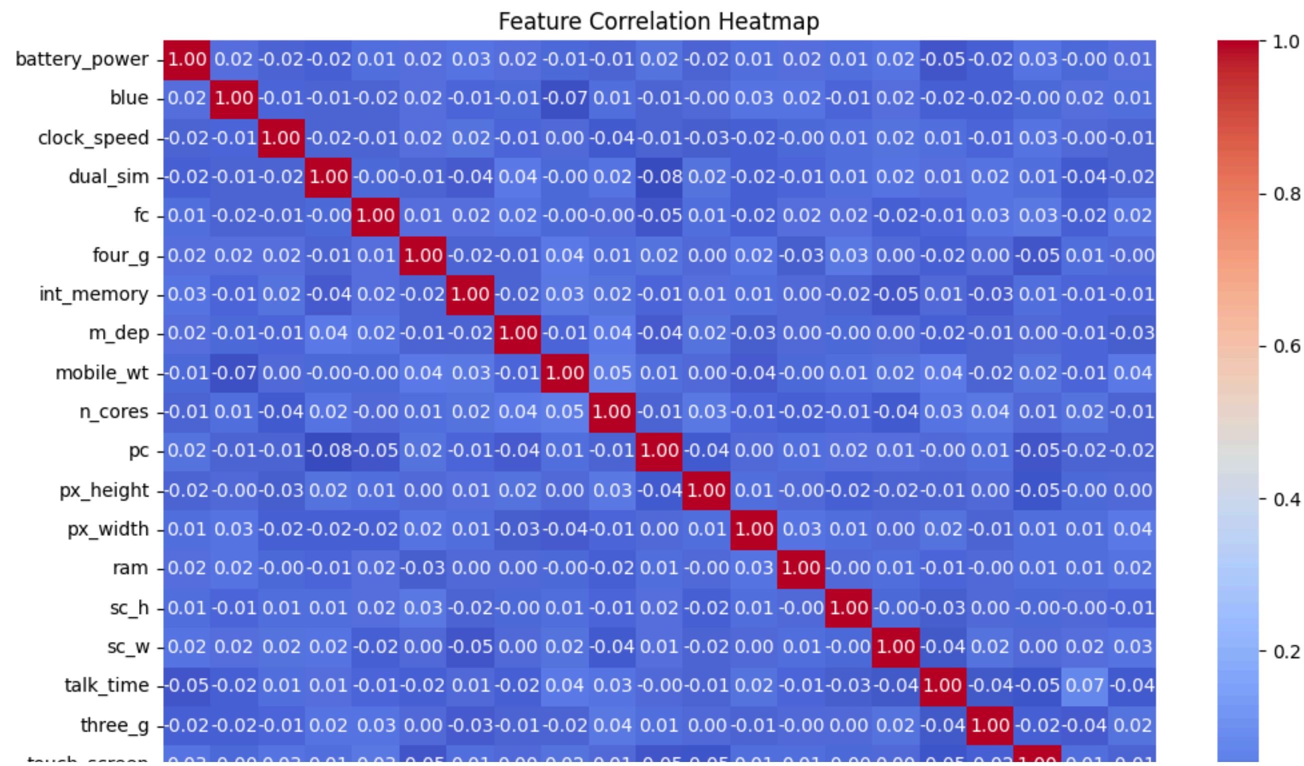
```
# Required Libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score

# Step 1: Generate a Synthetic Dataset
np.random.seed(42)
data = {
    'battery_power': np.random.randint(500, 4000, 2000),
    'blue': np.random.randint(0, 2, 2000),
    'clock_speed': np.random.uniform(0.5, 3.0, 2000),
    'dual_sim': np.random.randint(0, 2, 2000),
    'fc': np.random.randint(0, 20, 2000),
    'four_g': np.random.randint(0, 2, 2000),
    'int_memory': np.random.randint(2, 128, 2000),
    'm_dep': np.random.uniform(0.1, 1.0, 2000),
    'mobile_wt': np.random.randint(80, 250, 2000),
    'n_cores': np.random.randint(1, 9, 2000),
    'pc': np.random.randint(0, 20, 2000),
    'px_height': np.random.randint(0, 1960, 2000),
    'px_width': np.random.randint(500, 2000, 2000),
    'ram': np.random.randint(256, 8192, 2000),
    'sc_h': np.random.randint(5, 19, 2000),
    'sc_w': np.random.randint(0, 18, 2000),
    'talk_time': np.random.randint(2, 20, 2000),
    'three_g': np.random.randint(0, 2, 2000),
    'touch_screen': np.random.randint(0, 2, 2000),
    'wifi': np.random.randint(0, 2, 2000),
    'price_range': np.random.randint(0, 4, 2000)
}

df = pd.DataFrame(data)

# Step 2: Read the Mobile Price Dataset
# Already generated above as 'df'

# Step 3: Print the First Five Rows
print("First five rows of the dataset:")
print(df.head())

# Step 4: Basic Statistical Computations
print("\nBasic statistical description of the dataset:")
print(df.describe())

# Step 5: Columns and Their Data Types
print("\nColumns and their data types:")
print(df.dtypes)

# Step 6: Detect and Handle Null Values
print("\nChecking for null values:")
print(df.isnull().sum())

# In case of null values, fill with mode (here it's simulated, so there are no nulls)
# for column in df.columns:
#     df[column].fillna(df[column].mode()[0], inplace=True)

# Step 7: Data Exploration Using Heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(df.corr(), annot=True, fmt=".2f", cmap="coolwarm")
plt.title("Feature Correlation Heatmap")
plt.show()

# Step 8: Split the Data into Train and Test Sets
X = df.drop('price_range', axis=1)
y = df['price_range']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Step 9: Fit the Naive Bayes Classifier
nb_model = GaussianNB()
nb_model.fit(X_train, y_train)

# Step 10: Predict and Evaluate the Model
```

```
y_pred = nb_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)

print(f"\nAccuracy of the Naive Bayes model: {accuracy:.2f}")
```

```
First five rows of the dataset:
   battery_power  blue  clock_speed  dual_sim  fc  four_g  int_memory  \
0           3674     0     2.372066         0  10       0          57   
1           1360     0     2.127869         0  11       1          91   
2           1794     0     2.052321         0   4       0          96   
3           1630     1     1.380937         0  15       0         121   
4           1595     1     2.603619         1  12       0          52   

      m_dep  mobile_wt  n_cores  ...  px_height  px_width   ram  sc_h  sc_w  \
0  0.542688         96        7  ...        727      1909  6217    18    11   
1  0.117649        183        5  ...        627       693  8016     6    17   
2  0.704231        184        5  ...        117      1626  2653    11     2   
3  0.570842        183        6  ...        269      1899  3980    14     4   
4  0.908775        226        8  ...       1418       786  7453    14    13   

   talk_time  three_g  touch_screen  wifi  price_range  
0          4        0             1     1            1  
1          5        0             1     0            0  
2         19        1             1     1            1  
3         15        0             0     0            3  
4         16        0             1     1            2  

[5 rows x 21 columns]

Basic statistical description of the dataset:
       battery_power         blue  clock_speed     dual_sim           fc  \
count    2000.000000  2000.000000  2000.000000  2000.000000  2000.000000   
mean     2282.193000     0.524000     1.738788     0.503500     9.435000   
std      1026.460304     0.499549     0.721614     0.500113     5.753288   
min       501.000000     0.000000     0.500029     0.000000     0.000000   
25%      1398.250000     0.000000     1.127818     0.000000     4.000000   
50%      2278.500000     1.000000     1.732105     1.000000     9.000000   
75%      3231.500000     1.000000     2.357399     1.000000    15.000000   
max      3999.000000     1.000000     2.998894     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      0.479500    64.818000     0.550504   163.210500     4.539500  ...   
std       0.499705    36.576424     0.255575    49.845627     2.288899  ...   
min       0.000000     2.000000     0.100048    80.000000     1.000000  ...   
25%       0.000000    33.000000     0.332414   119.000000     3.000000  ...   
50%       0.000000    65.000000     0.551112   162.000000     5.000000  ...   
75%       1.000000    95.000000     0.767772   208.000000     7.000000  ...   
max       1.000000   127.000000     0.999555   249.000000     8.000000  ...   

         px_height     px_width          ram         sc_h         sc_w  \
count  2000.000000  2000.000000  2000.000000  2000.000000  2000.000000   
mean    979.965000  1263.289000  4280.538000    11.564000     8.463500   
std     567.443341   436.008547  2294.515552     4.125338     5.146932   
min       0.000000   502.000000   257.000000     5.000000     0.000000   
25%     482.000000   888.750000  2319.000000     8.000000     4.000000   
50%     977.000000  1266.000000  4253.500000    11.000000     8.000000   
75%    1472.000000  1635.000000  6319.750000    15.000000    13.000000   
max    1958.000000  1999.000000  8190.000000    18.000000    17.000000   

         talk_time      three_g  touch_screen         wifi  price_range  
count  2000.000000  2000.000000   2000.000000  2000.000000  2000.000000  
mean     10.571500     0.475000      0.506000     0.493500     1.488500  
std       5.171195     0.499499      0.500089     0.500083     1.111074  
min       2.000000     0.000000      0.000000     0.000000     0.000000  
25%       6.000000     0.000000      0.000000     0.000000     0.000000  
50%      11.000000     0.000000      1.000000     0.000000     2.000000  
75%      15.000000     1.000000      1.000000     1.000000     2.000000  
max      19.000000     1.000000      1.000000     1.000000     3.000000  

[8 rows x 21 columns]

Columns and their data types:
battery_power      int64
blue               int64
clock_speed      float64
dual_sim           int64
fc                 int64
four_g             int64
int_memory         int64
m_dep            float64
mobile_wt          int64
n_cores            int64
pc                 int64
px_height          int64
px_width           int64
ram                int64
sc_h               int64
sc_w               int64
```

```
talk_time          int64
three_g            int64
touch_screen       int64
wifi               int64
price_range        int64
dtype: object

Checking for null values:
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
```

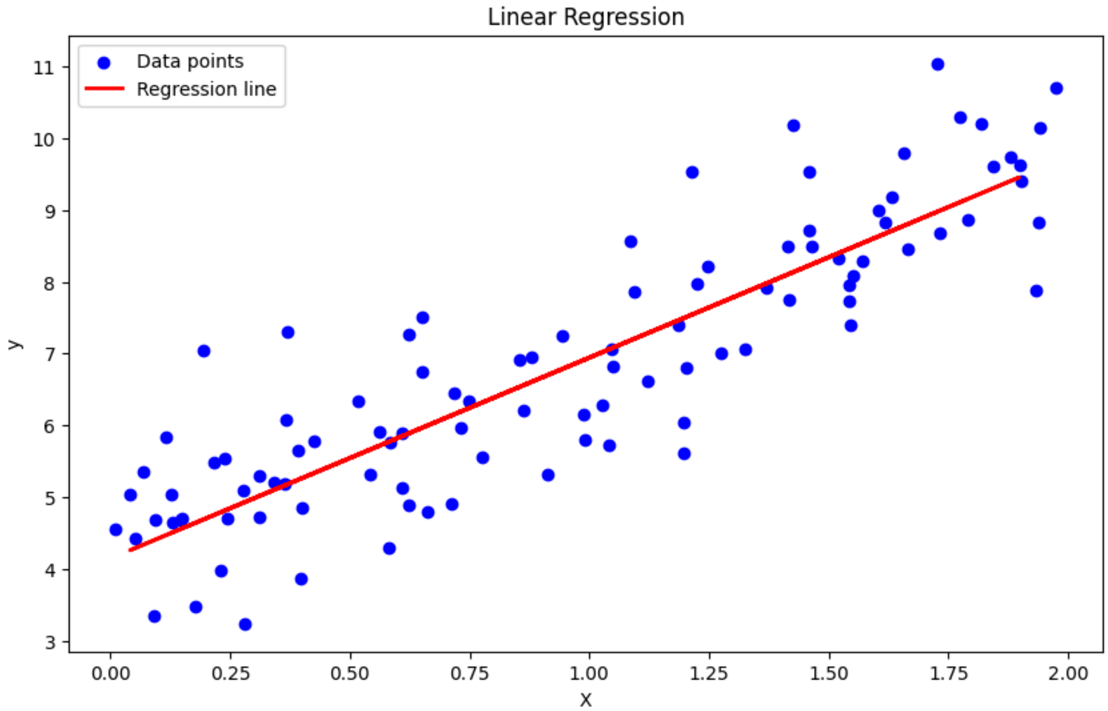


```
# Step 1: Define the dataset
data = [
    ['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same', 'Yes'],
    ['Sunny', 'Warm', 'High', 'Strong', 'Warm', 'Same', 'Yes'],
    ['Rainy', 'Cold', 'High', 'Strong', 'Warm', 'Change', 'No'],
    ['Sunny', 'Warm', 'High', 'Strong', 'Cool', 'Change', 'Yes']
]

# Step 2: Initialize the most specific hypothesis
hypothesis = ['0'] * (len(data[0]) - 1)  # Exclude the target value ('Enjoy Sport')

# Step 3: Apply Find-S algorithm
for example in data:
    if example[-1] == 'Yes':  # Only consider positive examples
        for i in range(len(hypothesis)):
            if hypothesis[i] == '0':  # If the hypothesis is '0', set it to the attribute value
                hypothesis[i] = example[i]
            elif hypothesis[i] != example[i]:  # If there's a conflict, set it to '?'
                hypothesis[i] = '?'

# Step 4: Print the final hypothesis
print("The most specific hypothesis is:")
print(hypothesis)
```

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

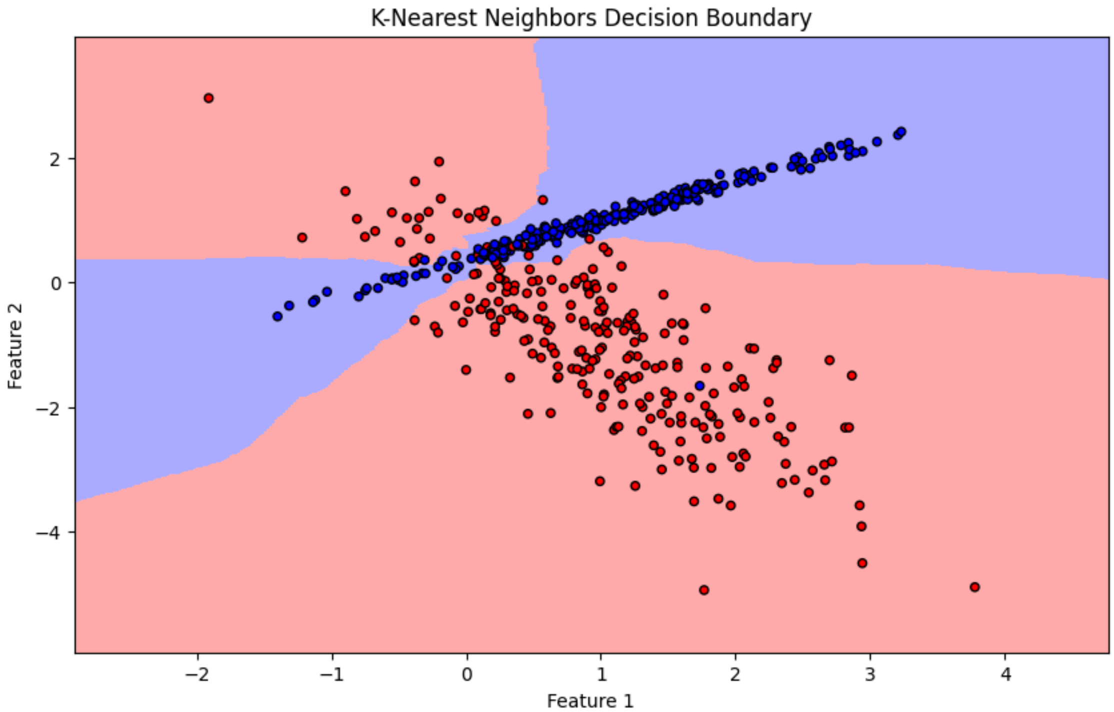
```
# Required Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Step 1: Generate a Synthetic Dataset
np.random.seed(42)
X = 2 * np.random.rand(100, 1)
y = 4 + 3 * X + np.random.randn(100, 1)

# Convert to pandas DataFrame for easier manipulation
data = pd.DataFrame(data=np.hstack((X, y)), columns=["X", "y"])

# 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: Fit a Linear Regression Model
lin_reg = LinearRegression()
lin_reg.fit(X_train, y_train)

# Step 4: Predict the Target Values for the Test Set
y_pred = lin_reg.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}")

# Plotting the results
plt.figure(figsize=(10, 6))
plt.scatter(X, y, color="blue", label="Data points")
plt.plot(X_test, y_pred, color="red", linewidth=2, label="Regression line")
plt.xlabel("X")
plt.ylabel("y")
plt.title("Linear Regression")
plt.legend()
plt.show()
```

```
Mean Squared Error: 0.65
R^2 Score: 0.81
```



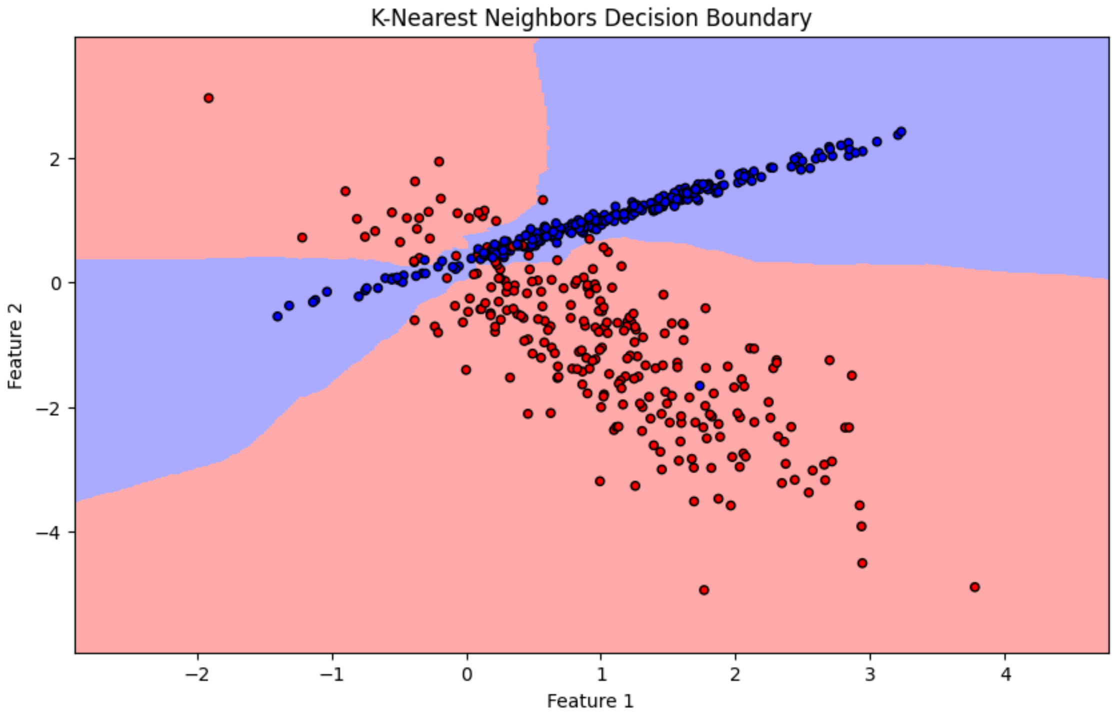
```
# Required Libraries
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap

# Step 1: Generate a Synthetic Dataset
from sklearn.datasets import make_classification

X, y = make_classification(n_samples=500, n_features=2, n_informative=2, n_redundant=0, n_clusters_per_class=1, random_state=42)

# Convert to pandas DataFrame for easier manipulation
data = pd.DataFrame(data=np.hstack((X, y.reshape(-1, 1))), columns=["Feature 1", "Feature 2", "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: Fit a KNN Classifier
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)

# Step 4: Predict the Target Values for the Test Set
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))
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

# Plotting the decision boundary
h = .02  # step size in the mesh
cmap_light = ListedColormap(['#FFAAAA', '#AAAAFF'])
cmap_bold = ListedColormap(['#FF0000', '#0000FF'])

# Create a mesh to plot the decision boundary
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))

# Predict class for each point in mesh
Z = knn.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

# Plot the decision boundary and training points
plt.figure(figsize=(10, 6))
plt.pcolormesh(xx, yy, Z, cmap=cmap_light)
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=cmap_bold, edgecolor='k', s=20)
plt.xlim(xx.min(), xx.max())
plt.ylim(yy.min(), yy.max())
plt.title("K-Nearest Neighbors Decision Boundary")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.show()
```

```
Accuracy: 0.97
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.94      0.97        51
           1       0.94      1.00      0.97        49

    accuracy                           0.97       100
   macro avg       0.97      0.97      0.97       100
weighted avg       0.97      0.97      0.97       100

Confusion Matrix:
[[48  3]
 [ 0 49]]
```



```
# Required Libraries
import numpy as np
import pandas as pd
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
from matplotlib.colors import ListedColormap

# Step 1: Generate a Synthetic Dataset
from sklearn.datasets import make_classification

X, y = make_classification(n_samples=500, n_features=2, n_informative=2, n_redundant=0, n_clusters_per_class=1, random_state=42)

# Convert to pandas DataFrame for easier manipulation
data = pd.DataFrame(data=np.hstack((X, y.reshape(-1, 1))), columns=["Feature 1", "Feature 2", "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: Fit a KNN Classifier
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)

# Step 4: Predict the Target Values for the Test Set
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))
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

# Plotting the decision boundary
h = .02  # step size in the mesh
cmap_light = ListedColormap(['#FFAAAA', '#AAAAFF'])
cmap_bold = ListedColormap(['#FF0000', '#0000FF'])

# Create a mesh to plot the decision boundary
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))

# Predict class for each point in mesh
Z = knn.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

# Plot the decision boundary and training points
plt.figure(figsize=(10, 6))
plt.pcolormesh(xx, yy, Z, cmap=cmap_light)
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=cmap_bold, edgecolor='k', s=20)
plt.xlim(xx.min(), xx.max())
plt.ylim(yy.min(), yy.max())
plt.title("K-Nearest Neighbors Decision Boundary")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.show()
```

```
Accuracy: 0.97
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.94      0.97        51
           1       0.94      1.00      0.97        49

    accuracy                           0.97       100
   macro avg       0.97      0.97      0.97       100
weighted avg       0.97      0.97      0.97       100

Confusion Matrix:
[[48  3]
 [ 0 49]]
```



If you're already familiar with Colab, check out this video to learn about interactive tables, the executed code history view and the command palette.

```
import numpy as np
import pandas as pd

# Step 1: Define the dataset
data = [
    ['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same', 'Yes'],
    ['Sunny', 'Warm', 'High', 'Strong', 'Warm', 'Same', 'Yes'],
    ['Rainy', 'Cold', 'High', 'Strong', 'Warm', 'Change', 'No'],
    ['Sunny', 'Warm', 'High', 'Strong', 'Cool', 'Change', 'Yes']
]

# Convert to pandas DataFrame
columns = ["Sky", "Air Temp", "Humidity", "Wind", "Water", "Forecast", "Enjoy Sport"]
df = pd.DataFrame(data, columns=columns)

# Step 2: Initialize the most specific hypothesis S0 and most general hypothesis G0
num_attributes = len(df.columns) - 1
S = ['0'] * num_attributes  # Most specific hypothesis
G = [['?'] * num_attributes]  # Most general hypothesis

# Step 3: Candidate Elimination Algorithm
def consistent(hypothesis, example):
    for i in range(len(hypothesis)):
        if hypothesis[i] != '?' and hypothesis[i] != example[i]:
            return False
    return True

for i, row in df.iterrows():
    example = row[:-1].tolist()
    target = row[-1]

    if target == 'Yes':  # Positive example
        for j in range(len(S)):
            if S[j] == '0':
                S[j] = example[j]
            elif S[j] != example[j]:
                S[j] = '?'

        G = [g for g in G if consistent(g, example)]
    else:  # Negative example
        G_temp = []
        for g in G:
            for j in range(num_attributes):
                if g[j] == '?':
                    for value in df.iloc[:, j].unique():
                        if value != example[j]:
                            new_g = g[:]
                            new_g[j] = value
                            if consistent(S, new_g):
                                G_temp.append(new_g)
                elif g[j] != example[j]:
                    new_g = g[:]
                    new_g[j] = '?'
                    if consistent(S, new_g):
                        G_temp.append(new_g)
        G = G_temp

    G = [g for g in G if any(consistent(g, row[:-1].tolist()) for _, row in df.iterrows())]

print("Final Specific Hypothesis S:")
print(S)
print("Final General Hypotheses G:")
for g in G:
    print(g)
```

```
Final Specific Hypothesis S:
['Sunny', 'Warm', '?', 'Strong', '?', '?']
Final General Hypotheses G:
```

```python
# Required Libraries
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 LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Step 1: Load the Dataset
iris = load_iris()
X = iris.data
y = iris.target

# Convert to binary classification: 1 if Iris-setosa, 0 otherwise
y = (y == 0).astype(int)  # Setosa is class 0 in the original dataset

# Step 2: Preprocess the Data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.3, random_state=42)

# Step 3: Fit a Logistic Regression Model
log_reg = LogisticRegression(random_state=42)
log_reg.fit(X_train, y_train)

# Step 4: Predict the Target Values for the Test Set
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}")

# Generate a classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))

# Generate a confusion matrix
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

```
Accuracy: 1.00
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        26
           1       1.00      1.00      1.00        19

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45

Confusion Matrix:
[[26  0]
 [ 0 19]]
```

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.mixture import GaussianMixture

# Generate synthetic data
np.random.seed(42)
n_samples = 500

# Generate random samples from two different normal distributions
shifted_gaussian = np.random.randn(n_samples, 2) + np.array([5, 5])
stretched_gaussian = np.dot(np.random.randn(n_samples, 2), np.array([[0.6, -0.6], [-0.4, 0.8]]))

X = np.vstack([shifted_gaussian, stretched_gaussian])

# Plot the synthetic data
plt.figure(figsize=(8, 6))
plt.scatter(X[:, 0], X[:, 1], s=5, alpha=0.5)
plt.title('Synthetic Data for GMM')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()

# Step 1: Initialize the Gaussian Mixture Model
gmm = GaussianMixture(n_components=2, max_iter=100, random_state=42)

# Step 2: Fit the model to the data
gmm.fit(X)

# Step 3: Predict the cluster for each data point
labels = gmm.predict(X)

# Step 4: Extract the parameters
weights = gmm.weights_
means = gmm.means_
covariances = gmm.covariances_

print("Weights:", weights)
print("Means:", means)
print("Covariances:", covariances)
```