Import Libraries

In [2]:

```
# import libraries
import pandas as pd
import seaborn as sns
```

Download, Load & Explore Dataset from Kaggle

In [15]:

```
# dataset name: Breast Cancer Wisconsin (Diagnostic) Data Set
# download dataset
# link: https://www.kaggle.com/datasets/uciml/breast-cancer-wisconsin-data/code?resource=download
# load data on dataframe
bcd = pd.read_csv('Breast Cancer Data.csv')
# display dataframe
bcd.head()
```

Out[15]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_me
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.277
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.078
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.159
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.283
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.132

5 rows × 33 columns

In [16]:

```
# (diagnosis): M=Malignant/Harmful, B=Benign?Not-Harmful
# the data is divided into 3 parts. the first part is mean, the second part is standard error (_se);
# and the third parameter is worst (_worst).
# each part contains three parameters each. in the end there is a unnamed column which contains null values a
# removed.
```

In [17]:

```
# count of rows and columns
bcd.shape
```

Out[17]:

```
(569, 33)
```

In [20]:

```
# count number of null(empty) values
# counting total number of null values for every column in our database.
# isna() used to detect the missing values in pandas.
# sum() used to detect the total number of null values present in the dataset.
bcd.isna().sum()
```

Out[20]:

```
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
```

In [21]:

```
# Drop the column with null values
# as we want to drop a column, so 1 and axis refers to rows. setting inplace as True will update the data fra
# now the data frame will be free from the column containing the null values.
bcd.dropna(axis=1, inplace=True)
```

In [23]:

```
# count of rows and columns
bcd.shape
```

Out[23]:

```
(569, 32)
```

In [24]:

```
# Get count of number of M or B cells in diagnosis
bcd['diagnosis'].value_counts()
```

Out[24]:

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

Label Encoding

In [26]:

```
# machine learning deals with datasets which contains multiple labels in one or more than one coloumns. these
# in the form of words or numbers. so to make the data more understandable and readable for human, the traini
# labelled into words.
# label encoding refers to converting these labels into numeric from  so as to convert it into the machine re
# it is an important pre-processing step for the structured dataset in supervised learning.
# Get Datatypes of each column in our dataset
bcd.dtypes
# id = integer type; we can get rid of this column.
# diagnosis = object type; it has string values. the values M/B needs to be transformed into numerical values
# Malignant = 1, Benign = 0
# others = float type.
```

Out[26]:

```
id                         int64
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
```

In [28]:

```
# Encode the diagnosis values
from sklearn.preprocessing import LabelEncoder
labelencoder = LabelEncoder()
labelencoder.fit_transform(bcd.iloc[:,1].values)
# diagnosis is present in column number 1, id is in number 0. the entire rows is provided, so ':' is applied.
```

Out[28]:

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

In [29]:

```
# now put the data back into the bcd dataframe.
bcd.iloc[:,1]=labelencoder.fit_transform(bcd.iloc[:,1].values)
```

In [30]:

```
# display dataframe
bcd
```

Out[30]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_r
0	842302	1	17.99	10.38	122.80	1001.0	0.11840	0.2
1	842517	1	20.57	17.77	132.90	1326.0	0.08474	0.0
2	84300903	1	19.69	21.25	130.00	1203.0	0.10960	0.1
3	84348301	1	11.42	20.38	77.58	386.1	0.14250	0.2
4	84358402	1	20.29	14.34	135.10	1297.0	0.10030	0.1
...	...	...	...	...	...	...	...	
564	926424	1	21.56	22.39	142.00	1479.0	0.11100	0.1
565	926682	1	20.13	28.25	131.20	1261.0	0.09780	0.1
566	926954	1	16.60	28.08	108.30	858.1	0.08455	0.1
567	927241	1	20.60	29.33	140.10	1265.0	0.11780	0.2
568	92751	0	7.76	24.54	47.92	181.0	0.05263	0.0

569 rows × 32 columns

Split Dataset & Feature Scaling

In [32]:

```
# Splitting the dataset into independent and dependent datasets
# dependent = diagnosis column, independent = rest of the columns.
X=bcd.iloc[:,2:].values
Y=bcd.iloc[:,1].values
# Y will tell us if the patient has cancer or not. X is going to tell us the features or attributes.
```

In [34]:

```
# Splitting datasets into training(75%) and testing(25%)
from sklearn.model_selection import train_test_split
X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size=0.25)
```

In [36]:

```
# Scaling the data(feature scaling)
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.fit_transform(X_test)
```

In [37]:

```
#print data
X_train
```

Out[37]:

```
array([[ 0.6014406 ,  0.34436702,  0.60633527, ...,  0.65530518,
         0.30411601, -0.12085337],
       [-0.67694552, -1.08313129, -0.6447511 , ..., -0.15359715,
        -0.52236259,  0.80581825],
       [ 0.44058407,  0.24951663,  0.37321359, ...,  0.59083858,
         0.8573101 ,  0.11702362],
       ...,
       [ 0.25432914, -0.59702306,  0.27219419, ..., -0.21192407,
        -1.22958977,  0.07176628],
       [-0.3072577 , -1.28468836, -0.38300044, ..., -1.23233825,
        -1.34552865, -1.01496178],
       [-0.7390305 , -0.12039987, -0.76621977, ..., -0.80501679,
         0.64033876, -0.76328682]])
```

Build a Logistic Regression Model

In [40]:

```
# build a logistic regression classifier
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression()
classifier.fit(X_train,Y_train)
```

Out[40]:

```
LogisticRegression()
```

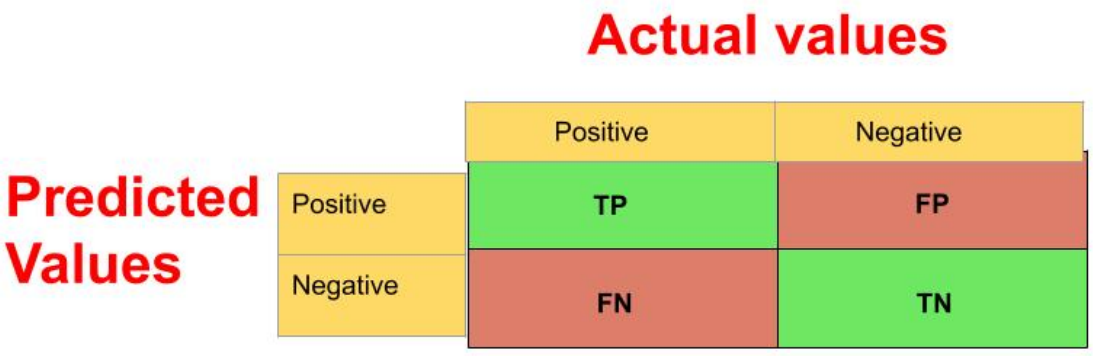
In [41]:

```
# make use of trained model to make predictions on test data
predictions = classifier.predict(X_test)
```

Performance Evaluation

In [42]:

```
# TP = True Positive, TN = True Negative, FP = False Positive, FN = False Negative
```



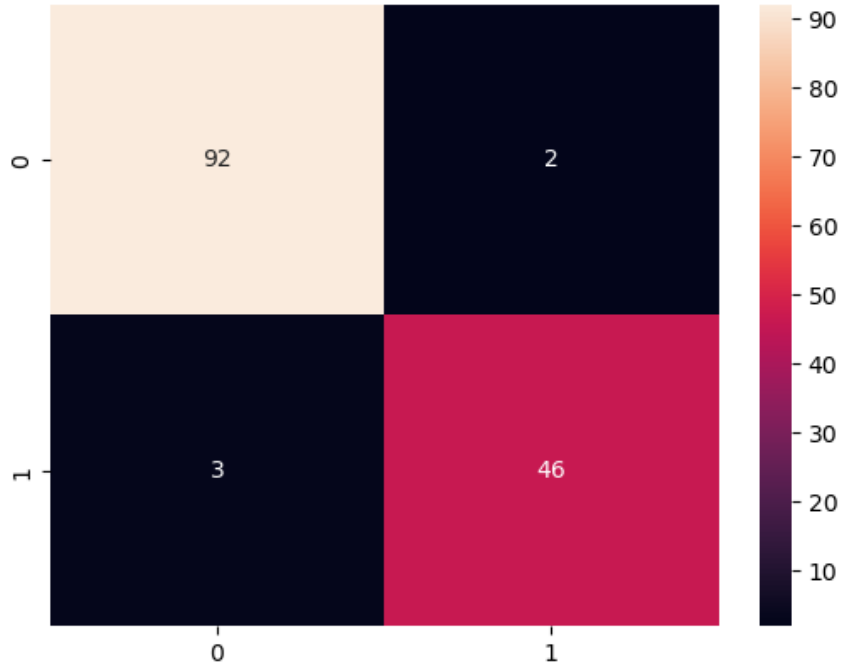
In [50]:

```
# plot confusion matrix
from sklearn.metrics import confusion_matrix
import seaborn as sns
cm = confusion_matrix(Y_test,predictions)
print(cm)
sns.heatmap(cm,annot=True)
```

```
[[92  2]
 [ 3 46]]
```

Out[50]:

```
<AxesSubplot:>
```



In [52]:

```
# the model has done a good job as the TP and TN values are good. TP = 92, TN = 46
# On the axis, 0 indicates benign and 1 represents malignant.
```

In [53]:

```
# get accuracy score for model
from sklearn.metrics import accuracy_score
print(accuracy_score(Y_test,predictions))
```

```
0.965034965034965
```

In [56]:

```
# model is 96% accurate on the testing data.
# (additional):compare the actual values present in the data frame with the predicted values by the model.
print(Y_test)
# Y_test is the actual values
```

```
[0 0 1 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 1 0 0 1 0 1 0
 1 1 1 0 1 0 0 1 0 1 0 0 0 0 0 0 0 1 1 0 0 0 0 0 1 0 1 0 1 1 1 0 0 1 1 0
 0 0 1 0 1 0 0 1 1 0 0 1 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 1 0 0 0 0 1 1
 1 0 1 1 0 0 0 0 0 0 0 0 1 0 0 1 0 1 1 1 1 1 1 0 0 1 0 1 0 0 1 1]
```

In [55]:

```
print(predictions)
```

```
[0 0 1 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 1 0 0 1 0 1 0
 1 0 0 1 1 0 0 1 0 1 0 0 0 0 0 0 0 1 1 0 0 0 0 0 1 0 1 0 1 1 1 0 0 1 1 1
 0 0 1 0 1 0 0 1 1 0 0 1 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 1 1
 1 0 1 1 0 0 0 0 0 0 0 0 1 0 0 1 0 1 1 1 1 1 1 0 0 1 0 1 0 0 1 1]
```

In [59]:

```
# it is time consuming to go through each and every values and compare them, so this is why the model accurac
# in [53]
```