

Machine learning day 1

#1

```
import pandas as pd
```

```
import seaborn as sns
```

```
import matplotlib.pyplot as plt
```

```
from sklearn.model_selection import train_test_split
```

```
from sklearn.naive_bayes import GaussianNB
```

```
from sklearn.metrics import accuracy_score
```

Step 1: Reading the dataset

```
data = pd.DataFrame({  
    'battery_power': [842, 1021, 563, 615, 1821, 1859, 1821, 1954, 1445, 509],  
    'blue': [0, 1, 1, 0, 0, 1, 1, 0, 1, 1],  
    'clock_speed': [2.2, 0.5, 0.5, 2.5, 1.2, 0.5, 1.0, 0.5, 1.7, 0.5],  
    'dual_sim': [0, 1, 1, 0, 1, 0, 1, 1, 1, 1],  
    'fc': [1, 0, 2, 13, 3, 4, 0, 3, 0, 0],  
    'four_g': [0, 1, 1, 1, 1, 1, 1, 1, 0, 1],  
    'int_memory': [7, 53, 41, 10, 44, 22, 4, 53, 22, 46],  
    'm_dep': [0.6, 0.7, 0.9, 0.2, 0.5, 0.3, 0.4, 0.1, 0.8, 0.1],  
    'mobile_wt': [188, 136, 145, 131, 141, 164, 139, 187, 174, 93],  
    'n_cores': [2, 3, 5, 6, 2, 3, 5, 6, 2, 5],  
    'pc': [2, 6, 6, 9, 14, 7, 0, 16, 10, 6],  
    'px_height': [20, 905, 1263, 1216, 1208, 1004, 381, 512, 1988, 754],  
    'px_width': [756, 1988, 1716, 1786, 1215, 1654, 1366, 1028, 858, 1784],  
    'ram': [2549, 2631, 2603, 2769, 1411, 1067, 3220, 700, 1099, 513],  
    'sc_h': [9, 17, 11, 16, 8, 13, 17, 19, 11, 16],  
    'sc_w': [7, 3, 2, 8, 2, 8, 1, 10, 0, 7],  
    'talk_time': [19, 7, 9, 11, 15, 10, 18, 5, 19, 2],  
    'three_g': [0, 1, 1, 1, 1, 1, 1, 1, 0, 1],  
    'touch_screen': [0, 1, 1, 0, 1, 1, 0, 1, 0, 1],  
    'wifi': [1, 0, 1, 0, 1, 0, 0, 1, 0, 1],
```

```
'price_range': [1, 2, 2, 2, 1, 1, 1, 3, 0, 0]
})
```

Step 2: Printing the first five rows

```
print(data.head())
```

Step 3: Basic statistical computations

```
print(data.describe())
```

Step 4: Identifying columns and their data types

```
print(data.info())
```

Step 5: Detecting and handling null values

```
print(data.isnull().sum())
```

Replacing null values with mode

```
for column in data.columns:
```

```
    if data[column].isnull().sum() > 0:
```

```
        mode_value = data[column].mode()[0]
```

```
        data[column].fillna(mode_value, inplace=True)
```

```
print(data.isnull().sum())
```

Step 6: Exploring the dataset using a heatmap

```
plt.figure(figsize=(10, 8))
```

```
sns.heatmap(data.corr(), annot=True, cmap='coolwarm')
```

```
plt.show()
```

Step 7: Splitting the data into training and testing sets

```
X = data.drop('price_range', axis=1)
```

```
y = data['price_range']
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Step 8: Fitting the Naive Bayes Classifier model

```
model = GaussianNB()
```

```
model.fit(X_train, y_train)
```

Step 9: Predicting with the model

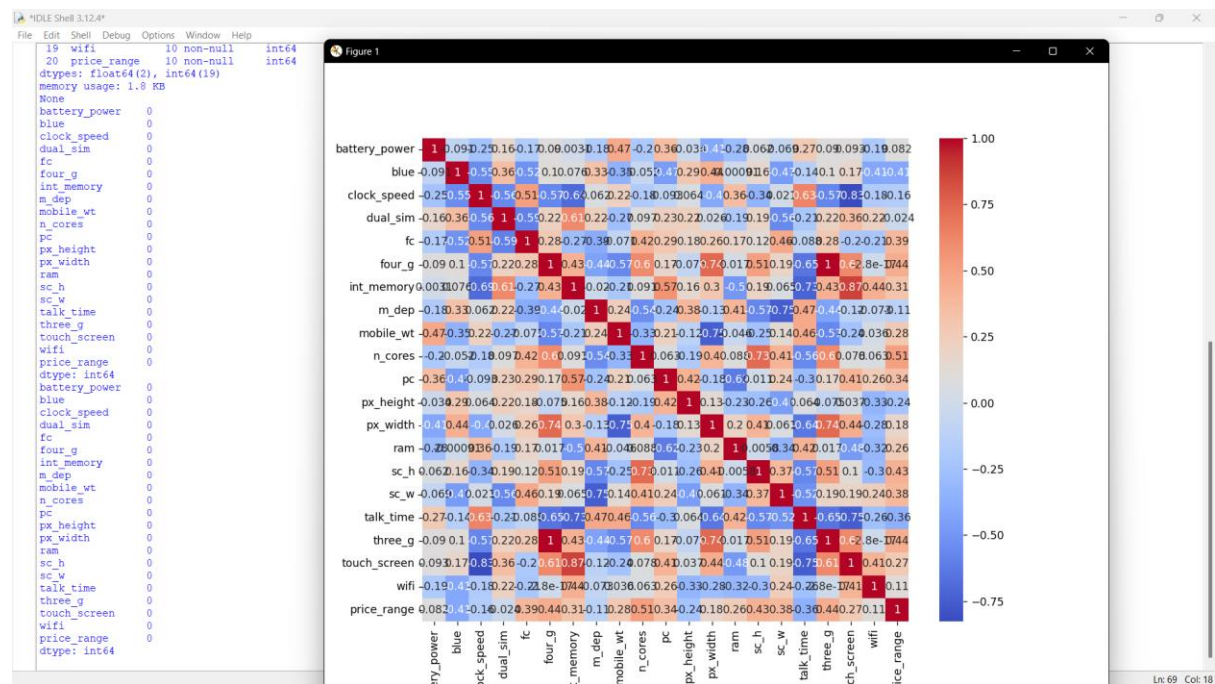
```
y_pred = model.predict(X_test)
```

Step 10: Finding the accuracy of the model

```
accuracy = accuracy_score(y_test, y_pred)
```

```
print(f'Accuracy of the Naive Bayes Classifier: {accuracy:.2f}')
```

#output



#02

import pandas as pd

data = {

```

'Sky': ['Sunny', 'Sunny', 'Rainy', 'Sunny'],
'Air Temp': ['Warm', 'Warm', 'Cold', 'Warm'],
'Humidity': ['Normal', 'High', 'High', 'High'],
'Wind': ['Strong', 'Strong', 'Strong', 'Strong'],
'Water': ['Warm', 'Warm', 'Warm', 'Cool'],
'Forecast': ['Same', 'Same', 'Change', 'Change'],
'Enjoy Sport': ['Yes', 'Yes', 'No', 'Yes']
}
df = pd.DataFrame(data)

```

```

def find_s_algorithm(dataframe):
    features = dataframe.iloc[:, :-1].values
    target = dataframe.iloc[:, -1].values
    hypothesis = None
    for i in range(len(target)):
        if target[i] == 'Yes':
            hypothesis = features[i].copy()
            break
    for i in range(len(features)):
        if target[i] == 'Yes':
            for j in range(len(hypothesis)):
                if hypothesis[j] != features[i][j]:
                    hypothesis[j] = '?'

    return hypothesis

hypothesis = find_s_algorithm(df)
print('The most specific hypothesis is:', hypothesis)

#output

```

```

File Edit Shell Debug Options Window Help
Python 3.12.4 (tags/v3.12.4:8e8a4ba, Jun 6 2024, 19:30:16) [MSC v.1940 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/DINESH/mlq2.py
The most specific hypothesis is: ['Sunny' 'Warm' '?' 'Strong' '?' '?']
>>>

```

#03

```

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression

from sklearn.datasets import fetch_california_housing

from sklearn.metrics import mean_squared_error, r2_score

# Step 1: Load the dataset

housing = fetch_california_housing()

X = pd.DataFrame(housing.data, columns=housing.feature_names)

y = pd.Series(housing.target)

# Step 2: 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)

# Step 3: Train the Linear Regression model

model = LinearRegression()

model.fit(X_train, y_train)

# Step 4: Make predictions

y_pred = model.predict(X_test)

# Step 5: Evaluate the model's performance

mse = mean_squared_error(y_test, y_pred)

r2 = r2_score(y_test, y_pred)

print(f'Mean Squared Error: {mse:.2f}')

print(f'R^2 Score: {r2:.2f}')

```

```

plt.figure(figsize=(10, 6))

plt.scatter(y_test, y_pred, color='blue')

plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red', linewidth=2)

plt.xlabel('Actual')

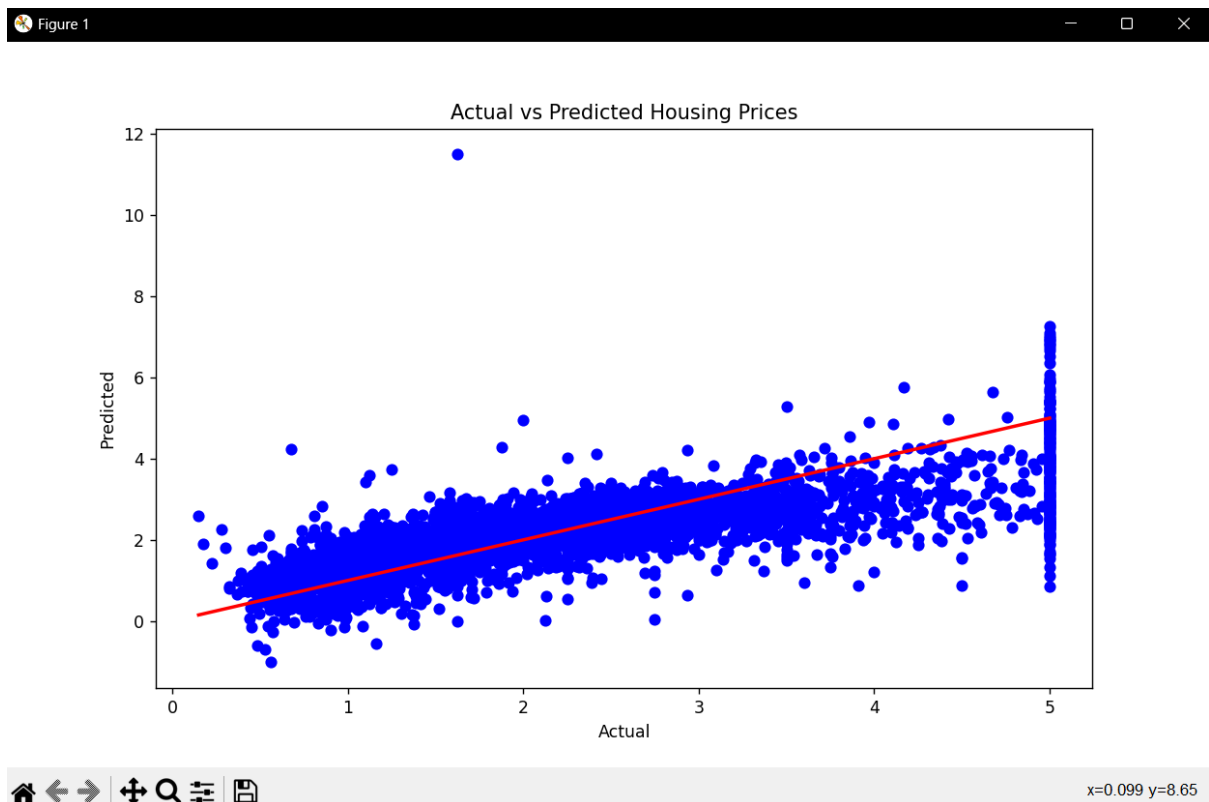
plt.ylabel('Predicted')

plt.title('Actual vs Predicted Housing Prices')

plt.show()

#output

```



#04

```

import pandas as pd

from sklearn.datasets import load_iris

from sklearn.model_selection import train_test_split

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

import seaborn as sns

import matplotlib.pyplot as plt

```

```
# Step 1: Load the Iris dataset
```

```
iris = load_iris()
```

```
X = pd.DataFrame(iris.data, columns=iris.feature_names)
```

```
y = pd.Series(iris.target)
```

```
# Step 2: 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)
```

```
# Step 3: Train the KNN model
```

```
knn = KNeighborsClassifier(n_neighbors=3)
```

```
knn.fit(X_train, y_train)
```

```
# Step 4: Make predictions
```

```
y_pred = knn.predict(X_test)
```

```
# Step 5: Evaluate the model's performance
```

```
accuracy = accuracy_score(y_test, y_pred)
```

```
print(f'Accuracy: {accuracy:.2f}')
```

```
print('Classification Report:')
```

```
print(classification_report(y_test, y_pred))
```

```
# Step 6: Visualize the results
```

```
conf_matrix = confusion_matrix(y_test, y_pred)
```

```
plt.figure(figsize=(8, 6))
```

```
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=iris.target_names,  
yticklabels=iris.target_names)
```

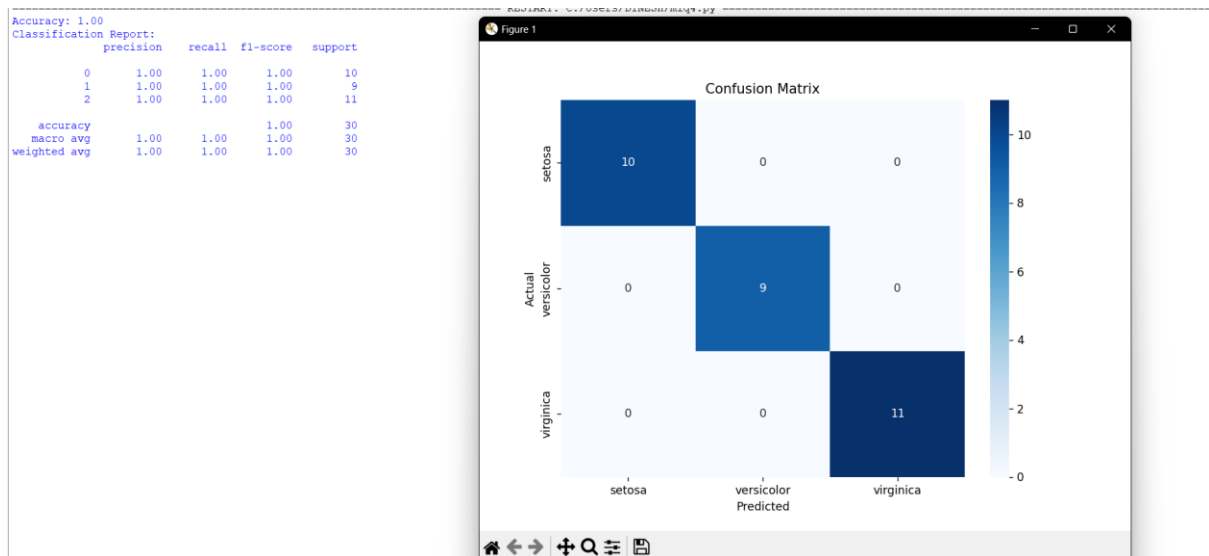
```
plt.xlabel('Predicted')
```

```
plt.ylabel('Actual')
```

```
plt.title('Confusion Matrix')
```

```
plt.show()
```

#output



#05

```
import numpy as np
```

```
import pandas as pd
```

```
from sklearn.datasets import load_iris
```

```
from sklearn.model_selection import train_test_split
```

```
from sklearn.linear_model import Perceptron
```

```
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

```
import seaborn as sns
```

```
import matplotlib.pyplot as plt
```

```
# Step 1: Load the Iris dataset
```

```
iris = load_iris()
```

```
X = pd.DataFrame(iris.data, columns=iris.feature_names)
```

```
y = pd.Series(iris.target)
```

```
# Step 2: 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)
```

```
# Step 3: Train the Perceptron model using One-vs-Rest (OvR) strategy
```

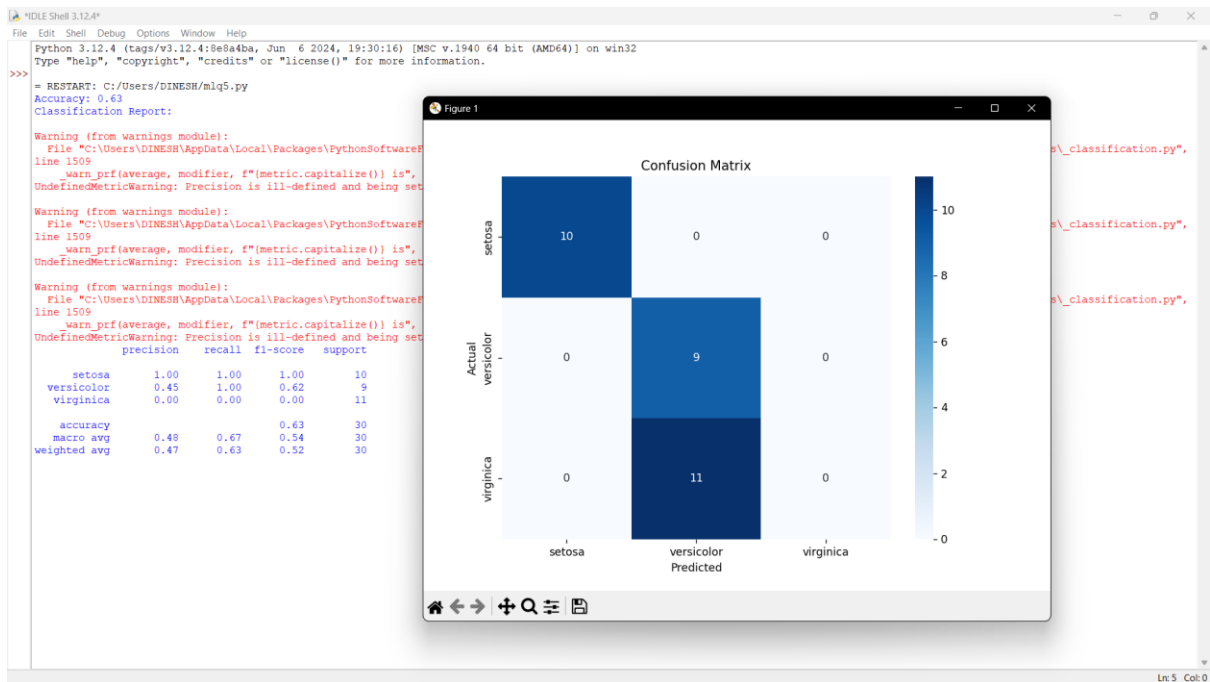


```
perceptron = Perceptron(max_iter=1000, tol=1e-3, random_state=42)
perceptron.fit(X_train, y_train)

# Step 4: Make predictions
y_pred = perceptron.predict(X_test)

# Step 5: Evaluate the model's performance
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
print('Classification Report:')
print(classification_report(y_test, y_pred, target_names=iris.target_names))

# Step 6: Visualize the results
# Confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=iris.target_names,
yticklabels=iris.target_names)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```



#06

import pandas as pd

Step 1: Load the training data from a CSV file

data = {

 'Sky': ['Sunny', 'Sunny', 'Rainy', 'Sunny'],

 'Air Temp': ['Warm', 'Warm', 'Cold', 'Warm'],

 'Humidity': ['Normal', 'High', 'High', 'High'],

 'Wind': ['Strong', 'Strong', 'Strong', 'Strong'],

 'Water': ['Warm', 'Warm', 'Warm', 'Cool'],

 'Forecast': ['Same', 'Same', 'Change', 'Change'],

 'Enjoy Sport': ['Yes', 'Yes', 'No', 'Yes']

}

df = pd.DataFrame(data)

Convert the dataframe to a list of lists

examples = df.values.tolist()


```
        if value != specific_h[j]:
            new_g = g.copy()
            new_g[j] = value
            if consistent(new_g, specific_h):
                new_general_h.append(new_g)
    general_h = new_general_h.copy()
```

```
    print(f"\nExample {i + 1} processed")
    print(f"Specific Hypothesis: {specific_h}")
    print(f"General Hypothesis: {general_h}")
```

```
    return specific_h, general_h
```

Step 3: Apply the Candidate-Elimination algorithm to the dataset

```
specific_h, general_h = candidate_elimination(examples)
```

Step 4: Output the final version space

```
print("\nFinal Version Space:")
print(f"Specific Hypothesis: {specific_h}")
print(f"General Hypothesis: {general_h}")
```

#output

```
IDLE Shell 3.12.4
File Edit Shell Debug Options Window Help
Python 3.12.4 (tags/v3.12.4:8e8a4ba, Jun 6 2024, 19:30:16) [MSC v.1940 64 bit (AMD64)] on win32
Type "help", "copyright", "credits()" or "license()" for more information.
>>>
= RESTART: C:/Users/DINESH/mlq6.py
Initial Specific Hypothesis: ['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same']
Initial General Hypothesis: [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Example 1 processed
Specific Hypothesis: ['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same']
General Hypothesis: [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Example 2 processed
Specific Hypothesis: ['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']
General Hypothesis: [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Example 3 processed
Specific Hypothesis: ['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']
General Hypothesis: []

Example 4 processed
Specific Hypothesis: ['Sunny', 'Warm', '?', 'Strong', '?', '?']
General Hypothesis: []

Final Version Space:
Specific Hypothesis: ['Sunny', 'Warm', '?', 'Strong', '?', '?']
General Hypothesis: []
>>>
```

#07

```
import pandas as pd
```

```
from sklearn.datasets import load_iris
```

```
from sklearn.model_selection import train_test_split
```

```
from sklearn.linear_model import LogisticRegression
```

```
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

```
import seaborn as sns
```

```
import matplotlib.pyplot as plt
```

```
# Step 1: Load the Iris dataset
```

```
iris = load_iris()
```

```
X = pd.DataFrame(iris.data, columns=iris.feature_names)
```

```
y = pd.Series(iris.target)
```

```
# Step 2: 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)
```

```
# Step 3: Train the Logistic Regression model
```

```
log_reg = LogisticRegression(max_iter=200)
```

```
log_reg.fit(X_train, y_train)
```

```
# Step 4: Make predictions
```

```
y_pred = log_reg.predict(X_test)
```

```
# Step 5: Evaluate the model's performance
```

```
accuracy = accuracy_score(y_test, y_pred)
```

```
print(f'Accuracy: {accuracy:.2f}')
```

```
print('Classification Report:')
```

```
print(classification_report(y_test, y_pred, target_names=iris.target_names))
```

```
# Step 6: Visualize the results
```

```
# Confusion matrix
```

```
conf_matrix = confusion_matrix(y_test, y_pred)
```

```
plt.figure(figsize=(8, 6))
```

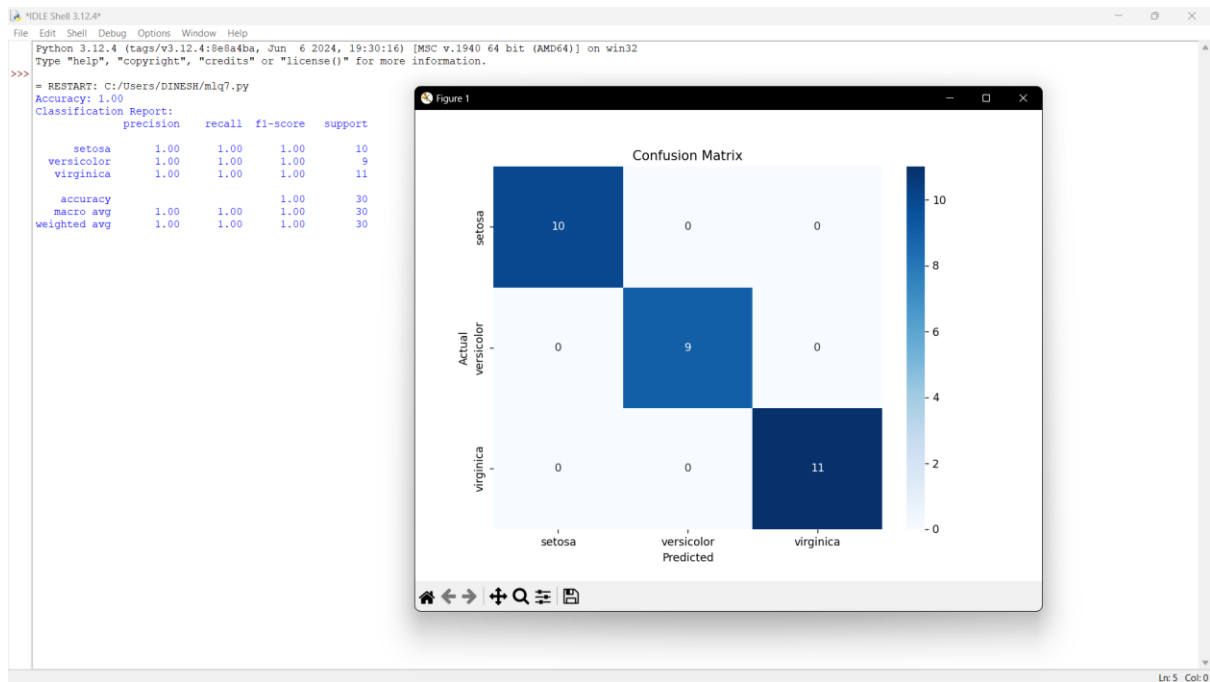
```
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=iris.target_names,  
yticklabels=iris.target_names)
```

```
plt.xlabel('Predicted')
```

```
plt.ylabel('Actual')
```

```
plt.title('Confusion Matrix')
```

```
plt.show()
```



#08

```
import numpy as np
```

```
import pandas as pd
```

```
import matplotlib.pyplot as plt
```

```
from sklearn.mixture import GaussianMixture
```

```
from sklearn.datasets import make_blobs
```

```
import seaborn as sns
```

```
# Step 1: Generate a synthetic dataset
```

```
X, y_true = make_blobs(n_samples=300, centers=3, cluster_std=0.60, random_state=0)
```

```
# Step 2: Initialize the Gaussian Mixture Model
```

```
gmm = GaussianMixture(n_components=3, random_state=42)
```

```
# Step 3: Fit the model using the EM algorithm
```

```
gmm.fit(X)
```

```
# Step 4: Predict cluster assignments
```

```
y_gmm = gmm.predict(X)
```

```
# Step 5: Visualize the results
```

```
plt.figure(figsize=(10, 6))
```

```
plt.scatter(X[:, 0], X[:, 1], c=y_gmm, s=40, cmap='viridis')
```

```
plt.scatter(gmm.means_[:, 0], gmm.means_[:, 1], s=300, c='red', marker='X') # Cluster centers
```

```
plt.title('Clusters found by Gaussian Mixture Model')
```

```
plt.xlabel('Feature 1')
```

```
plt.ylabel('Feature 2')
```

```
plt.show()
```

```
# Print the parameters of the Gaussian Mixture Model
```

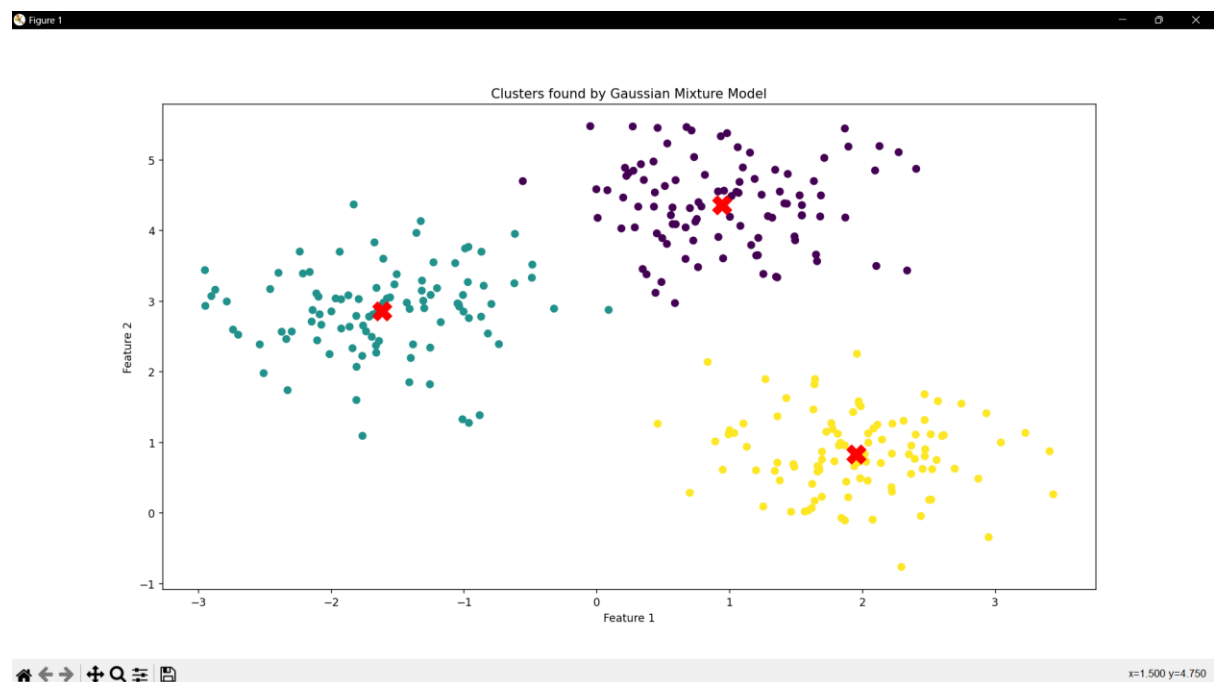
```
print('Means of the components:')
```

```
print(gmm.means_)
```

```
print('\nCovariances of the components:')
```

```
print(gmm.covariances_)
```

```
#output
```



#09


```

import pandas as pd

# Step 1: Create the dataset
data = {
    'Origin': ['Japan', 'Japan', 'Japan', 'USA', 'Japan'],
    'Manufacturer': ['Honda', 'Toyota', 'Toyota', 'Chrysler', 'Honda'],
    'Color': ['Blue', 'Green', 'Blue', 'Red', 'White'],
    'Decade': [1980, 1970, 1990, 1980, 1980],
    'Type': ['Economy', 'Sports', 'Economy', 'Economy', 'Economy'],
    'Example Type': ['Positive', 'Negative', 'Positive', 'Negative', 'Positive']
}

df = pd.DataFrame(data)

# Step 2: Initialize the most specific hypothesis
hypothesis = ['∅', '∅', '∅', '∅', '∅']

# Step 3: Find-S Algorithm
for index, row in df.iterrows():
    if row['Example Type'] == 'Positive':
        for i in range(len(hypothesis)):
            if hypothesis[i] == '∅':
                hypothesis[i] = row[i]
    elif hypothesis[i] != row[i]:
        hypothesis[i] = '?'

# Step 4: Output the most specific hypothesis
print("The most specific hypothesis is:", hypothesis)

#output

```

```
= RESTART: C:/Users/DINESH/mlq9.py
The most specific hypothesis is: ['Japan', 'Honda', '?', 1980, 'Economy']
|
```

#10

```
import pandas as pd
```

```
# Define the dataset
```

```
data = {
    'Origin': ['Japan', 'Japan', 'Japan', 'USA', 'Japan'],
    'Manufacturer': ['Honda', 'Toyota', 'Toyota', 'Chrysler', 'Honda'],
    'Color': ['Blue', 'Green', 'Blue', 'Red', 'White'],
    'Decade': ['1980', '1970', '1990', '1980', '1980'],
    'Type': ['Economy', 'Sports', 'Economy', 'Economy', 'Economy'],
    'Example Type': ['Positive', 'Negative', 'Positive', 'Negative', 'Positive']
}
```

```
# Create a DataFrame
```

```
df = pd.DataFrame(data)
```

```
# Function to implement Find-S algorithm
```

```
def find_s_algorithm(df):
```

```
    # Initialize the most specific hypothesis
```

```
    hypothesis = [' $\phi$ ] * (df.shape[1] - 1)
```

```
    # Iterate through the dataset
```

```
    for i in range(df.shape[0]):
```

```
        if df.iloc[i]['Example Type'] == 'Positive':
```

```
            if hypothesis == [' $\phi$ ] * (df.shape[1] - 1):
```

```
                hypothesis = list(df.iloc[i][:-1])
```

```
            else:
```

```
                for j in range(len(hypothesis)):
```

```

        if hypothesis[j] != df.iloc[i, j]:
            hypothesis[j] = '?'
    return hypothesis

# Apply the Find-S algorithm to the dataset
hypothesis = find_s_algorithm(df)

# Output the most specific hypothesis
print("The most specific hypothesis is:", hypothesis)

#output

#output

= RESTART: C:/Users/DINESH/mlq10.py
The most specific hypothesis is: ['Japan', '?', '?', '?', 'Economy']
|

```