In [28]:

```
#### Binary Classification
# Import necessary libraries
import pandas as pd
import numpy as np
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# Load the dataset
data = load_breast_cancer()
X = data.data
y = data.target

# Convert to a DataFrame for better visualization (optional)
df = pd.DataFrame(X, columns=data.feature_names)
df['target'] = y

# Data Preprocessing: Standardize the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,random_state=42)

# Train a classification model (Logistic Regression)
model = LogisticRegression()
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)

# Print the evaluation results
print(f"Accuracy: {accuracy:.2f}")
print("Confusion Matrix:")
print(conf_matrix)
```

```
print("Classification Report:")
print(class_report)
```

```
Accuracy: 0.97
Confusion Matrix:
[[41  2]
 [ 1 70]]
Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.95      0.96        43
           1       0.97      0.99      0.98        71

    accuracy                           0.97       114
   macro avg       0.97      0.97      0.97       114
weighted avg       0.97      0.97      0.97       114
```

In [31]:

```
### Multi CLassification
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix

# Load the Iris dataset
iris = load_iris()
X = iris.data
y = iris.target

# Convert to DataFrame for better readability
df = pd.DataFrame(X, columns=iris.feature_names)
df['target'] = y
print(df.head())

# Standardizing the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Splitting the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)
```

```
# Initialize the RandomForestClassifier
clf = RandomForestClassifier(n_estimators=100, random_state=42)

# Train the model
clf.fit(X_train, y_train)

# Make predictions
y_pred = clf.predict(X_test)

# Evaluate the model
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# Making predictions on new data
new_data = np.array([[5.0, 3.6, 1.4, 0.2]])
new_data_scaled = scaler.transform(new_data)
prediction = clf.predict(new_data_scaled)
predicted_class = iris.target_names[prediction]
print(f"Predicted class for the new data: {predicted_class[0]}")
```

```
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  \
0                5.1               3.5                1.4               0.2
1                4.9               3.0                1.4               0.2
2                4.7               3.2                1.3               0.2
3                4.6               3.1                1.5               0.2
4                5.0               3.6                1.4               0.2

   target
0       0
1       0
2       0
3       0
4       0
Confusion Matrix:
[[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00         9
           2       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30

Predicted class for the new data: setosa
```

In [10]:

```python
### Logistic Regression
import numpy as np
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix, accuracy_score

# Load the dataset
dataset = pd.read_csv('data.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
```

```
# Display the first 10 rows of the dataset
print(dataset.head(10))

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=42)

# Feature Scaling
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Initialize the Logistic Regression model
classifier = LogisticRegression(random_state=0, max_iter=100)
classifier.fit(X_train, y_train)

# Predict on the test set
y_pred = classifier.predict(X_test)

# Display the results (confusion matrix and accuracy)
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(cm)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
```

```
   SNo       X_1       X_2   y
0    0 -0.869144  0.389310  0.0
1    1 -0.993467 -0.610591  0.0
2    2 -0.834064  0.239236  0.0
3    3 -0.136471  0.632003  1.0
4    4  0.403887  0.310784  1.0
5    5 -0.569309 -0.246681  0.0
6    6 -0.109982  0.930917  1.0
7    7  0.288994 -0.532689  1.0
8    8  0.319782  0.664582  1.0
9    9  0.558686 -0.621185  1.0
Confusion Matrix:
[[ 8  1]
 [ 3 18]]
Accuracy: 0.87
```

In [12]:

```
### K-nearest neighbour
import numpy as np
import pandas as pd
```

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics

# Define column names and load the dataset
names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'Class']
dataset = pd.read_csv("iris_data.csv", names=names)
X = dataset.iloc[:, :-1]
y = dataset.iloc[:, -1]
print(X.head())

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.10, random_state=42)

# Initialize and train the K-Nearest Neighbors classifier
classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(X_train, y_train)

# Make predictions
y_pred = classifier.predict(X_test)


# Display the results with original and predicted labels
i=0
print("\n-------------------------------------------------------------------------------")
print("%-25s %-25s %-25s" % ("Original Label", "Predicted Label", "Correct/Wrong"))
print("-------------------------------------------------------------------------------")
for label in y_test:
    print('%-25s %-25s' % (label, y_pred[i]), end="")
    if label == y_pred[i]:
        print(" %-25s" % "Correct")
    else:
        print(" %-25s" % "Wrong")
    i=i+1

# Print the confusion matrix and classification report
print("-------------------------------------------------------------------------------")
print("\nConfusion Matrix:\n", metrics.confusion_matrix(y_test, y_pred))
print("-------------------------------------------------------------------------------")
print("\nClassification Report:\n", metrics.classification_report(y_test, y_pred))
print("-------------------------------------------------------------------------------")
print("Accuracy of the classifier is %0.2f" % metrics.accuracy_score(y_test, y_pred))
```

```
   sepal-length  sepal-width  petal-length  petal-width
0           5.1          3.5           1.4          0.2
1           4.9          3.0           1.4          0.2
2           4.7          3.2           1.3          0.2
3           4.6          3.1           1.5          0.2
4           5.0          3.6           1.4          0.2

-------------------------------------------------------------------------
Original Label            Predicted Label           Correct/Wrong
-------------------------------------------------------------------------
Iris-versicolor           Iris-versicolor           Correct
Iris-virginica            Iris-virginica            Correct
-------------------------------------------------------------------------

Confusion Matrix:
 [[1 0]
 [0 1]]
-------------------------------------------------------------------------

Classification Report:
                  precision    recall  f1-score   support

Iris-versicolor       1.00      1.00      1.00         1
 Iris-virginica       1.00      1.00      1.00         1

       accuracy                           1.00         2
      macro avg       1.00      1.00      1.00         2
   weighted avg       1.00      1.00      1.00         2

-------------------------------------------------------------------------
Accuracy of the classifier is 1.00
```

In [41]:

```python
### SVM Classification
import numpy as np
import pandas as pd

# After reading the dataset, divide the dataset into concepts and targets. Store the concepts in
dataset = pd.read_csv('Breast_Cancer (1).csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values

# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=0)
```

```
# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Training the Support Vector Machine (SVM) Classification model on the Training set
from sklearn.svm import SVC
classifier = SVC(kernel='linear', random_state=0)
classifier.fit(X_train, y_train)
```

Out[41]:

SVC

```
SVC(kernel='linear', random_state=0)
```

In [39]:

```
# Support Vector Machine (SVM) classifier model
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
    kernel='linear', max_iter=-1, probability=False, random_state=0,
    shrinking=True, tol=0.001, verbose=False)
```

Out[39]:

SVC

```
SVC(gamma='auto_deprecated', kernel='linear', random_state=0)
```

In [42]:

```
# Evaluating the SVM classifier model
from sklearn.metrics import confusion_matrix, accuracy_score
y_pred = classifier.predict(X_test)
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
```

Out[42]:

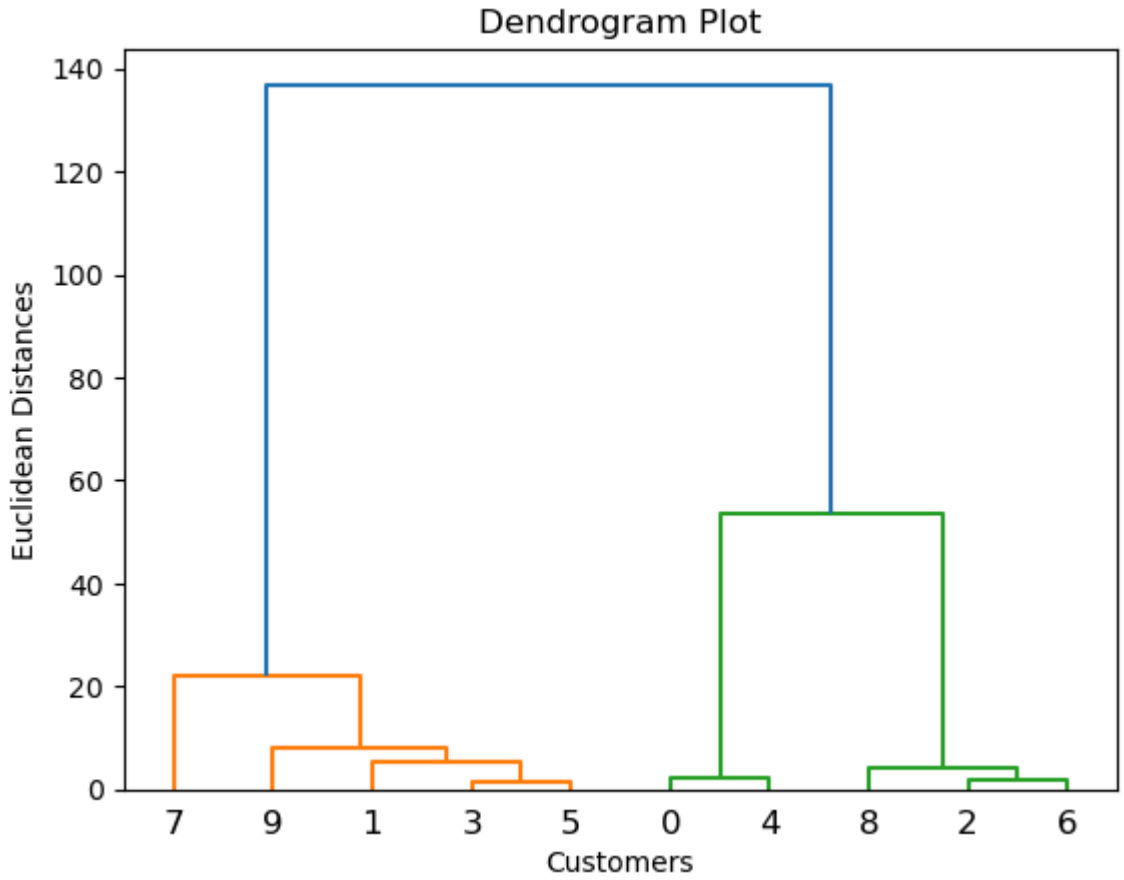
```
[[102   5]
 [  5  59]]
0.9415204678362573
```

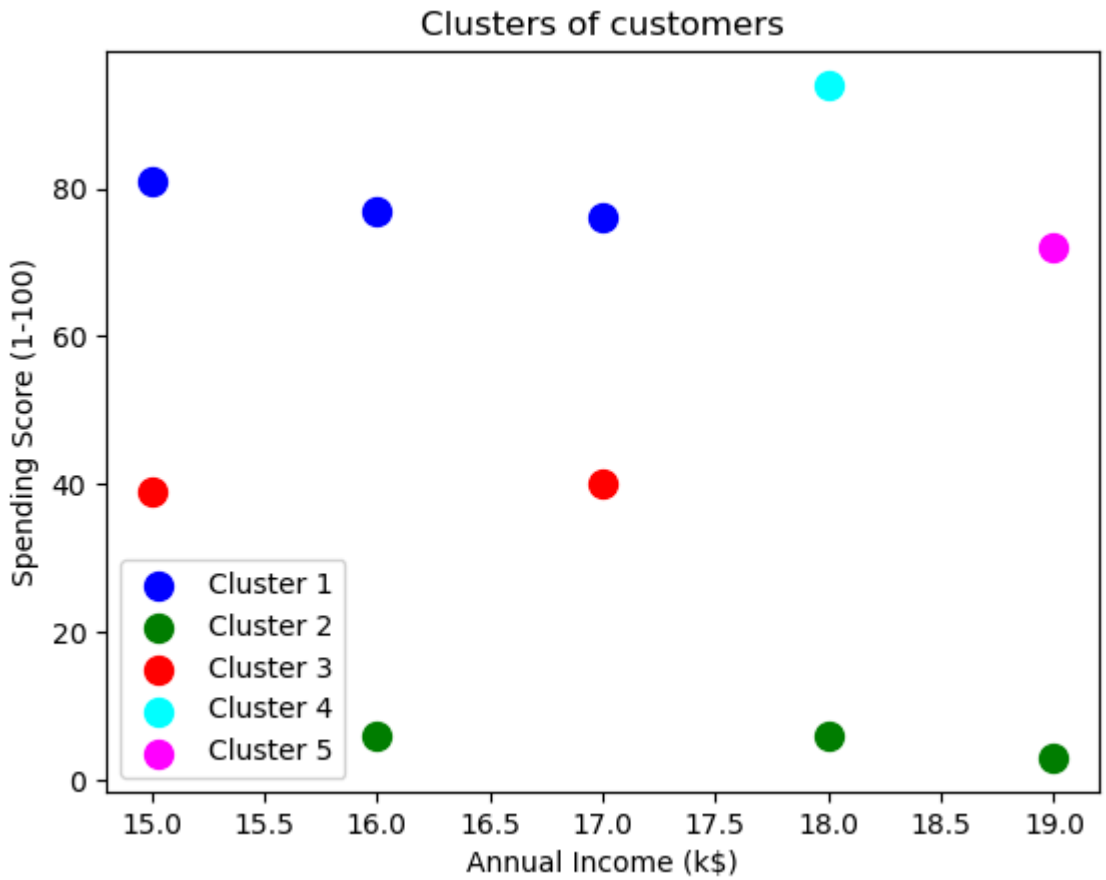
In [24]:

```
### Hierarchical Clustering
# Importing the libraries
import numpy as nm
import matplotlib.pyplot as mtp
```

```
import pandas as pd

# Importing the dataset
dataset = pd.read_csv('Mall_Customers.csv')
print(dataset.head(10))

x = dataset.iloc[:, [3, 4]].values

# Finding the optimal number of clusters using the dendrogram
import scipy.cluster.hierarchy as shc
dendro = shc.dendrogram(shc.linkage(x, method="ward"))
mtp.title("Dendrogram Plot")
mtp.ylabel("Euclidean Distances")
mtp.xlabel("Customers")
mtp.show()

# Training the hierarchical model on the dataset
from sklearn.cluster import AgglomerativeClustering
hc = AgglomerativeClustering(n_clusters=5, metric='euclidean', linkage='ward')
y_pred = hc.fit_predict(x)

# Visualizing the clusters
mtp.scatter(x[y_pred == 0, 0], x[y_pred == 0, 1], s = 100, c = 'blue', label = 'Cluster 1')
mtp.scatter(x[y_pred == 1, 0], x[y_pred == 1, 1], s = 100, c = 'green', label = 'Cluster 2')
mtp.scatter(x[y_pred == 2, 0], x[y_pred == 2, 1], s = 100, c = 'red', label = 'Cluster 3')
mtp.scatter(x[y_pred == 3, 0], x[y_pred == 3, 1], s = 100, c = 'cyan', label = 'Cluster 4')
mtp.scatter(x[y_pred == 4, 0], x[y_pred == 4, 1], s = 100, c = 'magenta', label = 'Cluster 5')

mtp.title('Clusters of customers')
mtp.xlabel('Annual Income (k$)')
mtp.ylabel('Spending Score (1-100)')
mtp.legend()
mtp.show()
```

	CustomerID	Genre	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40
5	6	Female	22	17	76
6	7	Female	35	18	6
7	8	Female	23	18	94
8	9	Male	64	19	3
9	10	Female	30	19	72




Clusters of customers
Spending Score (1-100)
Annual Income (k$)
Cluster 1
Cluster 2
Cluster 3
Cluster 4
Cluster 5
0
20
40
60
80
15.0
15.5
16.0
16.5
17.0
17.5
18.0
18.5
19.0

In [ ]: