

## Assignment on **Linear Regression**



Course name: Data Science

Course code: CSEL – 42---

**Submitted By**

**Farhana Akter Suci**

ID: B190305001

**&**

**Rifah Sajida Deya**

ID: B190305004

**Submitted To**

**Dr. Md. Manowarul Islam**

Associate Professor, Department of C S E, Jagannath University

**16<sup>th</sup> January , 2025**

## ✓ Task 1

### ✓ 1.Dataset Prepearation

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

```
np.random.seed(42)
```

```
x = np.random.rand(40) * 7
```

```
y= 4*x + np.random.randn(40) * 3
```

```
data = pd.DataFrame({'x': x, 'y': y})
```

```
data
```

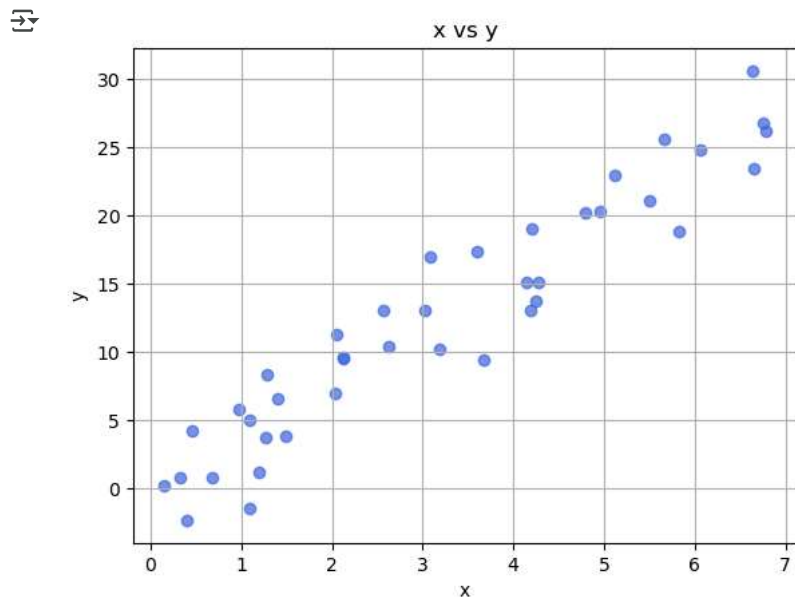


	x	y
0	2.621781	10.446632
1	6.655000	23.446868
2	5.123958	22.963465
3	4.190609	13.099907
4	1.092130	4.995113
5	1.091962	-1.511164
6	0.406585	-2.358217
7	6.063233	24.843516
8	4.207805	19.046620
9	4.956508	20.340137
10	0.144091	0.229421
11	6.789369	26.254165
12	5.827098	18.872828
13	1.486374	3.785962
14	1.272775	3.709183
15	1.283832	8.306693
16	2.129696	9.549638
17	3.673295	9.404060
18	3.023615	13.066712
19	2.038604	6.999169
20	4.282970	15.101115
21	0.976457	5.740857
22	2.045013	11.273049
23	2.564533	13.051972
24	3.192490	10.252307
25	5.496232	21.057290
26	1.397716	6.584656
27	3.599641	17.325200
28	4.146902	15.150085
29	0.325153	0.743635
30	4.252814	13.692251
31	1.193669	1.186056
32	0.455361	4.259022
33	6.642199	30.637515
34	6.759424	26.821667
35	5.658781	25.645724
36	2.132296	9.614094
37	0.683705	0.799460
38	4.789631	20.242712
39	3.081067	16.938380

## 2.Data Visualization

```
import matplotlib.pyplot as plt
```

```
# Scatter plot
plt.scatter(data['x'], data['y'], color='royalblue', alpha=0.7)
plt.title('x vs y')
plt.xlabel('x')
plt.ylabel('y')
plt.grid()
plt.show()
```



### ✓ 3. Linear Regression Implementation

#### ✓ number of data points

```
n= len(data)
```

#### ✓ Linear Regression Formula

Formula used for linear regressions is,  $y = mx + c$

calculate slope (m) and intercept (c) using formulas

$$m = \frac{n \sum(xy) - \sum x \sum y}{n \sum x^2 - (\sum x)^2}$$

$$c = \frac{(\sum y \sum x^2 - \sum x \sum xy)}{(n \sum x^2 - (\sum x)^2)}$$


y is Dependent Variable that Lies along Y-axis

c is Y-Intercept


m is Slope of Regression Line

#### ✓ x is Independent Variable that Lies along X-axis

```
sum_x= np.sum(x)
sum_x
```

 127.75437537361086


```
sum_y= np.sum(y)
sum_y
```

 501.60775122280336

```
sum_xy= np.sum(x*y)
sum_xy
```

 2279.9645395828084

```
sum_x2= np.sum(x**2)
sum_x2
```

 575.1074996478538

```
data1 = pd.DataFrame({'x': x,
                      'y': y,
                      'x2':(x**2),
                      'x*y':(x*y)})
```

```
data2 = pd.DataFrame({'Summation(x)': [np.sum(x)],
                      'Summation(y)': [np.sum(y)],
                      'Summation(x2)': [np.sum((x**2))],
                      'Summation(x*y)': [np.sum((x*y))])})
```

data1



	x	y	x2	x*y
0	2.621781	10.446632	6.873735	27.388779
1	6.655000	23.446868	44.289027	156.038909
2	5.123958	22.963465	26.254941	117.663821
3	4.190609	13.099907	17.561207	54.896592
4	1.092130	4.995113	1.192749	5.455315
5	1.091962	-1.511164	1.192380	-1.650133
6	0.406585	-2.358217	0.165312	-0.958816
7	6.063233	24.843516	36.762795	150.632025
8	4.207805	19.046620	17.705624	80.144465
9	4.956508	20.340137	24.566972	100.816053
10	0.144091	0.229421	0.020762	0.033058
11	6.789369	26.254165	46.095531	178.249212
12	5.827098	18.872828	33.955077	109.973827
13	1.486374	3.785962	2.209307	5.627355
14	1.272775	3.709183	1.619956	4.720954
15	1.283832	8.306693	1.648223	10.664395
16	2.129696	9.549638	4.535604	20.337822
17	3.673295	9.404060	13.493096	34.543885
18	3.023615	13.066712	9.142248	39.508709
19	2.038604	6.999169	4.155906	14.268534
20	4.282970	15.101115	18.343834	64.677627
21	0.976457	5.740857	0.953468	5.605700
22	2.045013	11.273049	4.182076	23.053526
23	2.564533	13.051972	6.576829	33.472212
24	3.192490	10.252307	10.191992	32.730386
25	5.496232	21.057290	30.208563	115.735744
26	1.397716	6.584656	1.953611	9.203482
27	3.599641	17.325200	12.957416	62.364500
28	4.146902	15.150085	17.196796	62.825918
29	0.325153	0.743635	0.105724	0.241795
30	4.252814	13.692251	18.086427	58.230596
31	1.193669	1.186056	1.424845	1.415758
32	0.455361	4.259022	0.207354	1.939393
33	6.642199	30.637515	44.118804	203.500465
34	6.759424	26.821667	45.689816	181.299023
35	5.658781	25.645724	32.021807	145.123549
36	2.132296	9.614094	4.546688	20.500097
37	0.683705	0.799460	0.467452	0.546595
38	4.789631	20.242712	22.940567	96.955123
39	3.081067	16.938380	9.492977	52.188290

data2



	Summation(x)	Summation(y)	Summation(x2)	Summation(x*y)
0	127.754375	501.607751	575.1075	2279.96454

## ✓ Slope (m)

```
m = (n*sum_xy - sum_x*sum_y) / (n* sum_x2 - sum_x**2)
m
```

```
↗ 4.057386136018626
```

## ✓ Intercept (c)

```
c=(sum_y - m * sum_x) / n
c
```

```
↗ -0.4185270058451181
```

```
print(f"Slope (m): {m}")
print(f"Intercept (c): {c}")
```

```
↗ Slope (m): 4.057386136018626
Intercept (c): -0.4185270058451181
```

## ✓ Predict Y values

```
y_pred= m * x + c
y_pred
```

```
↗ array([10.21905019, 26.58337832, 20.37134749, 16.58439343,  4.01266807,
         4.01198302,  1.23114649, 24.18235059, 16.654163  , 19.69194002,
         0.16610769, 27.12856451, 23.2242616  ,  5.61226534,  4.7456117  ,
         4.7904734  ,  8.2224708  , 14.48544929, 11.84944711,  7.85287653,
        16.95913716,  3.54333619,  7.87887852,  9.98677324, 12.53463721,
        21.88180741,  5.25254844, 14.18660676, 16.4070556  ,  0.90074382,
        16.83678141,  4.4246485  ,  1.42904901, 26.53143816, 27.00706716,
        22.54133434,  8.23302278,  2.35552736, 19.01485616, 12.08255337])
```

```
pd.DataFrame({'y_Actual':y,
              'y_predicted (y=mx+c)':y_pred}
            )
```



	y_Actual	y_predicted (y=mx+c)
0	10.446632	10.219050
1	23.446868	26.583378
2	22.963465	20.371347
3	13.099907	16.584393
4	4.995113	4.012668
5	-1.511164	4.011983
6	-2.358217	1.231146
7	24.843516	24.182351
8	19.046620	16.654163
9	20.340137	19.691940
10	0.229421	0.166108
11	26.254165	27.128565
12	18.872828	23.224262
13	3.785962	5.612265
14	3.709183	4.745612
15	8.306693	4.790473
16	9.549638	8.222471
17	9.404060	14.485449
18	13.066712	11.849447
19	6.999169	7.852877
20	15.101115	16.959137
21	5.740857	3.543336
22	11.273049	7.878879
23	13.051972	9.986773
24	10.252307	12.534637
25	21.057290	21.881807
26	6.584656	5.252548
27	17.325200	14.186607
28	15.150085	16.407056
29	0.743635	0.900744
30	13.692251	16.836781
31	1.186056	4.424649
32	4.259022	1.429049
33	30.637515	26.531438
34	26.821667	27.007067
35	25.645724	22.541334
36	9.614094	8.233023
37	0.799460	2.355527
38	20.242712	19.014856
39	16.938380	12.082553


#### 4. Coefficient of Determination ( $R^2$ )

R: It represents the proportion of the variance in Y explained by X. A value close to 1 indicates a strong model fit.



## ✓ Residual sum of squares

```
rss= np.sum((y - y_pred)**2)
rss
```

 278.60274624620354


## ✓ Total sum of squares

```
tss= np.sum((y - np.mean(y))**2)
tss
```

 3029.1044673656506

## ✓ $R^2$

```
R_squared= 1 - (rss/tss)
print(f"Coefficient of Determination ( $R^2$ ): {R_squared}")
```

 Coefficient of Determination ( $R^2$ ): 0.9080247151434501

## ✓ 5. Predictions and Visualization

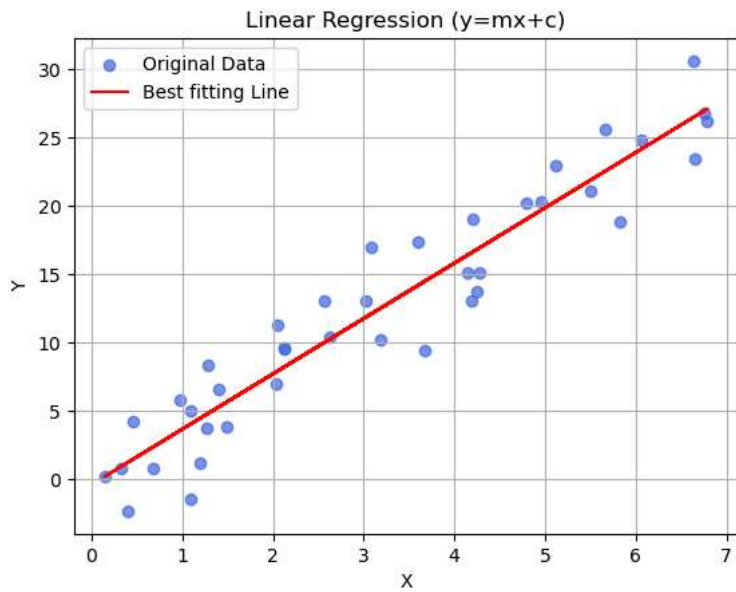
```
pd.DataFrame({ 'x':x,
               'y':y,
               'predicted (y=mx+c)':y_pred}
             )
```



	x	y	predicted (y=mx+c)
0	2.621781	10.446632	10.219050
1	6.655000	23.446868	26.583378
2	5.123958	22.963465	20.371347
3	4.190609	13.099907	16.584393
4	1.092130	4.995113	4.012668
5	1.091962	-1.511164	4.011983
6	0.406585	-2.358217	1.231146
7	6.063233	24.843516	24.182351
8	4.207805	19.046620	16.654163
9	4.956508	20.340137	19.691940
10	0.144091	0.229421	0.166108
11	6.789369	26.254165	27.128565
12	5.827098	18.872828	23.224262
13	1.486374	3.785962	5.612265
14	1.272775	3.709183	4.745612
15	1.283832	8.306693	4.790473
16	2.129696	9.549638	8.222471
17	3.673295	9.404060	14.485449
18	3.023615	13.066712	11.849447
19	2.038604	6.999169	7.852877
20	4.282970	15.101115	16.959137
21	0.976457	5.740857	3.543336
22	2.045013	11.273049	7.878879
23	2.564533	13.051972	9.986773
24	3.192490	10.252307	12.534637
25	5.496232	21.057290	21.881807
26	1.397716	6.584656	5.252548
27	3.599641	17.325200	14.186607
28	4.146902	15.150085	16.407056
29	0.325153	0.743635	0.900744
30	4.252814	13.692251	16.836781
31	1.193669	1.186056	4.424649
32	0.455361	4.259022	1.429049
33	6.642199	30.637515	26.531438
34	6.759424	26.821667	27.007067
35	5.658781	25.645724	22.541334
36	2.132296	9.614094	8.233023
37	0.683705	0.799460	2.355527
38	4.789631	20.242712	19.014856
39	3.081067	16.938380	12.082553

```
# scatter plot and regression line
plt.scatter(x, y, color='royalblue', alpha=0.7, label='Original Data')
plt.plot(x, y_pred, color='red', label='Best fitting Line')
plt.title('Linear Regression (y=mx+c)')
plt.xlabel('X')
plt.ylabel('Y')
plt.legend()
```

```
plt.grid()
plt.show()
```



## 6. Residual Analysis

✓ Residual = Observed value – predicted value

```
pd.DataFrame({'Observed value':y,
              'Predicted value':y_pred}
            )
```



	Observed value	Predicted value
0	10.446632	10.219050
1	23.446868	26.583378
2	22.963465	20.371347
3	13.099907	16.584393
4	4.995113	4.012668
5	-1.511164	4.011983
6	-2.358217	1.231146
7	24.843516	24.182351
8	19.046620	16.654163
9	20.340137	19.691940
10	0.229421	0.166108
11	26.254165	27.128565
12	18.872828	23.224262
13	3.785962	5.612265
14	3.709183	4.745612
15	8.306693	4.790473
16	9.549638	8.222471
17	9.404060	14.485449
18	13.066712	11.849447
19	6.999169	7.852877
20	15.101115	16.959137
21	5.740857	3.543336
22	11.273049	7.878879
23	13.051972	9.986773
24	10.252307	12.534637
25	21.057290	21.881807
26	6.584656	5.252548
27	17.325200	14.186607
28	15.150085	16.407056
29	0.743635	0.900744
30	13.692251	16.836781
31	1.186056	4.424649
32	4.259022	1.429049
33	30.637515	26.531438
34	26.821667	27.007067
35	25.645724	22.541334
36	9.614094	8.233023
37	0.799460	2.355527
38	20.242712	19.014856
39	16.938380	12.082553

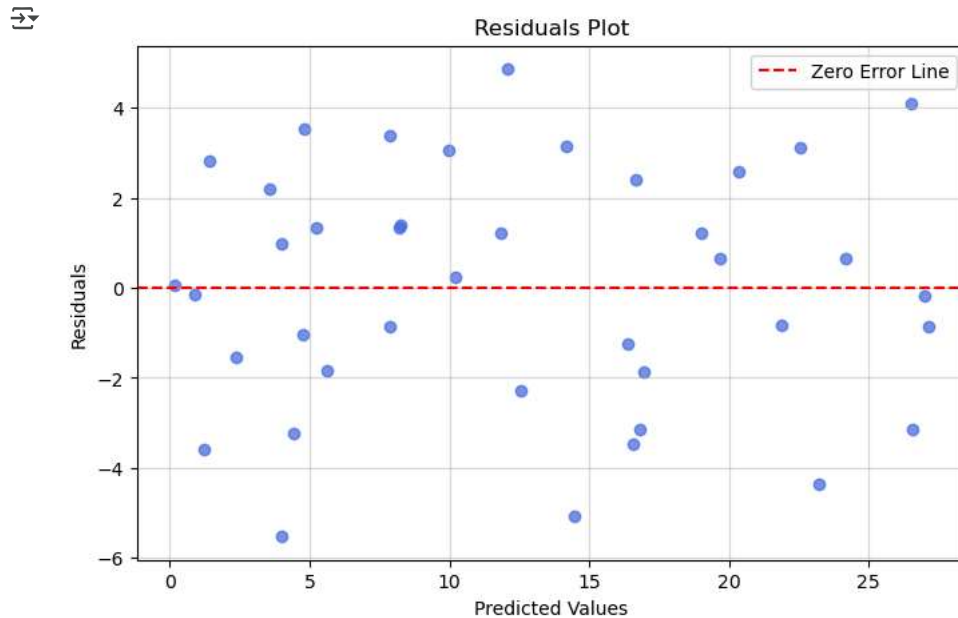
```
residuals = y-y_pred  
residuals
```



```
array([ 0.22758146, -3.13651052,  2.59211761, -3.48448682,  0.98244464,  
       -5.52314683, -3.5893635 ,  0.6611652 ,  2.39245707,  0.648197 ,  
        0.06331331, -0.87439973, -4.35143363, -1.82630287, -1.03642893,  
        3.51621955,  1.32716687, -5.08138967,  1.21726532, -0.85370744,
```

```
-1.85802211, 2.19752078, 3.39417021, 3.06519873, -2.28233022,
-0.82451762, 1.33210775, 3.1385929, -1.25697039, -0.15710919,
-3.14453048, -3.23859291, 2.82997306, 4.10607697, -0.1854006,
3.1043901, 1.38107083, -1.55606743, 1.2278554, 4.85582615])
```

```
plt.figure(figsize=(8, 5))
plt.scatter(y_pred, residuals,color='royalblue', alpha=0.7)
plt.axhline(0, color='red', linestyle='--', label='Zero Error Line')
plt.title("Residuals Plot")
plt.xlabel("Predicted Values")
plt.ylabel("Residuals")
plt.legend()
plt.grid(alpha=0.5)
plt.show()
```



A residual plot is a key diagnostic tool in linear regression analysis. It visualizes the residuals (differences

- ✓ between observed and predicted values) on the y-axis against the predicted values or another variable (such as an independent variable) on the x-axis.

```
pd.DataFrame({'Observed value':y,
              'Predicted value':y_pred,
              'Residuals':residuals}
)
```



	Observed value	Predicted value	Residuals
0	10.446632	10.219050	0.227581
1	23.446868	26.583378	-3.136511
2	22.963465	20.371347	2.592118
3	13.099907	16.584393	-3.484487
4	4.995113	4.012668	0.982445
5	-1.511164	4.011983	-5.523147
6	-2.358217	1.231146	-3.589363
7	24.843516	24.182351	0.661165
8	19.046620	16.654163	2.392457
9	20.340137	19.691940	0.648197
10	0.229421	0.166108	0.063313
11	26.254165	27.128565	-0.874400
12	18.872828	23.224262	-4.351434
13	3.785962	5.612265	-1.826303
14	3.709183	4.745612	-1.036429
15	8.306693	4.790473	3.516220
16	9.549638	8.222471	1.327167
17	9.404060	14.485449	-5.081390
18	13.066712	11.849447	1.217265
19	6.999169	7.852877	-0.853707
20	15.101115	16.959137	-1.858022
21	5.740857	3.543336	2.197521
22	11.273049	7.878879	3.394170
23	13.051972	9.986773	3.065199
24	10.252307	12.534637	-2.282330
25	21.057290	21.881807	-0.824518
26	6.584656	5.252548	1.332108
27	17.325200	14.186607	3.138593
28	15.150085	16.407056	-1.256970
29	0.743635	0.900744	-0.157109
30	13.692251	16.836781	-3.144530
31	1.186056	4.424649	-3.238593
32	4.259022	1.429049	2.829973
33	30.637515	26.531438	4.106077
34	26.821667	27.007067	-0.185401
35	25.645724	22.541334	3.104390
36	9.614094	8.233023	1.381071
37	0.799460	2.355527	-1.556067
38	20.242712	19.014856	1.227855
39	16.938380	12.082553	4.855826

## 7. Model Evaluation

```
# error metrics
mae = np.mean(np.abs(residuals))
mse = np.mean(residuals**2)
```

```
rmse = np.sqrt(mse)

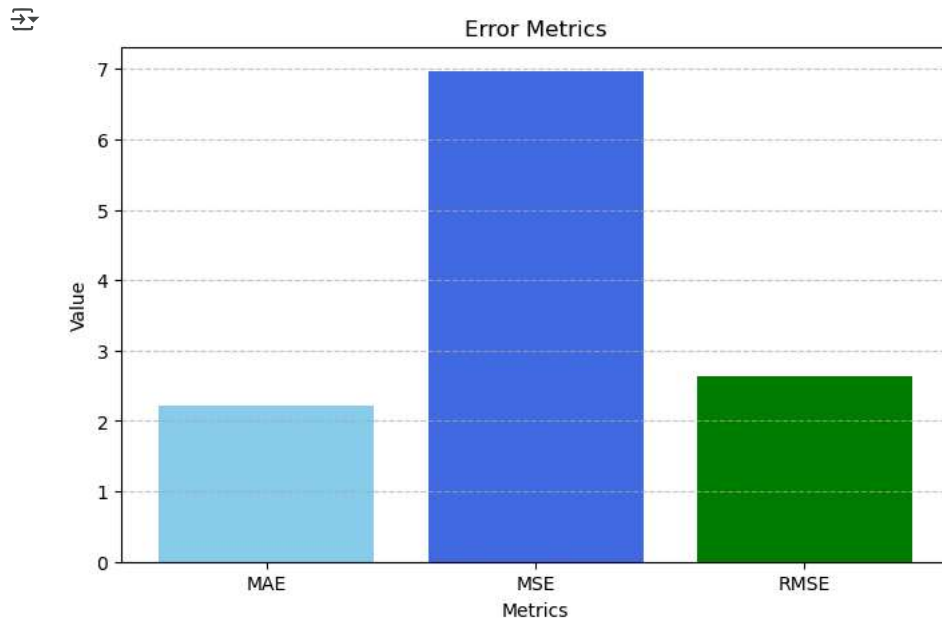
print(f"Mean Absolute Error (MAE): {mae}")
print(f"Mean Squared Error (MSE): {mse}")
print(f"Root Mean Squared Error (RMSE): {rmse}")
```

↗ Mean Absolute Error (MAE): 2.2130355450395514  
Mean Squared Error (MSE): 6.965068656155088  
Root Mean Squared Error (RMSE): 2.639141651400146

```
metrics = ['MAE', 'MSE', 'RMSE']
values = [mae, mse, rmse]
```

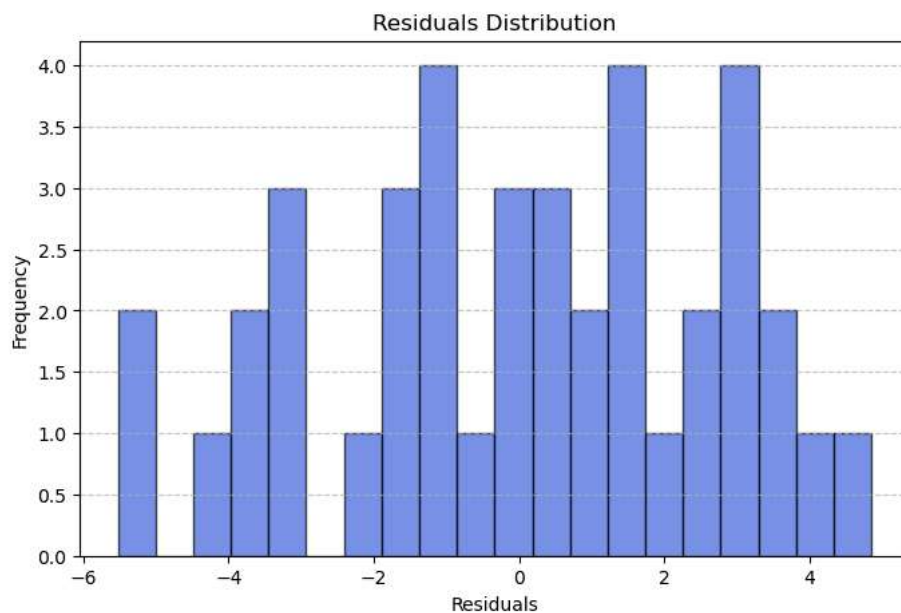
## ✓ Plotting the error metrics as a bar chart

```
plt.figure(figsize=(8, 5))
plt.bar(metrics, values, color=['skyblue', 'royalblue', 'green'])
plt.title("Error Metrics")
plt.ylabel("Value")
plt.xlabel("Metrics")
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```



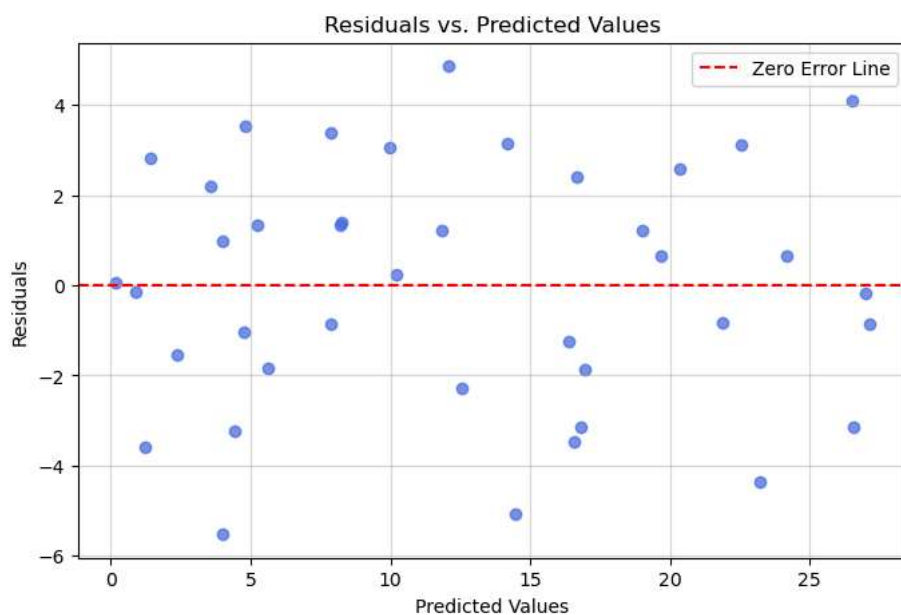
## ✓ Error distribution plot

```
plt.figure(figsize=(8, 5))
plt.hist(residuals, bins=20, color='royalblue', alpha=0.7, edgecolor='black')
plt.title("Residuals Distribution")
plt.xlabel("Residuals")
plt.ylabel("Frequency")
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```



## Residuals vs. predicted values (example predicted data)

```
plt.figure(figsize=(8, 5))
plt.scatter(y_pred, residuals,color='royalblue', alpha=0.7)
plt.axhline(0, color='red', linestyle='--', label='Zero Error Line')
plt.title("Residuals vs. Predicted Values")
plt.xlabel("Predicted Values")
plt.ylabel("Residuals")
plt.legend()
plt.grid(alpha=0.5)
plt.show()
```



End of Task 1

## Task 2



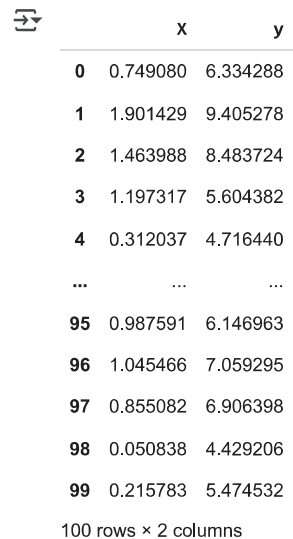
```
import numpy as np
import pandas as pd
```

## ✓ 1. Dataset Preparation

```
np.random.seed(42)
x = 2 * np.random.rand(100, 1)
y = 4 + 3 * x + np.random.randn(100, 1)
```

```
data = pd.DataFrame({
    'X': x.flatten(), # Flatten the array for a single column
    'y': y.flatten()  # Flatten the array for a single column
})
```

data



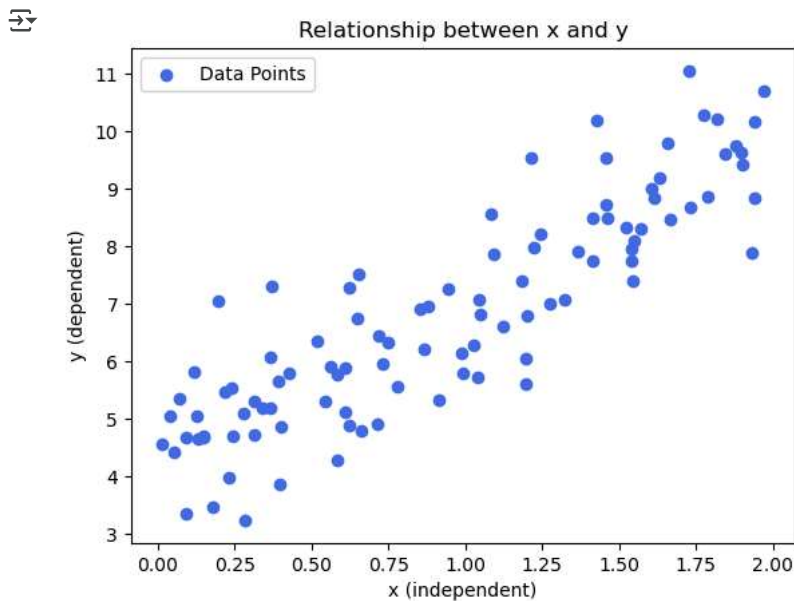
	X	y
0	0.749080	6.334288
1	1.901429	9.405278
2	1.463988	8.483724
3	1.197317	5.604382
4	0.312037	4.716440
...	...	...
95	0.987591	6.146963
96	1.045466	7.059295
97	0.855082	6.906398
98	0.050838	4.429206
99	0.215783	5.474532

100 rows × 2 columns

## ✓ 2. Data Visualization

```
import matplotlib.pyplot as plt
```

```
plt.scatter(x,y,color='royalblue',label='Data Points')
plt.title('Relationship between x and y')
plt.xlabel('x (independent)')
plt.ylabel('y (dependent)')
plt.legend()
plt.show()
```



### ✓ 3. Cost Function (Mean Squared Error)

In linear regression, the cost function quantifies the error between predicted values and actual data points. It is a measure of how far off a linear model's predictions are from the actual values. The most commonly used cost function in linear regression is the Mean Squared Error (MSE) function. The MSE function is defined as:

- ✓ The Mean Squared Error measures how close a regression line is to a set of data points. It is a risk function corresponding to the expected value of the squared error loss.

$$\text{MSE} = 1/n * (\text{Summation } (Y_i - \hat{Y}_i)^2)$$

```
def compute_cost(theta_0, theta_1, data):
    n = len(data)
    predictions = theta_0 + theta_1 * data['X'] # y_hat = theta_0 + theta_1 * X
    cost = (1 / (n)) * np.sum((predictions - data['y']) ** 2)
    return cost
```

```
compute_cost(np.random.randn(), np.random.randn(), data)
```

60.34873531065465

### ✓ 4. Gradient Descent Algorithm

```
def gradient_descent(data, learning_rate, n_iterations):
    """Perform gradient descent to minimize the cost function."""
    m = len(data)
    theta_0 = np.random.randn() # Randomly initialize theta_0 (intercept)
    theta_1 = np.random.randn() # Randomly initialize theta_1 (slope)
    cost_history = []

    for iteration in range(n_iterations):
        predictions = theta_0 + theta_1 * data['X']
        d_theta_0 = (1 / m) * np.sum(predictions - data['y']) # Derivative w.r.t theta_0
        d_theta_1 = (1 / m) * np.sum((predictions - data['y']) * data['X']) # Derivative w.r.t theta_1

        # Update rules
        theta_0 -= learning_rate * d_theta_0
        theta_1 -= learning_rate * d_theta_1
```

```
# Compute and store the cost
cost_history.append(compute_cost(theta_0, theta_1, data))

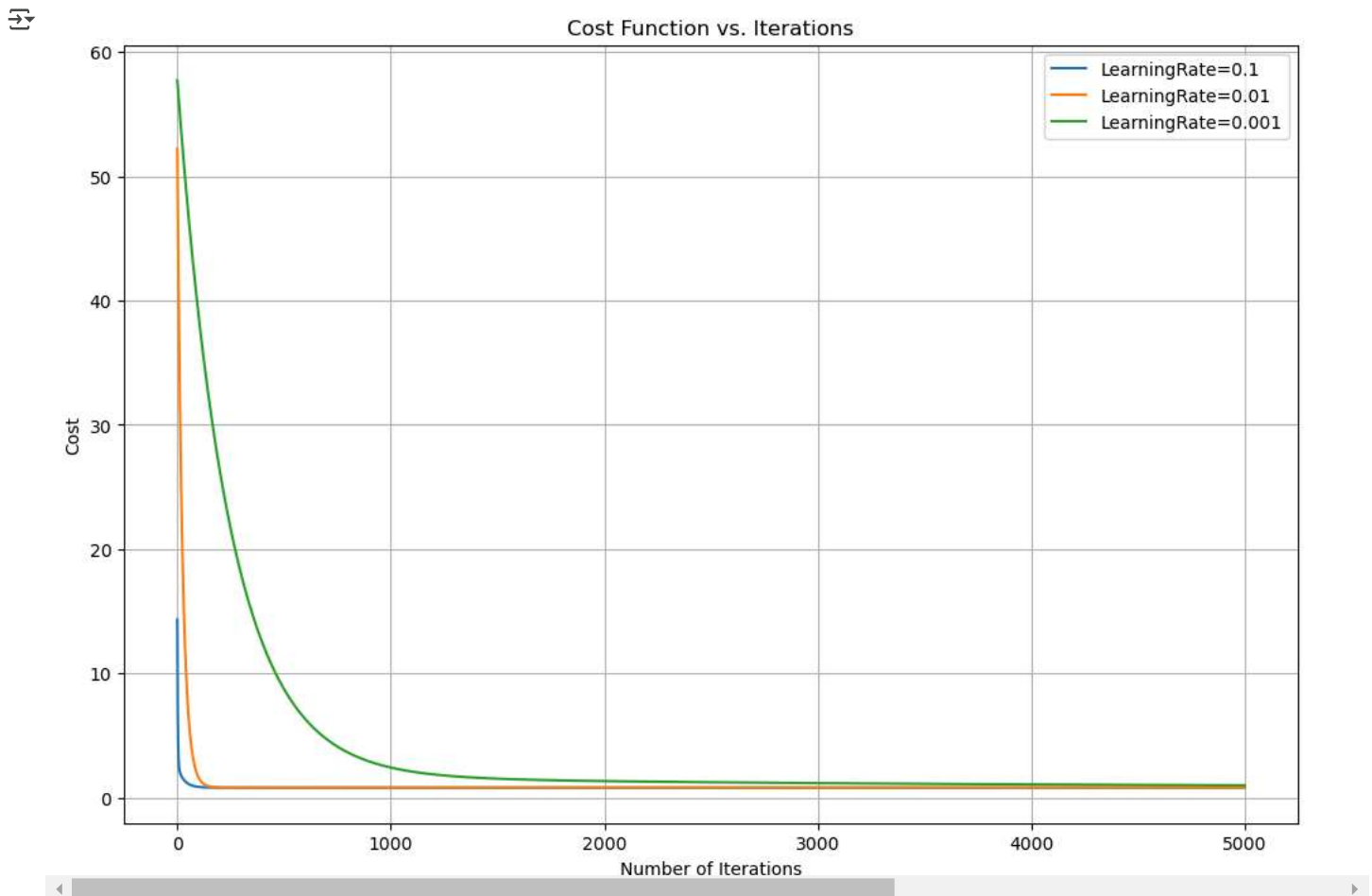
return theta_0, theta_1, cost_history
```

## Parameters for gradient descent

```
n_iterations = 5000
learning_rates = [0.1, 0.01, 0.001]
```

## Train models with different learning rates

```
plt.figure(figsize=(12, 8))
for lr in learning_rates:
    theta_0, theta_1, cost_history = gradient_descent(data, learning_rate=lr, n_iterations=n_iterations)
    plt.plot(range(n_iterations), cost_history, label=f'LearningRate={lr}')
# Plotting the cost function
plt.title('Cost Function vs. Iterations')
plt.xlabel('Number of Iterations')
plt.ylabel('Cost')
plt.legend()
plt.grid()
plt.show()
```



## 5. Model Prediction and Visualization

```
final_theta_0, final_theta_1, _ = gradient_descent(data, learning_rate=0.1, n_iterations=n_iterations)
data['y_prediction'] = final_theta_0 + final_theta_1 * data['X']
```

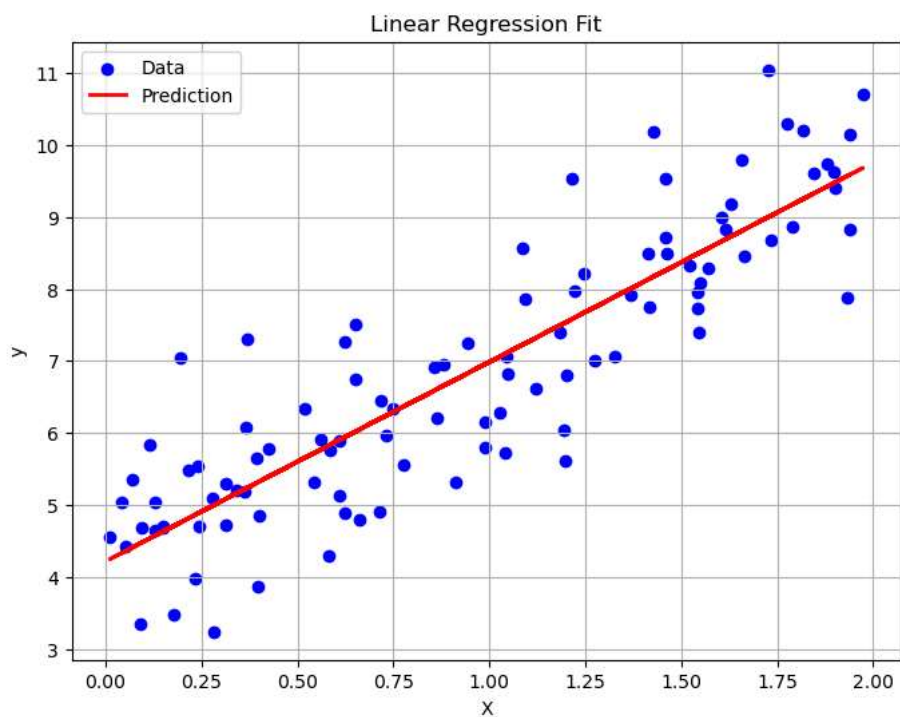
data



	X	y	y_prediction
0	0.749080	6.334288	6.290133
1	1.901429	9.405278	9.482269
2	1.463988	8.483724	8.270509
3	1.197317	5.604382	7.531800
4	0.312037	4.716440	5.079475
...	...	...	...
95	0.987591	6.146963	6.950836
96	1.045466	7.059295	7.111155
97	0.855082	6.906398	6.583770
98	0.050838	4.429206	4.355924
99	0.215783	5.474532	4.812839

100 rows × 3 columns

```
plt.figure(figsize=(8, 6))
plt.scatter(data['X'], data['y'], color='blue', label='Data')
plt.plot(data['X'], data['y_prediction'], color='red', label='Prediction', linewidth=2)
plt.title('Linear Regression Fit')
plt.xlabel('X')
plt.ylabel('y')
plt.legend()
plt.grid()
plt.show()
```



## ▼ final parameters

```
print(f"Final parameters: theta_0 = {final_theta_0}, theta_1 = {final_theta_1}")
```



Final parameters: theta\_0 = 4.215096157546728, theta\_1 = 2.7701133864385015

- Before gradient descent , the cost was :

```
compute_cost(np.random.randn(),np.random.randn(), data)
```

```
↗ 66.94572657178185
```

- After gradient descent , the cost is :

```
compute_cost(4.215096157546728, 2.7701133864385015, data)
```

```
↗ 0.8065845639670532
```

## 6. Model Evaluation

```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
```

```
mae = mean_absolute_error(data['y'],data['y_prediction'])
```

```
mse = mean_squared_error(data['y'],data['y_prediction'])
```

```
rmse = np.sqrt(mse)
```

```
r2 = r2_score(data['y'],data['y_prediction'])
```

```
print(f"Model Evaluation Metrics:\nMAE: {mae:.4f}\nMSE: {mse:.4f}\nRMSE: {rmse:.4f}\nR^2: {r2:.4f}")
```

```
↗ Model Evaluation Metrics:
```

```
MAE: 0.7010
```

```
MSE: 0.8066
```

```
RMSE: 0.8981
```

```
R^2: 0.7693
```

### Model's Evaluation Metrics:

#### Mean Absolute Error (MAE): 0.7010

This is a relatively low error, which indicates that your model is making predictions that are fairly close to the true values.

#### Mean Squared Error (MSE): 0.8066

Since MSE penalizes larger errors more than smaller ones, the value indicates that the model generally avoids making large errors. However, being in squared units, it's harder to interpret directly.

#### Root Mean Squared Error (RMSE): 0.8981

RMSE provides an intuitive sense of how far off the model's predictions are from actual values. A value below 1.0 indicates reasonably good performance, considering the scale of data.

#### R-Squared ( $R^2$ ): 0.7693

This is a fairly good value, suggesting that the model captures the majority of the variation in the data, but there's still around 23% of the variation unexplained, which could be due to noise or factors not included in the model.

## End of Task 2

✓ End of Assignment

Start coding or generate with AI.

Double-click (or enter) to edit

Double-click (or enter) to edit