```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Knn

```
from sklearn.neighbors import KNeighborsClassifier
knn_model = KNeighborsClassifier(n_neighbors=5)
knn_model.fit(X_train, y_train)
y_pred_knn = knn_model.predict(X_test)
from sklearn.metrics import accuracy_score
print("KNN Accuracy:", accuracy_score(y_test, y_pred_knn))
```

Decision Tree

```
from sklearn.tree import DecisionTreeClassifier
model_DS = DecisionTreeClassifier()
model_DS.fit(X_train,y_train)
y_pred_dt = model_DS.predict(X_test)
from sklearn.metrics import accuracy_score
print("Decision Tree Accuracy:", accuracy_score(y_test, y_pred_dt))
```

SVM

```
from sklearn.svm import SVC
model_svm = SVC()
model_svm.fit(X_train,y_train)
y_pred_svm = model_svm.predict(X_test)
from sklearn.metrics import accuracy_score
print("SVM Accuracy:", accuracy_score(y_test, y_pred_svm))
```

Naïve Bayes

```
from sklearn.naive_bayes import GaussianNB
nb_model = GaussianNB()
nb_model.fit(X_train, y_train)
y_pred_nb = nb_model.predict(X_test)
from sklearn.metrics import accuracy_score
print("Naive Bayes Accuracy:", accuracy_score(y_test, y_pred_nb))
```

Linear Regression

```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
from sklearn.metrics import mean_squared_error, r2_score
print("MSE:", mean_squared_error(y_test, y_pred))
print("R2 Score:", r2_score(y_test, y_pred))
```

Logistic Regression

```
from sklearn.linear_model import LogisticRegression
model_log = LogisticRegression(max_iter=1000)
model_log.fit(X_train, y_train)
y_pred_lr = model_log.predict(X_test)
from sklearn.metrics import accuracy_score
accuracy_score(y_pred_lr,y_test)
```

AIM: Evaluating a classification model using metrics such as accuracy, precision, recall, and F1 score.

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Prediction
y_pred_dt = model_DS.predict(X_test)

# Accuracy
print("Accuracy:", accuracy_score(y_test, y_pred_dt))

# Precision
print("Precision:", precision_score(y_test, y_pred_dt, average='weighted'))

# Recall
print("Recall:", recall_score(y_test, y_pred_dt, average='weighted'))

# F1 Score
print("F1 Score:", f1_score(y_test, y_pred_dt, average='weighted'))
```

AIM: Applying hierarchical clustering to group customer segments based on their purchasing behaviour.

```
import matplotlib.pyplot as plt
import scipy.cluster.hierarchy as sch

plt.figure(figsize=(10, 6))
dendrogram = sch.dendrogram(
    sch.linkage(df.select_dtypes(include='number'), method="ward")
)

plt.title("Dendrogram")
plt.xlabel("Customers")
plt.ylabel("Euclidean Distance")
plt.show()

from sklearn.cluster import AgglomerativeClustering

hc = AgglomerativeClustering(
    n_clusters=5,
    metric='euclidean',
    linkage='average'
)

y_hc = hc.fit_predict(df)
df['cluster'] = y_hc

y_hc

df["cluster"] = pd.DataFrame(y_hc)

import matplotlib.pyplot as plt

# Select columns: Age (0) and MaxHR (6)
X = df.iloc[:, [0, 6]].values   # Age, MaxHR
y_hc = df['cluster'].values

plt.figure(figsize=(6, 4))

plt.scatter(X[y_hc==0, 0], X[y_hc==0, 1], s=60, label='Cluster 1')
plt.scatter(X[y_hc==1, 0], X[y_hc==1, 1], s=60, label='Cluster 2')
plt.scatter(X[y_hc==2, 0], X[y_hc==2, 1], s=60, label='Cluster 3')
plt.scatter(X[y_hc==3, 0], X[y_hc==3, 1], s=60, label='Cluster 4')
plt.scatter(X[y_hc==4, 0], X[y_hc==4, 1], s=60, label='Cluster 5')

plt.title('Heart Disease Clusters (Hierarchical Clustering)')
plt.xlabel('Age')
plt.ylabel('Maximum Heart Rate')
plt.legend()
plt.show()
```

AIM: Implementing the K-means clustering algorithm

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score

# Load dataset
df = pd.read_csv("iris.csv")

# Select features
X = df[['sepal_length', 'sepal_width', 'petal_length', 'petal_width']]

# Create KMeans model
kmeans = KMeans(n_clusters=3, random_state=42)

# Train model
kmeans.fit(X)

# Get cluster labels
labels = kmeans.labels_

# Add cluster column to dataframe
df['cluster'] = labels

# Evaluation
print("Inertia:", kmeans.inertia_)
print("Silhouette Score:", silhouette_score(X, labels))

# Plot clusters (Petal Length vs Petal Width)
plt.scatter(
    df['petal_length'],
    df['petal_width'],
    c=labels
)

plt.xlabel("Petal Length")
plt.ylabel("Petal Width")
plt.title("K-Means Clustering on Iris Dataset")
plt.show()
```

AIM: Utilizing Principal Component Analysis (PCA) for dimensionality reduction to improve the efficiency and interpretability of a model.

```
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Features (remove target and cluster)
X = df.drop(['target', 'cluster'], axis=1)

# Target (cluster)
y_cluster = df['cluster']

# Scaling
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# PCA
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

print("Explained variance ratio:", pca.explained_variance_ratio_)

# Logistic Regression
model = LogisticRegression(max_iter=500)
model.fit(X_pca, y_cluster)

# Prediction
y_pred = model.predict(X_pca)

# Metrics
print("Accuracy:", accuracy_score(y_cluster, y_pred))
print("Precision:", precision_score(y_cluster, y_pred, average='weighted'))
print("Recall:", recall_score(y_cluster, y_pred, average='weighted'))
print("F1 Score:", f1_score(y_cluster, y_pred, average='weighted'))

# Plot
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y_cluster)
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.title("PCA Visualization (Cluster-wise)")
plt.show()
```