Experiment 1:

Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a .CSV file.

Source Code :

```
import pandas as pd
def find_s_algorithm(data):
    attributes = data.iloc[:, :-1].values
    target = data.iloc[:, -1].values
    h = None
    for i in range(len(target)):
        if target[i].strip().lower() == 'yes':
            if h is None:
                h = attributes[i].copy()  # Initialize with first positive example
            else:
                for j in range(len(h)):
                    if h[j] != attributes[i][j]:
                        h[j] = '?'
return h
data = pd.read_csv('weather.csv')
print("Dataset:\n", data)
print("Most specific hypothesis:", find_s_algorithm(data))
```

Dataset

Outlook	**Temperature**	**Humidity**	**Windy**	**Play**
Overcast	Hot	High	False	Yes
Overcast	Cool	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes

Output

Most specific hypothesis: ['Overcast', '?', '?', '?']

Experiment 2:

For a given set of training data examples stored in a .CSV file, implement and demonstrate the CandidateElimination algorithm to output a description of the set of all hypotheses consistent with the training examples.

Source Code :

```
import numpy as np
import pandas as pd
data = pd.DataFrame(data=pd.read_csv('label.csv'))
concepts = np.array(data.iloc[:, 0:-1])
target = np.array(data.iloc[:, -1])
def learn(concepts, target):
  specific_h = concepts[0].copy()
  general_h = [["?" for _ in range(len(specific_h))] for _ in range(len(specific_h))]
  for i, h in enumerate(concepts):
    if target[i] == "Yes":
      for x in range(len(specific_h)):
        if h[x] != specific_h[x]:
          specific_h[x] = '?'
          general_h[x][x] = '?'
    elif target[i] == "No":
      for x in range(len(specific_h)):
        if h[x] != specific_h[x]:
          general_h[x][x] = specific_h[x]
        else:
          general_h[x][x] = '?'
  general_h = [h for h in general_h if h != ['?'] * len(specific_h)
  return specific_h, general_h
s_final, g_final = learn(concepts, target)
print("\nFinal Specific Hypothesis:", s_final)
print("\nFinal General Hypothesis:", g_final)
```

Dataset

Sky	Temperature	Humidity	Wind	PlayTennis
Sunny	Warm	Normal	Strong	Yes
Sunny	Warm	High	Strong	Yes
Rainy	Cold	High	Strong	No
Sunny	Warm	High	Weak	Yes
Sunny	Cold	Normal	Weak	No
Rainy	Warm	High	Strong	No

Sky	Temperature	Humidity	Wind	PlayTennis
Sunny	Warm	Normal	Strong	Yes
Sunny	Warm	High	Strong	Yes
Rainy	Cold	High	Strong	No
Sunny	Warm	High	Weak	Yes
Sunny	Cold	Normal	Weak	No
Rainy	Warm	High	Strong	No

Output

Final Specific Hypothesis:
['Sunny' 'Warm' '?' '?']
Final General Hypothesis:
[['Sunny' '?' '?' '?']]

Experiment 3:

Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample

Source Code :

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier, export_text
from sklearn.preprocessing import LabelEncoder
df = pd.read_csv('weather.csv')
le = LabelEncoder()
for column in df.columns:
    df[column] = le.fit_transform(df[column])
X = df.drop(columns=['PlayTennis'])
y = df['PlayTennis']
clf = DecisionTreeClassifier(criterion='entropy')
clf.fit(X, y)
tree_rules = export_text(clf, feature_names=list(X.columns))
print("Decision Tree Rules:\n", tree_rules)
new_sample = pd.DataFrame({'Outlook': ['Sunny'], 'Temperature': ['Cool'], 'Humidity': ['High'],
'Wind': ['Strong']})
for column in new_sample.columns:
    new_sample[column] = le.fit_transform(new_sample[column])
prediction = clf.predict(new_sample)
print("\nPredicted Class for the new sample:", "Yes" if prediction[0] == 1 else "No")
```

Dataset:

```
Outlook,Temperature,Humidity,Wind,PlayTennis
Sunny,Hot,High,Weak,No
Sunny,Hot,High,Strong,No
Overcast,Hot,High,Weak,Yes
Rain,Mild,High,Weak,Yes
Rain,Cool,Normal,Weak,Yes
Rain,Cool,Normal,Strong,No
Overcast,Cool,Normal,Strong,Yes
Sunny,Mild,High,Weak,No
Sunny,Cool,Normal,Weak,Yes
Rain,Mild,Normal,Weak,Yes
Sunny,Mild,Normal,Strong,Yes
Overcast,Mild,High,Strong,Yes
Overcast,Hot,Normal,Weak,Yes
Rain,Mild,High,Strong,No
```

Output

Decision Tree Rules:

```
 |--- Outlook <= 0.50
|   |--- class: 1
|--- Outlook >  0.50
|   |--- Humidity <= 0.50
|   |   |--- Outlook <= 1.50
|   |   |   |--- Wind <= 0.50
|   |   |   |   |--- class: 0
|   |   |   |--- Wind >  0.50
|   |   |   |   |--- class: 1
|   |   |--- Outlook >  1.50
|   |   |   |--- class: 0
|   |--- Humidity >  0.50
|   |   |--- Wind <= 0.50
|   |   |   |--- Outlook <= 1.50
|   |   |   |   |--- class: 0
|   |   |   |--- Outlook >  1.50
|   |   |   |   |--- class: 1
|   |   |--- Wind >  0.50
|   |   |   |--- class: 1
```

Predicted Class for the new sample: Yes

Experiment 4:

Exercises to solve the real-world problems using the following machine learning methods: a) Linear Regression b) Logistic Regression c) Binary Classifier

Source Code

a. Linear Regression

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import mean_absolute_error, r2_score
df = pd.read_csv('house_prices.csv')
le = LabelEncoder()
df['Location'] = le.fit_transform(df['Location'])
X = df[['SquareFootage', 'Bedrooms', 'Location', 'Age']]
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)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f"Mean Absolute Error: {mae}")
print(f"R² Score: {r2}")
new_house = pd.DataFrame({'SquareFootage': [2000], 'Bedrooms': [3], 'Location':
[le.transform(['New York'])[0]], 'Age': [5]})
predicted_price = model.predict(new_house)
print(f"\nPredicted Price for the new house: ${predicted_price[0]:,.2f}")
```

DataSet:

```
SquareFootage,Bedrooms,Location,Age,Price
1500,3,New York,10,400000
1800,4,Los Angeles,5,500000
2200,4,Chicago,8,450000
1300,2,Miami,12,350000
2500,5,New York,3,600000
2100,4,Los Angeles,6,520000
1700,3,Chicago,7,410000
1600,3,Miami,9,380000
2000,4,New York,4,550000
1900,3,Los Angeles,6,490000
```

Output:

Mean Absolute Error: 34857.26230291449
R^2 Score: -1.6246118520337984
Predicted Price for the new house: $560,640.23

b. Logistic Regression

```
import pandas as pd
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
df = pd.read_csv('customer_data.csv')
X = df[['Age', 'Salary', 'PreviouslyPurchased']]
y = df['WillBuy']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
model = LogisticRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
print("\nClassification Report:\n", report)
new_customer = pd.DataFrame({'Age': [35], 'Salary': [60000], 'PreviouslyPurchased': [1]})
new_customer = scaler.transform(new_customer)
prediction = model.predict(new_customer)
print("\nPrediction for new customer:", "Will Buy" if prediction[0] == 1 else "Will Not Buy")
```

DataSet:

```
Age,Salary,PreviouslyPurchased,WillBuy
22,25000,0,0
25,32000,0,0
47,80000,1,1
52,90000,1,1
46,77000,1,1
56,110000,1,1
28,35000,0,0
30,40000,0,0
29,38000,0,0
53,102000,1,1
49,96000,1,1
31,43000,0,0
```

Output:

Accuracy: 1.00

Classification Report:

```
          precision    recall  f1-score   support

       0       1.00      1.00      1.00         1
       1       1.00      1.00      1.00         2

accuracy                           1.00         3
   macro avg       1.00      1.00      1.00         3
weighted avg       1.00      1.00      1.00         3
```

Prediction for new customer: Will Buy

c. Binary Classifier

```
import pandas as pd
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
df = pd.read_csv('emails.csv')
X = df[['WordCount', 'HasOffer', 'HasUrgent', 'EmailLength']]
y = df['Spam']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
model = LogisticRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
print("\nClassification Report:\n", report)
new_email = pd.DataFrame({'WordCount': [110], 'HasOffer': [1], 'HasUrgent': [0], 'EmailLength':
[320]})
new_email = scaler.transform(new_email)
prediction = model.predict(new_email)
print("\nPrediction for new email:", "Spam" if prediction[0] == 1 else "Not Spam")
```

Dataset:

WordCount,HasOffer,HasUrgent,EmailLength,Spam
120,1,0,300,1
150,0,1,450,0
80,1,1,200,1
200,0,0,600,0
130,1,0,350,1
90,0,1,250,0
170,1,1,500,1
160,0,0,550,0
140,1,0,400,1
180,0,1,520,0

Output:

Accuracy: 1.00

Classification Report:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	1
1	1.00	1.00	1.00	1
accuracy			1.00	2
macro avg	1.00	1.00	1.00	2
weighted avg	1.00	1.00	1.00	2

Prediction for new email: Spam

Experiment – 5

Develop a program for Bias, Variance, Remove duplicates , Cross Validation

Source Code:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error
data = {
  'Feature1': [10, 20, 30, 40, 50, 50, 60, 70, 80, 90, 100, 100],
  'Feature2': [5, 10, 15, 20, 25, 25, 30, 35, 40, 45, 50, 50],
  'Target': [12, 24, 36, 48, 60, 60, 72, 84, 96, 108, 120, 120]
}
df = pd.DataFrame(data)
df = df.drop_duplicates()
print("Dataset after removing duplicates:\n", df)
X = df[['Feature1', 'Feature2']]
y = df['Target']
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)
model = LinearRegression()
model.fit(X_train, y_train)
train_pred = model.predict(X_train)
test_pred = model.predict(X_test)
train_error = mean_squared_error(y_train, train_pred)
test_error = mean_squared_error(y_test, test_pred)
print("\nTraining Error (Bias):", train_error)
print("Testing Error (Variance):", test_error)
cv_scores = cross_val_score(model, X_scaled, y, cv=5, scoring='neg_mean_squared_error')
print("\nCross-Validation Scores (MSE):", -cv_scores)
print("Average Cross-Validation Score:", -np.mean(cv_scores))
```

Output:

```
Dataset after removing duplicates:
   Feature1  Feature2  Target
0       10        5      12
1       20       10      24
2       30       15      36
3       40       20      48
4       50       25      60
6       60       30      72
7       70       35      84
8       80       40      96
9       90       45     108
10     100       50     120
Training Error (Bias): 5.679798517591285e-29
Testing Error (Variance): 2.524354896707238e-29
```

Cross-Validation Scores (MSE): [2.5243549e-29 2.5243549e-29 0.0000000e+00 0.0000000e+00 0.0000000e+00]
Average Cross-Validation Score: 1.0097419586828952e-29

Experiment 6: Write a program to implement Categorical Encoding, One-hot Encoding

Source Code :

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
data = {
    'Color': ['Red', 'Blue', 'Green', 'Blue', 'Red', 'Green'],
    'Size': ['S', 'M', 'L', 'M', 'S', 'L'],
    'Label': ['Yes', 'No', 'Yes', 'No', 'Yes', 'No']
}
df = pd.DataFrame(data)
print("Original Dataset:\n", df)
label_encoder = LabelEncoder()
df['Color_Label'] = label_encoder.fit_transform(df['Color'])
df['Size_Label'] = label_encoder.fit_transform(df['Size'])
df['Label_Label'] = label_encoder.fit_transform(df['Label'])
print("\nDataset after Label Encoding:\n", df)
one_hot_encoder = OneHotEncoder(drop='first', sparse_output=False)
encoded_features = one_hot_encoder.fit_transform(df[['Color', 'Size']])
encoded_df = pd.DataFrame(encoded_features,
columns=one_hot_encoder.get_feature_names_out(['Color', 'Size']))
df = df.drop(columns=['Color', 'Size']).join(encoded_df)
print("\nDataset after One-Hot Encoding:\n", df)
```

Output:

```
Original Dataset:
   Color Size Label
0   Red    S   Yes
1  Blue    M    No
2  Green   L   Yes
3  Blue    M    No
4   Red    S   Yes
5  Green   L    No
Dataset after Label Encoding:
   Color Size Label  Color_Label  Size_Label  Label_Label
0   Red    S   Yes        2          2          1
1  Blue    M    No        0          1          0
2  Green   L   Yes        1          0          1
3  Blue    M    No        0          1          0
4   Red    S   Yes        2          2          1

5  Green   L    No        1          0          0

Dataset after One-Hot Encoding:

  Label  Color_Label  Size_Label  ...  Color_Red  Size_M  Size_S

0   Yes           2           2 ...       1.0     0.0     1.0

1    No           0           1 ...       0.0     1.0     0.0

2   Yes           1           0 ...       0.0     0.0     0.0
```

```
3    No           0          1  ...       0.0     1.0     0.0
4   Yes           2          2  ...       1.0     0.0     1.0
5    No           1          0  ...       0.0     0.0     0.0

[6 rows x 8 columns]
```

Experiment 7: Build an Artificial Neural Network by implementing the Back propagation algorithm and test the same using appropriate data sets.

Source Code

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.datasets import make_moons
from sklearn.preprocessing import StandardScaler
X, y = make_moons(n_samples=500, noise=0.2, random_state=42)
y = y.reshape(-1, 1)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
input_size = X_train.shape[1]
hidden_size = 5
output_size = 1
learning_rate = 0.1
epochs = 10000
np.random.seed(42)
W1 = np.random.randn(input_size, hidden_size)
b1 = np.zeros((1, hidden_size))
W2 = np.random.randn(hidden_size, output_size)
b2 = np.zeros((1, output_size))
def sigmoid(x):
    return 1 / (1 + np.exp(-x))
def sigmoid_derivative(x):
    return x * (1 - x)
for epoch in range(epochs):
    Z1 = np.dot(X_train, W1) + b1
    A1 = sigmoid(Z1)
    Z2 = np.dot(A1, W2) + b2
    A2 = sigmoid(Z2)
    loss = np.mean((y_train - A2) ** 2)
    dA2 = -(y_train - A2)
    dZ2 = dA2 * sigmoid_derivative(A2)
    dW2 = np.dot(A1.T, dZ2)
    db2 = np.sum(dZ2, axis=0, keepdims=True)
    dA1 = np.dot(dZ2, W2.T)
    dZ1 = dA1 * sigmoid_derivative(A1)
    dW1 = np.dot(X_train.T, dZ1)
    db1 = np.sum(dZ1, axis=0, keepdims=True)
    W2 -= learning_rate * dW2
    b2 -= learning_rate * db2
    W1 -= learning_rate * dW1
    b1 -= learning_rate * db1
    if epoch % 1000 == 0:
        print(f"Epoch {epoch}, Loss: {loss:.4f}")
Z1_test = np.dot(X_test, W1) + b1
A1_test = sigmoid(Z1_test)
```

```
Z2_test = np.dot(A1_test, W2) + b2
A2_test = sigmoid(Z2_test)
predictions = (A2_test > 0.5).astype(int)
accuracy = np.mean(predictions == y_test)
print("\nTest Accuracy:", accuracy)
```

Output:

```
Epoch 0, Loss: 0.4047
Epoch 1000, Loss: 0.0149
Epoch 2000, Loss: 0.0135
Epoch 3000, Loss: 0.0117
Epoch 4000, Loss: 0.0100
Epoch 5000, Loss: 0.0089
Epoch 6000, Loss: 0.0084
Epoch 7000, Loss: 0.0081
Epoch 8000, Loss: 0.0079
Epoch 9000, Loss: 0.0077
Test Accuracy: 0.97
```

Experiment 8: Write a program to implement k-Nearest Neighbor algorithm to classify the iris data set. Print both correct and wrong predictions.

Source Code:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
df = pd.read_csv("https://raw.githubusercontent.com/mwaskom/seaborn-data/master/iris.csv")
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)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
X_train_noisy = X_train + np.random.normal(0, 0.5, X_train.shape)
X_test_noisy = X_test + np.random.normal(0, 0.5, X_test.shape)
X_train_small, _, y_train_small, _ = train_test_split(X_train_noisy, y_train, test_size=0.5,
random_state=42)
y_train_noisy = y_train_small.copy()
flip_indices = np.random.choice(len(y_train_noisy), size=5, replace=False)
for idx in flip_indices:
   y_train_noisy.iloc[idx] = np.random.choice(y.unique())
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X_train_small, y_train_noisy)
y_pred = knn.predict(X_test_noisy)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.4f}\n")
correct_indices = np.where(y_pred == y_test)[0]
wrong_indices = np.where(y_pred != y_test)[0]
print("Correct Predictions:")
for i in correct_indices:
   print(f"Predicted: {y_pred[i]}, Actual: {y_test.iloc[i]}")
print("\nWrong Predictions:")
for i in wrong_indices:
   print(f"Predicted: {y_pred[i]}, Actual: {y_test.iloc[i]}")
```

Output:

```
Accuracy: 0.8000
Correct Predictions:
Predicted: versicolor, Actual: versicolor
Predicted: setosa, Actual: setosa
Predicted: virginica, Actual: virginica
Predicted: versicolor, Actual: versicolor
Predicted: versicolor, Actual: versicolor
Predicted: setosa, Actual: setosa
Predicted: versicolor, Actual: versicolor
```

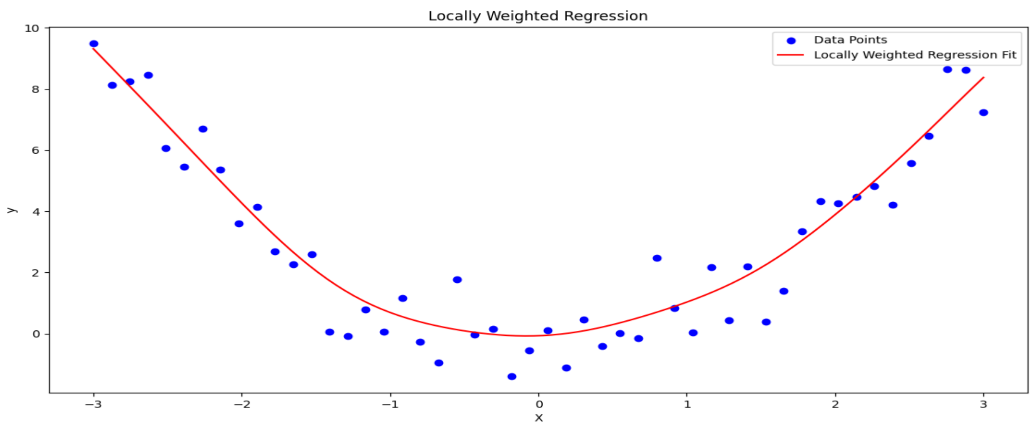
Predicted: virginica, Actual: virginica
Predicted: versicolor, Actual: versicolor
Predicted: setosa, Actual: setosa
Predicted: setosa, Actual: setosa
Predicted: setosa, Actual: setosa
Predicted: setosa, Actual: setosa
Predicted: versicolor, Actual: versicolor
Predicted: versicolor, Actual: versicolor
Predicted: virginica, Actual: virginica
Predicted: setosa, Actual: setosa
Predicted: virginica, Actual: virginica
Predicted: setosa, Actual: setosa
Predicted: virginica, Actual: virginica
Predicted: virginica, Actual: virginica
Predicted: virginica, Actual: virginica
Predicted: setosa, Actual: setosa
Predicted: setosa, Actual: setosa
Wrong Predictions:
Predicted: virginica, Actual: versicolor
Predicted: versicolor, Actual: virginica
Predicted: virginica, Actual: versicolor
Predicted: versicolor, Actual: virginica
Predicted: versicolor, Actual: virginica
Predicted: versicolor, Actual: virginica

Experiment 9: Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.

Source Code:

```
import numpy as np
import matplotlib.pyplot as plt
def kernel(x, x_point, tau):
    return np.exp(-np.sum((x - x_point) ** 2, axis=1) / (2 * tau ** 2))
def locally_weighted_regression(X, y, x_query, tau):
    m, n = X.shape
    X_bias = np.c_[np.ones(m), X]
    x_query_bias = np.array([1, x_query])
    weights = np.diag(kernel(X, x_query, tau))
    theta = np.linalg.pinv(X_bias.T @ weights @ X_bias) @ (X_bias.T @ weights @ y)
    return x_query_bias @ theta
np.random.seed(42)
X = np.linspace(-3, 3, 50)
y = X**2 + np.random.normal(0, 1, 50)
X = X.reshape(-1, 1)
y = y.reshape(-1, 1)
X_test = np.linspace(-3, 3, 100)
y_pred = [locally_weighted_regression(X, y, x, tau=0.5) for x in X_test]
plt.scatter(X, y, color='blue', label='Data Points')
plt.plot(X_test, y_pred, color='red', label='Locally Weighted Regression Fit')
plt.xlabel("X")
plt.ylabel("y")
plt.legend()
plt.title("Locally Weighted Regression")
plt.show()
```

Output:



Experiment 10: Assuming a set of documents that need to be classified, use the naïve Bayesian Classifier model to perform this task. Built-in Java classes/API can be used to write the program. Calculate the accuracy, precision, and recall for your data set.

Source Code:

```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import ComplementNB
from sklearn.metrics import accuracy_score, precision_score, recall_score
from sklearn.model_selection import train_test_split
df = pd.read_csv("documents.csv")
train_texts, test_texts, train_labels, test_labels = train_test_split(
   df['text'], df['category'], test_size=0.3, random_state=42, stratify=df['category']
)
vectorizer = TfidfVectorizer(stop_words='english')
X_train = vectorizer.fit_transform(train_texts)
X_test = vectorizer.transform(test_texts)
model = ComplementNB()
model.fit(X_train, train_labels)
predictions = model.predict(X_test)
accuracy = accuracy_score(test_labels, predictions)
precision = precision_score(test_labels, predictions, average='weighted', zero_division=1)
recall = recall_score(test_labels, predictions, average='weighted', zero_division=1)
print("\nPredictions:")
for text, actual, predicted in zip(test_texts, test_labels, predictions):
   print(f"Text: {text} | Actual: {actual} | Predicted: {predicted}")
print(f"\nAccuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
```

Dataset:

```
text,category
"The team won the championship",sports
"The football match was exciting",sports
"New AI models are revolutionizing the world",technology
"The latest smartphone has amazing features",technology
"Olympics will be held next year",sports
"The election results were surprising",politics
"The government announced a new policy",politics
"AI is transforming industries",technology
"The player scored a goal",sports
"New reforms were introduced",politics
```

Output:

```
Predictions:
Text: New AI models are revolutionizing the world | Actual: technology | Predicted: technology
Text: The team won the championship | Actual: sports | Predicted: politics
Text: New reforms were introduced | Actual: politics | Predicted: politics
Accuracy: 0.6667
Precision: 0.8333
Recall: 0.6667
```

Experiment 11: Apply EM algorithm to cluster a Heart Disease Data Set. Use the same data set for clustering using k-Means algorithm. Compare the results of these two algorithms and comment on the quality of clustering. You can add Java/Python ML library classes/API in the program.

Source code:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.mixture import GaussianMixture
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.metrics import silhouette_score
df = pd.read_csv("heart.csv")
features = ['age', 'trestbps', 'chol', 'thalach', 'oldpeak']
X = df[features]
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
kmeans = KMeans(n_clusters=2, random_state=42, n_init=10)
kmeans_labels = kmeans.fit_predict(X_scaled)
silhouette_kmeans = silhouette_score(X_scaled, kmeans_labels)
gmm = GaussianMixture(n_components=2, random_state=42)
gmm_labels = gmm.fit_predict(X_scaled)
silhouette_gmm = silhouette_score(X_scaled, gmm_labels)
print("\nComparison of Clustering Performance:")
print(f"K-Means Silhouette Score: {silhouette_kmeans:.4f}")
print(f"EM (GMM) Silhouette Score: {silhouette_gmm:.4f}")
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=kmeans_labels, cmap='viridis', edgecolors='k')
plt.title("K-Means Clustering")
plt.subplot(1, 2, 2)
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=gmm_labels, cmap='coolwarm', edgecolors='k')
plt.title("EM (GMM) Clustering")
plt.show()
```

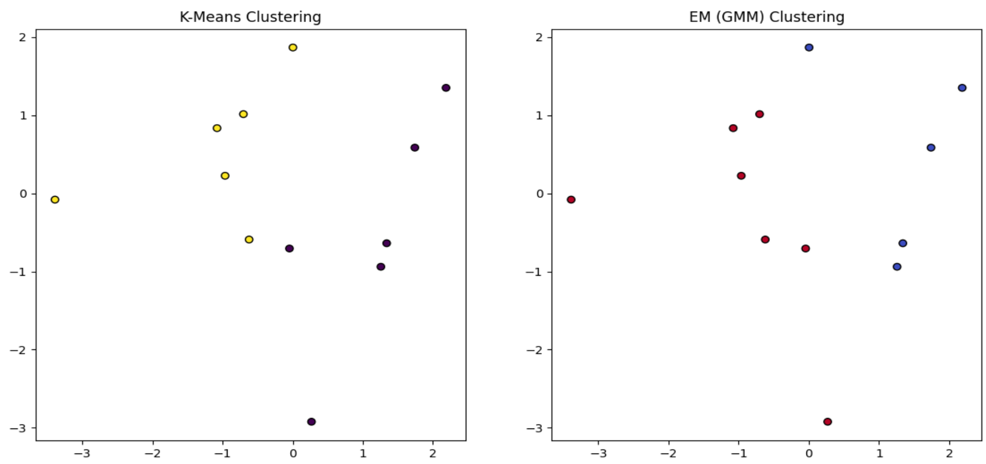
Dataset:

```
age,trestbps,chol,thalach,oldpeak
63,145,233,150,2.3
67,160,286,108,1.5
67,120,229,129,2.6
37,130,250,187,3.5
41,130,204,172,1.4
56,120,236,178,0.8
62,140,268,160,3.6
57,120,354,163,0.6
63,130,254,147,1.4
53,140,203,155,3.1
```

48,130,256,180,1.0
58,130,284,160,1.8

Output:

Comparison of Clustering Performance:
K-Means Silhouette Score: 0.2110
EM (GMM) Silhouette Score: 0.1822



Experiment 12: Exploratory Data Analysis for Classification using Pandas or Matplotlib.

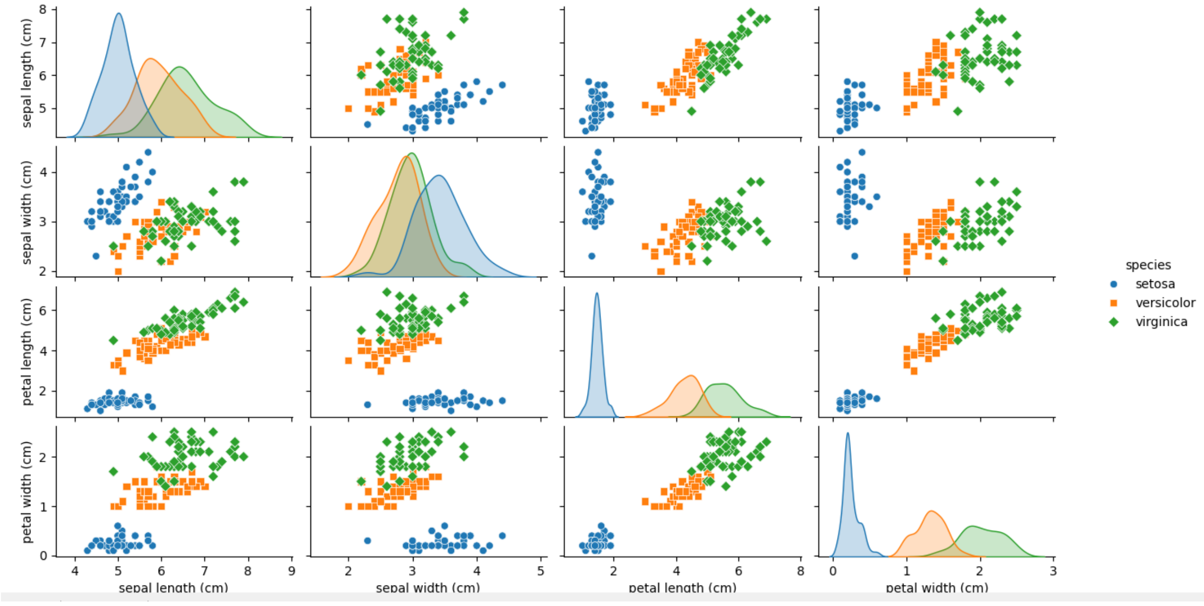
Source code:

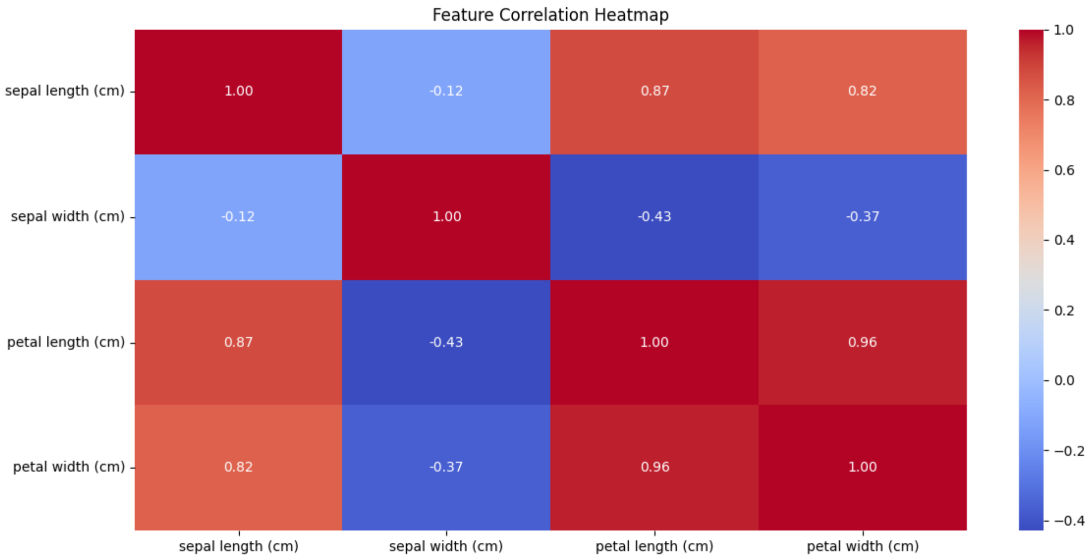
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn import datasets
iris = datasets.load_iris()
df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['species'] = iris.target
df['species'] = df['species'].map({0: 'setosa', 1: 'versicolor', 2: 'virginica'})
print("\nFirst 5 Rows:\n", df.head())
print("\nMissing Values:\n", df.isnull().sum())
print("\nSummary Statistics:\n", df.describe())
print("\nClass Distribution:\n", df['species'].value_counts())
sns.pairplot(df, hue='species', markers=["o", "s", "D"])
plt.show()
plt.figure(figsize=(8, 6))
sns.heatmap(df.iloc[:, :-1].corr(), annot=True, cmap="coolwarm", fmt=".2f")
plt.title("Feature Correlation Heatmap")
plt.show()
```

Output:

```
First 5 Rows:
   sepal length (cm)  sepal width (cm)  ...  petal width (cm)  species
0                5.1               3.5  ...               0.2   setosa
1                4.9               3.0  ...               0.2   setosa
2                4.7               3.2  ...               0.2   setosa
3                4.6               3.1  ...               0.2   setosa
4                5.0               3.6  ...               0.2   setosa
[5 rows x 5 columns]
Missing Values:
 sepal length (cm)    0
sepal width (cm)     0
petal length (cm)    0
petal width (cm)     0
species              0
dtype: int64
Summary Statistics:
        sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
count         150.000000        150.000000         150.000000        150.000000
mean            5.843333          3.057333           3.758000          1.199333
std             0.828066          0.435866           1.765298          0.762238
min             4.300000          2.000000           1.000000          0.100000
25%             5.100000          2.800000           1.600000          0.300000
50%             5.800000          3.000000           4.350000          1.300000
75%             6.400000          3.300000           5.100000          1.800000
max             7.900000          4.400000           6.900000          2.500000
```

Class Distribution:
 species
setosa   50
versicolor   50
virginica   50
Name: count, dtype: int64





Experiment 13: Write a Python program to construct a Bayesian network considering medical data. Use this model to demonstrate the diagnosis of heart patients using standard Heart Disease Data Set

Source code:

```
import pandas as pd
from pgmpy.models import BayesianNetwork
from pgmpy.estimators import MaximumLikelihoodEstimator
from pgmpy.inference import VariableElimination
df = pd.read_csv("heart1.csv")
model = BayesianNetwork([
    ('Age', 'HeartDisease'),
    ('Sex', 'HeartDisease'),
    ('ChestPainType', 'HeartDisease'),
    ('RestingBP', 'HeartDisease'),
    ('Cholesterol', 'HeartDisease'),
    ('FastingBS', 'HeartDisease'),
    ('MaxHR', 'HeartDisease'),
    ('ExerciseAngina', 'HeartDisease'),
])
model.fit(df, estimator=MaximumLikelihoodEstimator)
inference = VariableElimination(model)
evidence = {'Age': 55, 'Sex': 1, 'ChestPainType': 2, 'RestingBP': 140, 'Cholesterol': 230, 'FastingBS': 0,
'MaxHR': 150, 'ExerciseAngina': 1}
result = inference.query(variables=['HeartDisease'], evidence=evidence)
print("\nPredicted Probability of Heart Disease:\n", result)
```

Output:

```
Predicted Probability of Heart Disease:

+----------------+--------------------+

| HeartDisease  |   phi(HeartDisease) |

+===============+===================+

| HeartDisease(0) |      0.35        |

+----------------+--------------------+

| HeartDisease(1) |      0.65

| +----------------+--------------------+
```

Experiment 14: Write a program to Implement Support Vector Machines

Source code:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
iris = datasets.load_iris()
X = iris.data
y = iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
svm_model = SVC(kernel='linear', C=1.0)
svm_model.fit(X_train, y_train)
y_pred = svm_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.4f}")
print("\nClassification Report:\n", classification_report(y_test, y_pred,
target_names=iris.target_names))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nPredictions:")
for i in range(min(5, len(y_test))):
    print(f"Sample {i+1}: Predicted: {iris.target_names[y_pred[i]]}, Actual:
{iris.target_names[y_test[i]]}, {'□ Correct' if y_pred[i] == y_test[i] else '□ Wrong'}")
```

Output:

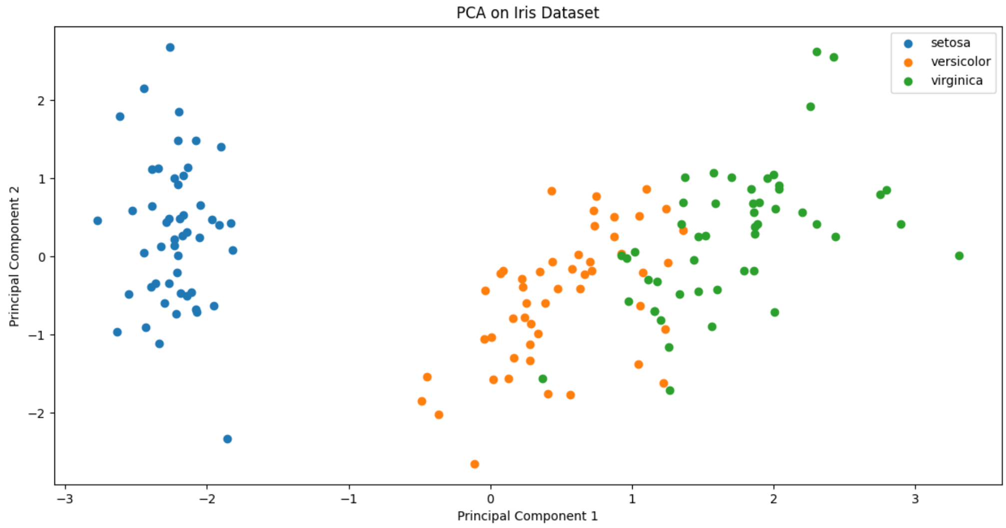
```
Accuracy: 1.0000
Classification Report:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        19
  versicolor       1.00      1.00      1.00        13
   virginica       1.00      1.00      1.00        13
    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45
Confusion Matrix:
 [[19  0  0]
 [ 0 13  0]
 [ 0  0 13]]
Predictions:
Sample 1: Predicted: versicolor, Actual: versicolor, □ Correct
Sample 2: Predicted: setosa, Actual: setosa, □ Correct
Sample 3: Predicted: virginica, Actual: virginica, □ Correct
Sample 4: Predicted: versicolor, Actual: versicolor, □ Correct
Sample 5: Predicted: versicolor, Actual: versicolor, □ Correct
```

Experiment 15: Write a program to Implement Principle Component Analysis.

Source code:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn import datasets
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
iris = datasets.load_iris()
X = iris.data
y = iris.target
target_names = iris.target_names
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)
plt.figure(figsize=(8, 6))
for i, target_name in enumerate(target_names):
   plt.scatter(X_pca[y == i, 0], X_pca[y == i, 1], label=target_name)
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.title("PCA on Iris Dataset")
plt.legend()
plt.show()
explained_variance = pca.explained_variance_ratio_
print(f"Explained Variance Ratio: {explained_variance}")
```

Output:



Explained Variance Ratio: [0.72962445 0.22850762]