Jupyter Classification Problem Last Checkpoint: 13 minutes ago

File Edit View Run Kernel Settings Help

Trusted

Code

JupyterLab Python 3 (ipykernel)

[1]:
```
import pandas as pd
import numpy as np
import sklearn as sklearn
```

1. Loading and Preprocessing (2 marks)et.

- Load the breast cancer dataset from sklearn.
- Preprocess the data to handle any missing values and perform necessary feature scaling.
- Explain the preprocessing steps you performed and justify why they are necessary for this datas

[4]:
```
from sklearn.datasets import load_breast_cancer
data=load_breast_cancer()
data
```

[4]:
```
{'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, 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, 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]),
 '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 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    - smo
othness (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, stan
dard 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 featu
res.  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\nperimet
er (mean):                     43.79  188.5\narea (mean):                          143.5  2501.0\nsmoothness (mean):                    0.053  0.163\ncompa
ctness (mean):                  0.019  0.345\nconcavity (mean):                     0.0    0.427\nconcave points (mean):               0.0    0.201\nsymm
etry (mean):                    0.106  0.304\nfractal dimension (mean):             0.05   0.097\nradius (standard error):              0.112  2.873\ntex
ture (standard error):          0.36   4.885\nperimeter (standard error):           0.757  21.98\narea (standard error):                6.802  542.2\nsm
oothness (standard error):      0.002  0.031\ncompactness (standard error):         0.002  0.135\nconcavity (standard error):           0.0    0.396\nc
oncave points (standard error):   0.0    0.053\nsymmetry (standard error):            0.008  0.079\nfractal dimension (standard error):   0.001  0.03\nr
adius (worst):                  7.93   36.04\ntexture (worst):                      12.02  49.54\nperimeter (worst):                    50.41  251.2\n
area (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  0.66
4\nfractal dimension (worst):       0.055  0.208\n===================================== ====== ======\n\n:Missing Attribute Values: None\n\n:Class Dis
tribution: 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: Novembe
r, 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 o
f a fine needle\naspirate (FNA) of a breast mass.  They describe\ncharacteristics of the cell nuclei present in the image.\n\nSeparating plane described ab
ove was obtained using\nMultisurface Method-Tree (MSM-T) [K. P. Bennett, "Decision Tree\nConstruction Via Linear Programming." Proceedings of the 4th\nMidw
est Artificial Intelligence and Cognitive Science Society,\npp. 97-101, 1992], a classification method which uses linear\nprogramming to construct a decisi
on 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 Di
scrimination of Two Linearly Inseparable Sets",\nOptimization Methods and Software 1, 1992, 23-34].\n\nThis database is also available through the UW CS ft
p 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. W
olberg and O.L. Mangasarian. Nuclear feature extraction\n  for breast tumor diagnosis. IS&T/SPIE 1993 International Symposium on\n  Electronic Imaging: Sci
ence 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  p
rognosis 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. Machin
e 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'}
```

[5]:
```
df=pd.DataFrame(data.data,columns=data.feature_names)
df["target"]=data.target
df
```

[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
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
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
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
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
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
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
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
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
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
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
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

569 rows × 31 columns

[6]:
```
# To get the the basic information about the dataset like no: of rows,no: of columns, datatypes etc.
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    int32  
dtypes: float64(30), int32(1)
memory usage: 135.7 KB
```

[7]:
```
# To find the no of duplicated values
df.duplicated().sum()
```

[7]: 0

This dataset do not have any duplicated entries

This dataset do not have any duplicated entries.

[9]:
```
# To check is there any null values or missing values in the dataset.
df.isnull().sum()
```

[9]:
```
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
```

This dataset set do not contain any missing values.

[11]:
```
# To analyse the statistical measures of the dataset.
df.describe()
```

[11]:

	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 are
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	569.00000
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	880.58312
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	569.35699
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	185.20000
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	515.30000
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	686.50000
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	1084.00000
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	4254.00000

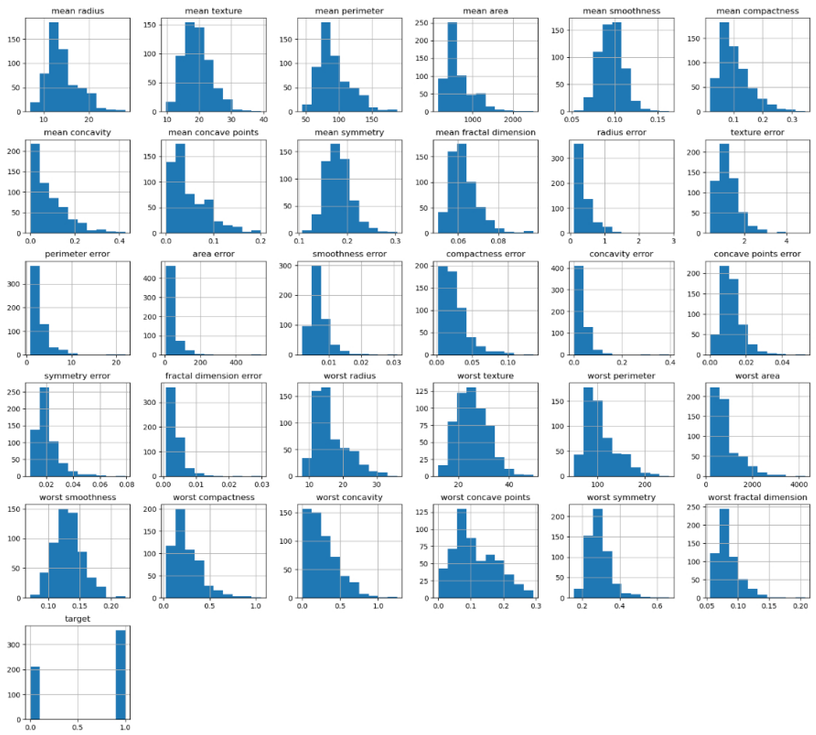
8 rows × 31 columns

In some of the columns, there is major difference between the mean and median values. This may be the indication of outliers in the dataset.

[13]:
```
# To check the presence of outliers visually
import seaborn as sns
import matplotlib.pyplot as plt
numericals=df.select_dtypes("number")
for column in numericals:
    numericals.boxplot(column)
    plt.show()
```

we can see few outliers in the boxplots

[15]:
```
df.hist(figsize=(20,18))
plt.show()
```



From the histogram, we can see that,the data does not have a uniform bell shape.This indicates the presence of outliers.

[17]:
```
# To check the skewness of the data
skewness=df.skew()
skewness
```

[17]:
```
mean radius                0.942380
mean texture               0.650450
mean perimeter             0.990650
mean area                  1.645732
mean smoothness            0.456324
mean compactness           1.190123
mean concavity             1.401180
mean concave points        1.171180
mean symmetry              0.725609
mean fractal dimension     1.304489
radius error               3.088612
texture error              1.646444
perimeter error            3.443615
area error                 5.447186
smoothness error           2.314450
compactness error          1.902221
concavity error            5.110463
concave points error       1.444678
symmetry error             2.195133
fractal dimension error    3.923969
worst radius               1.103115
worst texture              0.498321
worst perimeter            1.128164
```

```
worst area               1.859373
worst smoothness         0.415426
worst compactness        1.473555
worst concavity          1.150237
worst concave points     0.492616
worst symmetry           1.433928
worst fractal dimension  1.662579
target                  -0.528461
dtype: float64
```

From the skewness result, we can see most of the columns are possitively skewed.

[19]:
```
# to find the outliers using IQR method.
def find_outliers_iqr(df):
    outliers= {}
    numericals=df.select_dtypes(include=["number"])
    for column in numericals:
        Q1= numericals[column].quantile(0.25)
        Q3= numericals[column].quantile(0.75)
        IQR=Q3-Q1
        lower_whisker=Q1 - 1.5*IQR
        upper_whisker=Q3 + 1.5*IQR
        outliers[column] = df[(numericals[column] < lower_whisker) | (numericals[column] > upper_whisker)]
    return outliers
outliers_dict=find_outliers_iqr(df)
for column,outliers in outliers_dict.items():
    print("outliers in",column,":",outliers)
```

```
outliers in mean radius :      mean radius  mean texture  mean perimeter  mean area  mean smoothness  \
82          25.22         24.91           171.5     1878.0          0.10630
108         22.27         19.67           152.8     1509.0          0.13260
122         24.25         20.20           166.2     1761.0          0.14470
164         23.27         22.04           152.1     1686.0          0.08439
180         27.22         21.87           182.1     2250.0          0.10940
202         23.29         26.67           158.9     1685.0          0.11410
212         28.11         18.47           188.5     2499.0          0.11420
236         23.21         26.97           153.5     1670.0          0.09509
339         23.51         24.27           155.1     1747.0          0.10690
352         25.73         17.46           174.2     2010.0          0.11490
369         22.01         21.90           147.2     1482.0          0.10630
461         27.42         26.27           186.9     2501.0          0.10840
503         23.09         19.83           152.1     1682.0          0.09342
521         24.63         21.60           165.5     1841.0          0.10300

     mean compactness  mean concavity  mean concave points  mean symmetry  \
82             0.2665          0.3339              0.18450         0.1829
```

[20]:
```
#To capp the outliers

def cap_outliers(df):
    df_capped=df.copy()
    numericals=df.select_dtypes(include=["number"])

    for column in numericals.columns:
        Q1 = numericals[column].quantile(0.25)
        Q3 = numericals[column].quantile(0.75)
        IQR = Q3 - Q1
        lower_whisker = Q1 - 1.5 * IQR
        upper_whisker = Q3 + 1.5 * IQR
        df_capped[column]=df[column].apply(lambda x:lower_whisker if x < lower_whisker else upper_whisker if x > upper_whisker else x )

    return df_capped
df_capped=cap_outliers(df)
df_capped
```

[20]:

	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
0	17.99	10.38	122.80	1001.0	0.118400	0.22862	0.28241	0.14710	0.2419	0.07871	...	17.33	184.60	1937.05	0.16220	0.62695
1	20.57	17.77	132.90	1326.0	0.084740	0.07864	0.08690	0.07017	0.1812	0.05667	...	23.41	158.80	1937.05	0.12380	0.18660
2	19.69	21.25	130.00	1203.0	0.109600	0.15990	0.19740	0.12790	0.2069	0.05999	...	25.53	152.50	1709.00	0.14440	0.42450
3	11.42	20.38	77.58	386.1	0.133695	0.22862	0.24140	0.10520	0.2464	0.07875	...	26.50	98.87	567.70	0.19010	0.62695
4	20.29	14.34	135.10	1297.0	0.100300	0.13280	0.19800	0.10430	0.1809	0.05883	...	16.67	152.20	1575.00	0.13740	0.20500
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
564	21.56	22.39	142.00	1326.3	0.111000	0.11590	0.24390	0.13890	0.1726	0.05623	...	26.40	166.10	1937.05	0.14100	0.21130
565	20.13	28.25	131.20	1261.0	0.097800	0.10340	0.14400	0.09791	0.1752	0.05533	...	38.25	155.00	1731.00	0.11660	0.19220
566	16.60	28.08	108.30	858.1	0.084550	0.10230	0.09251	0.05302	0.1590	0.05648	...	34.12	126.70	1124.00	0.11390	0.30940
567	20.60	29.33	140.10	1265.0	0.117800	0.22862	0.28241	0.15200	0.2397	0.07016	...	39.42	184.60	1821.00	0.16500	0.62695
568	7.76	24.54	47.92	181.0	0.057975	0.04362	0.00000	0.00000	0.1587	0.05884	...	30.37	59.16	268.60	0.08996	0.06444

569 rows × 31 columns

[21]:
```
# checking the presence of outliers after capping the outliers.
numericals=df_capped.select_dtypes("number")
for column in numericals:
    numericals.boxplot(column)
    plt.show()
```

outliers capped successfully

[23]:
```
#checking the skewness of the data after capping the outliers
skewness=df_capped.skew()
skewness
```

[23]:
```
mean radius                0.655953
mean texture               0.449700
mean perimeter             0.701081
mean area                  0.922884
mean smoothness            0.257712
mean compactness           0.826755
mean concavity             1.023859
mean concave points        1.004049
mean symmetry              0.403621
mean fractal dimension     0.682430
radius error               1.025031
texture error              0.740987
perimeter error            1.034389
area error                 1.130940
smoothness error           0.780923
compactness error          0.990285
concavity error            0.916740
concave points error       0.539571
symmetry error             0.869297
fractal dimension error    0.979344
worst radius               0.849779
worst texture              0.386858
worst perimeter            0.874870
worst area                 1.048970
worst smoothness           0.247199
worst compactness          0.915295
worst concavity            0.809174
worst concave points       0.492616
worst symmetry             0.521772
worst fractal dimension    0.831581
target                    -0.528461
dtype: float64
```

since the data is still skewed. we have to apply transformation technique.

[25]:
```
#To transform data using Square root transformation
df_transformed=df_capped.copy()
numericals=df_capped.select_dtypes(include=["number"])
for column in numericals.columns:
    df_transformed[column]=np.sqrt(numericals[column])
df_transformed.skew()
```

[25]:
```
mean radius                0.423146
mean texture               0.180940
mean perimeter             0.456977
mean area                  0.553411
mean smoothness            0.065613
mean compactness           0.372436
mean concavity             0.214135
mean concave points        0.177938
mean symmetry              0.201485
mean fractal dimension     0.566282
radius error               0.656397
texture error              0.297205
perimeter error            0.651761
area error                 0.798276
smoothness error           0.415840
compactness error          0.463838
concavity error           -0.095279
concave points error      -0.822304
symmetry error             0.565571
fractal dimension error    0.536136
worst radius               0.612512
worst texture              0.109939
```

```
worst perimeter          0.620939
worst area               0.693672
worst smoothness         0.018433
worst compactness        0.344224
worst concavity         -0.092976
worst concave points    -0.443414
worst symmetry           0.243989
worst fractal dimension  0.635767
target                  -0.528461
dtype: float64
```

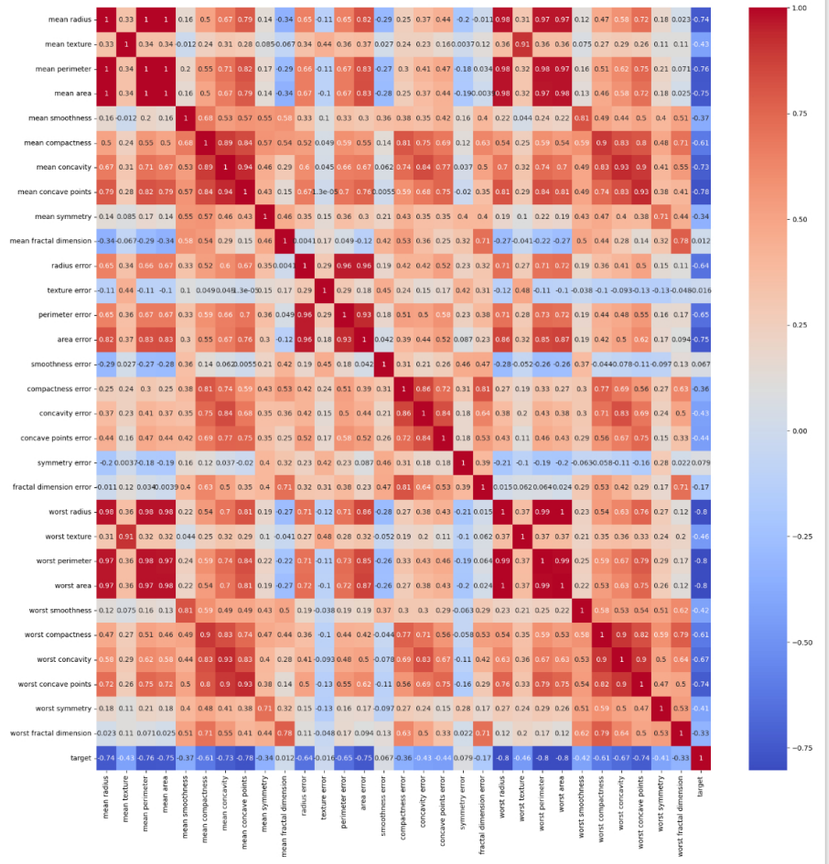
[26]:
```
corr=df_transformed.corr()
corr
```

[26]:

	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	smoo
mean radius	1.000000	0.331432	0.997845	0.997693	0.164113	0.499859	0.670608	0.791855	0.137927	-0.337797	...	0.308940	0.970047	0.971626	0.
mean texture	0.331432	1.000000	0.337959	0.339155	-0.012197	0.242016	0.307556	0.276617	0.084836	-0.066660	...	0.914445	0.364612	0.360100	0.
mean perimeter	0.997845	0.337959	1.000000	0.995357	0.201304	0.549985	0.708967	0.820942	0.171528	-0.286349	...	0.315611	0.976236	0.972685	0.
mean area	0.997693	0.339155	0.995357	1.000000	0.159363	0.495928	0.671410	0.789275	0.140161	-0.335419	...	0.315553	0.971082	0.977411	0.
mean smoothness	0.164113	-0.012197	0.201304	0.159363	1.000000	0.677998	0.528306	0.570484	0.552303	0.582067	...	0.044194	0.244414	0.216397	0.
mean compactness	0.499859	0.242016	0.549985	0.495928	0.677998	1.000000	0.892266	0.840888	0.573652	0.542063	...	0.251987	0.594228	0.539349	0.
mean concavity	0.670608	0.307556	0.708967	0.671410	0.528306	0.892266	1.000000	0.937028	0.456286	0.290317	...	0.319878	0.738102	0.699889	0.
mean concave points	0.791855	0.276617	0.820942	0.789275	0.570484	0.840888	0.937028	1.000000	0.434842	0.152554	...	0.288815	0.836569	0.808324	0.
mean symmetry	0.137927	0.084836	0.171528	0.140161	0.552303	0.573652	0.456286	0.434842	1.000000	0.461381	...	0.104028	0.215173	0.188338	0.
mean fractal dimension	-0.337797	-0.066660	-0.286349	-0.335419	0.582067	0.542063	0.290317	0.152554	0.461381	1.000000	...	-0.040730	-0.222877	-0.268188	0.
radius error	0.650093	0.341369	0.662758	0.668330	0.332057	0.524935	0.601269	0.670300	0.348775	0.004085	...	0.268726	0.708238	0.720982	0.
texture error	-0.114931	0.442316	-0.105197	-0.103344	0.100393	0.048895	0.044585	-0.000013	0.147348	0.173819	...	0.481631	-0.107629	-0.104804	-0.
perimeter error	0.654374	0.358651	0.673220	0.670477	0.331950	0.589063	0.656472	0.703249	0.355764	0.049297	...	0.283149	0.725480	0.715727	0.
area error	0.817185	0.374078	0.825117	0.833179	0.296793	0.548491	0.670342	0.760327	0.299566	-0.115892	...	0.318443	0.854004	0.870812	0.
smoothness error	-0.293128	0.026610	-0.272146	-0.277784	0.357722	0.139904	0.062222	0.005474	0.211585	0.416008	...	-0.052307	-0.262318	-0.259658	0.
compactness error	0.252011	0.236295	0.299988	0.253473	0.379316	0.806996	0.738731	0.585190	0.432602	0.532937	...	0.187397	0.325628	0.270436	0.
concavity error	0.373505	0.234160	0.412939	0.373747	0.349382	0.749162	0.844474	0.683889	0.353839	0.358901	...	0.204673	0.426419	0.379887	0.
concave points error	0.439212	0.162450	0.470339	0.435049	0.423215	0.689283	0.766290	0.754564	0.347512	0.252411	...	0.113717	0.463326	0.425902	0.
symmetry error	-0.199880	0.003730	-0.181274	-0.190674	0.155318	0.124085	0.037043	-0.020432	0.400831	0.319625	...	-0.104036	-0.191456	-0.201956	-0.
fractal dimension error	-0.010952	0.122131	0.034178	-0.003890	0.395132	0.628846	0.500698	0.351937	0.400011	0.714518	...	0.061652	0.063868	0.024365	0.
worst radius	0.975683	0.356884	0.976921	0.977434	0.220503	0.544115	0.700792	0.812718	0.185462	-0.272199	...	0.365005	0.994107	0.996804	0.
worst texture	0.308940	0.914445	0.315611	0.315553	0.044194	0.251987	0.319878	0.288815	0.104028	-0.040730	...	1.000000	0.372883	0.365790	0.
worst perimeter	0.970047	0.364612	0.976236	0.971082	0.244414	0.594228	0.738102	0.836569	0.215173	-0.222877	...	0.372883	1.000000	0.990409	0.
worst area	0.971626	0.360100	0.972685	0.977411	0.216397	0.539349	0.699889	0.808324	0.188338	-0.268188	...	0.365790	0.990409	1.000000	0.
worst smoothness	0.124388	0.074509	0.156180	0.126639	0.814667	0.588735	0.486732	0.488195	0.432076	0.499317	...	0.212161	0.245906	0.224990	1.
worst compactness	0.469404	0.270758	0.513655	0.464452	0.493891	0.897525	0.832663	0.743051	0.465134	0.440353	...	0.347217	0.589572	0.530336	0.
worst concavity	0.582224	0.288655	0.618254	0.579542	0.438550	0.830277	0.932962	0.829859	0.396127	0.275693	...	0.359639	0.672296	0.626435	0.
worst concave points	0.724294	0.258143	0.752009	0.717043	0.496673	0.804585	0.900519	0.934142	0.382132	0.141389	...	0.329513	0.791873	0.753972	0.
worst symmetry	0.181983	0.109775	0.206531	0.181845	0.396970	0.484258	0.405225	0.376211	0.714199	0.316293	...	0.241773	0.291882	0.263467	0.
worst fractal dimension	0.023178	0.110732	0.070696	0.024748	0.505534	0.707800	0.547706	0.411244	0.444736	0.778991	...	0.200718	0.166815	0.117938	0.
target	-0.740794	-0.425978	-0.755799	-0.751596	-0.368886	-0.609640	-0.727489	-0.775461	-0.335266	0.011787	...	-0.461307	-0.803721	-0.803976	-0.

31 rows × 31 columns

[27]:
```
plt.figure(figsize=(20,20))
sns.heatmap(corr,annot=True,cmap="coolwarm")
plt.show()
```



some features are highly correlated each other such as worst radius, mean radius.mean perimeter, worst perimeterets etc. most of the features have high possitive correlation with the target.

[29]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

[30]:
```
X= df_transformed.drop("target",axis=1)
X
```

[30]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst radius	worst texture	worst perimeter	worst area	smoot

0	4.241462	3.221801	11.081516	31.638584	0.344093	0.478142	0.531423	0.383536	0.491833	0.280553	...	5.037857	4.162932	13.586758	44.011930	0.4
1	4.535416	4.215448	11.528226	36.414283	0.291101	0.280428	0.294788	0.264896	0.425676	0.238055	...	4.999000	4.838388	12.601587	44.011930	0.3
2	4.437342	4.609772	11.401754	34.684290	0.331059	0.399875	0.444297	0.357631	0.454863	0.244929	...	4.854894	5.052722	12.349089	41.340053	0.3
3	3.379349	4.514421	8.807951	19.649427	0.365643	0.478142	0.491325	0.324345	0.496387	0.280624	...	3.861347	5.147815	9.943339	23.826456	0.4
4	4.504442	3.786819	11.623253	36.013886	0.316702	0.364417	0.444972	0.322955	0.425323	0.242549	...	4.747631	4.082891	12.336936	39.686270	0.3
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
564	4.643275	4.731807	11.916375	36.418402	0.333167	0.340441	0.493862	0.372693	0.415452	0.237129	...	5.044799	5.138093	12.887979	44.011930	0.3
565	4.486647	5.315073	11.454257	35.510562	0.312730	0.321559	0.379473	0.312906	0.418569	0.235223	...	4.867237	6.184658	12.449900	41.605288	0.3
566	4.074310	5.299057	10.406729	29.293344	0.290775	0.319844	0.304155	0.230261	0.398748	0.237655	...	4.356604	5.841233	11.256109	33.526109	0.3
567	4.538722	5.415718	11.836385	35.566838	0.343220	0.478142	0.531423	0.389872	0.489592	0.264877	...	5.073460	6.278535	13.586758	42.673177	0.4
568	2.785678	4.953786	6.922427	13.453624	0.240780	0.208854	0.000000	0.000000	0.398372	0.242570	...	3.075061	5.510898	7.691554	16.389021	0.2

569 rows × 30 columns

[31]:
```
y=df_transformed["target"]
y
```

[31]:
```
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
```

[32]:
```
# to scale the features using standard scaler
std_scaler=StandardScaler()
X_scaled=std_scaler.fit_transform(X)
X_scaled
```

[32]:
```
array([[ 1.18494752, -2.40759365,  1.34848788, ...,  1.81800772,
         2.28121616,  2.12913984],
       [ 1.85886273, -0.31014107,  1.72824063, ...,  1.03130129,
        -0.19274764,  0.40559561],
       [ 1.63401819,  0.52222373,  1.62072553, ...,  1.60964298,
         1.34934646,  0.31570159],
       ...,
       [ 0.80173661,  1.9772083 ,  0.77484417, ...,  0.51822124,
        -1.26765623, -0.28890213],
       [ 1.86644225,  2.22346439,  1.99020916, ...,  1.81436579,
         2.11788291,  2.34836511],
       [-2.15256321,  1.24839051, -2.18719554, ..., -3.01301387,
         0.03677695, -0.82060183]])
```

[33]:
```
X_train,X_test,y_train,y_test=train_test_split(X_scaled,y,test_size=0.2,random_state=42)
```

[34]:
```
X_train.shape,X_test.shape
```

[34]:
```
((455, 30), (114, 30))
```

[35]:
```
y_train.shape, y_test.shape
```

[35]:
```
((455,), (114,))
```

2. Classification Algorithm Implementation (5 marks)

- Implement the following five classification algorithms:

1. Logistic Regression
2. Decision Tree Classifier
3. Random Forest Classifier
4. Support Vector Machine (SVM)
5. k-Nearest Neighbors (k-NN)

- For each algorithm, provide a brief description of how it works and why it might be suitable for this dataset.

[40]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
```

Logistic Regression

How it Work :

- Logistic Regression is a statistical model that uses the logistic (sigmoid) function to predict the probability of a binary outcome (0 or 1).
- It calculates a weighted sum of the input features and applies the sigmoid function to obtain probabilities.

Why it's suitable:

- Works well for binary classification problems like out dataset.
- Outputs probabilities, which can help in medical diagnosis.

[43]:
```
logreg=LogisticRegression()
logreg.fit(X_train,y_train)
```

[43]:
```
LogisticRegression()
```

[44]:
```
logreg_y_pred=logreg.predict(X_test)
logreg_y_pred
```

[44]:
```
array([1., 0., 0., 1., 1., 0., 0., 0., 1., 1., 1., 0., 1., 0., 1., 0.,
       1., 1., 0., 1., 1., 0., 1., 1., 1., 1., 1., 1., 1., 1., 1., 0.,
       1., 1., 0., 1., 0., 1., 1., 1., 1., 1., 1., 1., 1., 0., 0., 0.,
       0., 1., 1., 1., 1., 0., 0., 1., 1., 0., 0., 1., 1., 1., 0., 0.,
       1., 1., 0., 0., 1., 0., 1., 1., 0., 1., 1., 0., 1., 0., 0., 0.,
       0., 0., 0., 1., 1., 1., 1., 1., 1., 1., 0., 0., 1., 0., 0., 1.,
       0., 0., 1., 1., 1., 0., 1., 0., 1., 0., 0.])
```

Decision Tree Classifier

How it Work :

- A Decision Tree is a supervised learning algorithm used for classification and regression tasks.
- The structure resembles a tree, with nodes representing decisions, branches representing choices, and leaves representing outcomes or predictions.

Why it's suitable:

- Easy to interpret and visualize, making it useful in medical applications.
- Handles non-linearly separable data.
- Can automatically capture interactions between features.

[47]:
```
dtcl=DecisionTreeClassifier()
dtcl.fit(X_train,y_train)
```

[47]:
```
DecisionTreeClassifier()
```

[48]:
```
dtcl_y_pred=dtcl.predict(X_test)
dtcl_y_pred
```

[48]:
```
array([1., 0., 0., 1., 1., 0., 0., 0., 1., 1., 0., 1., 0., 1., 0., 1.,
       1., 1., 0., 1., 1., 0., 1., 1., 1., 1., 1., 1., 1., 1., 1., 0.,
       1., 1., 0., 1., 0., 1., 1., 1., 1., 1., 1., 1., 1., 0., 0., 0.,
       0., 1., 1., 1., 1., 0., 0., 1., 1., 0., 0., 1., 1., 1., 0., 0.,
       1., 1., 0., 0., 1., 0., 1., 1., 1., 1., 1., 0., 1., 1., 0., 0.,
       0., 0., 0., 1., 1., 1., 1., 1., 1., 1., 0., 0., 1., 0., 0., 1.,
       0., 0., 1., 1., 1., 0., 0., 1., 0., 1., 1.])
```

Random Forest Classifier

How it Works:

- Random Forest is an ensemble learning method that builds multiple Decision Trees and averages their predictions to improve accuracy and reduce overfitting.
- Each tree is built on a random subset of data and features.

Why it's suitable:

- More accurate and robust than a single Decision Tree.
- Less prone to overfitting due to averaging across trees.
- Can handle large feature sets and complex relationships in data.

[51]:
```
rfcl=RandomForestClassifier()
rfcl.fit(X_train,y_train)
```

[51]:
```
RandomForestClassifier()
```

[52]:
```
rfcl_y_pred=rfcl.predict(X_test)
rfcl_y_pred
```

[52]:
```
array([1., 0., 0., 1., 1., 0., 0., 0., 1., 1., 0., 1., 0., 1., 0., 1.,
       1., 1., 0., 1., 1., 0., 1., 1., 1., 1., 1., 1., 0., 1., 1., 1.,
```

```
       1., 1., 0., 1., 0., 1., 1., 0., 1., 1., 1., 1., 1., 1., 1., 0.,
       0., 1., 1., 1., 1., 1., 0., 0., 1., 1., 0., 0., 1., 1., 1., 0.,
       1., 1., 0., 0., 1., 0., 1., 1., 1., 1., 1., 0., 1., 1., 0., 0.,
       0., 0., 0., 1., 1., 1., 1., 1., 1., 1., 1., 0., 0., 1., 0., 0.,
       0., 0., 1., 1., 1., 0., 1., 1., 0., 1., 1., 0., 0., 0., 1., 1.,
       0., 0., 1., 1., 1., 0., 1., 1., 0., 1., 1., 0.])
```

Support Vector Machine (SVM)

How it Works:

- SVC uses a set of labeled training examples to find a decision boundary that separates the data points into different classes.
- The decision boundary is represented as a linear function, and the goal is to find the boundary that maximizes the separation between the classes.

Why it's suitable:

- Works well when features are properly scaled.
- Performs well for high-dimensional data, like this dataset with 30 features.
- SVC is commonly used in image recognition, text classification, and medical diagnosis.

[55]:
```
svc=SVC()
svc.fit(X_train,y_train)
```

[55]:
```
SVC()
```

[56]:
```
svc_y_pred=svc.predict(X_test)
svc_y_pred
```

[56]:
```
array([1., 0., 0., 1., 1., 0., 0., 0., 1., 1., 1., 0., 1., 0., 1., 0.,
       1., 1., 0., 1., 1., 0., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 0., 1., 0., 1., 0., 1., 1., 1., 1., 1., 1., 1., 1., 0.,
       0., 1., 1., 1., 1., 1., 0., 0., 1., 1., 0., 0., 1., 1., 1., 0.,
       1., 1., 0., 0., 1., 0., 1., 1., 1., 1., 1., 0., 1., 0., 0., 0.,
       0., 0., 0., 1., 1., 1., 0., 1., 1., 1., 1., 0., 0., 1., 0., 0.,
       0., 0., 1., 1., 1., 0., 1., 1., 0., 1., 1., 0.])
```

k-Nearest Neighbors (k-NN)

How it works:

- A non-parametric, instance-based algorithm that classifies a new point based on the majority class among its k closest neighbors.
- Uses distance metrics (e.g., Euclidean distance) to determine proximity.

Why it's suitable:

- it is Simple.
- Can handle non-linear decision boundaries.
- Sensitive to feature scaling and computationally expensive for large datasets.

[59]:
```
knn=KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train,y_train)
```

[59]:
```
KNeighborsClassifier()
```

[60]:
```
knn_y_pred=knn.predict(X_test)
knn_y_pred
```

[60]:
```
array([1., 0., 0., 1., 1., 0., 0., 0., 1., 1., 0., 1., 1., 1., 0., 1.,
       1., 1., 0., 1., 1., 0., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 0., 1., 0., 1., 1., 0., 1., 1., 1., 1., 1., 1., 1., 0.,
       0., 0., 1., 1., 1., 0., 0., 0., 1., 1., 0., 1., 1., 0., 0., 0.,
       1., 1., 0., 0., 1., 0., 1., 1., 1., 1., 1., 0., 1., 0., 0., 0.,
       0., 0., 0., 1., 1., 1., 1., 1., 1., 1., 0., 0., 1., 0., 0., 1.,
       0., 0., 1., 1., 1., 0., 1., 1., 0., 1., 1., 0.])
```

3. Model Comparison (2 marks)

- Compare the performance of the five classification algorithms.
- Which algorithm performed the best and which one performed the worst?

[63]:
```
from sklearn.metrics import accuracy_score,f1_score,precision_score,recall_score
```

[64]:
```
predictions={
    "Logistic Regression":logreg_y_pred ,
    "Decision Tree Classifier":dtcl_y_pred,
    "Random Forest Classifier":rfcl_y_pred,
    "Support Vector Machine (SVM)":svc_y_pred,
    "k-Nearest Neighbors (k-NN)":knn_y_pred
}

result={
    "Model":[],
    "Accuracy Score":[],
    "F1 Score":[],
    "Precision Score":[],
    "Recall Score":[]
}

# Compute metrics for each model

for model_name,y_pred in predictions.items():
    result["Model"].append(model_name)
    result["Accuracy Score"].append(accuracy_score(y_test,y_pred))
    result["F1 Score"].append(f1_score(y_test,y_pred))
    result["Precision Score"].append(precision_score(y_test,y_pred))
    result["Recall Score"].append(recall_score(y_test,y_pred))

print(result)
```

```
{'Model': ['Logistic Regression', 'Decision Tree Classifier', 'Random Forest Classifier', 'Support Vector Machine (SVM)', 'k-Nearest Neighbors (k-NN)'], 'Accuracy Score': [0.9824561403508771, 0.9473684210526315, 0.9649122807017544, 0.9736842105263158, 0.9561403508771931], 'F1 Score': [0.9859154929577465, 0.9577464788732394, 0.9722222222222222, 0.9790209790209791, 0.9650349650349651], 'Precision Score': [0.9859154929577465, 0.9577464788732394, 0.9589041095890411, 0.9722222222222222, 0.9583333333333334], 'Recall Score': [0.9859154929577465, 0.9577464788732394, 0.9859154929577465, 0.9859154929577465, 0.9718309859154933]}
```

[65]:
```
Results=pd.DataFrame(result)
Results
```

[65]:

	Model	Accuracy Score	F1 Score	Precision Score	Recall Score
0	Logistic Regression	0.982456	0.985915	0.985915	0.985915
1	Decision Tree Classifier	0.947368	0.957746	0.957746	0.957746
2	Random Forest Classifier	0.964912	0.972222	0.958904	0.985915
3	Support Vector Machine (SVM)	0.973684	0.979021	0.972222	0.985915
4	k-Nearest Neighbors (k-NN)	0.956140	0.965035	0.958333	0.971831

[66]:
```
Results.sort_values(by="Accuracy Score",ascending=False)
```

[66]:

	Model	Accuracy Score	F1 Score	Precision Score	Recall Score
0	Logistic Regression	0.982456	0.985915	0.985915	0.985915
3	Support Vector Machine (SVM)	0.973684	0.979021	0.972222	0.985915
2	Random Forest Classifier	0.964912	0.972222	0.958904	0.985915
4	k-Nearest Neighbors (k-NN)	0.956140	0.965035	0.958333	0.971831
1	Decision Tree Classifier	0.947368	0.957746	0.957746	0.957746

- From the above result table,Logistic Regression is the best model with highest accuracy score, F1 score, precision score and recall score.
- The worst model is Decision Tree Classifier which has the lowest accuracy score compared to other models.

[ ]: