```
## Breast Cancer Detection


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


 # reading data set and converting them to data frame df
df = pd.read_csv("data.csv")


# pringitn data set using head function and it will guve 5 row only , if want moew pass the integer inside head function
df.head()
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothn
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

```
df.info()   # provides summary of data set
```

```
<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
```

```
df.isna().sum() # returns all the column with null value cnt
```

```
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
```

```
df.shape  # returns the no of rows and column
```

```
(569, 32)
```

```
df = df.dropna(axis=1)
```

```
df.shape
```

```
(569, 32)
```

```
df.describe()
```

	id	radius_mean	texture_mean	perimeter_mean	area_mean	smoothne
count	5.690000e+02	569.000000	569.000000	569.000000	569.000000	569
mean	3.037183e+07	14.127292	19.289649	91.969033	654.889104	0
std	1.250206e+08	3.524049	4.301036	24.298981	351.914129	0
min	8.670000e+03	6.981000	9.710000	43.790000	143.500000	0
25%	8.692180e+05	11.700000	16.170000	75.170000	420.300000	0
50%	9.060240e+05	13.370000	18.840000	86.240000	551.100000	0
75%	8.813129e+06	15.780000	21.800000	104.100000	782.700000	0
max	9.113205e+08	28.110000	39.280000	188.500000	2501.000000	0

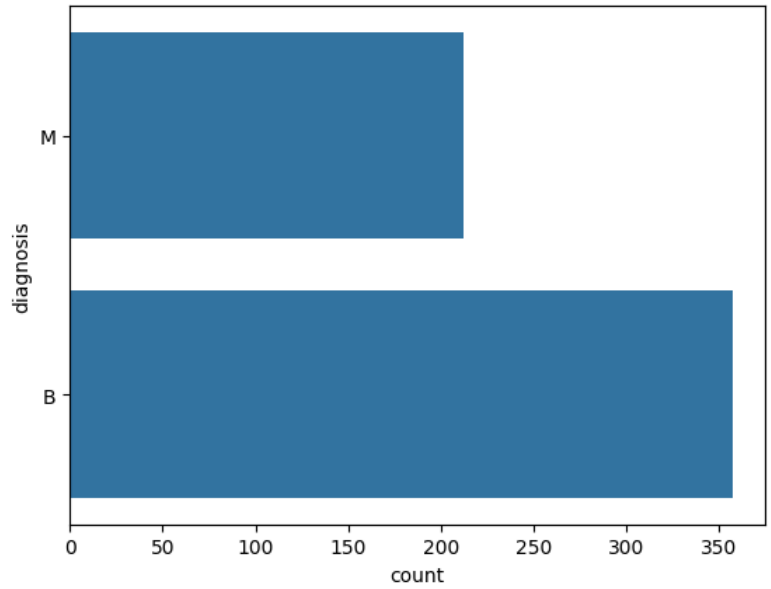
8 rows × 31 columns

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

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

```
sns.countplot(df['diagnosis'] , label = "count")
```

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




```
# label encoding(convert the value of M and B into 1 and 0)
from sklearn.preprocessing import LabelEncoder
labelencoder_Y = LabelEncoder()
df.iloc[:,1]=labelencoder_Y.fit_transform(df.iloc[:,1].values)
```

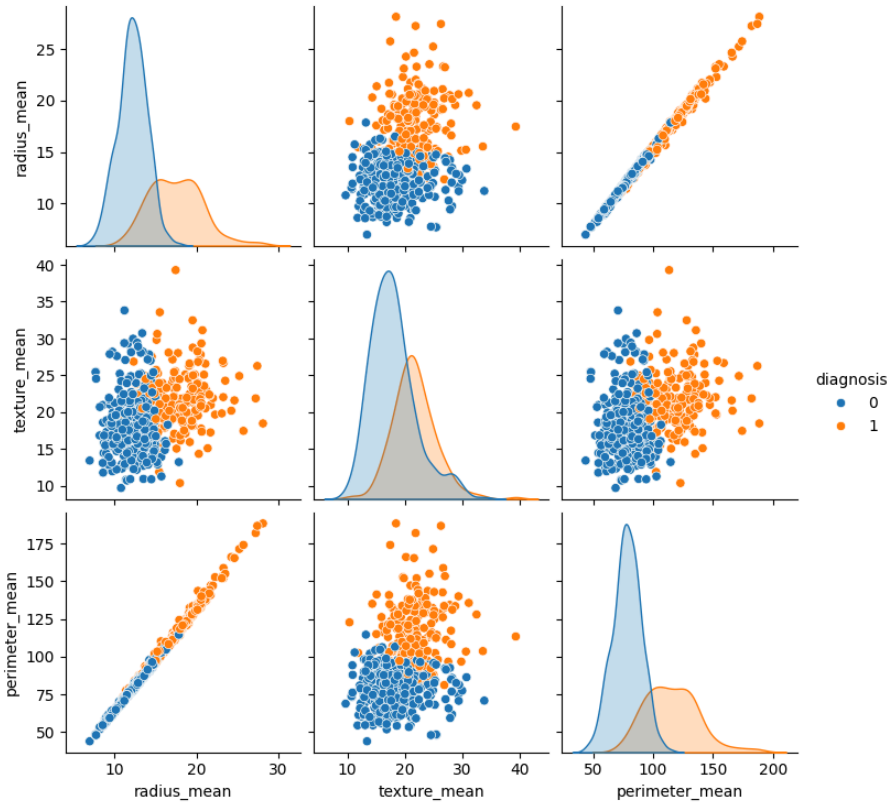
```
df.head()
```

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

5 rows × 32 columns

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

```
<seaborn.axisgrid.PairGrid at 0x7fbf7bc1beb0>
```




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

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_me
diagnosis	1.000000	0.730029	0.415185	0.742636	0.7089
radius_mean	0.730029	1.000000	0.323782	0.997855	0.9873
texture_mean	0.415185	0.323782	1.000000	0.329533	0.3210
perimeter_mean	0.742636	0.997855	0.329533	1.000000	0.9865
area_mean	0.708984	0.987357	0.321086	0.986507	1.0000
smoothness_mean	0.358560	0.170581	-0.023389	0.207278	0.1770
compactness_mean	0.596534	0.506124	0.236702	0.556936	0.4985
concavity_mean	0.696360	0.676764	0.302418	0.716136	0.6859
concave points_mean	0.776614	0.822529	0.293464	0.850977	0.8232
symmetry_mean	0.330499	0.147741	0.071401	0.183027	0.1512
fractal_dimension_mean	-0.012838	-0.311631	-0.076437	-0.261477	-0.2831
radius_se	0.567134	0.679090	0.275869	0.691765	0.7325
texture_se	-0.008303	-0.097317	0.386358	-0.086761	-0.0662
perimeter_se	0.556141	0.674172	0.281673	0.693135	0.7266
area_se	0.548236	0.735864	0.259845	0.744983	0.8000
smoothness_se	-0.067016	-0.222600	0.006614	-0.202694	-0.1667
compactness_se	0.292999	0.206000	0.191975	0.250744	0.2125
concavity_se	0.253730	0.194204	0.143293	0.228082	0.2076
concave points_se	0.408042	0.376169	0.163851	0.407217	0.3723
symmetry_se	-0.006522	-0.104321	0.009127	-0.081629	-0.0724
fractal_dimension_se	0.077972	-0.042641	0.054458	-0.005523	-0.0198
radius_worst	0.776454	0.969539	0.352573	0.969476	0.9627
texture_worst	0.456903	0.297008	0.912045	0.303038	0.2874
perimeter_worst	0.782914	0.965137	0.358040	0.970387	0.9591
area_worst	0.733825	0.941082	0.343546	0.941550	0.9592
smoothness_worst	0.421465	0.119616	0.077503	0.150549	0.1235
compactness_worst	0.590998	0.413463	0.277830	0.455774	0.3904
concavity_worst	0.659610	0.526911	0.301025	0.563879	0.5126
concave points_worst	0.793566	0.744214	0.295316	0.771241	0.7220
symmetry_worst	0.416294	0.163953	0.105008	0.189115	0.1435
fractal_dimension_worst	0.323872	0.007066	0.119205	0.051019	0.0037

31 rows × 31 columns

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

sns.heatmap(df.iloc[:,1:5].corr() , annot=True ,fmt="0%")
```

```
<Axes: >
```


```
# split the data set in to x and y , x = all row and col except output
X = df.iloc[:,2:31].values
Y = df.iloc[:,1].values
```

```
print(X)
```

```
[[ 17.99    10.38   122.8    ...   0.7119   0.2654   0.4601]
 [ 20.57    17.77   132.9    ...   0.2416   0.186    0.275 ]
 [ 19.69    21.25   130.     ...   0.4504   0.243    0.3613]
 ...
 [ 16.6     28.08   108.3    ...   0.3403   0.1418   0.2218]
 [ 20.6     29.33   140.1    ...   0.9387   0.265    0.4087]
 [  7.76    24.54    47.92   ...   0.       0.       0.2871]]
```

```
#splitted the data set in to 80percent training and 20 percent testing rom sklearn.model_selection import 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=0.2 , random_state=0)
```

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

```
def models(X_train,Y_train):
        #logistic regression
        from sklearn.linear_model import LogisticRegression
        log=LogisticRegression(random_state=0)
        log.fit(X_train,Y_train)
                                                #LEAVE THIS

        #Decision Tree
        from sklearn.tree import DecisionTreeClassifier
```

```
        tree=DecisionTreeClassifier(random_state=0,criterion="entropy")
        tree.fit(X_train,Y_train)

        #Random Forest
        from sklearn.ensemble import RandomForestClassifier
        forest=RandomForestClassifier(random_state=0,criterion="entropy",n_estimators=10)
        forest.fit(X_train,Y_train)

        print('[0]logistic regression accuracy:',log.score(X_train,Y_train))
        print('[1]Decision tree accuracy:',tree.score(X_train,Y_train))
        print('[2]Random forest accuracy:',forest.score(X_train,Y_train))

        return log, tree,forest
```



```
# model= models(X_train, Y_train)
```

```
[0]Logistic Regression accuracy 0.9912087912087912
[1]Decision Tree accuracy 1.0
[2]Random Forest accuracy 0.9978021978021978
```



```
def eshi(X_train, Y_train):
    # Import necessary libraries
    from sklearn.linear_model import LogisticRegression
    from sklearn.tree import DecisionTreeClassifier
    from sklearn.ensemble import RandomForestClassifier
    from sklearn.preprocessing import LabelEncoder

    # Initialize models
    log = LogisticRegression(random_state=0)
    tree = DecisionTreeClassifier(random_state=0, criterion="entropy")
    forest = RandomForestClassifier(random_state=0, criterion="entropy", n_estimators=10)

    # Encode the labels in Y_train
    le = LabelEncoder()
    Y_train_encoded = le.fit_transform(Y_train)

    # Fit the models
    log.fit(X_train, Y_train_encoded)
    tree.fit(X_train, Y_train_encoded)
    forest.fit(X_train, Y_train_encoded)

    # Print training accuracy
    print('[0] Logistic Regression accuracy', log.score(X_train, Y_train_encoded))
    print('[1] Decision Tree accuracy', tree.score(X_train, Y_train_encoded))
    print('[2] Random Forest accuracy', forest.score(X_train, Y_train_encoded))

    return log, tree, forest, le

# Train the models
log, tree, forest, le = eshi(X_train, Y_train)

# Encode the Y_test labels
Y_test_encoded = le.transform(Y_test)

# Import necessary metrics
from sklearn.metrics import accuracy_score, classification_report
```