Internet of Things

Data Analysis Using Raspberry Pi

Weather Parameters Analysis



Project’s Working and Explanation

Project’s Contributor: Abdullah Imran

# 

[**1. What is Data Analysis?**](#_y6hldz5c9v5b) **4**

[1.1: How data analysis is done?](#_bbp66lg2ifaz) 4

[**2. What is Rasberry Pi?**](#_nwascc3lttdi) **5**

[2.1: Why use Rasberry Pi for data analysis?](#_unu6isc024cw) 6

[**3. Motive Behind the Project:**](#_8yzok5kynpvi) **6**

[**4. What was involved in this project?**](#_67ad72t57u0k) **7**

[4.1: BME 280:](#_dqkjz4133rik) 7

[4.2: I2C:](#_2p0qj3yfc94v) 8

[4.3: Use of Java:](#_5c0zoonf8nd3) 8

[4.4: Use of Kotlin:](#_v5t8ilj42ed4) 9

[**5. Project Phase 1: Recording Data**](#_e7ece9quy10y) **9**

[5.1: Materials Required:](#_ayxkjj7kbv6j) 9

[5.2: Connections & Setup:](#_gzx4kyg6nf91) 11

[5.3: Coding](#_5e8783opb3ih) 12

[5.3.1: Making it global:](#_h6dqefj56qrr) 13

[5.3.2: Recording the data into a spreadsheet:](#_9d20puy0ir3n) 15

[**6. Project Phase 2: Data Analysis:**](#_k5l99sebb15l) **19**

[6.1: Numerical Data Analysis](#_k58kzjrasgme) 19

[6.2: Simple Graphs:](#_zd80rsffqcu7) 22

[6.2.1: Weather graph w.r.t time:](#_s4b3fdlgevbv) 22

[6.2.2: Changes in temperature w.r.t humidity & pressure:](#_a3n7jv70lbbg) 23

[6.2.3: Simple Heatmap and Violin Graphs:](#_ljiyset8oj8g) 23

[6.2.4: Temp vs Pressure + Temp vs Humidity Scatter Graph:](#_8ggsi29s5tds) 24

[6.3: Complex Graphs:](#_oz0u7gqofdnq) 25

[6.3.1: Temperature peak and low graph:](#_wlxdwud5uwyp) 25

[6.3.2: Comparative Analysis Graph:](#_7lt1v67n193q) 26

[6.3.3: K-Means Data Clustering:](#_8klma3hmhgfw) 26

[**7. Project Phase 3: Simple Data Prediction:**](#_1h9up21brnvt) **27**

[**8. Project Phase 4: Implementation**](#_5bde7u8jz4r3) **30**

[8.1: Introduction:](#_xc41nd26ahjs) 30

[8.2: Features:](#_wty213wb9xlb) 30

[8.2.1. Hourly Forecast:](#_zflcyfaylrk8) 30

[8.2.2. Current Weather:](#_h33jlh4wi4q) 31

[8.3: Uses:](#_vsrlf6y92cdr) 31

[8.3.1: Daily Planning:](#_e19mhkrof2ax) 31

[8.3.2: Outdoor Activities:](#_ps79niy61blx) 31

[8.3.3: Travel Planning:](#_xn9y35i21byz) 31

[8.4: External API:](#_6yhrrlrd6aex) 31

[8.5: User Interface (UI):](#_psmoumny5drq) 31

[8.5.1. Design Principles:](#_wpt975sr3go4) 31

[8.5.2. UI Components:](#_eyt58m2yoxck) 31

[8.5.3. Theming:](#_w5864pboyooj) 32

[8.6: Dependencies:](#_ibdljmsj2clo) 32

[8.6.1: Dagger-Hilt:](#_5g5jjf39s4po) 32

[8.6.2: Retrofit:](#_2mppjc8rg9jz) 32

[8.6.3: Google Play Services Location API:](#_1999hprzih9p) 32

[8.6.4: Compose:](#_fzlaj248f3cp) 32

[8.6.5: Development Environment:](#_5n6a8e3evvgl) 32

[8.7: Design:](#_dwpyd8m2wvgj) 33

[**9. Learning Outcomes:**](#_mfbb82as14gi) **34**

[9.1. Insights into Feature Importance:](#_bl3qg8e9pysf) 34

[9.2. Practical Data Collection:](#_r6xxn2igmc8d) 34

[9.3. Proficient Data Analysis:](#_z7qthibavdpd) 34

[9.4. Comprehension of Model Metrics:](#_tplh6hi311kr) 34

[9.5. Tuning Model Hyperparameters:](#_mku22447w72l) 34

[9.6. Application of Standardization Techniques:](#_nll1ejp7fq9u) 34

[9.7. User Interaction Implementation:](#_b2mkajfewbcd) 35

[9.8. Refinement through Problem-Solving:](#_p63jotqpo7ub) 35

[9.9. Interdisciplinary Insight:](#_ozer6zk9c8ay) 35

[9.10. Critical Thinking in Model Deployment:](#_azq3ro4x3iqt) 35

[**10. Conclusion:**](#_s65d8om2clc3) **35**

# 

# 1. What is Data Analysis?

Data analysis is the process of inspecting, cleaning, transforming, and modeling data to discover useful information, draw conclusions, and support decision-making. It involves using various techniques and tools to analyze raw data and extract meaningful insights. Data analysis aims to uncover patterns, trends, relationships, and other valuable information that can be used to make informed decisions. This analysis of the data can then further be used for predicting the outcome of the data, making better decisions based on the provided information, and even just being informed about it.

## 1.1: How data analysis is done?

* **Data Collection:** Gathering relevant and accurate data from various sources and then recording it into some digital form.
* **Data Cleaning:** Identifying and correcting errors, inconsistencies, and missing values in the dataset.
* **Data Exploration:** Understanding the structure and characteristics of the data through summary statistics, visualizations, and other exploratory techniques.
* **Data Transformation:** Preparing the data for analysis by transforming it into a suitable format, which may involve normalization, scaling, or encoding.
* **Data Modeling:** Applying statistical and machine learning techniques to build models that can help uncover patterns or make predictions.
* **Data Interpretation:** Drawing meaningful conclusions from the analysis and translating them into actionable insights.
* **Visualization:** Creating visual representations of the data to facilitate easier understanding and communication of findings.

# 2. What is Rasberry Pi?

Raspberry Pi is a series of small, affordable, single-board computers developed by the Raspberry Pi Foundation. These credit card-sized computers are equipped with various input/output ports, general-purpose input/output (GPIO) pins, and connectivity options. The primary goal of the Raspberry Pi Foundation is to promote computer science education and make computing accessible to people around the world.

Here are some key features of Raspberry Pi:

* **Affordability:** Raspberry Pi boards are cost-effective, making them accessible to a wide range of users, including students, hobbyists, and professionals.
* **Versatility:** Raspberry Pi devices are versatile and can be used for various applications, including educational projects, DIY electronics, home automation, media centers, and more.
* **Compact Size:** The small form factor of Raspberry Pi makes it easy to integrate into various projects and applications with limited space.
* **Community Support:** The Raspberry Pi community is large and active, providing a wealth of resources, tutorials, and community-driven support.

## 2.1: Why use Rasberry Pi for data analysis?

We can use Rasberry Pi for several reasons:

* **Cost-Effective Solution:** Raspberry Pi provides a budget-friendly option for those interested in exploring data analysis without the need for more expensive hardware. It is particularly suitable for educational purposes and smaller-scale projects.
* **Skill Development:** Utilizing Raspberry Pi for data analysis enables individuals to cultivate skills in programming, data manipulation, and analysis within a compact and accessible computing environment.
* **Integration with IoT Devices:** Raspberry Pi easily integrates with various sensors and devices, making it well-suited for projects involving data collection from Internet of Things (IoT) devices.
* **Portability:** The small size of Raspberry Pi allows for portability, facilitating on-the-go data analysis in situations where a larger computer may be impractical.
* **Customization:** Users can tailor their Raspberry Pi setups to specific data analysis needs, combining the computing power of the device with peripherals and sensors as required.

# 3. Motive Behind the Project:

There was a pool of options available to me to do my final project on the Internet of Things, but I specifically chose this project for several reasons:

* This project was different from what the other people were doing and influenced me to do things as I had already done some minor projects on Rasberry Pi, so doing this project on the same device would have enabled me to learn more about the device itself.
* Raspberry Pi can be easily used for intergenerational purposes, such as when we need to have sensors connected to it so that we can analyze its data.
* Raspberry Pi mostly uses Python as a coding language to execute instructions which I happened to know somewhat. So, doing this project could also mean learning more about the language itself.
* Raspberry Pi is not an extremely hard platform to get familiar with, and hence that would mean that I could accomplish more at the same time.
* Data Analysis is also one of those areas that I am generally interested in, and was not able to find the opportunity to do such kind of project until I saw this topic.
* Data Analysis also involves having to develop some machine learning-based models for prediction, which is also one of the areas that I am interested in, but have not really gotten familiar with before doing this project.
* Data Analysis also involves some of the mathematical concepts that are related to statistics, and I had not found such a project that I could do which would also involve some of these concepts until I found out about this project.
* Within data analysis, weather is something that we always hear about but really don’t see the chemistry behind it. So doing this project would also mean getting familiar with some of the weather parameters.

# 4. What was involved in this project?

Tools/software/languages used in this project:

* Raspberry Pi
* BME 280
* I2C
* Excel/spreadsheet
* Python
* Java
* Kotlin
* VS Code
* Android Studio

## 4.1: BME 280:

The BME280 is a sensor module that combines three environmental sensing capabilities into a single package. The three main functions of the BME280 are:

* **Barometric Pressure Measurement (B):** The sensor measures atmospheric pressure, providing data that can be used to determine altitude or predict changes in the weather. Barometric pressure is often used in weather forecasting and altitude estimation.
* **Relative Humidity Measurement (H):** BME280 can measure the relative humidity of the environment. Humidity is the amount of moisture present in the air, and this data is valuable in various applications, such as climate control systems and weather monitoring.
* **Temperature Measurement (T):** The BME280 also includes a temperature sensor, providing accurate temperature readings. Temperature sensing is essential in numerous applications, including climate control, industrial processes, and scientific research.

## 4.2: I2C:

The BME280 has two possible I2C addresses: 0x76 and 0x77. This address depends on the connection of the sensor's address pin (SDO) to either VCC (0x77) or GND (0x76). The default address is often 0x76.

**Connection to Raspberry Pi:**

Connect the SDA (data line) and SCL (clock line) pins of the BME280 to the corresponding GPIO pins on the Raspberry Pi. The specific GPIO pins used may depend on your Raspberry Pi model, but commonly, GPIO2 is used for SDA, and GPIO3 is used for SCL.

**Power Supply:**

Ensure that the BME280 is appropriately powered. Typically, it operates at 3.3V, so connect its VCC pin to the 3.3V output on the Raspberry Pi. Connect the GND to the ground (GND) pin on the Pi.

**I2C Communication:**

To interact with the BME280 over I2C using Python on a Raspberry Pi, you can use libraries like smbus/smbus2 or specialized BME280 libraries.

## 4.3: Use of Java:

In a Kotlin-based Android Studio app, Java serves various purposes, primarily due to the interoperability between the two languages. Java code can seamlessly coexist with Kotlin, enabling developers to leverage existing Java libraries, frameworks, and SDKs within their Kotlin projects. This interoperability is especially valuable during gradual migrations, allowing teams to integrate legacy code, collaborate effectively, and utilize the extensive resources available for both languages. Additionally, the Android Studio IDE provides robust support for mixed-language projects, facilitating smooth development workflows and making it easier for developers with diverse language preferences to contribute to the project. Overall, the use of Java in a Kotlin-based Android app enhances flexibility, compatibility, and the ability to tap into the rich ecosystem of Java-based tools and resources.

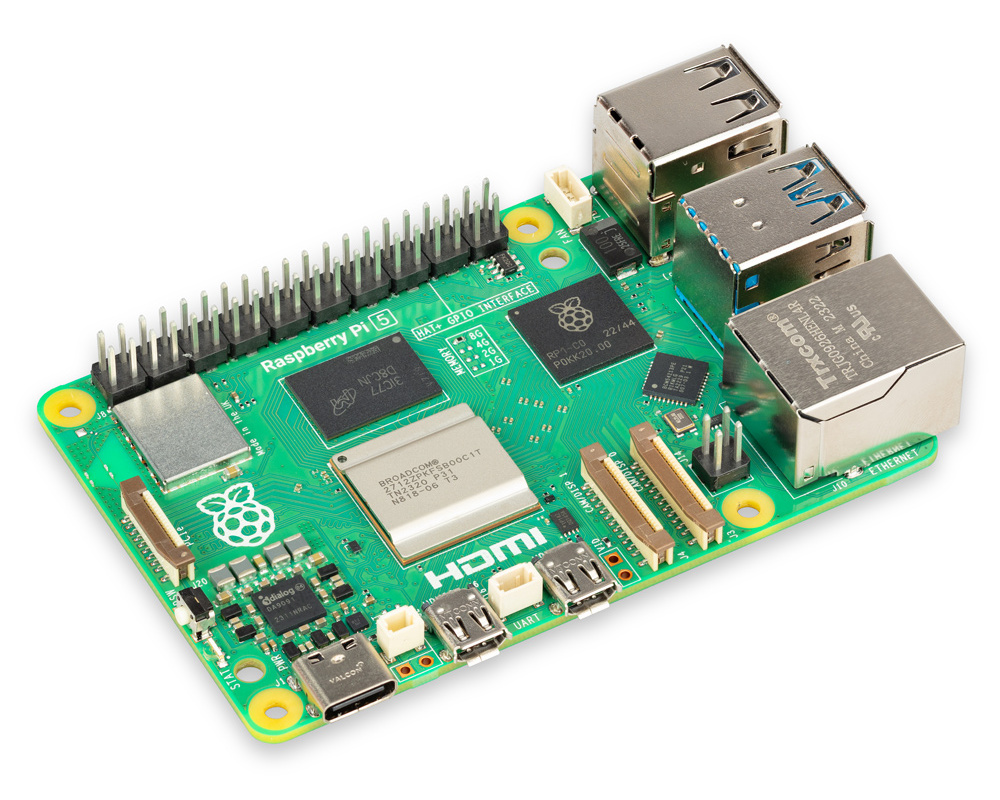
## 4.4: Use of Kotlin:

Kotlin is a versatile programming language that offers a modern and concise syntax, making it an excellent choice for building and designing Android apps. Its interoperability with Java ensures a smooth transition for developers, allowing them to incorporate existing Java code and libraries seamlessly. Kotlin's null safety, extension functions, and expressive syntax contribute to cleaner and more robust code. Features like data classes simplify the creation of data models, while Kotlin's concise syntax reduces boilerplate code, enhancing developer productivity. The language supports functional programming paradigms, offering higher-order functions and lambda expressions, promoting concise and expressive logic. Additionally, Kotlin's coroutine support aids in managing asynchronous tasks efficiently. Its compatibility with Android Studio and comprehensive tooling make Kotlin an ideal language for crafting modern, feature-rich Android applications with improved readability, maintainability, and overall development experience.

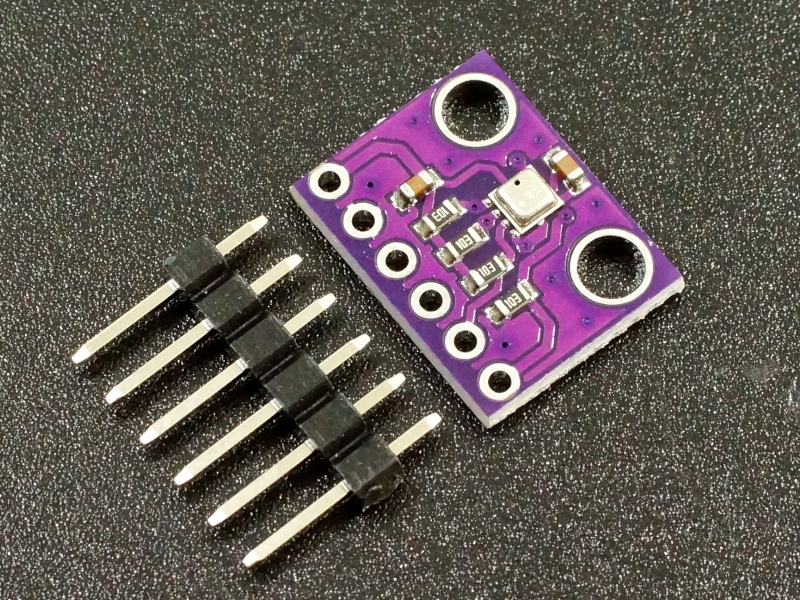
# 5. Project Phase 1: Recording Data

## 5.1: Materials Required:

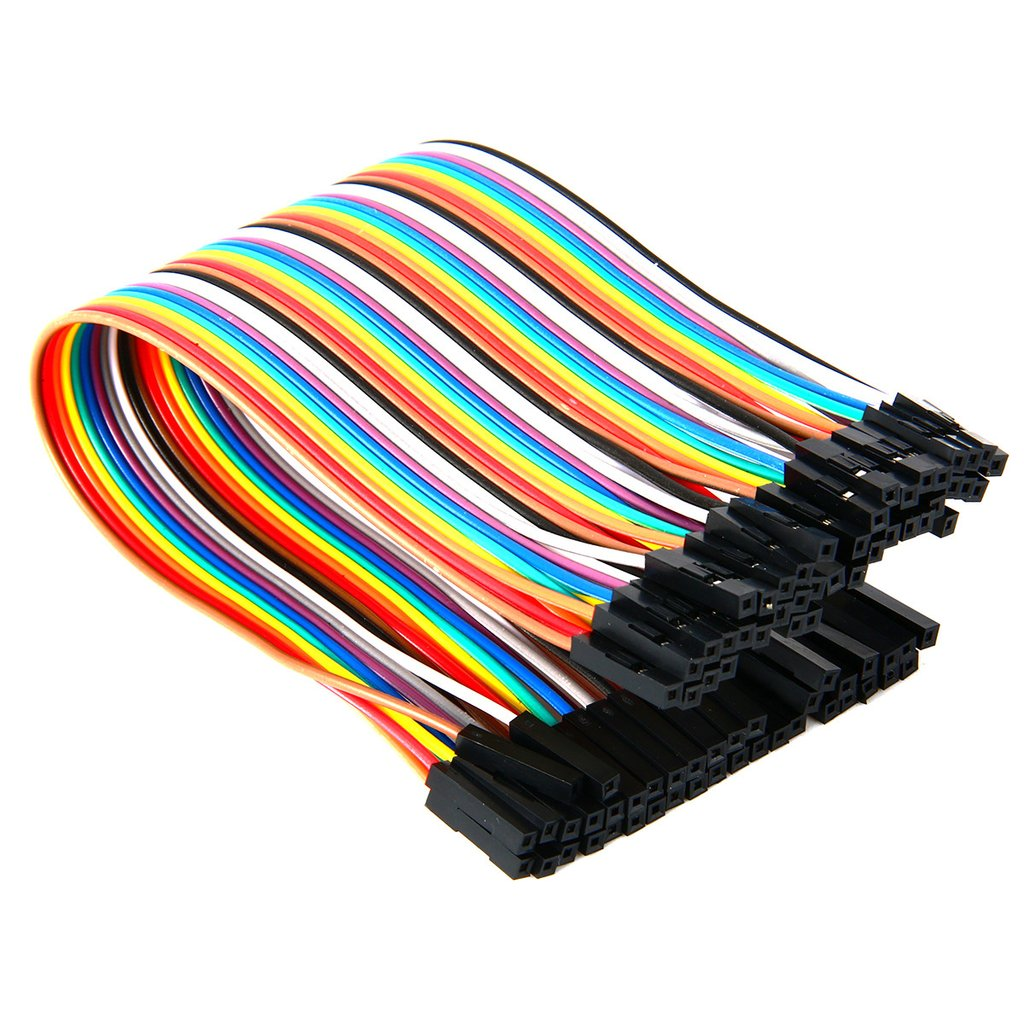
**Raspberry Pi:**



**BME 280:**



**Jumper Wires:**



## 5.2: Connections & Setup:

**Step 1: Gather Hardware and Software**

Hardware:

Raspberry Pi (any model with GPIO pins)

BME280 sensor

Jumper wires

Software:

Raspbian OS installed on your Raspberry Pi

Access to the terminal or command prompt on your Raspberry Pi

**Step 2: Power Off the Raspberry Pi**

Ensure that your Raspberry Pi is powered off before connecting any hardware components.

**Step 3: Identify GPIO Pins**

Identify the GPIO (General Purpose Input/Output) pins on the Raspberry Pi. We'll need to connect the BME280 sensor to these pins. Refer to our specific Raspberry Pi model's GPIO pinout diagram.

**Step 4: Connect BME280 to Raspberry Pi**

Use jumper wires to connect the BME280 sensor to the Raspberry Pi. The BME280 typically has four pins: VCC, GND, SDA (data), and SCL (clock). Connect them as follows:

Connect the VCC pin to a 3.3V GPIO pin on the Raspberry Pi.

Connect the GND pin to a ground (GND) GPIO pin on the Raspberry Pi.

Connect the SDA pin to the GPIO 2 (SDA) pin on the Raspberry Pi.

Connect the SCL pin to the GPIO 3 (SCL) pin on the Raspberry Pi.

Step 5: Power On the Raspberry Pi

After making the connections, power on the Raspberry Pi.

**Step 6: Enable I2C Interface:**

If you're using the I2C interface to communicate with the BME280 (which is common), you may need to enable it. Open the Raspberry Pi Configuration tool by running the commands or just simply enabling it via settings.

**Step 7: Enable SSH:**

We can also make things global by enabling SSH and then accessing the data from anywhere within the terminal.

## 5.3: Coding

Head to the built-in Python IDE, and start writing the code. I wrote the following code:

# This code was tried on Raspberry Pi:

# This code was giving slightly different results, hence it was modified in another file

#!usr/bin/env python

import time

try:

from smbus2 import SMBus

except ImportError:

from smbus import SMBus

from bme280 import BME280

print("""all-values.py - Read temperature, pressure, and humidity

Press Ctrl+C to exit!

""")

# Initialize the BME280:

bus = SMBus(1)

bme280 = BME280(i2c\_dev=bus)

while True:

temperature = bme280.get\_temperature()

pressure = bme280.get\_pressure()

humidity = bme280.get\_humidity()

print('{:05.2f}\*C {:05.2f}hPa {:05.2f}%'.format(temperature, pressure, humidity))

time.sleep(1)

As soon as you execute this code, you will the data of temperature, humidity, and pressure after each second.

### 5.3.1: Making it global:

However, there was one issue the temperature was wrong. So, I installed longer jumper wires with the sensor to keep it away from the heat of the computer and then it started to report the correct temperature and other values.

Use PuTTy to make the connection global.



Now let us try a simple version of the code by executing the file on the terminal outside the file’s own settings:

#!/usr/bin/env python

#This code was run on Raspberry Pi, gives the readings of temperature, pressure and humidity on raspberry pi's terminal.

# Import modules for time and to access sensor

import time

from smbus2 import SMBus

from bme280 import BME280

# Initialise the BME280

bus = SMBus(1)

bme280 = BME280(i2c\_dev=bus)

# Get data and discard to avoid garbage first reading

temperature = bme280.get\_temperature()

pressure = bme280.get\_pressure()

humidity = bme280.get\_humidity()

time.sleep(1)

while True:

temperature = bme280.get\_temperature()

pressure = bme280.get\_pressure()

humidity = bme280.get\_humidity()

print('{:05.2f}\*C {:05.2f}hPa {:05.2f}%'.format(temperature, pressure, humidity))

time.sleep(1)

It will give us the same data and upon investigation, it was accurate.

### 5.3.2: Recording the data into a spreadsheet:

Since we have analyzed the sensor was working perfectly, we can now record it into a spreadsheet:

# This code was run on raspberry pi with few tweaks that collected the actual temperature data from the BME280 sensor and saved it in the workbook called "weather".

#!/usr/bin/env python

# Import all libraries we need!

from smbus2 import SMBus

from bme280 import BME280

import time

import datetime

from datetime import date

from openpyxl import load\_workbook

# Initialise the BME280

bus = SMBus(1)

bme280 = BME280(i2c\_dev=bus)

# Take first reading and discard it to avoid garbage first row

temperature = bme280.get\_temperature()

pressure = bme280.get\_pressure()

humidity = bme280.get\_humidity()

time.sleep(1)

# Load the workbook and select the sheet

wb = load\_workbook('/home/pi/Python\_Code/weather.xlsx')

sheet = wb['weather\_data']

try:

while True:

# Read the sensor and get date and time

temperature = round(bme280.get\_temperature(),1)

pressure = round(bme280.get\_pressure(),1)

humidity = round(bme280.get\_humidity(),1)

today = date.today()

now = datetime.datetime.now().time()

# Inform the user!

print('Adding this data to the spreadsheet:')

print(today)

print(now)

print('{}\*C {}hPa {}%'.format(temperature, pressure, humidity))

# Append data to the spreadsheet

row = (today, now, temperature, pressure, humidity)

sheet.append(row)

#Save the workbook

wb.save('/home/pi/Python\_Code/weather.xlsx')

# Wait for 2 minutes seconds (120 seconds)

time.sleep(120)

finally:

# Make sure the workbook is saved!

wb.save('/home/pi/Python\_Code/weather.xlsx')

print('Goodbye!')

This is how our spreadsheet will look like:

| **Date** | **Time** | **Temp (C)** | **Pressure (hPa)** | **Humidity (%age)** |
| --- | --- | --- | --- | --- |
| 2024-01-31 | 2:09:44 | 23.9 | 1014.5 | 39.4 |
| 2024-01-31 | 2:10:44 | 23.6 | 1014.3 | 39.2 |
| 2024-01-31 | 2:11:44 | 23.2 | 1014.6 | 39.5 |
| 2024-01-31 | 2:12:44 | 23.1 | 1014.2 | 39.1 |
| 2024-01-31 | 2:13:44 | 22.1 | 1014.4 | 39.3 |
| 2024-01-31 | 2:14:44 | 23.2 | 1014.5 | 39.5 |
| 2024-01-31 | 2:15:44 | 23.3 | 1014.7 | 39.2 |
| 2024-01-31 | 2:16:44 | 23.4 | 1015.4 | 39.4 |
| 2024-01-31 | 2:17:44 | 23.5 | 1014.6 | 40.5 |
| 2024-01-31 | 2:18:44 | 23.4 | 1014.3 | 39.3 |
| 2024-01-31 | 2:19:44 | 23.3 | 1014.5 | 40.1 |
| 2024-01-31 | 2:20:44 | 23.2 | 1014.8 | 39.2 |
| 2024-01-31 | 2:21:44 | 23.4 | 1012.6 | 39.4 |
| 2024-01-31 | 2:22:44 | 23.5 | 1014.3 | 39.1 |
| 2024-01-31 | 2:23:44 | 23.3 | 1014.5 | 39.3 |
| 2024-01-31 | 2:24:44 | 23.2 | 1014.7 | 39.5 |
| 2024-01-31 | 2:25:44 | 23.1 | 1014.4 | 38.3 |
| 2024-01-31 | 2:26:44 | 23.7 | 1014.6 | 39.4 |
| 2024-01-31 | 2:27:44 | 23.2 | 1014.8 | 39.1 |
| 2024-01-31 | 2:28:44 | 23.4 | 1014.5 | 39.3 |
| 2024-01-31 | 2:29:44 | 23.5 | 1014.3 | 40.1 |
| 2024-01-31 | 2:30:44 | 23.3 | 1010.4 | 39.2 |
| 2024-01-31 | 2:31:44 | 23.1 | 1014.7 | 39.4 |
| 2024-01-31 | 2:32:44 | 23.2 | 1014.5 | 39.7 |
| 2024-01-31 | 2:33:44 | 23.2 | 1014.3 | 39.3 |
| 2024-01-31 | 2:34:44 | 23.4 | 1014.6 | 39.5 |
| 2024-01-31 | 2:35:44 | 23.5 | 1014.8 | 39.2 |
| 2024-01-31 | 2:36:44 | 23.3 | 1014.4 | 41.2 |
| 2024-01-31 | 2:37:44 | 23.1 | 1013.7 | 39.1 |
| 2024-01-31 | 2:38:44 | 23.5 | 1014.5 | 39.3 |
| 2024-01-31 | 2:39:44 | 23.2 | 1014.3 | 39.5 |
| 2024-01-31 | 2:40:44 | 23.3 | 1014.6 | 39.2 |
| 2024-01-31 | 2:41:44 | 23.4 | 1014.8 | 39.4 |
| 2024-01-31 | 2:42:44 | 23.5 | 1014.5 | 39.1 |
| 2024-01-31 | 2:43:44 | 23.3 | 1014.6 | 39.3 |
| 2024-01-31 | 2:44:44 | 23.2 | 1014.3 | 39.5 |
| 2024-01-31 | 2:45:44 | 23.1 | 1014.5 | 39.2 |
| 2024-01-31 | 2:46:44 | 23.4 | 1014.7 | 39.4 |
| 2024-01-31 | 2:47:44 | 23.2 | 1014.4 | 39.1 |
| 2024-01-31 | 2:48:44 | 23.4 | 1012.6 | 39.3 |
| 2024-01-31 | 2:49:44 | 23.5 | 1014.3 | 39.5 |
| 2024-01-31 | 2:50:44 | 23.3 | 1014.5 | 39.2 |
| 2024-01-31 | 2:51:44 | 23.1 | 1014.7 | 39.4 |
| 2024-01-31 | 2:52:44 | 23.7 | 1014.4 | 39.1 |
| 2024-01-31 | 2:53:44 | 23.2 | 1014.6 | 39.3 |
| 2024-01-31 | 2:54:44 | 23.4 | 1014.3 | 39.5 |
| 2024-01-31 | 2:55:44 | 23.5 | 1013.5 | 39.2 |
| 2024-01-31 | 2:56:44 | 23.3 | 1014.7 | 39.4 |
| 2024-01-31 | 2:57:44 | 23.2 | 1014.4 | 39.1 |
| 2024-01-31 | 2:58:44 | 23.1 | 1014.6 | 39.3 |
| 2024-01-31 | 2:59:44 | 23.1 | 1014.3 | 39.5 |
| 2024-01-31 | 3:00:44 | 23.2 | 1014.3 | 39.2 |
| 2024-01-31 | 3:01:44 | 23.4 | 1014.4 | 39.4 |
| 2024-01-31 | 3:02:44 | 23.5 | 1014.4 | 39.1 |
| 2024-01-31 | 3:03:44 | 23.3 | 1014.6 | 39.3 |
| 2024-01-31 | 3:04:44 | 23.1 | 1014.3 | 39.5 |
| 2024-01-31 | 3:05:44 | 23.2 | 1014.5 | 39.2 |
| 2024-01-31 | 3:06:44 | 23.2 | 1014.7 | 39.4 |
| 2024-01-31 | 3:07:44 | 23.4 | 1014.4 | 39.1 |
| 2024-01-31 | 3:08:44 | 23.5 | 1014.6 | 39.3 |
| 2024-01-31 | 3:09:44 | 23.3 | 1014.3 | 39.5 |
| 2024-01-31 | 3:10:44 | 23.2 | 1014.5 | 39.2 |
| 2024-01-31 | 3:11:44 | 23.1 | 1014.7 | 39.4 |
| 2024-01-31 | 3:12:44 | 23.4 | 1014.4 | 39.1 |
| 2024-01-31 | 3:13:44 | 23.2 | 1014.6 | 39.3 |
| 2024-01-31 | 3:14:44 | 23.4 | 1014.3 | 39.5 |
| 2024-01-31 | 3:15:44 | 23.5 | 1014.5 | 39.2 |
| 2024-01-31 | 3:16:44 | 23.3 | 1015.3 | 39.4 |
| 2024-01-31 | 3:17:44 | 23.1 | 1015.1 | 39.1 |
| 2024-01-31 | 3:18:44 | 23.5 | 1014.8 | 39.3 |

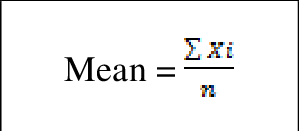
Above data was recorded thrice, and the data that is included was the final version of it once it was confirmed that there were no signs of data inconsistency.

# 6. Project Phase 2: Data Analysis:

## 6.1: Numerical Data Analysis

First, I just simply did some calculations for data analysis.

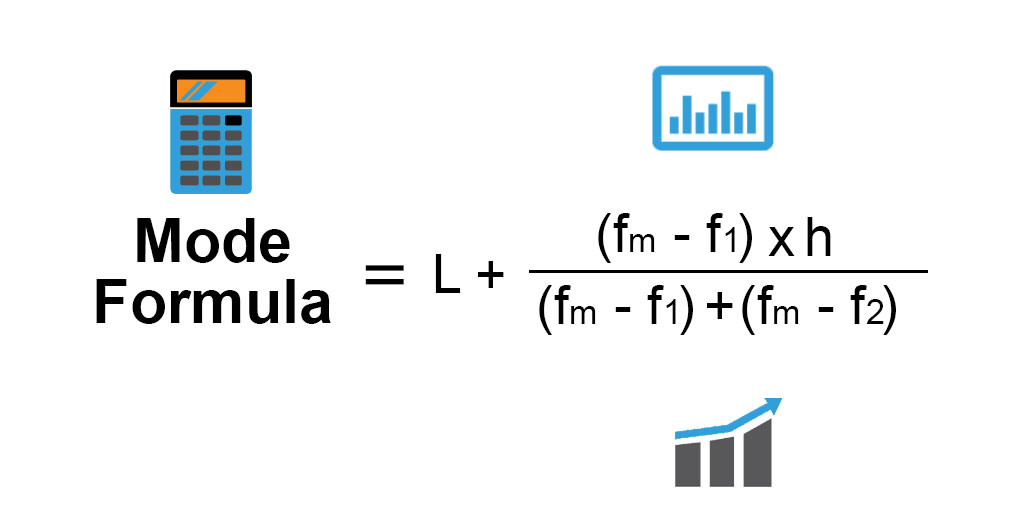
1. Mean:



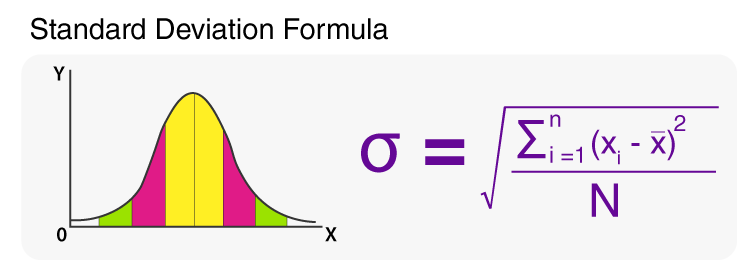
1. Median:



1. Mode:



1. Standard deviation:



Here is the code and its results:

import pandas as pd

# Load the data from the Excel file

df = pd.read\_excel('weather.xlsx', sheet\_name='weather\_data')

# Exclude datetime columns

numerical\_columns = df.select\_dtypes(include=['number']).columns

df\_numeric = df[numerical\_columns]

# Calculate mean, median, mode, etc.

mean\_values = df\_numeric.mean()

median\_values = df\_numeric.median()

mode\_values = df\_numeric.mode().iloc[0]

std\_deviation = df\_numeric.std()

# Display the results

print("Mean values:")

print(mean\_values)

print("\nMedian values:")

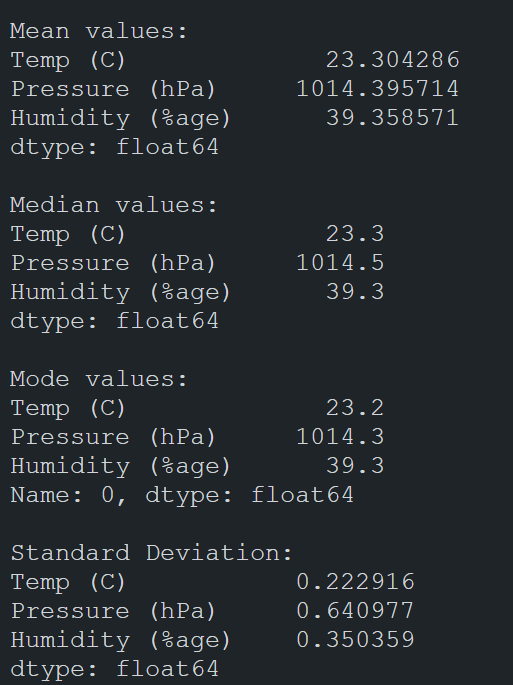
print(median\_values)

print("\nMode values:")

print(mode\_values)

print("\nStandard Deviation:")

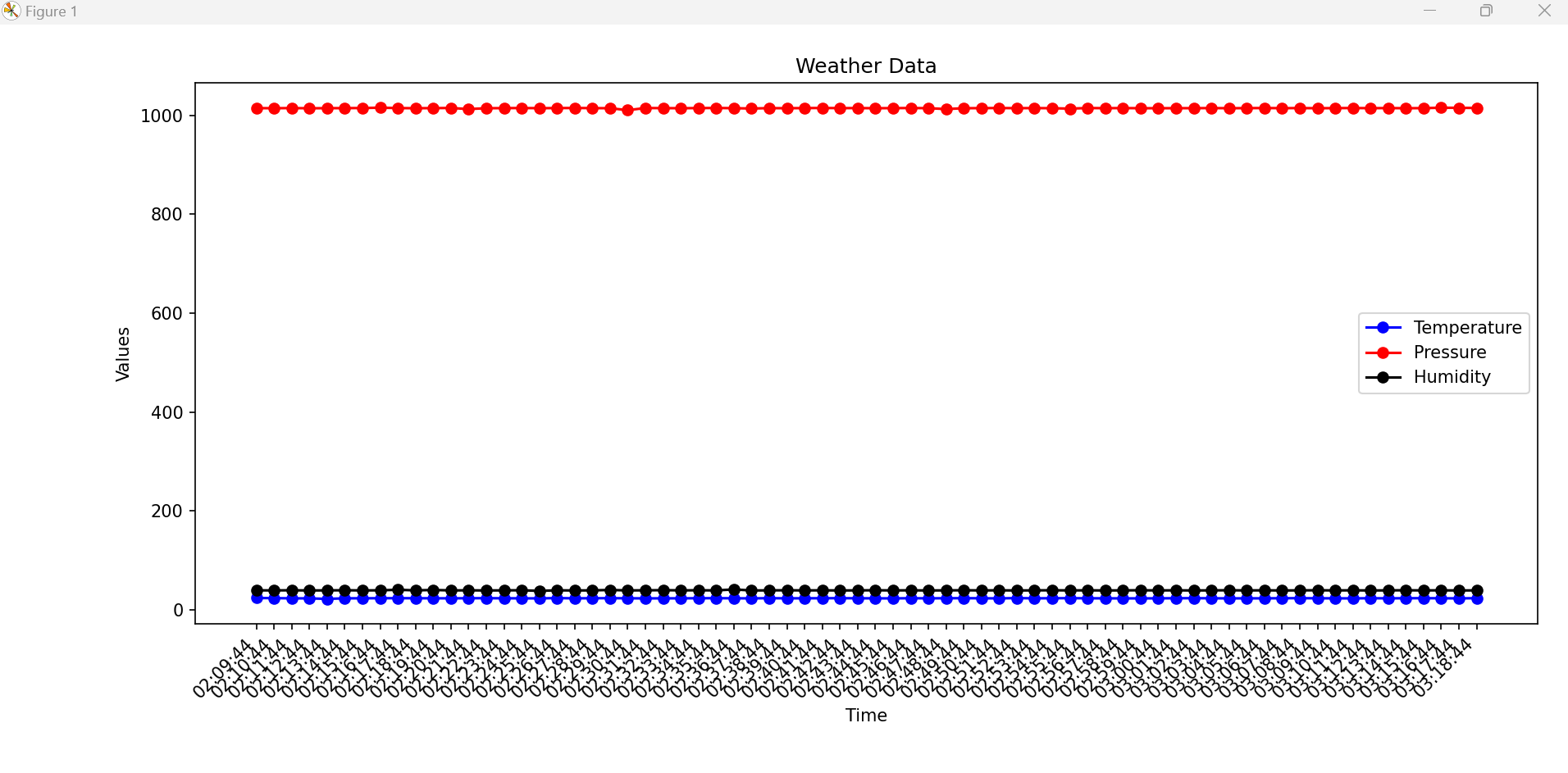
print(std\_deviation)



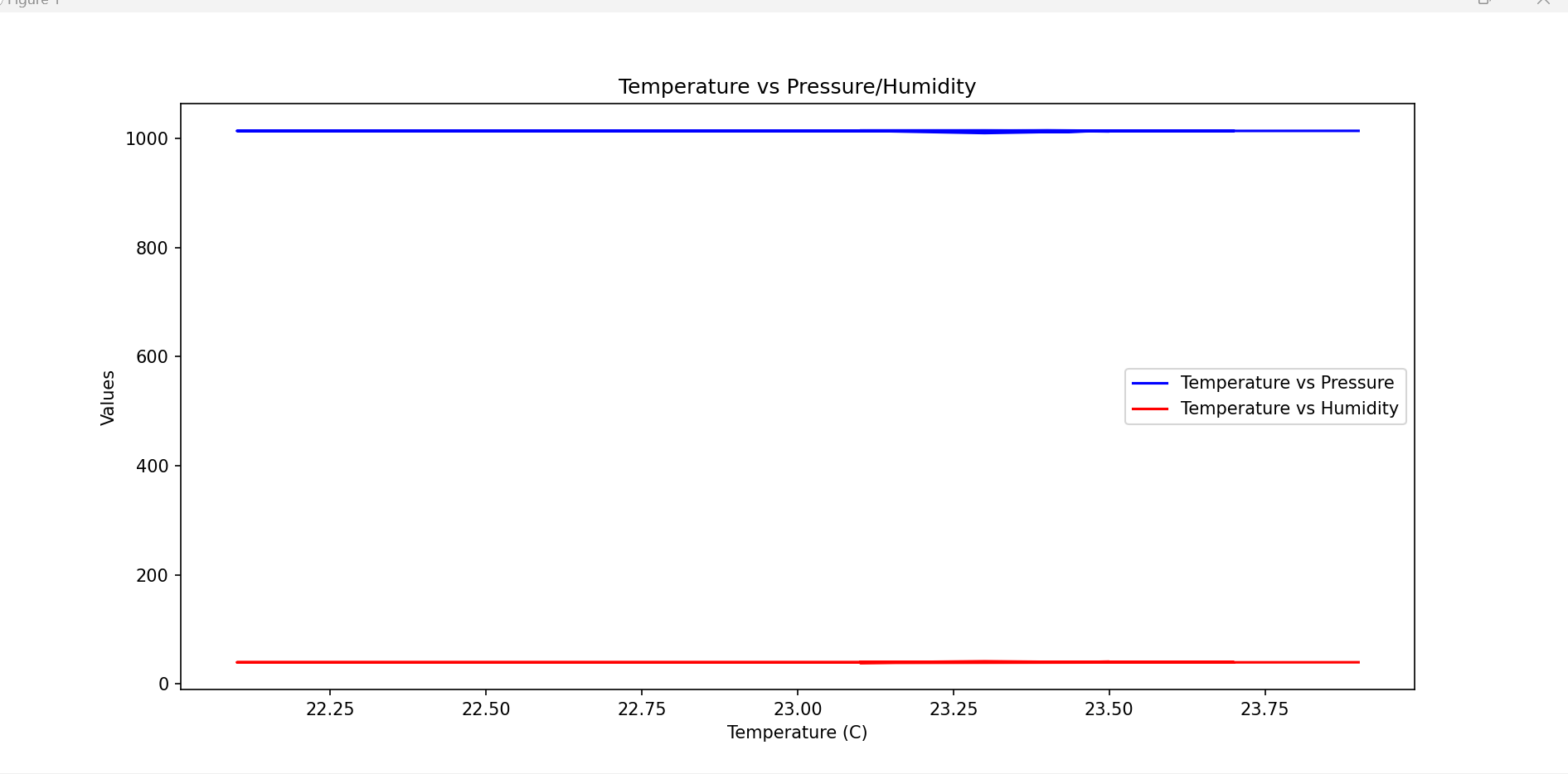
## 6.2: Simple Graphs:

Here are some simple graphs that give us a generic view into our data:

### 6.2.1: Weather graph w.r.t time:

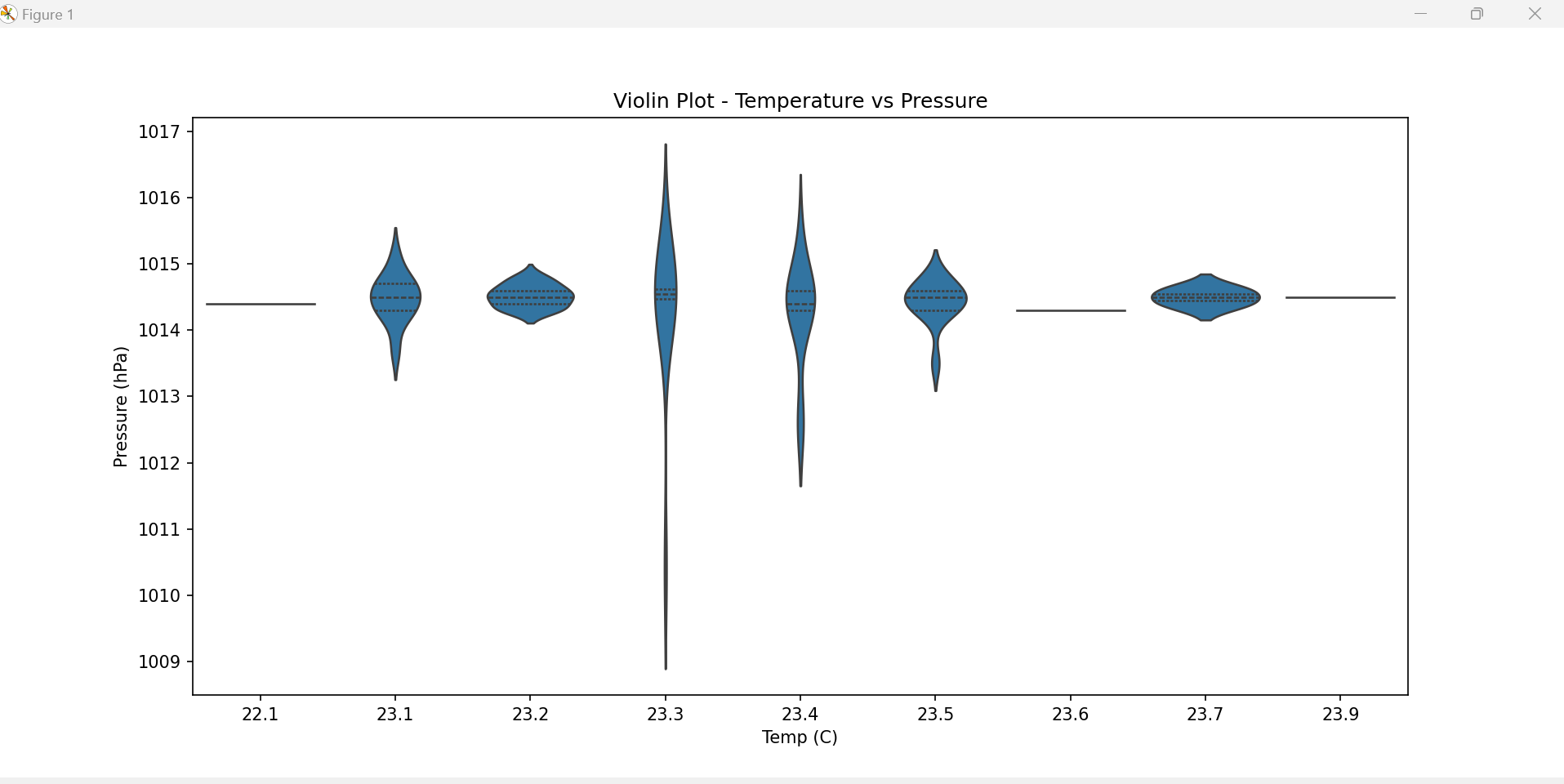


### 6.2.2: Changes in temperature w.r.t humidity & pressure:

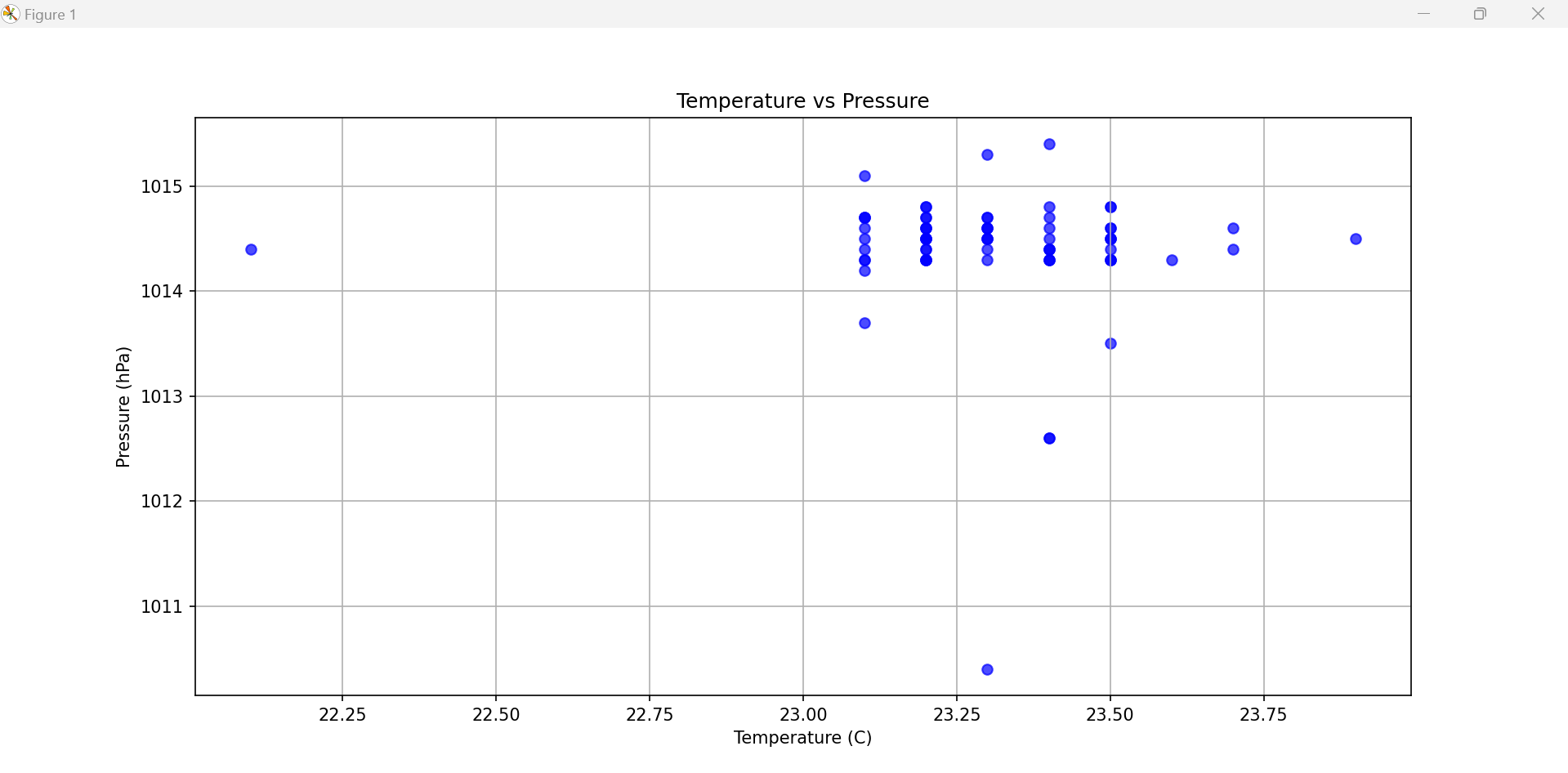


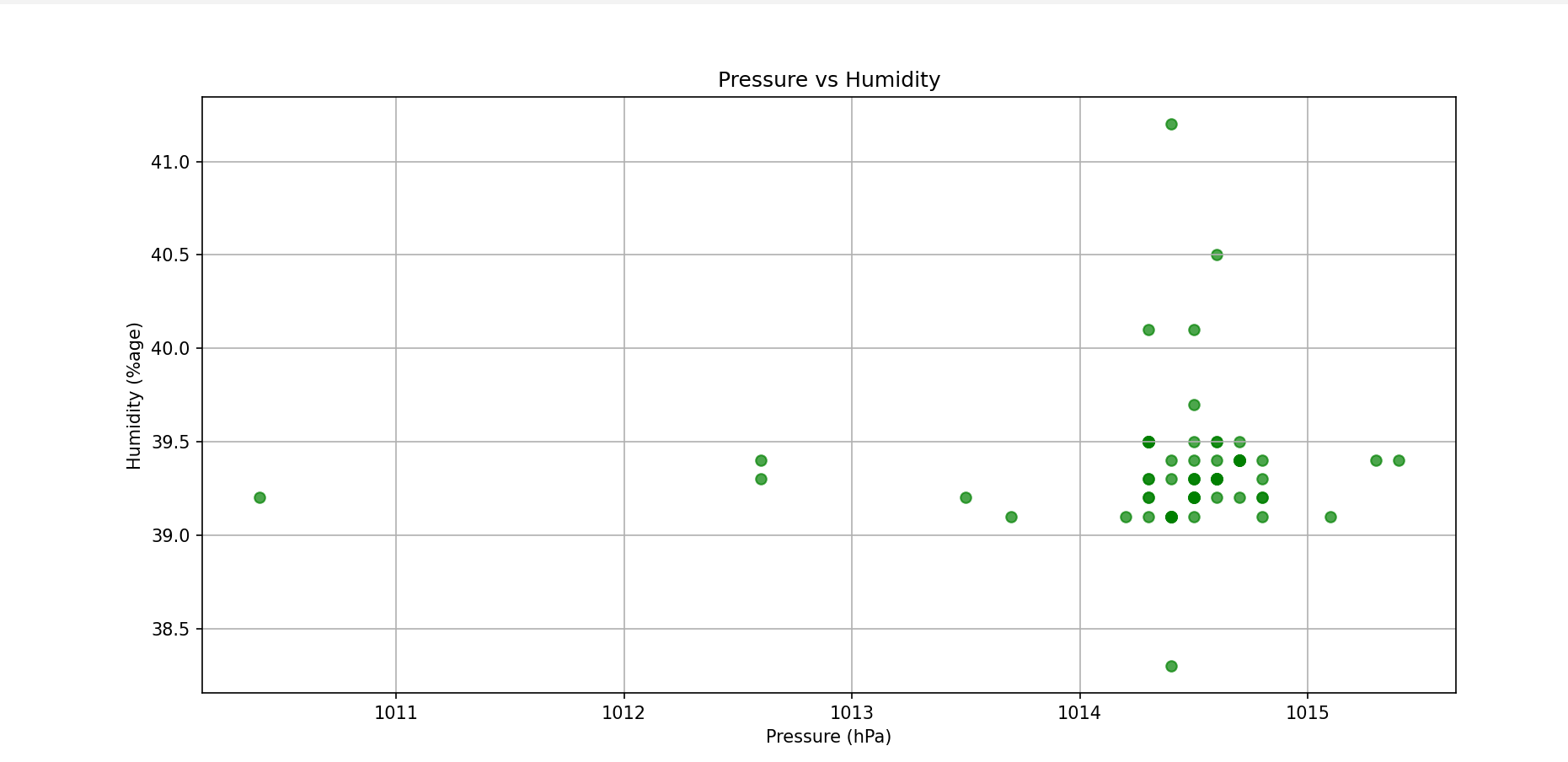
### 6.2.3: Simple Heatmap and violin graphs:





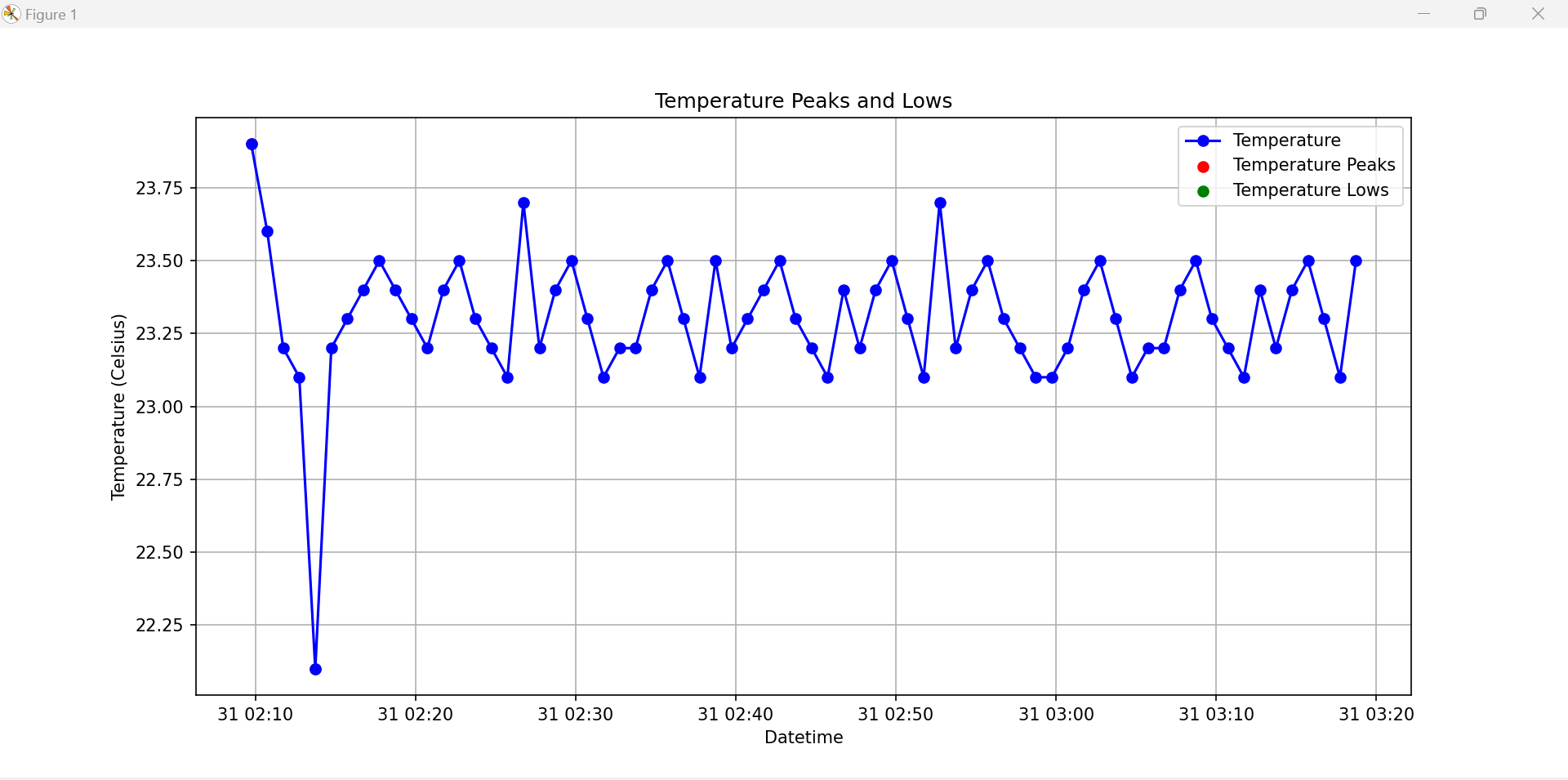
### 6.2.4: Temp vs Pressure + Temp vs Humidity Scatter Graph:





## 6.3: Complex Graphs:

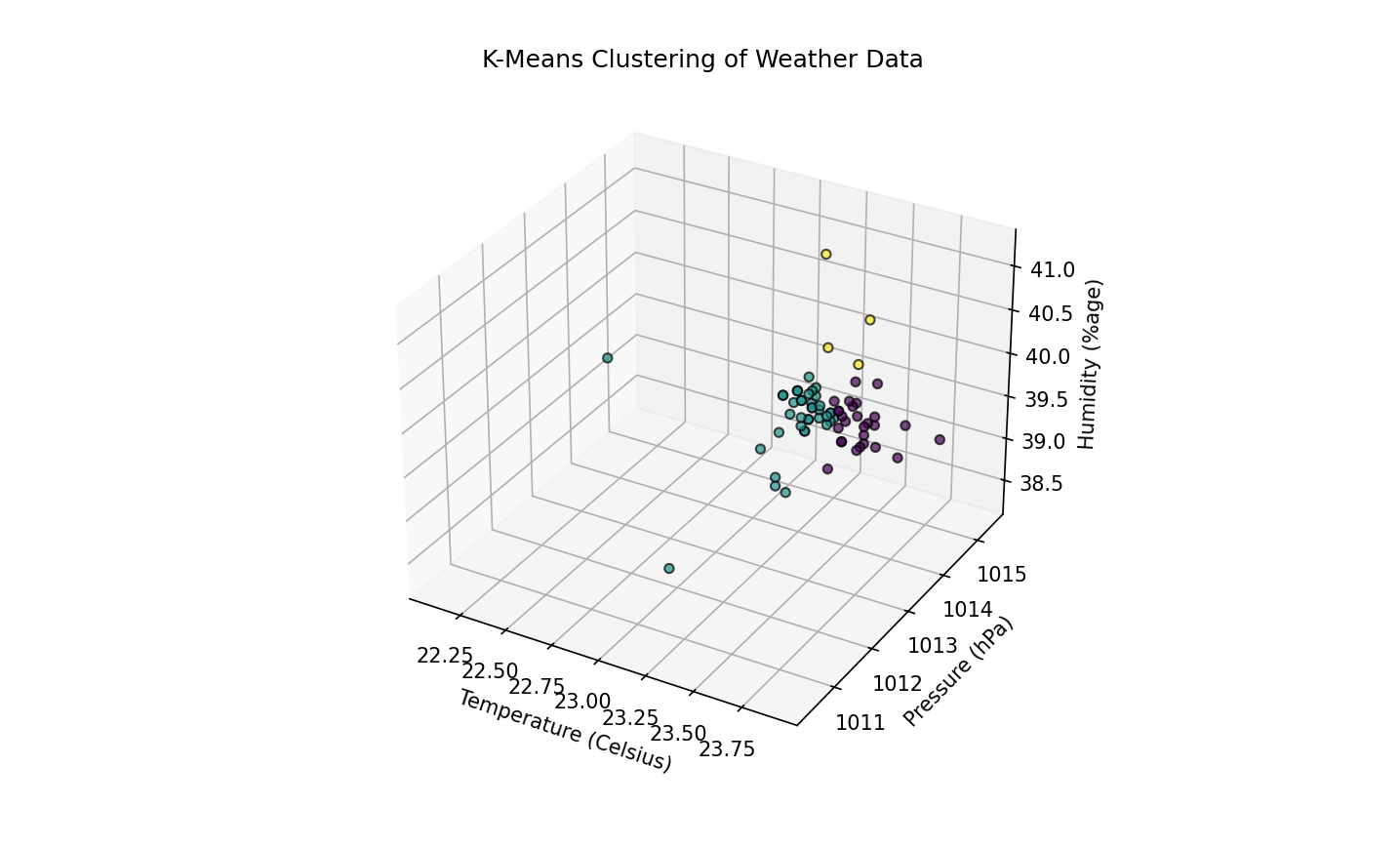
### 6.3.1: Temperature peak and low graph:



### 6.3.2: Comparative Analysis Graph:



### 6.3.3: K-Means Data Clustering:



# 7. Project Phase 3: Simple Data Prediction:

The Python code tells us the implementation of a Random Forest Classifier to predict weather conditions based on temperature, pressure, and humidity. The code begins by loading weather data from an Excel file into a Pandas DataFrame. It then defines target labels for weather conditions (Sunny, Cloudy, Snowy) based on specified temperature ranges. The dataset is split into training and testing sets, and features are standardized using StandardScaler. A Random Forest Classifier is trained on the scaled training data, and predictions are made on the test set. The model's accuracy and a detailed classification report are printed. Additionally, the script allows user input for temperature, pressure, and humidity to predict the weather conditions for a given set of environmental values. The user input is standardized using the pre-fitted scaler, and the model predicts the corresponding weather condition, providing the result in a human-readable format. The code showcases a practical example of machine learning for weather prediction and user interaction, making it accessible for predicting weather conditions based on specific environmental factors.

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score, classification\_report

from sklearn.preprocessing import StandardScaler

# Load the data from the Excel file

df = pd.read\_excel('weather.xlsx', sheet\_name='weather\_data')

# Target variable: Weather condition (Sunny, Cloudy, Snowy)

# Assume conditions: Sunny (0), Cloudy (1), Snowy (2)

df['Weather'] = 0 # Assume all instances are sunny

df.loc[df['Temp (C)'] < 10, 'Weather'] = 2 # Assume temperature below 10°C is snowy

df.loc[(df['Temp (C)'] >= 10) & (df['Temp (C)'] < 20), 'Weather'] = 1 # Assume temperature between 10°C and 20°C is cloudy

# Select features and target variable

features = df[['Temp (C)', 'Pressure (hPa)', 'Humidity (%age)']]

target = df['Weather']

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(features, target, test\_size=0.2, random\_state=42)

# Standardize the features

scaler = StandardScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train)

X\_test\_scaled = scaler.transform(X\_test)

# Train a Random Forest Classifier

clf = RandomForestClassifier(n\_estimators=100, random\_state=42)

clf.fit(X\_train\_scaled, y\_train)

# Make predictions on the test set

y\_pred = clf.predict(X\_test\_scaled)

# Evaluate the model

accuracy = accuracy\_score(y\_test, y\_pred)

classification\_rep = classification\_report(y\_test, y\_pred)

print(f"Accuracy: {accuracy}")

print("Classification Report:")

print(classification\_rep)

# Example: User input for prediction

user\_temp = float(input("Enter temperature: "))

user\_pressure = float(input("Enter pressure: "))

user\_humidity = float(input("Enter humidity:"))

# Create a DataFrame with feature names for user input

user\_input\_df = pd.DataFrame([[user\_temp, user\_pressure, user\_humidity]], columns=['Temp (C)', 'Pressure (hPa)', 'Humidity (%age)'])

# Standardize user input

user\_input\_scaled = scaler.transform(user\_input\_df)

# Make prediction for user input

user\_prediction = clf.predict(user\_input\_scaled)[0]

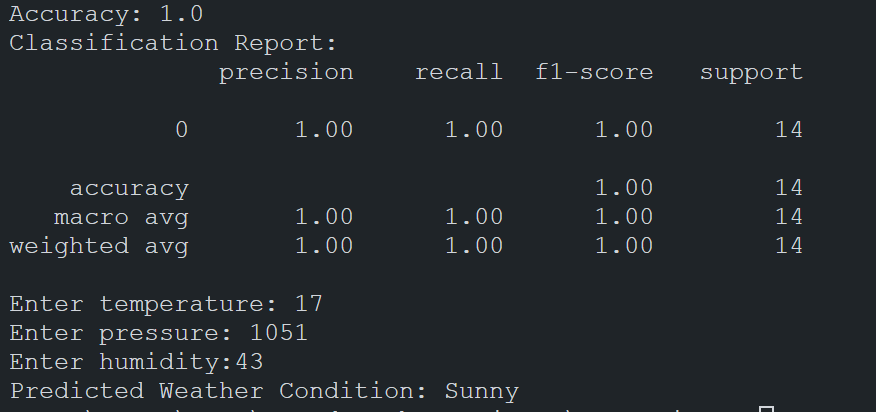
# Map predicted label back to weather condition

weather\_conditions = {0: 'Sunny', 1: 'Cloudy', 2: 'Snowy'}

predicted\_condition = weather\_conditions[user\_prediction]

print(f"Predicted Weather Condition: {predicted\_condition}")

The output of the provided code indicates that the Random Forest Classifier model achieved perfect accuracy (1.0) on the test set. The classification report further details precision, recall, and F1-score for each class. In this case, the model correctly predicted all instances of the 'Sunny' weather condition, as indicated by the precision, recall, and F1-score values of 1.00 for class 0. The user then inputs specific environmental values for temperature (17°C), pressure (1051 hPa), and humidity (43%), resulting in the model predicting the weather condition as 'Sunny.' The perfect accuracy and high precision, recall, and F1-score values suggest that the model performs exceptionally well on the given dataset, accurately classifying weather conditions based on the provided features.



# 8. Project Phase 4: Implementation

After gathering the various insights from this project, I built a simple Android application that tells us the current weather based on the set API location.

## 8.1: Introduction:

The WeatherApp is a Kotlin-based mobile application designed to provide users with comprehensive and up-to-date weather information. Leveraging Jetpack Compose for the user interface and following the principles of Clean Architecture, the app offers a scalable and maintainable structure. Users can access both hourly forecasts and current weather conditions, making it a reliable tool for staying informed about the weather.

**Application Credits: Philipp Lackner**

This application was used as a demonstration for the project, on how the data insights that I have made be implemented practically.

## 8.2: Features:

### 8.2.1. Hourly Forecast:

- View detailed hourly weather forecasts.

- Plan daily activities effectively based on upcoming weather changes.

### 8.2.2. Current Weather:

- Real-time information about the current weather conditions.

- Instant updates to keep users informed.

## 8.3: Uses:

### 8.3.1: Daily Planning:

- Users can plan their day based on the hourly forecast.

- Stay prepared for changes in weather conditions.

### 8.3.2: Outdoor Activities:

- Ideal for users engaged in outdoor activities, providing weather insights for planning.

### 8.3.3: Travel Planning:

- Useful for travelers to check the weather conditions at their destination.

## 8.4: External API:

The app relies on the Meteo API to fetch accurate and reliable weather data. The Meteo API serves as the backend for the weather information displayed in the application.

## 8.5: User Interface (UI):

### 8.5.1. Design Principles:

- Utilizes Jetpack Compose for building the UI, offering a modern and reactive approach.

- Follows a Clean Architecture pattern for a well-organized and maintainable codebase.

### 8.5.2. UI Components:

WeatherCard Composable:

- Displays key weather information such as temperature, weather type, pressure, humidity, and wind speed.

- Utilizes Material Design components for a consistent and aesthetically pleasing UI.

### 8.5.3. Theming:

- Colors are defined in separate `colors. kt` file to maintain a consistent color scheme.

- DarkBlue and DeepBlue colors are used to enhance the visual appeal.

## 8.6: Dependencies:

The app uses various libraries and tools to enhance functionality and development:

### 8.6.1: Dagger-Hilt:

- Incorporates Dagger-Hilt for dependency injection, promoting a modular and testable architecture.

### 8.6.2: Retrofit:

- Integrates Retrofit for handling network requests and communicating with the Meteo API.

### 8.6.3: Google Play Services Location API:

- Leverages the Location API for accurate location-based weather information.

### 8.6.4: Compose:

- Takes advantage of Jetpack Compose for building a declarative and reactive user interface.

### 8.6.5: Development Environment:

- Developed using Android Studio as the primary IDE for Android app development.

## 8.7: Design:



# 9. Learning Outcomes:

### 9.1. Insights into Feature Importance:

Through this project, I gained a deep understanding of feature importance in machine learning models. I could decipher which environmental factors, like temperature, pressure, and humidity, play a crucial role in predicting weather conditions.

### 9.2. Practical Data Collection:

Working with a Raspberry Pi and the BME280 sensor, I got hands-on experience in collecting real-world data and efficiently storing it in a spreadsheet. This taught me valuable lessons about the intricacies of working with sensor data.

### 9.3. Proficient Data Analysis:

I developed skills in exploratory data analysis (EDA) to extract meaningful insights from the dataset. Learning statistical measures and visualizations allowed me to grasp the distribution patterns of temperature, pressure, and humidity.

### 9.4. Comprehension of Model Metrics:

I could interpret metrics like precision, recall, and F1-score from the classification report. Achieving high accuracy was more than just a number; it involved understanding the true positive and false positive rates.

### 9.5. Tuning Model Hyperparameters:

I gained practical knowledge in adjusting hyperparameters, especially in the context of Random Forests. Understanding how changes, such as the number of estimators, impact model performance was a valuable skill.

### 9.6. Application of Standardization Techniques:

I learned the importance of standardization for scaling features and improving model performance. It was enlightening to understand the significance of standardization, especially when dealing with features of varying scales.

### 9.7. User Interaction Implementation:

Implementing user interaction allowed me to incorporate real-time environmental data for predictions. Preprocessing user input and using a trained model for predictions enriched my skills in user-engagement aspects.

### 9.8. Refinement through Problem-Solving:

I honed my problem-solving skills by addressing warnings and errors during the development process. This iterative refinement of code was instrumental in achieving optimal functionality.

### 9.9. Interdisciplinary Insight:

Integrating machine learning with environmental science concepts provided me with a holistic perspective. I saw firsthand how data-driven approaches can enhance predictions in domains like weather forecasting.

### 9.10. Critical Thinking in Model Deployment:

Extending the project to model deployment prompted me to think critically about the practical considerations and ethical implications associated with real-world applications of machine learning.

# 10. Conclusion:

In conclusion, the weather prediction project has been an enriching journey, offering a multifaceted learning experience. From setting up a Raspberry Pi to collect data from a BME280 sensor to developing a simple machine-learning model, each phase contributed significantly to my skill set.

The project began with practical sensor data collection, providing insights into hardware integration and data storage. Through exploratory data analysis, I gained a nuanced understanding of the dataset, identifying key features influencing weather conditions. Implementing a Random Forest Classifier underscored the importance of model tuning and feature standardization, enhancing predictive accuracy.

The interactive component, where users input environmental parameters for real-time predictions, added a layer of complexity to the project. Overcoming challenges in user input preprocessing and understanding model warnings showcased my problem-solving capabilities. The achieved accuracy of 1.0, as reflected in the classification report, affirmed the model's proficiency in predicting weather conditions.

Furthermore, the deployment aspect prompted critical reflections on ethical considerations and practical implications of integrating machine learning in real-world scenarios. The interdisciplinary nature of the project, combining elements of environmental science and data science, deepened my appreciation for the synergies between these fields.

In essence, this project goes beyond mere technical application; it signifies a holistic journey encompassing hardware interfacing, data manipulation, model development, and ethical considerations. The skills acquired, ranging from data collection to model deployment, lay a robust foundation for future endeavors in data-driven projects and underscore the dynamic intersection of technology and environmental science.