day2-q5

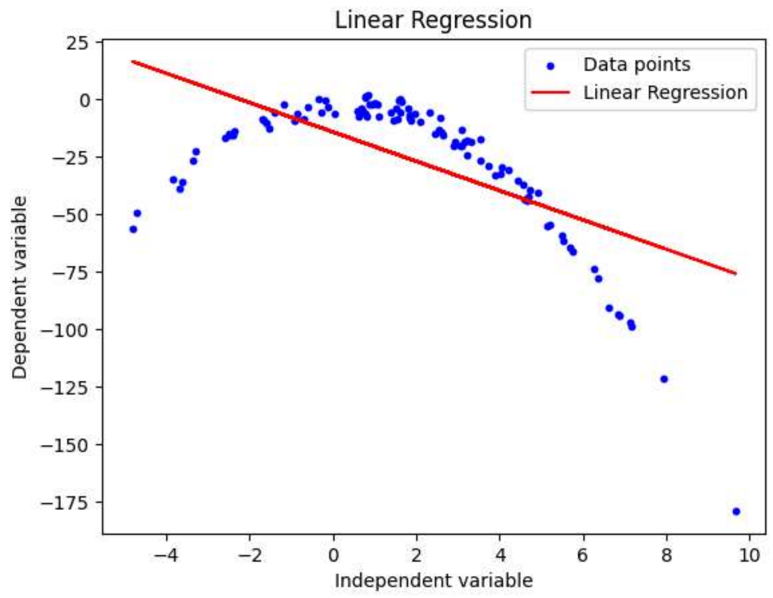
```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error, r2_score

# Predefined dataset
data = {
    'Age': [25, 30, 35, 40, 45],
    'Annual_Income': [50000, 60000, 80000, 100000, 120000],
    'Employment_Years': [3, 5, 10, 15, 20],
    'Debt_Amount': [5000, 10000, 20000, 30000, 40000],
    'Credit_History_Length': [5, 10, 15, 20, 25],
    'Number_of_Credit_Cards': [2, 3, 4, 5, 6],
    'Credit_Score': [650, 700, 750, 800, 850]
}

df = pd.DataFrame(data)

                            # Standardize features
scaler = StandardScaler()
features = df.drop('Credit_Score', axis=1)
scaled_features = scaler.fit_transform(features)

# Create a DataFrame for the scaled features
scaled_features_df = pd.DataFrame(scaled_features, columns=features.columns)
scaled_features_df['Credit_Score'] = df['Credit_Score']

                            # Feature Selection
X = scaled_features_df.drop('Credit_Score', axis=1)
y = scaled_features_df['Credit_Score']

                            # Split the Data (Here, we use the entire data for training since it's a small dataset)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

                            # KNN Algorithm
knn = KNeighborsRegressor(n_neighbors=3)
knn.fit(X_train, y_train)

                            # New data point for prediction
new_data = {
    'Age': 32,
    'Annual_Income': 70000,
    'Employment_Years': 7,
    'Debt_Amount': 15000,
    'Credit_History_Length': 12,
    'Number_of_Credit_Cards': 4
}

new_data_df = pd.DataFrame([new_data])
new_data_scaled = scaler.transform(new_data_df)

                                                # Predict the credit score
predicted_credit_score = knn.predict(new_data_scaled)
print(f'Predicted Credit Score: {predicted_credit_score[0]:.2f}')
```

```
Predicted Credit Score: 733.33
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but KNeighborsRegressor w
  warnings.warn(
```

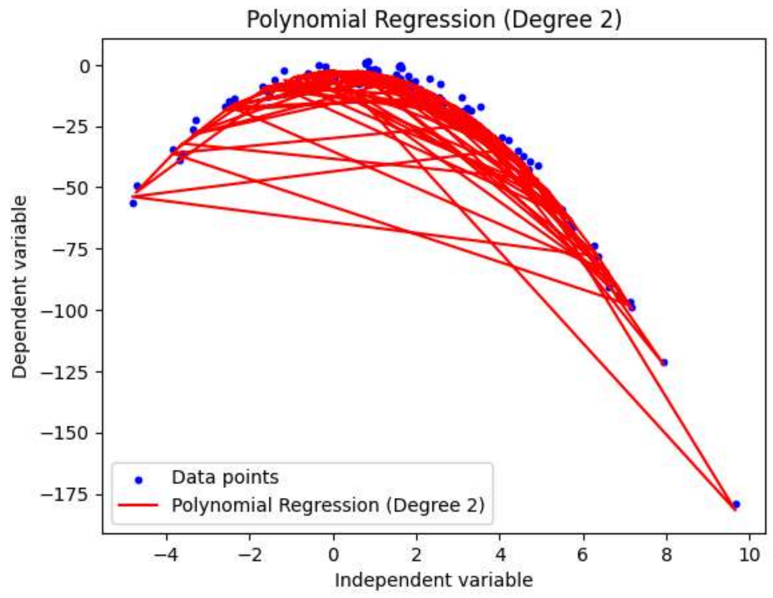
DAY2-Q7

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

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

# Reshape the data for sklearn
X = X[:, np.newaxis]

# Linear Regression
linear_regressor = LinearRegression()
linear_regressor.fit(X, y)
y_pred_linear = linear_regressor.predict(X)

# Performance metrics for Linear Regression
mse_linear = mean_squared_error(y, y_pred_linear)
r2_linear = r2_score(y, y_pred_linear)

print("Linear Regression Performance:")
print("Mean Squared Error:", mse_linear)
print("R2 Score:", r2_linear)

# Plotting Linear Regression results
plt.scatter(X, y, color='blue', s=10, label='Data points')
plt.plot(X, y_pred_linear, color='red', label='Linear Regression')
plt.xlabel("Independent variable")
plt.ylabel("Dependent variable")
plt.legend()
plt.title("Linear Regression")
plt.show()

# Polynomial Regression (degree 2)
polynomial_features = PolynomialFeatures(degree=2)
X_poly = polynomial_features.fit_transform(X)

polynomial_regressor = LinearRegression()
polynomial_regressor.fit(X_poly, y)
y_pred_poly = polynomial_regressor.predict(X_poly)

# Performance metrics for Polynomial Regression
mse_poly = mean_squared_error(y, y_pred_poly)
r2_poly = r2_score(y, y_pred_poly)

print("Polynomial Regression Performance (Degree 2):")
print("Mean Squared Error:", mse_poly)
print("R2 Score:", r2_poly)

# Plotting Polynomial Regression results
plt.scatter(X, y, color='blue', s=10, label='Data points')
plt.plot(X, y_pred_poly, color='red', label='Polynomial Regression (Degree 2)')
plt.xlabel("Independent variable")
plt.ylabel("Dependent variable")
plt.legend()
plt.title("Polynomial Regression (Degree 2)")
plt.show()
```

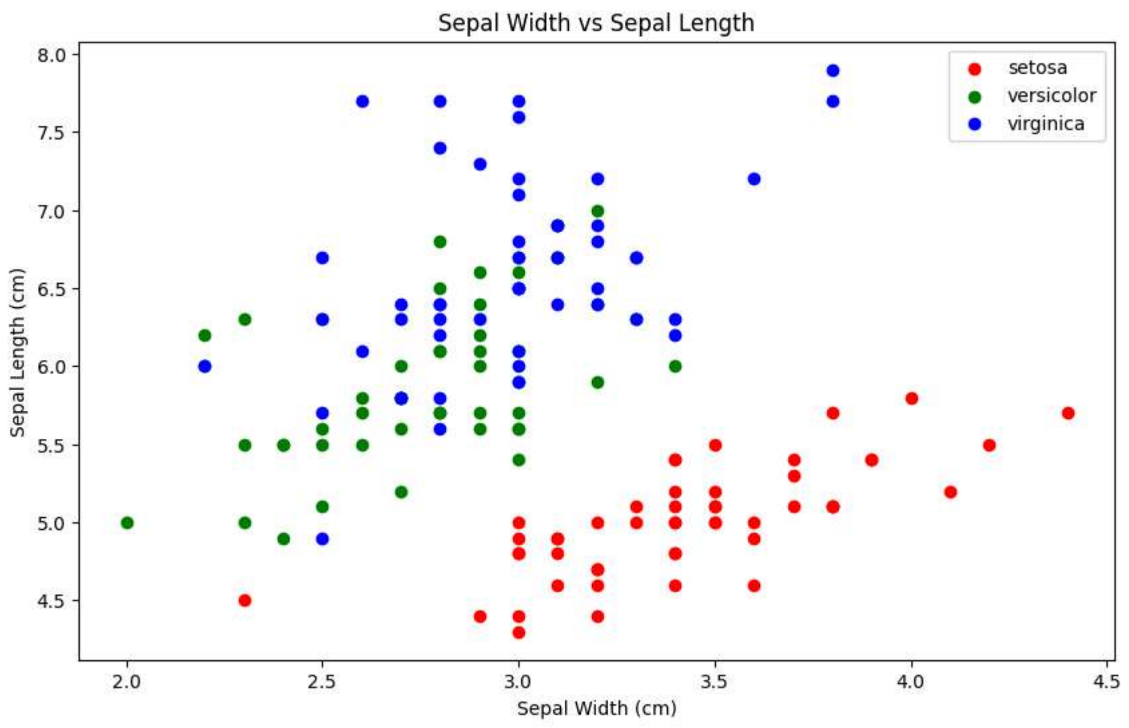
```
Linear Regression Performance:
Mean Squared Error: 563.744852984755
R2 Score: 0.3965603659439171
```



```
Polynomial Regression Performance (Degree 2):
Mean Squared Error: 9.447441952450275
R2 Score: 0.9898873384220381
```



```
# Import necessary libraries
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier

# Step a: Load the Iris dataset
iris = load_iris()
df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
df['species'] = iris.target

# Map target integers to actual species names
species_map = {0: 'setosa', 1: 'versicolor', 2: 'virginica'}
df['species'] = df['species'].map(species_map)

# Step b: Plot the data using a scatter plot
plt.figure(figsize=(10, 6))
colors = {'setosa': 'red', 'versicolor': 'green', 'virginica': 'blue'}

for species, color in colors.items():
    subset = df[df['species'] == species]
    plt.scatter(subset['sepal width (cm)'], subset['sepal length (cm)'], label=species, color=color)

plt.xlabel('Sepal Width (cm)')
plt.ylabel('Sepal Length (cm)')
plt.title('Sepal Width vs Sepal Length')
plt.legend()
plt.show()

        # Step c: 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 d: Fit the data to the model
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

        # Step e: Predict the model with new test data
new_data = [[5, 3, 1, 0.3]]
prediction = model.predict(new_data)
prediction
```



```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but RandomForestClassifie
  warnings.warn(
array(['setosa'], dtype=object)
```

```
# Define the dataset
dataset = [
    ['Circular', 'Large', 'Light', 'Smooth', 'Thick', 'Malignant'],
    ['Circular', 'Large', 'Light', 'Irregular', 'Thick', 'Malignant'],
    ['Oval', 'Large', 'Dark', 'Smooth', 'Thin', 'Benign'],
    ['Oval', 'Large', 'Light', 'Irregular', 'Thick', 'Malignant']
]

                # Extract the target concepts and attributes
attributes = dataset[0][:-1]
target = [row[-1] for row in dataset]

                # Initialize the most specific hypothesis
hypothesis = ['null'] * len(attributes)

                # Find-S Algorithm
for i, example in enumerate(dataset):
    if target[i] == 'Malignant':
       for j in range(len(hypothesis)):
           if hypothesis[j] == 'null':
              hypothesis[j] = example[j]
           elif hypothesis[j] != example[j]:
                hypothesis[j] = '?'

print("The most specific hypothesis is:", hypothesis)
```

```
The most specific hypothesis is: ['?', 'Large', 'Light', '?', 'Thick']
```

```
# Import necessary libraries
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, classification_report

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

# Step 2: Split the dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Step 3: Train a Naive Bayes classifier on the training data
model = GaussianNB()
model.fit(X_train, y_train)

# Step 4: Evaluate the classifier on the test data
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)

print("Accuracy:", accuracy)
print("\nClassification Report:")
print(classification_report(y_test, y_pred, target_names=iris.target_names))

# Step 5: Make predictions with the trained model
new_data = [[5, 3, 1, 0.3]]
prediction = model.predict(new_data)
predicted_species = iris.target_names[prediction[0]]

print("\nPredicted species for new data [5, 3, 1, 0.3]:", predicted_species)
```

```
Accuracy: 1.0

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
```

```
     Predicted species for new data [5, 3, 1, 0.3]: setosa
```

```
# Import necessary libraries
import pandas as pd
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, r2_score

# Step a: Read the house dataset using the Pandas module and print the first five rows
df = pd.read_csv('HousePricePrediction.csv')
print("First five rows of the dataset:")
print(df.head())

# Step b: Perform basic statistical computations on the dataset
print("\nBasic statistical computations:")
print(df.describe())

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

# Step d: Detect and handle null values in the dataset
print("\nNull values in the dataset:")
null_values = df.isnull().sum()
print(null_values)

# Replace null values with the mode
for column in df.columns:
    if df[column].isnull().sum() > 0:
       mode_value = df[column].mode()[0]
       df[column].fillna(mode_value, inplace=True)
 # Select only numeric columns

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

numeric_df = df.select_dtypes(include=['number'])  # Select only numeric columns
                    # Step e: Explore the dataset using a heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(numeric_df.corr(), annot=True, cmap='coolwarm')
plt.title("Heatmap of Feature Correlations")
plt.show()

                    # Step f: Split the data into training and test sets
                    # Define features and target variable

y = df['Price']

                    # Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

                    # Step g: Train a regression model to predict house prices
model = LinearRegression()
model.fit(X_train, y_train)

                    # Predict on the test set
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(f'\nMean Squared Error: {mse}')
print(f'R-squared: {r2}')

                    # Predict the price of a new house (example data)
                    # Example: new house features - [SquareFeet, Bedrooms, Bathrooms, Garage, Pool, LotSize]
new_house = [[3000, 4, 3, 1, 1, 0.5]]  # Example features
predicted_price = model.predict(new_house)
print(f'\nPredicted price for the new house: {predicted_price[0]}')
```

Show hidden output

Next steps: Explain error

```
import pandas as pd

# Load the dataset
df = pd.read_csv('example.txt')

# Extract features and target
data = df.values
features = data[:, :-1]
target = data[:, -1]

# Initialize the most specific hypothesis
specific_hypothesis = ['0'] * (features.shape[1])

# Initialize the most general hypothesis
general_hypothesis = [['?' for _ in range(features.shape[1])] for _ in range(features.shape[1])]
def consistent(hypothesis, example):
      for h, e in zip(hypothesis, example):
         if h != '?' and h != e:
                      return False
      return True
for i, example in enumerate(features):
      if target[i] == 'Malignant':
        for x in range(len(specific_hypothesis)):
                      if specific_hypothesis[x] == '0':
                         specific_hypothesis[x] = example[x]
```

Could not connect to the reCAPTCHA service. Please check your internet connection and reload to get a reCAPTCHA challenge.