```
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error,
mean_squared_error, r2_score

# Load the student data set
student_data = np.loadtxt(r"Datasets/Student_Dataset[Slip1].csv",
delimiter=",")

# Split the data into features and target
X_train = student_data[:, :-1]
y_train = student_data[:, -1]

# Create a linear regression model
model = LinearRegression()

# Fit the model to the training data
model.fit(X_train, y_train)

# Make predictions on the training data
y_pred = model.predict(X_train)

# Calculate the mean absolute error, mean squared error, and root
mean squared error
mae = mean_absolute_error(y_train, y_pred)
mse = mean_squared_error(y_train, y_pred)
rmse = np.sqrt(mse)

# Print the results
print("Mean Absolute Error:", mae)
print("Mean Squared Error:", mse)
print("Root Mean Squared Error:", rmse)
```

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs

# Generate synthetic data
data, _ = make_blobs(n_samples=300, centers=4, random_state=42)

# Visualize the synthetic data
plt.scatter(data[:, 0], data[:, 1], s=50)
plt.title("Synthetic Data")
plt.show()

# Apply k-means algorithm
k = 4  # Number of clusters
kmeans = KMeans(n_clusters=k)
kmeans.fit(data)

# Get the centroids and labels
centroids = kmeans.cluster_centers_
labels = kmeans.labels_

# Visualize the clusters
colors = ['b', 'g', 'r', 'y']
for i in range(k):
    cluster_points = data[labels == i]
    plt.scatter(cluster_points[:, 0], cluster_points[:, 1], c=colors[i],
label=f'Cluster {i + 1}')

# Plot centroids
plt.scatter(centroids[:, 0], centroids[:, 1], marker='x', s=200,
linewidths=3, color='black', label='Centroids')
plt.title("K-Means Clustering")
plt.legend()
plt.show()
```

```
import numpy as np

x = np.array([0,1,2,3,4,5,6,7,8,9,11,13])
y = np.array([1, 3, 2, 5, 7, 8, 8, 9, 10, 12,16, 18])

# calculate mean of x and y
mean_x = np.mean(x)
mean_y = np.mean(y)

# calculate covariance of x and y
cov_xy = np.sum((x - mean_x) * (y - mean_y)) / (len(x) - 1)

# calculate variance of x
var_x = np.sum((x - mean_x)**2) / (len(x) - 1)

# calculate slope (b1)
b1 = cov_xy / var_x

# calculate intercept (b0)
b0 = mean_y - b1 * mean_x

# print estimated coefficients
print("Estimated coefficients:")
print("B0:", b0)
print("B1:", b1)
```

```
# Given dataset
weather = ['Sunny', 'Sunny', 'Overcast', 'Rainy', 'Rainy', 'Rainy',
'Overcast', 'Sunny', 'Sunny', 'Rainy', 'Sunny', 'Overcast', 'Overcast',
'Rainy']
temp = ['Hot', 'Hot', 'Hot', 'Mild', 'Cool', 'Cool', 'Cool', 'Mild', 'Cool',
'Mild', 'Mild', 'Mild', 'Hot', 'Mild']
play = ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'Yes',
'Yes', 'Yes', 'No']

# Tuple to predict
new_data = [('Overcast', 'Mild')]

# Count occurrences of each class
play_count = {'Yes': play.count('Yes'), 'No': play.count('No')}

# Calculate P(Outlook | Play) and P(Temperature | Play)
outlook_given_play = {}
temp_given_play = {}

for outlook, temp, p in zip(weather, temp, play):
    if p not in outlook_given_play:
        outlook_given_play[p] = {}
        temp_given_play[p] = {}

    if outlook not in outlook_given_play[p]:
        outlook_given_play[p][outlook] = 0
    if temp not in temp_given_play[p]:
        temp_given_play[p][temp] = 0

    outlook_given_play[p][outlook] += 1
    temp_given_play[p][temp] += 1

# Calculate probabilities for the new tuple
new_outlook, new_temp = new_data[0]
p_play_yes = (outlook_given_play['Yes'].get(new_outlook, 0) /
play_count['Yes']) * (temp_given_play['Yes'].get(new_temp, 0) /
play_count['Yes']) * (play_count['Yes'] / len(play))
p_play_no = (outlook_given_play['No'].get(new_outlook, 0) /
play_count['No']) * (temp_given_play['No'].get(new_temp, 0) /
play_count['No']) * (play_count['No'] / len(play))

# Print the probabilities
print("P(Play=Yes | Outlook={}, Temperature={}) =
{:.4f}".format(new_outlook, new_temp, p_play_yes))
print("P(Play=No | Outlook={}, Temperature={}) =
{:.4f}".format(new_outlook, new_temp, p_play_no))

# Predict the class
predicted_class = 'Yes' if p_play_yes > p_play_no else 'No'
print("Predicted class: {}".format(predicted_class))
```

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix
from sklearn import preprocessing

diabetes_data = pd.read_csv("Datasets/
Diabetes_Dataset[Slip5].csv")

# Display the first few rows of the dataset to understand its
structure
print(diabetes_data.head())

# Split the data into features (X) and target variable (y)
X = diabetes_data.drop('Outcome', axis=1)  # Features
y = diabetes_data['Outcome']  # Target variable

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

# Standardize the features (optional but recommended for
Decision Trees)
scaler = preprocessing.StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Initialize the Decision Tree Classifier
clf = DecisionTreeClassifier(random_state=42)

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

# Make predictions on the test set
y_pred = clf.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)

# Display the results
print(f"Accuracy: {accuracy:.2f}")
print("\nConfusion Matrix:")
print(conf_matrix)
print("\nClassification Report:")
print(class_report)
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import AgglomerativeClustering
from scipy.cluster.hierarchy import dendrogram, linkage

df = pd.read_csv("Datasets/Customer[Slip6].csv")

# Display the first few rows of the dataset to understand its
structure
print(df.head())

# Ensure that the column indices are correct
# Adjust these indices based on the actual structure of your
dataset
X = df.iloc[:, [2, 3, 4]].values  # Considering 'age', 'annual_income',
'spending_score'

# Hierarchical clustering using Ward's method
linked = linkage(X, 'ward')

# Plotting the dendrogram
plt.figure(figsize=(12, 8))
dendrogram(linked, orientation='top',
distance_sort='descending',show_leaf_counts=True)
plt.title('Hierarchical Clustering Dendrogram')
plt.xlabel('Customer Index')
plt.ylabel('Euclidean Distance')
plt.show()

# Performing Agglomerative Clustering
num_clusters = 5  # You can choose the number of clusters based
on the dendrogram
hc = AgglomerativeClustering(n_clusters=num_clusters,
metric='euclidean', linkage='ward')
y_hc = hc.fit_predict(X)

# Add cluster labels to the original dataset
df['Cluster'] = y_hc

# Display the first few rows of the dataset with cluster labels
print(df.head())
```

```
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt

# Data
x = np.array([1, 2, 3, 4, 5, 6, 7, 8]).reshape(-1, 1)
y = np.array([7, 14, 15, 18, 19, 21, 26, 23])

# Create a linear regression model
model = LinearRegression()

# Fit the model
model.fit(x, y)

# Get the coefficients
b0 = model.intercept_
b1 = model.coef_[0]

# Predict the values
y_pred = model.predict(x)

# Analyze the performance of the model
mse = mean_squared_error(y, y_pred)
r2 = r2_score(y, y_pred)

# Print the coefficients
print(f"Intercept (b0): {b0}")
print(f"Slope (b1): {b1}")

# Print the performance metrics
print(f"Mean Squared Error: {mse}")
print(f"R-squared: {r2}")

# Plot the data and the regression line
plt.scatter(x, y, color='blue', label='Actual data')
plt.plot(x, y_pred, color='red', linewidth=2, label='Regression line')
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Simple Linear Regression')
plt.legend()
plt.show()
```

```
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

# Load the dataset
data = pd.read_csv('Datasets/Credit_Card_CC_General[Slip8].csv')
# data = data.drop('Time', axis=1)

# Handle missing values (if needed)
data = data.dropna()

# Standardize the data
scaler = StandardScaler()
data_scaled = scaler.fit_transform(data)

# Determine the optimal number of clusters using the Elbow
Method
wcss = []  # within-cluster sum of squares

for i in range(1, 11):
   kmeans = KMeans(n_clusters=i, init='k-means++', max_iter=300,
n_init=10, random_state=0)
   kmeans.fit(data_scaled)
   wcss.append(kmeans.inertia_)

# Plot the Elbow Method graph
plt.plot(range(1, 11), wcss)
plt.title('Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')  # within-cluster sum of squares
plt.show()

# Based on the Elbow Method, choose the optimal number of
clusters
optimal_clusters = 4

# Apply k-means clustering with the optimal number of clusters
kmeans = KMeans(n_clusters=optimal_clusters, init='k-means++',
max_iter=300, n_init=10, random_state=0)
data['cluster'] = kmeans.fit_predict(data_scaled)

print(data.head())
```

```
# Import necessary libraries
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score,
recall_score

# Load the Breast Cancer Wisconsin dataset
cancer_data = datasets.load_breast_cancer()

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(
    cancer_data.data, cancer_data.target, test_size=0.2,
random_state=42
)

# Initialize the Support Vector Machine model
svm_model = SVC(kernel='linear', C=1)
print("Features: ", cancer_data.feature_names)
print("Labels: ", cancer_data.target_names)
cancer_data.data.shape
11
print(cancer_data.data[0:5])
print(cancer_data.target)
# Train the model
svm_model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = svm_model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)

# Print the results
print(f"Accuracy: {accuracy:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
```

```
import pandas as pd
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori, association_rules

df = pd.read_csv('Datasets/MarketBasket[Slip10].csv',
header=None)

# Display the first few rows of the dataset to understand its
structure
print("Dataset structure:")
print(df.head())

# Preprocess the data for Apriori algorithm
transactions = df.apply(lambda row: row.dropna().tolist(),
axis=1).values.tolist()

# Use TransactionEncoder to one-hot encode the transaction data
te = TransactionEncoder()
te_ary = te.fit(transactions).transform(transactions)
df_encoded = pd.DataFrame(te_ary, columns=te.columns_)

# Apply Apriori algorithm to find frequent itemsets
min_support = 0.01  # You can adjust this parameter based on your
dataset
frequent_itemsets = apriori(df_encoded,
min_support=min_support, use_colnames=True)

# Display frequent itemsets with support
print("\nFrequent Itemsets:")
print(frequent_itemsets)

# Generate association rules
rules = association_rules(frequent_itemsets, metric="confidence",
min_threshold=0.5)

# Display association rules with support, confidence, and lift
print("\nAssociation Rules:")
print(rules[['antecedents', 'consequents', 'support', 'confidence',
'lift']])
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import AgglomerativeClustering
from sklearn.preprocessing import StandardScaler
from scipy.cluster.hierarchy import dendrogram, linkage

data = pd.read_csv("Datasets/Wholesale_Customers[Slip11].csv")

# Select relevant fields for clustering
fields = ['Fresh', 'Milk', 'Grocery', 'Frozen', 'Detergents_Paper',
'Delicassen']
data_numeric = data[fields]

# Standardize the data
scaler = StandardScaler()
data_scaled = scaler.fit_transform(data_numeric)

# Hierarchical clustering using AgglomerativeClustering
num_clusters = 3  # You can adjust the number of clusters as
needed
model = AgglomerativeClustering(n_clusters=num_clusters,
linkage='ward')
clusters = model.fit_predict(data_scaled)

# Add the cluster labels to the original dataset
data['Cluster'] = clusters

# Print the count of points in each cluster
print("Number of points in each cluster:")
print(data['Cluster'].value_counts())

# Plot the dendrogram
plt.figure(figsize=(12, 6))
dendrogram(linkage(data_scaled, method='ward'))
plt.title('Dendrogram')
plt.xlabel('Data points')
plt.ylabel('Euclidean distance')
plt.show()
```

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics

df = pd.read_csv('Datasets/Car_Dataset[Slip12].csv')

# Display the first few rows of the dataset
print(df.head())

# Selecting features (independent variables) and target (dependent
variable)
X = df[['Volume', 'Weight']]
y = df['CO2']

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

# Create a linear regression model
model = LinearRegression()

# Train the model on the training set
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Evaluate the model
print('\nModel Evaluation:')
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test,
y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test,
y_pred))
print('Root Mean Squared Error:',
metrics.mean_squared_error(y_test, y_pred, squared=False))

# Display the coefficients and intercept
print('\nModel Coefficients:')
for feature, coefficient in zip(X.columns, model.coef_):
   print(f'{feature}: {coefficient}')

print('Intercept:', model.intercept_)
```

```
import pandas as pd

# Read the CSV file
df = pd.read_csv('Datasets/Student_Performance[Slip13].csv')

# Display the shape of the dataset
print("Shape of the dataset:", df.shape)

# Display the top rows of the dataset with their columns
print("\nTop rows of the dataset:")
print(df.head(10))
```

```
import pandas as pd
from mlxtend.frequent_patterns import apriori, association_rules

df = pd.read_csv('Datasets/Groceries[Slip14].csv')

# Display the first few rows of the dataset
print("First few rows of the dataset:")
print(df.head())

# Data preprocessing
basket = df.groupby(['Member_number', 'itemDescription'])
['itemDescription'].count().unstack().reset_index().fillna(0).set_inde
x('Member_number')

# Convert the count values to binary (1 or 0)
basket_sets = basket.map(lambda x: 1 if x > 0 else 0).astype(bool)

# Apply Apriori algorithm
frequent_itemsets = apriori(basket_sets, min_support=0.01,
use_colnames=True)

# Generate association rules
rules = association_rules(frequent_itemsets, metric="confidence",
min_threshold=0.2)

# Display support and confidence for each rule
print("\nAssociation Rules:")
print(rules[['antecedents', 'consequents', 'support', 'confidence']])
```

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn import metrics

data = pd.read_csv('Datasets/TV_Shows[Slip15].csv')

# Display the dataset to understand its structure
print("Dataset:")
print(data)

# Map 'Go' column to numeric values (Yes: 1, No: 0)
data['Go'] = data['Go'].map({'Yes': 1, 'No': 0})

# Separate features (X) and target variable (y)
X = data[['Age', 'Experience', 'Rank', 'Nationality']]
y = data['Go']

# Convert 'Nationality' to numerical values using one-hot encoding
X = pd.get_dummies(X, columns=['Nationality'], drop_first=True)

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

# Create a Decision Tree Classifier
dt_classifier = DecisionTreeClassifier()

# Train the classifier
dt_classifier.fit(X_train, y_train)

# Predict class label for a comedian with specific characteristics
comedian_data = {'Age': 40, 'Experience': 10, 'Rank': 7,
'Nationality_USA': 1 }
comedian_df = pd.DataFrame([comedian_data],
columns=X_train.columns)

# Make predictions
prediction = dt_classifier.predict(comedian_df)

# Display the prediction
print("\nPrediction:")
if prediction[0] == 1:
   print("The comedian should 'Go'.")
else:
   print("The comedian should not 'Go'.")

# Evaluate the model on the test set
y_pred = dt_classifier.predict(X_test)
accuracy = metrics.accuracy_score(y_test, y_pred)
print("\nModel Accuracy on Test Set:", accuracy)
```

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import LabelEncoder

# Load the dataset
data = pd.read_csv('Datasets/TV_Shows[Slip15].csv')

# Encode categorical variables
le = LabelEncoder()
data['Nationality'] = le.fit_transform(data['Nationality'])
data['Go'] = le.fit_transform(data['Go'])

# Features and target variable
X = data[['Age', 'Experience', 'Rank', 'Nationality']]
y = data['Go']

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

# Build Decision Tree Classifier
dt_classifier = DecisionTreeClassifier()
dt_classifier.fit(X_train, y_train)

# Predict class label for a new data point
new_data_point = pd.DataFrame({'Age': [40], 'Experience': [10],
'Rank': [7], 'Nationality': [1]})
predicted_label = dt_classifier.predict(new_data_point)

# Convert the predicted label back to original class label
predicted_class = le.inverse_transform(predicted_label)

print(f"The predicted class label for the new data point is:
{predicted_class[0]}")
```

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix
from sklearn import preprocessing

diabetes_data = pd.read_csv("Datasets/
Diabetes_Dataset[Slip16].csv")

# Display the first few rows of the dataset to understand its
structure
print(diabetes_data.head())

# Split the data into features (X) and target variable (y)
X = diabetes_data.drop('Outcome', axis=1)  # Features
y = diabetes_data['Outcome']  # Target variable

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

# Standardize the features (optional but recommended for
Decision Trees)
scaler = preprocessing.StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Initialize the Decision Tree Classifier
clf = DecisionTreeClassifier(random_state=42)

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

# Make predictions on the test set
y_pred = clf.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)

# Display the results
print(f"Accuracy: {accuracy:.2f}")
print("\nConfusion Matrix:")
print(conf_matrix)
print("\nClassification Report:")
print(class_report)
```

```
import pandas as pd
import matplotlib.pyplot as plt

Stock_Market = {
   'Year': [2017, 2017, 2017, 2017, 2017, 2017, 2017, 2017, 2017, 2017,
2017, 2017, 2016, 2016, 2016, 2016, 2016, 2016, 2016, 2016, 2016,
2016, 2016, 2016],
   'Month': [12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3,
2, 1],
   'Interest_Rate': [2.75, 2.5, 2.5, 2.5, 2.5, 2.5, 2.5, 2.25, 2.25, 2.25, 2, 2,
2, 1.75, 1.75, 1.75, 1.75, 1.75, 1.75, 1.75, 1.75, 1.75, 1.75, 1.75],
   'Unemployment_Rate': [5.3, 5.3, 5.3, 5.3, 5.4, 5.6, 5.5, 5.5, 5.5, 5.6,
5.7, 5.9, 6, 5.9, 5.8, 6.1, 6.2, 6.1, 6.1, 6.1, 5.9, 6.2, 6.2, 6.1],
   'Stock_Index_Price': [1464, 1394, 1357, 1293, 1256, 1254, 1234,
1195, 1159, 1167, 1130, 1075, 1047, 965, 943, 958, 971, 949, 884, 866,
876, 822, 704, 719]
}

df = pd.DataFrame(Stock_Market, columns=['Year', 'Month',
'Interest_Rate', 'Unemployment_Rate', 'Stock_Index_Price'])

plt.scatter(df['Interest_Rate'], df['Stock_Index_Price'], color='red')
plt.title('Stock Index Price Vs Interest Rate', fontsize=14)
plt.xlabel('Interest Rate', fontsize=14)
plt.ylabel('Stock Index Price', fontsize=14)
plt.grid(True)
plt.show()
```

```
# install this using terminal : pip install statsmodels matplotlib
import pandas as pd
import statsmodels.api as sm
import matplotlib.pyplot as plt

# Create the stock market data frame
Stock_Market = {
    'Year': [2017, 2017, 2017, 2017, 2017, 2017, 2017, 2017, 2017, 2017,
2017, 2017, 2016, 2016, 2016, 2016, 2016, 2016, 2016, 2016, 2016,
2016, 2016, 2016],
    'Month': [12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3,
2, 1],
    'Interest_Rate': [2.75, 2.5, 2.5, 2.5, 2.5, 2.5, 2.5, 2.25, 2.25, 2.25, 2, 2,
2, 1.75, 1.75, 1.75, 1.75, 1.75, 1.75, 1.75, 1.75, 1.75, 1.75, 1.75],
    'Unemployment_Rate': [5.3, 5.3, 5.3, 5.3, 5.4, 5.6, 5.5, 5.5, 5.5, 5.6,
5.7, 5.9, 6, 5.9, 5.8, 6.1, 6.2, 6.1, 6.1, 6.1, 5.9, 6.2, 6.2, 6.1],
    'Stock_Index_Price': [1464, 1394, 1357, 1293, 1256, 1254, 1234,
1195, 1159, 1167, 1130, 1075, 1047, 965, 943, 958, 971, 949, 884, 866,
876, 822, 704, 719]
}

df = pd.DataFrame(Stock_Market)

# Add a constant term to the independent variables matrix
X = sm.add_constant(df[['Interest_Rate', 'Unemployment_Rate']])

# Fit the regression model
model = sm.OLS(df['Stock_Index_Price'], X).fit()

# Print the regression summary
print(model.summary())

# Plot the graph of Stock Market Price versus Interest Rate
plt.scatter(df['Interest_Rate'], df['Stock_Index_Price'], label='Data
Points')
plt.plot(df['Interest_Rate'], model.predict(X), color='red',
label='Regression Line')
plt.title('Stock Market Price vs Interest Rate')
plt.xlabel('Interest Rate')
plt.ylabel('Stock Market Price')
plt.legend()
plt.show()
```

```
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt

# Data
x = np.array([1, 2, 3, 4, 5, 6, 7, 8]).reshape(-1, 1)
y = np.array([7, 14, 15, 18, 19, 21, 26, 23])

# Create a linear regression model
model = LinearRegression()

# Fit the model
model.fit(x, y)

# Get the coefficients
b0 = model.intercept_
b1 = model.coef_[0]

# Predict the values
y_pred = model.predict(x)

# Analyze the performance of the model
mse = mean_squared_error(y, y_pred)
r2 = r2_score(y, y_pred)

# Print the coefficients
print(f"Intercept (b0): {b0}")
print(f"Slope (b1): {b1}")

# Print the performance metrics
print(f"Mean Squared Error: {mse}")
print(f"R-squared: {r2}")

# Plot the data and the regression line
plt.scatter(x, y, color='blue', label='Actual data')
plt.plot(x, y_pred, color='red', linewidth=2, label='Regression line')
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Simple Linear Regression')
plt.legend()
plt.show()
```

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics

df = pd.read_csv('Datasets/Car_Dataset[Slip19].csv')

# Display the first few rows of the dataset
print(df.head())

# Selecting features (independent variables) and target (dependent
variable)
X = df[['Volume', 'Weight']]
y = df['CO2']

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

# Create a linear regression model
model = LinearRegression()

# Train the model on the training set
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Evaluate the model
print('\nModel Evaluation:')
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test,
y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test,
y_pred))
print('Root Mean Squared Error:',
metrics.mean_squared_error(y_test, y_pred, squared=False))

# Display the coefficients and intercept
print('\nModel Coefficients:')
for feature, coefficient in zip(X.columns, model.coef_):
    print(f'{feature}: {coefficient}')

print('Intercept:', model.intercept_)
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import AgglomerativeClustering
from scipy.cluster.hierarchy import dendrogram, linkage

df = pd.read_csv("Datasets/Customer[Slip20].csv")

# Display the first few rows of the dataset to understand its
structure
print(df.head())

# Ensure that the column indices are correct
# Adjust these indices based on the actual structure of your
dataset
X = df.iloc[:, [2, 3, 4]].values  # Considering 'age', 'annual_income',
'spending_score'

# Hierarchical clustering using Ward's method
linked = linkage(X, 'ward')

# Plotting the dendrogram
plt.figure(figsize=(12, 8))
dendrogram(linked, orientation='top',
distance_sort='descending',show_leaf_counts=True)
plt.title('Hierarchical Clustering Dendrogram')
plt.xlabel('Customer Index')
plt.ylabel('Euclidean Distance')
plt.show()

# Performing Agglomerative Clustering
num_clusters = 5  # You can choose the number of clusters based
on the dendrogram
hc = AgglomerativeClustering(n_clusters=num_clusters,
metric='euclidean', linkage='ward')
y_hc = hc.fit_predict(X)

# Add cluster labels to the original dataset
df['Cluster'] = y_hc

# Display the first few rows of the dataset with cluster labels
print(df.head())
```