```
In [1]: from sklearn.datasets import load_breast_cancer
        import pandas as pd
```

1. Loading and Preprocessing

```
In [2]: df1=load_breast_cancer()
        df1
```

```
Out[2]: {'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 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\nThe 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'}
```

```
In [3]: df = pd.DataFrame(data=df1.data, columns=df1.feature_names)
        df['target'] = df1.target
```

```
In [4]: df
```

Out[4]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst texture	worst perimeter	worst area	worst smoothness	worst compactness	worst concavity	worst concave points	worst symmetry	worst fractal dimension	target
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	0.2419	0.07871	...	17.33	184.60	2019.0	0.16220	0.66560	0.7119	0.2654	0.4601	0.11890	0
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	0.1812	0.05667	...	23.41	158.80	1956.0	0.12380	0.18660	0.2416	0.1860	0.2750	0.08902	0
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	0.2069	0.05999	...	25.53	152.50	1709.0	0.14440	0.42450	0.4504	0.2430	0.3613	0.08758	0
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	0.2597	0.09744	...	26.50	98.87	567.7	0.20980	0.86630	0.6869	0.2575	0.6638	0.17300	0
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	0.1809	0.05883	...	16.67	152.20	1575.0	0.13740	0.20500	0.4000	0.1625	0.2364	0.07678	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
564	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	0.1726	0.05623	...	26.40	166.10	2027.0	0.14100	0.21130	0.4107	0.2216	0.2060	0.07115	0
565	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	0.1752	0.05533	...	38.25	155.00	1731.0	0.11660	0.19220	0.3215	0.1628	0.2572	0.06637	0
566	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	0.1590	0.05648	...	34.12	126.70	1124.0	0.11390	0.30940	0.3403	0.1418	0.2218	0.07820	0
567	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	0.2397	0.07016	...	39.42	184.60	1821.0	0.16500	0.86810	0.9387	0.2650	0.4087	0.12400	0
568	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	0.1587	0.05884	...	30.37	59.16	268.6	0.08996	0.06444	0.0000	0.0000	0.2871	0.07039	1

569 rows × 31 columns

```
In [5]: 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
```

```
In [6]: df.isnull().sum()
```

```
Out[6]: 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
```

There are no null values

```
In [7]: 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
```

```
In [8]: df1.target_names
        target_mapping = {0: df1.target_names[0], 1: df1.target_names[1]}
        print("Target mapping:", target_mapping)
```

```
Target mapping: {0: 'malignant', 1: 'benign'}
```

```
In [9]: df['target'].unique()
```

```
Out[9]: array([0, 1])
```

```
In [10]: from sklearn.preprocessing import StandardScaler
```

```
In [11]: sk=StandardScaler()
         scaled_data=sk.fit_transform(df.drop("target",axis=1))
```

By performing Standard scaler we ensure that all features values are transformed on similar scales.This also ensured that data is centered around origin with standard deviation 1.

2. Classification Algorithm Implementation

1. Logistic Regression

Logistic Regression model uses Logistic function to decide that given set of features belong to a particular class.It works well for binary classification problems like this one.

```
In [12]: import pandas as pd
         import numpy as np
         from sklearn.model_selection import train_test_split
         from sklearn.linear_model import LogisticRegression
         from sklearn.metrics import accuracy_score,confusion_matrix,classification_report
```

```
In [13]: x=scaled_data
         y=df["target"]
```

```
In [14]: x_train,x_test,y_train,y_test=train_test_split(x,y)
```

```
In [15]: print(x_train.shape)
         print(y_train.shape)
         print(x_test.shape)
         print(y_test.shape)
```

```
(426, 30)
(426,)
(143, 30)
(143,)
```

```
In [16]: logr=LogisticRegression()
```

```
In [17]: logr.fit(x_train,y_train)
```

```
Out[17]: ▾ LogisticRegression
         LogisticRegression()
```

```
In [18]: logr_ypred=logr.predict(x_test)
         logr_ypred
```

```
Out[18]: array([1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0,
        1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0,
        0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1,
        0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1,
        1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0,
        0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0,
        1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1])
```

```
In [19]: con=confusion_matrix(y_test,logr_ypred)
         con
```

```
Out[19]: array([[53,  4],
        [ 1, 85]], dtype=int64)
```

```
In [20]: cr=classification_report(y_test,logr_ypred)
         print(cr)
```

```
              precision    recall  f1-score   support

           0       0.98      0.93      0.95        57
           1       0.96      0.99      0.97        86

    accuracy                           0.97       143
   macro avg       0.97      0.96      0.96       143
weighted avg       0.97      0.97      0.96       143
```

```
In [21]: print("accuracy score:",accuracy_score(y_test,logr_ypred))
```

```
accuracy score: 0.965034965034965
```

2. Decision Tree Classifier

The algorithm works by splitting the data at each node based on the feature that results in the largest information gain. This process is repeated until the leaves are pure, meaning that all the samples at each leaf node belong to the same class.

Decision trees are easy to understand and interpret because they mimic human level thinking.

```
In [22]: from sklearn.tree import DecisionTreeClassifier
         dt = DecisionTreeClassifier()
         dt.fit(x_train, y_train)
         dt_y_pred = dt.predict(x_test)
         print("Accuracy:", accuracy_score(y_test, dt_y_pred))
         cr=classification_report(y_test,dt_y_pred)
         print("Classification Report",cr)
```

```
Accuracy: 0.9440559440559441
Classification Report               precision    recall  f1-score   support

           0       0.92      0.95      0.93        57
           1       0.96      0.94      0.95        86

    accuracy                           0.94       143
   macro avg       0.94      0.94      0.94       143
weighted avg       0.94      0.94      0.94       143
```

3. Random Forest Classifier

Random Forest is an ensemble of Decision Trees, where each tree is trained on a random subset of data and features.It is a type of bagging method where the predictions are combined.It reduces overfittiting.

```
In [23]: from sklearn.ensemble import RandomForestClassifier
         rf_model = RandomForestClassifier()
         rf_model.fit(x_train, y_train)
         rf_y_pred = rf_model.predict(x_test)
         print("Accuracy:", accuracy_score(y_test, rf_y_pred))
         cr=classification_report(y_test,rf_y_pred)
         print("Classification Report",cr)
```

```
Accuracy: 0.958041958041958
Classification Report               precision    recall  f1-score   support

           0       1.00      0.89      0.94        57
           1       0.93      1.00      0.97        86

    accuracy                           0.96       143
   macro avg       0.97      0.95      0.96       143
weighted avg       0.96      0.96      0.96       143
```

4. Support Vector Machine (SVM)

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.It is commonly used in medical diagnosis like this one.

```
In [24]: from sklearn.svm import SVC
         svm_classifier = SVC()
         svm_classifier.fit(x_train, y_train)
         svm_y_pred = svm_classifier.predict(x_test)
         print("Accuracy:", accuracy_score(y_test, svm_y_pred))
         cr=classification_report(y_test,svm_y_pred)
         print("Classification Report",cr)
```

```
Accuracy: 0.9790209790209791
Classification Report               precision    recall  f1-score   support

           0       0.98      0.96      0.97        57
           1       0.98      0.99      0.98        86

    accuracy                           0.98       143
   macro avg       0.98      0.98      0.98       143
weighted avg       0.98      0.98      0.98       143
```

5. k-Nearest Neighbors (k-NN)

In this algorithm,a class is assigned to a data point based on the most common class among its 'k' nearest neighbors. It classifies by finding the 'k' closest points in the training set and using their labels to determine the class of the new point.It is simple and can be used for small datasets

```
In [25]: from sklearn.neighbors import KNeighborsClassifier
         knn=KNeighborsClassifier(n_neighbors=5)
         knn.fit(x_train,y_train)
         knn_y_pred=knn.predict(x_test)
         print("Accuracy:", accuracy_score(y_test, knn_y_pred))
         cr=classification_report(y_test,knn_y_pred)
         print("Classification Report",cr)
```

```
Accuracy: 0.958041958041958
Classification Report               precision    recall  f1-score   support

           0       1.00      0.89      0.94        57
           1       0.93      1.00      0.97        86

    accuracy                           0.96       143
   macro avg       0.97      0.95      0.96       143
weighted avg       0.96      0.96      0.96       143
```

```
In [26]: from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
         import pandas as pd
         predictions = {
             "Logistic Regression":logr_ypred,
             "Decision Tree Classifier":dt_y_pred,
             "Random Forest Classifier":rf_y_pred,
             "Support Vector Machine":svm_y_pred,
             "KNN":knn_y_pred
         }
         results = {
             "Model": [],
             "Accuracy": [],
             "Precision": [],
             "Recall": [],
             "F1-Score": []
         }

         for model_name, y_pred in predictions.items():
             results["Model"].append(model_name)
             results["Accuracy"].append(accuracy_score(y_test, y_pred))
             results["Precision"].append(precision_score(y_test, y_pred, average='macro'))
             results["Recall"].append(recall_score(y_test, y_pred, average='macro'))
             results["F1-Score"].append(f1_score(y_test, y_pred, average='macro'))

         results=pd.DataFrame(results)
         print(results)
```

```
                      Model  Accuracy  Precision    Recall  F1-Score
0       Logistic Regression  0.965035   0.968269  0.959098  0.963192
1  Decision Tree Classifier  0.944056   0.939770  0.944614  0.941988
2  Random Forest Classifier  0.958042   0.967391  0.947368  0.955368
3    Support Vector Machine  0.979021   0.979577  0.976642  0.978055
4                       KNN  0.958042   0.967391  0.947368  0.955368
```

```
In [27]: results_accuracy = results.sort_values(by="Accuracy",ascending=False)
         print(results_accuracy)
```

```
                      Model  Accuracy  Precision    Recall  F1-Score
3    Support Vector Machine  0.979021   0.979577  0.976642  0.978055
0       Logistic Regression  0.965035   0.968269  0.959098  0.963192
2  Random Forest Classifier  0.958042   0.967391  0.947368  0.955368
4                       KNN  0.958042   0.967391  0.947368  0.955368
1  Decision Tree Classifier  0.944056   0.939770  0.944614  0.941988
```

```
In [28]: results_F1_score = results.sort_values(by="F1-Score",ascending=False)
         print(results_F1_score)
```

```
                      Model  Accuracy  Precision    Recall  F1-Score
3    Support Vector Machine  0.979021   0.979577  0.976642  0.978055
0       Logistic Regression  0.965035   0.968269  0.959098  0.963192
2  Random Forest Classifier  0.958042   0.967391  0.947368  0.955368
4                       KNN  0.958042   0.967391  0.947368  0.955368
1  Decision Tree Classifier  0.944056   0.939770  0.944614  0.941988
```

Based on both accuracy score and F1 score, we cans see that Support Vector Machine model perfomed the best and Decison Tree Classifier has perfomed the worst.