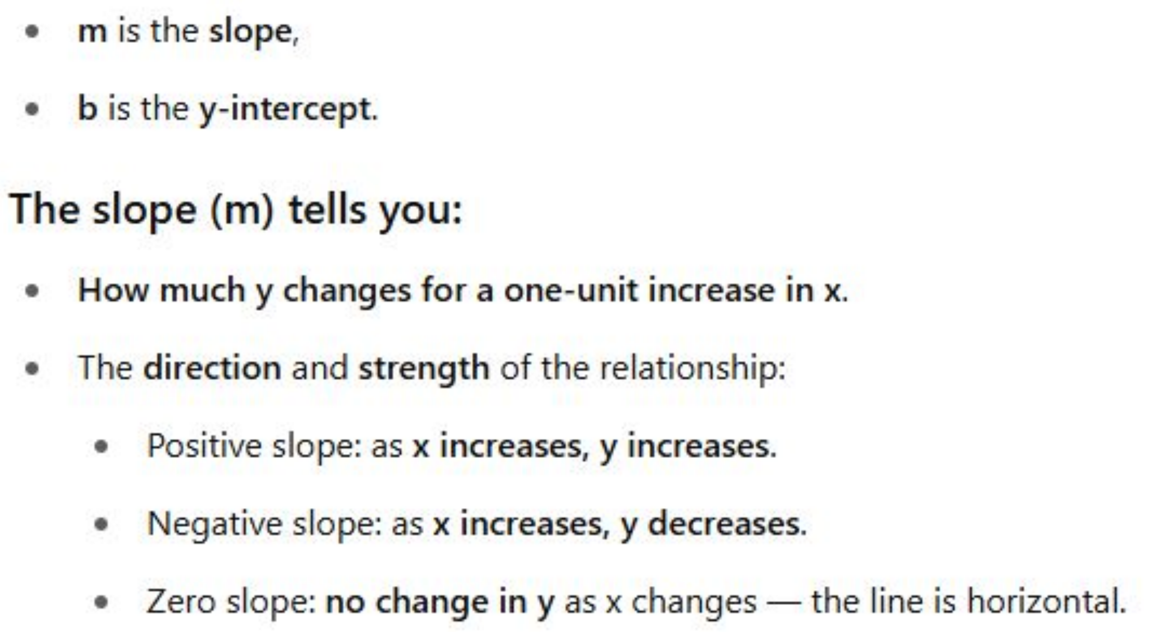
In statistics, the **slope** refers to the rate of change between two variables in a **linear relationship**. It is a key component of the **linear regression equation**, which models the relationship between an independent variable (x) and a dependent variable (y).

$$y=mx+b$$



- **m** is the **slope**,
- **b** is the **y-intercept**.

The slope (m) tells you:

- **How much y changes for a one-unit increase in x.**
- The **direction** and **strength** of the relationship:
  - Positive slope: as **x increases, y increases.**
  - Negative slope: as **x increases, y decreases.**
  - Zero slope: **no change in y** as x changes — the line is horizontal.

If the slope is **2**, it means that for every 1-unit increase in x, **y increases by 2 units.**

In statistical terms (from regression analysis):

$$\text{slope} = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sum(x_i - \bar{x})^2}$$

This formula calculates the **least squares estimate** of the slope in simple linear regression.

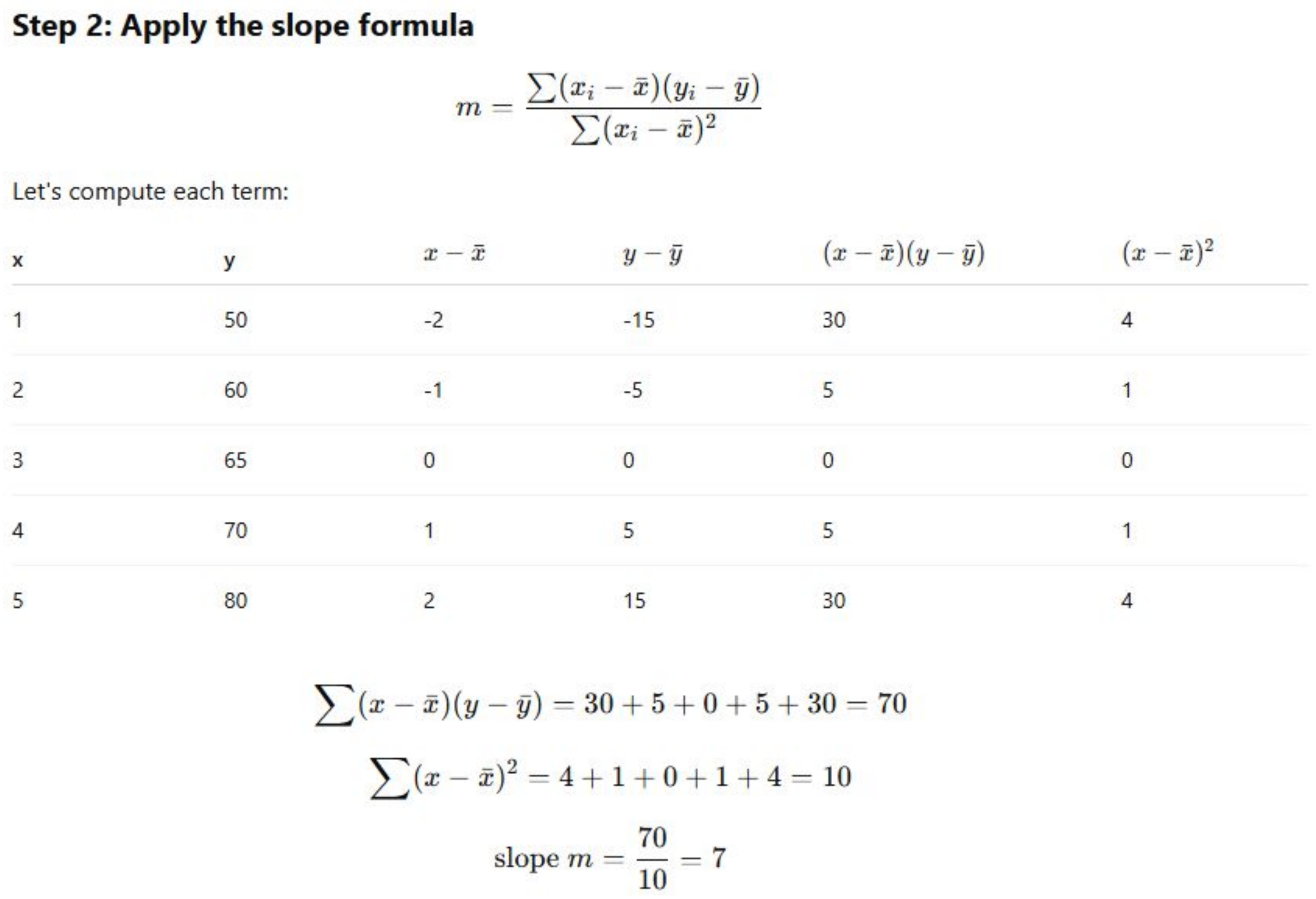
x (Study Hours)	y (Test Score)
1	50
2	60
3	65
4	70
5	80

We want to find the **slope (m)** of the best-fit line (linear regression line) for this data.

Step 1: Find the means

$$\bar{x} = \frac{1+2+3+4+5}{5} = \frac{15}{5} = 3$$

$$\bar{y} = \frac{50+60+65+70+80}{5} = \frac{325}{5} = 65$$



Step 2: Apply the slope formula

$$m = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sum(x_i - \bar{x})^2}$$

Let's compute each term:

x	y	$x - \bar{x}$	$y - \bar{y}$	$(x - \bar{x})(y - \bar{y})$	$(x - \bar{x})^2$
1	50	-2	-15	30	4
2	60	-1	-5	5	1
3	65	0	0	0	0
4	70	1	5	5	1
5	80	2	15	30	4

$$\sum(x - \bar{x})(y - \bar{y}) = 30 + 5 + 0 + 5 + 30 = 70$$

$$\sum(x - \bar{x})^2 = 4 + 1 + 0 + 1 + 4 = 10$$

$$\text{slope } m = \frac{70}{10} = 7$$

for each additional hour of study, the test score increases by 7 points, on average

- **Slope (m) = 7**
- Mean of x: $\bar{x} = 3$
- Mean of y: $\bar{y} = 65$

The linear equation is:

$$y = mx + b$$

To find **b**, use the formula:

$$b = \bar{y} - m\bar{x}$$

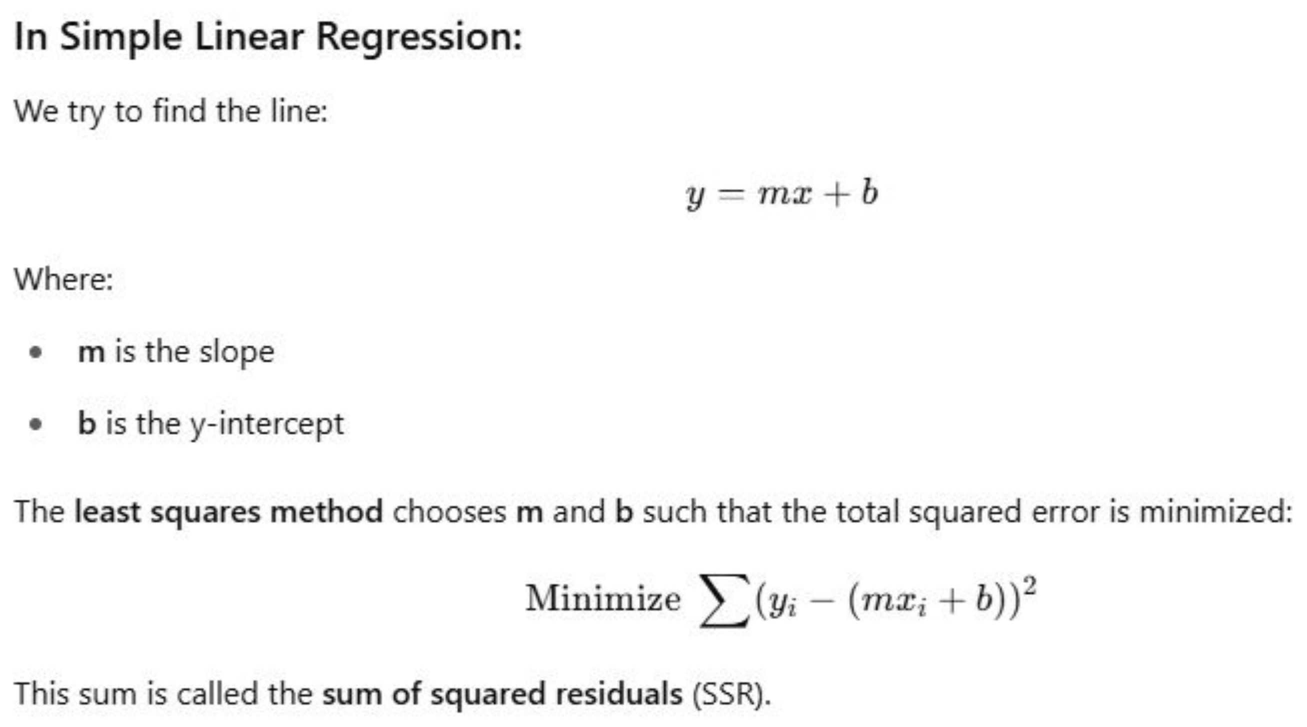
$$b = 65 - 7 \times 3 = 65 - 21 = 44$$

Final Regression Equation:

$$\boxed{y = 7x + 44}$$

What is the **Least Squares Estimate**?

The **least squares estimate** is a method in statistics used to **find the best-fitting line** (or curve) through a set of points by minimizing the **sum of the squared differences** (errors) between the observed values and the values predicted by the model.



In Simple Linear Regression:

We try to find the line:

$$y = mx + b$$

Where:

- **m** is the slope
- **b** is the y-intercept

The **least squares method** chooses **m** and **b** such that the total squared error is minimized:

$$\text{Minimize } \sum (y_i - (mx_i + b))^2$$

This sum is called the **sum of squared residuals** (SSR).

x (Study Hours)	y (Test Score)
1	50
2	60
3	65
4	70
5	80

We want to find the line $y = mx + b$ that minimizes the squared errors using **least squares**.

let's check the **errors** and their **squares**

x	y (Actual)	y (Predicted) = 7x+44	Error = y - ŷ	Error²
1	50	51	-1	1
2	60	58	2	4
3	65	65	0	0
4	70	72	-2	4
5	80	79	1	1

$$\text{Total Squared Error (SSR)} = 1 + 4 + 0 + 4 + 1 = 10$$

This total error (10) is **as small as possible** for any line through the data. That's what the **least squares method** ensures.

Calculate Mean of x and y

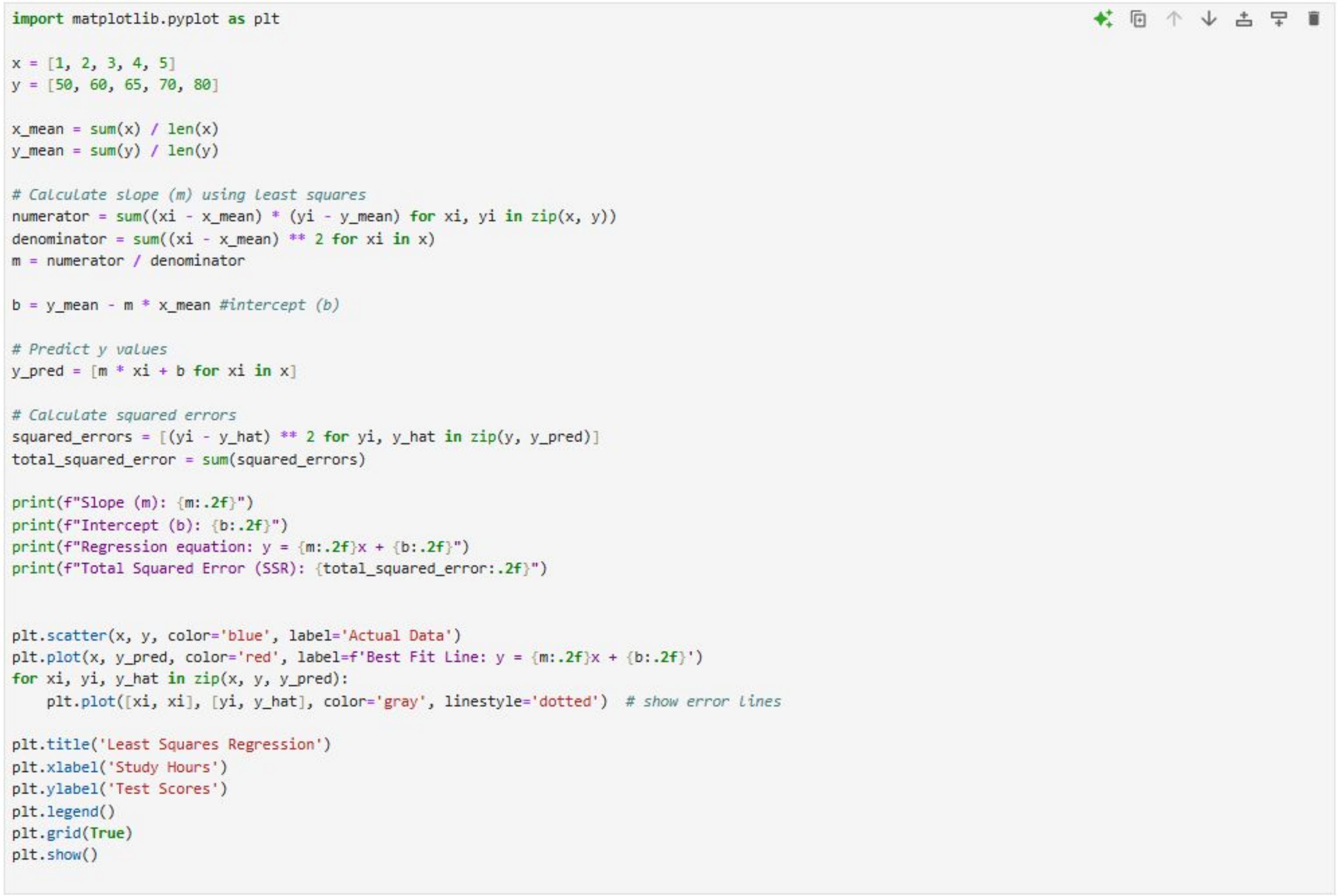
$$\bar{x} = 3, \quad \bar{y} = 65$$

Calculate slope (m) using least squares formula

$$m = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sum(x_i - \bar{x})^2} = \frac{70}{10} = 7$$

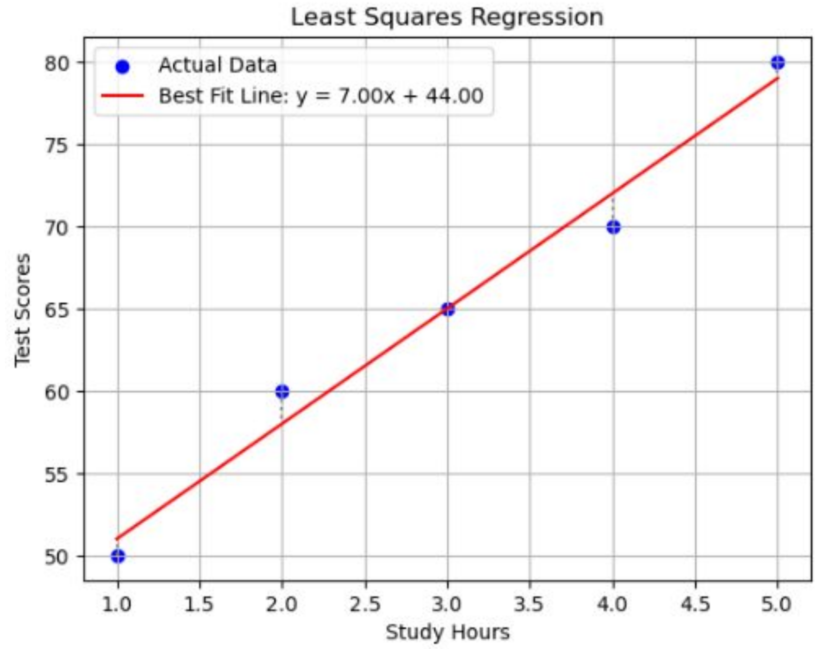
Calculate intercept (b)

$$b = \bar{y} - m\bar{x} = 65 - 7 \cdot 3 = 44$$



```python
import matplotlib.pyplot as plt

x = [1, 2, 3, 4, 5]
y = [50, 60, 65, 70, 80]

x_mean = sum(x) / len(x)
y_mean = sum(y) / len(y)

# Calculate slope (m) using least squares
numerator = sum((xi - x_mean) * (yi - y_mean) for xi, yi in zip(x, y))
denominator = sum((xi - x_mean) ** 2 for xi in x)
m = numerator / denominator

b = y_mean - m * x_mean #intercept (b)

# Predict y values
y_pred = [m * xi + b for xi in x]

# Calculate squared errors
squared_errors = [(yi - y_hat) ** 2 for yi, y_hat in zip(y, y_pred)]
total_squared_error = sum(squared_errors)

print(f"Slope (m): {m:.2f}")
print(f"Intercept (b): {b:.2f}")
print(f"Regression equation: y = {m:.2f}x + {b:.2f}")
print(f"Total Squared Error (SSR): {total_squared_error:.2f}")


plt.scatter(x, y, color='blue', label='Actual Data')
plt.plot(x, y_pred, color='red', label=f'Best Fit Line: y = {m:.2f}x + {b:.2f}')
for xi, yi, y_hat in zip(x, y, y_pred):
    plt.plot([xi, xi], [yi, y_hat], color='gray', linestyle='dotted')  # show error lines

plt.title('Least Squares Regression')
plt.xlabel('Study Hours')
plt.ylabel('Test Scores')
plt.legend()
plt.grid(True)
plt.show()
```

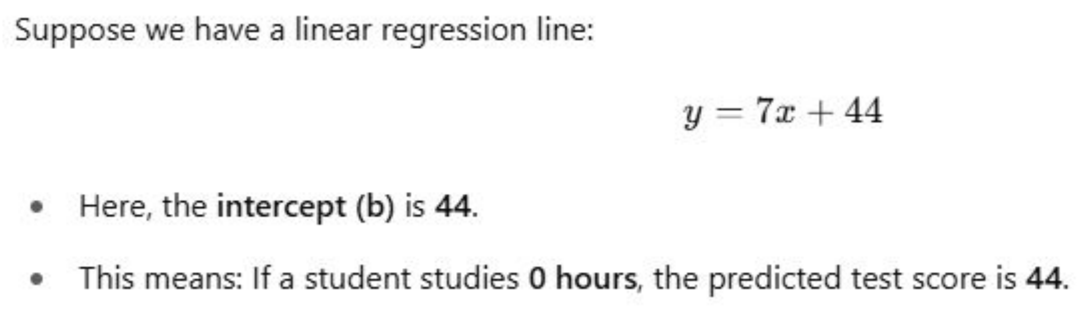
```
Slope (m): 7.00
Intercept (b): 44.00
Regression equation: y = 7.00x + 44.00
Total Squared Error (SSR): 10.00
```



Intercept

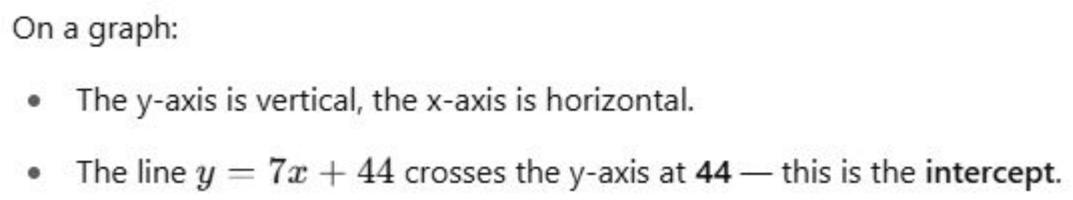
It represents the **starting value** or **baseline** of the dependent variable (**y**) when the independent variable (**x**) is zero.

In many real-world scenarios, it gives a reference point or base level before any influence from x.


Suppose we have a linear regression line:

$$y = 7x + 44$$

- Here, the **intercept (b)** is **44**.
- This means: If a student studies **0 hours**, the predicted test score is **44**.

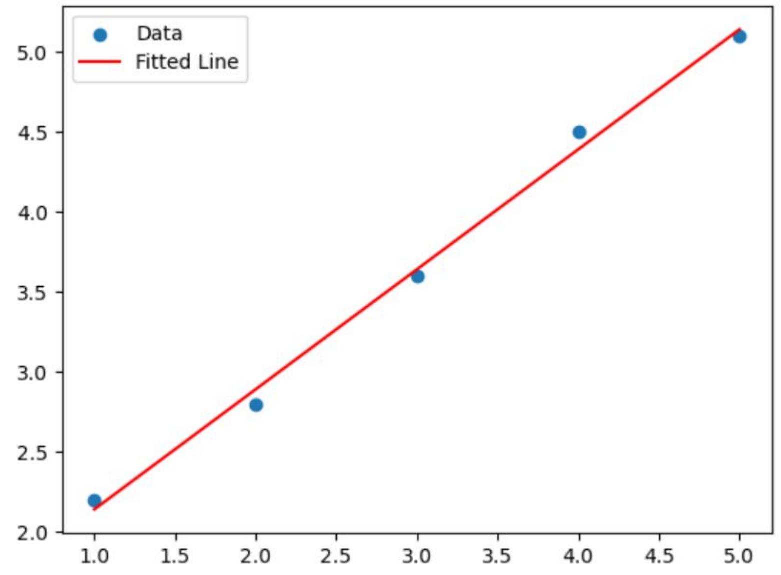

On a graph:

- The y-axis is vertical, the x-axis is horizontal.
- The line $y = 7x + 44$ crosses the y-axis at **44** — this is the **intercept**.

Least Square value Using Numpy, Scipy & SKlearn

```python
import numpy as np
import matplotlib.pyplot as plt

# Sample data
x = np.array([1, 2, 3, 4, 5])
y = np.array([2.2, 2.8, 3.6, 4.5, 5.1])

# Calculate the least squares coefficients
A = np.vstack([x, np.ones(len(x))]).T
m, b = np.linalg.lstsq(A, y, rcond=None)[0]

print(f"Slope (m): {m:.4f}")
print(f"Intercept (b): {b:.4f}")

# Plotting
plt.scatter(x, y, label='Data')
plt.plot(x, m*x + b, 'r', label='Fitted Line')
plt.legend()
plt.show()
```

```
Slope (m): 0.7500
Intercept (b): 1.3900
```



```python
from scipy.stats import linregress

x = [1, 2, 3, 4, 5]
y = [2.2, 2.8, 3.6, 4.5, 5.1]

result = linregress(x, y)

print(f"Slope: {result.slope}")
print(f"Intercept: {result.intercept}")
print(f"R-squared: {result.rvalue**2}")
```

```
Slope: 0.75
Intercept: 1.3899999999999997
R-squared: 0.9952229299363061
```

- The least squares method minimizes:

$$\sum_{i=1}^{n}(y_i-(mx_i+b))^2$$

- It's the foundation for linear regression and other curve-fitting techniques.

```python
from sklearn.linear_model import LinearRegression
import numpy as np

x = np.array([1, 2, 3, 4, 5]).reshape(-1, 1)  # 2D array
y = np.array([2.2, 2.8, 3.6, 4.5, 5.1])

model = LinearRegression()
model.fit(x, y)

print(f"Slope: {model.coef_[0]}")
print(f"Intercept: {model.intercept_}")
```

```
Slope: 0.7500000000000002
Intercept: 1.3899999999999988
```

To **manually calculate the least squares coefficients** for a linear regression model (to fit a model):

$$y = X\beta + \varepsilon$$

Where:

- y is the target/output vector
- X is the design/input matrix
- β is the vector of coefficients (what we want to find)
- ε is the error term

Suppose you have the following data:

x	y
1	2
2	3
3	5

We want to fit a line:

$$y = \beta_0 + \beta_1 x$$

So, the **design matrix** X (including intercept column of ones):

$$X = \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \end{bmatrix}, \quad y = \begin{bmatrix} 2 \\ 3 \\ 5 \end{bmatrix}$$

We include a column of ones in the design matrix X to account for the intercept term β0 in a linear regression model.

Including the intercept (via a column of ones in X allows the regression model to adjust for vertical offset, improving accuracy unless you *know for sure* the line should go through the origin.

Without 1s column	With 1s column
Only fits line through origin (no intercept)	Fits a line with intercept
Model: $y = \beta_1 x$	Model: $y = \beta_0 + \beta_1 x$

Least Squares Formula:

The **closed-form solution** for the least squares coefficients is:

$$\hat{\beta} = (X^T X)^{-1} X^T y$$

Step-by-step:

1. Compute $X^T X$:

$$X^T X = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \end{bmatrix} = \begin{bmatrix} 3 & 6 \\ 6 & 14 \end{bmatrix}$$

2. Compute $X^T y$:

$$X^T y = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} 2 \\ 3 \\ 5 \end{bmatrix} = \begin{bmatrix} 2+3+5 \\ 2 \times 1 + 3 \times 2 + 5 \times 3 \end{bmatrix} = \begin{bmatrix} 10 \\ 23 \end{bmatrix}$$

3. Compute $(X^T X)^{-1}$:

$$(X^T X)^{-1} = \begin{bmatrix} 3 & 6 \\ 6 & 14 \end{bmatrix}^{-1}$$

4. Now compute $\hat{\beta} = (X^T X)^{-1} X^T y$:

$$\hat{\beta} = \frac{1}{6} \begin{bmatrix} 14 & -6 \\ -6 & 3 \end{bmatrix} \begin{bmatrix} 10 \\ 23 \end{bmatrix} = \frac{1}{6} \begin{bmatrix} 14 \times 10 + (-6) \times 23 \\ -6 \times 10 + 3 \times 23 \end{bmatrix} = \frac{1}{6} \begin{bmatrix} 140 - 138 \\ -60 + 69 \end{bmatrix} = \frac{1}{6} \begin{bmatrix} 2 \\ 9 \end{bmatrix}$$

$$\hat{\beta} = \begin{bmatrix} \frac{2}{6} \\ \frac{9}{6} \end{bmatrix} = \begin{bmatrix} 0.333\ldots \\ 1.5 \end{bmatrix}$$

Final Answer:

$$\beta_0 = 0.\overline{3}, \quad \beta_1 = 1.5$$

So, the best-fit line is:

$$\boxed{y = 0.33 + 1.5x}$$

The general form of a **simple linear regression** is:

$$y = \beta_0 + \beta_1 x + \varepsilon$$

But in **matrix form**, this becomes:

$$y = X\beta + \varepsilon$$

how excluding the intercept changes the fit

This means we are fitting:

$$y = \beta_1 x \quad \text{(no intercept)}$$

So the design matrix X becomes:

$$X = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}, \quad y = \begin{bmatrix} 2 \\ 3 \\ 5 \end{bmatrix}$$

Compute least square

$$\hat{\beta}_1 = \left(X^T X\right)^{-1} X^T y$$

Step by step:

1. $X^T X = 1^2 + 2^2 + 3^2 = 14$
2. $X^T y = 1 \times 2 + 2 \times 3 + 3 \times 5 = 23$

So:

$$\hat{\beta}_1 = \frac{23}{14} \approx 1.643$$

Thus, the best-fit model **without intercept** is:

$$\boxed{y = 1.643x}$$

Model	Equation	Prediction at x=0	RMSE (error)
With Intercept	$y = 0.33 + 1.5x$	0.33	Lower error (better fit)
Without Intercept	$y = 1.643x$	0	Higher error (worse fit unless line passes through origin)

Visual Insight

With intercept: the line can **shift up/down** to fit data better.

Without intercept: the line is **forced through the origin**, which may **distort the fit** if the real data doesn't behave that way.

When to exclude the intercept?

Only when you have a **solid theoretical reason** — for example:

- Measuring physical laws like $F = ma$
- You know the response must be 0 when input is 0

Otherwise, **always include the intercept** (i.e., include a column of 1s in X).

Standard deviation

Standard deviation in statistics is a measure of how **spread out or dispersed** the values in a data set are from the **mean (average)** of that data set. It tells you how much the individual data points typically **deviate from the average value**.

A **low standard deviation** means the data points are **close to the mean**.

A **high standard deviation** means the data points are **spread out over a wider range** of values.

For a **population**:

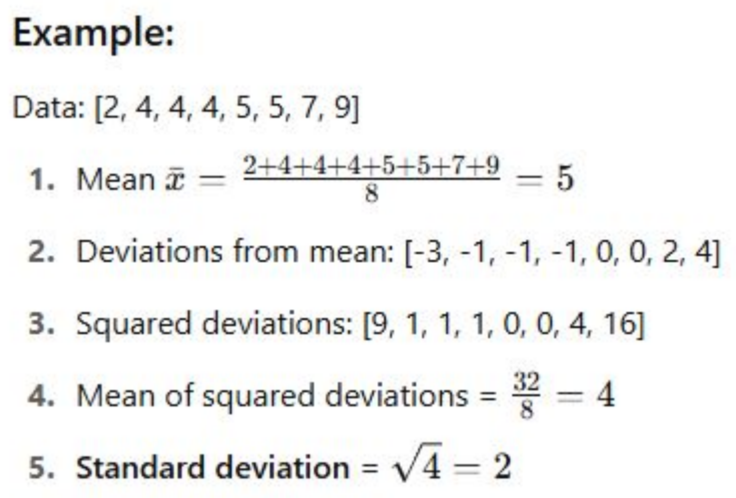
$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2}$$

- σ = population standard deviation
- N = number of data points
- x_i = each individual data point
- μ = population mean

For a **sample**:

$$s = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

- s = sample standard deviation
- n = number of sample data points
- x_i = each data point
- $\bar{x}$ = sample mean

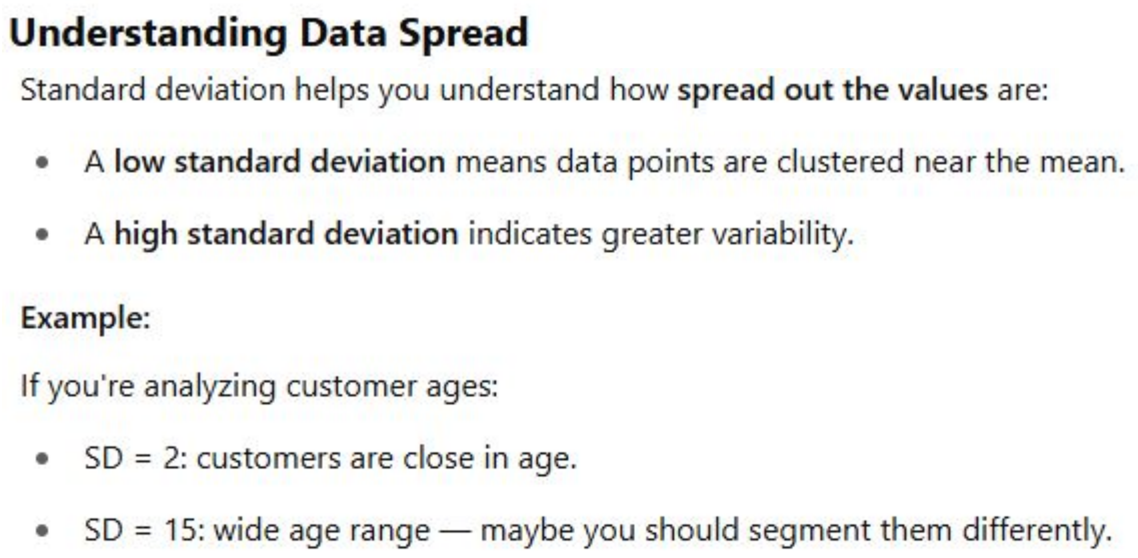


Example:

Data: [2, 4, 4, 4, 5, 5, 7, 9]

1. Mean $\bar{x} = \frac{2+4+4+4+5+5+7+9}{8} = 5$
2. Deviations from mean: [-3, -1, -1, -1, 0, 0, 2, 4]
3. Squared deviations: [9, 1, 1, 1, 0, 0, 4, 16]
4. Mean of squared deviations = $\frac{32}{8} = 4$
5. **Standard deviation** = $\sqrt{4} = 2$

Standard deviation is widely used in **data science** and **machine learning** for many purposes, especially for understanding data distributions, detecting outliers, and normalizing data. Here's how it's applied in practice:



Understanding Data Spread

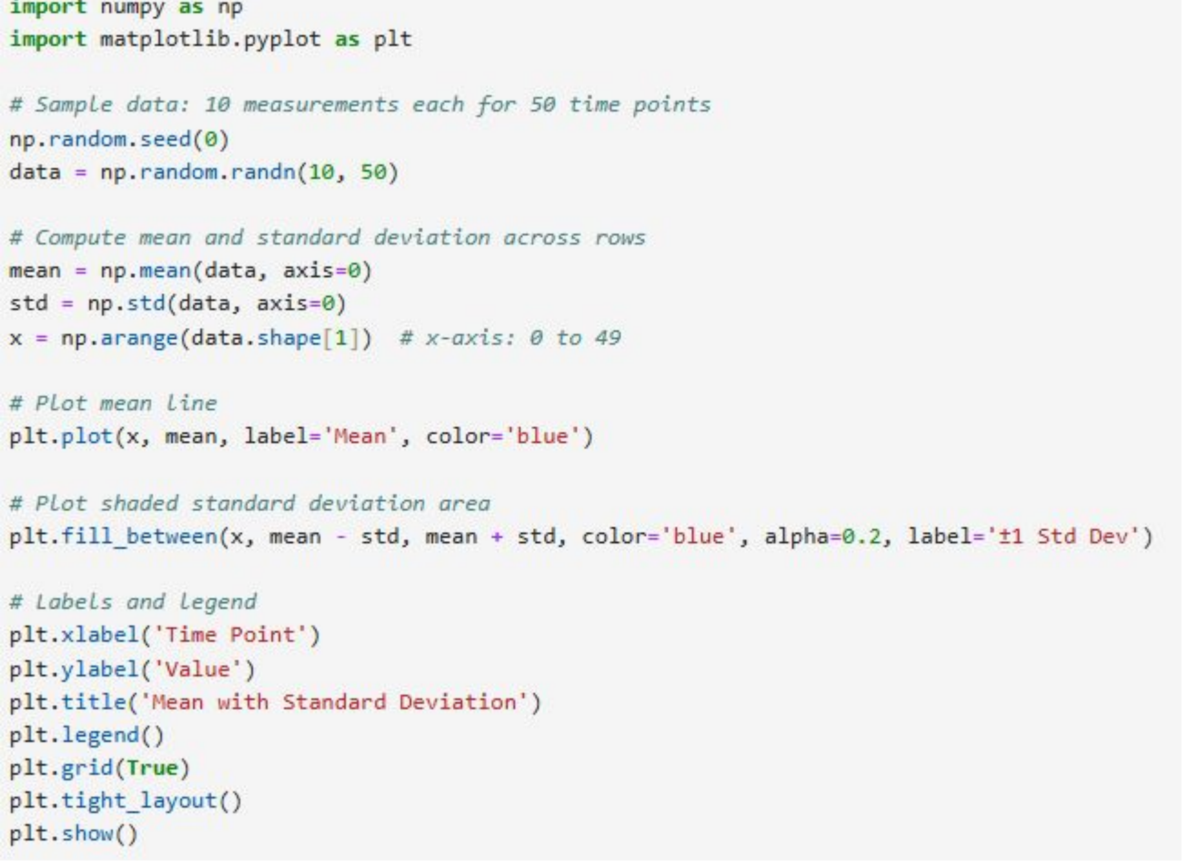
Standard deviation helps you understand how **spread out the values** are:

- A **low standard deviation** means data points are clustered near the mean.
- A **high standard deviation** indicates greater variability.

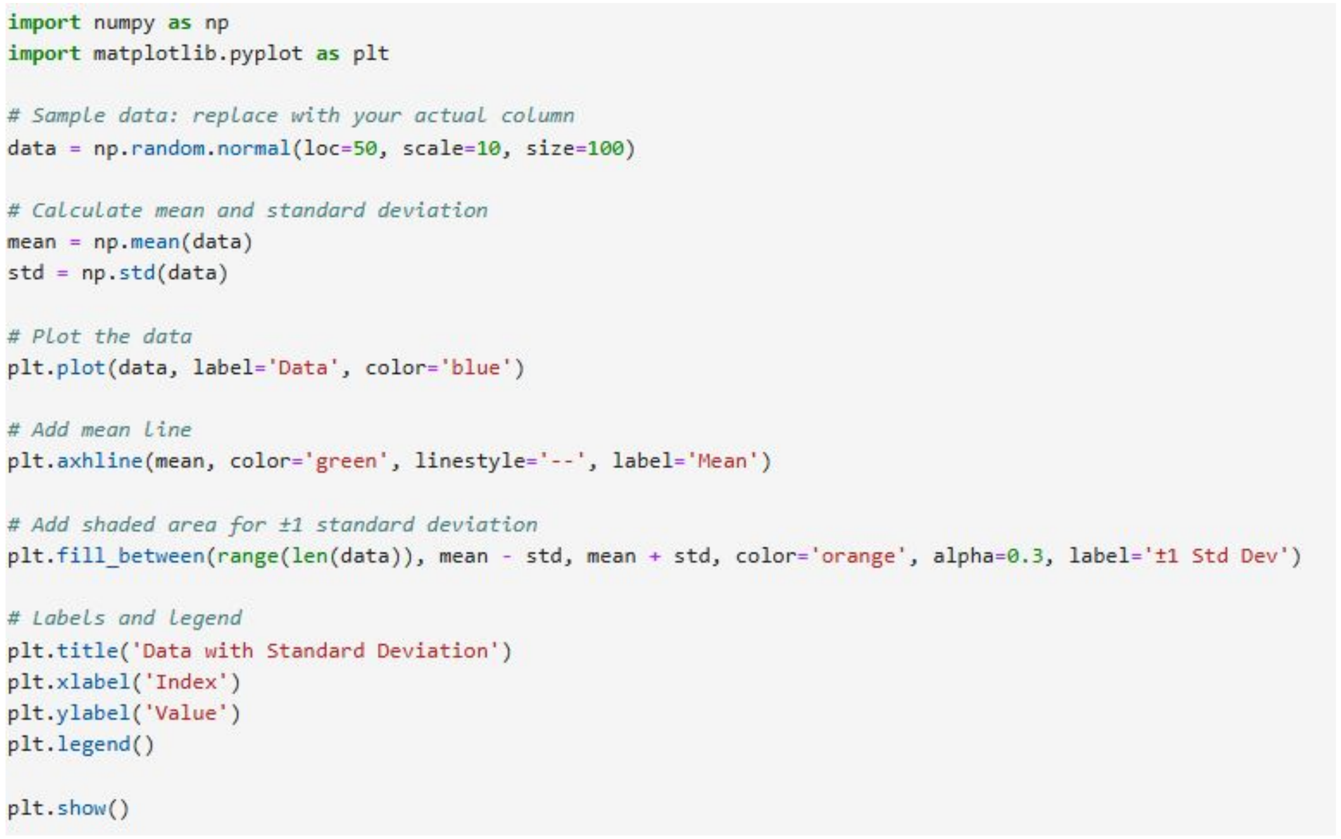
Example:

If you're analyzing customer ages:

- SD = 2: customers are close in age.
- SD = 15: wide age range — maybe you should segment them differently.



```python
import numpy as np
import matplotlib.pyplot as plt

# Sample data: 10 measurements each for 50 time points
np.random.seed(0)
data = np.random.randn(10, 50)

# Compute mean and standard deviation across rows
mean = np.mean(data, axis=0)
std = np.std(data, axis=0)
x = np.arange(data.shape[1])  # x-axis: 0 to 49

# Plot mean line
plt.plot(x, mean, label='Mean', color='blue')

# Plot shaded standard deviation area
plt.fill_between(x, mean - std, mean + std, color='blue', alpha=0.2, label='±1 Std Dev')

# Labels and legend
plt.xlabel('Time Point')
plt.ylabel('Value')
plt.title('Mean with Standard Deviation')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```



```python
import numpy as np
import matplotlib.pyplot as plt

# Sample data: replace with your actual column
data = np.random.normal(loc=50, scale=10, size=100)

# Calculate mean and standard deviation
mean = np.mean(data)
std = np.std(data)

# Plot the data
plt.plot(data, label='Data', color='blue')

# Add mean line
plt.axhline(mean, color='green', linestyle='--', label='Mean')

# Add shaded area for ±1 standard deviation
plt.fill_between(range(len(data)), mean - std, mean + std, color='orange', alpha=0.3, label='±1 Std Dev')

# Labels and legend
plt.title('Data with Standard Deviation')
plt.xlabel('Index')
plt.ylabel('Value')
plt.legend()

plt.show()
```

Coefficient of Variation

The coefficient of variation is used to get an idea of how large the standard deviation is.

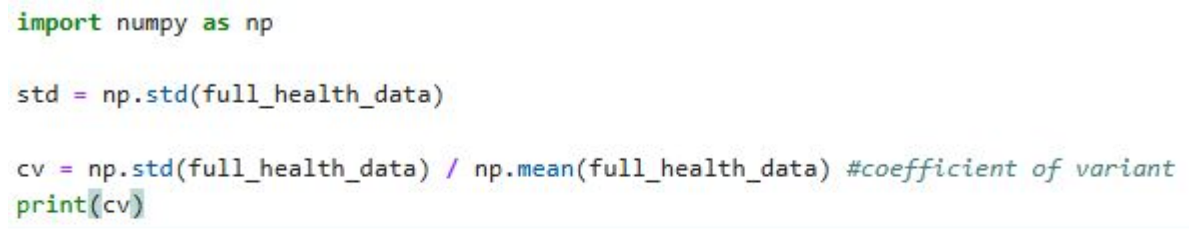
Mathematically, the coefficient of variation is defined as:

Coefficient of Variation = Standard Deviation / Mean

$$\mathrm{CV} = \frac{\sigma}{\mu} \times 100$$

Where:

- σ = standard deviation
- μ = mean
- Result is usually expressed as a **percentage**



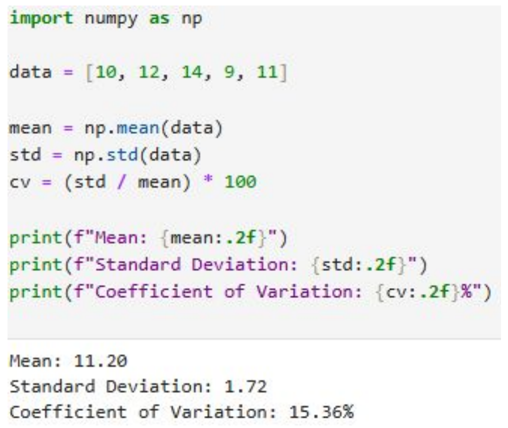
```python
import numpy as np

std = np.std(full_health_data)

cv = np.std(full_health_data) / np.mean(full_health_data) #coefficient of variant
print(cv)
```

The **Coefficient of Variation (CV)** is a **statistical measure** that shows the **relative variability** of data — it tells you how much variation exists **in relation to the mean** of the data.

Use Cases of Coefficient of Variation

Comparing variability between datasets	=> Especially useful when the means of datasets are different.
Assessing risk in finance	=> In investing, a higher CV means higher risk per unit of return.
Scientific measurements	=> Helps assess measurement consistency — lower CV = more precise.
Biological and medical studies	=> Compares variability across experiments or groups.
Quality control	=> Tracks product consistency in manufacturing processes.

standard deviation is **absolute** — it doesn't tell you how big or small the variation is **relative to the mean**.



```
import numpy as np

data = [10, 12, 14, 9, 11]

mean = np.mean(data)
std = np.std(data)
cv = (std / mean) * 100

print(f"Mean: {mean:.2f}")
print(f"Standard Deviation: {std:.2f}")
print(f"Coefficient of Variation: {cv:.2f}%")
```

```
Mean: 11.20
Standard Deviation: 1.72
Coefficient of Variation: 15.36%
```

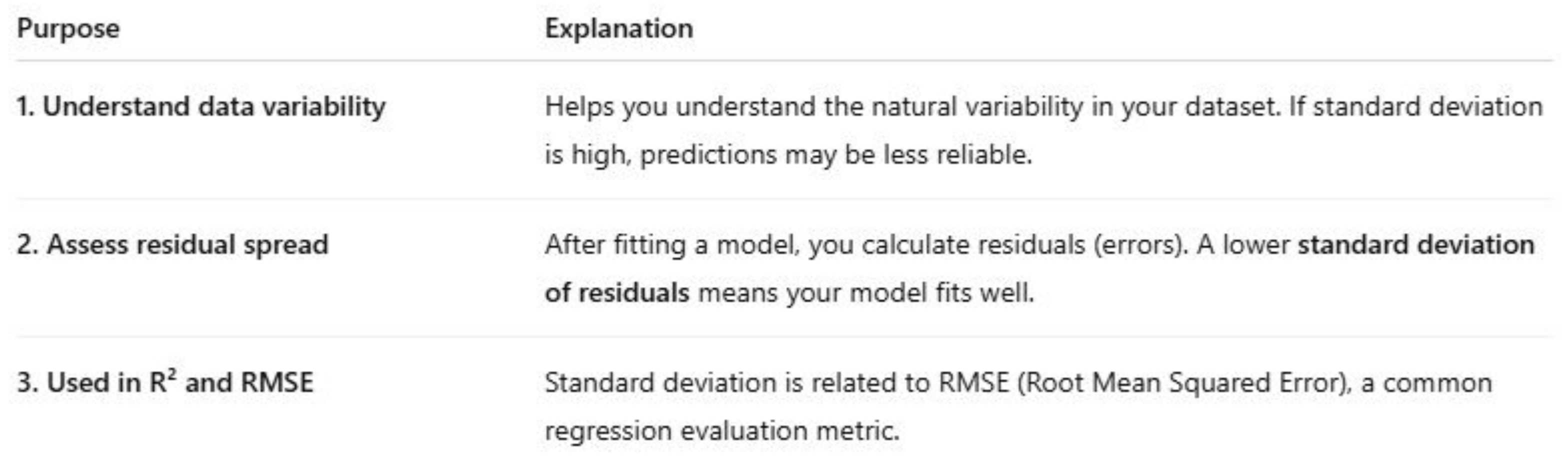
Dataset A: mean = 100, std = 10 → CV = 10%

Dataset B: mean = 5, std = 2 → CV = 40%

Even though Dataset B has a smaller std, it **varies more relative to its mean**.

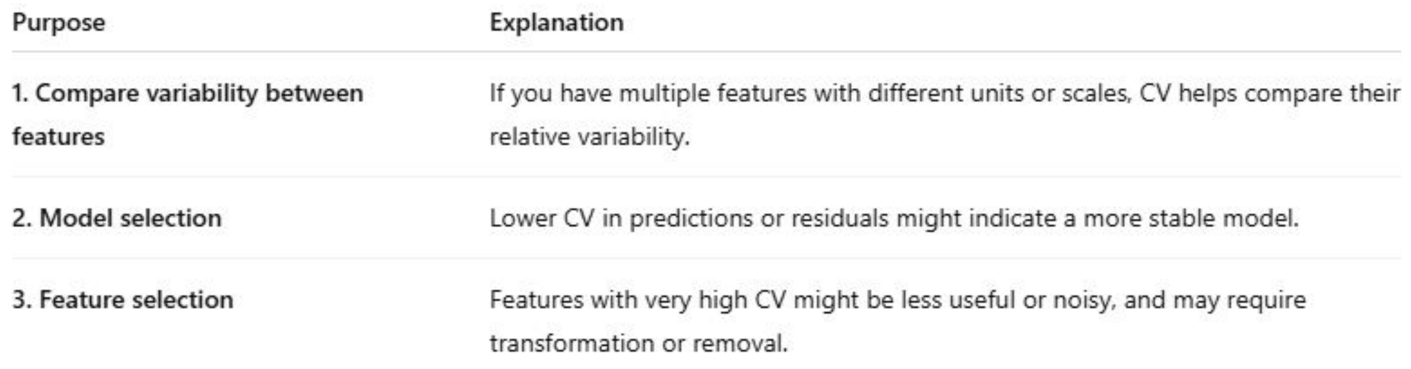
In **linear regression**, both **standard deviation** and **coefficient of variation (CV)** are important statistical tools that help in **understanding the spread, reliability, and relative variability** of data. Let’s break down their roles:

Standard Deviation (SD) in Linear Regression



Purpose	Explanation
1. Understand data variability	Helps you understand the natural variability in your dataset. If standard deviation is high, predictions may be less reliable.
2. Assess residual spread	After fitting a model, you calculate residuals (errors). A lower **standard deviation of residuals** means your model fits well.
3. Used in R^2 and RMSE	Standard deviation is related to RMSE (Root Mean Squared Error), a common regression evaluation metric.

Coefficient of Variation (CV) in Linear Regression



Purpose	Explanation
1. Compare variability between features	If you have multiple features with different units or scales, CV helps compare their relative variability.
2. Model selection	Lower CV in predictions or residuals might indicate a more stable model.
3. Feature selection	Features with very high CV might be less useful or noisy, and may require transformation or removal.

Variance

- Variance is another number that indicates how spread out the values are.
- In fact, if you take the square root of the variance, you get the standard deviation.
- Or the other way around, if you multiply the standard deviation by itself, you get the variance!
- **Variance** is a **fundamental statistical concept** used to measure **how spread out a set of data points is** from the **mean**.
- It plays a critical role in data analysis, modeling, and decision-making.

$$\text{Variance}(\sigma^2) = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2$$

Where:

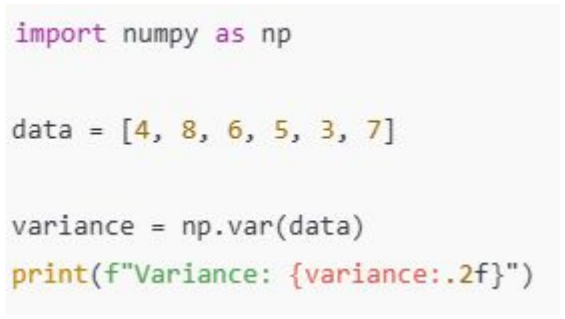
- x_i: each data point
- μ: mean of the data
- n: number of data points

Key Uses of Variance

Measure of Spread	=> Variance quantifies how much the data points deviate from the mean.
Descriptive Statistics	=> Used to summarize data and understand distribution characteristics.
Comparing Datasets	=> Helps compare the variability between two or more datasets.
Machine Learning & Regression	=> Used to calculate error and model performance metrics like MSE (Mean Squared Error).
Probability & Distribution Analysis	=> Essential for understanding distributions (e.g., normal, binomial).
Quality Control	=> Identifies consistency or inconsistency in process or product performance.
Finance and Risk	=> Used to measure the volatility (risk) of investment returns. Higher variance = more uncertainty.

Variance gives a squared unit (e.g., cm² if you're measuring in cm).

Standard deviation is the **square root of variance** — more intuitive and in the same unit as the data.



```
import numpy as np

data = [4, 8, 6, 5, 3, 7]

variance = np.var(data)
print(f"Variance: {variance:.2f}")
```

Stock A has a return variance of 2.5%

Stock B has a return variance of 8.0%

Stock B is **more volatile** and carries **more risk** than Stock A.

Variance Calculation Method (Average_Pulse)

- Step 1: Find the Mean
- Step 2: For Each Value - Find the Difference From the Mean
- Step 3: For Each Difference - Find the Square Value (We must square the values to get the total spread)
- Step 4: The Variance is the Average Number of These Squared Values

Correlation

- Correlation measures the relationship between two variables.
- It has a purpose to predict a value, by converting input (x) to output (f(x)).
- We can also say that a function uses the relationship between two variables for prediction.

Correlation Coefficient

- The correlation coefficient measures the relationship between two variables.
- The correlation coefficient can never be less than -1 or higher than 1.
- 1 = there is a perfect linear relationship between the variables (like Average_Pulse against Calorie_Burnage)
- 0 = there is no linear relationship between the variables
- -1 = there is a perfect negative linear relationship between the variables (e.g. Less hours worked, leads to higher calorie burnage during a training session)

$$r = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum(x_i - \bar{x})^2 \cdot \sum(y_i - \bar{y})^2}}$$

Where:

- x_i, y_i: data values
- $\bar{x}, \bar{y}$: means of x and y

The **correlation coefficient (r)** quantifies how **strongly two variables are related** — and in which **direction** (positive or negative).

What is R^2 value (R-squared)?

R^2 (R-squared) is a statistical measure in regression that indicates how well the independent variables explain the variability of the dependent variable.

Definition:

R^2 = Explained Variation / Total Variation

- **Range:** 0 to 1
  - **$R^2 = 1$** → Perfect fit (100% of the variance is explained by the model)
  - **$R^2 = 0$** → No explanatory power (model explains none of the variance)

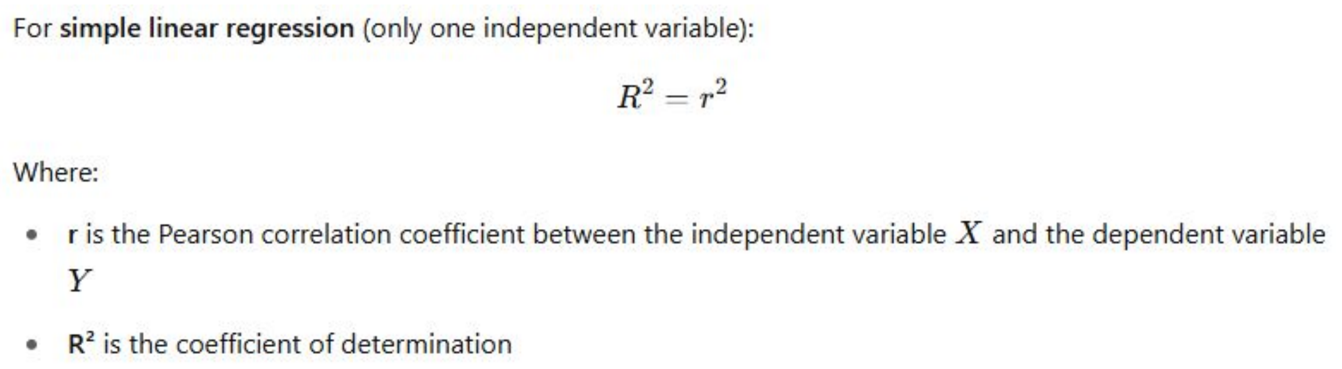
What is the **Correlation Coefficient (r)?**

The **correlation coefficient** (usually **r**) measures the **strength and direction** of a **linear relationship** between two variables.

Range: -1 to +1

- **+1**: perfect positive correlation
- **-1**: perfect negative correlation
- **0**: no linear correlation

Relationship between **R^2** and **r**



For **simple linear regression** (only one independent variable):

$$R^2 = r^2$$

Where:

- **r** is the Pearson correlation coefficient between the independent variable X and the dependent variable Y
- **R^2** is the coefficient of determination

If **r = 0.8**, then:

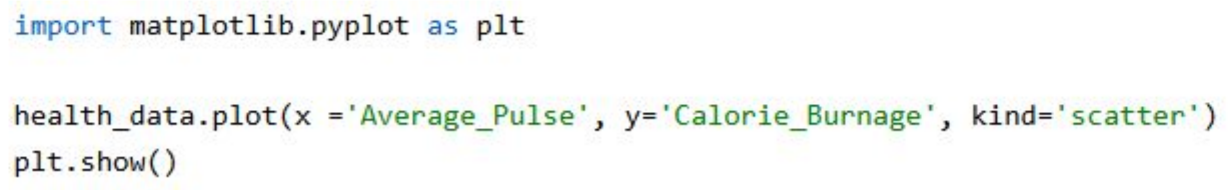
$$R^2 = (0.8)^2 = 0.64$$

Which means **64%** of the variability in Y is explained by X.

Notes

- The equation $R^2 = r^2$ **only holds in simple linear regression.**
- In **multiple linear regression** (more than one predictor), the R^2 cannot be directly derived from a correlation coefficient.
- R^2 tells you **how well the model explains the variance**, while **r** tells you the **direction and strength** of a linear relationship.

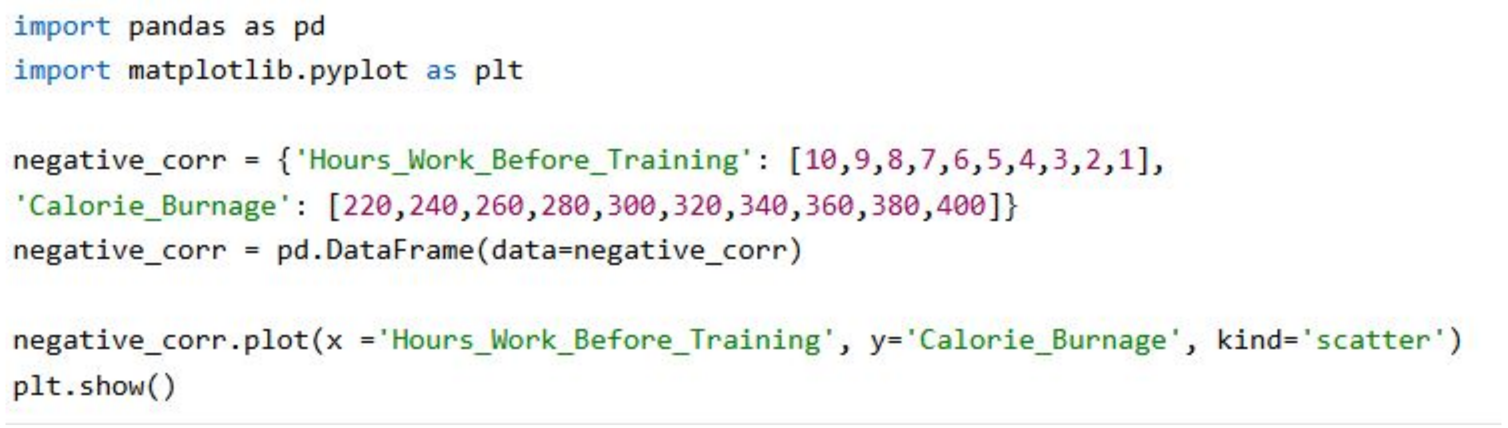
Perfect Linear Relationship (Correlation Coefficient = 1)



```
import matplotlib.pyplot as plt

health_data.plot(x ='Average_Pulse', y='Calorie_Burnage', kind='scatter')
plt.show()
```

perfect linear relationship between Average_Pulse and Calorie_Burnage.

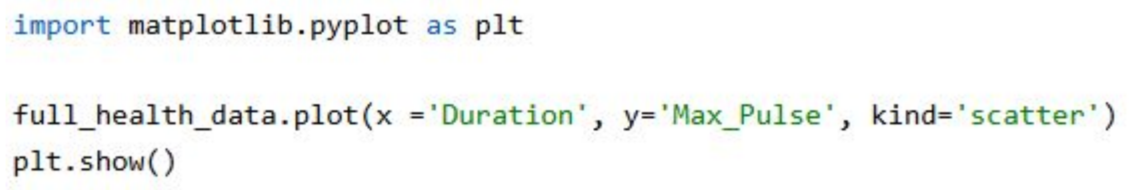
Perfect Negative Linear Relationship (Correlation Coefficient = -1)



```
import pandas as pd
import matplotlib.pyplot as plt

negative_corr = {'Hours_Work_Before_Training': [10,9,8,7,6,5,4,3,2,1],
'Calorie_Burnage': [220,240,260,280,300,320,340,360,380,400]}
negative_corr = pd.DataFrame(data=negative_corr)

negative_corr.plot(x ='Hours_Work_Before_Training', y='Calorie_Burnage', kind='scatter')
plt.show()
```

correlation coefficient here is -1

No Linear Relationship (Correlation coefficient = 0)

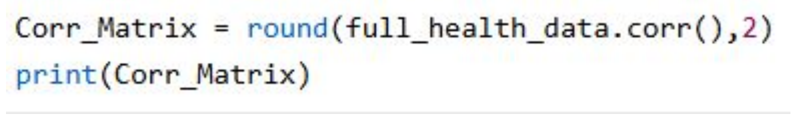


```
import matplotlib.pyplot as plt

full_health_data.plot(x ='Duration', y='Max_Pulse', kind='scatter')
plt.show()
```

The correlation coefficient here is 0

Correlation Matrix

- A matrix is an array of numbers arranged in rows and columns.
- A correlation matrix is simply a table showing the correlation coefficients between variables.



```
Corr_Matrix = round(full_health_data.corr(),2)
print(Corr_Matrix)
```

•We observe that Duration and Calorie_Burnage are closely related, with a correlation coefficient of 0.89. This makes sense as the longer we train,

the more calories we burn

•We observe that there is almost no linear relationships between Average_Pulse and Calorie_Burnage (correlation coefficient of 0.02)

•Can we conclude that Average_Pulse does not affect Calorie_Burnage? No. We will come back to answer this question later!

Correlation vs. Causality

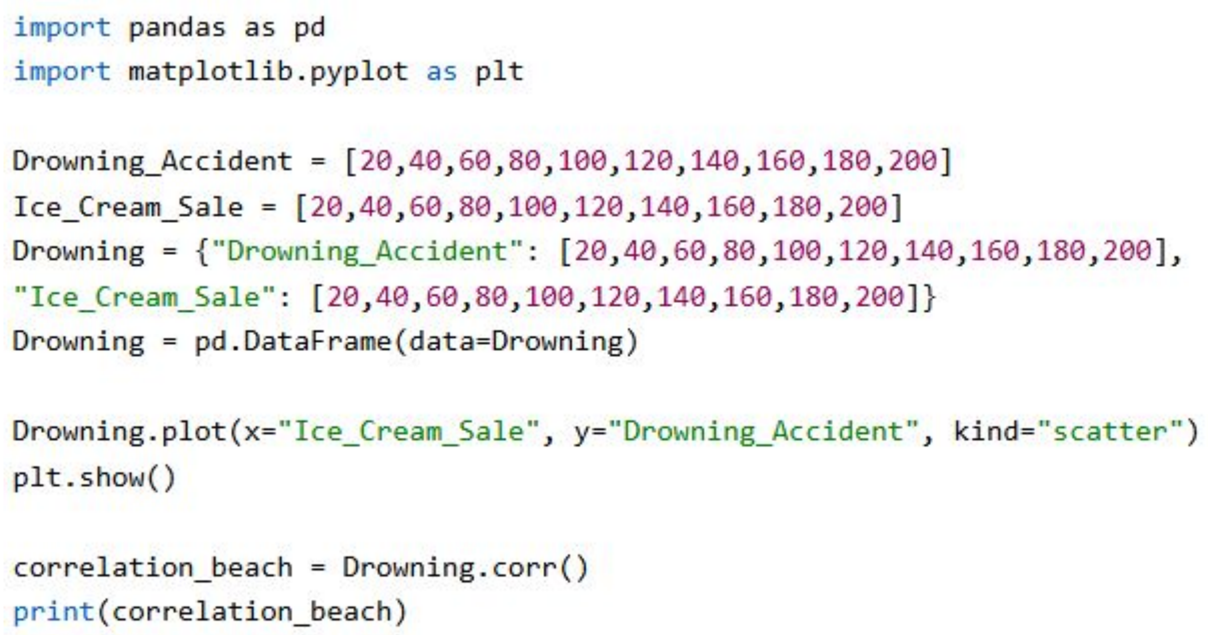
- Correlation measures the numerical relationship between two variables.
- A high correlation coefficient (close to 1), does not mean that we can for sure conclude an actual relationship between two variables.

A classic example:

- During the summer, the sale of ice cream at a beach increases
- Simultaneously, drowning accidents also increase as well

Does this mean that increase of ice cream sale is a direct cause of increased drowning accidents?

The Beach Example in Python



```python
import pandas as pd
import matplotlib.pyplot as plt

Drowning_Accident = [20,40,60,80,100,120,140,160,180,200]
Ice_Cream_Sale = [20,40,60,80,100,120,140,160,180,200]
Drowning = {"Drowning_Accident": [20,40,60,80,100,120,140,160,180,200],
"Ice_Cream_Sale": [20,40,60,80,100,120,140,160,180,200]}
Drowning = pd.DataFrame(data=Drowning)

Drowning.plot(x="Ice_Cream_Sale", y="Drowning_Accident", kind="scatter")
plt.show()

correlation_beach = Drowning.corr()
print(correlation_beach)
```

Correlation vs Causality – The Beach Example

In other words: can we use ice cream sale to predict drowning accidents?

The answer is - Probably not.

It is likely that these two variables are accidentally correlating with each other.

What causes drowning then?

- Unskilled swimmers
- Waves
- Cramp
- Seizure disorders
- Lack of supervision
- Alcohol (mis)use
- etc.

Let us reverse the argument:

Does a low correlation coefficient (close to zero) mean that change in x does not affect y?

Back to the question:

- Can we conclude that Average_Pulse does not affect Calorie_Burnage because of a low correlation coefficient?

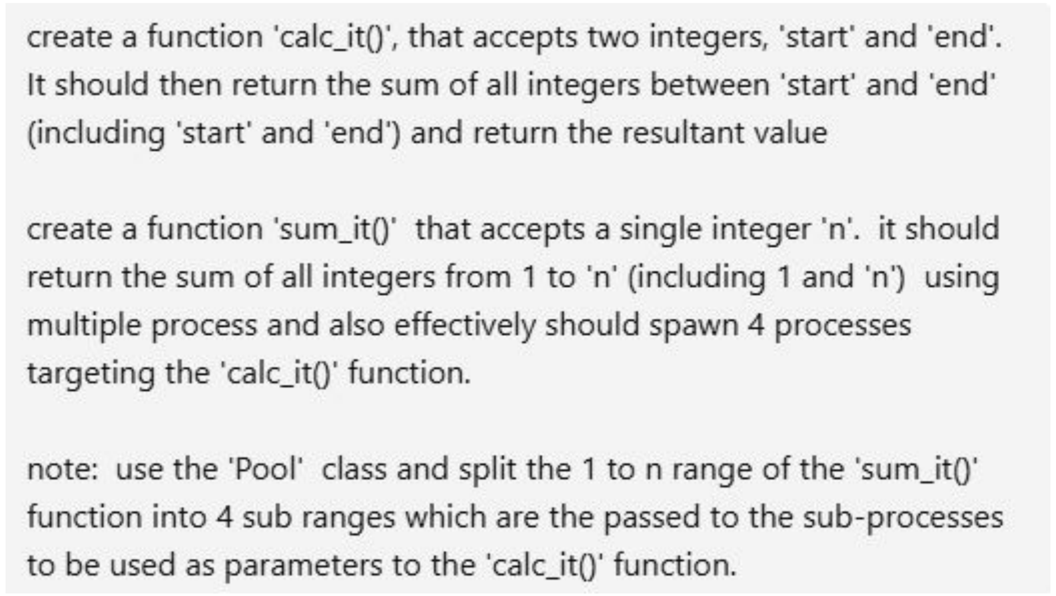
The answer is no.

There is an important difference between correlation and causality:

- Correlation is a number that measures how closely the data are related
- Causality is the conclusion that x causes y.

Types of Correlation Coefficients:

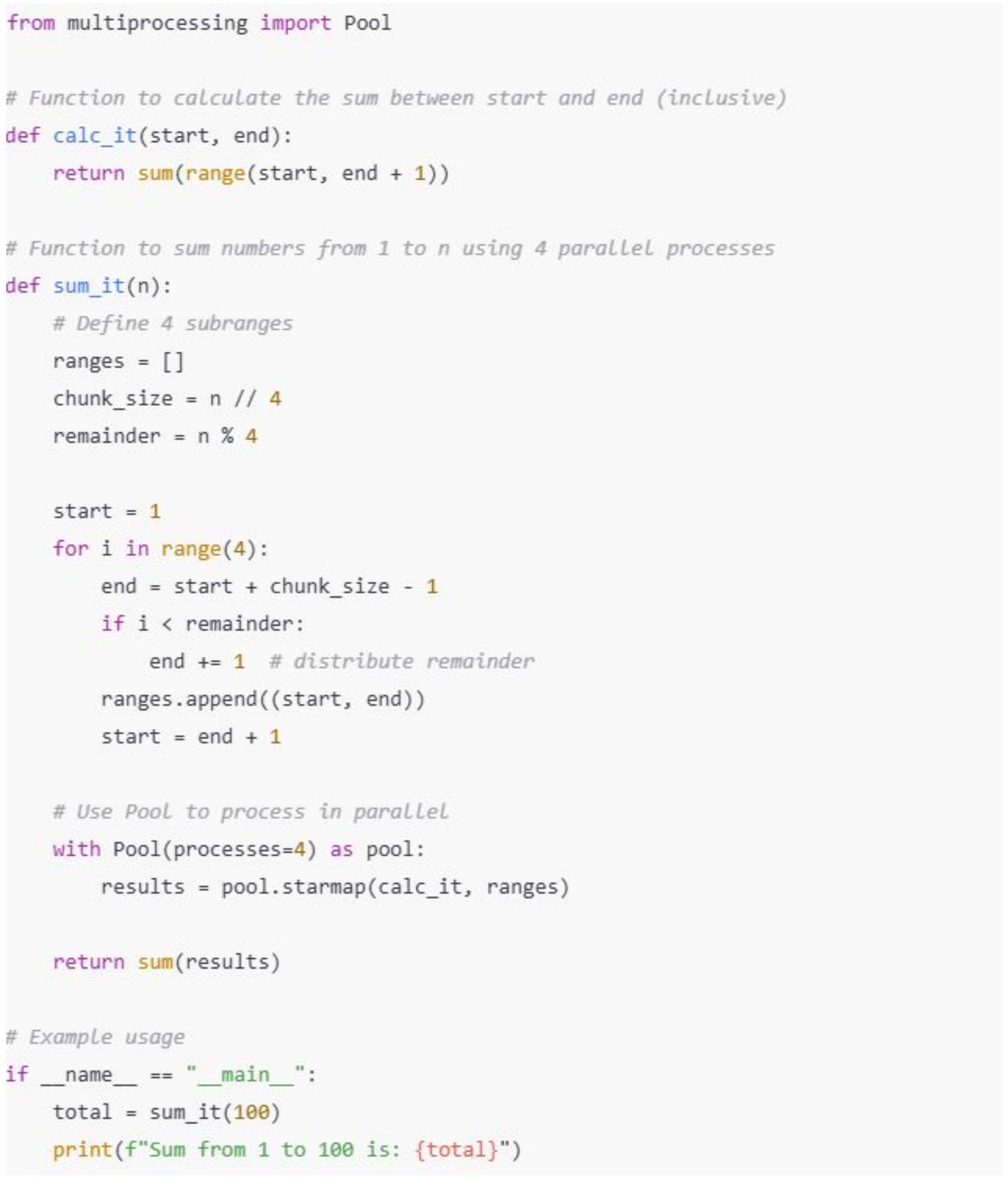
- **Pearson's** – for linear relationships, assumes normal distribution
- **Spearman's** – for ranked data or monotonic relationships
- **Kendall's tau** – for ordinal data, less sensitive to ties



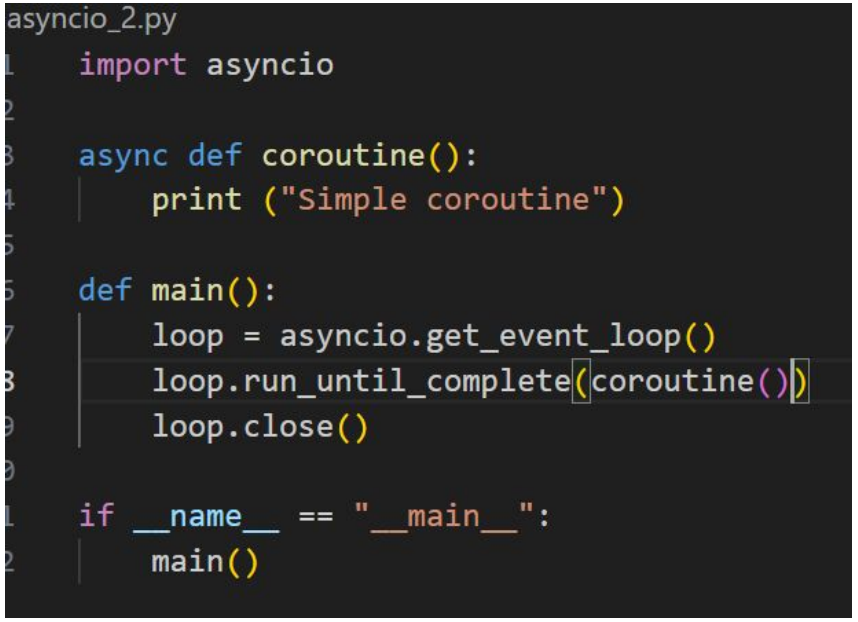
create a function 'calc_it()', that accepts two integers, 'start' and 'end'. It should then return the sum of all integers between 'start' and 'end' (including 'start' and 'end') and return the resultant value

create a function 'sum_it()' that accepts a single integer 'n'. it should return the sum of all integers from 1 to 'n' (including 1 and 'n') using multiple process and also effectively should spawn 4 processes targeting the 'calc_it()' function.

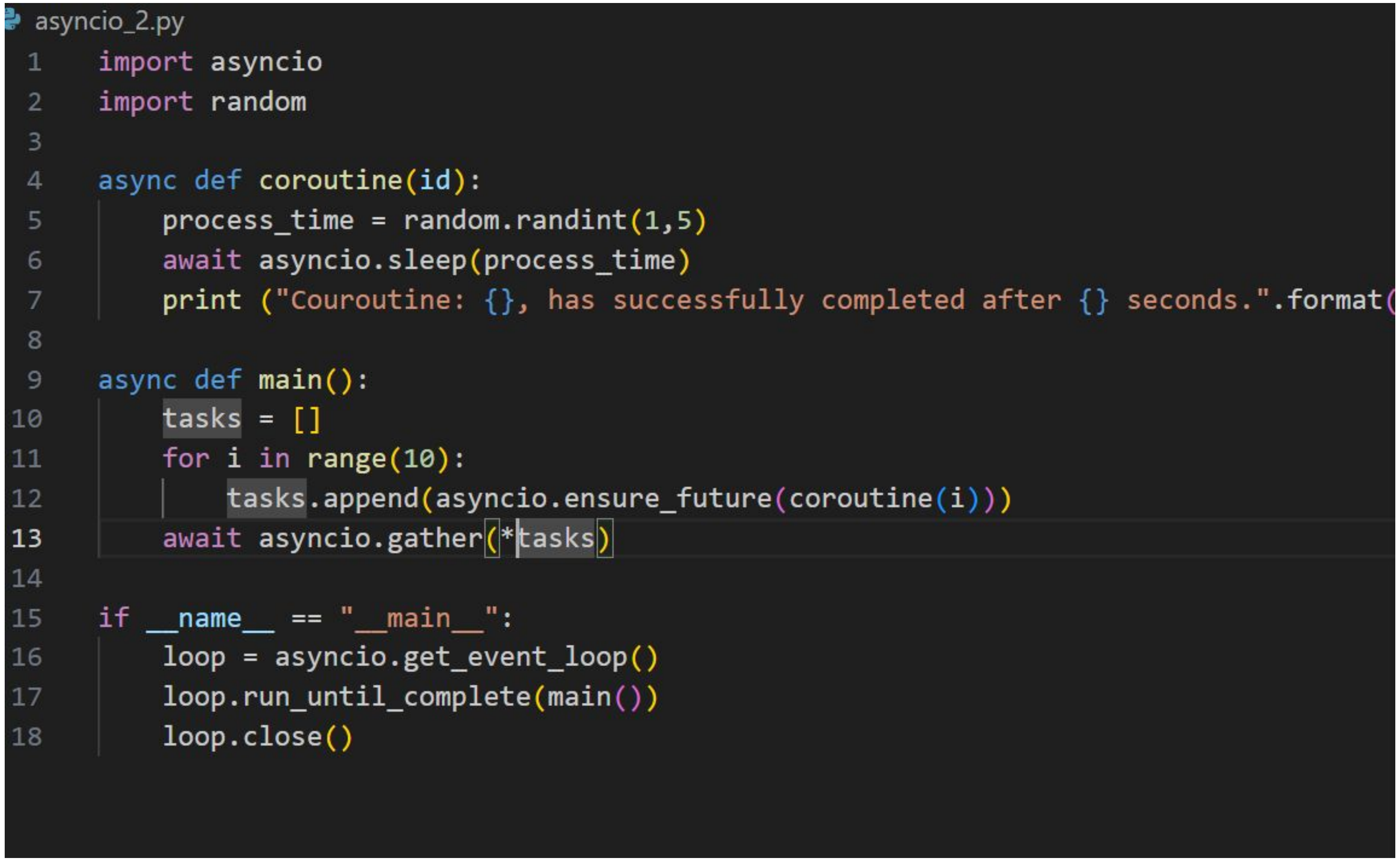
note: use the 'Pool' class and split the 1 to n range of the 'sum_it()' function into 4 sub ranges which are the passed to the sub-processes to be used as parameters to the 'calc_it()' function.



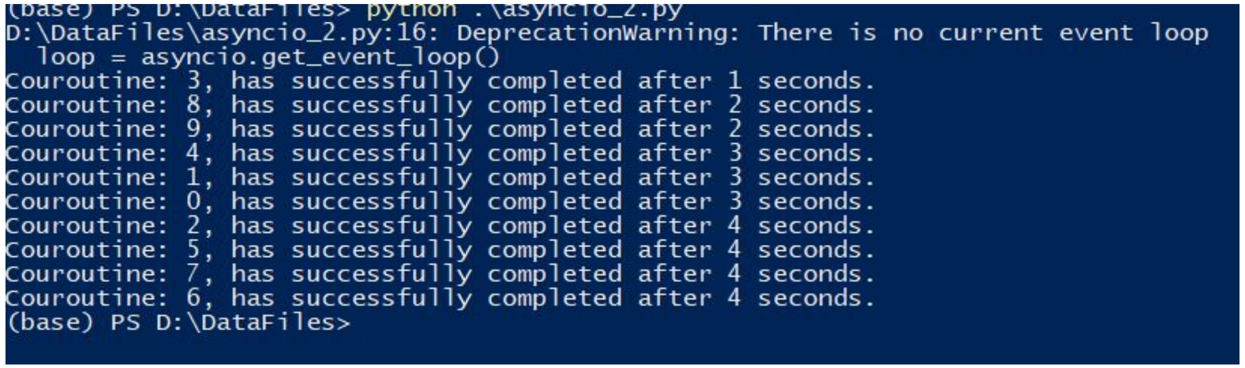
```python
from multiprocessing import Pool

# Function to calculate the sum between start and end (inclusive)
def calc_it(start, end):
    return sum(range(start, end + 1))

# Function to sum numbers from 1 to n using 4 parallel processes
def sum_it(n):
    # Define 4 subranges
    ranges = []
    chunk_size = n // 4
    remainder = n % 4

    start = 1
    for i in range(4):
        end = start + chunk_size - 1
        if i < remainder:
            end += 1  # distribute remainder
        ranges.append((start, end))
        start = end + 1

    # Use Pool to process in parallel
    with Pool(processes=4) as pool:
        results = pool.starmap(calc_it, ranges)

    return sum(results)

# Example usage
if __name__ == "__main__":
    total = sum_it(100)
    print(f"Sum from 1 to 100 is: {total}")
```


asyncio_2.py

```python
import asyncio

async def coroutine():
    print ("Simple coroutine")

def main():
    loop = asyncio.get_event_loop()
    loop.run_until_complete(coroutine())
    loop.close()

if __name__ == "__main__":
    main()
```


asyncio_2.py

```python
import asyncio
import random

async def coroutine(id):
    process_time = random.randint(1,5)
    await asyncio.sleep(process_time)
    print ("Couroutine: {}, has successfully completed after {} seconds.".format(

async def main():
    tasks = []
    for i in range(10):
        tasks.append(asyncio.ensure_future(coroutine(i)))
    await asyncio.gather(*tasks)

if __name__ == "__main__":
    loop = asyncio.get_event_loop()
    loop.run_until_complete(main())
    loop.close()
```



```
(base) PS D:\DataFiles> python .\asyncio_2.py
D:\DataFiles\asyncio_2.py:16: DeprecationWarning: There is no current event loop
  loop = asyncio.get_event_loop()
Couroutine: 3, has successfully completed after 1 seconds.
Couroutine: 8, has successfully completed after 2 seconds.
Couroutine: 9, has successfully completed after 2 seconds.
Couroutine: 4, has successfully completed after 3 seconds.
Couroutine: 1, has successfully completed after 3 seconds.
Couroutine: 0, has successfully completed after 3 seconds.
Couroutine: 2, has successfully completed after 4 seconds.
Couroutine: 5, has successfully completed after 4 seconds.
Couroutine: 7, has successfully completed after 4 seconds.
Couroutine: 6, has successfully completed after 4 seconds.
(base) PS D:\DataFiles>
```