UNIT

1 The Digital Lab Book

Learning Outcomes

By the end of this unit, students will be able to

- identify the main parts of a computational notebook and their functions (Remember).
- explain how Python executes commands and displays output (Understand).
- perform basic arithmetic operations using Python (Apply).
- plot a straight-line graph for a given linear equation and observe changes (Apply).

Programming Activity

Activity 1.1: Meet Your Environment

What is a "code editor" or "IDE"?

A code editor is like a simple, blank piece of parchment (***patra***) and a pen (***lekhani***). It is a clean space where you write your code, but that's often all it does. Common examples include **Notepad** (on Windows) and **Sublime Text.**

An **IDE** (Integrated Development Environment) is like a complete **scholar's desk.** It is a comprehensive setup that includes:

- **The Editor:** The same parchment and pen for writing.
- **A "Run" Button:** A tool that reads your code and displays the result instantly.
- **A Guide** (***shabdakosh***): A tool that suggests the right words as you type (auto - completion).
- **A Corrector** (***vyakaran***): A tool that puts a red line under any spelling or grammar mistakes in your code, helps you fix them (debugging).

Popular IDEs, especially for Python, include *PyCharm Community*, the beginner-friendly *Thonny*, and *Visual Studio Community*, which provides advanced features.

In short, an editor is primarily used for writing, whereas an IDE is a complete workshop that helps you write, check, and run your code in one place.

Understanding the Difference between a Plain-Text (.py) File and the "Run" Button

In ancient Bharat, a master sculptor (*Shilpi*) first created a clear, complete vision of a murti in his mind. He planned every detail, from the curve of the hand to the folds of the cloth. This flawless mental plan was his *Sankalpa* (a structured intention).

However, the plan alone did not carve the stone. The sculptor then had to pick up his chisel and begin the physical, step-by-step act of carving. This action, guided by the plan, was the *Kriya* (the execution).

This is exactly how your code works.

- Your .py file is the *Sankalpa*. It is the complete and detailed plan holding all your instructions, but it does nothing by itself.
- The "Run" button is the *Kriya*. It is the action that tells the computer to read your plan and execute it, step-by-step, to produce a real result.

Writing and Running Your First Command to See Output in the "Console" or "Terminal"

Think of yourself as a director 🎬 and the computer as an actor.

A command is an instruction you write in your script. One of the most common first command is print( ). When you write print("Hello, world!"), you are writing a line in the script that tells your actor: "Say 'Hello, world!'"

When **you run the code**, you are shouting "Action!" 🎬 The computer (your actor) reads that line of the script and performs the instruction.

The "**Console**" or "**Terminal**" is the **stage**. It is the special output area where the actor delivers the line for you (the audience) to see. When you run your print command, the text "Hello, world!" appears in the console. It is the computer's way of showing that it has followed your command.

Your First Task:

Run the code cell below. This is your first *Kriya*!

```
print("Hello, world! I am ready to analyze data.")
```

```
Hello, world! I am ready to analyze data.
```

You should see the text “Hello, world! I am ready to analyze data” appear in the output area (the “stage”). Congratulations! you have just given your first command to the computer!

Activity 1.2: Code as a Calculator

In Python, you can perform calculations just like on a calculator. The computer understands these basic symbols:

- + for Addition
- - for Subtraction
- * for Multiplication
- / for Division

In our last activity, we used print( ) to tell the computer (our “actor”) to say a sentence. We can also use it to tell the actor to “calculate this andW say the answer.”

Task 1: Checking the Order of Operations

Does the computer follow the rules of The BODMAS rule specifies the sequence in which mathematical operations must be performed: Brackets (Parentheses) first, followed by Orders (Exponents), then Division and Multiplication, and finally Addition and Subtraction.

B - Brackets

O - Orders (also called "of", meaning powers/exponents and roots)

D - Division

M - Multiplication

A - Addition

S - Subtraction

Let’s test it.

Mentally calculate the value of 10 + 5 * 2. Is the answer 30 or 20?

Run the code cell below to see how the computer solves it.

```
print(10 + 5 * 2)```
```

```
print(10 + 5 * 2)
```

```
20
```

Task 2: Your Turn

Write the code in the cell below to solve and print the answer for this problem: (50 - 14) / (2 * 3)

```
# Type your code here to solve print(50 - 14) / (2 * 3)
```

Task 3: Dealing with Messy Numbers

When you divide, you often get long, messy decimal numbers. Run the code cell below:

```
print(10 / 3)
```

```
3.3333333333333335
```

You will see an output like 3.3333333333333335. This is not very readable.

We can instruct the computer how to print the number. This is called formatting. Formatting allows us to control not only what the computer prints, but also how it prints it.

In Python, we use a special f-string for formatting. The notation .2f tells Python to format this as a number with two decimal places.

Run this code to see the difference:

```
# We save our messy number in a "variable"
my_number = 10 / 3

# Now we print it using formatting
print(f"The formatted answer is {my_number:.2f}")
```

```
The formatted answer is 3.33
```

Activity 1.3: Plotting Your First Graph

Understanding the concept of a "library"

Think of Python as a basic toolbox  . It has all the essential tools like print( ) and basic mathematical operators.

But what if you need a specialised tool, like a power drill for a larger task? You would not build one from scratch; instead, you would get it from your workshop.

A **library** in Python functions in a similar way. It is a pre-built collection of advanced tools.

For data analysis, we need tools to handle number lists (from 'numpy') and tools to draw graphs (from 'matplotlib').

How to import a library

To use these tools, you must first include them into your script. We use the import command, which is like saying, "Go to the workshop and get me this specific toolbox."

```
# We import the 'matplotlib.pyplot' toolbox and give it a short nickname 'plt'
import matplotlib.pyplot as plt
# We import the 'numpy' toolbox and give it the nickname 'np'
import numpy as np
```


```
# We import the 'matplotlib.pyplot' toolbox and give it a short nickname 'plt'
import matplotlib.pyplot as plt
# We import the 'numpy' toolbox and give it the nickname 'np'
import numpy as np
```

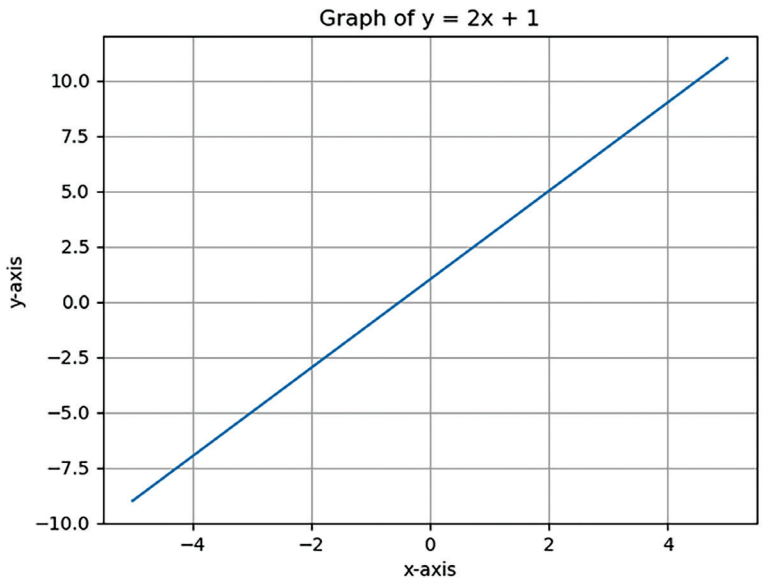

Running a complete script to plot y = 2x + 1

Now, let's put it all together. The script below will:

1. Import the required libraries.
2. Define the x and y values for the equation.
3. Use the plt tools to draw the graph.
4. Use plt.show() to display the graph in a new window.

Task: Run the complete script below. A new window displaying your first graph should appear.

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

# 1. Define x-values (100 points from -5 to 5)
x = np.linspace(-5, 5, 100)

# 2. Calculate y-values using our equation
y = 2*x + 1

# 3. Use the 'plt' library to set up the graph
plt.plot(x, y)
plt.title("Graph of y = 2x + 1")
plt.xlabel("x-axis")
plt.ylabel("y-axis")
plt.grid(True)

# 4. Show the final graph
plt.show()
```



What did np.linspace do?

The line x = np.linspace(-5, 5, 100) might look complex, but it is simple.

It is a command from the numpy library (which we nicknamed np) that creates a list of evenly spaced numbers.

Think of it like this:

- -5 represents the starting point.
- 5 represents the ending point.
- 100 specifies the total number of points we want.

Thus, this command gives us 100 numbers perfectly spaced out between -5 and 5, which we use as the coordinates for our x-axis to draw a smooth line.

Interactive Activity1.1

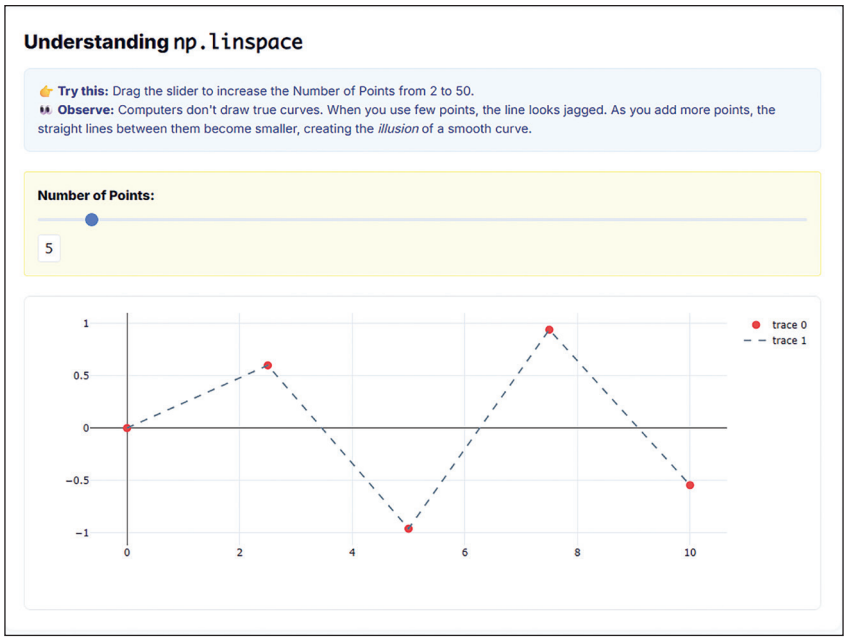
To understand this concept more clearly, watch the animation below.

The animation shows the same mathematical function drawn using different numbers of points generated by np.linspace. When only a few points are used, the graph looks broken or rough. As more points are added, the computer connects them more closely, and the curve appears smoother.

This helps you see that smooth graphs are created by calculating many points and joining them, not by drawing curves directly.

np.linspace Demo

https://oldrusti.com/BSB/Activities.html#linspace



Activity 1.4: Experimenting by Changing Code

You will now take on the role of a data scientist👩‍🔬. Your job is to experiment by changing the code you just ran.

This "manual edit and re-run" cycle is the core workflow for all IDEs. It is a fundamental skill that allows you test ideas and see the results immediately.

Interactive Activity1.2

Before changing the code manually, let us first experiment in a simpler way.

In the interactive activity below, you can move sliders to change the values of m and c in the equation: $y = mx + c$

As you move the sliders, the graph updates instantly.

This allows you to observe:

- how changing m affects the steepness of the line
- how changing c shifts the line upward or downward

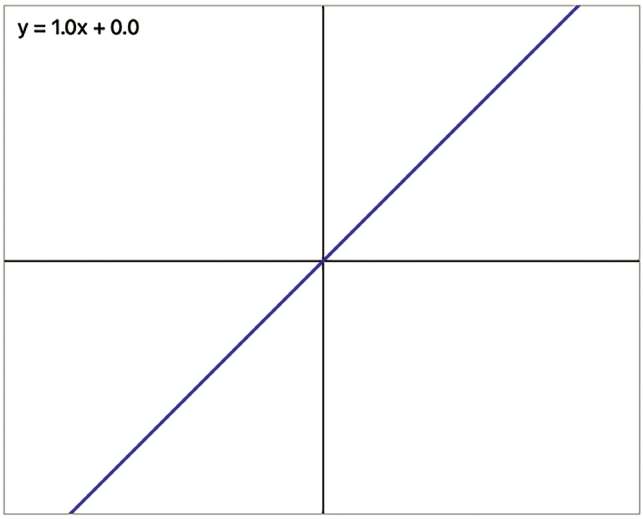
Think of this as a quick visual experiment before you perform the same changes directly in the code.

Equation of a Line

https://oldrusti.com/BSB/Activities.html#linear (1.1)

m (steepness): 1

c (touch y-axis at): 0

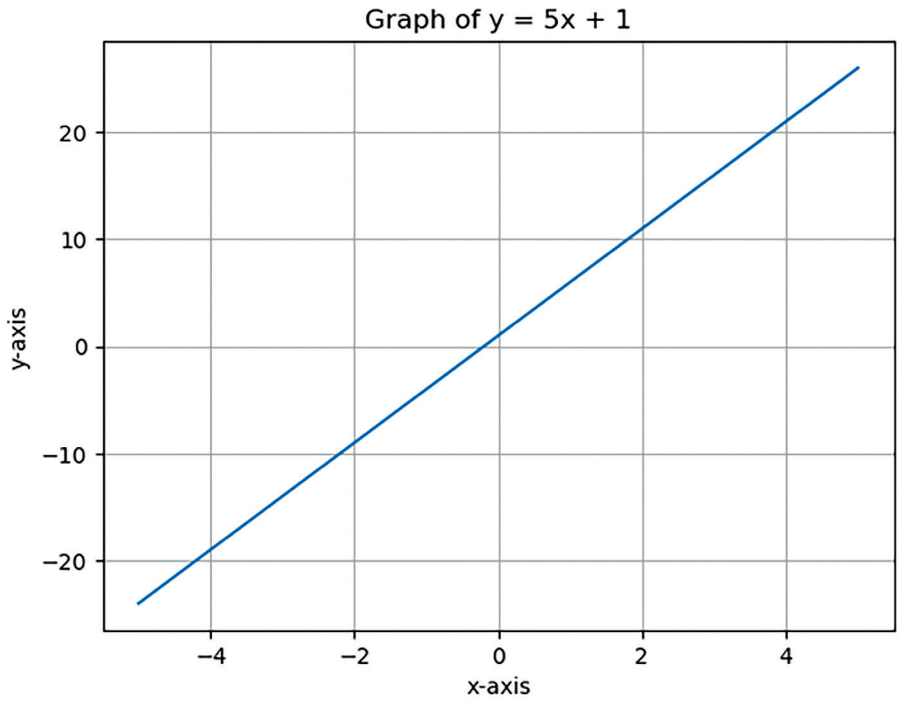


Task 1: Change the Steepness (m)

1. Copy the complete code from Activity 1.3 into the empty cell below.
2. Find the line y = 2 * x + 1.
3. Change it to y = 5 * x + 1.
4. Re-run the entire script.

Observe the new graph: Did the line get flatter or steeper?

```
# Copy the code from Activity 1.3 here
# and make your first change

import numpy as np
import matplotlib.pyplot as plt

# 1. Define x-values
x = np.linspace(-5, 5, 100)

# 2. Calculate y-values (CHANGED)
y = 5*x + 1

# 3. Use the 'plt' library to set up the graph
plt.plot(x, y)
plt.title("Graph of y = 5x + 1")
plt.xlabel("x-axis")
plt.ylabel("y-axis")
plt.grid(True)

# 4. Show the final graph
plt.show()
```

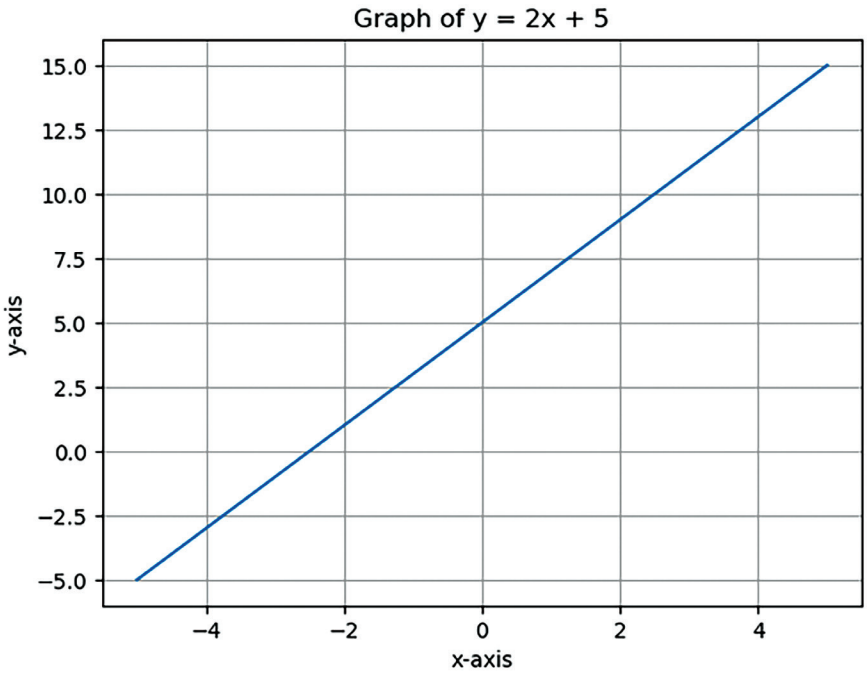


Task 2: Change the point where graphs meets y-axis (C)

1. Go back to the code you just pasted (or copy the original code again).
2. This time, change the line y = 5x + 1 (*or the original y* = 2x + 1) to y = 2*x + 5.
3. Re-run the script.

Observe the change: Did the steepness of the line change, or did it shift upward and downward?

```
# Copy the code again and make your second change
import numpy as np
import matplotlib.pyplot as plt

# 1. Define x-values
x = np.linspace(-5, 5, 100)

# 2. Calculate y-values (CHANGED)
y = 2*x + 5

# 3. Use the 'plt' library to set up the graph
plt.plot(x, y)
plt.title("Graph of y = 2x + 5")
plt.xlabel("x-axis")
plt.ylabel("y-axis")
plt.grid(True)

# 4. Show the final graph
plt.show()
```



Reflection

- Based on your experiments, which part of $y = mx + c$ controls the steepness?
- Which part determines whether the line shifts upward or downward?
- How is this process of "changing a value and seeing the result" similar to a scientific experiment?

UNIT

2

The Data Detective

Learning Outcomes

By the end of this unit, students will be able to

- recognise rows and columns in a dataset stored as a DataFrame.
- explain the meaning of mean, median, and mode.
- calculate basic statistical measures using Python.
- interpret patterns in data using tables and histograms.

Introduction

Have you ever watched a detective in a story? They do not guess. They observe, compare, collect clues, and finally discover the hidden pattern.

A Data Detective does exactly the same thing, but instead of fingerprints and footprints, we look at numbers inside a dataset.

A story from the past

Long before computers existed, Indian scholars were already practising the skills of a Data Detective - observing nature carefully, noting patterns, and drawing reasoned conclusions. Their tools were simple: the sky, the seasons, and years of patient observation.

In ancient Bharat, astronomers and scholars followed the principle of *Parīkṣā* - systematic observation. Texts such as the *Bṛhat Saṁhitā* (6th century CE) describe how scholars studied cloud formations, winds, rainfall patterns, and the movement of the sun and stars. Their goal was not guesswork, but careful measurement and logical inference, known as Anumāna.

These observations were recorded over long periods, sometimes in palm-leaf manuscripts, and later used to understand:

- When the monsoon might arrive

- How seasonal temperatures varied
- Which months were favourable for farming
- How celestial patterns related to timekeeping

While they did not have digital datasets, they did something very similar: they collected data, looked for patterns, and made predictions based on evidence.

In that sense, when you analyse weather data today, you are continuing a tradition that goes back more than a thousand years — only with tools that scholars of the past might have considered magical.

Programming Activity

Activity 2.1: Creating and Loading Your First Dataset

We will work with a small weather dataset. This dataset has three columns:

- Temperature (°C)
- Humidity (%)
- Air Pressure (hPa)

We will start with just 10 entries, similar to 10 days of simple observations.

Here is the dataset for you to read:

Day	Temperature (°C)	Humidity (%)	Air Pressure (hPa)
1	22	60	1012
2	24	62	1010
3	25	58	1011
4	23	65	1013
5	26	70	1014
6	27	72	1012
7	28	68	1011
8	24	64	1013
9	22	59	1010
10	23	61	1012

Now we will put this same information into Python so the computer can work with it. We will use the pandas library, which helps Python handle tabular data. Pandas stores a data in a structure called a DataFrame, which is simply a digital version of the table you saw above.

LearningNote

In Python, a dictionary is a way of storing information in pairs. Each pair has a name and a list of values. You can think of it like a labelled box, where each label has its own list. For example, in our weather data:

- "Temperature" is a label, and it has a list of temperature values
- "Humidity" is another label, with its own list
- "AirPressure" is another label, with its own list
- All these lists have the same length, so each position together forms one row of data. When we give this dictionary to pandas, it converts it into a Data Frame. A Data Frame is simply a digital table, similar to one you draw in your notebook, with:
- Column names at the top
- Rows of values blow

so in simple terms:

- Dictionar helps us organise data by column
- A DataFrame helps us see and work with the data as a table. This is why we first write the data as dictionary and then convert it into a DataFrame.

Interactive Activity 2.1

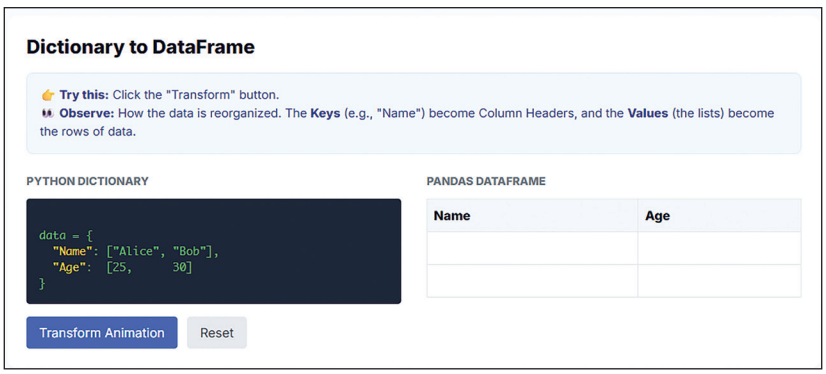
The animation below shows how data written as a Python dictionary is converted into a table called a DataFrame.

In a dictionary, data is stored as key–value pairs. In a DataFrame, the same data is organised into rows and columns, similar to a table in a notebook. The animation helps you understand how each key becomes a column and how the data is organised into rows.

This visual connection makes it easier to understand how code creates the tables you see on the screen.

Dict to DataFrame

https://oldrusti.com/BSB/Activities.html#dict2df



Let us create this DataFrame directly inside Python.

```
import pandas as pd

data = {
    "Temperature": [22, 24, 25, 23, 26, 27, 28, 24, 22, 23],
    "Humidity":    [60, 62, 58, 65, 70, 72, 68, 64, 59, 61],
    "AirPressure": [1012, 1010, 1011, 1013, 1014, 1012, 1011, 1013, 1010, 1012]
}

df = pd.DataFrame(data)
df
```

	Temperature	Humidity	AirPressure
0	22	60	1012
1	24	62	1010
2	25	58	1011
3	23	65	1013
4	26	70	1014
5	27	72	1012
6	28	68	1011
7	24	64	1013
8	22	59	1010
9	23	61	1012

when you run this cell, Python will display the DataFrame — the same table you read earlier, now stored inside the computer.

You will use this table throughout the rest of the unit, and in some later chapters. Since we will need it again, it is useful to save it as a file.

A common way to store tables is in a format called CSV, which stands for “Comma-Separated Values.” It is a simple text format where each row of the table is written on a new line, and commas separate the values. Many programs, not just Python, understand CSV files.

Let us save our table in this format so we can load it whenever needed.

```
df.to_csv("weather_data.csv", index=False)
```

This creates a file named weather_data.csv in your notebook folder. Now your dataset is ready to be used again in the next activities.

Activity 2.2: Becoming a Statistical Detective

Now that your dataset is ready, you can begin examining it just like a detective examines clues. One of the first ways to understand a list of numbers is to identify what is "typical" or "central." You have already learned three useful ideas: the **mean**, the **median**, and the **mode**. Python can calculate these for you using simple commands.

We will begin with the **Temperature** column. The DataFrame stores this column under the name "Temperature", so we can ask Python to look at it and compute these values.

```
mean_temp = df["Temperature"].mean()
median_temp = df["Temperature"].median()
mode_temp = df["Temperature"].mode()[0]

print(f"Mean Temperature: {mean_temp} \nMedian Temperature: {median_temp} \nMode
```

```
Mean Temperature: 24.4
Median Temperature: 24.0
Mode Temperature: 22
```

When you run this, Python will give you three numbers. These help you understand the overall behaviour of the temperature across the ten days. The mean tells you the average, the median tells you the middle value when the temperatures are arranged in order, and the mode tells you which temperature occurred most often.

Try performing the same exercise manually. Compare the time taken to calculate these values with the time taken by the computer.

You can do the same for the Humidity column. This will help you compare the two measures and determine whether humidity varies more or less than temperature in this dataset.

Interactive Activity 2.2

Before moving on, let us look at one more important idea.

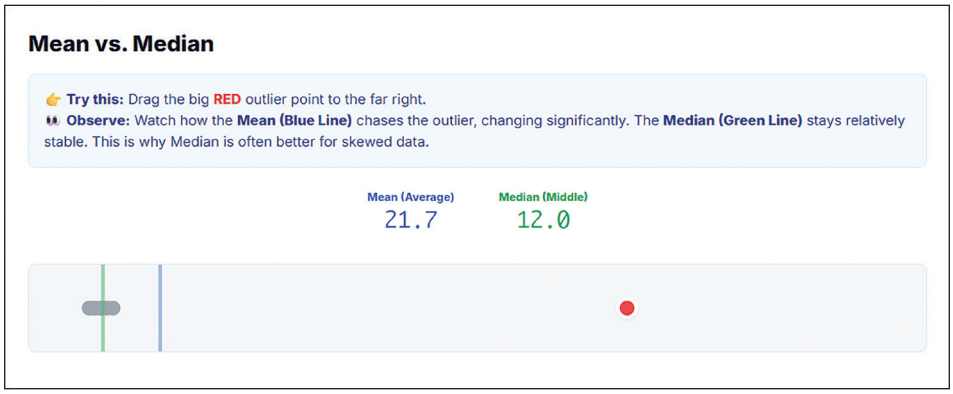
Sometimes a dataset contains an unusual value that is much higher or lower than the others. Such a value is called an outlier. The animation below shows how adding an outlier can change

the mean significantly, while the median often remains nearly the same.

This helps you understand why the median can sometimes give a better idea of a typical value.

Outliers

https://oldrusti.com/BSB/Activities.html#outliers




```
mean_hum = df["Humidity"].mean()
median_hum = df["Humidity"].median()
mode_hum = df["Humidity"].mode()[0]

print(f"Mean Humidity: {mean_hum} \nMedian Humidity: {median_hum} \nMode Humidity
```

```
Mean Humidity: 63.9
Median Humidity: 63.0
Mode Humidity: 58
```

As you look at the results, observe how these values summaries the entire column. Instead of examining all ten values one by one, these three measures help you understand the overall pattern at a glance. This is exactly what a Data Detective does—use a few effective calculations to interpret a large set of data.

If you want to compare columns directly, print the results together:

```
print("Temperature:", mean_temp, median_temp, mode_temp)
print("Humidity:   ", mean_hum, median_hum, mode_hum)
```

```
Temperature: 24.4 24.0 22
Humidity:    63.9 63.0 58
```

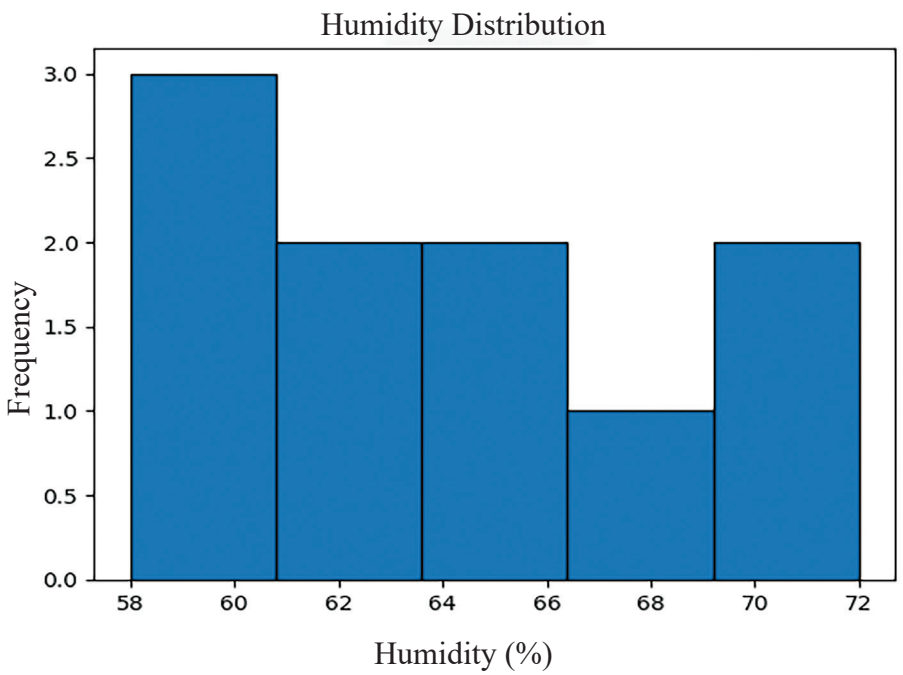
Activity 2.3: Visualising Humidity with a Histogram

Numbers become easier to understand when they are presented visually. In the same way that ancient scholars observed cloud shapes or star positions to identify patterns, you can look at a picture of your data to understand how values are spread out. One effective way to do this is by using a histogram.

A histogram shows how often certain ranges of values appear. If you think of the humidity values as grains collected in different pots, a histogram shows how full each pot is. Higher bars indicate that those values occurred more frequently; lower bars indicate that they were less common.

Python can draw this picture for you using the plotting library matplotlib. We will create a histogram of the Humidity column.

```
import matplotlib.pyplot as plt

plt.hist(df["Humidity"], bins=5, edgecolor="black")
plt.xlabel("Humidity (%)")
plt.ylabel("Frequency")
plt.title("Humidity Distribution")
plt.show()
```



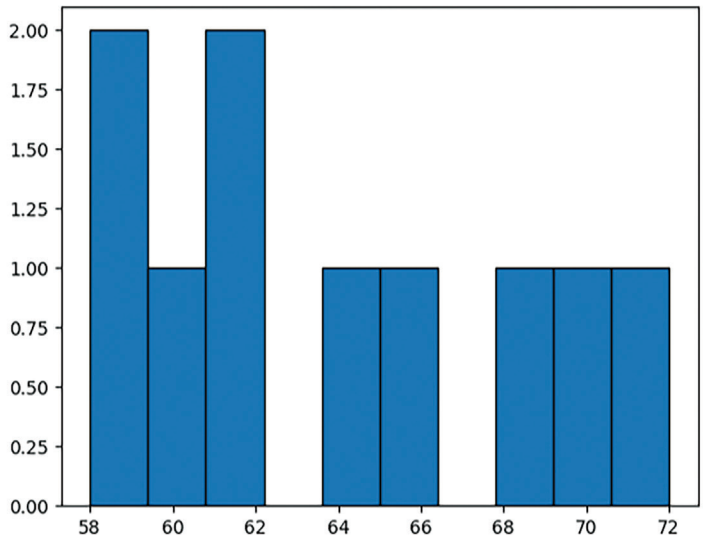
When you run this, you will see a plot where the x-axis shows humidity ranges and the y-axis shows the number of days in each range. By looking at the height of the bars, you can quickly

tell whether most days were more humid or less humid.

Try looking at the shape of the histogram. Are the bars mostly together, or are they spread out? Is there a group where most of the days fall? These observations help you understand the behaviour of the humidity values without examining each number individually.

You can also experiment by changing the number of bins. Try changing it to 10:

```
plt.hist(df["Humidity"], bins=10, edgecolor="black")
plt.show()
```



More bins divide the humidity values into smaller ranges, giving you a more detailed picture. Fewer bins provide a simpler overview. A Data Detective tries both to get a clear understanding of the data.

Interactive Activity 2.3

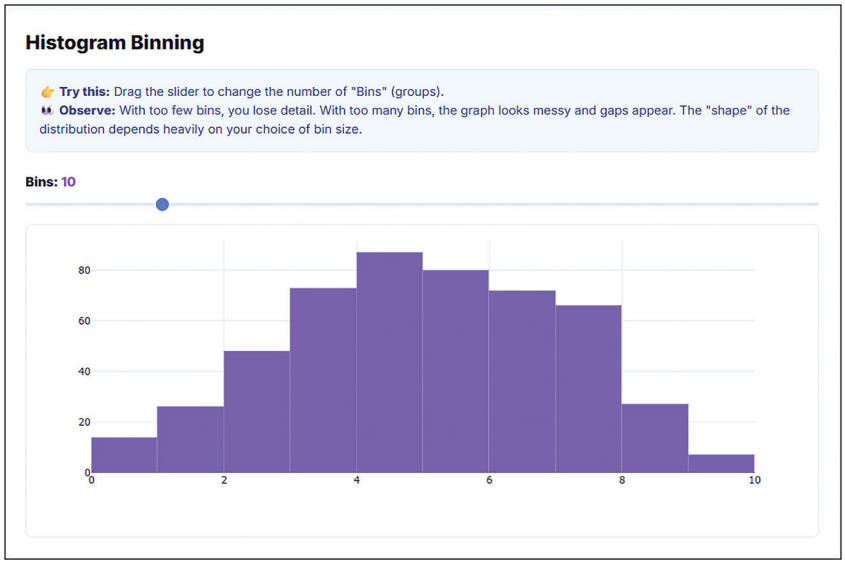
In the interactive activity below, you can explore how histograms change based on how data is grouped.

First, choose a variable such as Temperature, Humidity, or Air Pressure. Then change the number of bins using the slider. As you adjust the bins, the histogram updates instantly.

This helps you see that the same data can look different depending on how it is grouped. By experimenting with different variables and bin sizes, you will understand why choosing bins carefully is important when interpreting graphs.

Histogram Binning

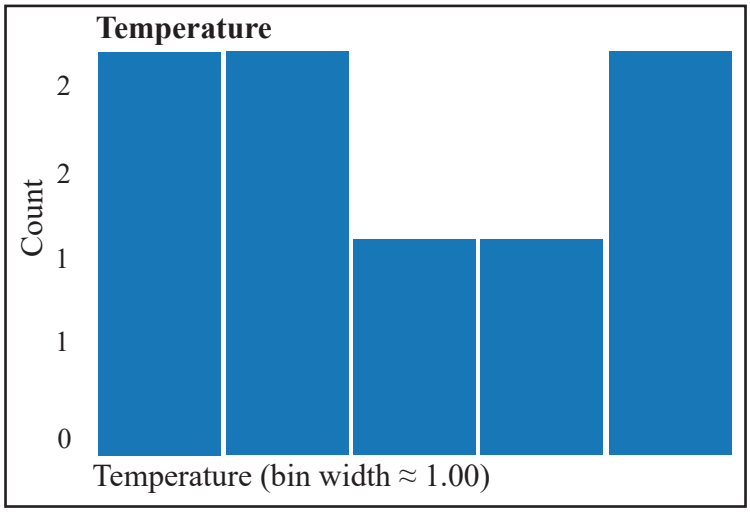
https://oldrusti.com/BSB/Activities.html#histograms



Histogram bins demo — status: drawn ✅

Variable: Temperature

bins: 6



In the next activity, you will continue this detective work by interpreting the patterns you see and connecting them to the idea of making simple inferences from data.

Activity 2.4: Interpreting the Patterns You See

Now that you have drawn a histogram, you can begin to interpret it in the way a detective interprets clues. A histogram is more than a picture; it summarises many values at once. Instead of reading every humidity number, you can look at the bars and understand what the data is trying to tell you.

Begin by identifying which bars are the tallest. Taller bars indicate that the humidity values in that range occurred more often. Shorter bars indicate that those values were less common. This simple observation already tells you something about the overall pattern of humidity across the ten days.

Let's look at the actual values:

```
df["Humidity"]
```

```
0    60
1    62
2    58
3    65
4    70
5    72
6    68
7    64
8    59
9    61
Name: Humidity, dtype: int64
```

Now compare the list with the shape of your histogram. You will notice how the histogram gathers nearby values into a group and shows them as a single bar. This makes it easier to see whether the humidity values are mostly high, mostly low, or nicely spread out in the middle.

To reflect as a Data Detective, consider the following questions:

- Do most days fall into a single humidity range, or are they spread across different ranges?
- Are there any humidity values that seem unusual or far from the others?
- How would you describe the humidity over these ten days to someone who has not seen the table or the plot?

You can also compare the histogram with the statistical measures calculated earlier. For example, if the mean and median humidity are close together, and the histogram looks fairly balanced on both sides, this suggests that the humidity values are not highly skewed or uneven.

Sometimes it helps to print the mean and median again to keep them in mind while you look at the histogram:

```
print("Mean Humidity:", mean_hum)
print("Median Humidity:", median_hum)
```

```
Mean Humidity: 63.9
Median Humidity: 63.0
```

The goal of this activity is not to get a single correct answer but to practise the habit of observing, comparing, and drawing simple conclusions. This is the same habit ancient scholars used when they studied patterns in the sky and weather. By combining numbers with visual patterns, you get a clearer picture of what the dataset is telling you.

In the next part, you will wrap up your detective work by creating a short summary of your findings. This small reflection will help you strengthen your understanding of how to read data and explain what you discover.

Activity 2.5: Your Data Detective Summary

You have now explored the dataset in three different ways: by reading the numbers, by calculating the mean–median–mode, and by looking at a histogram. Each method gives you a different kind of clue. The final step is to put these clues together and write a short summary of what you discovered.

Look at your DataFrame again, along with your statistical measures, and your histogram, and try to describe what the data shows. A summary does not need to be long. It simply needs to explain what patterns you noticed.

Here are some guiding questions you can think about while writing:

- What can you say about the temperature values over these ten days?

  Hint: *Look at how the temperatures change from day to day. Are they fluctuating widely or increasing gradually? Do they stay within a narrow range or a wide range?*

- Does the humidity seem mostly high, mostly low, or somewhere in the middle?

  Hint: *Think about the numbers in the Humidity column and the bars in your histogram Which humidity range has the tallest bar? That tells you where most of the days are.*

- Did the histogram show a clear cluster or spread-out values?

  Hint: *Look at whether many bars are tall in one area, or whether the bars are more evenly spread out across the x-axis. A cluster means that many values are gathered close together.*

- Which measure (mean, median, or mode) helps you the most in understanding the data?

  Hint: *Think about which value seems closest to what you observe in the table and the histogram. Does the mean appear typical? Does the median represent the middle clearly? Does the mode occur frequently in the list?*

A **Data Detective** always pauses at the end to make sense of the clues. This habit of reflection-asking, "What does this data really tell me?" - marks the beginning of meaningful analysis.

Practice Exercises

These exercises will help you strengthen the skills you learned in this unit. Try to solve them on your own first, and then check the solutions.

1. What is the highest humidity value in the dataset? What is the lowest?

   Hint: You can use df[“Humidity”].max() and df[“Humidity”].

2. How many days had a temperature greater than 24°C?

   Hint: Use a condition: df[df[“Temperature”] > 24].

3. Create a histogram for the Temperature column. Does it look similar to the humidity histogram or is it different? Explain your answer.

   Hint: Use plt.hist(df[“Temperature”], bins=5).

4. Calculate the range of Air Pressure (maximum minus minimum).

   Hint: Subtract min( ) from max( ).

5. If you had to predict the humidity for an 11th day, which value would you choose: mean, median, or mode? Explain.

Solutions

You can compare your answers with the examples below.

1. Highest and lowest humidity

```
df["Humidity"].max(), df["Humidity"].min()
```

```
(72, 58)
```

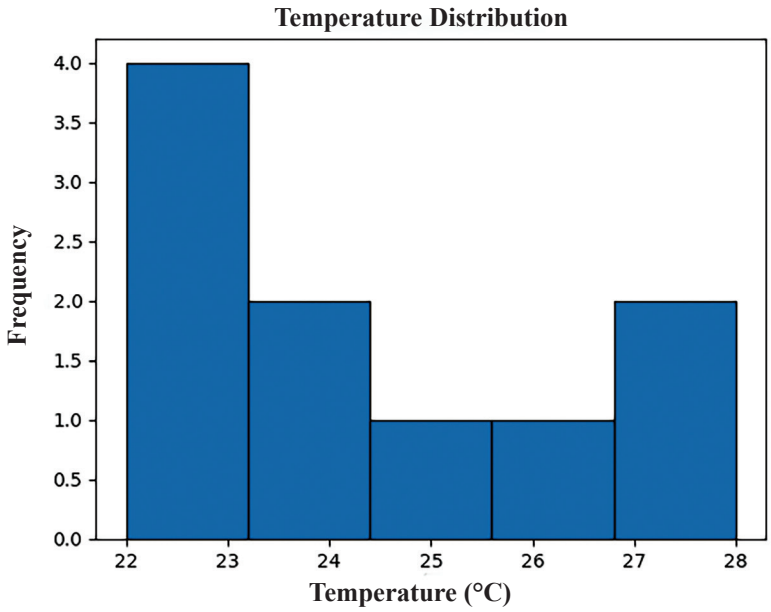
2. Days with temperature greater than 24°C

```
df[df["Temperature"] > 24]
```

	Temperature	Humidity	AirPressure
2	25	58	1011
4	26	70	1014
5	27	72	1012
6	28	68	1011

3. Histogram of Temperature

```
plt.hist(df["Temperature"], bins=5, edgecolor="black")
plt.xlabel("Temperature (°C)")
plt.ylabel("Frequency")
plt.title("Temperature Distribution")
plt.show()
```



Explanation: Most students will notice that the temperature values are closer together than the humidity values, so the distribution may look tighter or less spread out.

4. Range of Air Pressure

```
df["AirPressure"].max() - df["AirPressure"].min()
```

```
4
```

5. Choosing a humidity value for prediction

A reasonable explanation might be:

"I would choose the median because it represents the middle value and is less affected by unusual readings. It gives a stable idea of a typical humidity."

Students may also choose mean or mode. The key is that they explain their choice logically.

Reflection

You have now acted as a Data Detective — collecting clues from numbers, asking questions, and using both mathematical and visual tools to understand a small weather dataset. Take a moment to reflect on your journey through this unit.

- When you looked at the table for the first time, what did you notice immediately?
- How did the mean, median, and mode help you understand the data better than simply reading the numbers one by one?
- Did the histogram make any patterns clearer or easier to see?
- If someone asked you, “What kind of weather do these ten days show?”, what would you say?

Think about how this connects to the work of scholars from the past. Just as they observed clouds, winds, and stars patiently, you observed numbers and patterns patiently. You used tools like pandas and matplotlib instead of palm leaves, but the habit behind the work is the same: observe carefully, compare thoughtfully, and draw conclusions based on evidence.

Write a few sentences in the cell below about what you learned, or what felt interesting. This simple reflection helps you understand not just the data, but also your own way of thinking as a young scientist.

UNIT

3

The Great Equaliser

Learning Outcomes

By the end of this unit, students will be able to

- recognising why data with different scales cannot be compared directly.
- applying Min–Max scaling to numerical data.
- comparing original and scaled data using graphs.
- explaining how scaling enables fair comparison between quantities.

Introduction

In the previous unit, you explored a small weather dataset and learned how to describe it using statistics and simple visualisations. In this unit, you will explore a new idea. You will compare two different columns in the same graph.

Before you begin, think about this simple idea. In many Indian knowledge traditions, balance and proportion were considered important for understanding the world. Whether it was the ratio of ingredients in italicized, the harmony of notes in classical music, or the careful measurement of shadows in ancient astronomy, scholars always paid close attention to how different quantities relate to each other.

In the same way, when you compare numbers in a dataset, you must be careful about proportions. Some numbers may be small, such as daily temperatures, while others may be large, such as air pressure. If you try to compare them directly in a single graph, the larger numbers will dominate the picture and the smaller numbers will look almost invisible. To make a fair comparison, you need a way to place them on similar scales. This is called scaling.

You will now see this problem for yourself and then learn how to solve it.

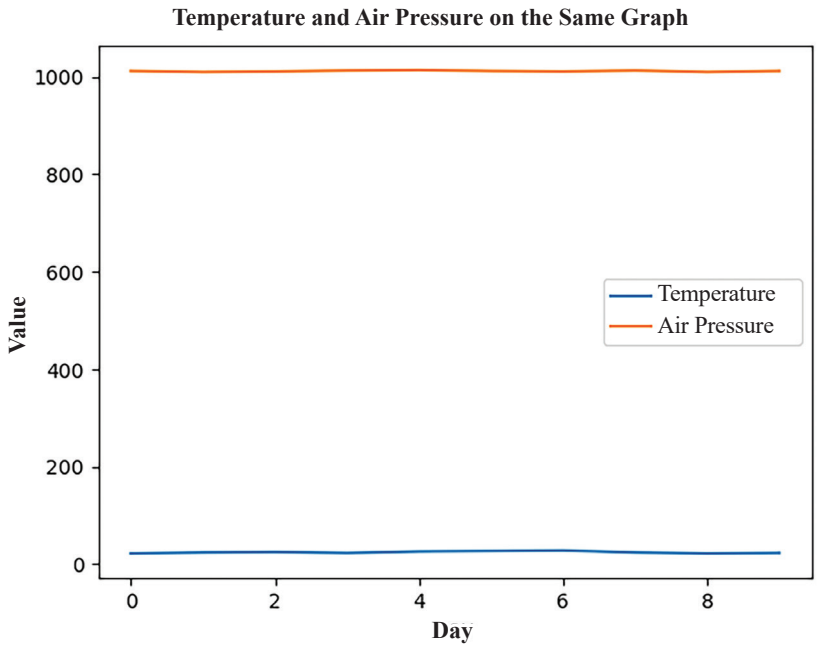
Programming Activity

Activity 3.1: Trying to Compare Without Scaling

We will begin with a simple attempt. In this activity, you will plot Temperature and Air Pressure on the same graph. Before proceeding, upload the weather dataset file we saved in Unit 2.

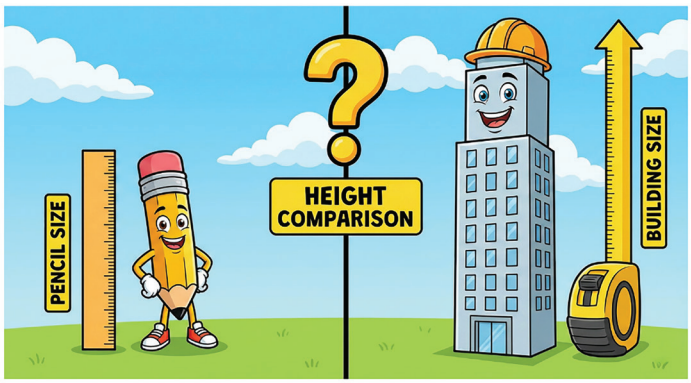
```
import pandas as pd
df = pd.read_csv('weather_data.csv')
```

```
import matplotlib.pyplot as plt

plt.plot(df["Temperature"], label="Temperature")
plt.plot(df["AirPressure"], label="Air Pressure")
plt.xlabel("Day")
plt.ylabel("Value")
plt.title("Temperature and Air Pressure on the Same Graph")
plt.legend()
plt.show()
```



When you run this, you will notice something unusual. The line representing Air Pressure appears high on the graph, while the line representing Temperature looks almost flat near

the bottom. The two lines cannot be compared properly because their numbers lie within very different ranges. Temperature values range from approximately 22 to 28, whereas Air Pressure values range from approximately 1010 to 1014.



This is the same as trying to compare the height of a pencil with the height of a building using the same measuring stick. The building will always dominate the picture, and the pencil will seem almost invisible.

This is why you need a method that equalises the scale.

Activity 3.2: Understanding the Idea of Scaling

Scaling is a simple idea. You take a list of numbers and convert them into a new list where the smallest value becomes 0 and the largest value becomes 1. All other numbers fall between 0 and 1. The relative relationships stay the same, but the scale becomes equal.

This idea is very similar to ratios, which you have alread y learned in mathematics. A ratio compares quantities by reducing them to a simpler form. Similarly, scaling reduces numbers to a simpler, common form so that they can be compared fairly.

The most common scaling method is called Min-Max scaling. You do not need to memorise a formula. You only need to understand that scaling helps the computer place different columns on the same level without changing their basic behaviour.

Now you will apply scaling to the Temperature and Air Pressure columns.

Interactive Activity 3.1

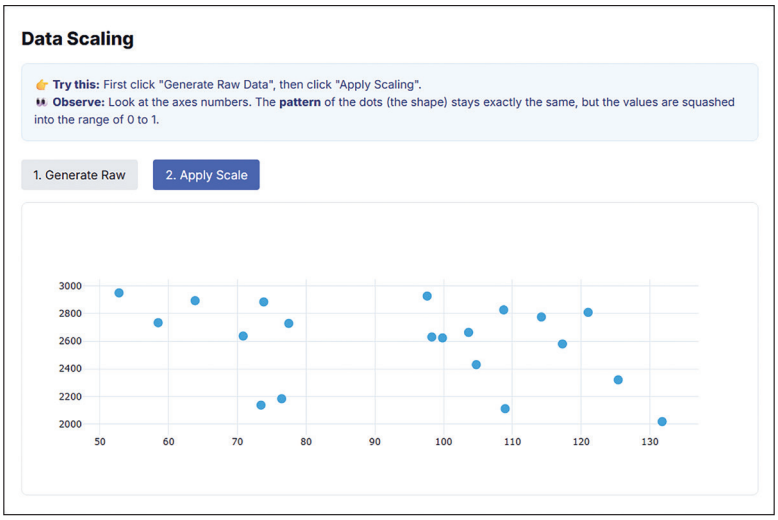
The animation shows how Min-Max scaling works step by step. It first displays the original data values on a number line and highlights the smallest and largest values.

Next, each value is rescaled so that all values lie between 0 and 1. As this happens, the positions of the points change, but their order remains the same.

This makes it clear that Min-Max scaling changes the scale of the data, not its pattern or relationships.

Data Scaling

https://oldrusti.com/BSB/Activities.html#scaling



Activity 3.3: Performing Min Max Scaling in Python

To scale a column, you subtract the minimum value and divide by the range.

Here is how to scale Temperature:

```
temp_min = df["Temperature"].min()
temp_max = df["Temperature"].max()

df["Temperature_scaled"] = (df["Temperature"] - temp_min) / (temp_max - temp_min)
df[["Temperature", "Temperature_scaled"]]
```

	Temperature	Temperature_scaled
0	22	0.000000
1	24	0.333333
2	25	0.500000
3	23	0.166667
4	26	0.666667
5	27	0.833333
6	28	1.000000
7	24	0.333333
8	22	0.000000
9	23	0.166667

Now do the same for Air Pressure:

```
press_min = df["AirPressure"].min()
press_max = df["AirPressure"].max()

df["AirPressure_scaled"] = (df["AirPressure"] - press_min) / (press_max - press_m
df[["AirPressure", "AirPressure_scaled"]]
```

	AirPressure	AirPressure_scaled
0	1012	0.50
1	1010	0.00
2	1011	0.25
3	1013	0.75
4	1014	1.00
5	1012	0.50
6	1011	0.25
7	1013	0.75
8	1010	0.00
9	1012	0.50

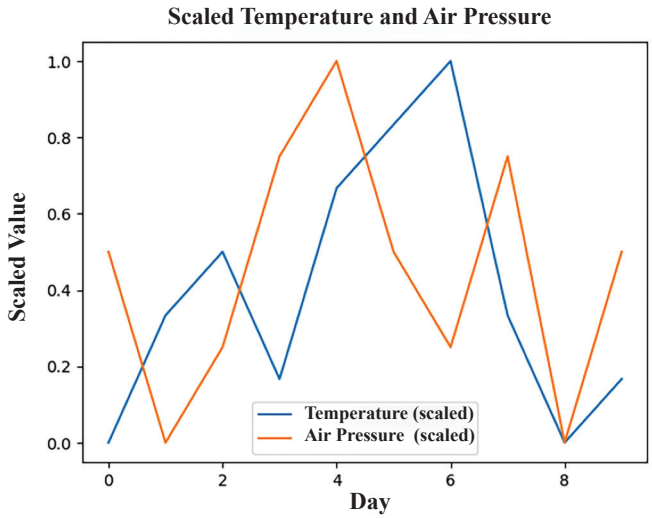
Look at the new values. The smallest original value becomes 0, and the largest becomes 1. All other values lie between 0 and 1. The meaning of the data remains unchanged; only the scale has changed.

This lets you compare the two columns more fairly.

Activity 3.4: Re-plotting After Scaling

Now you will create the same plot as before, but using the scaled columns.

```
plt.plot(df["Temperature_scaled"], label="Temperature (scaled)")
plt.plot(df["AirPressure_scaled"], label="Air Pressure (scaled)")
plt.xlabel("Day")
plt.ylabel("Scaled Value")
plt.title("Scaled Temperature and Air Pressure")
plt.legend()
plt.show()
```



This time, both lines appear in a similar range. You can clearly see how the Temperature and Air Pressure change over the ten days. Neither line hides the other. Both are visible and fair to compare.

Scaling does not change the story of the data. It only brings the characters to the same stage, so you can see how they move together.

Practice Exercises

Try these exercises to strengthen your understanding. Solutions are provided below each question.

1. What are the minimum and maximum values of the Temperature and Air Pressure columns?
2. After scaling, which value becomes 0 and which value becomes 1 in each column?

   Hint: Look at your tmperature_scaled and AirPressure_scaled columns.
3. Plot the original Temperature and the scaled Temperature on two separate graphs. What difference do you observe?

   Hint: The shape should stay the same even though the scale changes.
4. Scale the Humidity column using the same method.

   Hint: Follow the pattern used for Temperature and Air Pressure.
5. If two columns have very different units, such as Celsius and hPa, why is scaling helpful before comparing them?

   Hint: Think about whether large numbers will overshadow small ones in a graph.

Solutions

1. Maximum and Minimum values of the original columns

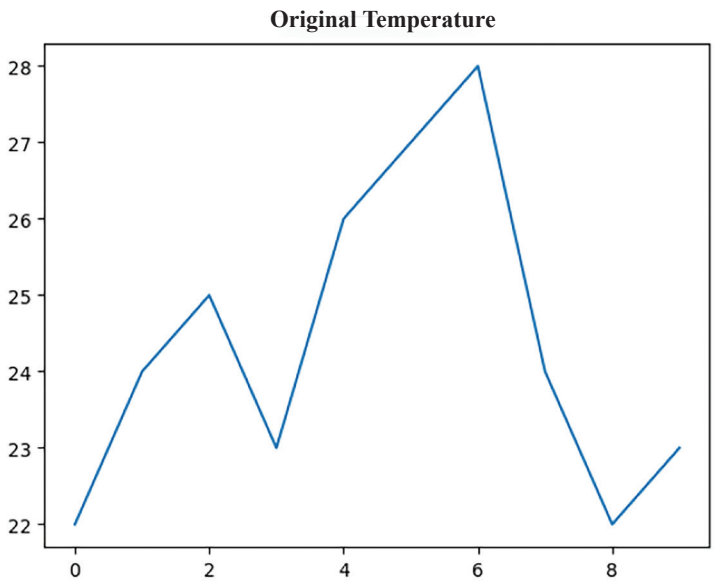
```
df["Temperature"].min(), df["Temperature"].max()
df["AirPressure"].min(), df["AirPressure"].max()
```

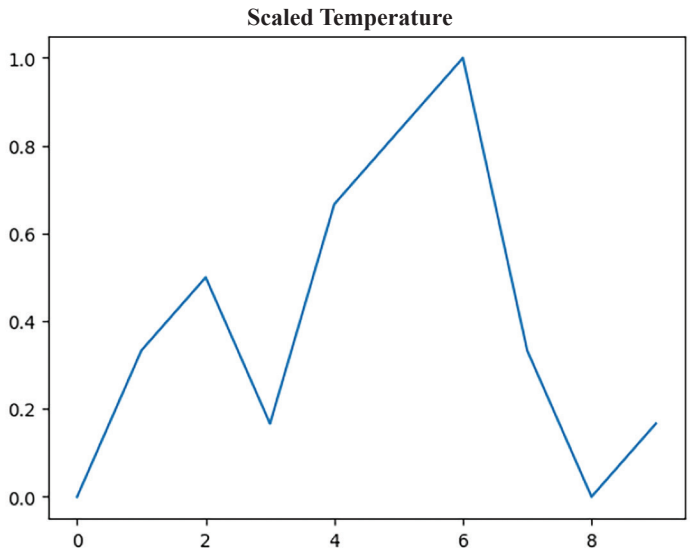
```
(1010, 1014)
```

2. The minimum becomes 0. The maximum becomes 1. All other values fall between 0 and 1.

3. Plotting original vs scaled Temperature

```
plt.plot(df["Temperature"])
plt.title("Original Temperature")
plt.show()

plt.plot(df["Temperature_scaled"])
plt.title("Scaled Temperature")
plt.show()
```





The shape remains the same. However, observe the y-axis, the scales are different.

4. Scaling the Humidity column

```
hum_min = df["Humidity"].min()
hum_max = df["Humidity"].max()

df["Humidity_scaled"] = (df["Humidity"] - hum_min) / (hum_max - hum_min)
df[["Humidity", "Humidity_scaled"]]
```

Humidity	Humidity_scaled	
0	60	0.142857
1	62	0.285714
2	58	0.000000
3	65	0.500000
4	70	0.857143
5	72	1.000000
6	68	0.714286
7	64	0.428571
8	59	0.071429
9	61	0.214286

Scaling is helpful because it removes the effect of large numbers overpowering small numbers. This allows different quantities to be compared fairly in the same graph.