breast-cancer-detection

April 20, 2024

[49]:
```
# import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import accuracy_score
```

[3]:
```
#Load breast cancer dataset
from sklearn.datasets import load_breast_cancer
cancer_dataset = load_breast_cancer()
```

[4]:
```
type(cancer_dataset)
```

[4]:
```
sklearn.utils._bunch.Bunch
```

[5]:
```
#Data Manipulation
cancer_dataset
```

[5]:
```
{'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, 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, 1, 1, 0, 1, 1, 0, 1, 0, 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, 1, 0, 1, 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,
        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, 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, 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
Characteristics:**\n\n:Number of Instances: 569\n\n:Number of Attributes: 30
numeric, 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    - compactness (perimeter^2 /
area - 1.0)\n    - concavity (severity of concave portions of the contour)\n
- concave points (number of concave portions of the contour)\n    - symmetry\n
- fractal dimension ("coastline approximation" - 1)\n\n    The mean, standard
error, and "worst" or largest (mean of the three\n    worst/largest values) of
these features were computed for each image,\n    resulting in 30 features.  For
instance, 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===================================== ====== ======\nradius (mean):
6.981  28.11\ntexture (mean):                      9.71   39.28\nperimeter
(mean):                    43.79  188.5\narea (mean):
143.5  2501.0\nsmoothness (mean):                   0.053  0.163\ncompactness
(mean):                  0.019  0.345\nconcavity (mean):
0.0    0.427\nconcave points (mean):               0.0    0.201\nsymmetry
(mean):                     0.106  0.304\nfractal dimension (mean):
0.05   0.097\nradius (standard error):             0.112  2.873\ntexture
```

(standard error): 0.36 4.885\nperimeter (standard error):
0.757 21.98\narea (standard error): 6.802 542.2\nsmoothness
(standard error): 0.002 0.031\ncompactness (standard error):
0.002 0.135\nconcavity (standard error): 0.0 0.396\nconcave points
(standard error): 0.0 0.053\nsymmetry (standard error): 0.008
0.079\nfractal dimension (standard error): 0.001 0.03\nradius (worst):
7.93 36.04\ntexture (worst): 12.02 49.54\nperimeter
(worst): 50.41 251.2\narea (worst):
185.2 4254.0\nsmoothness (worst): 0.071 0.223\ncompactness
(worst): 0.027 1.058\nconcavity (worst):
0.0 1.252\nconcave points (worst): 0.0 0.291\nsymmetry
(worst): 0.156 0.664\nfractal 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|details-
start|\n**References**\n|details-split|\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.\n\n|details-end|\n',
 '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'}
```

[12]:
```
#keys in dataset
cancer_dataset.keys()
```

[12]:
```
dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names',
'filename', 'data_module'])
```

[13]:
```
# Features of each cells in numeric format
cancer_dataset['data']
```

[13]:
```
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]])
```

[14]:
```
# Description of data
print(cancer_dataset['DESCR'])
```

```
.. _breast_cancer_dataset:

Breast cancer wisconsin (diagnostic) dataset
--------------------------------------------

**Data Set Characteristics:**

:Number of Instances: 569

:Number of Attributes: 30 numeric, predictive attributes and the class

:Attribute Information:
    - 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)

    The mean, standard error, and "worst" or largest (mean of the three
    worst/largest values) of these features were computed for each image,
    resulting in 30 features.  For instance, field 0 is Mean Radius, field
    10 is Radius SE, field 20 is Worst Radius.

    - class:
            - WDBC-Malignant
            - WDBC-Benign

:Summary Statistics:

===================================== ====== ======
                                        Min    Max
===================================== ====== ======
radius (mean):                        6.981  28.11
texture (mean):                       9.71   39.28
perimeter (mean):                     43.79  188.5
area (mean):                          143.5  2501.0
smoothness (mean):                    0.053  0.163
compactness (mean):                   0.019  0.345
concavity (mean):                     0.0    0.427
concave points (mean):                0.0    0.201
symmetry (mean):                      0.106  0.304
fractal dimension (mean):             0.05   0.097
radius (standard error):              0.112  2.873
texture (standard error):             0.36   4.885
perimeter (standard error):           0.757  21.98
area (standard error):                6.802  542.2
smoothness (standard error):          0.002  0.031
compactness (standard error):         0.002  0.135
concavity (standard error):           0.0    0.396
concave points (standard error):      0.0    0.053
symmetry (standard error):            0.008  0.079
fractal dimension (standard error):   0.001  0.03
radius (worst):                       7.93   36.04
texture (worst):                      12.02  49.54
perimeter (worst):                    50.41  251.2
area (worst):                         185.2  4254.0
```

```
smoothness (worst):                   0.071  0.223
compactness (worst):                  0.027  1.058
concavity (worst):                    0.0    1.252
concave points (worst):               0.0    0.291
symmetry (worst):                     0.156  0.664
fractal dimension (worst):            0.055  0.208
==================================== ====== ======

:Missing Attribute Values: None

:Class Distribution: 212 - Malignant, 357 - Benign

:Creator:  Dr. William H. Wolberg, W. Nick Street, Olvi L. Mangasarian

:Donor: Nick Street

:Date: November, 1995

This is a copy of UCI ML Breast Cancer Wisconsin (Diagnostic) datasets.
https://goo.gl/U2Uwz2

Features are computed from a digitized image of a fine needle
aspirate (FNA) of a breast mass.  They describe
characteristics of the cell nuclei present in the image.

Separating plane described above was obtained using
Multisurface Method-Tree (MSM-T) [K. P. Bennett, "Decision Tree
Construction Via Linear Programming." Proceedings of the 4th
Midwest Artificial Intelligence and Cognitive Science Society,
pp. 97-101, 1992], a classification method which uses linear
programming to construct a decision tree.  Relevant features
were selected using an exhaustive search in the space of 1-4
features and 1-3 separating planes.

The actual linear program used to obtain the separating plane
in the 3-dimensional space is that described in:
[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/

|details-start|
**References**
|details-split|
```

- W.N. Street, W.H. Wolberg and O.L. Mangasarian. Nuclear feature extraction 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, July-August 1995.
- W.H. Wolberg, W.N. Street, and O.L. Mangasarian. Machine learning techniques to diagnose breast cancer from fine-needle aspirates. Cancer Letters 77 (1994) 163-171.

|details-end|

[15]:
```
# Name of features
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']
```

[16]:
```
# Location/path of data file
print(cancer_dataset['filename'])
```

```
breast_cancer.csv
```

[17]:
```
# Create datafrmae
cancer_df = pd.DataFrame(np.c_[cancer_dataset['data'],cancer_dataset['target']],
             columns = np.append(cancer_dataset['feature_names'], ['target']))
```

[18]:
```
# Head of cancer DataFrame
cancer_df.head(6)
```

[18]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	\
0	17.99	10.38	122.80	1001.0	0.11840	
1	20.57	17.77	132.90	1326.0	0.08474	
2	19.69	21.25	130.00	1203.0	0.10960	
3	11.42	20.38	77.58	386.1	0.14250	
4	20.29	14.34	135.10	1297.0	0.10030	
5	12.45	15.70	82.57	477.1	0.12780	

```
   mean compactness  mean concavity  mean concave points  mean symmetry  \
0           0.27760          0.3001              0.14710         0.2419
1           0.07864          0.0869              0.07017         0.1812
2           0.15990          0.1974              0.12790         0.2069
3           0.28390          0.2414              0.10520         0.2597
4           0.13280          0.1980              0.10430         0.1809
5           0.17000          0.1578              0.08089         0.2087

   mean fractal dimension  …  worst texture  worst perimeter  worst area  \
0                 0.07871  …          17.33           184.60      2019.0
1                 0.05667  …          23.41           158.80      1956.0
2                 0.05999  …          25.53           152.50      1709.0
3                 0.09744  …          26.50            98.87       567.7
4                 0.05883  …          16.67           152.20      1575.0
5                 0.07613  …          23.75           103.40       741.6

   worst smoothness  worst compactness  worst concavity  worst concave points  \
0            0.1622             0.6656           0.7119                0.2654
1            0.1238             0.1866           0.2416                0.1860
2            0.1444             0.4245           0.4504                0.2430
3            0.2098             0.8663           0.6869                0.2575
4            0.1374             0.2050           0.4000                0.1625
5            0.1791             0.5249           0.5355                0.1741

   worst symmetry  worst fractal dimension  target
0          0.4601                  0.11890     0.0
1          0.2750                  0.08902     0.0
2          0.3613                  0.08758     0.0
3          0.6638                  0.17300     0.0
4          0.2364                  0.07678     0.0
5          0.3985                  0.12440     0.0

[6 rows x 31 columns]
```

[19]:
```
# Tail of cancer DataFrame
cancer_df.tail(6)
```

[19]:
```
     mean radius  mean texture  mean perimeter  mean area  mean smoothness  \
563        20.92         25.09          143.00     1347.0          0.10990
564        21.56         22.39          142.00     1479.0          0.11100
565        20.13         28.25          131.20     1261.0          0.09780
566        16.60         28.08          108.30      858.1          0.08455
567        20.60         29.33          140.10     1265.0          0.11780
568         7.76         24.54           47.92      181.0          0.05263

     mean compactness  mean concavity  mean concave points  mean symmetry  \
563           0.22360         0.31740              0.14740         0.2149
```

```
564          0.11590         0.24390              0.13890         0.1726
565          0.10340         0.14400              0.09791         0.1752
566          0.10230         0.09251              0.05302         0.1590
567          0.27700         0.35140              0.15200         0.2397
568          0.04362         0.00000              0.00000         0.1587

     mean fractal dimension  …  worst texture  worst perimeter  worst area  \
563                 0.06879  …          29.41           179.10      1819.0
564                 0.05623  …          26.40           166.10      2027.0
565                 0.05533  …          38.25           155.00      1731.0
566                 0.05648  …          34.12           126.70      1124.0
567                 0.07016  …          39.42           184.60      1821.0
568                 0.05884  …          30.37            59.16       268.6

     worst smoothness  worst compactness  worst concavity  \
563           0.14070            0.41860           0.6599
564           0.14100            0.21130           0.4107
565           0.11660            0.19220           0.3215
566           0.11390            0.30940           0.3403
567           0.16500            0.86810           0.9387
568           0.08996            0.06444           0.0000

     worst concave points  worst symmetry  worst fractal dimension  target
563                0.2542          0.2929                  0.09873     0.0
564                0.2216          0.2060                  0.07115     0.0
565                0.1628          0.2572                  0.06637     0.0
566                0.1418          0.2218                  0.07820     0.0
567                0.2650          0.4087                  0.12400     0.0
568                0.0000          0.2871                  0.07039     1.0

[6 rows x 31 columns]
```

[20]:
```
# Information of cancer Dataframe
cancer_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
```

[21]:
```
# Numerical distribution of data
cancer_df.describe()
```

[21]:

	mean radius	mean texture	mean perimeter	mean area
count	569.000000	569.000000	569.000000	569.000000
mean	14.127292	19.289649	91.969033	654.889104
std	3.524049	4.301036	24.298981	351.914129
min	6.981000	9.710000	43.790000	143.500000
25%	11.700000	16.170000	75.170000	420.300000
50%	13.370000	18.840000	86.240000	551.100000
75%	15.780000	21.800000	104.100000	782.700000
max	28.110000	39.280000	188.500000	2501.000000

	mean smoothness	mean compactness	mean concavity	mean concave points
count	569.000000	569.000000	569.000000	569.000000
mean	0.096360	0.104341	0.088799	0.048919
std	0.014064	0.052813	0.079720	0.038803
min	0.052630	0.019380	0.000000	0.000000
25%	0.086370	0.064920	0.029560	0.020310
50%	0.095870	0.092630	0.061540	0.033500

```
75%          0.105300          0.130400        0.130700             0.074000
max          0.163400          0.345400        0.426800             0.201200

       mean symmetry  mean fractal dimension  …  worst texture  \
count     569.000000              569.000000  …     569.000000
mean        0.181162                0.062798  …      25.677223
std         0.027414                0.007060  …       6.146258
min         0.106000                0.049960  …      12.020000
25%         0.161900                0.057700  …      21.080000
50%         0.179200                0.061540  …      25.410000
75%         0.195700                0.066120  …      29.720000
max         0.304000                0.097440  …      49.540000

       worst perimeter   worst area  worst smoothness  worst compactness  \
count       569.000000   569.000000        569.000000         569.000000
mean        107.261213   880.583128          0.132369           0.254265
std          33.602542   569.356993          0.022832           0.157336
min          50.410000   185.200000          0.071170           0.027290
25%          84.110000   515.300000          0.116600           0.147200
50%          97.660000   686.500000          0.131300           0.211900
75%         125.400000  1084.000000          0.146000           0.339100
max         251.200000  4254.000000          0.222600           1.058000

       worst concavity  worst concave points  worst symmetry  \
count       569.000000            569.000000      569.000000
mean          0.272188              0.114606        0.290076
std           0.208624              0.065732        0.061867
min           0.000000              0.000000        0.156500
25%           0.114500              0.064930        0.250400
50%           0.226700              0.099930        0.282200
75%           0.382900              0.161400        0.317900
max           1.252000              0.291000        0.663800

       worst fractal dimension      target
count               569.000000  569.000000
mean                  0.083946    0.627417
std                   0.018061    0.483918
min                   0.055040    0.000000
25%                   0.071460    0.000000
50%                   0.080040    1.000000
75%                   0.092080    1.000000
max                   0.207500    1.000000

[8 rows x 31 columns]
```

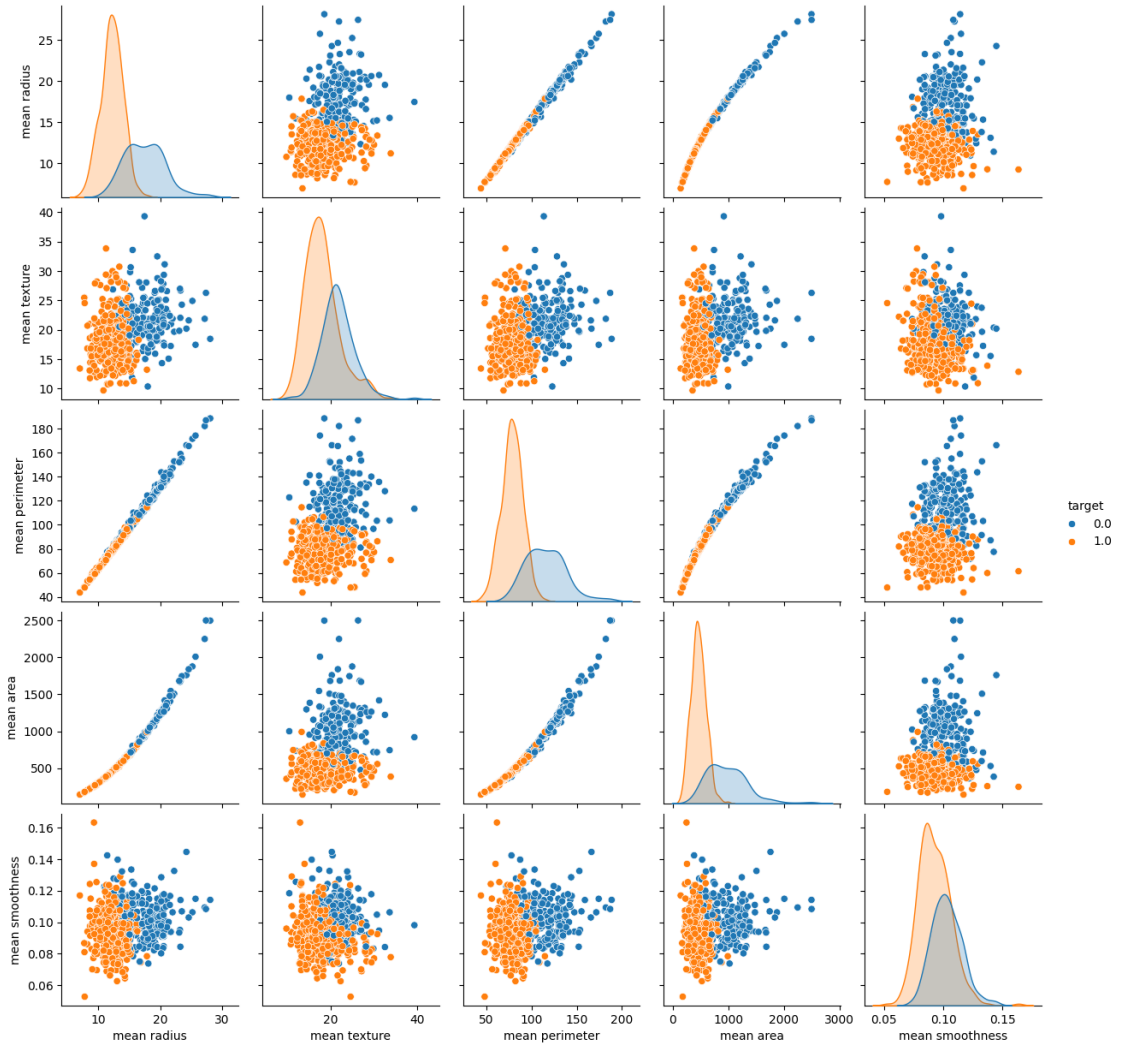
[18]:
```
#DATA VISUALIZATION
#PAIR PLOT OF BREAST CANCER DATA
```

```
# Pairplot of cancer dataframe
sns.pairplot(cancer_df, hue = 'target')
```

[18]: <seaborn.axisgrid.PairGrid at 0x1cac4514590>

```
# pair plot of sample feature
sns.pairplot(cancer_df, hue = 'target',
             vars = ['mean radius', 'mean texture', 'mean perimeter', 'mean␣
 ↪area', 'mean smoothness'] )
```

[40]: <seaborn.axisgrid.PairGrid at 0x1f3325d9cd0>



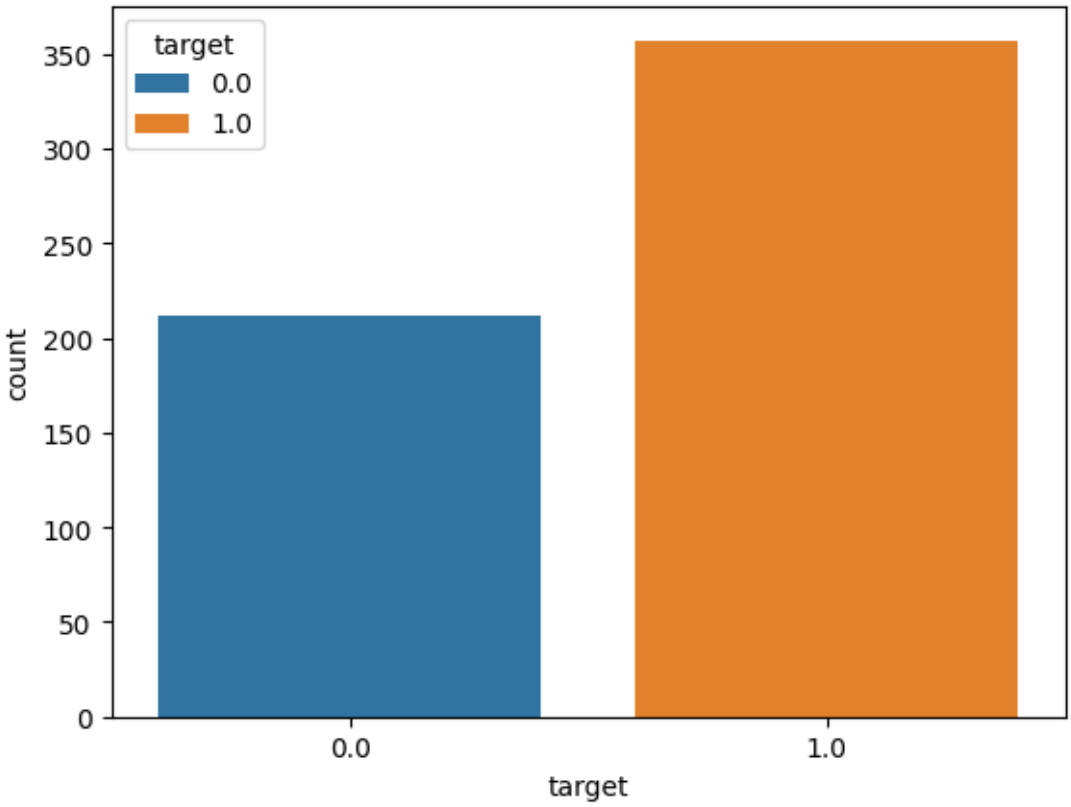
[22]:
```
import seaborn as sns
```

[19]:
```
print(cancer_df["target"])
```

```
0      0.0
1      0.0
2      0.0
3      0.0
4      0.0
      ...
564    0.0
565    0.0
566    0.0
567    0.0
568    1.0
```

```
Name: target, Length: 569, dtype: float64
```

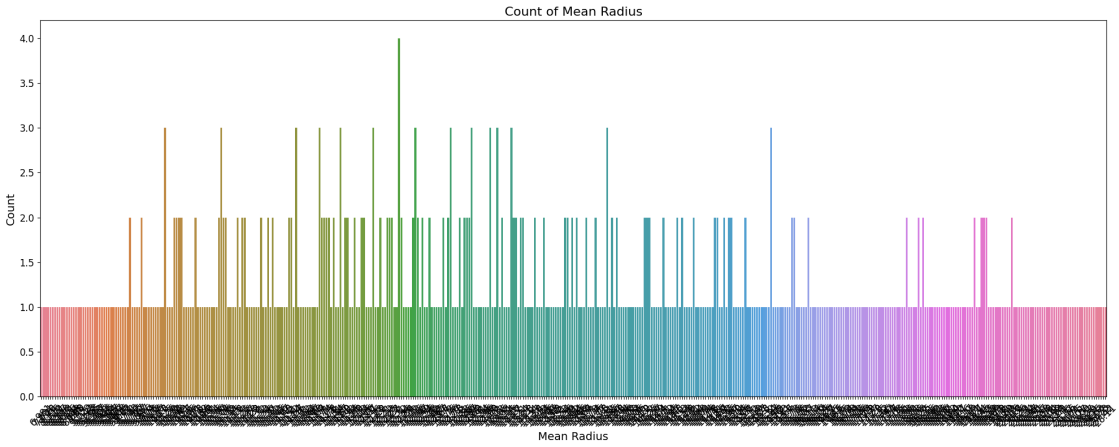
[25]:
```
#COUNTERPLOT

# Count the target class
# X la nehmi define karayche ahe nahi tar each input la target mannar
ax=sns.countplot(x="target",data=cancer_df,hue = 'target')
```



[41]:
```
custom_palette = sns.color_palette("husl", len(cancer_df['mean radius'].
 ↪unique()))

plt.figure(figsize=(20, 8))
a = sns.countplot(x='mean radius', data=cancer_df, palette=custom_palette)


a.set_xlabel('Mean Radius', fontsize=14)  # Set x-axis label with fontsize
a.set_ylabel('Count', fontsize=14)  # Set y-axis label with fontsize
a.set_title('Count of Mean Radius', fontsize=16)  # Set plot title with fontsize
a.tick_params(axis='x', labelrotation=45, labelsize=12)  # Rotate and adjust␣
 ↪fontsize of x-axis labels
a.tick_params(axis='y', labelsize=12)  # Adjust fontsize of y-axis labels
```

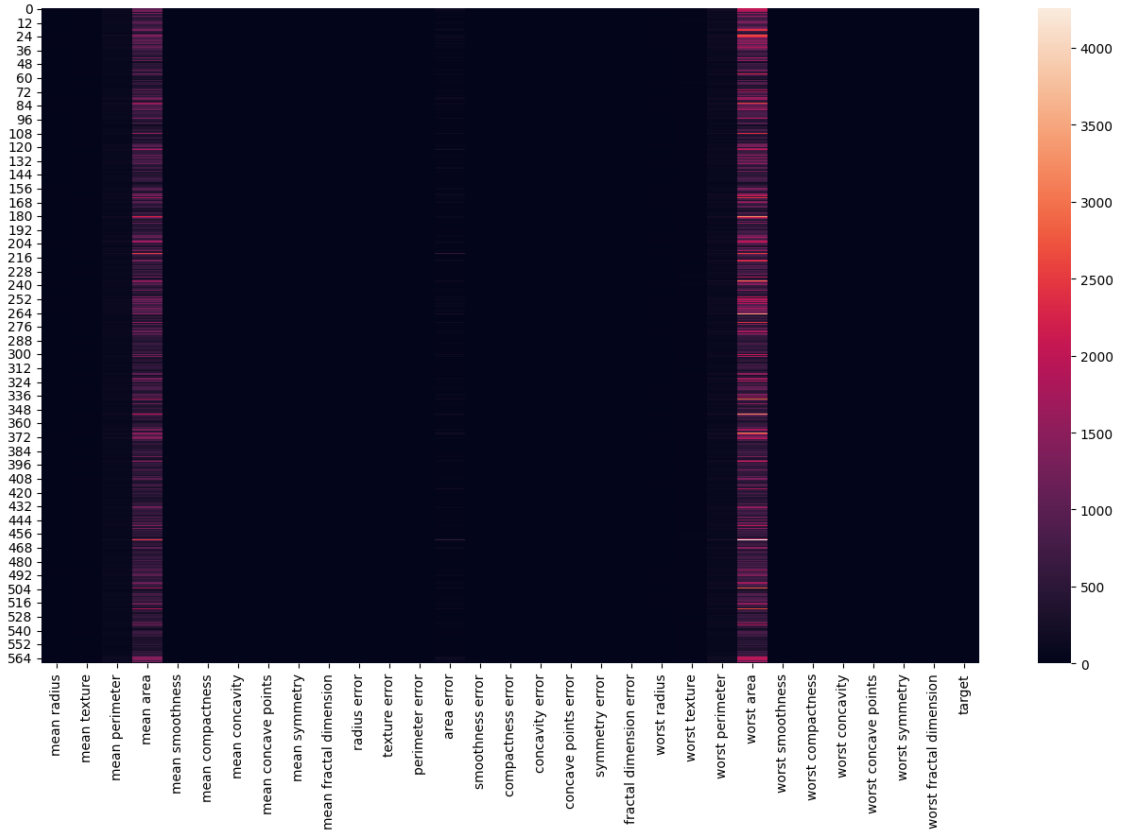
```
plt.tight_layout()  # Adjust layout to prevent labels from overlapping
plt.show()
```

```
C:\Users\ASUS\AppData\Local\Temp\ipykernel_13908\2148108818.py:4: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  a = sns.countplot(x='mean radius', data=cancer_df, palette=custom_palette)
```



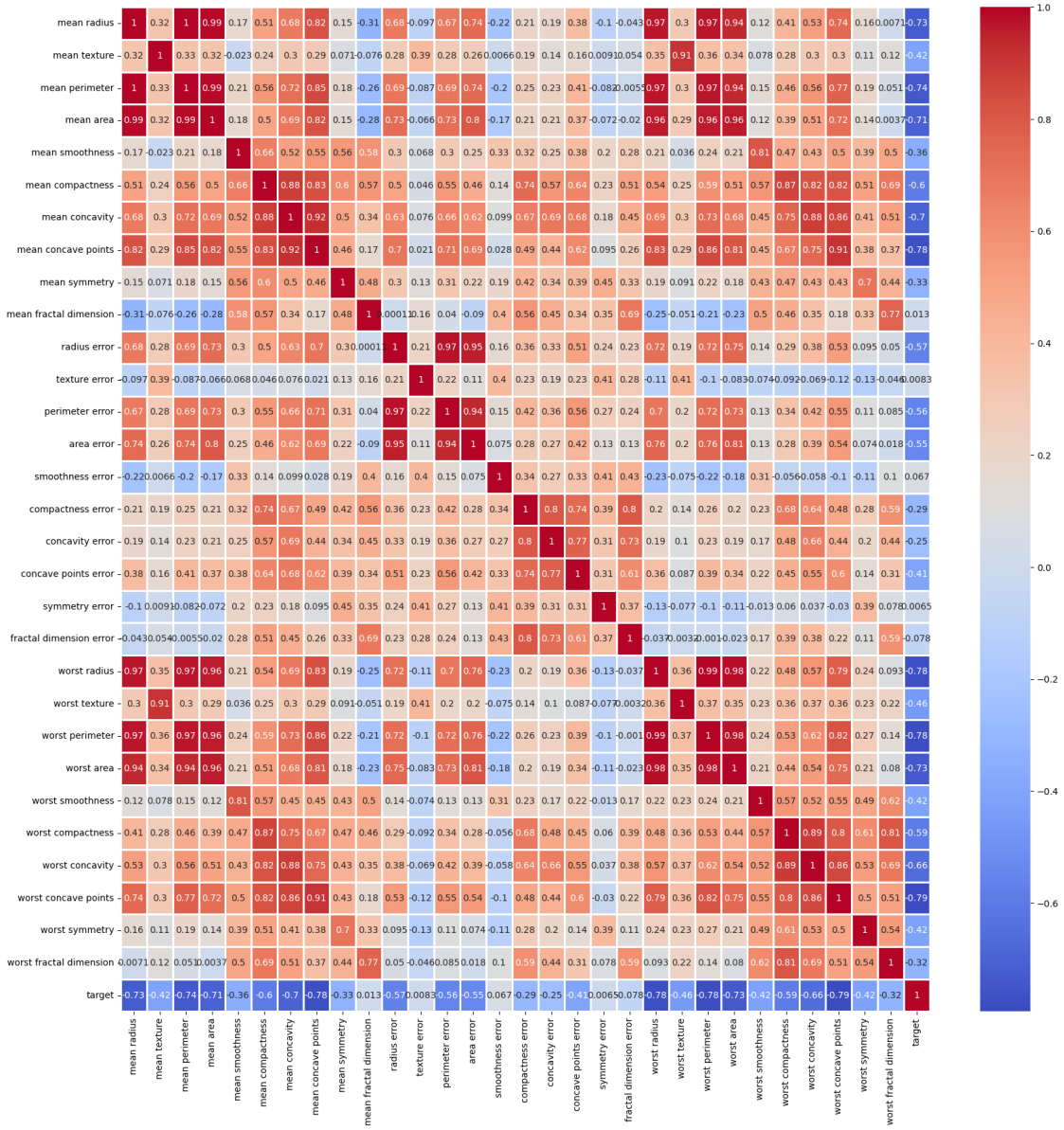
[31]:
```
#HEATMAP

# heatmap of DataFrame
plt.figure(figsize=(16,9))
sns.heatmap(cancer_df)
```

[31]: <Axes: >



[32]:
```
#Heatmap of a correlation matrix

# Heatmap of Correlation matrix of breast cancer DataFrame
plt.figure(figsize=(20,20))
sns.heatmap(cancer_df.corr(), annot = True, cmap ='coolwarm', linewidths=2)
```

[32]: <Axes: >



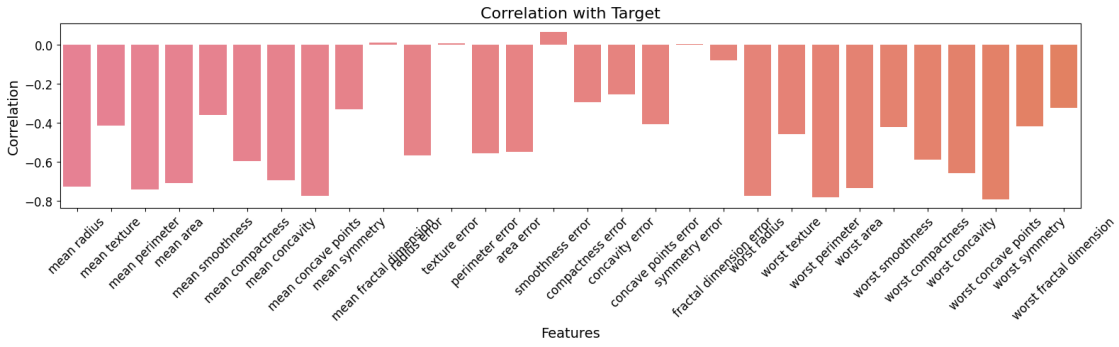
[33]:
```
#Correlation barplot

# create second DataFrame by droping target
cancer_df2 = cancer_df.drop(['target'], axis = 1)
print("The shape of 'cancer_df2' is : ", cancer_df2.shape)
```

```
The shape of 'cancer_df2' is :  (569, 30)
```

[40]:
```
colors = sns.color_palette('husl', len(cancer_df2))

plt.figure(figsize=(16, 5))
```

```
ax = sns.barplot(x=cancer_df2.corrwith(cancer_df['target']).index, y=cancer_df2.
 ↪corrwith(cancer_df['target']), palette=colors)


ax.set_xlabel('Features', fontsize=14)  # Set x-axis label with fontsize
ax.set_ylabel('Correlation', fontsize=14)  # Set y-axis label with fontsize
ax.set_title('Correlation with Target', fontsize=16)  # Set plot title with␣
 ↪fontsize
ax.tick_params(axis='x', labelrotation=45, labelsize=12)  # Rotate and adjust␣
 ↪fontsize of x-axis labels
ax.tick_params(axis='y', labelsize=12)  # Adjust fontsize of y-axis labels

plt.tight_layout()
plt.show()
```

```
C:\Users\ASUS\AppData\Local\Temp\ipykernel_13908\2288618361.py:4: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  ax = sns.barplot(x=cancer_df2.corrwith(cancer_df['target']).index,
y=cancer_df2.corrwith(cancer_df['target']), palette=colors)
C:\Users\ASUS\AppData\Local\Temp\ipykernel_13908\2288618361.py:4: UserWarning:
The palette list has more values (569) than needed (30), which may not be
intended.
  ax = sns.barplot(x=cancer_df2.corrwith(cancer_df['target']).index,
y=cancer_df2.corrwith(cancer_df['target']), palette=colors)
```



[42]:
```
#DATA PREPROCESSING

# input variable
X = cancer_df.drop(['target'], axis = 1)
X.head(6)
```

[42]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	\
0	17.99	10.38	122.80	1001.0	0.11840	
1	20.57	17.77	132.90	1326.0	0.08474	
2	19.69	21.25	130.00	1203.0	0.10960	
3	11.42	20.38	77.58	386.1	0.14250	
4	20.29	14.34	135.10	1297.0	0.10030	
5	12.45	15.70	82.57	477.1	0.12780	

	mean compactness	mean concavity	mean concave points	mean symmetry	\
0	0.27760	0.3001	0.14710	0.2419	
1	0.07864	0.0869	0.07017	0.1812	
2	0.15990	0.1974	0.12790	0.2069	
3	0.28390	0.2414	0.10520	0.2597	
4	0.13280	0.1980	0.10430	0.1809	
5	0.17000	0.1578	0.08089	0.2087	

	mean fractal dimension	…	worst radius	worst texture	worst perimeter	\
0	0.07871	…	25.38	17.33	184.60	
1	0.05667	…	24.99	23.41	158.80	
2	0.05999	…	23.57	25.53	152.50	
3	0.09744	…	14.91	26.50	98.87	
4	0.05883	…	22.54	16.67	152.20	
5	0.07613	…	15.47	23.75	103.40	

	worst area	worst smoothness	worst compactness	worst concavity	\
0	2019.0	0.1622	0.6656	0.7119	
1	1956.0	0.1238	0.1866	0.2416	
2	1709.0	0.1444	0.4245	0.4504	
3	567.7	0.2098	0.8663	0.6869	
4	1575.0	0.1374	0.2050	0.4000	
5	741.6	0.1791	0.5249	0.5355	

	worst concave points	worst symmetry	worst fractal dimension
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
5	0.1741	0.3985	0.12440

[6 rows x 30 columns]

[43]:
```
# output variable
y = cancer_df['target']
y.head(6)
```

[43]:
```
0    0.0
1    0.0
2    0.0
3    0.0
4    0.0
5    0.0
Name: target, dtype: float64
```

[44]:
```
#split dataset 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= 5)
```

[45]:
```
# Feature scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train_sc = sc.fit_transform(X_train)
X_test_sc = sc.transform(X_test)
```

[ ]:
```
#Model Building
```

[50]:
```
#Machine using SVC
from sklearn.svm import SVC

svc_classifier = SVC(kernel='linear')
svc_classifier.fit(X_train, y_train)


y_pred_svc = svc_classifier.predict(X_test)

accuracy = accuracy_score(y_test, y_pred_svc)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.9736842105263158
```

[51]:
```
# Train with Standard scaled Data
svc_classifier2 = SVC()
svc_classifier2.fit(X_train_sc, y_train)
y_pred_svc_sc = svc_classifier2.predict(X_test_sc)
accuracy_score(y_test, y_pred_svc_sc)
```

[51]:
```
0.9649122807017544
```

[55]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
import numpy as np
```

```
# Assuming X_train, X_test, y_train, y_test are already defined

lr_classifier = LogisticRegression(random_state=51, solver='lbfgs')  # Using␣
 ↪default penalty ('l2')
lr_classifier.fit(X_train, y_train)
y_pred_lr = lr_classifier.predict(X_test)
accuracy = accuracy_score(y_test, y_pred_lr)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.956140350877193

C:\Users\ASUS\AppData\Local\Programs\Python\Python312\Lib\site-
packages\sklearn\linear_model\_logistic.py:469: 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
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

[57]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
import numpy as np

# Assuming X_train_sc, X_test_sc, y_train, y_test are already defined and␣
 ↪X_train_sc and X_test_sc are scaled versions of X_train and X_test␣
 ↪respectively

lr_classifier2 = LogisticRegression(random_state=51, solver='lbfgs')  # Using␣
 ↪default penalty ('l2')
lr_classifier2.fit(X_train_sc, y_train)
y_pred_lr_sc = lr_classifier2.predict(X_test_sc)  # Use lr_classifier2 instead␣
 ↪of lr_classifier
accuracy = accuracy_score(y_test, y_pred_lr_sc)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.9736842105263158
```

[58]:
```
#K-NEAREST NEIGHBOR CLASSIFIER


from sklearn.neighbors import KNeighborsClassifier
knn_classifier = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p␣
 ↪= 2)
```

```
knn_classifier.fit(X_train, y_train)
y_pred_knn = knn_classifier.predict(X_test)
accuracy_score(y_test, y_pred_knn)
```

[58]: 0.9385964912280702

[59]:
```
# Train with Standard scaled Data
knn_classifier2 = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p␣
↪= 2)
knn_classifier2.fit(X_train_sc, y_train)
y_pred_knn_sc = knn_classifier.predict(X_test_sc)
accuracy_score(y_test, y_pred_knn_sc)
```

```
C:\Users\ASUS\AppData\Local\Programs\Python\Python312\Lib\site-
packages\sklearn\base.py:493: UserWarning: X does not have valid feature names,
but KNeighborsClassifier was fitted with feature names
  warnings.warn(
```

[59]: 0.5789473684210527

[60]:
```
#NAIVFE BAYES CLASSIFIER
from sklearn.naive_bayes import GaussianNB
nb_classifier = GaussianNB()
nb_classifier.fit(X_train, y_train)
y_pred_nb = nb_classifier.predict(X_test)
accuracy_score(y_test, y_pred_nb)
```

[60]: 0.9473684210526315

[61]:
```
# Train with Standard scaled Data
nb_classifier2 = GaussianNB()
nb_classifier2.fit(X_train_sc, y_train)
y_pred_nb_sc = nb_classifier2.predict(X_test_sc)
accuracy_score(y_test, y_pred_nb_sc)
```

[61]: 0.9385964912280702

[62]:
```
#DECISION TREE CLASSIFIER
from sklearn.tree import DecisionTreeClassifier
dt_classifier = DecisionTreeClassifier(criterion = 'entropy', random_state = 51)
dt_classifier.fit(X_train, y_train)
y_pred_dt = dt_classifier.predict(X_test)
accuracy_score(y_test, y_pred_dt)
```

[62]: 0.9473684210526315

[63]:
```
# Train with Standard scaled Data
dt_classifier2 = DecisionTreeClassifier(criterion = 'entropy', random_state =␣
 ↪51)
dt_classifier2.fit(X_train_sc, y_train)
y_pred_dt_sc = dt_classifier.predict(X_test_sc)
accuracy_score(y_test, y_pred_dt_sc)
```

```
C:\Users\ASUS\AppData\Local\Programs\Python\Python312\Lib\site-
packages\sklearn\base.py:493: UserWarning: X does not have valid feature names,
but DecisionTreeClassifier was fitted with feature names
  warnings.warn(
```

[63]:
```
0.7543859649122807
```

[64]:
```
#RANDOM FOREST CLASSIFIER
from sklearn.ensemble import RandomForestClassifier
rf_classifier = RandomForestClassifier(n_estimators = 20, criterion =␣
 ↪'entropy', random_state = 51)
rf_classifier.fit(X_train, y_train)
y_pred_rf = rf_classifier.predict(X_test)
accuracy_score(y_test, y_pred_rf)
```

[64]:
```
0.9736842105263158
```

[65]:
```
# Train with Standard scaled Data
rf_classifier2 = RandomForestClassifier(n_estimators = 20, criterion =␣
 ↪'entropy', random_state = 51)
rf_classifier2.fit(X_train_sc, y_train)
y_pred_rf_sc = rf_classifier.predict(X_test_sc)
accuracy_score(y_test, y_pred_rf_sc)
```

```
C:\Users\ASUS\AppData\Local\Programs\Python\Python312\Lib\site-
packages\sklearn\base.py:493: UserWarning: X does not have valid feature names,
but RandomForestClassifier was fitted with feature names
  warnings.warn(
```

[65]:
```
0.7543859649122807
```

[66]:
```
#ADABOOST CLASSIFIER
from sklearn.ensemble import AdaBoostClassifier
adb_classifier = AdaBoostClassifier(DecisionTreeClassifier(criterion =␣
 ↪'entropy', random_state = 200),
                                    n_estimators=2000,
                                    learning_rate=0.1,
                                    algorithm='SAMME.R',
                                    random_state=1,)
adb_classifier.fit(X_train, y_train)
```

```
y_pred_adb = adb_classifier.predict(X_test)
accuracy_score(y_test, y_pred_adb)
```

```
C:\Users\ASUS\AppData\Local\Programs\Python\Python312\Lib\site-
packages\sklearn\ensemble\_weight_boosting.py:519: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
```

[66]: 0.9473684210526315

[67]:
```
# Train with Standard scaled Data
adb_classifier2 = AdaBoostClassifier(DecisionTreeClassifier(criterion =␣
 ↪'entropy', random_state = 200),
                                     n_estimators=2000,
                                     learning_rate=0.1,
                                     algorithm='SAMME.R',
                                     random_state=1,)
adb_classifier2.fit(X_train_sc, y_train)
y_pred_adb_sc = adb_classifier2.predict(X_test_sc)
accuracy_score(y_test, y_pred_adb_sc)
```

```
C:\Users\ASUS\AppData\Local\Programs\Python\Python312\Lib\site-
packages\sklearn\ensemble\_weight_boosting.py:519: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
```

[67]: 0.9473684210526315

[70]:
```
#XGBOOST CLASSIFIER
import xgboost
from xgboost import XGBClassifier
xgb_classifier = XGBClassifier()
xgb_classifier.fit(X_train, y_train)
y_pred_xgb = xgb_classifier.predict(X_test)
accuracy_score(y_test, y_pred_xgb)
```

[70]: 0.9824561403508771

[71]:
```
# Train with Standard scaled Data
xgb_classifier2 = XGBClassifier()
xgb_classifier2.fit(X_train_sc, y_train)
y_pred_xgb_sc = xgb_classifier2.predict(X_test_sc)
accuracy_score(y_test, y_pred_xgb_sc)
```

[71]: 0.9824561403508771

[72]:
```
#XGBoost Parameter Tuning Randomized Search
params={
 "learning_rate"    : [0.05, 0.10, 0.15, 0.20, 0.25, 0.30 ] ,
 "max_depth"        : [ 3, 4, 5, 6, 8, 10, 12, 15],
 "min_child_weight" : [ 1, 3, 5, 7 ],
 "gamma"            : [ 0.0, 0.1, 0.2 , 0.3, 0.4 ],
 "colsample_bytree" : [ 0.3, 0.4, 0.5 , 0.7 ]
}
```

[73]:
```
# Randomized Search
from sklearn.model_selection import RandomizedSearchCV
random_search = RandomizedSearchCV(xgb_classifier, param_distributions=params,␣
 ↪scoring= 'roc_auc', n_jobs= -1, verbose= 3)
random_search.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
```

[73]:
```
RandomizedSearchCV(estimator=XGBClassifier(base_score=None, booster=None,
                                           callbacks=None,
                                           colsample_bylevel=None,
                                           colsample_bynode=None,
                                           colsample_bytree=None, device=None,
                                           early_stopping_rounds=None,
                                           enable_categorical=False,
                                           eval_metric=None, feature_types=None,
                                           gamma=None, grow_policy=None,
                                           importance_type=None,
                                           interaction_constraints=None,
                                           learning_rate=None…
                                           monotone_constraints=None,
                                           multi_strategy=None,
                                           n_estimators=None, n_jobs=None,
                                           num_parallel_tree=None,
                                           random_state=None, …),
                   n_jobs=-1,
                   param_distributions={'colsample_bytree': [0.3, 0.4, 0.5,
                                                             0.7],
                                        'gamma': [0.0, 0.1, 0.2, 0.3, 0.4],
                                        'learning_rate': [0.05, 0.1, 0.15, 0.2,
                                                          0.25, 0.3],
                                        'max_depth': [3, 4, 5, 6, 8, 10, 12,
                                                      15],
                                        'min_child_weight': [1, 3, 5, 7]},
                   scoring='roc_auc', verbose=3)
```

[74]:
```
{'min_child_weight': 1,
 'max_depth': 3,
```

```
 'learning_rate': 0.3,
 'gamma': 0.4,
 'colsample_bytree': 0.3}
```

[74]:
```
{'min_child_weight': 1,
 'max_depth': 3,
 'learning_rate': 0.3,
 'gamma': 0.4,
 'colsample_bytree': 0.3}
```

[75]:
```
random_search.best_estimator_
```

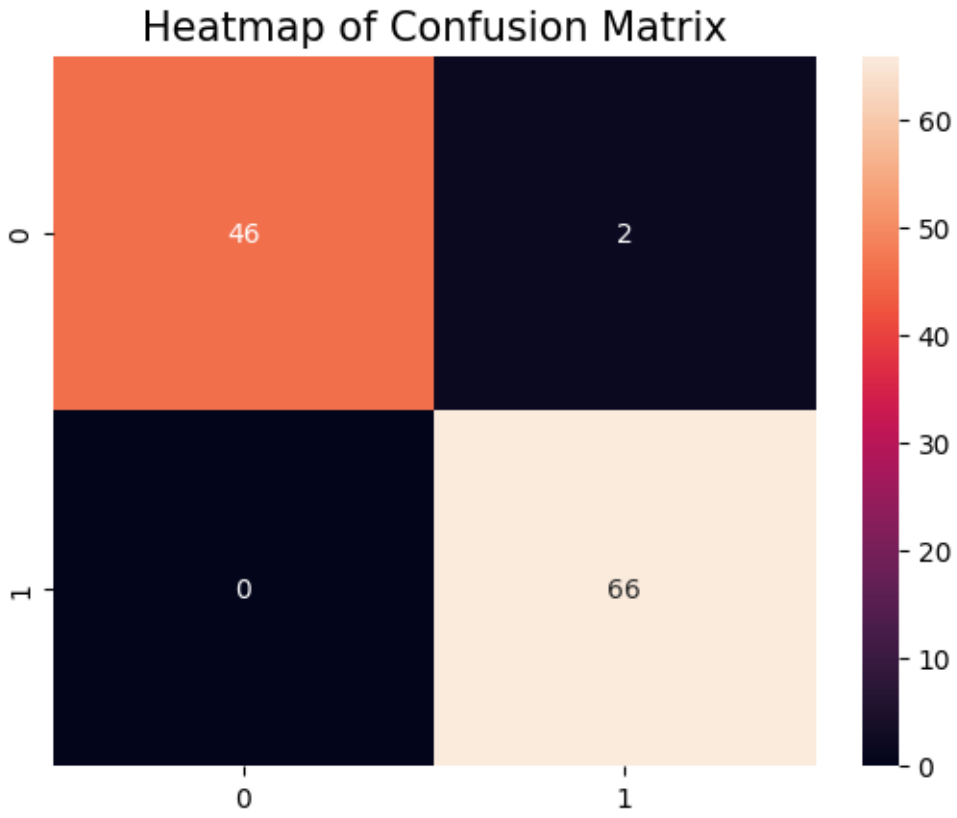
[75]:
```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=0.5, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=0.0, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=0.1, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=10, max_leaves=None,
              min_child_weight=3, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=None, n_jobs=None,
              num_parallel_tree=None, random_state=None, …)
```

[76]:
```
# training XGBoost classifier with best parameters
xgb_classifier_pt = XGBClassifier(base_score=0.5, booster='gbtree',␣
 ↪colsample_bylevel=1,
       colsample_bynode=1, colsample_bytree=0.4, gamma=0.2,
       learning_rate=0.1, max_delta_step=0, max_depth=15,
       min_child_weight=1, missing=None, n_estimators=100, n_jobs=1,
       nthread=None, objective='binary:logistic', random_state=0,
       reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
       silent=None, subsample=1, verbosity=1)
```

[78]:
```
import xgboost as xgb
from sklearn.metrics import accuracy_score

# Assuming X_train, X_test, y_train, and y_test are already defined

xgb_classifier_pt = xgb.XGBClassifier()  # Assuming you've initialized your␣
 ↪XGBoost classifier
xgb_classifier_pt.fit(X_train, y_train)
y_pred_xgb_pt = xgb_classifier_pt.predict(X_test)
accuracy = accuracy_score(y_test, y_pred_xgb_pt)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.9824561403508771
```

[80]:
```
#CONFUSION MATRIX
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

cm = confusion_matrix(y_test, y_pred_xgb_pt)
plt.title('Heatmap of Confusion Matrix', fontsize=15)
sns.heatmap(cm, annot=True)
plt.show()
```



[82]:
```
#CLASSIFICATION REPORT OF MODEL
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred_xgb_pt))
```

```
              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
```

[85]:
```
#Cross-Validation of the ML Model
import xgboost as xgb


xgb_model_pt2 = xgb.XGBClassifier()

# Now you can perform cross-validation
cross_validation = cross_val_score(estimator=xgb_model_pt2, X=X_train_sc,␣
 ↪y=y_train, cv=10)
print("Cross validation of XGBoost model =", cross_validation)
print("Cross validation of XGBoost model (in mean) =", cross_validation.mean())
```

```
Cross validation of XGBoost model = [1.         0.97826087 0.97826087 1.
0.91304348 1.
 1.         1.         0.97777778 0.91111111]
Cross validation of XGBoost model (in mean) = 0.9758454106280194
```

[86]:
```
#SAVING THE MACHINE LEARNING MODEL
## Pickle
import pickle

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

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

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

# confusion matrix
print('Confusion matrix of XGBoost model: \n',confusion_matrix(y_test,␣
 ↪y_pred),'\n')

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

```
Confusion matrix of XGBoost model:
 [[46  2]
 [ 0 66]]

Accuracy of XGBoost model =  0.9824561403508771
```

[ ]: