

**EXPT NO: 1**                      **A python program to implement univariate regression**  
**DATE:16/08/2024**                      **bivariate regression and multivariate regression.**

### **AIM:**

To write a python program to implement univariate regression, bivariate regression and multivariate regression.

### **PROCEDURE:**

Implementing univariate, bivariate, and multivariate regression using the Iris dataset involve the following steps:

#### **Step 1: Import Necessary Libraries**

First, import the libraries that are essential for data manipulation, visualization, and model building.

```
import numpy as np

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 2: Load the Iris Dataset**

The Iris dataset can be loaded and display the first few rows of the dataset .

```
# Load the Iris dataset

iris = sns.load_dataset('iris')
```

```
# Display the first few rows of the dataset
```

```
print(iris.head())
```

## OUTPUT :

```
➡ sepal_length  sepal_width  petal_length  petal_width  species
0          5.1         3.5         1.4         0.2    setosa
1          4.9         3.0         1.4         0.2    setosa
2          4.7         3.2         1.3         0.2    setosa
3          4.6         3.1         1.5         0.2    setosa
4          5.0         3.6         1.4         0.2    setosa
```

## Step 3: Data Preprocessing

Ensure the data is clean and ready for modeling. Since the Iris dataset is clean, minimal preprocessing is needed.

```
# Check for missing values
```

```
print(iris.isnull().sum())
```

```
# Display the basic statistical details
```

```
print(iris.describe())
```

## OUTPUT :

```
➡ sepal_length    0
   sepal_width    0
   petal_length    0
   petal_width    0
   species        0
   dtype: int64

      sepal_length  sepal_width  petal_length  petal_width
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
```

## Step 4: Univariate Regression

Univariate regression involves predicting one variable based on a single predictor.

### 4.1 : Select the Features

Choose one feature (e.g., sepal\_length) and one target variable (e.g., sepal\_width).

```
X_uni = iris[['sepal_length']]  
y_uni = iris['sepal_width']
```

### 4.2 : Split the Data

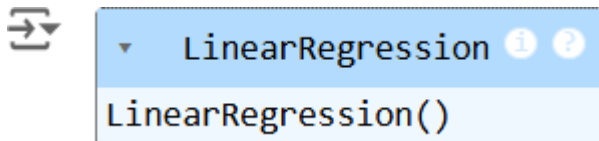
Split the data into training and testing sets.

Fit the linear regression model on the training data.

```
X_uni_train, X_uni_test, y_uni_train, y_uni_test = train_test_split(X_uni,  
y_uni,  
test_size=0.2, random_state=42)
```

### 4.3 : Train the model

```
uni_model = LinearRegression()  
uni_model.fit(X_uni_train, y_uni_train)
```



### 4.4 : Make Predictions

Use the model to make predictions on the test data.

```
y_uni_pred = uni_model.predict(X_uni_test)
```

### 4.5 : Evaluate the Model

Evaluate the model performance using metrics like Mean Squared Error (MSE) and R-squared.

```
print(f'Univariate MSE: {mean_squared_error(y_uni_test, y_uni_pred)}')  
print(f'Univariate R-squared: {r2_score(y_uni_test, y_uni_pred)}')
```

## OUTPUT :



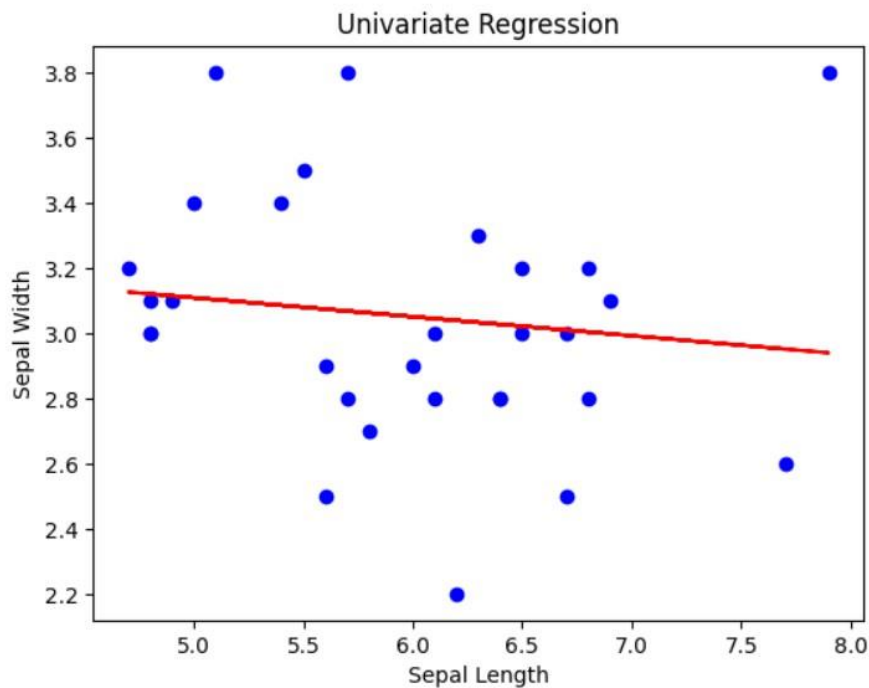
```
Univariate MSE: 0.13961895650579023  
Univariate R-squared: 0.024098626473972984
```

### 4.6 : Visualize the Results

Visualize the relationship between the predictor and the target variable.

```
plt.scatter(X_uni_test, y_uni_test, color='blue')  
plt.plot(X_uni_test, y_uni_pred, color='red')  
plt.xlabel('Sepal Length')  
plt.ylabel('Sepal Width')  
plt.title('Univariate Regression')  
plt.show()
```

## OUTPUT :



### Step 5 : Bivariate Regression

Bivariate regression involves predicting one variable based on two predictors.

#### 5.1 : Select the Features

Choose two features (e.g., sepal\_length, petal\_length) and one target variable (e.g., sepal\_width).

```
x_bi = iris[['sepal_length', 'petal_length']]  
  
y_bi = iris['sepal_width']
```

## 5.2: Split the Data

Split the data into training and testing sets.

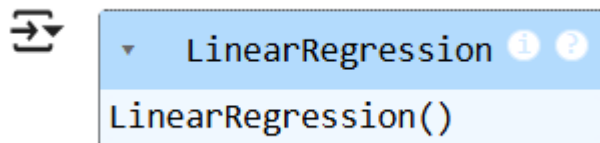
```
x_bi_train, x_bi_test, y_bi_train, y_bi_test = train_test_split(x_bi, y_bi,  
test_size=0.2, random_state=42)
```

## 5.3: Train the Model

Fit the linear regression model on the training data.

```
bi_model = LinearRegression()  
  
bi_model.fit(x_bi_train, y_bi_train)
```

### OUTPUT :



## 5.4: Make Predictions

Use the model to make predictions on the test data.

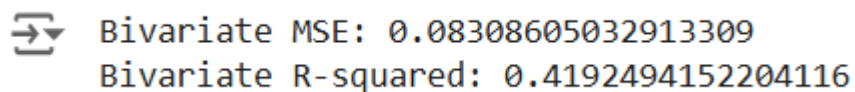
```
y_bi_pred = bi_model.predict(x_bi_test)
```

## 5.5: Evaluate the Model

Evaluate the model performance using metrics like MSE and R-squared.

```
print(f'Bivariate MSE: {mean_squared_error(y_bi_test, y_bi_pred)}')  
print(f'Bivariate R-squared: {r2_score(y_bi_test, y_bi_pred)}')
```

### OUTPUT :

A screenshot of a Jupyter Notebook interface. On the left, there is a blue icon with a right-pointing arrow. To its right is a code cell with a blue header bar containing a dropdown arrow, the text 'Bivariate MSE: 0.08308605032913309', and two circular icons (one with an 'i' and one with a '?'). Below the header bar, the text 'Bivariate R-squared: 0.4192494152204116' is displayed in a monospaced font.

## 5.6: Visualize the Results

Since visualizing in 3D is challenging, we can plot the relationships between the target and each predictor separately.

```
# Sepal Length vs Sepal Width

plt.subplot(1, 2, 1)

plt.scatter(X_bi_test['sepal_length'], y_bi_test, color='blue')

plt.plot(X_bi_test['sepal_length'], y_bi_pred, color='red')

plt.xlabel('Sepal Length')

plt.ylabel('Sepal Width')

# Petal Length vs Sepal Width

plt.subplot(1, 2, 2)

plt.scatter(X_bi_test['petal_length'], y_bi_test, color='blue')

plt.plot(X_bi_test['petal_length'], y_bi_pred, color='red')

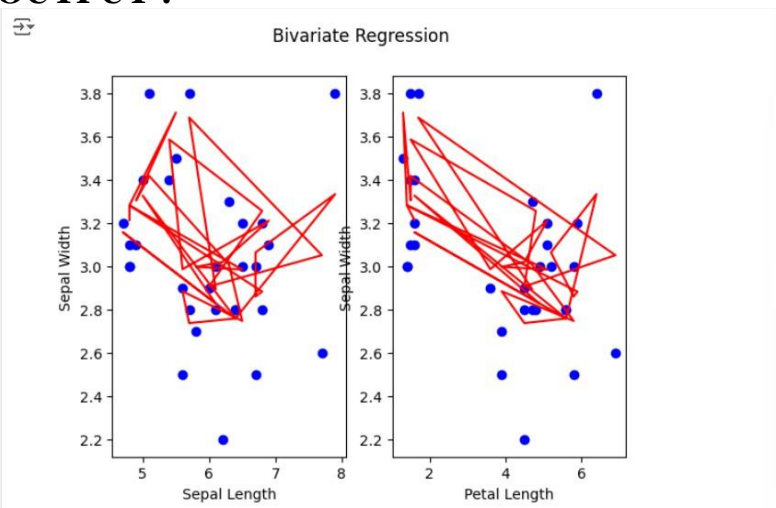
plt.xlabel('Petal Length')

plt.ylabel('Sepal Width')

plt.suptitle('Bivariate Regression')

plt.show()
```

### OUTPUT :



## Step 6: Multivariate Regression

Multivariate regression involves predicting one variable based on multiple predictors.

### 6.1: Select the Features

Choose multiple features (e.g., sepal\_length, petal\_length, petal\_width) and one target variable (e.g., sepal\_width).

```
X_multi = iris[['sepal_length', 'petal_length', 'petal_width']]  
  
y_multi = iris['sepal_width']
```

### 6.2: Split the Data

Split the data into training and testing sets.

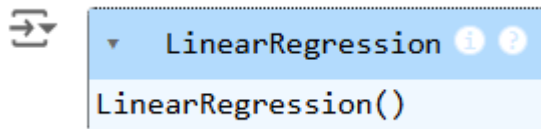
```
X_multi_train, X_multi_test, y_multi_train, y_multi_test =  
train_test_split(X_multi,  
  
y_multi, test_size=0.2, random_state=42)
```

### 6.3: Train the Model

Fit the linear regression model on the training data.

```
multi_model = LinearRegression()  
multi_model.fit(X_multi_train, y_multi_train)
```

### OUTPUT :



### 6.4: Make Predictions

Use the model to make predictions on the test data.

```
y_multi_pred = multi_model.predict(X_multi_test)
```

## 6.5: Evaluate the Model

Evaluate the model performance using metrics like MSE and R-squared.

```
print(f'Multivariate MSE: {mean_squared_error(y_multi_test, y_multi_pred)}')
print(f'Multivariate R-squared: {r2_score(y_multi_test, y_multi_pred)}')
```

### OUTPUT :

```
➡ Multivariate MSE: 0.0868353771078583
   Multivariate R-squared: 0.39304256448374897
```

## Step 7: Visualize the multivariate regression

```
plt.figure(figsize=(15,4))

plt.subplot(1, 2, 1)

plt.scatter(X_multi_test['sepal_length'], y_multi_test, color='blue')
plt.plot(X_multi_test['sepal_length'], y_multi_pred, color='red')

plt.xlabel('sepal_length')
plt.ylabel('sepal_width')

plt.title('Multivariate Regression-1')

plt.show()

plt.figure(figsize=(15,4))

plt.subplot(1, 2, 1)

plt.scatter(X_multi_test['petal_length'], y_multi_test, color='blue')
plt.plot(X_multi_test['petal_length'], y_multi_pred, color='red')

plt.xlabel('petal_length')
plt.ylabel('sepal_width')

plt.title('Multivariate Regression-2')

plt.show()
```



```
plt.figure(figsize=(15,4))

plt.subplot(1, 2, 2 )

plt.scatter(X_multi_test['petal_length'], y_multi_test, color='blue')

plt.plot(X_multi_test['petal_length'], y_multi_pred, color='red')

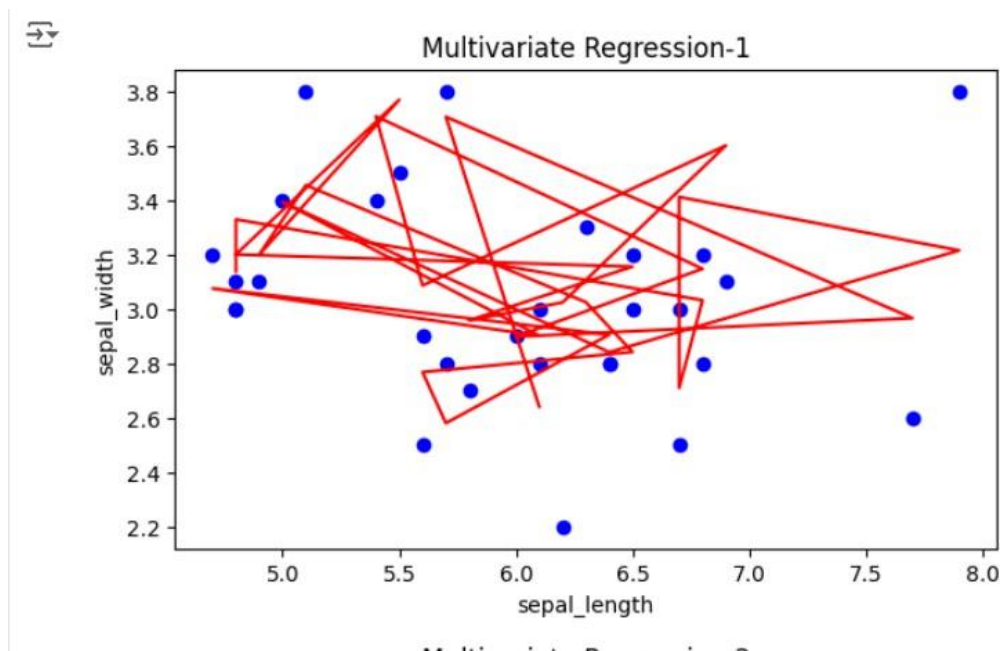
plt.xlabel('petal_length')

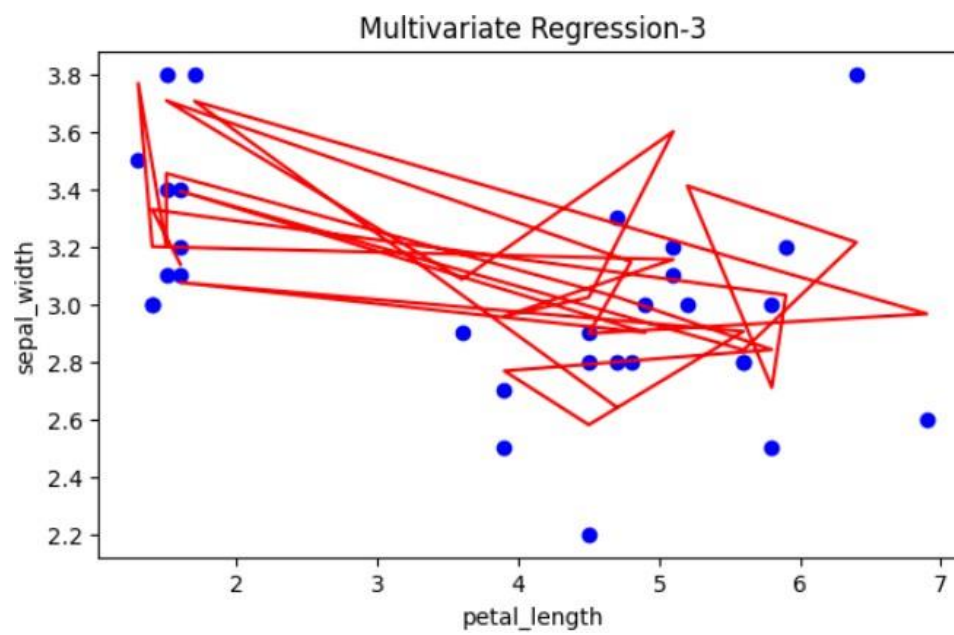
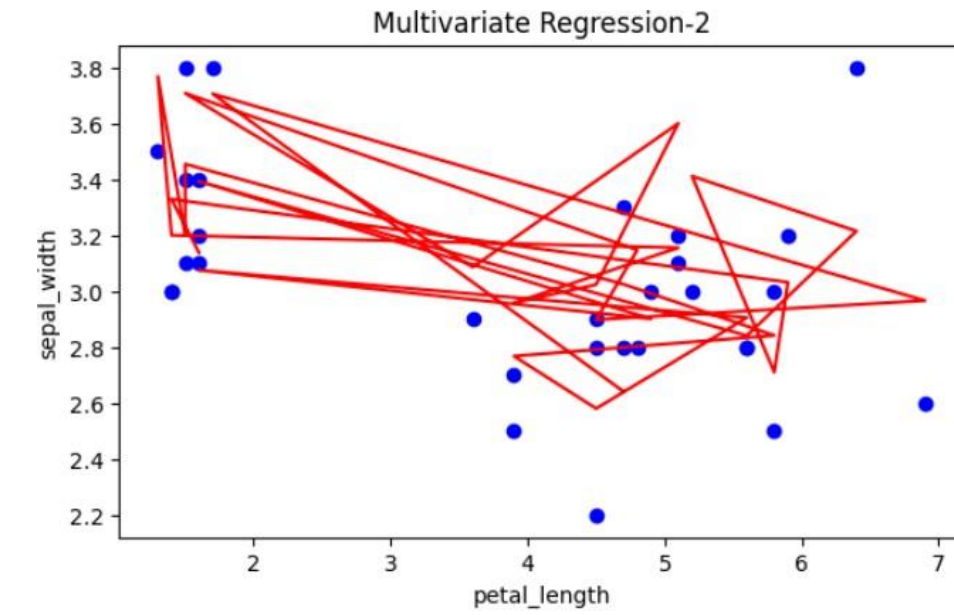
plt.ylabel('sepal_width')

plt.title('Multivariate Regression-3')

plt.show()
```

## OUTPUT :





## Step 8: Interpret the Results

After implementing and evaluating the models, interpret the coefficients to understand the influence of each predictor on the target variable.

```
print('Univariate Coefficients:', uni_model.coef_)  
print('Bivariate Coefficients:', bi_model.coef_)  
print('Multivariate Coefficients:', multi_model.coef_)
```

### OUTPUT :

```
⇒ Univariate Coefficients: [-0.05829418]  
   Bivariate Coefficients: [ 0.56420418 -0.33942806]  
   Multivariate Coefficients: [ 0.62934965 -0.63196673  0.6440201 ]
```

### RESULT:

This step-by-step process will help us to implement univariate, bivariate, and

multivariate regression models using the Iris dataset and analyze their performance.