model = LogisticRegression()


# train the model using the training sets


model.fit(X_train , Y_train)




X_train_prediction = model.predict(X_train)

training_data_accuracy = accuracy_score(Y_train, X_train_prediction)


print('Accuracy on training data = ', training_data_accuracy)
```

```
# accuracy on test data

X_test_prediction = model.predict(X_test)


# comparing actual response values (y_test) with predicted response
values (X_test_prediction)

test_data_accuracy = accuracy_score(Y_test, X_test_prediction)


print('Accuracy on test data = ', test_data_accuracy)

Accuracy on training data =  0.989010989010989

Accuracy on test data =  0.956140350877193

# models/ Algorithms


def models(X_train,Y_train):

        #Random Forest

        from sklearn.ensemble import RandomForestClassifier


forest=RandomForestClassifier(random_state=0,criterion="entropy",n_e
stimators=10)

        forest.fit(X_train,Y_train)




        #Decision Tree

        from sklearn.tree import DecisionTreeClassifier
```

```
tree=DecisionTreeClassifier(random_state=0,criterion="entropy")

        tree.fit(X_train,Y_train)




        print('[0]Random forest
accuracy:',forest.score(X_train,Y_train))

        print('[1]Decision tree
accuracy:',tree.score(X_train,Y_train))




        return forest,tree


model=models(X_train,Y_train)

[0]Random forest accuracy: 0.9978021978021978

[1]Decision tree accuracy: 1.0

# testing the models/result


from sklearn.metrics import accuracy_score

from sklearn.metrics import classification_report


for i in range(len(model)):

    print("Model",i)

    print(classification_report(Y_test,model[i].predict(X_test)))

    print('Accuracy :
```

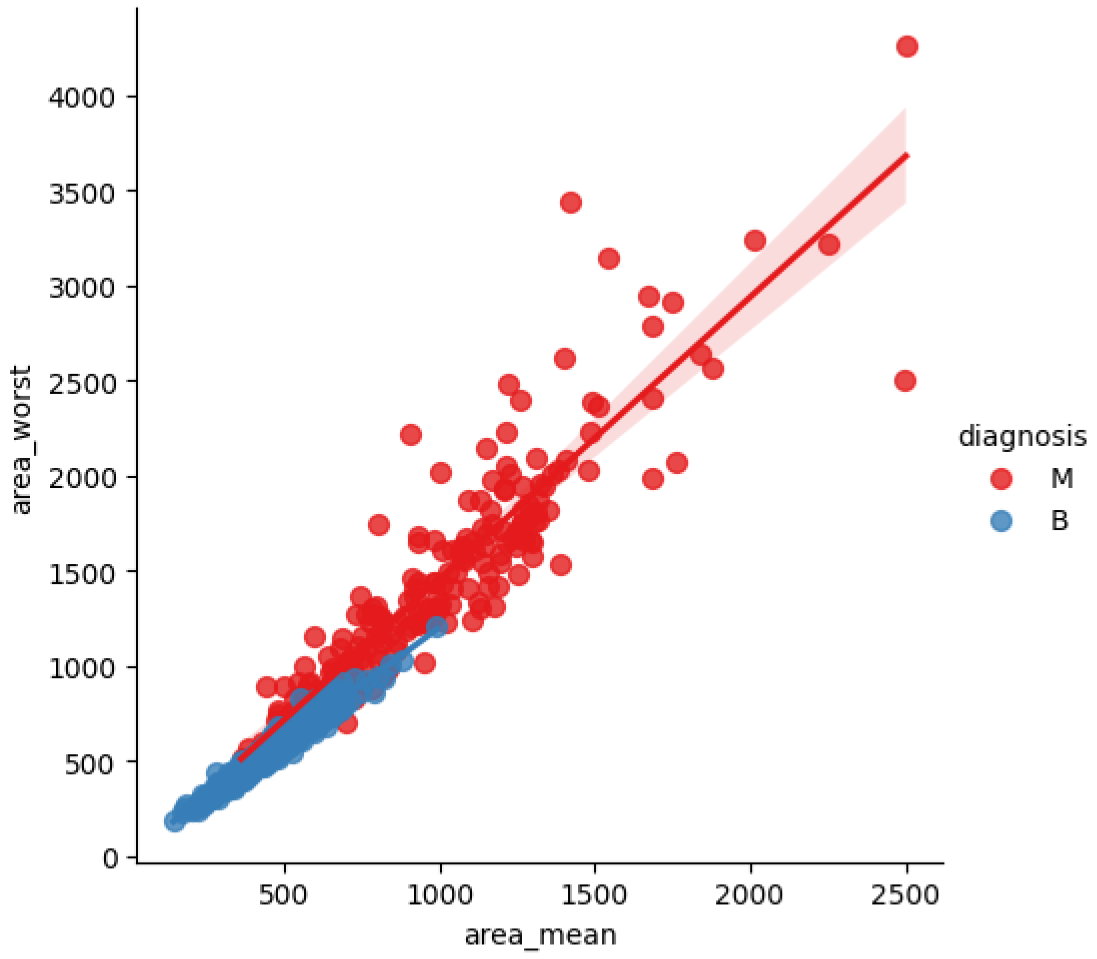
```
',accuracy_score(Y_test,model[i].predict(X_test)))

Model 0

              precision    recall  f1-score   support


           B       0.96      1.00      0.98        67

           M       1.00      0.94      0.97        47


    accuracy                           0.97       114

   macro avg       0.98      0.97      0.97       114

weighted avg       0.97      0.97      0.97       114


Accuracy :  0.9736842105263158

Model 1

              precision    recall  f1-score   support


           B       0.91      0.94      0.93        67

           M       0.91      0.87      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


Accuracy :  0.9122807017543859

from sklearn import svm
```

```
classifier = svm.SVC(kernel='linear')


#training the support vector Machine Classifier

classifier.fit(X_train, Y_train)


# accuracy score on the training data

X_train_prediction = classifier.predict(X_train)

training_data_accuracy = accuracy_score(X_train_prediction, Y_train)


print('Accuracy score of the training data : ',
training_data_accuracy)


# accuracy score on the test data

X_test_prediction = classifier.predict(X_test)

test_data_accuracy = accuracy_score(X_test_prediction, Y_test)


print('Accuracy score of the test data : ', test_data_accuracy)

Accuracy score of the training data :  0.989010989010989

Accuracy score of the test data :  0.9649122807017544

from sklearn.preprocessing import StandardScaler


#input_data =
(20.55,20.86,137.8,1308,0.1046,0.1739,0.2085,0.1322,0.2127,0.06251,0
.6986,0.9901,4.706,87.78,0.004578,0.02616,0.04005,0.01421,0.01948,0.
002689,24.3,25.48,160.2,1809,0.1268,0.3135,0.4433,0.2148,0.3077,0.45
,0.07569)


input_data = (13.54,14.36,87.46,566.3,0.09779,0.08129,0.06664,0.14
,0.04781,0.1885,0.05766,0.2699,0.7886,2.058,23.56,0.008462,0.0146,0.
02387,0.01315,0.0198,0.0023,15.11,19.26,99.7,711.2,0.144,0.1773,0.23
9,0.1288,0.2977,0.07259)


# changing the input_data to numpy array

input_data_as_numpy_array = np.asarray(input_data)


# reshape the array as we are predicting for one instance

input_data_reshaped = input_data_as_numpy_array.reshape(1,-1)


# standardize the input data

std_data = scaler.transform(input_data_reshaped)


prediction = classifier.predict(std_data)

print(prediction)


if (prediction[0] == 'M'):

  print('The Breast cancer is Malignant')#Cancerous cells


else:

  print('The Breast Cancer is Benign')#non-cancerous cells
```

```
sns.lmplot(x='area_mean',y='area_worst',data=dataset,hue='diagnosis'
,palette='Set1',fit_reg=True, scatter_kws={"s":50})
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but StandardScaler was fitted with feature names
  warnings.warn(
['M']
The Breast cancer is Malignant
```

```
<seaborn.axisgrid.FacetGrid at 0x7f80f7ac76a0>
```



CONCLUSION

Breast cancer can be detected in the early stages. It can be treated effectively. Breast cancer when it is most treatable, does not produce any symptoms, therefore be diligent with breast health. Combining multiple risk factors in modeling for breast cancer prediction could help the early diagnosis of the disease with necessary care plans. Collection, storage, and management of different data and intelligent systems based on multiple factors for predicting breast cancer are effective in disease management.

REFERENCES