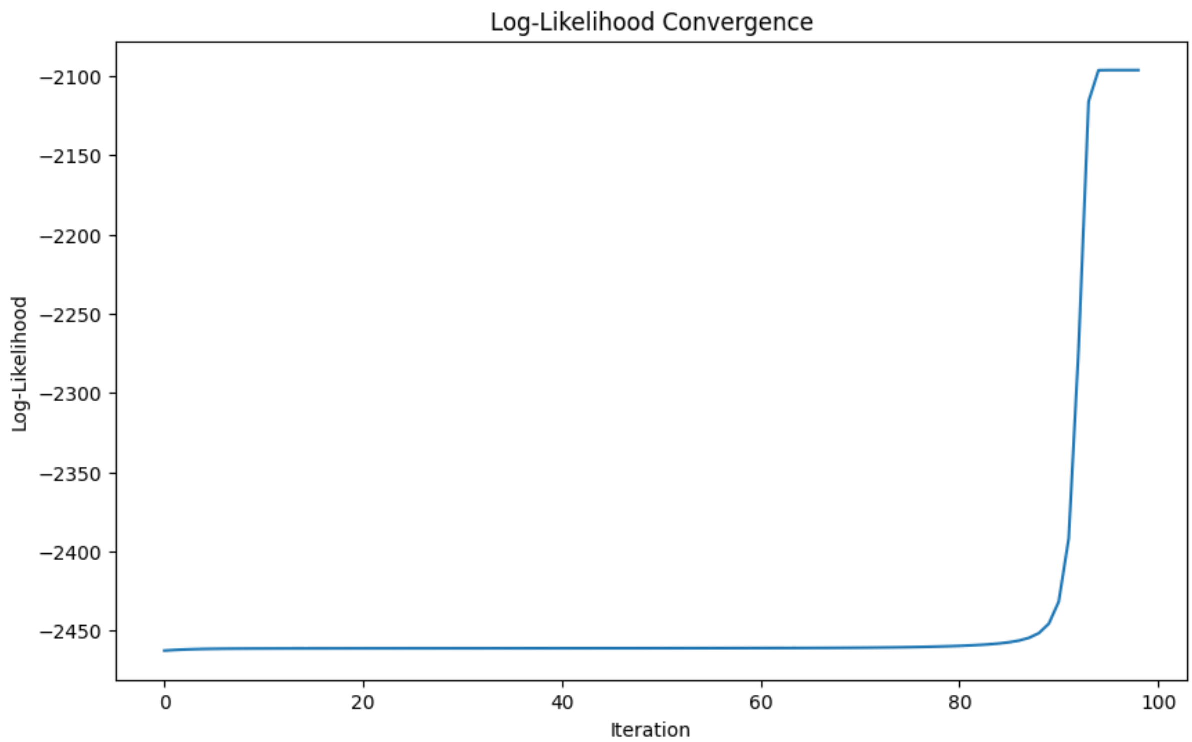
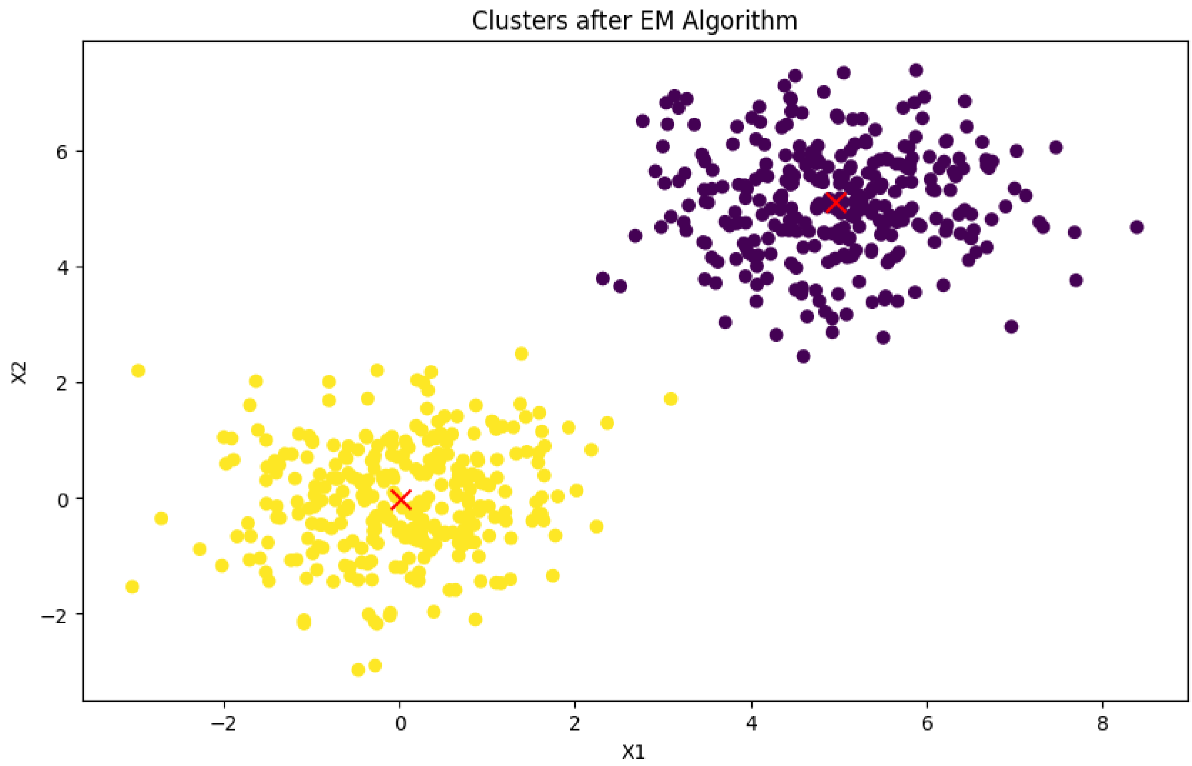
#16

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import multivariate_normal

# Generate synthetic data
np.random.seed(42)
n_samples = 300
mean1 = [0, 0]
cov1 = [[1, 0.1], [0.1, 1]]
mean2 = [5, 5]
cov2 = [[1, -0.1], [-0.1, 1]]

X = np.vstack([
    np.random.multivariate_normal(mean1, cov1, n_samples),
    np.random.multivariate_normal(mean2, cov2, n_samples)
])

# Number of components
k = 2

# Initialize the parameters
np.random.seed(42)
pi = np.ones(k) / k  # Mixing coefficients
means = np.random.rand(k, 2)  # Means of the Gaussians
covariances = np.array([np.eye(2)] * k)  # Covariances of the Gaussians

def e_step(X, pi, means, covariances):
    N = X.shape[0]
    r = np.zeros((N, k))
```

```
    for i in range(k):
        r[:, i] = pi[i] * multivariate_normal.pdf(X, mean=means[i], cov=covariances[i])
    r = r / r.sum(axis=1, keepdims=True)
    return r

def m_step(X, r):
    N, D = X.shape
    pi = r.sum(axis=0) / N
    means = np.dot(r.T, X) / r.sum(axis=0)[:, np.newaxis]
    covariances = np.zeros((k, D, D))

    for i in range(k):
        diff = X - means[i]
        covariances[i] = np.dot(r[:, i] * diff.T, diff) / r[:, i].sum()

    return pi, means, covariances

def log_likelihood(X, pi, means, covariances):
    N = X.shape[0]
    log_likelihood = 0

    for i in range(k):
        log_likelihood += pi[i] * multivariate_normal.pdf(X, mean=means[i], cov=covariances[i])
    return np.log(log_likelihood).sum()

# Run the EM algorithm
max_iter = 100
tol = 1e-6
log_likelihoods = []

for iteration in range(max_iter):
```

```
    r = e_step(X, pi, means, covariances)
    pi, means, covariances = m_step(X, r)
    log_likelihoods.append(log_likelihood(X, pi, means, covariances))

    # Check for convergence
    if iteration > 0 and abs(log_likelihoods[-1] - log_likelihoods[-2]) < tol:
        break

# Plot the log-likelihoods
plt.figure(figsize=(10, 6))
plt.plot(log_likelihoods)
plt.xlabel('Iteration')
plt.ylabel('Log-Likelihood')
plt.title('Log-Likelihood Convergence')
plt.show()

# Plot the final clusters
plt.figure(figsize=(10, 6))
plt.scatter(X[:, 0], X[:, 1], c=r.argmax(axis=1), cmap='viridis', marker='o')
plt.scatter(means[:, 0], means[:, 1], c='red', marker='x', s=100)
plt.title('Clusters after EM Algorithm')
plt.xlabel('X1')
plt.ylabel('X2')
plt.show()
#output
```





#17

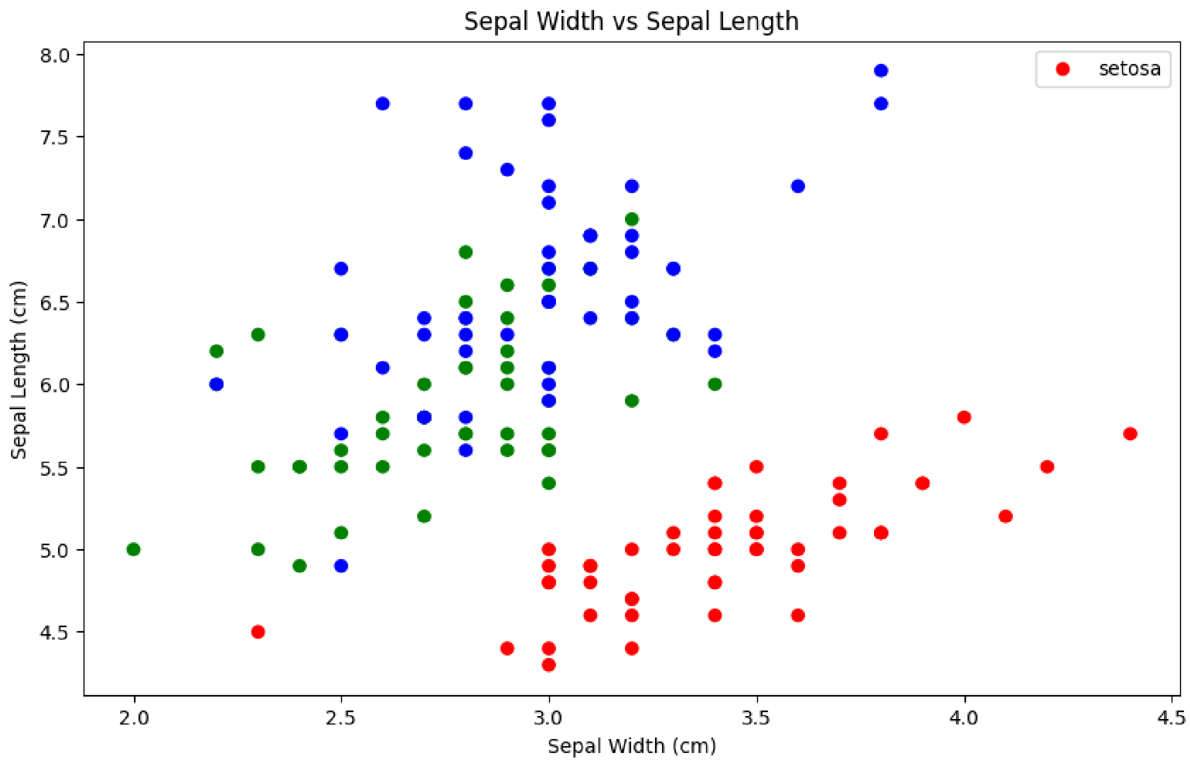
import pandas as pd

import numpy as np

```
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

from sklearn.datasets import load_iris
iris = load_iris()
df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
df['species'] = iris.target
df['species'] = df['species'].map({0: 'setosa', 1: 'versicolor', 2: 'virginica'})

# Step 2: Plot the data using a scatter plot "sepal_width" versus "sepal_length" and color species
plt.figure(figsize=(10, 6))
colors = {'setosa': 'red', 'versicolor': 'green', 'virginica': 'blue'}
plt.scatter(df['sepal width (cm)'], df['sepal length (cm)'], c=df['species'].apply(lambda x: colors[x]),
label=colors)
plt.xlabel('Sepal Width (cm)')
plt.ylabel('Sepal Length (cm)')
plt.title('Sepal Width vs Sepal Length')
plt.legend(colors)
plt.show()

# Step 3: Split the data
X = df.drop(columns='species')
y = df['species']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Step 4: Fit the data to the model
model = LogisticRegression(max_iter=200)
model.fit(X_train, y_train)
```

```
# Evaluate the model
y_pred_train = model.predict(X_train)
y_pred_test = model.predict(X_test)
print(f'Training Accuracy: {accuracy_score(y_train, y_pred_train):.2f}')
print(f'Testing Accuracy: {accuracy_score(y_test, y_pred_test):.2f}')

# Step 5: Predict the model with new test data [5, 3, 1, 0.3]
new_sample = np.array([[5, 3, 1, 0.3]])
prediction = model.predict(new_sample)
predicted_species = prediction[0]
print(f'The predicted species for the new sample [5, 3, 1, 0.3] is: {predicted_species}')
```

#output



Training Accuracy: 0.97

Testing Accuracy: 1.00

The predicted species for the new sample [5, 3, 1, 0.3] is: setosa

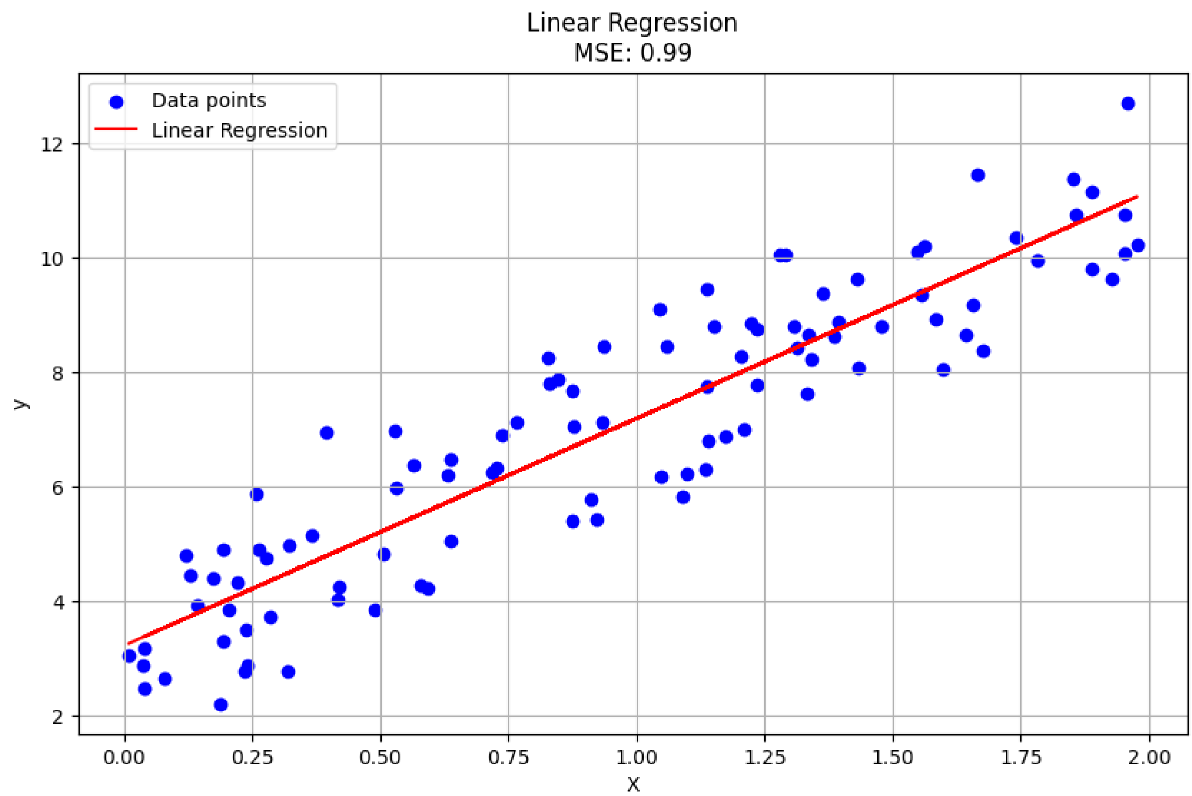
```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but LogisticRegression was fitted with feature names
  warnings.warn(
```

```
#18
import pandas as pd

# Load the CSV file
data = {
    'Shape':['Circular','Circular','Oval','Oval'],
    'Size' :['Large','Large','Large','Large'],
    'Color':['Light','Light','Dark','Light'],
    'Surface' :['Smooth','Irregular','Smooth','Irregular'],
    'Thickness' :['Thick','Thick','Thin','Thick'],
    'Target Concept' :['Maligant(+)','Malignant(+)','Benign(-)','Malignant(+)']
}
df = pd.DataFrame(data)
# Initialize S and G
# S is initialized to the most specific hypothesis
# G is initialized to the most general hypothesis
S = ['0'] * (len(df.columns) - 2)
G = [['?' for _ in range(len(df.columns) - 2)]]

# Function to update S
def update_S(s, example):
    for i in range(len(s)):
        if s[i] == '0':
            s[i] = example[i]
        elif s[i] != example[i]:
            s[i] = '?'
    return s
```

```
# Function to update G
def update_G(g, s, example):
    new_g = []
    for h in g:
        consistent = True
        for i in range(len(h)):
            if h[i] != '?' and h[i] != example[i]:
                consistent = False
                break
        if not consistent:
            for i in range(len(h)):
                if h[i] == '?':
                    new_h = h[:]
                    new_h[i] = s[i]
                    if new_h not in new_g:
                        new_g.append(new_h)
    return new_g

# Process each training example
for index, row in df.iterrows():
    example = list(row[:-1])
    target = row[-1]
    if target == 'Malignant (+)':
        # Positive example
        S = update_S(S, example)
        G = [h for h in G if all(h[i] == '?' or h[i] == example[i] for i in range(len(h)))]
    else:
        # Negative example
        G = update_G(G, S, example)
        S = [s for s in S if not all(s[i] == '?' or s[i] == example[i] for i in range(len(s)))]
```

```
# Output the final S and G
print("Final specific hypothesis S:", S)
print("Final general hypotheses G:", G)

#output
Final specific hypothesis S: ['0', '0', '0', '0']
Final general hypotheses G: []

#19
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error

# Step 1: Generate synthetic data
np.random.seed(0)
X = 2 * np.random.rand(100, 1)
y = 3 + 4 * X + np.random.randn(100, 1)

# Step 2: Implement Linear Regression
def linear_regression(X, y):
    # Add bias term
    X_b = np.c_[np.ones((len(X), 1)), X]
    # Normal equation: theta = (X^T * X)^-1 * X^T * y
    theta_best = np.linalg.inv(X_b.T.dot(X_b)).dot(X_b.T).dot(y)
    return theta_best

# Fit linear regression model
theta_best = linear_regression(X, y)

# Extract coefficients
intercept, slope = theta_best[0], theta_best[1]
```

```
# Step 3: Show Performance
# Predictions
y_predict = np.dot(np.c_[np.ones((len(X), 1)), X], theta_best)

# Calculate Mean Squared Error (MSE)
mse = mean_squared_error(y, y_predict)

# Plotting
plt.figure(figsize=(10, 6))
plt.scatter(X, y, color='blue', label='Data points')
plt.plot(X, y_predict, color='red', label='Linear Regression')
plt.title(f'Linear Regression\nMSE: {mse:.2f}')
plt.xlabel('X')
plt.ylabel('y')
plt.legend()
plt.grid(True)
plt.show()

print(f'Intercept (theta_0): {intercept[0]:.2f}')
print(f'Slope (theta_1): {slope[0]:.2f}')
print(f'Mean Squared Error (MSE): {mse:.2f}')

#output
```



Intercept (theta_0): 3.22

Slope (theta_1): 3.97

Mean Squared Error (MSE): 0.99

#20

```
import numpy as np

from collections import defaultdict


class NaiveBayes:

    def __init__(self):

        self.classes = None

        self.class_priors = None

        self.feature_probs = None


    def fit(self, X, y):

        self.classes = np.unique(y)

        self.class_priors = np.bincount(y) / len(y)  # Class probabilities (prior)
```

```
        self.feature_probs = {c: defaultdict(lambda: np.zeros(X.shape[1])) for c in self.classes} # Initialize
as np.zeros

        for c in self.classes:
            X_class = X[y == c]
            for i in range(X.shape[1]):  # Iterate over features
                self.feature_probs[c][i] += np.sum(X_class[:, i]) # Update feature counts
            for i in range(X.shape[1]):
                self.feature_probs[c][i] /= (X_class.shape[0] + X.shape[1])  # Apply Laplace smoothing

    def predict(self, X):
        if self.classes is None:
            raise Exception("Model not fitted yet. Call fit(X, y) first.")

        y_pred = []
        for x in X:
            posteriors = {}
            for c in self.classes:
                prior = self.class_priors[c]
                likelihood = np.prod([self.feature_probs[c][i]**x[i] * (1 - self.feature_probs[c][i])**(1 - x[i])
for i in range(len(x))])
                posteriors[c] = prior * likelihood
            y_pred.append(max(posteriors, key=posteriors.get))

        return np.array(y_pred)

# Example usage
# Sample email data (presence/absence of words) and labels
X = np.array([
    [1, 1, 0, 0, 0, 0],  # "free money" email (spam)
    [1, 0, 0, 1, 1, 0],  # Meeting invitation email (not spam)
    [0, 0, 1, 0, 0, 1],  # Project update email (not spam)
```

```
    [1, 1, 0, 1, 0, 0],  # "Get rich quick" email (spam)
])
y = np.array([1, 0, 0, 1])  # 1 for spam, 0 for not spam

# Create and train the Naive Bayes model
model = NaiveBayes()
model.fit(X, y)

# New email to classify (presence/absence of words)
new_email = np.array([0, 1, 0, 1, 1, 1])  # Email about a meeting and deadline

# Predict class label for the new email
predicted_class = model.predict(new_email.reshape(1, -1))  # Reshape for single data point
print("Predicted class:", "spam" if predicted_class[0] == 1 else "not spam")



#output
Predicted class: not spam

#21
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.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

# Sample data
data = {
    'size': [1400, 1600, 1700, 1800, 1900, 2000],
```

```
    'bedrooms': [3, 3, 4, 4, 4, 5],
    'bathrooms': [2, 3, 2, 3, 3, 4],
    'location': [1, 1, 2, 2, 3, 3],
    'price': [300000, 350000, 400000, 420000, 450000, 500000]
}

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

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

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

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

# Step 4: Detect and handle null values
print("\nDetecting null values:")
print(df.isnull().sum())

print("\nNull values after replacement:")
print(df.isnull().sum())

plt.figure(figsize=(10, 8))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm')
plt.title('Heatmap of Feature Correlations')
```

```
plt.show()

X = df.drop('price', axis=1)
y = df['price']

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

model = LinearRegression()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)

print("\nModel Performance:")
print(f"Root Mean Squared Error: {rmse}")

new_house = pd.DataFrame({
    'size': [1500],
    'bedrooms': [3],
    'bathrooms': [2],
    'location': [1]
})

predicted_price = model.predict(new_house)
print("\nPredicted price for the new house:", predicted_price[0])

#output
First five rows of the dataset:
  size  bedrooms  bathrooms  location   price
```

```
0  1400      3      2      1  300000
1  1600      3      3      1  350000
2  1700      4      2      2  400000
3  1800      4      3      2  420000
4  1900      4      3      3  450000
```

Basic statistical computations:

```
              size  bedrooms  bathrooms  location          price
count     6.000000  6.000000   6.000000  6.000000       6.000000
mean   1733.333333  3.833333   2.833333  2.000000  403333.333333
std     216.024690  0.752773   0.752773  0.894427   71180.521680
min    1400.000000  3.000000   2.000000  1.000000  300000.000000
25%    1625.000000  3.250000   2.250000  1.250000  362500.000000
50%    1750.000000  4.000000   3.000000  2.000000  410000.000000
75%    1875.000000  4.000000   3.000000  2.750000  442500.000000
max    2000.000000  5.000000   4.000000  3.000000  500000.000000
```

Columns and their data types:

```
size         int64
bedrooms     int64
bathrooms    int64
location     int64
price        int64
dtype: object
```

Detecting null values:

```
size         0
bedrooms     0
bathrooms    0
location     0
price        0
```

dtype: int64

Null values after replacement:

size   0

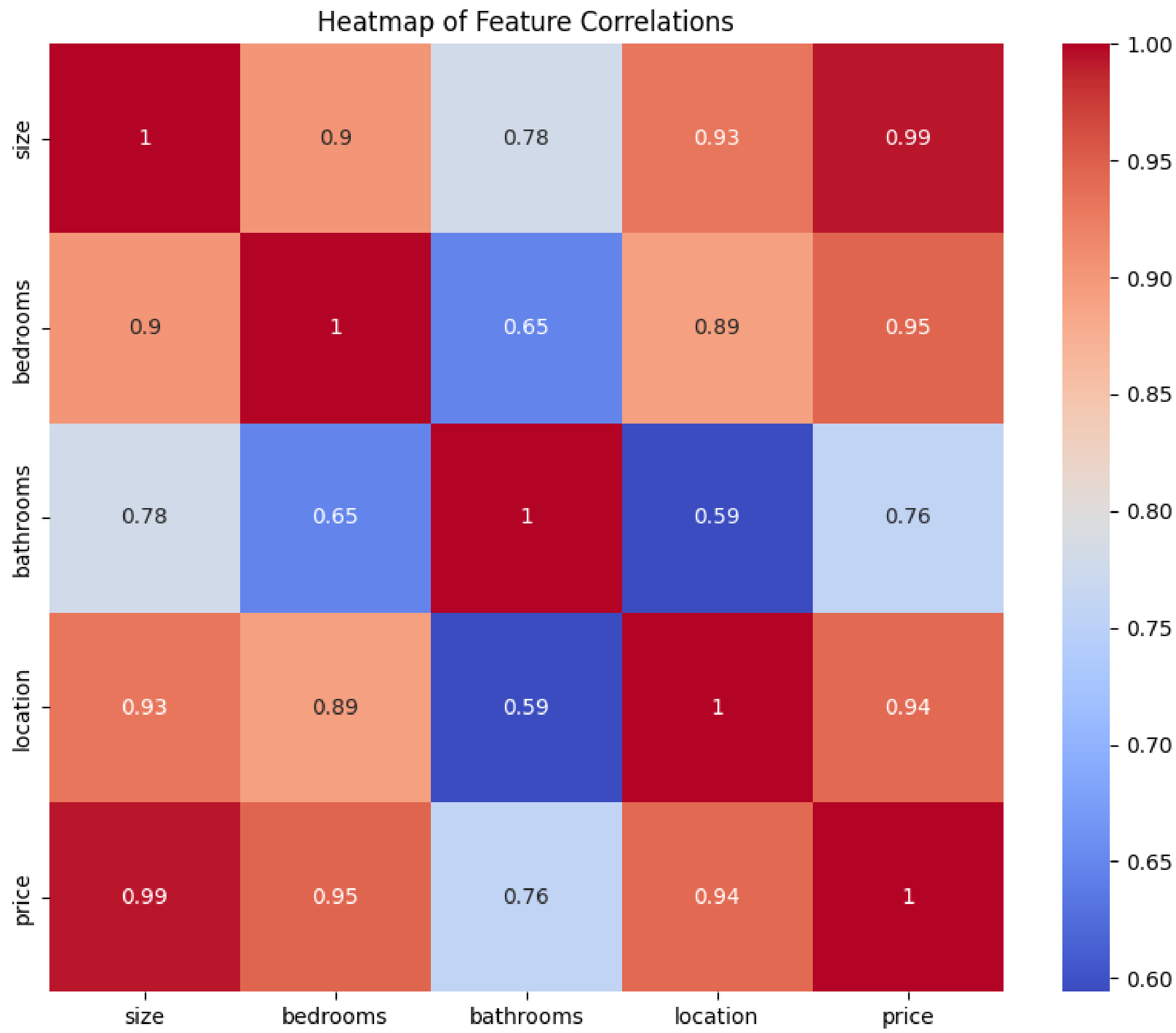
bedrooms   0

bathrooms   0

location   0

price   0

dtype: int64



Model Performance:

Root Mean Squared Error: 12745.832891312073

Predicted price for the new house: 315001.249937503

#22

```
import pandas as pd

# Function to check if a hypothesis is consistent with an example
def is_consistent(hypothesis, example):
    for h, e in zip(hypothesis, example):
        if h != '?' and h != e:
            return False
    return True

# Function to find the minimal generalization of S
def generalize_minimally(h, x):
    new_h = list(h)
    for i in range(len(h)):
        if not is_consistent([h[i]], [x[i]]):
            new_h[i] = '?' if h[i] != x[i] else x[i]
    return tuple(new_h)

# Function to find the minimal specialization of G
def specialize_minimally(h, domains, x):
    specializations = []
    for i in range(len(h)):
        if h[i] == '?':
            for val in domains[i]:
                if val != x[i]:
                    new_h = list(h)
                    new_h[i] = val
                    specializations.append(tuple(new_h))
        elif h[i] != x[i]:
```

```
        new_h = list(h)
        new_h[i] = '?'
        specializations.append(tuple(new_h))
    return specializations

# Candidate-Elimination Algorithm
def candidate_elimination(examples):
    domains = [set(examples[col]) for col in examples.columns[:-1]]
    n_features = len(domains)

    # Initialize S to the most specific hypothesis
    S = tuple(['Ø'] * n_features)

    # Initialize G to the most general hypothesis
    G = [tuple(['?'] * n_features)]

    for index, row in examples.iterrows():
        x, y = row[:-1], row[-1]
        x = tuple(x)

        if y == 'yes':  # Positive example
            # Remove from G any hypothesis inconsistent with x
            G = [g for g in G if is_consistent(g, x)]

            S = [s for s in S if is_consistent(s, x)]
            for s in S:
                if not is_consistent(s, x):
                    S.remove(s)
                    S.append(generalize_minimally(s, x))

            # Remove from S any hypothesis that is more general than another hypothesis in S
```

```
        S = [s for s in S if not any(s != s2 and is_consistent(s2, s) for s2 in S)]

      else:  # Negative example
        # Remove from S any hypothesis inconsistent with x
        S = [s for s in S if not is_consistent(s, x)]

        # For each hypothesis g in G that is consistent with x, remove g from G
        # Add to G all minimal specializations h of g such that h is not consistent with x and some
member of S is more specific than h
        new_G = []
        for g in G:
          if is_consistent(g, x):
            new_G.extend(specialize_minimally(g, domains, x))
          else:
            new_G.append(g)
        G = new_G

        # Remove from G any hypothesis that is more specific than another hypothesis in G
        G = [g for g in G if not any(g != g2 and is_consistent(g, g2) for g2 in G)]

  return S, G

data = {
  'Sky': ['Sunny', 'Sunny', 'Rainy', 'Sunny', 'Sunny'],
  'AirTemp': ['Warm', 'Warm', 'Cold', 'Warm', 'Warm'],
  'Humidity': ['Normal', 'High', 'High', 'High', 'Normal'],
  'Wind': ['Strong', 'Strong', 'Strong', 'Strong', 'Weak'],
  'Water': ['Warm', 'Warm', 'Warm', 'Warm', 'Cool'],
  'Forecast': ['Same', 'Same', 'Change', 'Same', 'Same'],
  'EnjoySport': ['yes', 'yes', 'no', 'yes', 'yes']
}
```

```
examples = pd.DataFrame(data)

# Apply Candidate-Elimination algorithm
S, G = candidate_elimination(examples)

print("Most Specific Hypothesis (S):", S)
print("Most General Hypothesis (G):", G)

#output
Most Specific Hypothesis (S): []
Most General Hypothesis (G): [('Sunny', '?', '?', '?', '?', '?'), ('?', 'Warm', '?', '?', '?', '?'), ('?', '?', '?', '?', '?', 'Same')]
```