```
In [37]: from sklearn.datasets import load_breast_cancer
         from sklearn.preprocessing import StandardScaler
         import pandas as pd
```

```
In [38]: data = load_breast_cancer()
         x = pd.DataFrame(data.data,columns=data.feature_names)
         y = pd.Series(data.target)
```

```
In [39]: data
```

```
Out[39]: {'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 o
```

```
f Attributes: 30 numeric, predictive attributes and the class\n\n    :Attribute Information:\n        - rad
ius (mean of distances from center to points on the perimeter)\n        - texture (standard deviation of gr
ay-scale values)\n        - perimeter\n        - area\n        - smoothness (local variation in radius leng
ths)\n        - compactness (perimeter^2 / area - 1.0)\n        - concavity (severity of concave portions o
f 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 lar
gest (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 i
s Worst Radius.\n\n        - class:\n                - WDBC-Malignant\n                - WDBC-Benign\n\n
:Summary Statistics:\n\n    ===================================== ====== ======\n
Min    Max\n    ===================================== ====== ======\n    radius (mean):
6.981  28.11\n    texture (mean):                       9.71   39.28\n    perimeter (mean):
43.79  188.5\n    area (mean):                          143.5  2501.0\n    smoothness (mean):
0.053  0.163\n    compactness (mean):                   0.019  0.345\n    concavity (mean):
0.0    0.427\n    concave points (mean):                0.0    0.201\n    symmetry (mean):
0.106  0.304\n    fractal dimension (mean):             0.05   0.097\n    radius (standard error):
0.112  2.873\n    texture (standard error):             0.36   4.885\n    perimeter (standard error):
0.757  21.98\n    area (standard error):                6.802  542.2\n    smoothness (standard error):
0.002  0.031\n    compactness (standard error):         0.002  0.135\n    concavity (standard error):
0.0    0.396\n    concave points (standard error):      0.0    0.053\n    symmetry (standard error):
0.008  0.079\n    fractal dimension (standard error):   0.001  0.03\n    radius (worst):
7.93   36.04\n    texture (worst):                      12.02  49.54\n    perimeter (worst):
50.41  251.2\n    area (worst):                         185.2  4254.0\n    smoothness (worst):
0.071  0.223\n    compactness (worst):                  0.027  1.058\n    concavity (worst):
0.0    1.252\n    concave points (worst):               0.0    0.291\n    symmetry (worst):
0.156  0.664\n    fractal dimension (worst):            0.055  0.208\n    =================================
==== ====== ======\n\n    :Missing Attribute Values: None\n\n    :Class Distribution: 212 - Malignant, 357
- Benign\n\n    :Creator:  Dr. William H. Wolberg, W. Nick Street, Olvi L. Mangasarian\n\n    :Donor: Nick
Street\n\n    :Date: November, 1995\n\nThis is a copy of UCI ML Breast Cancer Wisconsin (Diagnostic) datase
ts.\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 pl
ane described above was obtained using\nMultisurface Method-Tree (MSM-T) [K. P. Bennett, "Decision Tree\nCo
nstruction Via Linear Programming." Proceedings of the 4th\nMidwest Artificial Intelligence and Cognitive S
cience Society,\npp. 97-101, 1992], a classification method which uses linear\nprogramming to construct a d
ecision tree.  Relevant features\nwere selected using an exhaustive search in the space of 1-4\nfeatures an
d 1-3 separating planes.\n\nThe actual linear program used to obtain the separating plane\nin the 3-dimensi
onal space is that described in:\n[K. P. Bennett and O. L. Mangasarian: "Robust Linear\nProgramming Discrim
ination of Two Linearly Inseparable Sets",\nOptimization Methods and Software 1, 1992, 23-34].\n\nThis data
base is also available through the UW CS ftp server:\n\nftp ftp.cs.wisc.edu\ncd math-prog/cpo-dataset/machi
ne-learn/WDBC/\n\n.. topic:: References\n\n   - W.N. Street, W.H. Wolberg and O.L. Mangasarian. Nuclear fea
ture extraction \n     for breast tumor diagnosis. IS&T/SPIE 1993 International Symposium on \n     Electro
nic Imaging: Science and Technology, volume 1905, pages 861-870,\n     San Jose, CA, 1993.\n   - O.L. Manga
sarian, 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. Can
cer Letters 77 (1994) \n     163-171.',
 'feature_names': array(['mean radius', 'mean texture', 'mean perimeter', 'mean area',
        'mean smoothness', 'mean compactness', 'mean concavity',
        'mean concave points', 'mean symmetry', 'mean fractal dimension',
        'radius error', 'texture error', 'perimeter error', 'area error',
        'smoothness error', 'compactness error', 'concavity error',
        'concave points error', 'symmetry error',
        'fractal dimension error', 'worst radius', 'worst texture',
        'worst perimeter', 'worst area', 'worst smoothness',
        'worst compactness', 'worst concavity', 'worst concave points',
        'worst symmetry', 'worst fractal dimension'], dtype='<U23'),
 'filename': 'breast_cancer.csv',
 'data_module': 'sklearn.datasets.data'}
```

In [40]:
```
x
```

Out[40]:

	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	peri
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	0.2419	0.07871	...	25.380	17.33	
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	0.1812	0.05667	...	24.990	23.41	
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	0.2069	0.05999	...	23.570	25.53	
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	0.2597	0.09744	...	14.910	26.50	
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	0.1809	0.05883	...	22.540	16.67	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
564	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	0.1726	0.05623	...	25.450	26.40	
565	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	0.1752	0.05533	...	23.690	38.25	
566	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	0.1590	0.05648	...	18.980	34.12	
567	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	0.2397	0.07016	...	25.740	39.42	
568	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	0.1587	0.05884	...	9.456	30.37	

569 rows × 30 columns

In [41]:
```
y
```

Out[41]:
```
0      0
1      0
2      0
3      0
4      0
      ..
564    0
565    0
566    0
567    0
568    1
Length: 569, dtype: int32
```

In [ ]:
```
#1. Handling Missing Values:

#The breast cancer dataset from sklearn does not have any missing values. However, it's good practice to chec
#missing data and handle it appropriately (e.g., using mean imputation or removing the samples).



#2. Feature Scaling:

#Why necessary: Some classification algorithms like SVM and k-NN are sensitive to the scale of input features
#For instance, SVM with radial kernels and k-NN depend on distance measures,
#which can be dominated by features with large ranges.
```

In [42]:
```
scaler = StandardScaler()
x_scaled = scaler.fit_transform(x)
```

In [43]:
```
#LOGISTIC REGRESSION
#How it works: Logistic regression models the probability of a binary outcome based on the features using a l
#It outputs a probability between 0 and 1, which is used to classify the observation into one of the two cate

#Why suitable: It’s a simple, interpretable algorithm that works well with linear relationships between the f
#the target variable.
```

In [44]:
```
from sklearn.linear_model import LogisticRegression
log_reg = LogisticRegression()
log_reg.fit(x_scaled, y)
```

Out[44]:
```
▾ LogisticRegression
LogisticRegression()
```

In [45]:
```
from sklearn.metrics import accuracy_score
```

In [46]:
```
from sklearn.model_selection import train_test_split
```

In [47]:
```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.3, random_state=42)
```

In [48]:
```
x_train_scaled = scaler.fit_transform(x_train)
x_test_scaled = scaler.transform(x_test)
```

In [49]:
```
from sklearn.linear_model import LogisticRegression
log_reg = LogisticRegression()
log_reg.fit(x_train_scaled, y_train)
```

Out[49]:
```
▾ LogisticRegression
LogisticRegression()
```

In [50]:
```
y_pred = log_reg.predict(x_test_scaled)
```

In [18]:
```
print("logistic regression prediction:", y_pred)
print("logistic regression accuracy:", accuracy_score(y_test,y_pred))
```

```
logistic regression prediction: [1 0 0 1 1 0 0 0 1 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 1 0
 1 0 1 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 0 1 1 0 0 1 0
 1 1 1 0 1 1 0 1 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 1 0 0 1 0 0 1 1 1 0 1 1 0
 1 0 0 0 0 1 1 1 0 1 1 1 0 1 0 0 1 1 0 0 0 1 1 1 0 1 1 1 0 1 0 1 1 0 1 0 0
 0 1 0 1 1 1 1 0 0 1 1 1 1 1 1 1 0 1 1 1 1 0 1]
logistic regression accuracy: 0.9824561403508771
```

In [ ]:
```
#DECISION TREE CLASSIFIER
#How it works: A decision tree splits the dataset into smaller subsets based on feature values,
#creating a tree-like model of decisions. Each internal node represents a decision, and the leaf nodes repres

#Why suitable: Decision trees can handle non-linear relationships and interactions between features
#without the need for scaling.
```

In [54]:
```
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier()
dt.fit(x_train_scaled, y_train)
```

Out[54]:
```
▾ DecisionTreeClassifier

DecisionTreeClassifier()
```

In [55]:
```
y_pred = dt.predict(x_test_scaled)
```

In [21]:
```
print("Decision Tree prediction:", y_pred)
print("Decision Tree accuracy:", accuracy_score(y_test,y_pred))
```

```
Decision Tree prediction: [1 0 0 1 1 0 0 1 1 1 0 0 1 0 1 0 1 1 1 0 1 1 0 1 1 1 1 1 0 1 1 1 1 1 1 0
 1 0 1 1 0 1 1 1 1 0 0 1 1 0 0 1 1 1 1 1 0 0 1 1 0 0 1 1 1 0 0 1 1 0 0 1 0
 1 1 1 0 1 1 0 1 1 0 1 0 0 0 1 1 1 1 0 1 1 1 0 0 1 0 0 1 0 0 1 1 1 0 0 1 0
 1 1 0 1 0 1 1 1 0 0 1 1 0 1 0 0 1 1 0 0 0 0 1 1 0 0 1 1 0 1 0 1 1 0 1 0 0
 0 1 0 1 1 1 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 1]
Decision Tree accuracy: 0.9181286549707602
```

In [ ]:
```
#RANDOM FOREST CLASSIFIER
#How it works: Random forests are an ensemble of decision trees, where each tree is built on a random subset
#The final prediction is based on the majority vote or average of the individual trees' predictions.

#Why suitable: It improves upon decision trees by reducing overfitting and provides a more robust model,
#especially for larger datasets.
```

In [58]:
```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier()
rf.fit(x_train_scaled, y_train)
```

Out[58]:
```
▾ RandomForestClassifier

RandomForestClassifier()
```

In [23]:
```
y_pred = rf.predict(x_test_scaled)
```

In [24]:
```
print("Random Forest prediction:", y_pred)
print("Random Forest accuracy:", accuracy_score(y_test,y_pred))
```

```
Random Forest prediction: [1 0 0 1 1 0 0 0 0 1 1 0 1 0 1 0 1 1 1 0 1 1 0 1 1 1 1 1 0 1 1 1 1 1 1 0
 1 0 1 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 0 1 1 0 0 1 0
 1 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 1 0 0 1 1 1 0 1 1 0
 1 1 0 1 0 1 1 1 0 1 1 1 0 1 0 0 1 1 0 0 0 1 1 1 0 1 1 1 0 1 0 1 1 0 1 0 0
 0 1 0 1 1 1 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 1]
Random Forest accuracy: 0.9707602339181286
```

In [ ]:
```
#SUPPORT VECTOR MACHINE(SVM)
#How it works: SVM constructs a hyperplane in a high-dimensional space that best separates the two classes by
#maximizing the margin between them. SVMs can handle non-linear boundaries using kernel functions.

#Why suitable: It works well for both linear and non-linear decision boundaries and
#can be effective in high-dimensional spaces.
```

In [61]:
```
from sklearn.svm import SVC
svm = SVC(kernel='linear')  # Using linear kernel for simplicity
svm.fit(x_train_scaled, y_train)
```

Out[61]:
```
▾          SVC
SVC(kernel='linear')
```

In [26]:
```
y_pred = svm.predict(x_test_scaled)
```

In [27]:
```
print("SVM prediction:", y_pred)
print("SVM accuracy:", accuracy_score(y_test,y_pred))
```

```
SVM prediction: [1 0 0 1 1 0 0 0 1 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 1 0
 1 0 1 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 0 1 1 0 0 1 0
 1 1 1 1 1 1 0 1 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 1 0 0 1 0 0 1 1 1 0 1 1 0
 1 0 0 0 0 1 1 1 0 1 1 1 0 1 0 0 1 1 0 0 0 1 1 1 0 1 1 1 0 1 0 1 1 0 1 0 0
 0 1 0 1 1 1 1 0 0 1 1 1 1 1 1 1 0 1 1 1 1 0 1]
SVM accuracy: 0.9766081871345029
```

In [ ]:
```
#K-NEAREST NEIGHBORS (K-NN)
#How it works: k-NN classifies a new observation based on the majority class of its k nearest neighbors in th
#using a distance metric like Euclidean distance.

#Why suitable: It’s a simple, non-parametric method that can handle non-linear decision boundaries,
#but it’s computationally expensive and sensitive to feature scaling.
```

In [64]:
```python
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(x_train_scaled, y_train)
```

Out[64]:
```
▾ KNeighborsClassifier

KNeighborsClassifier()
```

In [29]:
```python
y_pred = knn.predict(x_test_scaled)
```

In [30]:
```python
print("KNN prediction:", y_pred)
print("KNN accuracy:", accuracy_score(y_test,y_pred))
```

```
KNN prediction: [1 0 0 1 1 0 0 0 0 1 1 0 1 1 1 0 1 1 1 0 1 1 0 1 1 1 1 1 1 0 1 1 1 1 1 1 0
 1 0 1 1 0 1 1 1 1 1 1 1 1 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 1 0 1 0 0 0 0 0 0 1 1 1 0 1 1 1 1 0 0 1 0 0 1 0 0 1 1 1 0 1 1 0
 1 1 0 1 0 1 1 1 0 1 1 1 0 1 0 0 1 1 0 0 0 1 1 1 0 1 1 1 0 1 0 1 1 0 1 0 0
 0 1 0 1 1 1 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 1]
KNN accuracy: 0.9590643274853801
```

In [ ]:
```python
# MODEL COMPARISON
```

In [51]:
```python
y_pred = log_reg.predict(x_test_scaled)
```

In [52]:
```python
print("logistic regression accuracy:", accuracy_score(y_test,y_pred))
```

```
logistic regression accuracy: 0.9824561403508771
```

In [56]:
```python
y_pred = dt.predict(x_test_scaled)
```

In [57]:
```python
print("Decision Tree accuracy:", accuracy_score(y_test,y_pred))
```

```
Decision Tree accuracy: 0.9298245614035088
```

In [59]:
```
y_pred = rf.predict(x_test_scaled)
```

In [60]:
```
print("Random Forest accuracy:", accuracy_score(y_test,y_pred))
```

```
Random Forest accuracy: 0.9649122807017544
```

In [62]:
```
y_pred = svm.predict(x_test_scaled)
```

In [63]:
```
print("SVM accuracy:", accuracy_score(y_test,y_pred))
```

```
SVM accuracy: 0.9766081871345029
```

In [65]:
```
y_pred = knn.predict(x_test_scaled)
```

In [66]:
```
print("KNN accuracy:", accuracy_score(y_test,y_pred))
```

```
KNN accuracy: 0.9590643274853801
```

In [ ]:
```
#CONCLUSION
#Best perorming algorithm: Logistic regression with an accuracy of 0.98,
#as it is a robust ensemble model that reduces overfitting and captures complex
#patterns in the data
#Worst performing algorithm: Decision tree with an accuracy of 0.92, which
#tends to overfit on small datasets like this one due to its high variance
```