```
import pandas as pd
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import confusion_matrix, classification_report

# Load breast cancer dataset from sklearn
data = load_breast_cancer()

# Convert to DataFrame
df = pd.DataFrame(data.data, columns=data.feature_names)
df['target'] = data.target  # Adding the target column (0 or 1)

# (a) Print the first five rows
print("First five rows of the dataset:")
print(df.head())

# (b) Basic statistical computations
print("\nBasic statistical computations:")
print(df.describe())

# (c) Columns and their data types
print("\nColumns and their data types:")
print(df.dtypes)

# Check for null values
print("\nChecking for null values:")
print(df.isnull().sum())

# Replace null values with mode (if any)
mode_values = df.mode().iloc[0]
df = df.fillna(mode_values)

# Separate features (X) and target (y)
X = df.drop('target', axis=1)
y = df['target']

# Split data into train and test sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize Gaussian Naive Bayes classifier
nb_classifier = GaussianNB()

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

# Predictions on test set
y_pred = nb_classifier.predict(X_test)

# Confusion matrix
cm = confusion_matrix(y_test, y_pred)
print("\nConfusion Matrix:")
print(cm)

# Additional metrics (precision, recall, F1-score)
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

```
First five rows of the dataset:
   mean radius  mean texture  mean perimeter  mean area  mean smoothness  \
0        17.99         10.38          122.80     1001.0          0.11840
1        20.57         17.77          132.90     1326.0          0.08474
2        19.69         21.25          130.00     1203.0          0.10960
3        11.42         20.38           77.58      386.1          0.14250
4        20.29         14.34          135.10     1297.0          0.10030

   mean compactness  mean concavity  mean concave points  mean symmetry  \
0           0.27760          0.3001              0.14710         0.2419
1           0.07864          0.0869              0.07017         0.1812
2           0.15990          0.1974              0.12790         0.2069
3           0.28390          0.2414              0.10520         0.2597
4           0.13280          0.1980              0.10430         0.1809

   mean fractal dimension  ...  worst texture  worst perimeter  worst area  \
0                 0.07871  ...          17.33           184.60      2019.0
1                 0.05667  ...          23.41           158.80      1956.0
```

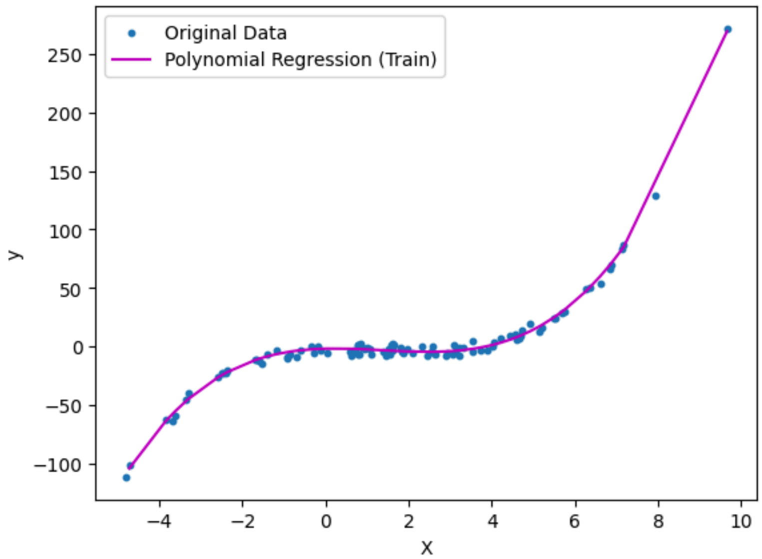
```
2                 0.05999  ...          25.53           152.50      1709.0
3                 0.09744  ...          26.50            98.87       567.7
4                 0.05883  ...          16.67           152.20      1575.0

   worst smoothness  worst compactness  worst concavity  worst concave points  \
0            0.1622             0.6656           0.7119                0.2654
1            0.1238             0.1866           0.2416                0.1860
2            0.1444             0.4245           0.4504                0.2430
3            0.2098             0.8663           0.6869                0.2575
4            0.1374             0.2050           0.4000                0.1625

   worst symmetry  worst fractal dimension  target
0          0.4601                  0.11890       0
1          0.2750                  0.08902       0
2          0.3613                  0.08758       0
3          0.6638                  0.17300       0
4          0.2364                  0.07678       0

[5 rows x 31 columns]

Basic statistical computations:
       mean radius  mean texture  mean perimeter    mean area  \
count   569.000000    569.000000      569.000000   569.000000
mean     14.127292     19.289649       91.969033   654.889104
std       3.524049      4.301036       24.298981   351.914129
min       6.981000      9.710000       43.790000   143.500000
25%      11.700000     16.170000       75.170000   420.300000
50%      13.370000     18.840000       86.240000   551.100000
75%      15.780000     21.800000      104.100000   782.700000
max      28.110000     39.280000      188.500000  2501.000000

       mean smoothness  mean compactness  mean concavity  mean concave points  \
count       569.000000        569.000000      569.000000           569.000000
mean          0.096360          0.104341        0.088799             0.048919
std           0.014064          0.052813        0.079720             0.038803
min           0.052630          0.019380        0.000000             0.000000
25%           0.086370          0.064920        0.029560             0.020310
50%           0.095870          0.092630        0.061540             0.033500
75%           0.105300          0.130400        0.130700             0.074000
max           0.163400          0.345400        0.426800             0.201200
```

```
import pandas as pd

# Define the dataset
data = {
    'Size': ['Big', 'Small', 'Small', 'Big', 'Small'],
    'Color': ['Red', 'Red', 'Red', 'Blue', 'Blue'],
    'Shape': ['Circle', 'Triangle', 'Circle', 'Circle', 'Circle'],
    'Class': ['No', 'No', 'Yes', 'No', 'Yes']
}

# Convert to DataFrame
df = pd.DataFrame(data)

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

    # Iterate over the examples
    for index, row in df.iterrows():
        if row['Class'] == 'Yes':
            for i in range(len(hypothesis)):
                if hypothesis[i] == '0':  # If the hypothesis is at its initial state
                    hypothesis[i] = row[i]
                elif hypothesis[i] != row[i]:  # Generalize if the attribute value is different
                    hypothesis[i] = '?'
    return hypothesis

# Apply the Find-S algorithm
most_specific_hypothesis = find_s_algorithm(df)

# Show the output
print("The most specific hypothesis found by Find-S algorithm is:")
print(most_specific_hypothesis)
```

```
The most specific hypothesis found by Find-S algorithm is:
['Small', '?', 'Circle']
```

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

# Generate synthetic data
np.random.seed(0)
X = 2 - 3 * np.random.normal(0, 1, 100)
y = X - 2 * (X ** 2) + 0.5 * (X ** 3) + np.random.normal(-3, 3, 100)

X = X[:, np.newaxis]

# 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=0)

# Transform the features to polynomial features
polynomial_features = PolynomialFeatures(degree=3)
X_train_poly = polynomial_features.fit_transform(X_train)
X_test_poly = polynomial_features.transform(X_test)

# Train the Polynomial Regression model
model = LinearRegression()
model.fit(X_train_poly, y_train)

# Predict using the model
y_train_pred = model.predict(X_train_poly)
y_test_pred = model.predict(X_test_poly)

# Evaluate the model
mse_train = mean_squared_error(y_train, y_train_pred)
mse_test = mean_squared_error(y_test, y_test_pred)
r2_train = r2_score(y_train, y_train_pred)
r2_test = r2_score(y_test, y_test_pred)

print("Training set performance:")
print(f"Mean Squared Error (MSE): {mse_train}")
print(f"R² score: {r2_train}")

print("\nTest set performance:")
print(f"Mean Squared Error (MSE): {mse_test}")
print(f"R² score: {r2_test}")

# Visualize the Polynomial Regression results
plt.scatter(X, y, s=10, label='Original Data')
# Sort the values for plotting
sorted_zip = sorted(zip(X_train, y_train_pred))
X_train_sorted, y_train_pred_sorted = zip(*sorted_zip)
plt.plot(X_train_sorted, y_train_pred_sorted, color='m', label='Polynomial Regression (Train)')
plt.xlabel("X")
plt.ylabel("y")
plt.legend()
plt.show()
```

```
Training set performance:
Mean Squared Error (MSE): 9.715696673923548
R² score: 0.9943181060190821

Test set performance:
Mean Squared Error (MSE): 8.727114881750316
R² score: 0.9945040930087029
```

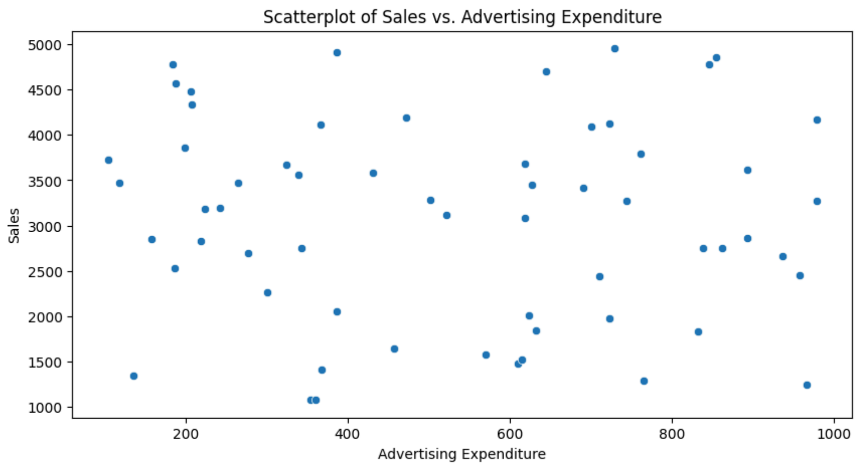


```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score

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

# Convert to DataFrame for better visualization (optional)
df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
df['target'] = iris.target

print("First five rows of the dataset:")
print(df.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)

# Standardize the features (important for KNN)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Initialize the KNN classifier with k=3
knn = KNeighborsClassifier(n_neighbors=3)

# Train the model
knn.fit(X_train, y_train)

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

# Evaluate the model
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

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

print("\nAccuracy Score:")
print(accuracy_score(y_test, y_pred))
```

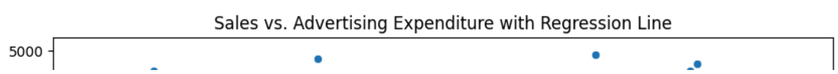
```
First five rows of the dataset:
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  \
0                5.1               3.5                1.4               0.2
1                4.9               3.0                1.4               0.2
2                4.7               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
1       0
2       0
3       0
4       0
Confusion Matrix:
[[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]

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


Accuracy Score:
1.0
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
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 Random Data
np.random.seed(0)
dates = pd.date_range(start='2018-01-01', periods=60, freq='M')  # Monthly data for 5 years
sales = np.random.uniform(1000, 5000, size=(60,))  # Random sales data
ad_expenditure = np.random.uniform(100, 1000, size=(60,))  # Random advertising expenditures

# Create DataFrame
df = pd.DataFrame({'Date': dates, 'Sales': sales, 'Ad_Expenditure': ad_expenditure})

# Step 2: Print the first five rows
print("First five rows of the dataset:")
print(df.head())

# Step 3: Basic statistical computations
print("\nBasic statistical computations:")
print(df.describe())

# Step 4: Columns and their data types
print("\nColumns and their data types:")
print(df.dtypes)

# Step 5: Explore the data using scatterplot
plt.figure(figsize=(10, 5))
sns.scatterplot(x='Ad_Expenditure', y='Sales', data=df)
plt.title('Scatterplot of Sales vs. Advertising Expenditure')
plt.xlabel('Advertising Expenditure')
plt.ylabel('Sales')
plt.show()

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

# No null values in the randomly generated data; if there were, replace with mode
# mode_values = df.mode().iloc[0]
# df = df.fillna(mode_values)

# Step 7: Split the data into train and test sets
X = df[['Ad_Expenditure']]
y = df['Sales']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Step 8: Train the linear regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Predict using the model
y_pred = model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("\nModel Performance:")
print(f"Mean Squared Error (MSE): {mse}")
print(f"R² Score: {r2}")

# Plotting the regression line
plt.figure(figsize=(10, 5))
sns.scatterplot(x='Ad_Expenditure', y='Sales', data=df, label='Actual Data')
plt.plot(X_test, y_pred, color='red', label='Regression Line')
plt.title('Sales vs. Advertising Expenditure with Regression Line')
plt.xlabel('Advertising Expenditure')
plt.ylabel('Sales')
plt.legend()
plt.show()
```

```
First five rows of the dataset:
        Date        Sales  Ad_Expenditure
0 2018-01-31  3195.254016      243.072625
1 2018-02-28  3860.757465      199.337627
2 2018-03-31  3411.053504      690.696631
3 2018-04-30  3179.532732      224.364656
4 2018-05-31  2694.619197      276.924126

Basic statistical computations:
                      Date        Sales  Ad_Expenditure
count                   60    60.000000       60.000000
mean   2020-07-15 12:00:00  3065.653028      529.633412
min    2018-01-31 00:00:00  1075.159202      104.225929
25%    2019-04-22 12:00:00  2210.840665      294.811133
50%    2020-07-15 12:00:00  3187.393374      546.354665
75%    2021-10-07 18:00:00  3808.082954      733.322132
max    2022-12-31 00:00:00  4953.495352      979.084979
std                    NaN  1106.913479      266.827253

Columns and their data types:
Date              datetime64[ns]
Sales                    float64
Ad_Expenditure           float64
dtype: object
```




```
Checking for null values:
Date              0
Sales             0
Ad_Expenditure    0
dtype: int64

Model Performance:
Mean Squared Error (MSE): 698321.8811120327
R² Score: -0.00047767753061500606
```



```python
import pandas as pd
import numpy as np

# Load data into DataFrame
data = {
    'Size': ['Big', 'Small', 'Small', 'Big', 'Small'],
    'Color': ['Red', 'Red', 'Red', 'Blue', 'Blue'],
    'Shape': ['Circle', 'Triangle', 'Circle', 'Circle', 'Circle'],
    'Class': ['No', 'No', 'Yes', 'No', 'Yes']
}

df = pd.DataFrame(data)

# Print the dataset
print("Dataset:")
print(df)

# Initialize the most specific hypothesis (S) and the most general hypothesis (G)
def initialize_S_and_G(df):
    num_attributes = len(df.columns) - 1  # Exclude the class column
    S = ['0'] * num_attributes  # Most specific hypothesis
    G = [['?'] * num_attributes]  # Most general hypothesis
    return S, G

# Update the most specific hypothesis (S)
def update_S(S, instance):
    for i, value in enumerate(instance[:-1]):
        if S[i] == '0':
            S[i] = value
        elif S[i] != value:
            S[i] = '?'
    return S

# Update the most general hypothesis (G)
def update_G(G, instance):
    new_G = []
    for hypothesis in G:
        if all(h == '?' or h == val for h, val in zip(hypothesis, instance[:-1])):
            for i in range(len(hypothesis)):
                if hypothesis[i] == '?':
                    new_hypothesis = hypothesis[:]
                    new_hypothesis[i] = instance[i]
                    new_G.append(new_hypothesis)
    G.extend(new_G)
    G = [hypothesis for hypothesis in G if any(h == '?' for h in hypothesis)]
    return G

# Candidate-Elimination algorithm
def candidate_elimination(df):
    S, G = initialize_S_and_G(df)

    for index, instance in df.iterrows():
        if instance['Class'] == 'Yes':
            S = update_S(S, instance)
            G = [hypothesis for hypothesis in G if any(h == '?' or h == val for h, val in zip(hypothesis, instance[:-1]))]
        else:
            G = update_G(G, instance)
            G = [hypothesis for hypothesis in G if any(h != val for h, val in zip(hypothesis, instance[:-1]))]

    return S, G

# Run the algorithm
S, G = candidate_elimination(df)

# Print the final hypotheses
print("\nMost specific hypothesis (S):")
print(S)
print("\nMost general hypotheses (G):")
for hypothesis in G:
    print(hypothesis)
```

```
Dataset:
    Size Color     Shape Class
0    Big   Red    Circle    No
1  Small   Red  Triangle    No
2  Small   Red    Circle   Yes
```

```
3    Big  Blue    Circle    No
4  Small  Blue    Circle   Yes

Most specific hypothesis (S):
['Small', '?', 'Circle']

Most general hypotheses (G):
['?', '?', '?']
['Big', '?', '?']
['?', 'Red', '?']
['?', '?', 'Circle']
['Small', '?', '?']
['?', 'Red', '?']
['?', '?', 'Triangle']
['Small', 'Red', '?']
['?', 'Red', 'Triangle']
['Big', '?', '?']
['?', 'Blue', '?']
['?', '?', 'Circle']
['Big', 'Blue', '?']
['Big', '?', 'Circle']
['Big', '?', 'Circle']
['?', 'Blue', 'Circle']
```

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

# Load the Breast Cancer dataset
from sklearn.datasets import load_breast_cancer
data = load_breast_cancer()

# Create a DataFrame
df = pd.DataFrame(data.data, columns=data.feature_names)
df['target'] = data.target

# Step 1: Print the first five rows
print("First five rows of the dataset:")
print(df.head())

# Step 2: Basic statistical computations
print("\nBasic statistical computations:")
print(df.describe())

# Step 3: Columns and their data types
print("\nColumns and their data types:")
print(df.dtypes)

# Step 4: Explore the data using scatterplot
plt.figure(figsize=(10, 5))
sns.scatterplot(x='mean radius', y='mean texture', hue='target', data=df)
plt.title('Scatterplot of Mean Radius vs. Mean Texture')
plt.xlabel('Mean Radius')
plt.ylabel('Mean Texture')
plt.show()

# Step 5: Detect and handle null values
print("\nChecking for null values:")
print(df.isnull().sum())

# Step 6: Split the data into train and test sets
X = df.drop(columns=['target'])
y = df['target']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Step 7: Train the Logistic Regression model
model = LogisticRegression(max_iter=10000)
model.fit(X_train, y_train)

# Predict using the model
y_pred = model.predict(X_test)

# Step 8: Evaluate the model
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)
accuracy = accuracy_score(y_test, y_pred)

print("\nConfusion Matrix:")
print(conf_matrix)
print("\nClassification Report:")
print(class_report)
print(f"\nAccuracy Score: {accuracy}")

# Optional: Visualize the confusion matrix
plt.figure(figsize=(6, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```