In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
from sklearn.datasets import load_breast_cancer
cancer_dataset = load_breast_cancer()
#print(cancer_dataset)
print(cancer_dataset.keys())
type(cancer_dataset)
```

```
dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names', 'filename'])
```

Out[2]:
```
sklearn.utils.Bunch
```

In [3]:
```
#Load some features Values
print(cancer_dataset['feature_names'])
```

```
['mean radius' 'mean texture' 'mean perimeter' 'mean area'
 'mean smoothness' 'mean compactness' 'mean concavity'
 'mean concave points' 'mean symmetry' 'mean fractal dimension'
 'radius error' 'texture error' 'perimeter error' 'area error'
 'smoothness error' 'compactness error' 'concavity error'
 'concave points error' 'symmetry error' 'fractal dimension error'
 'worst radius' 'worst texture' 'worst perimeter' 'worst area'
 'worst smoothness' 'worst compactness' 'worst concavity'
 'worst concave points' 'worst symmetry' 'worst fractal dimension']
```

In [4]:
```
#Create Dataframe
can_df=pd.DataFrame(np.c_[cancer_dataset['data'],cancer_dataset['target']],
                    columns=np.append(cancer_dataset['feature_names'],['target']))
can_df
#appends target as 0's and 1's(1-no cancer, 0-cancer)
```

Out[4]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst texture	worst perimeter	worst area	worst smoothness	worst compactness	worst concavity	w conc po
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	0.2419	0.07871	...	17.33	184.60	2019.0	0.16220	0.66560	0.7119	0.2
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	0.1812	0.05667	...	23.41	158.80	1956.0	0.12380	0.18660	0.2416	0.1
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	0.2069	0.05999	...	25.53	152.50	1709.0	0.14440	0.42450	0.4504	0.2
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	0.2597	0.09744	...	26.50	98.87	567.7	0.20980	0.86630	0.6869	0.2
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	0.1809	0.05883	...	16.67	152.20	1575.0	0.13740	0.20500	0.4000	0.1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
564	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	0.1726	0.05623	...	26.40	166.10	2027.0	0.14100	0.21130	0.4107	0.2
565	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	0.1752	0.05533	...	38.25	155.00	1731.0	0.11660	0.19220	0.3215	0.1
566	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	0.1590	0.05648	...	34.12	126.70	1124.0	0.11390	0.30940	0.3403	0.1
567	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	0.2397	0.07016	...	39.42	184.60	1821.0	0.16500	0.86810	0.9387	0.2
568	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	0.1587	0.05884	...	30.37	59.16	268.6	0.08996	0.06444	0.0000	0.0

569 rows × 31 columns

In [5]:
```
#Cerete this data into CSV form for future use
can_df.to_csv("breast_cancer_dataframe.csv")
can_df.head()
```

Out[5]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst texture	worst perimeter	worst area	worst smoothness	worst compactness	worst concavity	wors concav point
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...	17.33	184.60	2019.0	0.1622	0.6656	0.7119	0.265
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	...	23.41	158.80	1956.0	0.1238	0.1866	0.2416	0.186
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	...	25.53	152.50	1709.0	0.1444	0.4245	0.4504	0.243
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	...	26.50	98.87	567.7	0.2098	0.8663	0.6869	0.257
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	...	16.67	152.20	1575.0	0.1374	0.2050	0.4000	0.162

5 rows × 31 columns

In [6]:
```
#Taking info of the dataset
can_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 31 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   mean radius              569 non-null    float64
 1   mean texture             569 non-null    float64
 2   mean perimeter           569 non-null    float64
 3   mean area                569 non-null    float64
 4   mean smoothness          569 non-null    float64
 5   mean compactness         569 non-null    float64
 6   mean concavity           569 non-null    float64
 7   mean concave points      569 non-null    float64
 8   mean symmetry            569 non-null    float64
 9   mean fractal dimension   569 non-null    float64
 10  radius error             569 non-null    float64
 11  texture error            569 non-null    float64
 12  perimeter error          569 non-null    float64
 13  area error               569 non-null    float64
 14  smoothness error         569 non-null    float64
 15  compactness error        569 non-null    float64
 16  concavity error          569 non-null    float64
 17  concave points error     569 non-null    float64
 18  symmetry error           569 non-null    float64
 19  fractal dimension error  569 non-null    float64
 20  worst radius             569 non-null    float64
 21  worst texture            569 non-null    float64
 22  worst perimeter          569 non-null    float64
 23  worst area               569 non-null    float64
 24  worst smoothness         569 non-null    float64
 25  worst compactness        569 non-null    float64
 26  worst concavity          569 non-null    float64
 27  worst concave points     569 non-null    float64
 28  worst symmetry           569 non-null    float64
 29  worst fractal dimension  569 non-null    float64
 30  target                   569 non-null    float64
dtypes: float64(31)
memory usage: 137.9 KB
```

In [7]:
```
#To check the null values
can_df.isnull().sum()
```

Out[7]:
```
mean radius                0
mean texture               0
mean perimeter             0
mean area                  0
mean smoothness            0
mean compactness           0
mean concavity             0
mean concave points        0
mean symmetry              0
mean fractal dimension     0
radius error               0
texture error              0
perimeter error            0
area error                 0
smoothness error           0
compactness error          0
concavity error            0
concave points error       0
symmetry error             0
fractal dimension error    0
worst radius               0
worst texture              0
worst perimeter            0
worst area                 0
worst smoothness           0
worst compactness          0
worst concavity            0
worst concave points       0
worst symmetry             0
worst fractal dimension    0
target                     0
dtype: int64
```

In [8]:
```
#So there is no such null Values
```

In [9]:
```
#To describe
can_df.describe()
```

Out[9]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst texture	worst perimeter	worst area	wors smoothnes
count	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	...	569.000000	569.000000	569.000000	569.00000
mean	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	0.088799	0.048919	0.181162	0.062798	...	25.677223	107.261213	880.583128	0.13236
std	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	0.079720	0.038803	0.027414	0.007060	...	6.146258	33.602542	569.356993	0.02283
min	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	0.000000	0.000000	0.106000	0.049960	...	12.020000	50.410000	185.200000	0.07117
25%	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	0.029560	0.020310	0.161900	0.057700	...	21.080000	84.110000	515.300000	0.11660
50%	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630	0.061540	0.033500	0.179200	0.061540	...	25.410000	97.660000	686.500000	0.13130
75%	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400	0.130700	0.074000	0.195700	0.066120	...	29.720000	125.400000	1084.000000	0.14600
max	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	0.426800	0.201200	0.304000	0.097440	...	49.540000	251.200000	4254.000000	0.22260

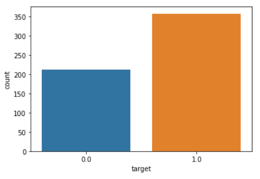
8 rows × 31 columns

In [10]:
```
#To plt pairplot
#sns.pairplot(can_df, hue = 'target')
#"Blue poits show that person is having Cancer and Orange colour Shows he is not affected"
```

In [11]:
```
#Using Count Plot
sns.countplot(can_df['target'])
#Again blue shows person is affected and orange defines not affected,
#This will take more time to plot.
```

```
C:\Users\win7\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version
0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinte
rpretation.
  warnings.warn(
```

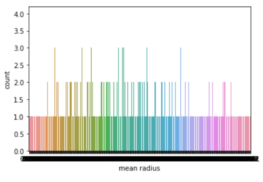
Out[11]:
```
<AxesSubplot:xlabel='target', ylabel='count'>
```



In [12]:
```
sns.countplot(can_df['mean radius'])
# This shows Mean radius whose values is >1 are affected and <1 are not affected
```

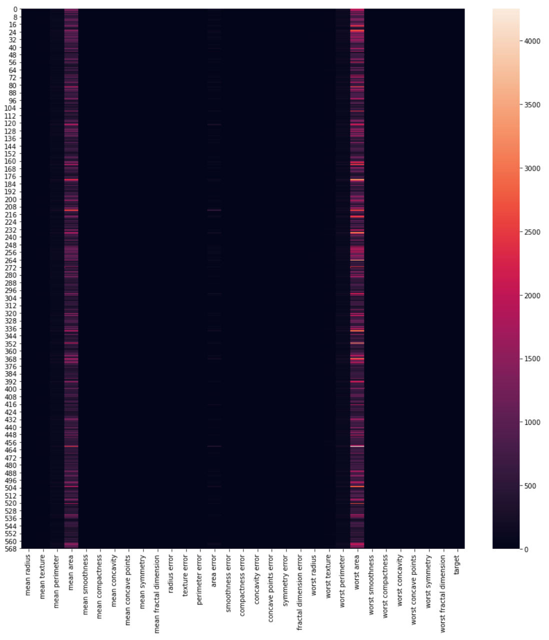
```
C:\Users\win7\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version
0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinte
rpretation.
  warnings.warn(
```

Out[12]:
```
<AxesSubplot:xlabel='mean radius', ylabel='count'>
```



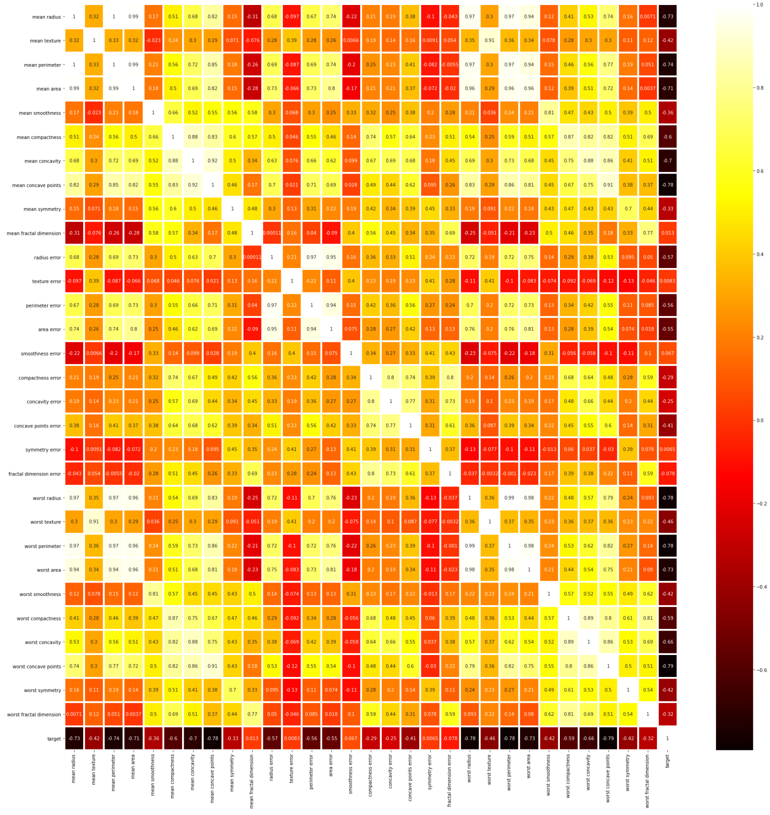
In [13]:
```
plt.figure(figsize=(15,15))
sns.heatmap(can_df)
```

Out[13]:
```
<AxesSubplot:>
```



In [14]:
```
#To see the co relation
plt.figure(figsize=(30,30))
sns.heatmap(can_df.corr(), annot= True, cmap='hot',linewidths=3)
```

Out[14]:
```
<AxesSubplot:>
```



Splitting the data into train and test datasets

In [15]:

```
# 'X' which is independent variable, and y is dependent variable
X = can_df.drop(["target"], axis=1) #This drops the target values
X
y = can_df['target']
```

In [16]:

```
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,random_state=0)
print(X_train)
print(X_test)
print(y_test)
print(y_train)
```

```
     mean radius  mean texture  mean perimeter  mean area  mean smoothness  \
338       10.050         17.53           64.41      310.8          0.10070   
427       10.800         21.98           68.79      359.9          0.08801   
406       16.140         14.86          104.30      800.0          0.09495   
96        12.180         17.84           77.79      451.1          0.10450   
490       12.250         22.44           78.18      466.5          0.08192   
..           ...           ...             ...        ...              ...   
277       18.810         19.98          120.90     1102.0          0.08923   
9         12.460         24.04           83.97      475.9          0.11860   
359        9.436         18.32           59.82      278.6          0.10090   
192        9.720         18.22           60.73      288.1          0.06950   
559       11.510         23.93           74.52      403.5          0.09261   

     mean compactness  mean concavity  mean concave points  mean symmetry  \
338           0.07326         0.02511              0.01775         0.1890   
427           0.05743         0.03614              0.01404         0.2016   
406           0.08501         0.05500              0.04528         0.1735   
96            0.07057         0.02490              0.02941         0.1900   
490           0.05200         0.01714              0.01261         0.1544   
..                ...             ...                  ...            ...   
277           0.05884         0.08020              0.05843         0.1550   
9             0.23960         0.22730              0.08543         0.2030   
359           0.05956         0.02710              0.01406         0.1506   
192           0.02344         0.00000              0.00000         0.1653   
559           0.10210         0.11120              0.04105         0.1388   

     mean fractal dimension  ...  worst radius  worst texture  \
338                 0.06331  ...        11.160          26.84   
427                 0.05977  ...        12.760          32.04   
406                 0.05875  ...        17.710          19.58   
96                  0.06635  ...        12.830          20.92   
490                 0.05976  ...        14.170          31.99   
..                      ...  ...           ...            ...   
277                 0.04996  ...        19.960          24.30   
9                   0.08243  ...        15.090          40.68   
359                 0.06959  ...        12.020          25.02   
192                 0.06447  ...         9.968          20.83   
559                 0.06570  ...        12.480          37.16   

     worst perimeter  worst area  worst smoothness  worst compactness  \
338            71.98       384.0           0.14020            0.14020   
427            83.69       489.5           0.13030            0.16960   
406           115.90       947.9           0.12060            0.17220   
96             82.14       495.2           0.11400            0.09358   
490            92.74       622.9           0.12560            0.18040   
..               ...         ...               ...                ...   
277           129.00      1236.0           0.12430            0.11600   
9              97.65       711.4           0.18530            1.05800   
359            75.79       439.6           0.13330            0.10490   
192            62.25       303.8           0.07117            0.02729   
559            82.28       474.2           0.12980            0.25170   

     worst concavity  worst concave points  worst symmetry  \
338           0.1055               0.06499          0.2894   
427           0.1927               0.07485          0.2965   
406           0.2310               0.11290          0.2778   
96            0.0498               0.05882          0.2227   
490           0.1230               0.06335          0.3100   
..               ...                   ...             ...   
277           0.2210               0.12940          0.2567   
9             1.1050               0.22100          0.4366   
359           0.1144               0.05052          0.2454   
192           0.0000               0.00000          0.1909   
559           0.3630               0.09653          0.2112   

     worst fractal dimension  
338                  0.07664  
427                  0.07662  
406                  0.07012  
96                   0.07376  
490                  0.08203  
..                       ...  
277                  0.05737  
9                    0.20750  
359                  0.08136  
192                  0.06559  
559                  0.08732  

[455 rows x 30 columns]
     mean radius  mean texture  mean perimeter  mean area  mean smoothness  \
512        13.40         20.52           88.64      556.7          0.11060   
457        13.21         25.25           84.10      537.9          0.08791   
439        14.02         15.66           89.59      606.5          0.07966   
298        14.26         18.17           91.22      633.1          0.06576   
37         13.03         18.42           82.61      523.8          0.08983   
..           ...           ...             ...        ...              ...   
213        17.42         25.56          114.50      948.0          0.10060   
519        12.75         16.70           82.51      493.8          0.11250   
432        20.18         19.54          133.80     1250.0          0.11330   
516        18.31         20.58          120.80     1052.0          0.10680   
500        15.04         16.74           98.73      689.4          0.09883   

     mean compactness  mean concavity  mean concave points  mean symmetry  \
512           0.14690         0.14450              0.08172         0.2116   
457           0.05205         0.02772              0.02068         0.1619   
439           0.05581         0.02087              0.02652         0.1589   
298           0.05220         0.02475              0.01374         0.1635   
37            0.03766         0.02562              0.02923         0.1467   
..                ...             ...                  ...            ...   
213           0.11460         0.16820              0.06597         0.1308   
519           0.11170         0.03880              0.02995         0.2120   
432           0.14890         0.21330              0.12590         0.1724   
516           0.12480         0.15690              0.09451         0.1860   
500           0.13640         0.07721              0.06142         0.1668   

     mean fractal dimension  ...  worst radius  worst texture  \
512                 0.07325  ...         16.41          29.66   
457                 0.05584  ...         14.35          34.23   
439                 0.05586  ...         14.91          19.31   
298                 0.05586  ...         16.22          25.26   
37                  0.05863  ...         13.30          22.81   
..                      ...  ...           ...            ...   
213                 0.05866  ...         18.07          28.07   
519                 0.06623  ...         14.45          21.74   
432                 0.06053  ...         22.03          25.07   
516                 0.05941  ...         21.86          26.20   
500                 0.06869  ...         16.76          20.43   

     worst perimeter  worst area  worst smoothness  worst compactness  \
512           113.30       844.4           0.15740            0.38560   
457            91.29       632.9           0.12890            0.10630   
439            96.53       688.9           0.10340            0.10170   
298           105.80       819.7           0.09445            0.21670   
37             84.46       545.9           0.09701            0.04619   
..               ...         ...               ...                ...   
213           120.40      1021.0           0.12430            0.17930   
519            93.63       624.1           0.14750            0.19790   
432           146.00      1479.0           0.16650            0.29420   
516           142.20      1493.0           0.14920            0.25360   
500           109.70       856.9           0.11350            0.21760   

     worst concavity  worst concave points  worst symmetry  \
512          0.51060               0.20510          0.3585   
457          0.13900               0.06005          0.2444   
439          0.06260               0.08216          0.2136   
298          0.15650               0.07530          0.2636   
37           0.04833               0.05013          0.1987   
..               ...                   ...             ...   
213          0.28030               0.10990          0.1603   
519          0.14230               0.08045          0.3071   
432          0.53080               0.21730          0.3032   
516          0.37590               0.15100          0.3074   
500          0.18560               0.10180          0.2177   

     worst fractal dimension  
512                  0.11090  
457                  0.06788  
439                  0.06710  
298                  0.07676  
37                   0.06169  
..                       ...  
213                  0.06818  
519                  0.08557  
432                  0.08075  
516                  0.07863  
500                  0.08549  

[114 rows x 30 columns]
512    0.0
457    1.0
439    1.0
298    1.0
37     1.0
      ... 
213    0.0
519    1.0
432    0.0
516    0.0
500    1.0
Name: target, Length: 114, dtype: float64
338    1.0
427    1.0
406    1.0
96     1.0
490    1.0
      ... 
277    0.0
9      0.0
359    1.0
192    1.0
559    1.0
Name: target, Length: 455, dtype: float64
```

Feature Scalling

Feature scalling is a technique to standardize the independent features present in the data in the fixed range. its performed during the data preprocessing to handel highly varying magnnitudes or values or units.

In [17]:

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.fit_transform(X_test)
X_train
#So this create the value ranges from -2 to +2 with the help of standard scaller
```

Out[17]:

```
array([[-1.15036482, -0.39064196, -1.12855021, ..., -0.75798367,
        -0.01614761, -0.38503402],
       [-0.93798972,  0.68051405, -0.94820146, ..., -0.60687023,
         0.09669004, -0.38615797],
       [ 0.574121  , -1.03333557,  0.51394098, ..., -0.02371948,
        -0.20050207, -0.75144254],
       ...,
       [-1.32422924, -0.20048168, -1.31754581, ..., -0.97974953,
        -0.71542314, -0.11978123],
       [-1.24380987, -0.2245526 , -1.28007609, ..., -1.75401433,
        -1.58157125, -1.00601779],
       [-0.73694129,  1.14989702, -0.71226578, ..., -0.27460457,
        -1.25895095,  0.21515662]])
```

Support Vector Classifier

In [18]:

```
#Train Model Basis of support vector classifer
from sklearn.svm import SVC
classifier = SVC()
classifier.fit(X_train, y_train)
```

Out[18]:

```
SVC()
```

In [19]:

```
y_pred = classifier.predict(X_test)
y_pred
```

Out[19]:

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

In [20]:

```
#Now see the accuracy using Confusion Matrix
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report
cm = confusion_matrix(y_pred, y_test)
print(cm)
accuracy_score(y_pred, y_test)
#classification_report(y_pred, y_test) #This shows the Entire result and accuracy
```

```
[[45  0]
 [ 2 67]]
```

Out[20]:

```
0.9824561403508771
```

So SVM gives 98% of accuracy

Logistic regression

In [21]:

```
from sklearn.linear_model import LogisticRegression
lg_classifier = LogisticRegression(random_state = 0)
lg_classifier.fit(X_train, y_train)
```

Out[21]:

```
LogisticRegression(random_state=0)
```

In [22]:

```
y_pred = lg_classifier.predict(X_test)
y_pred
```

Out[22]:

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

In [23]:

```
#Now see the accuracy using Confusion Matrix
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report
cm = confusion_matrix(y_pred, y_test)
print(cm)
accuracy_score(y_pred, y_test)
```

```
[[44  2]
 [ 3 65]]
```

Out[23]:

```
0.956140350877193
```

Logistic Regression Gives 95% accuracy

KNN Classifier

In [24]:

```
from sklearn.neighbors import KNeighborsClassifier
KN_classifier = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p=2)
KN_classifier.fit(X_train, y_train)
```

Out[24]:

```
KNeighborsClassifier()
```

In [25]:

```
y_pred = KN_classifier.predict(X_test)
y_pred
```

Out[25]:

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

In [26]:

```
#Now see the accuracy using Confusion Matrix
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report
cm = confusion_matrix(y_pred, y_test)
print(cm)
accuracy_score(y_pred, y_test)
#classification_report(y_pred, y_test) #This shows the Entire result and accuracy
```

```
[[43  0]
 [ 4 67]]
```

Out[26]:

```
0.9649122807017544
```

In [27]:

```
#So KNN gives 96% accuracy
```

Naive bayes classifier

In [28]:

```
from sklearn.naive_bayes import GaussianNB
NB_classifier = GaussianNB()
NB_classifier.fit(X_train, y_train)
```

Out[28]:

```
GaussianNB()
```

In [29]:

```
y_pred = NB_classifier.predict(X_test)
y_pred
```

Out[29]:

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

In [30]:

```
#Now see the accuracy using Confusion Matrix
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report
cm = confusion_matrix(y_pred, y_test)
print(cm)
accuracy_score(y_pred, y_test)
```

```
[[43  3]
 [ 4 64]]
```

Out[30]:

```
0.9385964912280702
```

In [31]:

```
#Naive bayes gives 93% accuracy
```

Decision Tree Classifier

In [32]:

```
from sklearn.tree import DecisionTreeClassifier
D_classifier = DecisionTreeClassifier(criterion = 'entropy', random_state=0)
D_classifier.fit(X_train, y_train)
```

Out[32]:

```
DecisionTreeClassifier(criterion='entropy', random_state=0)
```

In [33]:

```
y_pred = D_classifier.predict(X_test)
y_pred
```

Out[33]:

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

In [34]:

```
#Now see the accuracy using Confusion Matrix
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report
cm = confusion_matrix(y_pred, y_test)
print(cm)
accuracy_score(y_pred, y_test)
```

```
[[43  3]
 [ 4 64]]
```

Out[34]:

```
0.9385964912280702
```

In [35]:

```
#Decission Tree gives 93% accuracy
```

Random Forest Classifier

In [36]:

```
from sklearn.ensemble import RandomForestClassifier
rc_classifier = RandomForestClassifier(n_estimators = 20, criterion = 'entropy', random_state = 0)
rc_classifier.fit(X_train, y_train)
```

Out[36]:

```
RandomForestClassifier(criterion='entropy', n_estimators=20, random_state=0)
```

In [37]:

```
y_pred = rc_classifier.predict(X_test)
y_pred
```

Out[37]:

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

In [38]:

```
#Now see the accuracy using Confusion Matrix
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report
cm = confusion_matrix(y_pred, y_test)
print(cm)
accuracy_score(y_pred, y_test)
```

```
[[45  0]
 [ 2 67]]
```

Out[38]:

```
0.9824561403508771
```

In [39]:

```
#Random forest gives 98% of accuracy
```

We have got same accuracy with Random forest and logistic regression so we use any of it

In [40]:

```
from sklearn.model_selection import cross_val_score
cross_validation = cross_val_score(estimator = rc_classifier, X = X_train, y=y_train)
cross_validation
print("cross validation mean accuracy",cross_validation.mean())
```

```
cross validation mean accuracy 0.956043956043956
```

Importing Pickle

This converts Python objects like Lists, Dictonaries, etc into byte stremes (Zeros and ones)

In [41]:

```
import pickle
pickle.dump(rc_classifier, open("breast_cancer.pickle","wb"))
breast_cancer_model = pickle.load(open("breast_cancer.pickle","rb"))
y_pred = breast_cancer_model.predict(X_test)
print(confusion_matrix(y_pred, y_test))
print(accuracy_score(y_pred, y_test))
```

```
[[45  0]
 [ 2 67]]
0.9824561403508771
```

In [42]:

```
#So we have 98% accuracy with Random forest
```

In [ ]: