```python
# Importing necessary 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, confusion_matrix, classification_report

# Step (a): Read the Mobile price dataset using the Pandas module
data = pd.read_csv('/path/to/your/mobile_price_data.csv')  # Update the path to your dataset

# Step (b): Print the 1st five rows
print("First five rows of the dataset:")
print(data.head())

# Step (c): Basic statistical computations on the data set or distribution of data
print("\nBasic statistical description of the dataset:")
print(data.describe())

# Step (d): The columns and their data types
print("\nColumns and their data types:")
print(data.dtypes)

# Step (e): Detects null values in the dataset. If there is any null value, replace it with mode value
print("\nChecking for null values:")
print(data.isnull().sum())

# If there are null values, replace them with the mode of the respective columns
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("\nNull values after replacement:")
print(data.isnull().sum())

# Step (f): Explore the data set using heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(data.corr(), annot=True, cmap='coolwarm')
plt.title("Heatmap of Correlation Matrix")
plt.show()

# Step (g): Split the data into test and train
X = data.drop('price_range', axis=1)  # Features
y = data['price_range']  # Target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Step (h): Fit into the model Naive Bayes Classifier
model = GaussianNB()
model.fit(X_train, y_train)

# Step (i): Predict the model
y_pred = model.predict(X_test)

# Step (j): Find the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print("\nAccuracy of the Naive Bayes Classifier model:")
print(accuracy)

# Additional metrics
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

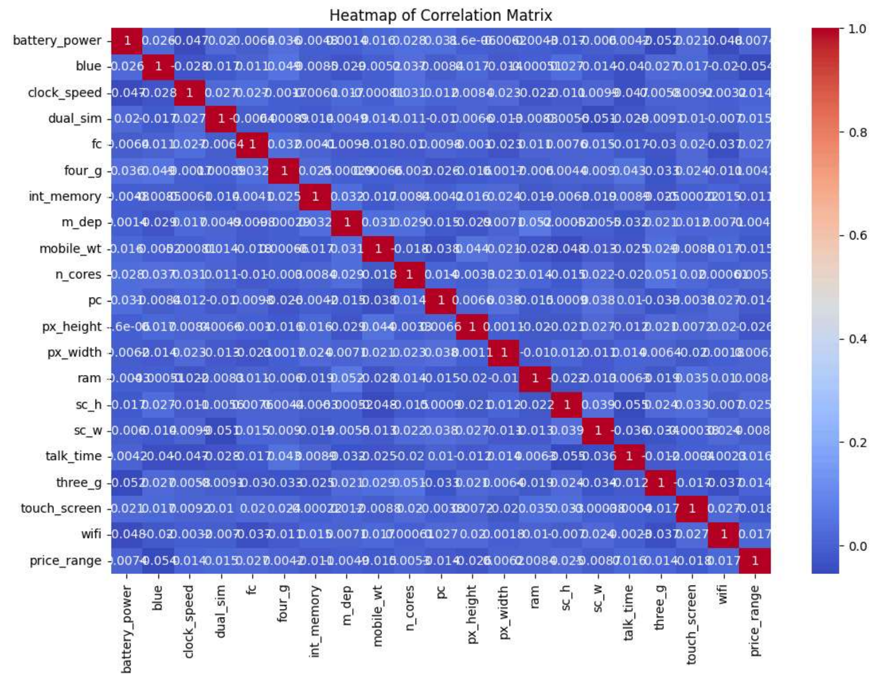
```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-1-2b1e8ad946ae> in <cell line: 11>()
      9 
     10 # Step (a): Read the Mobile price dataset using the Pandas module
---> 11 data = pd.read_csv('/path/to/your/mobile_price_data.csv')  # Update the path to
your dataset
     12 
     13 # Step (b): Print the 1st five rows

                              4 frames
/usr/local/lib/python3.10/dist-packages/pandas/io/common.py in get_handle(path_or_buf,
mode, encoding, compression, memory_map, is_text, errors, storage_options)
    857         if ioargs.encoding and "b" not in ioargs.mode:
    858             # Encoding
--> 859             handle = open(
    860                 handle,
    861                 ioargs.mode,

FileNotFoundError: [Errno 2] No such file or directory:
```

Next steps: Explain error

```
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, confusion_matrix, classification_report

# Generating a mock dataset
np.random.seed(42)
data = pd.DataFrame({
    'battery_power': np.random.randint(500, 2000, 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, 64, 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, 4000, 2000),
    'sc_h': np.random.randint(5, 20, 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)
})

# Step (b): Print the 1st five rows
print("First five rows of the dataset:")
print(data.head())

# Step (c): Basic statistical computations on the data set or distribution of data
print("\nBasic statistical description of the dataset:")
print(data.describe())

# Step (d): The columns and their data types
print("\nColumns and their data types:")
print(data.dtypes)

# Step (e): Detects null values in the dataset. If there is any null value, replace it with mode value
print("\nChecking for null values:")
print(data.isnull().sum())

# If there are null values, replace them with the mode of the respective columns
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("\nNull values after replacement:")
print(data.isnull().sum())

# Step (f): Explore the data set using heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(data.corr(), annot=True, cmap='coolwarm')
plt.title("Heatmap of Correlation Matrix")
plt.show()

# Step (g): Split the data into test and train
X = data.drop('price_range', axis=1)  # Features
y = data['price_range']  # Target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Step (h): Fit into the model Naive Bayes Classifier
model = GaussianNB()
model.fit(X_train, y_train)

# Step (i): Predict the model
y_pred = model.predict(X_test)
```

```
# Step (j): Find the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print("\nAccuracy of the Naive Bayes Classifier model:")
print(accuracy)

# Additional metrics
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

```
talk_time        0
three_g          0
touch_screen     0
wifi             0
price_range      0
dtype: int64
```



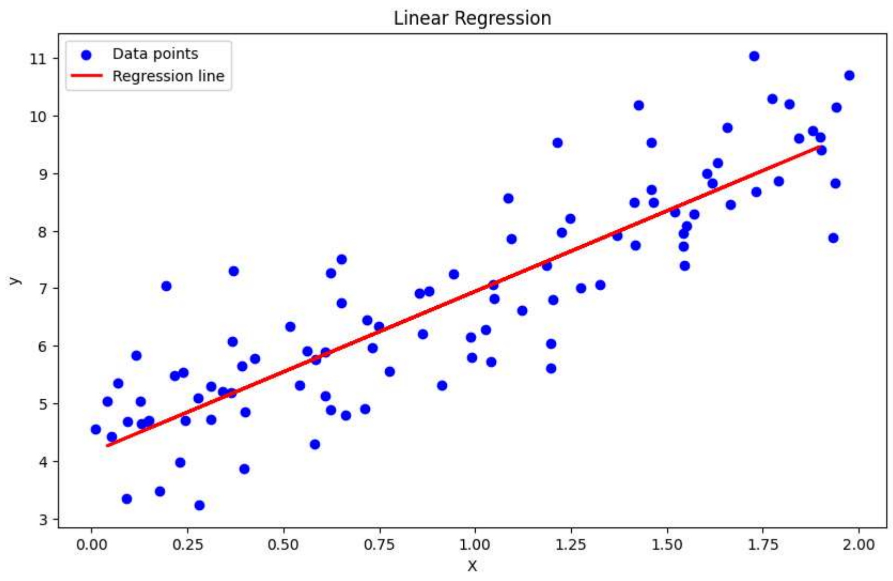
```
Accuracy of the Naive Bayes Classifier model:
0.23

Confusion Matrix:
[[47 52 15 40]
 [45 43 15 36]
 [57 65 23 26]
 [52 51  8 25]]

Classification Report:
              precision    recall  f1-score   support

           0       0.23      0.31      0.26       154
           1       0.20      0.31      0.25       139
           2       0.38      0.13      0.20       171
           3       0.20      0.18      0.19       136

    accuracy                           0.23       600
   macro avg       0.25      0.23      0.22       600
weighted avg       0.26      0.23      0.22       600
```

```
import numpy as np
import pandas as pd

# Defining the dataset
data = pd.DataFrame({
    'Sky': ['Sunny', 'Sunny', 'Rainy', 'Sunny'],
    'AirTemp': ['Warm', 'Warm', 'Cold', 'Warm'],
    'Humidity': ['Normal', 'High', 'High', 'High'],
    'Wind': ['Strong', 'Strong', 'Strong', 'Strong'],
    'Water': ['Warm', 'Warm', 'Warm', 'Cool'],
    'Forecast': ['Same', 'Same', 'Change', 'Change'],
    'EnjoySport': ['Yes', 'Yes', 'No', 'Yes']
})

print("Dataset:")
print(data)

# Step 1: Initialize the most specific hypothesis
hypothesis = ['0'] * (len(data.columns) - 1)
print("\nInitial hypothesis:", hypothesis)

# Step 2: Iterate through the dataset to update the hypothesis
for i in range(len(data)):
    if data['EnjoySport'][i] == 'Yes':
        for j in range(len(hypothesis)):
            if hypothesis[j] == '0':
                hypothesis[j] = data.iloc[i, j]
            elif hypothesis[j] != data.iloc[i, j]:
                hypothesis[j] = '?'

print("\nFinal hypothesis after applying Find-S algorithm:")
print(hypothesis)
```

```
Dataset:
     Sky AirTemp Humidity    Wind Water Forecast EnjoySport
0  Sunny    Warm   Normal  Strong  Warm     Same        Yes
1  Sunny    Warm     High  Strong  Warm     Same        Yes
2  Rainy    Cold     High  Strong  Warm   Change         No
3  Sunny    Warm     High  Strong  Cool   Change        Yes

Initial hypothesis: ['0', '0', '0', '0', '0', '0']

Final hypothesis after applying Find-S algorithm:
['Sunny', 'Warm', '?', 'Strong', '?', '?']
```

```
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

# Generating a sample dataset
np.random.seed(42)
X = 2 * np.random.rand(100, 1)
y = 4 + 3 * X + np.random.randn(100, 1)

# Converting to DataFrame for better visualization (optional)
data = pd.DataFrame({'X': X.flatten(), 'y': y.flatten()})
print("Sample Dataset:")
print(data.head())

# Splitting 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)

# Fitting a Linear Regression model
model = LinearRegression()
model.fit(X_train, y_train)

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

# Evaluating the model's performance
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("\nMean Squared Error:", mse)
print("R-squared:", r2)

# 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()
```

```
Sample Dataset:
          X         y
0  0.749080  6.334288
1  1.901429  9.405278
2  1.463988  8.483724
3  1.197317  5.604382
4  0.312037  4.716440

Mean Squared Error: 0.6536995137170021
R-squared: 0.8072059636181392
```



```
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

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

# Convert to DataFrame for better visualization (optional)
data = pd.DataFrame(data=np.c_[X, y], columns=iris.feature_names + ['target'])
print("Iris Dataset:")
print(data.head())

# 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)

# Fit a KNN model
k = 3  # Number of neighbors
knn = KNeighborsClassifier(n_neighbors=k)
knn.fit(X_train, y_train)

# Predict the model on test set
y_pred = knn.predict(X_test)

# Evaluate the model's performance
accuracy = accuracy_score(y_test, y_pred)
print("\nAccuracy of the KNN model:", accuracy)

# Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("\nConfusion Matrix:")
print(conf_matrix)
```

```
# Classification Report
class_report = classification_report(y_test, y_pred, target_names=iris.target_names)
print("\nClassification Report:")
print(class_report)
```

```
Iris Dataset:
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  \
0                5.1               3.5                1.4               0.2   
1                4.9               3.0                1.4               0.2   
2                4.7               3.2                1.3               0.2   
3                4.6               3.1                1.5               0.2   
4                5.0               3.6                1.4               0.2   

   target  
0     0.0  
1     0.0  
2     0.0  
3     0.0  
4     0.0  

Accuracy of the KNN model: 1.0

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

Classification Report:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        10
  versicolor       1.00      1.00      1.00         9
   virginica       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
```