DAY1-Q6

```
import pandas as pd
data=pd.read_csv('ex.txt')

def is_more_general(h1, h2):
      more_general_parts = []
      for x, y in zip(h1, h2):
         mg = x == '?' or (x != '0' and (x == y or y == '0'))
         more_general_parts.append(mg)
      return all(more_general_parts)

                              # Function to get the general boundary (G) and specific boundary (S)
def candidate_elimination(df):
                                  # Initialize G and S
   G = [['?' for _ in range(len(df.columns) - 1)]]
   S = [['0' for _ in range(len(df.columns) - 1)]]

                                                  # Iterate over the DataFrame
   for i in range(len(df)):
      instance = df.iloc[i, :-1].tolist()
      label = df.iloc[i, -1]

      if label == 'Yes':  # Positive example
         G = [g for g in G if is_more_general(g, instance)]
         for s in S:
            for index in range(len(s)):
                if s[index] != instance[index]:
                   s[index] = '?'
      else:  # Negative example
        S = [s for s in S if not is_more_general(s, instance)]
        new_G = []
        for g in G:
            for index in range(len(g)):
                if g[index] == '?':
                   for value in set(df.iloc[:, index]):
                          if value != instance[index]:
                              new_hypothesis = g.copy()
                              new_hypothesis[index] = value
                              if any(is_more_general(new_hypothesis, sg) for sg in S):
                                    new_G.append(new_hypothesis)
                                    G = new_G

   return S, G


S, G = candidate_elimination(df)
```

```
print("Specific boundary (S):", S)
print("General boundary (G):", G)
```

```
  Temp  Humidity     Wind  Water  Forecast  Enjoy Sport
     Warm    Normal   Strong   Warm      Same          Yes
     Warm      High   Strong   Warm      Same          Yes
     Cold      High   Strong   Warm    Change           No
```

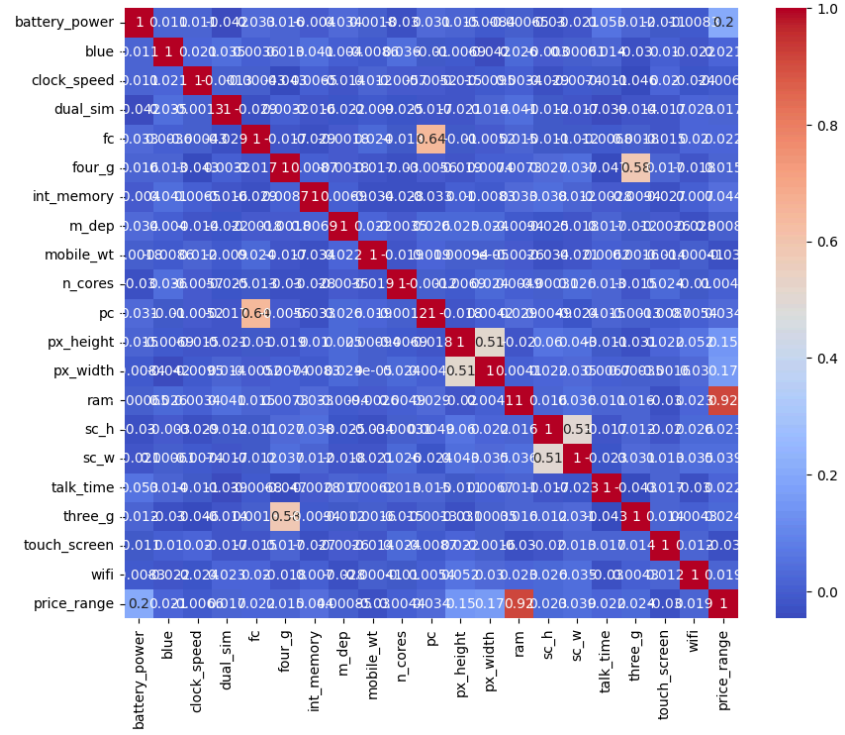
```
 Warm      High   Strong   Cool    Change          Yes
dary (S): [['0', '0', '0', '0', '0', '0']]
ary (G): [['Rainy', 'Warm', ' Normal', '?', ' Warm', '?'], ['Rainy', 'Warm', ' Normal', '?', '?', ' Same'], ['Rainy', 'Warm', ' Norma
```

DAY-1-Q1

```
import pandas as pd

# Read the dataset
data = pd.read_csv('mobileprice.csv')
# Print the first five rows of the dataset
print(data.head())
# Basic statistical computations
print(data.describe())
# Detect null values
print(data.isnull().sum())
# Columns and their data types


# Columns and their data types
print(data.info())
# Replace null values with mode value if any
for column in data.columns:
    if data[column].isnull().sum() > 0:
            mode_value = data[column].mode()[0]
            data[column].fillna(mode_value, inplace=True)
import seaborn as sns
import matplotlib.pyplot as plt

# Heatmap to explore correlations
plt.figure(figsize=(10, 8))
sns.heatmap(data.corr(), annot=True, cmap='coolwarm')
plt.show()
from sklearn.model_selection import train_test_split

# Features and target variable
X = data.drop('price_range', axis=1)
y = data['price_range']

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
from sklearn.naive_bayes import GaussianNB

# Initialize the model
nb_model = GaussianNB()

# Fit the model
nb_model.fit(X_train, y_train)
# Make predictions
y_pred = nb_model.predict(X_test)
from sklearn.metrics import accuracy_score

# Calculate the accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy of the model: {accuracy * 100:.2f}%")
```

```
4          1821     1          1.2         0  13       1          44    0.6

   mobile_wt  n_cores  ...  px_height  px_width   ram  sc_h  sc_w  talk_time  \
0        188        2  ...         20       756  2549     9     7         19
1        136        3  ...        905      1988  2631    17     3          7
2        145        5  ...       1263      1716  2603    11     2          9
3        131        6  ...       1216      1786  2769    16     8         11
4        141        2  ...       1208      1212  1411     8     2         15

   three_g  touch_screen  wifi  price_range
0        0             0     1            1
1        1             1     0            2
2        1             1     0            2
3        1             0     0            2
4        1             1     0            1

[5 rows x 21 columns]
       battery_power       blue  clock_speed     dual_sim           fc  \
count    2000.000000  2000.0000  2000.000000  2000.000000  2000.000000
mean     1238.518500     0.4950     1.522250     0.509500     4.309500
std       439.418206     0.5001     0.816004     0.500035     4.341444
min       501.000000     0.0000     0.500000     0.000000     0.000000
25%       851.750000     0.0000     0.700000     0.000000     1.000000
50%      1226.000000     0.0000     1.500000     1.000000     3.000000
75%      1615.250000     1.0000     2.200000     1.000000     7.000000
max      1998.000000     1.0000     3.000000     1.000000    19.000000

            four_g   int_memory        m_dep    mobile_wt      n_cores  ...  \
count  2000.000000  2000.000000  2000.000000  2000.000000  2000.000000  ...
mean      0.521500    32.046500     0.501750   140.249000     4.520500  ...
std       0.499662    18.145715     0.288416    35.399655     2.287837  ...
min       0.000000     2.000000     0.100000    80.000000     1.000000  ...
25%       0.000000    16.000000     0.200000   109.000000     3.000000  ...
50%       1.000000    32.000000     0.500000   141.000000     4.000000  ...
75%       1.000000    48.000000     0.800000   170.000000     7.000000  ...
max       1.000000    64.000000     1.000000   200.000000     8.000000  ...

         px_height     px_width          ram         sc_h         sc_w  \
count  2000.000000  2000.000000  2000.000000  2000.000000  2000.000000
mean    645.108000  1251.515500  2124.213000    12.306500     5.767000
std     443.780811   432.199447  1084.732044     4.213245     4.356398
min       0.000000   500.000000   256.000000     5.000000     0.000000
25%     282.750000   874.750000  1207.500000     9.000000     2.000000
50%     564.000000  1247.000000  2146.500000    12.000000     5.000000
75%     947.250000  1633.000000  3064.500000    16.000000     9.000000
max    1960.000000  1998.000000  3998.000000    19.000000    18.000000

         talk_time      three_g  touch_screen         wifi  price_range
count  2000.000000  2000.000000   2000.000000  2000.000000  2000.000000
mean     11.011000     0.761500      0.503000     0.507000     1.500000
std       5.463955     0.426273      0.500116     0.500076     1.118314
min       2.000000     0.000000      0.000000     0.000000     0.000000
25%       6.000000     1.000000      0.000000     0.000000     0.750000
50%      11.000000     1.000000      1.000000     1.000000     1.500000
75%      16.000000     1.000000      1.000000     1.000000     2.250000
max      20.000000     1.000000      1.000000     1.000000     3.000000

[8 rows x 21 columns]
battery_power    0
blue             0
clock_speed      0
dual_sim         0
fc               0
four_g           0
int_memory       0
m_dep            0
mobile_wt        0
n_cores          0
pc               0
px_height        0
px_width         0
ram              0
sc_h             0
sc_w             0
talk_time        0
three_g          0
touch_screen     0
wifi             0
price_range      0
dtype: int64
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2000 entries, 0 to 1999
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   battery_power  2000 non-null   int64
 1   blue           2000 non-null   int64
 2   clock_speed    2000 non-null   float64
 3   dual_sim       2000 non-null   int64
 4   fc             2000 non-null   int64
 5   four_g         2000 non-null   int64
```

```
 5   four_g          2000 non-null   int64
 6   int_memory      2000 non-null   int64
 7   m_dep           2000 non-null   float64
 8   mobile_wt       2000 non-null   int64
 9   n_cores         2000 non-null   int64
 10  pc              2000 non-null   int64
 11  px_height       2000 non-null   int64
 12  px_width        2000 non-null   int64
 13  ram             2000 non-null   int64
 14  sc_h            2000 non-null   int64
 15  sc_w            2000 non-null   int64
 16  talk_time       2000 non-null   int64
 17  three_g         2000 non-null   int64
 18  touch_screen    2000 non-null   int64
 19  wifi            2000 non-null   int64
 20  price_range     2000 non-null   int64
dtypes: float64(2), int64(19)
memory usage: 328.2 KB
None
```




```
Accuracy of the model: 79.75%
```

DAY1-Q2

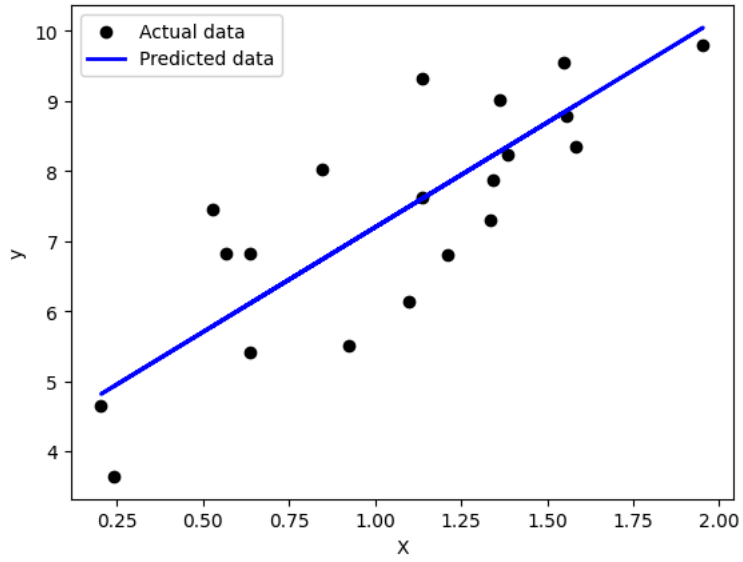
```
import pandas as pd

# Define the dataset
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']
}

                            # Convert the dataset to a DataFrame
df = pd.DataFrame(data)

                            # Function to find the most specific hypothesis using Find-S algorithm
def find_s_algorithm(df):
                                # Initialize the hypothesis
   hypothesis = ['0'] * (len(df.columns) - 1)

                                        # Iterate over the DataFrame
   for i in range(len(df)):
      if df.iloc[i, -1] == 'Yes':  # Only consider positive examples
          for j in range(len(hypothesis)):
                if hypothesis[j] == '0':
                    hypothesis[j] = df.iloc[i, j]
                elif hypothesis[j] != df.iloc[i, j]:
                    hypothesis[j] = '?'

   return hypothesis


hypothesis = find_s_algorithm(df)


print("The most specific hypothesis is:", hypothesis)
```

```
The most specific hypothesis is: ['Sunny', 'Warm', '?', 'Strong', '?', '?']
```

DAY1-Q3

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

# Generate a sample dataset
np.random.seed(0)
X = 2 * np.random.rand(100, 1)
y = 4 + 3 * X + np.random.randn(100, 1)

# Convert to DataFrame
data = pd.DataFrame(np.hstack((X, y)), columns=['X', 'y'])

# 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)

# Create and train the linear regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Calculate performance metrics
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# Print performance metrics
print(f"Mean Squared Error: {mse}")
print(f"R^2 Score: {r2}")

# Plot the results
plt.scatter(X_test, y_test, color='black', label='Actual data')
plt.plot(X_test, y_pred, color='blue', linewidth=2, label='Predicted data')
plt.xlabel('X')
plt.ylabel('y')
plt.legend()
plt.show()
```

```
Mean Squared Error: 0.9177532469714291
R^2 Score: 0.6521157503858556
```

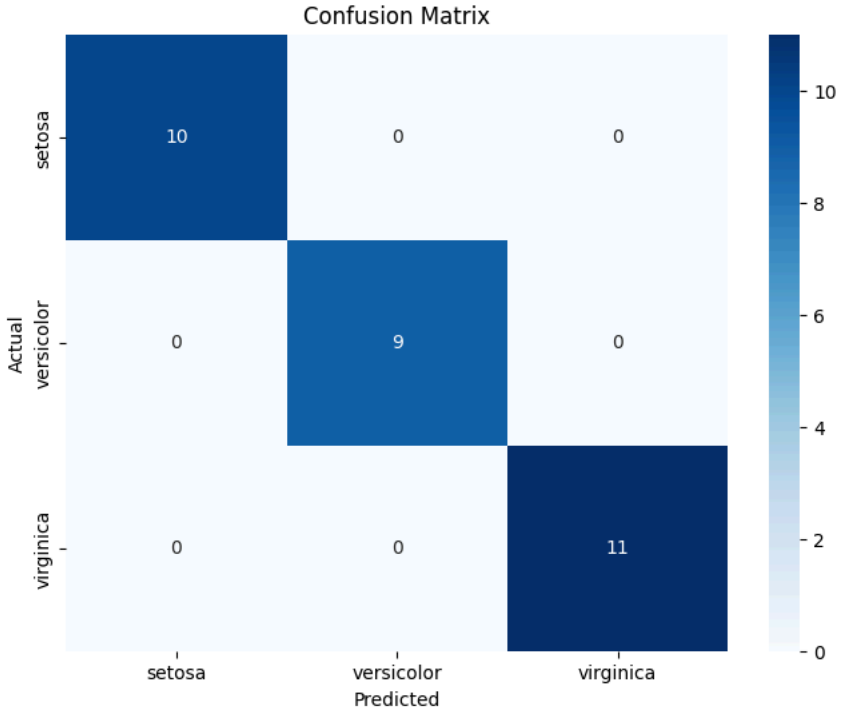


DAY1-Q4

```
import numpy as np
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 matplotlib.pyplot as plt
import seaborn as sns

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

```
y = iris.target

# 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)

# Create and train the KNN model
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)

# Make predictions
y_pred = knn.predict(X_test)

# Evaluate the model's performance
accuracy = accuracy_score(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)

# Print the evaluation metrics
print(f"Accuracy: {accuracy}")
print("Classification Report:")
print(classification_rep)
print("Confusion Matrix:")
print(conf_matrix)

# Plot the confusion matrix
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()
```

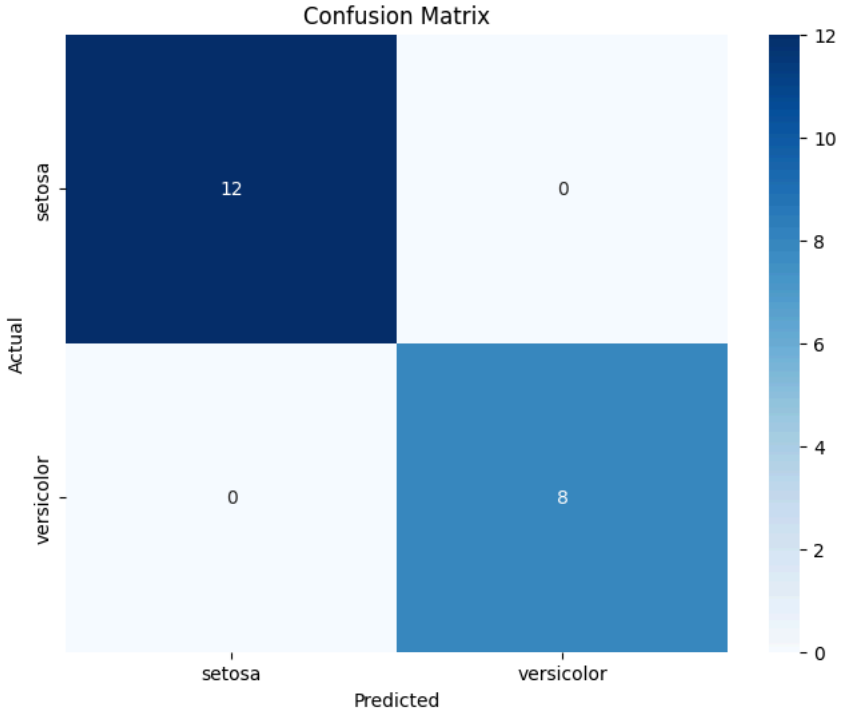
```
Accuracy: 1.0
Classification Report:
              precision    recall  f1-score   support

           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

Confusion Matrix:
[[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
```



DAY1-Q5

```
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.linear_model import Perceptron
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

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

# Since Perceptron is a binary classifier, we'll simplify the problem
# by selecting only two classes for this example.
# We'll use classes 0 and 1 (Setosa and Versicolour).
X = X[y != 2]
y = y[y != 2]

# 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
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Create and train the Perceptron model
perceptron = Perceptron(max_iter=1000, random_state=42)
perceptron.fit(X_train, y_train)

# Make predictions
y_pred = perceptron.predict(X_test)

# Evaluate the model's performance
accuracy = accuracy_score(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)

# Print the evaluation metrics
print(f"Accuracy: {accuracy}")
print("Classification Report:")
print(classification_rep)
print("Confusion Matrix:")
print(conf_matrix)

# Plot the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=iris.target_names[:2], yticklabels=iris.target_names[:2])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

```
Accuracy: 1.0
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        12
           1       1.00      1.00      1.00         8

    accuracy                           1.00        20
   macro avg       1.00      1.00      1.00        20
weighted avg       1.00      1.00      1.00        20

Confusion Matrix:
[[12  0]
 [ 0  8]]
```



DAY1-Q7

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, roc_auc_score, confusion_matrix, classification_report
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_iris

# Load the dataset
iris = load_iris()
X = iris.data
y = (iris.target == 2).astype(int)  # Convert to binary classification problem

# Split the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Train the model
model = LogisticRegression(solver='liblinear')
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)
y_pred_prob = model.predict_proba(X_test)[:, 1]

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
roc_auc = roc_auc_score(y_test, y_pred_prob)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)

print(f"Accuracy: {accuracy:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"ROC-AUC: {roc_auc:.2f}")
print("Confusion Matrix:")
print(conf_matrix)
print("Classification Report:")
print(class_report)

# Visualize the confusion matrix
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```