Breast_Cancer_Detection_Using_Machine_Learnir

In this notebook, we have to detect the Brest Cancer using Machine Learning Algorithms. As a Machine Learning Engineer and Data Scientist, We have to create Machine Learning Model to classify malignant and benign tumor. To solve this problem, we used Supervised Machine Learning Algorithms

To solve This Problem, We have Used Scikit-learn Dataset

Designed and Developed by Mabtoor Mabx

1- Import Libraries

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

2- Load Dataset

For loading Dataset, We have to used scikit-learn datasets

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

3- Data Manipulation

In [3]:
```
cancer_dataset
```

Out[3]:
```
{'data': array([[1.799e+01, 1.038e+01, 1.228e+02, ..., 2.654e-01, 4.601e-01,
         1.189e-01],
        [2.057e+01, 1.777e+01, 1.329e+02, ..., 1.860e-01, 2.750e-01,
         8.902e-02],
        [1.969e+01, 2.125e+01, 1.300e+02, ..., 2.430e-01, 3.613e-01,
         8.758e-02],
        ...,
        [1.660e+01, 2.808e+01, 1.083e+02, ..., 1.418e-01, 2.218e-01,
         7.820e-02],
        [2.060e+01, 2.933e+01, 1.401e+02, ..., 2.650e-01, 4.087e-01,
         1.240e-01],
        [7.760e+00, 2.454e+01, 4.792e+01, ..., 0.000e+00, 2.871e-01,
         7.039e-02]]),
 'target': array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
1, 1,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
        0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0,
        1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0,
        1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1,
        1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0,
        0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1,
        1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0,
        0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0,
        1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1,
        1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0,
        0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0,
        0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0,
        1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1,
        1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1,
        1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0,
        1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1,
        1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1,
        1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1]),
 'frame': None,
 'target_names': array(['malignant', 'benign'], dtype='<U9'),
 'DESCR': '.. _breast_cancer_dataset:\n\nBreast cancer wisconsin (diagnostic)
dataset\n--------------------------------------------\n\n**Data Set Character
istics:**\n\n    :Number of Instances: 569\n\n    :Number of Attributes: 30 n
umeric, predictive attributes and the class\n\n    :Attribute Information:\n
- radius (mean of distances from center to points on the perimeter)\n
- texture (standard deviation of gray-scale values)\n        - perimeter\n
- area\n        - smoothness (local variation in radius lengths)\n        - c
ompactness (perimeter^2 / area - 1.0)\n        - concavity (severity of conca
ve portions of the contour)\n        - concave points (number of concave port
ions of the contour)\n        - symmetry\n        - fractal dimension ("coast
line approximation" - 1)\n\n        The mean, standard error, and "worst" or
largest (mean of the three\n        worst/largest values) of these features w
ere computed for each image,\n        resulting in 30 features.  For instanc
```

e, field 0 is Mean Radius, field\n 10 is Radius SE, field 20 is Worst Radius.\n\n - class:\n - WDBC-Malignant\n - WDBC-Benign\n\n :Summary Statistics:\n\n ===== ====== ======\n Min Max\n ===== ====== ======\n radius (mean): 6.981 28.11\n texture (mean): 9.71 39.28\n perimeter (mean): 43.79 188.5\n area (mean): 143.5 2501.0\n smoothness (mean): 0.053 0.163\n compactness (mean): 0.019 0.345\n concavity (mean): 0.0 0.427\n concave points (mean): 0.0 0.201\n symmetry (mean): 0.106 0.304\n fractal dimension (mean): 0.05 0.097\n radius (standard error): 0.112 2.873\n texture (standard error): 0.36 4.885\n perimeter (standard error): 0.757 21.98\n area (standard error): 6.802 542.2\n smoothness (standard error): 0.002 0.031\n compactness (standard error): 0.002 0.135\n concavity (standard error): 0.0 0.396\n concave points (standard error): 0.0 0.053\n symmetry (standard error): 0.008 0.079\n fractal dimension (standard error): 0.001 0.03\n radius (worst): 7.93 36.04\n texture (worst): 12.02 49.54\n perimeter (worst): 50.41 251.2\n area (worst): 185.2 4254.0\n smoothness (worst): 0.071 0.223\n compactness (worst): 0.027 1.058\n concavity (worst): 0.0 1.252\n concave points (worst): 0.0 0.291\n symmetry (worst): 0.156 0.664\n fractal dimension (worst): 0.055 0.208\n ===== ====== ======\n\n :Missing Attribute Values: None\n\n :Class Distribution: 212 - Malignant, 357 - Benign\n\n :Creator: Dr. William H. Wolberg, W. Nick Street, Olvi L. Mangasarian\n\n :Donor: Nick Street\n\n :Date: November, 1995\n\nThis is a copy of UCI ML Breast Cancer Wisconsin (Diagnostic) datasets.\nhttps://goo.gl/U2Uwz2\n\nFeatures are computed from a digitized image of a fine needle\naspirate (FNA) of a breast mass. They describe\ncharacteristics of the cell nuclei present in the image.\n\nSeparating plane described above was obtained using\nMultisurface Method-Tree (MSM-T) [K. P. Bennett, "Decision Tree\nConstruction Via Linear Programming." Proceedings of the 4th\nMidwest Artificial Intelligence and Cognitive Science Society,\npp. 97-101, 1992], a classification method which uses linear\nprogramming to construct a decision tree. Relevant features\nwere selected using an exhaustive search in the space of 1-4\nfeatures and 1-3 separating planes.\n\nThe actual linear program used to obtain the separating plane\nin the 3-dimensional space is that described in:\n[K. P. Bennett and O. L. Mangasarian: "Robust Linear\nProgramming Discrimination of Two Linearly Inseparable Sets",\nOptimization Methods and Software 1, 1992, 23-34].\n\nThis database is also available through the UW CS ftp server:\n\nftp ftp.cs.wisc.edu\ncd math-prog/cpo-dataset/machine-learn/WDBC/\n\n.. topic:: References\n\n - W.N. Street, W.H. Wolberg and O.L. Mangasarian. Nuclear feature extraction \n for breast tumor diagnosis. IS&T/SPIE 1993 International Symposium on \n Electronic Imaging: Science and Technology, volume 1905, pages 861-870,\n San Jose, CA, 1993.\n - O. L. Mangasarian, W.N. Street and W.H. Wolberg. Breast cancer diagnosis and \n prognosis via linear programming. Operations Research, 43(4), pages 570-577,\n July-August 1995.\n - W.H. Wolberg, W.N. Street, and O.L. Mangasarian. Machine learning techniques\n to diagnose breast cancer from fine-needle aspirates. Cancer Letters 77 (1994) \n 163-171.',
 'feature_names': array(['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'], dtype='<U23'),
 'filename': 'breast_cancer.csv',
 'data_module': 'sklearn.datasets.data'}
```

In [4]:
```
# Keys in Dataset
cancer_dataset.keys()
```

Out[4]:
```
dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names',
'filename', 'data_module'])
```

In [5]:
```
# Feature of Each cell in Numeric Format
cancer_dataset['data']
```

Out[5]:
```
array([[1.799e+01, 1.038e+01, 1.228e+02, ..., 2.654e-01, 4.601e-01,
        1.189e-01],
       [2.057e+01, 1.777e+01, 1.329e+02, ..., 1.860e-01, 2.750e-01,
        8.902e-02],
       [1.969e+01, 2.125e+01, 1.300e+02, ..., 2.430e-01, 3.613e-01,
        8.758e-02],
       ...,
       [1.660e+01, 2.808e+01, 1.083e+02, ..., 1.418e-01, 2.218e-01,
        7.820e-02],
       [2.060e+01, 2.933e+01, 1.401e+02, ..., 2.650e-01, 4.087e-01,
        1.240e-01],
       [7.760e+00, 2.454e+01, 4.792e+01, ..., 0.000e+00, 2.871e-01,
        7.039e-02]])
```

In [6]:
```
type(cancer_dataset['data'])
```

Out[6]:
```
numpy.ndarray
```

In [7]:
```
# Check Malignant and Benign Value
cancer_dataset['target']
```

Out[7]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0,
       1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0,
       1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1,
       1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0,
       0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1,
       1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0,
       0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0,
       1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0,
       0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0,
       0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0,
       1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1,
       1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0,
       1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1,
       1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1,
       1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1])
```

In [8]:
```
# Target Value Names

cancer_dataset['target_names']
```

Out[8]:
```
array(['malignant', 'benign'], dtype='<U9')
```

In [9]:
```
# Description of Data

print(cancer_dataset['DESCR'])
```

```
[K. P. Bennett and O. L. Mangasarian: "Robust Linear
Programming Discrimination of Two Linearly Inseparable Sets",
Optimization Methods and Software 1, 1992, 23-34].

This database is also available through the UW CS ftp server:

ftp ftp.cs.wisc.edu
cd math-prog/cpo-dataset/machine-learn/WDBC/

.. topic:: References

   - W.N. Street, W.H. Wolberg and O.L. Mangasarian. Nuclear feature extracti

on
     for breast tumor diagnosis. IS&T/SPIE 1993 International Symposium on
     Electronic Imaging: Science and Technology, volume 1905, pages 861-870,
     San Jose, CA, 1993.
   - O.L. Mangasarian, W.N. Street and W.H. Wolberg. Breast cancer diagnosis
 and
     prognosis via linear programming. Operations Research, 43(4), pages 570-
577
```

In [10]:
```
# Location of  Data File

print(cancer_dataset['filename'])
```

```
breast_cancer.csv
```

4- Create DataFrame

In [11]:
```
cancer_dataframe = pd.DataFrame(np.c_[cancer_dataset['data'], cancer_dataset['tar
                                columns=np.append(cancer_dataset['feature_names'],
```

In [12]:
```
# Data Frame to CSV Files

cancer_dataframe.to_csv('breast_cancer_dataset_frame.csv')
```

5- Exploring and Cleaning the Data

In [13]:
```
cancer_dataframe.head(5)
```

Out[13]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	d
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	

5 rows × 31 columns

In [14]:
```
cancer_dataframe.tail(5)
```

Out[14]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry
564	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	0.1726
565	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	0.1752
566	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	0.1590
567	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	0.2397
568	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	0.1587

5 rows × 31 columns

In [15]:
```
cancer_dataframe.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 [16]:
```
cancer_dataframe.describe()
```

Out[16]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	
count	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	56
mean	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	0.088799	
std	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	0.079720	
min	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	0.000000	
25%	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	0.029560	
50%	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630	0.061540	
75%	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400	0.130700	
max	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	0.426800	

8 rows × 31 columns

In [17]:
```
# Check Data is Clean or not

cancer_dataframe.isnull().sum()
```

Out[17]:
```
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
```

6- Data Visualization

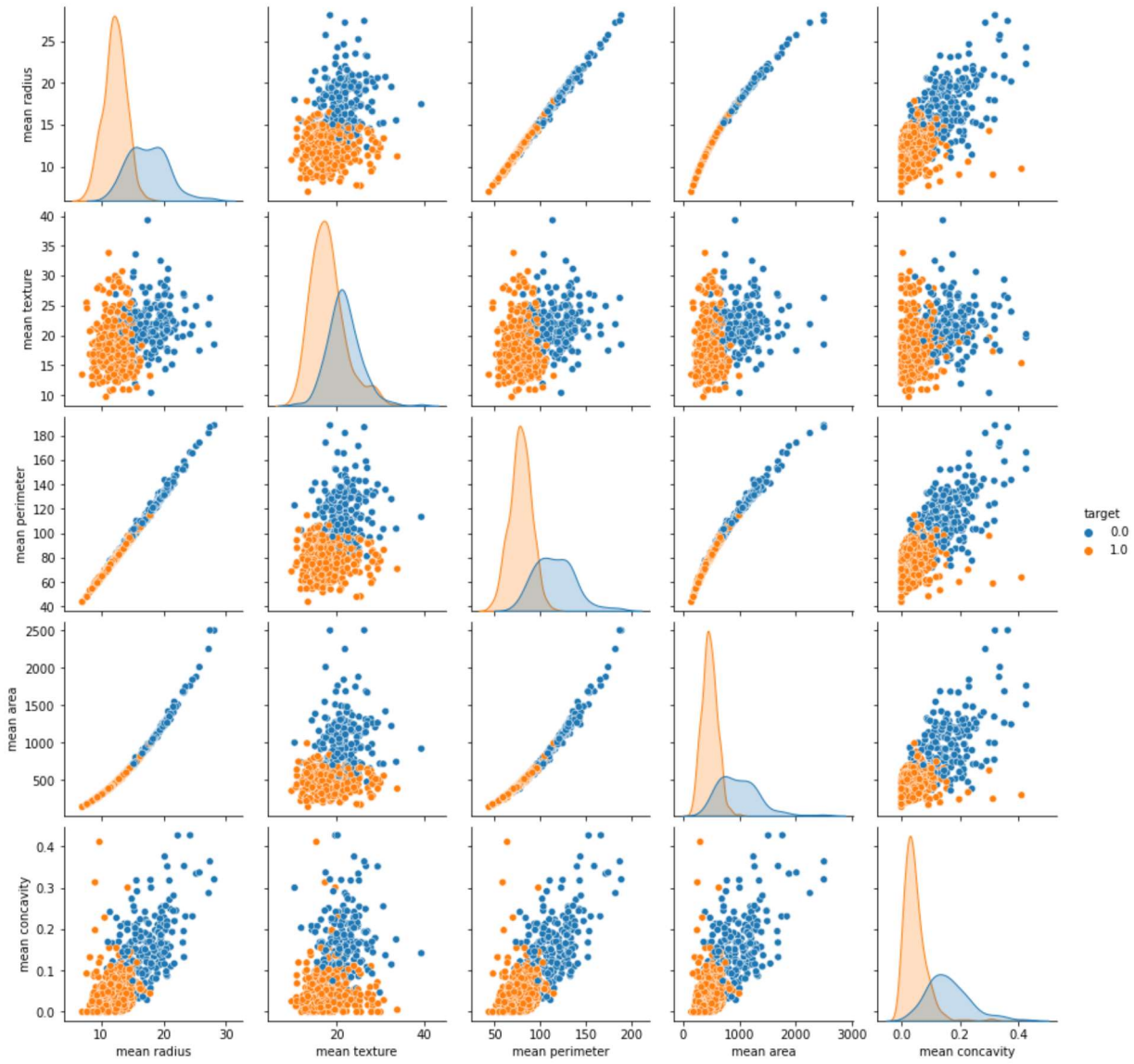
Pair Plot

In [18]:
```
# PairPlot of all Values

sns.pairplot(cancer_dataframe, hue='target')
```

Out[18]: <seaborn.axisgrid.PairGrid at 0x259316125b0>

In [19]:
```
# Pair Plot of Sample Features

sns.pairplot(cancer_dataframe, hue='target',
             vars=['mean radius', 'mean texture', 'mean perimeter', 'mean area',
```

Out[19]: <seaborn.axisgrid.PairGrid at 0x25958e020a0>

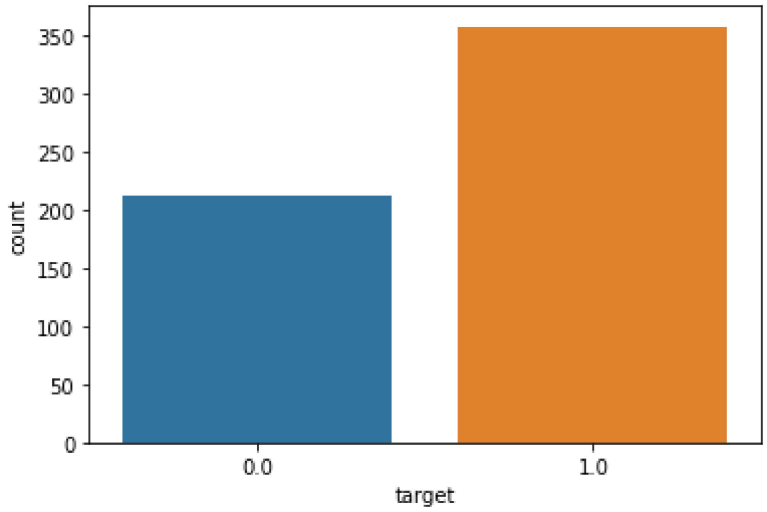


Counter Plot

In [20]:
```
sns.countplot(cancer_dataframe['target'])
```

```
D:\Annaconda Navigator\lib\site-packages\seaborn\_decorators.py:36: FutureWarni
ng: Pass the following variable as a keyword arg: x. From version 0.12, the onl
y valid positional argument will be `data`, and passing other arguments without
an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

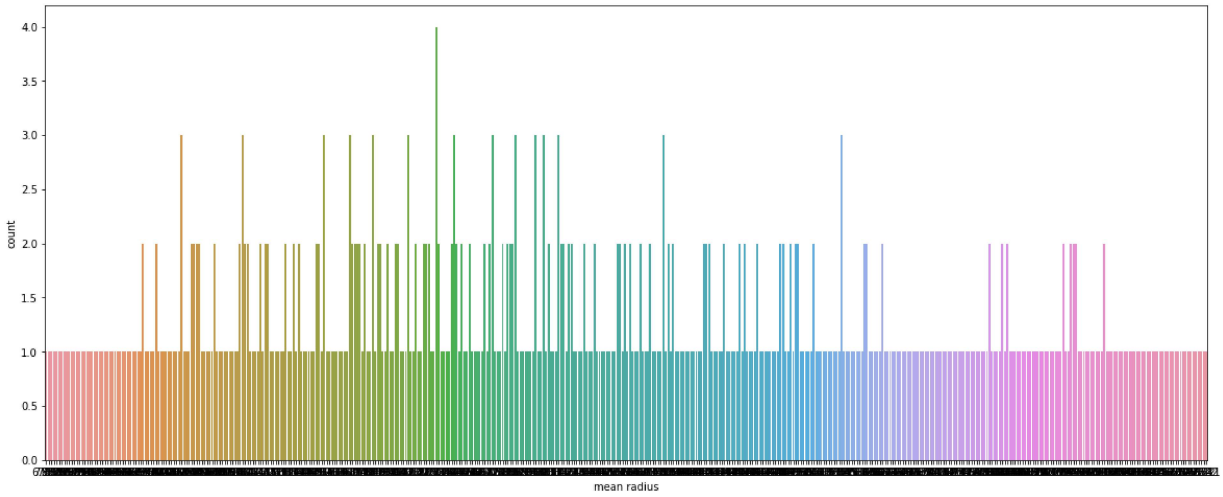
Out[20]:
```
<AxesSubplot:xlabel='target', ylabel='count'>
```



In [21]:
```
# Counter Plot of Mean Radius
plt.figure(figsize=(20,8))
sns.countplot(cancer_dataframe['mean radius'])
```

```
D:\Annaconda Navigator\lib\site-packages\seaborn\_decorators.py:36: FutureWarni
ng: Pass the following variable as a keyword arg: x. From version 0.12, the onl
y valid positional argument will be `data`, and passing other arguments without
an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

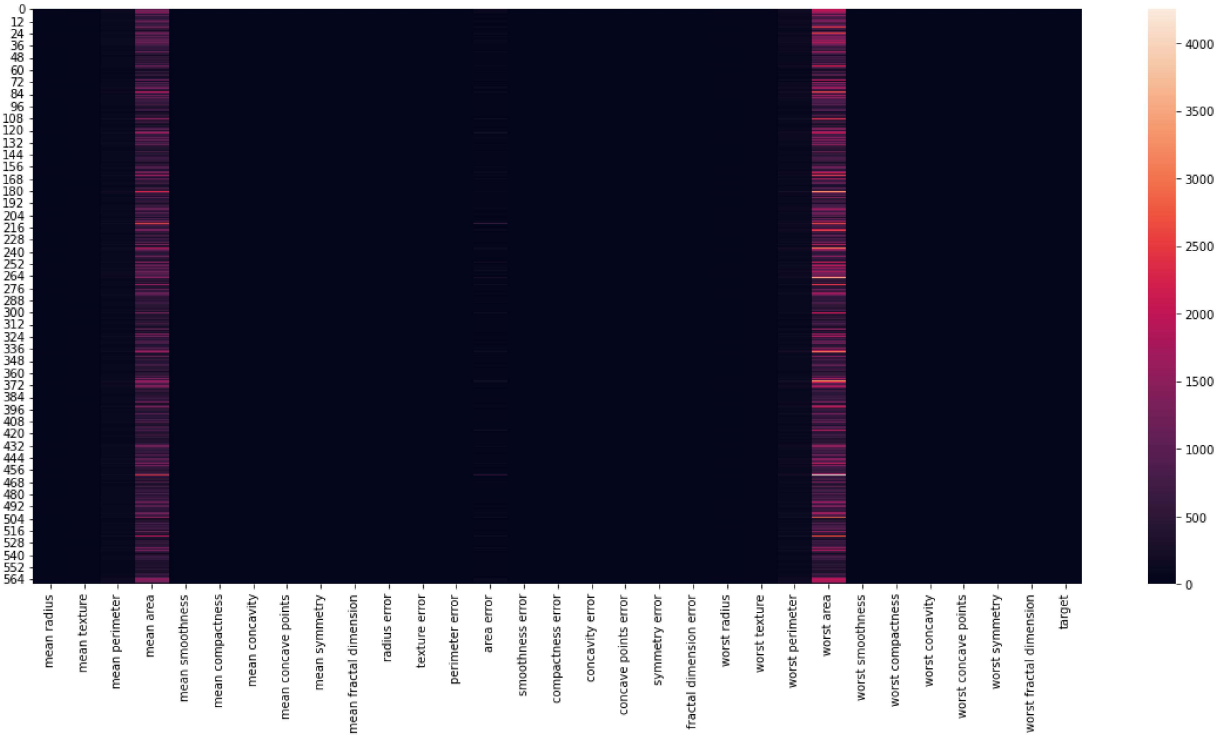
Out[21]: <AxesSubplot:xlabel='mean radius', ylabel='count'>



Heatmap

In [22]:
```
plt.figure(figsize=(20,9))
sns.heatmap(cancer_dataframe)
```

Out[22]: <AxesSubplot:>



Correlation Matrix

In [23]:
```
cancer_dataframe.corr()
```

Out[23]:

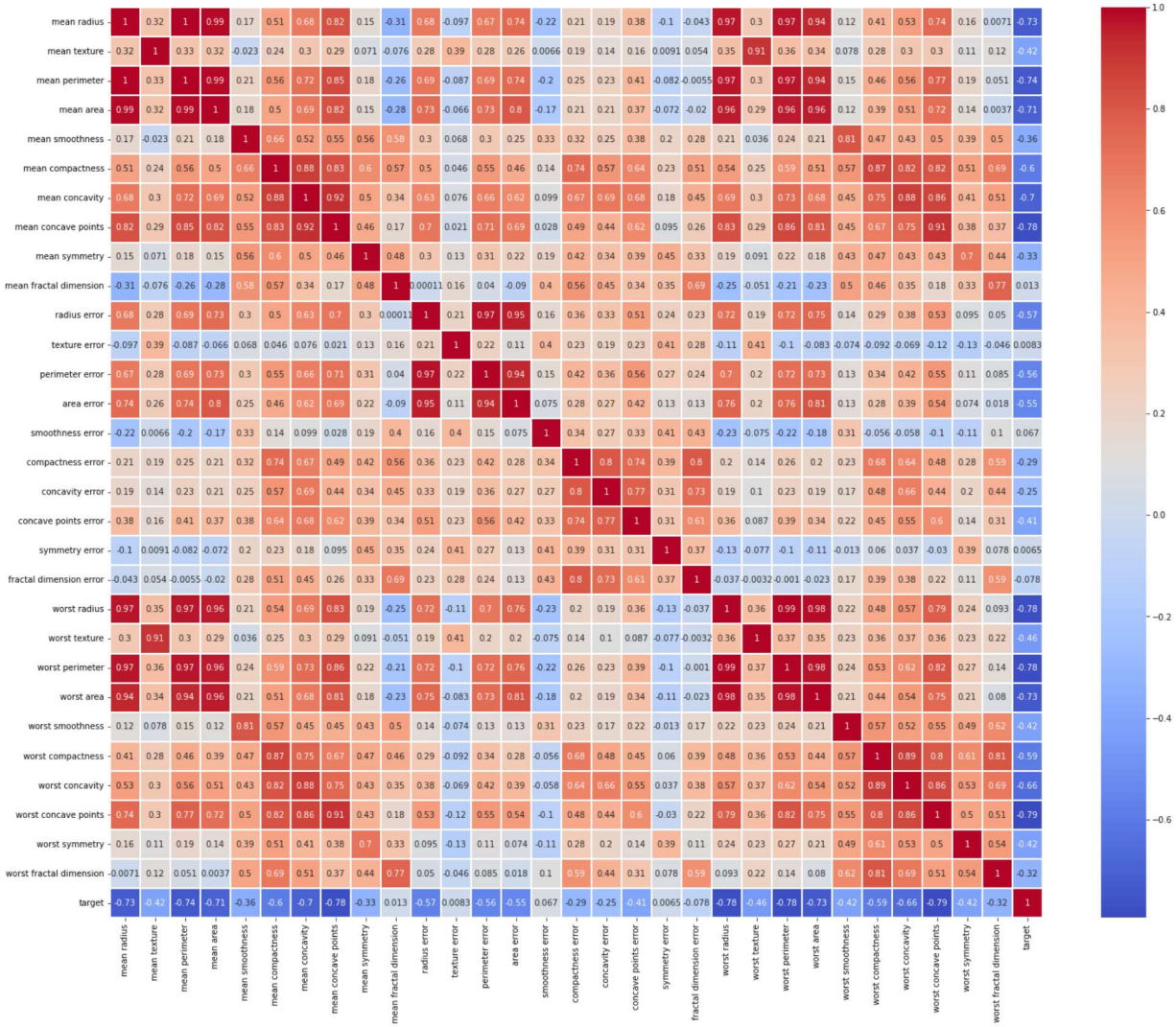
	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	co
mean radius	1.000000	0.323782	0.997855	0.987357	0.170581	0.506124	0.676764	0.8
mean texture	0.323782	1.000000	0.329533	0.321086	-0.023389	0.236702	0.302418	0.2
mean perimeter	0.997855	0.329533	1.000000	0.986507	0.207278	0.556936	0.716136	0.8
mean area	0.987357	0.321086	0.986507	1.000000	0.177028	0.498502	0.685983	0.8
mean smoothness	0.170581	-0.023389	0.207278	0.177028	1.000000	0.659123	0.521984	0.5
mean compactness	0.506124	0.236702	0.556936	0.498502	0.659123	1.000000	0.883121	0.8
mean concavity	0.676764	0.302418	0.716136	0.685983	0.521984	0.883121	1.000000	0.9
mean concave points	0.822529	0.293464	0.850977	0.823269	0.553695	0.831135	0.921391	1.0
mean symmetry	0.147741	0.071401	0.183027	0.151293	0.557775	0.602641	0.500667	0.4
mean fractal dimension	-0.311631	-0.076437	-0.261477	-0.283110	0.584792	0.565369	0.336783	0.1
radius error	0.679090	0.275869	0.691765	0.732562	0.301467	0.497473	0.631925	0.6
texture error	-0.097317	0.386358	-0.086761	-0.066280	0.068406	0.046205	0.076218	0.0
perimeter error	0.674172	0.281673	0.693135	0.726628	0.296092	0.548905	0.660391	0.7
area error	0.735864	0.259845	0.744983	0.800086	0.246552	0.455653	0.617427	0.6
smoothness error	-0.222600	0.006614	-0.202694	-0.166777	0.332375	0.135299	0.098564	0.0
compactness error	0.206000	0.191975	0.250744	0.212583	0.318943	0.738722	0.670279	0.4
concavity error	0.194204	0.143293	0.228082	0.207660	0.248396	0.570517	0.691270	0.4
concave points error	0.376169	0.163851	0.407217	0.372320	0.380676	0.642262	0.683260	0.6
symmetry error	-0.104321	0.009127	-0.081629	-0.072497	0.200774	0.229977	0.178009	0.0
fractal dimension error	-0.042641	0.054458	-0.005523	-0.019887	0.283607	0.507318	0.449301	0.2
worst radius	0.969539	0.352573	0.969476	0.962746	0.213120	0.535315	0.688236	0.8
worst texture	0.297008	0.912045	0.303038	0.287489	0.036072	0.248133	0.299879	0.2
worst perimeter	0.965137	0.358040	0.970387	0.959120	0.238853	0.590210	0.729565	0.8
worst area	0.941082	0.343546	0.941550	0.959213	0.206718	0.509604	0.675987	0.8

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	co
worst smoothness	0.119616	0.077503	0.150549	0.123523	0.805324	0.565541	0.448822	0.4
worst compactness	0.413463	0.277830	0.455774	0.390410	0.472468	0.865809	0.754968	0.6
worst concavity	0.526911	0.301025	0.563879	0.512606	0.434926	0.816275	0.884103	0.7
worst concave points	0.744214	0.295316	0.771241	0.722017	0.503053	0.815573	0.861323	0.9
worst symmetry	0.163953	0.105008	0.189115	0.143570	0.394309	0.510223	0.409464	0.3
worst fractal dimension	0.007066	0.119205	0.051019	0.003738	0.499316	0.687382	0.514930	0.3
target	-0.730029	-0.415185	-0.742636	-0.708984	-0.358560	-0.596534	-0.696360	-0.7

31 rows × 31 columns

In [24]:
```
# Heatmap of Correlation Matrix of Breast Cancer

plt.figure(figsize=(25,20))
sns.heatmap(cancer_dataframe.corr(),annot=True, cmap='coolwarm', linewidths=2)
```

Out[24]: <AxesSubplot:>



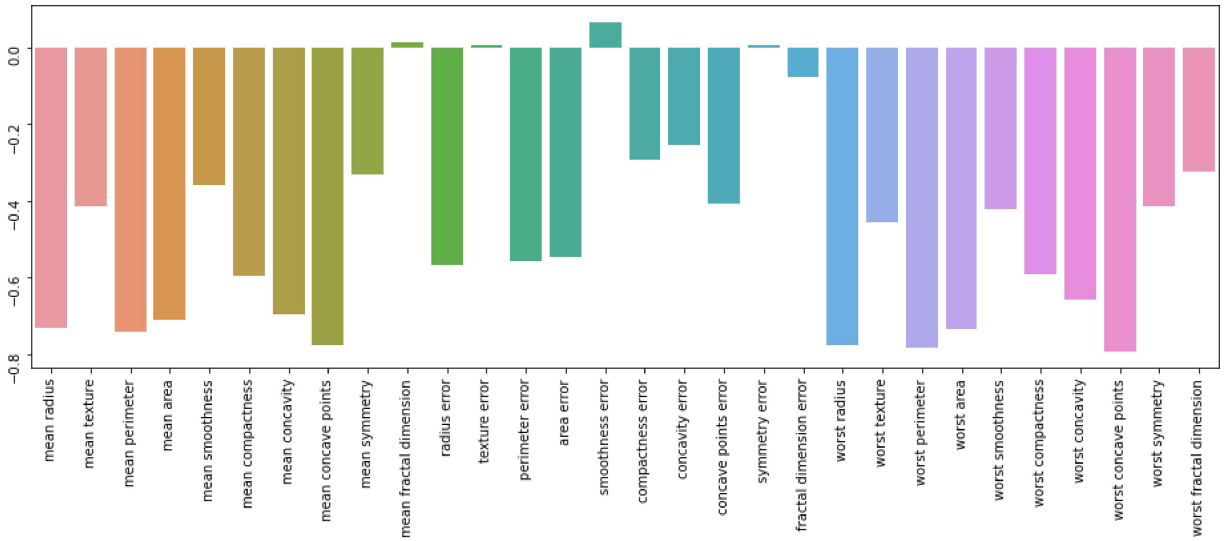
Correlation Barplot

In [25]:
```
# Create Second DataFrame by Dropping a Target

cancer_dataframe_2 = cancer_dataframe.drop(['target'], axis=1)
print('The Shape of Cancer DataFrame 2 is : ', cancer_dataframe_2.shape)
```

```
The Shape of Cancer DataFrame 2 is :  (569, 30)
```

In [26]:
```
# Visulize Correlation Barplot

plt.figure(figsize=(16,5))
ax = sns.barplot(cancer_dataframe_2.corrwith(cancer_dataframe.target).index, canc
ax.tick_params(labelrotation=90)
```

```
D:\Annaconda Navigator\lib\site-packages\seaborn\_decorators.py:36: FutureWarni
ng: Pass the following variables as keyword args: x, y. From version 0.12, the
only valid positional argument will be `data`, and passing other arguments with
out an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



In [27]:
```
cancer_dataframe_2.corrwith(cancer_dataframe.target).index
```

Out[27]:
```
Index(['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'],
      dtype='object')
```

7- Split The Dataset into train and test

In [28]:
```
# Input Variable
X =cancer_dataframe.drop(['target'], axis=1)
X.head(7)
```

Out[28]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	d
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	
5	12.45	15.70	82.57	477.1	0.12780	0.17000	0.1578	0.08089	0.2087	
6	18.25	19.98	119.60	1040.0	0.09463	0.10900	0.1127	0.07400	0.1794	

7 rows × 30 columns

In [29]:
```
Y = cancer_dataframe['target']
Y.head(6)
```

Out[29]:
```
0    0.0
1    0.0
2    0.0
3    0.0
4    0.0
5    0.0
Name: target, dtype: float64
```

In [30]:
```
# Split Datasets into Train and Test
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
```

In [31]:
```
X_train.head(5)
```

Out[31]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry
306	13.20	15.82	84.07	537.3	0.08511	0.05251	0.001461	0.003261	0.1632
410	11.36	17.57	72.49	399.8	0.08858	0.05313	0.027830	0.021000	0.1601
197	18.08	21.84	117.40	1024.0	0.07371	0.08642	0.110300	0.057780	0.1770
376	10.57	20.22	70.15	338.3	0.09073	0.16600	0.228000	0.059410	0.2188
244	19.40	23.50	129.10	1155.0	0.10270	0.15580	0.204900	0.088860	0.1978

5 rows × 30 columns

In [32]:
```
X_train.shape, X_test.shape, Y_train.shape, Y_test.shape
```

Out[32]:
```
((455, 30), (114, 30), (455,), (114,))
```

8- Feature Scaling

In [33]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train_sc = sc.fit_transform(X_train)
X_test_sc = sc.transform(X_test)
```

9- Building Machine Learning Models

In [34]:
```
from sklearn.metrics import accuracy_score, classification_report, confusion_matr
```

I- Support Vector Classifier

In [35]:
```
# Support Vector Classifier For Non Scaled Data
from sklearn.svm import SVC
support_vector_classifier = SVC()
support_vector_classifier.fit(X_train, Y_train)
y_pred_support_vector_classifier =support_vector_classifier.predict(X_test)

accuracy_score(Y_test, y_pred_support_vector_classifier)
```

Out[35]:
```
0.9385964912280702
```

In [36]:
```
# Support Vector Classifier For Scaled Data

from sklearn.svm import SVC
support_vector_classifier_with_scaled_data = SVC()
support_vector_classifier_with_scaled_data.fit(X_train_sc, Y_train)
y_pred_support_vector_classifier_with_scaled_data  = support_vector_classifier_wi

accuracy_score(Y_test, y_pred_support_vector_classifier_with_scaled_data)
```

Out[36]:
```
0.9649122807017544
```

II- Logistic Regression

In [37]:
```
# Logistic Regression for Non Scaled Data

from sklearn.linear_model import LogisticRegression
logistic_regression_model = LogisticRegression(random_state=51, penalty='l2')
logistic_regression_model.fit(X_train, Y_train)
y_pred_logistic_regression_model = logistic_regression_model.predict(X_test)

accuracy_score(Y_test, y_pred_logistic_regression_model)
```

```
D:\Annaconda Navigator\lib\site-packages\sklearn\linear_model\_logistic.py:814:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-
learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regressi
on (https://scikit-learn.org/stable/modules/linear_model.html#logistic-regressi
on)
  n_iter_i = _check_optimize_result(
```

Out[37]:
```
0.9649122807017544
```

In [38]:
```
# Logistic Regression For Scaled Data

from sklearn.linear_model import LogisticRegression
logistic_regression_model_with_scaled_data = LogisticRegression(random_state=52,
logistic_regression_model_with_scaled_data.fit(X_train_sc, Y_train)
y_pred_logistic_regression_model_with_scaled_data = logistic_regression_model_wit

accuracy_score(Y_test, y_pred_logistic_regression_model_with_scaled_data)
```

Out[38]:
```
0.9736842105263158
```

III- K-Nearest-Neighbours Classifier

In [39]:
```
# K-NN Classifier for Non_scaled Data

from sklearn.neighbors import KNeighborsClassifier
k_nearest_neighbours = KNeighborsClassifier(n_neighbors=5, metric='minkowski', p=
k_nearest_neighbours.fit(X_train, Y_train)
y_pred_k_nearest_neighbours_classifier = k_nearest_neighbours.predict(X_test)

accuracy_score(Y_test, y_pred_k_nearest_neighbours_classifier)
```

Out[39]:
```
0.9385964912280702
```

In [40]:
```
# K-NN Classifier For Scaled Data

from sklearn.neighbors import KNeighborsClassifier
k_nearest_neighbours_with_scaled_data = KNeighborsClassifier(n_neighbors=5, metri
k_nearest_neighbours_with_scaled_data.fit(X_train_sc, Y_train)
y_pred_k_nearest_neighbours_classifier_with_scaled_data = k_nearest_neighbours_wi

accuracy_score(Y_test, y_pred_k_nearest_neighbours_classifier_with_scaled_data)
```

Out[40]: 0.9649122807017544

IV- Naive Bayes Classifer

In [41]:
```
# Naive Bayes Classifier For non-scaled Data

from sklearn.naive_bayes import GaussianNB
naive_bayes_classifier = GaussianNB()
naive_bayes_classifier.fit(X_train,Y_train)
y_pred_naive_bayes_classifier = naive_bayes_classifier.predict(X_test)

accuracy_score(Y_test,y_pred_naive_bayes_classifier)
```

Out[41]: 0.9473684210526315

In [42]:
```
# Naive Bayes Classifier For Scaled Data

from sklearn.naive_bayes import GaussianNB
naive_bayes_classifier_with_scaled_data = GaussianNB()
naive_bayes_classifier_with_scaled_data.fit(X_train_sc, Y_train)
y_pred_naive_bayes_classifier_with_scaled_data = naive_bayes_classifier_with_scal

accuracy_score(Y_test, y_pred_naive_bayes_classifier_with_scaled_data)
```

Out[42]: 0.9385964912280702

V- Decision Tree Classifier

In [43]:
```
# Decision Tree Classifier For Non_Scaled Data

from sklearn.tree import DecisionTreeClassifier
decision_tree_classifier = DecisionTreeClassifier(random_state=52, criterion='ent
decision_tree_classifier.fit(X_train, Y_train)
y_pred_decision_tree_classifier = decision_tree_classifier.predict(X_test)

accuracy_score(Y_test,y_pred_decision_tree_classifier)
```

Out[43]: 0.956140350877193

In [44]:
```
# Decision Tree Classifier For Scaled Data

from sklearn.tree import DecisionTreeClassifier
decision_tree_classifier_with_scaled_data = DecisionTreeClassifier(random_state=5
decision_tree_classifier_with_scaled_data.fit(X_train_sc, Y_train)
y_pred_decision_tree_classifier_with_scaled_data = decision_tree_classifier_with_

accuracy_score(Y_test, y_pred_decision_tree_classifier_with_scaled_data)
```

Out[44]: 0.956140350877193

VI- Random Forest Classifier

In [45]:
```
# Random Forest Classifer For Non_Scaled_data

from sklearn.ensemble import RandomForestClassifier
random_forest_classifier = RandomForestClassifier(n_estimators=20, random_state=5
random_forest_classifier.fit(X_train,Y_train)
y_pred_random_forest_classifier = random_forest_classifier.predict(X_test)

accuracy_score(Y_test,y_pred_random_forest_classifier)
```

Out[45]: 0.9649122807017544

In [46]:
```
# Random Forest Classifier For Scaled Data

from sklearn.ensemble import RandomForestClassifier
random_forest_classifier_with_scaled_data = RandomForestClassifier(n_estimators=2
random_forest_classifier_with_scaled_data.fit(X_train_sc,Y_train)
y_pred_random_forest_classifier_with_scaled_data = random_forest_classifier_with_

accuracy_score(Y_test,y_pred_random_forest_classifier_with_scaled_data)
```

Out[46]: 0.9649122807017544

VII- ADABOOST Classifer

In [47]:
```
# ADABoost Classifier For Non_Scaled Data

from sklearn.ensemble import AdaBoostClassifier
ada_boost_classifier = AdaBoostClassifier(DecisionTreeClassifier(criterion='entro
                                          n_estimators=2000,
                                          learning_rate=0.1,
                                          algorithm='SAMME.R',
                                          random_state=52)

ada_boost_classifier.fit(X_train,Y_train)
y_pred_ada_boost_classifier = ada_boost_classifier.predict(X_test)

accuracy_score(Y_test,y_pred_ada_boost_classifier)
```

Out[47]:
```
0.9385964912280702
```

In [48]:
```
# ADABosst Classifier For Scaled Data

from sklearn.ensemble import AdaBoostClassifier
ada_boost_classifier_with_scaled_data = AdaBoostClassifier(DecisionTreeClassifier
                                                           n_estimators=2000,
                                                          learning_rate=0.1,
                                                          algorithm='SAMME.R',
                                                          random_state=52)

ada_boost_classifier_with_scaled_data.fit(X_train_sc,Y_train)
y_pred_ada_boost_classifier_with_scaled_data = ada_boost_classifier_with_scaled_d

accuracy_score(Y_test, y_pred_ada_boost_classifier_with_scaled_data)
```

Out[48]:
```
0.9385964912280702
```

VIII- XGBOOST Classifier

In [49]:
```
!pip install xgboost
```

```
Requirement already satisfied: xgboost in d:\annaconda navigator\lib\site-packa
ges (1.6.2)
Requirement already satisfied: scipy in d:\annaconda navigator\lib\site-package
s (from xgboost) (1.7.3)
Requirement already satisfied: numpy in d:\annaconda navigator\lib\site-package
s (from xgboost) (1.21.5)
```

In [50]:
```
#XGBoost Classifier With non_scaled data

from xgboost import XGBClassifier
xgb_classifier = XGBClassifier()
xgb_classifier.fit(X_train,Y_train)
y_pred_xgb_classifier = xgb_classifier.predict(X_test)

accuracy_score(Y_test,y_pred_xgb_classifier)
```

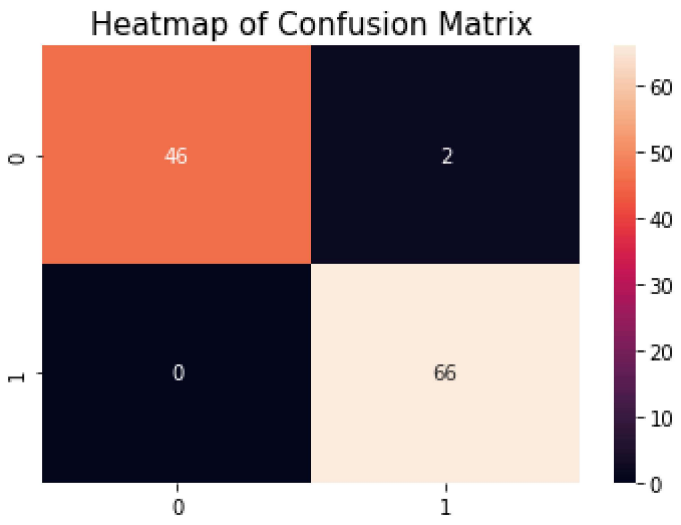
Out[50]: 0.9824561403508771

In [51]:
```
#XGBoost Classifier With Scaled data

from xgboost import XGBClassifier
xgb_classifier_with_scaled_data = XGBClassifier()
xgb_classifier_with_scaled_data.fit(X_train_sc,Y_train)
y_pred_xgb_classifier_with_scaled_data = xgb_classifier_with_scaled_data.predict(

accuracy_score(Y_test,y_pred_xgb_classifier_with_scaled_data)
```

Out[51]: 0.9824561403508771

10 Confusion Matrix

In [52]:
```
confusion_matrix = confusion_matrix(Y_test,y_pred_xgb_classifier_with_scaled_data
plt.title('Heatmap of Confusion Matrix', fontsize=15)
sns.heatmap(confusion_matrix,annot=True)
plt.show()
```



11- Classification Report

In [53]:
```
print(classification_report(Y_test,y_pred_xgb_classifier_with_scaled_data))
```

```
              precision    recall  f1-score   support

         0.0       1.00      0.96      0.98        48
         1.0       0.97      1.00      0.99        66

    accuracy                           0.98       114
   macro avg       0.99      0.98      0.98       114
weighted avg       0.98      0.98      0.98       114
```

12- Cross Validation of ML Model

In [54]:
```
# Cross Validation
from sklearn.model_selection import cross_val_score
cross_validation = cross_val_score(estimator=xgb_classifier_with_scaled_data, X=X
print('Cross Validation of Accuracy of XGB Model is : \n\n', cross_validation)
print('\n\nCross Validation of Mean Accuracy of XGB Model is : \n ', cross_valida
```

```
Cross Validation of Accuracy of XGB Model is :

 [1.         0.97826087 0.97826087 0.97826087 0.91304348 0.93333333
 1.         1.         0.95555556 0.88888889]


Cross Validation of Mean Accuracy of XGB Model is :
  0.96256038647343
```

13- Save This Model

In [55]:
```
## Pickle
import pickle

# save model
pickle.dump(xgb_classifier, open('breast_cancer_detector.pickle', 'wb'))

# load model
breast_cancer_detector_model = pickle.load(open('breast_cancer_detector.pickle',

# predict the output
y_pred = breast_cancer_detector_model.predict(X_test)

# show the accuracy
print('Accuracy of XGBoost model = ',accuracy_score(Y_test, y_pred))
```

```
Accuracy of XGBoost model =  0.9824561403508771
```

In [ ]:
```

```