Student name: Zakarya Guerinat

Student ID: 217090531

SIT225: Data Capture Technologies

Activity 5.1: Firebase Realtime database

The Firebase Realtime Database is a cloud-hosted NoSQL database that lets you store and sync data between your users in real-time. Data is stored as JSON and synchronized in real-time to every connected client. In this activity, you will set up and perform operations such as queries and updates on the database using Python programming language.

Hardware Required

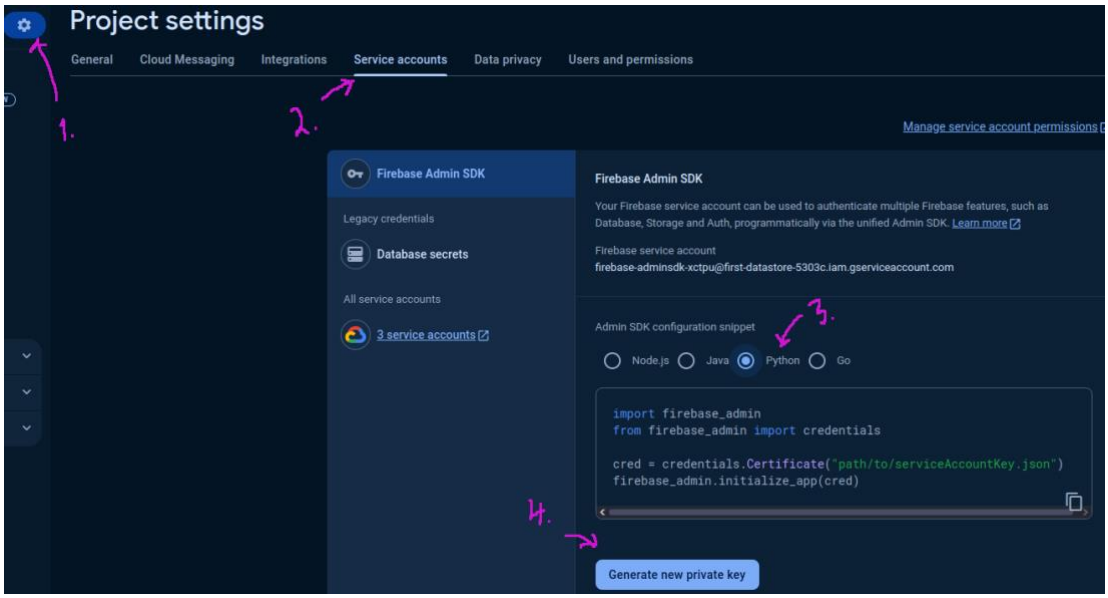
No hardware is required.

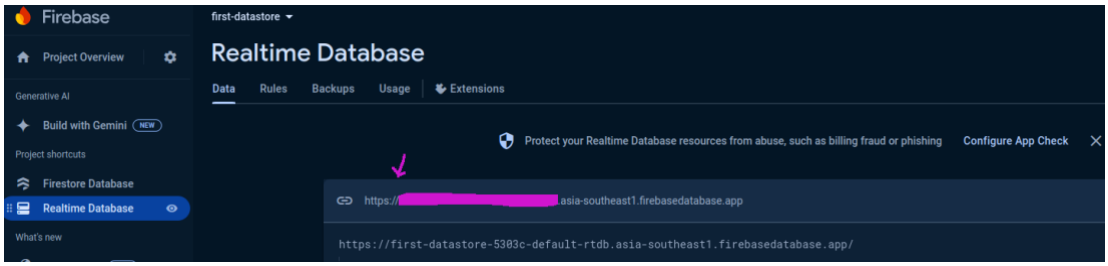
Software Required

Firebase Realtime database
Python 3

Steps

Step	Action
1	**Create an Account**: First, you will need to create an account in the Firebase console, follow instructions in the official Firebase document (https://firebase.google.com/docs/database/rest/start).
2	**Create a Database**: Follow the above Firebase document to create a database. When you click on Create Database, you have to specify the location of the database and the security rules. Two rules are available – locked mode and test mode; since we will be using the database for reading, writing, and editing, we choose test mode.
3	**Setup Python library for Firebase access**: We will be using Admin Database API, which is available in *firebase_admin* library. Use the below command in the command line to install. You can

	follow a Firebase tutorial here (https://www.freecodecamp.org/news/how-to-get-started-with-firebase-using-python). $ pip install firebase_admin Firebase will allow access to Firebase server APIs from Google Service Accounts. To authenticate the Service Account, we require a private key in JSON format. To generate the key, go to project settings, click Generate new private key, download the file, and place it in your current folder where you will create your Python script. 
4	**Connect to Firebase using Python version of Admin Database API**: A credential object needs to be created to initialise the Python library which can be done using the Python code below. Python notebook can be downloaded here (https://github.com/deakin-deep-dreamer/sit225/blob/main/week_5/firebase_explore.ipynb). `import firebase_admin` `databaseURL = 'https://XXX.firebasedatabase.app/'` `cred_obj = firebase_admin.credentials.Certificate(` `'first-datastore-5303c-firebase-adminsdk-xctpu-c9902044ac.json'` `)` `default_app = firebase_admin.initialize_app(cred_obj, {` `'databaseURL':databaseURL` `})` The databaseURL is a web address to reach your Firebase database that you have created in step 2. This URL can be found in the Data tab of Realtime Database.



If you compile the code snippet above, it should do with no error.

5 **Write to database Using the set() Function**:

We set the reference to the root of the database (or we could also set it to a key value or child key value). Data needs to be in JSON format as below.

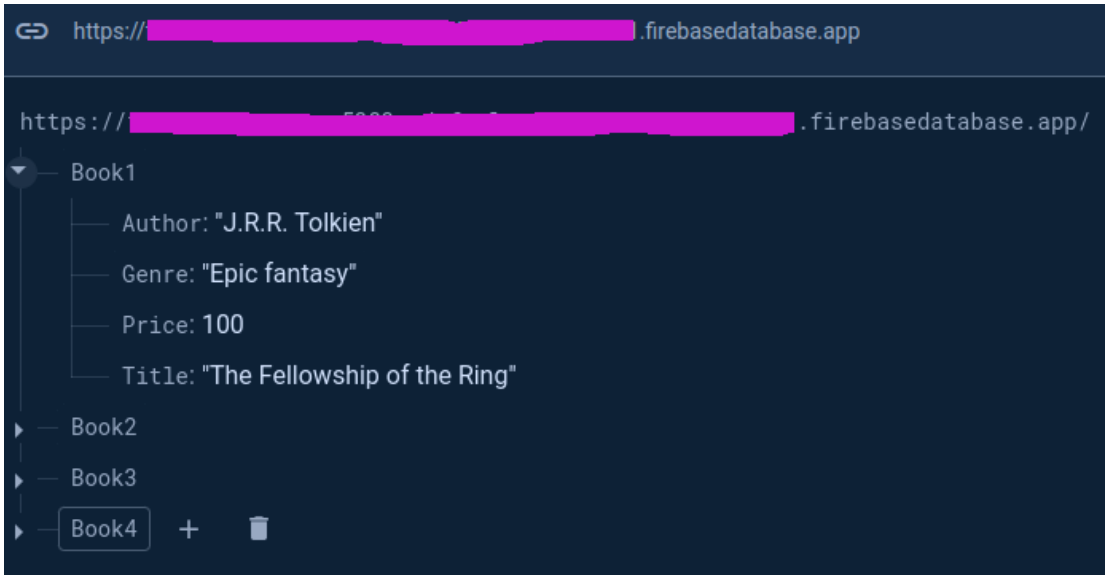
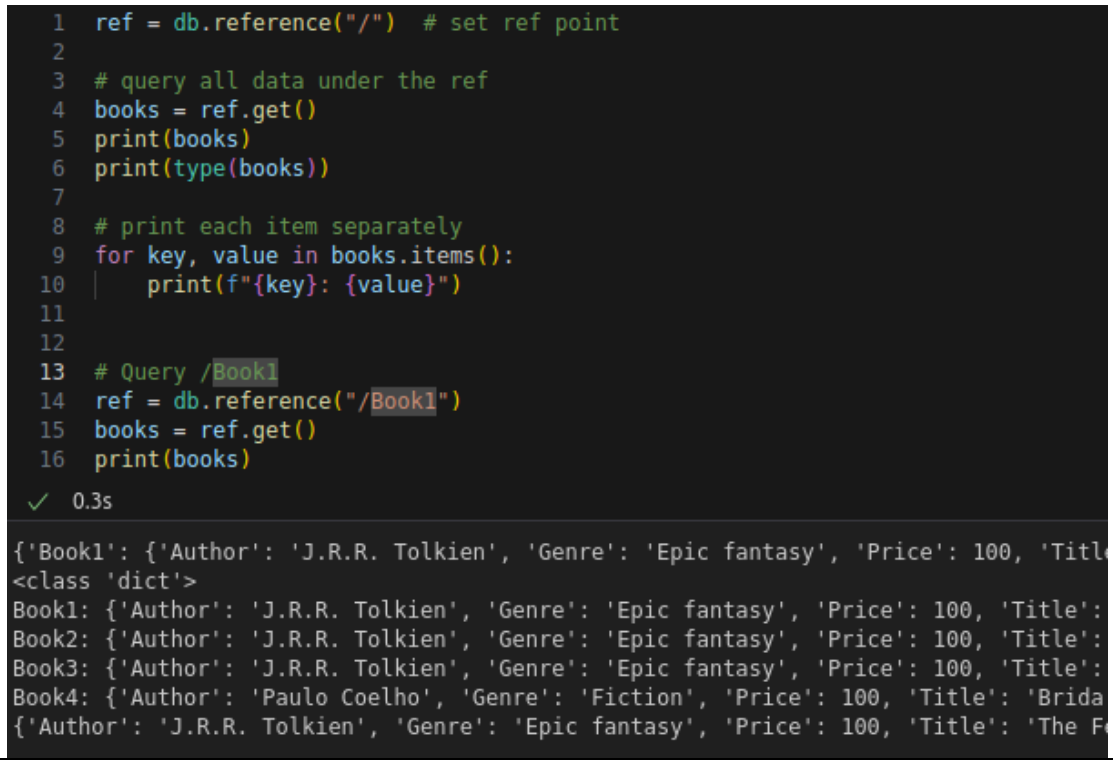
```
from firebase_admin import db

# A reference point is always needed to be set
# before any operation is carried out on a database.
#
ref = db.reference("/")

# JSON format data (key/value pair)
data = {  # Outer {} contains inner data structure
    "Book1":
    {
        "Title": "The Fellowship of the Ring",
        "Author": "J.R.R. Tolkien",
        "Genre": "Epic fantasy",
        "Price": 100
    },
    "Book2":
    {
        "Title": "The Two Towers",
        "Author": "J.R.R. Tolkien",
        "Genre": "Epic fantasy",
        "Price": 100
    },
    "Book3":
    {
        "Title": "The Return of the King",
        "Author": "J.R.R. Tolkien",
        "Genre": "Epic fantasy",
        "Price": 100
    },
    "Book4":
    {
        "Title": "Brida",
        "Author": "Paulo Coelho",
        "Genre": "Fiction",
        "Price": 100
    }
}

# JSON format data is set (overwritten) to the reference
# point set at /, which is the root node.
#
ref.set(data)
```

A reference point always needed to be set where the data read/write will take place. In the code above, the reference point is set at the root of the NoSQL Document, where consider the database is a JSON tree and / is the root node

	of the tree). The set() function writes (overwrites) data at the set reference point. You can visualise the data in the Firebase console as below - 
6	**Read data using get() function**: Data can be read using get() function on the reference set beforehand, as shown below. 

	Consider the reference set in line 1 and the output compared to the reference set at line 14 and the bottom output line to understand the use of db.reference() and ref.get().
7	**Write to database Using the push() Function**: The push() function saves data under a *unique system generated key*. This is different than set() where you set the keys such as Book1, Book2, Book3 and Book4 under which the content (author, genre, price and title) appears. Let's try to push the same data in the root reference. Note that since we already has data under root / symbol, setting (or pushing) in the same reference point will eventually rewrite the original data. ``` 1 # Write using push() function 2 # Note that a set() is called on top of push() 3 # 4 ref = db.reference("/") 5 ref.set({ 6 "Books": 7 { 8 "Best_Sellers": -1 9 } 10 }) 11 12 ref = db.reference("/Books/Best_Sellers") 13 14 for key, value in data.items(): 15 ref.push().set(value) ✓ 2.0s ``` The output will reset the previous data set in / node. The current data is shown below.

As you can see, under /Books/Best_Sellers there are 4 nodes where the node head (or node ID) is a randomly generated key which is due to the use of push() function. When data key does not matter, the use of push() function desirable.

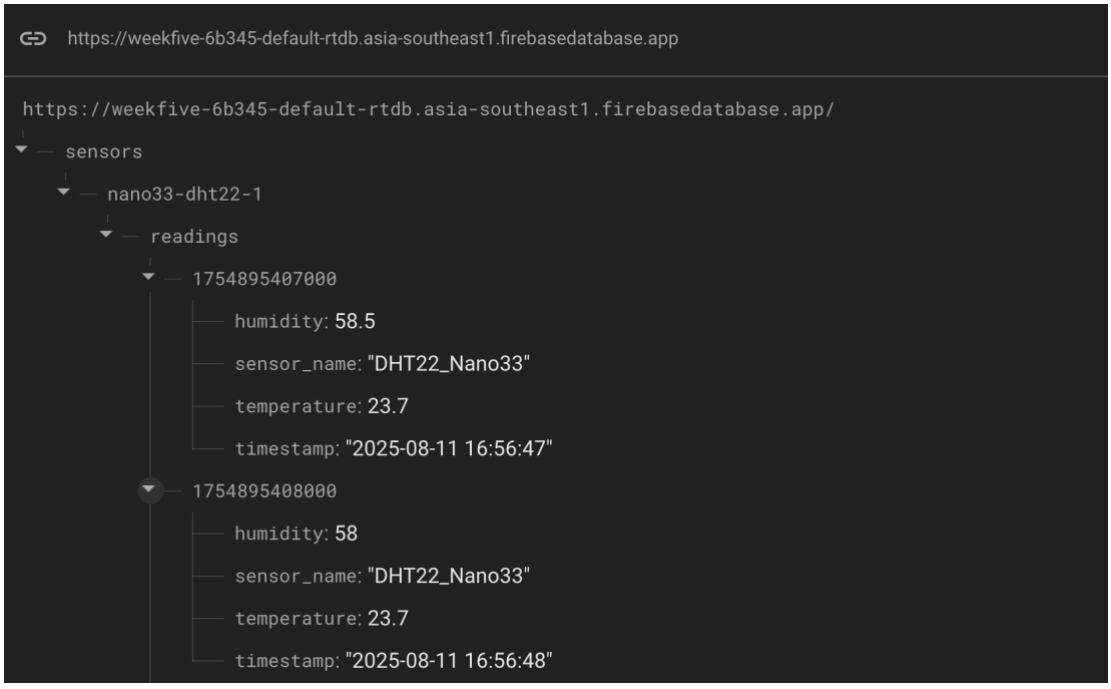
8 **Update data**:
Let's say the price of the books by J. R. R. Tolkien is reduced to 80 units to offer a discount. The first 3 books are written by this author, and we want to apply for a discount on all of them.

```
1  # Update data
2  #
3  # Requirement: The price of the books by
4  # J. R. R. Tolkien is reduced to 80 units to
5  # offer a discount.
6  #
7  ref = db.reference("/Books/Best_Sellers/")
8  best_sellers = ref.get()
9  print(best_sellers)
10 for key, value in best_sellers.items():
11     if(value["Author"] == "J.R.R. Tolkien"):
12         value["Price"] = 90
13         ref.child(key).update({"Price":80})
✓ 0.9s
```

As you can see, the author name is compared and the new price is set in the best_sellers dictionary and finally, an update() function is called on the ref, however, the current ref is a '/Books/Best_Sellers/', so we need to locate the

	child under the ref node, so ref.child(key) is used in line 13. The output is shown below with a discounted price. 
9	**Delete data**: Let's delete all bestseller books with J.R.R. Tolkien as the author. You can locate the node using db.reference() (line 4) and then locate specific record (for loop in line 6) and calling set() with empty data {} as a parameter, such as set({}). The particular child under the ref needs to be located first by using ref.child(key), otherwise, the ref node will be removed – BE CAREFUL.

	`` 1 # Let's delete all best seller books 2 # with J.R.R. Tolkien as the author. 3 # 4 ref = db.reference("/Books/Best_Sellers") 5 6 for key, value in best_sellers.items(): 7 if(value["Author"] == "J.R.R. Tolkien"): 8 ref.child(key).set({}) ``` This keeps only the other author data, as shown below.  If ref.child() not used, as shown the code below, all data will be removed. ``` 1 ref = db.reference("/Books/Best_Sellers") 2 ref.set({}) ``` Now in Firebase console you will see no data exists.
10	Question: Run all the cells in the Notebook you have downloaded in Step 4, fill in the student information at the top cell of the Notebook. Convert the Notebook to PDF and merge with this activity sheet PDF. Answer: Convert the Notebook to PDF and merge with this activity sheet PDF.
10	Question: Create a sensor data structure for DHT22 sensor which contains attributes such as sensor_name, timestamp, temperature and humidity. Remember there will be other sensors with different sensor variables such as DHT22 has 2 variables, accelerometer sensor has 3. For each such sensor, you will need to gather data over time. Discuss how you are going to handle multiple data values in JSON format? Justify your design.

	Answer: the dht22 data I haveis in csv format, the values will be parsed, the timestamp will be kept as is and converted to milliseconds and use that as the firebase key, itll make sorting out by time easy and fast. The temperature input will be converted to a float and rounder to two decimal places this will remove noisydigits, we will also have a range of acceptable temperature, humidity will converted to a flaot and rounder to 1 decimal place, missing values and messy rows will be skipped. Units will be in Celsius for temperature and percentage for humidity and the type for both units will be number. This makes sense because it will allow for easier reading and sorting also nicer numbers to work with. the json output will look like this { "sensor_name": "DHT22_Nano33", "timestamp": "2025-08-11 16:56:47", "temperature": 23.7, "humidity": 58.4 }
11	Question: Generate some random data for DHT22 sensor, insert data to database, query all data and screenshot the output here. Answer: 

	(.venv) ➜ ArduinoSensorLogger python query_latest_20.py Device: nano33-dht22-1 | Returned: 20 reading(s) 2025-08-11 17:36:07 | T=22.8°C H=60.5% | key=1754897767000 2025-08-11 17:36:12 | T=22.8°C H=60.1% | key=1754897772000 2025-08-11 17:36:13 | T=22.9°C H=60.1% | key=1754897773000 2025-08-11 17:36:17 | T=22.9°C H=60.4% | key=1754897777000 2025-08-11 17:36:23 | T=22.9°C H=60.1% | key=1754897783000 2025-08-11 17:36:24 | T=22.8°C H=60.1% | key=1754897784000 2025-08-11 17:36:27 | T=22.8°C H=59.9% | key=1754897787000 2025-08-11 17:36:28 | T=22.9°C H=59.9% | key=1754897788000 2025-08-11 17:36:32 | T=22.9°C H=60.3% | key=1754897792000 2025-08-11 17:36:38 | T=22.9°C H=60.4% | key=1754897798000 2025-08-11 17:36:42 | T=22.9°C H=60.6% | key=1754897802000 2025-08-11 17:36:47 | T=22.9°C H=60.7% | key=1754897807000 2025-08-11 17:36:52 | T=22.9°C H=60.5% | key=1754897812000 2025-08-11 17:36:57 | T=22.9°C H=60.0% | key=1754897817000 2025-08-11 17:37:02 | T=22.9°C H=60.1% | key=1754897822000 2025-08-11 17:37:03 | T=23.0°C H=60.1% | key=1754897823000 2025-08-11 17:37:07 | T=23.0°C H=60.2% | key=1754897827000 2025-08-11 17:37:12 | T=23.0°C H=60.3% | key=1754897832000 2025-08-11 17:37:13 | T=22.9°C H=60.3% | key=1754897833000 2025-08-11 17:37:18 | T=22.9°C H=60.4% | key=1754897838000
12	Question: Generate some random data for the SR04 Ultrasonic sensor, insert data to database, query all data and screenshot the output here.

Answer:

https://weekfive-6b345-default-rtdb.asia-southeast1.firebasedatabase.app

```
Device: nano33-sr04-1 | Returned: 30 reading(s)
2025-08-13 21:35:51  |  distance_cm=32.8  sensor_name=HC-SR04
2025-08-13 21:35:53  |  distance_cm=32.5  sensor_name=HC-SR04
2025-08-13 21:35:54  |  distance_cm=38.1  sensor_name=HC-SR04
2025-08-13 21:35:55  |  distance_cm=38.9  sensor_name=HC-SR04
2025-08-13 21:35:56  |  distance_cm=35.5  sensor_name=HC-SR04
2025-08-13 21:35:57  |  distance_cm=32.7  sensor_name=HC-SR04
2025-08-13 21:35:58  |  distance_cm=33.1  sensor_name=HC-SR04
2025-08-13 21:35:59  |  distance_cm=31.9  sensor_name=HC-SR04
2025-08-13 21:36:01  |  distance_cm=39.3  sensor_name=HC-SR04
2025-08-13 21:36:02  |  distance_cm=33.2  sensor_name=HC-SR04
2025-08-13 21:36:03  |  distance_cm=38.1  sensor_name=HC-SR04
2025-08-13 21:36:04  |  distance_cm=33.3  sensor_name=HC-SR04
2025-08-13 21:36:05  |  distance_cm=39.8  sensor_name=HC-SR04
2025-08-13 21:36:06  |  distance_cm=32.8  sensor_name=HC-SR04
2025-08-13 21:36:07  |  distance_cm=32.2  sensor_name=HC-SR04
2025-08-13 21:36:09  |  distance_cm=33.6  sensor_name=HC-SR04
2025-08-13 21:36:10  |  distance_cm=34.8  sensor_name=HC-SR04
2025-08-13 21:36:11  |  distance_cm=34.1  sensor_name=HC-SR04
2025-08-13 21:36:12  |  distance_cm=32.6  sensor_name=HC-SR04
2025-08-13 21:36:13  |  distance_cm=31.5  sensor_name=HC-SR04
2025-08-13 21:36:14  |  distance_cm=37.8  sensor_name=HC-SR04
2025-08-13 21:36:15  |  distance_cm=31.6  sensor_name=HC-SR04
2025-08-13 21:36:16  |  distance_cm=39.0  sensor_name=HC-SR04
2025-08-13 21:36:18  |  distance_cm=31.2  sensor_name=HC-SR04
2025-08-13 21:36:19  |  distance_cm=33.9  sensor_name=HC-SR04
2025-08-13 21:36:20  |  distance_cm=31.4  sensor_name=HC-SR04
2025-08-13 21:36:21  |  distance_cm=35.5  sensor_name=HC-SR04
2025-08-13 21:36:22  |  distance_cm=37.2  sensor_name=HC-SR04
2025-08-13 21:36:23  |  distance_cm=36.9  sensor_name=HC-SR04
```

13

Question: Firebase Realtime database generates events on data operations. You can refer to section ‘Handling Realtime Database events’ in the document (https://firebase.google.com/docs/functions/database-events?gen=2nd). Discuss in the active learning session and summarise the idea of database events and how it is handled using Python SDK.

Note that these events are useful when your sensors (from Arduino script) store data directly to Firebase Realtime database and you would like to track data update actions from a central Python application such as a monitoring dashboard.

Answer: <Your answer>

Activity 5.2: Data wrangling

Data wrangling is the process of converting raw data into a usable form. The process includes collecting, processing, analyzing, and tidying the raw data so that it can be easily read and analyzed. In this activity, you will use the common library in python, "pandas".

Hardware Required

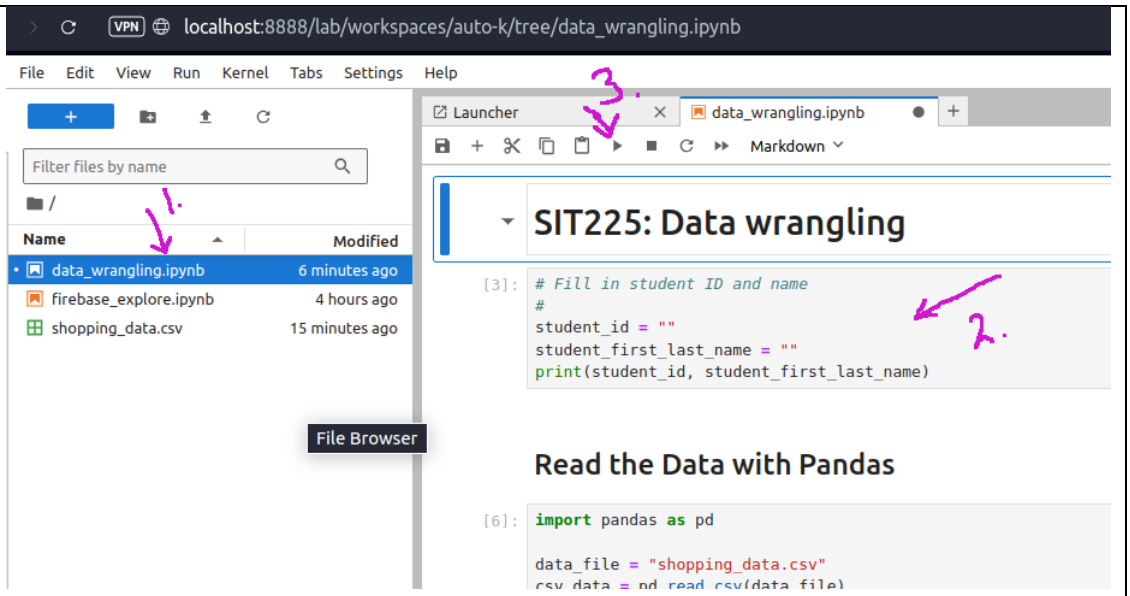
No hardware is required.

Software Required

Python 3
Pandas Python library

Steps

Step	Action
1	Install Pandas using the command below. Most likely you already have Pandas installed if you have installed Python using Anaconda disribution (https://www.anaconda.com/download). $ pip install pandas A Python notebook is shared in the GitHub link (https://github.com/deakin-deep-dreamer/sit225/tree/main/week_5). There will be a data_wrangling.ipynb, shopping_data.csv and shopping_data_missingvalue.csv files among others. Download the week_5 folder in your computer, open a command prompt in that folder, and write the command below in the command line: $ jupyter lab This will open Python Jupyter Notebook where in the left panel you can see the files (labeled as 1 in figure).

	 Each cell contains Python code (labeled as 2 in figure), you can run a cell by clicking on the cell, so the cursor appears in that cell and then click on the play button at the top of the panel (labeled as 3 in the figure).
2	Question: Run each cell to produce output. Follow instructions in the notebook to complete codes in some of the cells. Convert the notebook to PDF from menu File > Save and Export Notebook As > PDF. Convert this activity sheet to PDF and merge with the notebook PDF. Answer: There is no answer to write here. You have to answer in the Jupyter Notebook.
3	Question: Once you went through the cells in the Notebook, you now have a basic understanding of data wrangling. Pandas are a powerful tool and can be used for reading CSV data. Can you use Pandas in reading sensor CSV data that you generated earlier? Describe if any modification you think necessary? Answer: yes pandas works well with our sensor data we generated.depending on how the data is saved. If it is not formatted properly it will need wrangling
4	Question: What do you understand of the Notebook section called Handling Missing Value? Discuss in group and briefly summarise different missing value imputation methods and their applicability on different data conditions. Answer: the simplest approach is dropping missing rows and columns which is useful only if the missing data isn't great in number but it risks losing data,

	we use mean and median for when the data is symmetric but median is better when the data has outliers .

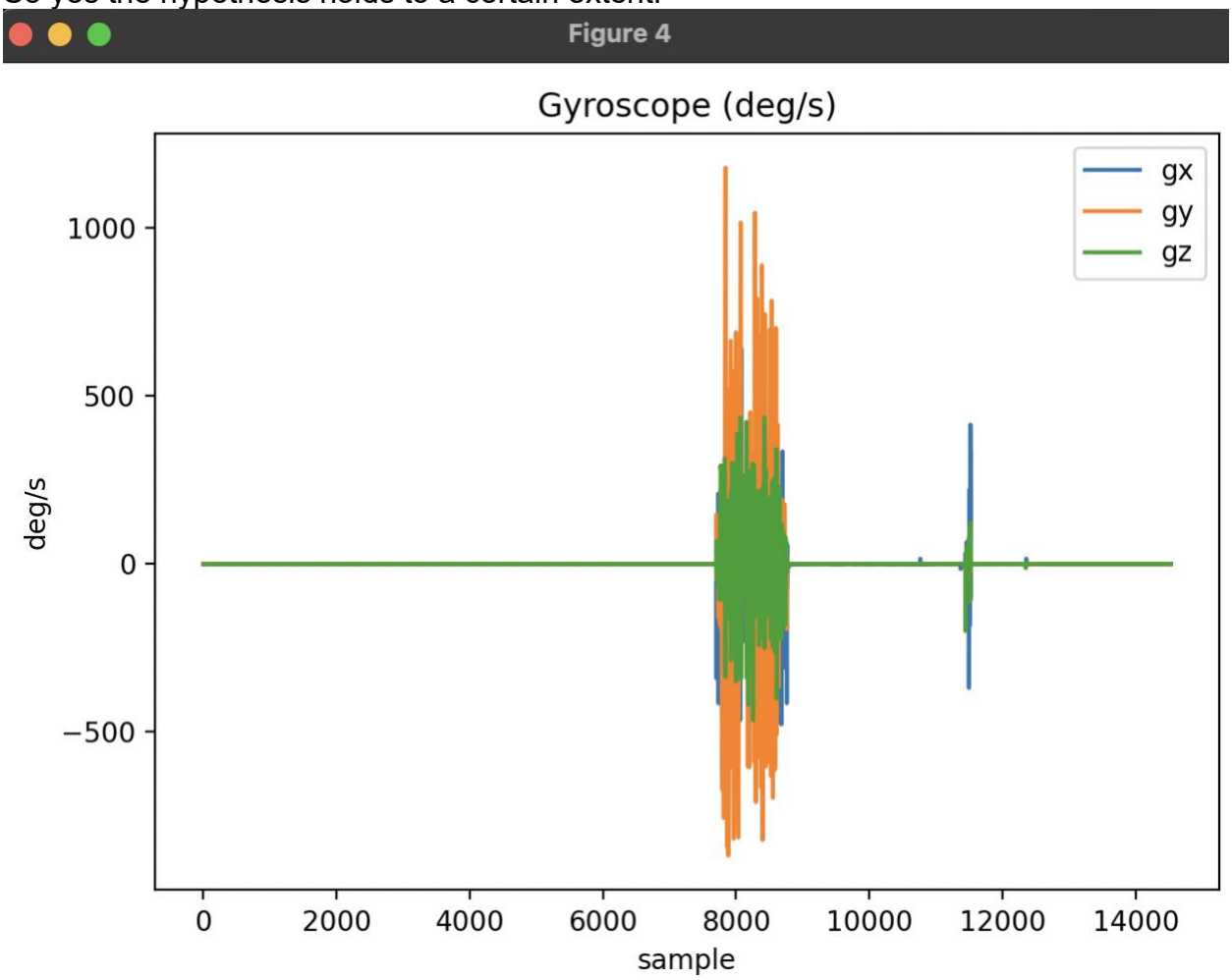
Q2)

The hypothesis is when the arduinp board is roatated primarily around the yaxis the gyroscope will record larger angular velocity x compared to the x and z axis.

- Long flat lines = board is stationary.
- Cluster of high activity = he board was rotated.
- Orange (gy) spikes higher than blue (gx) and green (gz).

- stillness: The long sections at ~0 deg/s confirm the board was not moving for large portions of time.
- Activity cluster (samples ~7000–9000): This is when the board was rotated. The gy axis dominates with the largest swings (both positive and negative), which aligns with movement primarily around the Y-axis.
- Relative change**:** gx shows smaller fluctuations, while gz remains almost flat, suggesting minimal rotation along those axes.
- Noise/outliers: A few extreme spikes are visible, which may be due to quick jerks, hand tremors, or sensor noise.

So yes the hypethesis holds to a certain extent.



Q3)

```python
import sys, time
from datetime import datetime
import serial

from firebase_client import get_db

# ---------- SETTINGS ----------
DEVICE_ID = "nano33-gyro-1"
PORT = "/dev/tty.usbmodem21401"   # fixed port for your Nano
BAUD = 115200
ROUND_DP = 3
# ------------------------------

def now_epoch_ms() -> int:
    return int(time.time() * 1000)

def now_str() -> str:
    return datetime.now().strftime("%Y-%m-%d %H:%M:%S")

def is_header(line: str) -> bool:
    s = line.strip().lower()
    return s.startswith("gx") or "gx,gy,gz" in s

def main(device_id: str = DEVICE_ID):
    print(f"Using serial port: {PORT} @ {BAUD}")
    db = get_db()
    base_ref = db.reference(f"/sensors/{device_id}/readings")
    print(f"Firebase path: /sensors/{device_id}/readings")

    try:
        with serial.Serial(PORT, BAUD, timeout=2) as ser:
            print("Streaming… (Ctrl+C to stop)")
            line_no = 0
            while True:
                raw = ser.readline().decode("utf-8", errors="ignore").strip()
                if not raw:
                    continue
                line_no += 1
                if is_header(raw):
                    continue

                parts = [p.strip() for p in raw.split(",")]
                if len(parts) != 3:
                    continue  # malformed line
```

```
                try:
                    gx = round(float(parts[0]), ROUND_DP)
                    gy = round(float(parts[1]), ROUND_DP)
                    gz = round(float(parts[2]), ROUND_DP)
                except ValueError:
                    continue  # skip bad numbers

                if any(abs(v) > 4000 for v in (gx, gy, gz)):
                    continue  # discard absurd spikes

                key = str(now_epoch_ms())
                record = {
                    "sensor_name": "LSM6DS3_Gyro",
                    "timestamp": now_str(),   # human-readable
                    "gx": gx, "gy": gy, "gz": gz
                }
                base_ref.child(key).set(record)

                if line_no % 50 == 0:
                    print(f"[ok] sent {line_no} samples; last {gx},{gy},{gz} @
{record['timestamp']}")

    except KeyboardInterrupt:
        print("\nStopped by user.")
    except serial.SerialException as e:
        print(f"Serial error: {e}")
    print("Done.")

if __name__ == "__main__":
    # Optional CLI override for device id
    dev = sys.argv[1] if len(sys.argv) > 1 else DEVICE_ID
    main(dev)
```

```
#include <Arduino_LSM6DS3.h>


const unsigned long SAMPLE_PERIOD_MS = 20; // 50 Hz
unsigned long lastMs = 0;

void setup() {
  Serial.begin(115200);
  while (!Serial) { ; }       // wait for USB
```

```
  if (!IMU.begin()) {
    Serial.println("ERR: IMU init failed");
    while (1) { delay(1000); }
  }
  Serial.println("gx,gy,gz");  // header (Python will skip this)
}

void loop() {
  unsigned long now = millis();
  if (now - lastMs >= SAMPLE_PERIOD_MS) {
    lastMs = now;
    float gx, gy, gz;
    if (IMU.gyroscopeAvailable()) {
      IMU.readGyroscope(gx, gy, gz); // deg/s
      Serial.print(gx, 3); Serial.print(',');
      Serial.print(gy, 3); Serial.print(',');
      Serial.println(gz, 3);
    }
  }
}
```

the arudiono sketch read gyroscope values(x,yz) from the built in sensor sensor and sent them out over the USB serial connection in a comma-separated format The Python code on the computer listened to that same serial port, received each line, and cleaned it up into numbers. For every reading, the Python script added a timestamp and then uploaded the data as a JSON record into Firebase Realtime Database. In this way, the Arduino acted like a live data generator, and Python acted like a middle-man that collected the raw sensor data, gave it context (time), and stored it safely in the cloud database.

Q4)

https://deakin.au.panopto.com/Panopto/Pages/Viewer.aspx?id=81e6b36c-2a19-4519-861d-b33f0087d70b

SIT225: Data wrangling

Run each cell to generate output and finally convert this notebook to PDF.

In [1]:
```
# Fill in student ID and name
#
student_id = "217090531"
student_first_last_name = "Zakarya Guerinat"
print(student_id, student_first_last_name)
```

```
217090531 Zakarya Guerinat
```

Read the Data with Pandas

Pandas has a dedicated function read_csv() to read CSV files.

Just in case we have a large number of data, we can just show into only five rows with head function. It will show you 5 rows data automatically.

In [2]:
```
import pandas as pd

data_file = "shopping_data.csv"
csv_data = pd.read_csv(data_file)

print(csv_data)

# show into only five rows with head function
print(csv_data.head())
```

```
     CustomerID   Genre  Age  Annual Income (k$)  Spending Score (1-100)
0             1    Male   19                  15                      39
1             2    Male   21                  15                      81
2             3  Female   20                  16                       6
3             4  Female   23                  16                      77
4             5  Female   31                  17                      40
..          ...     ...  ...                 ...                     ...
195         196  Female   35                 120                      79
196         197  Female   45                 126                      28
197         198    Male   32                 126                      74
198         199    Male   32                 137                      18
199         200    Male   30                 137                      83

[200 rows x 5 columns]
   CustomerID   Genre  Age  Annual Income (k$)  Spending Score (1-100)
0           1    Male   19                  15                      39
1           2    Male   21                  15                      81
2           3  Female   20                  16                       6
3           4  Female   23                  16                      77
4           5  Female   31                  17                      40
```

Access the Column

Pandas has provided function .columns to access the column of the data source.

In [3]:
```
print(csv_data.columns)

# if we want to access just one column, for example "Age"
print("Age:")
print(csv_data["Age"])
```

```
Index(['CustomerID', 'Genre', 'Age', 'Annual Income (k$)',
       'Spending Score (1-100)'],
      dtype='object')
Age:
0      19
1      21
2      20
3      23
4      31
       ..
195    35
196    45
197    32
198    32
199    30
Name: Age, Length: 200, dtype: int64
```

Access the Row

In addition to accessing data through columns, using pandas can also access using rows. In contrast to access through columns, the function to display data from a row is the .iloc[i] function where [i] indicates the order of the rows to be displayed where the index starts from 0.

In [4]:
```
# we want to know what line 5 contains

print(csv_data.iloc[5])

print()

# We can combine both of those function to show row and column we want.
# For the example, we want to show the value in column "Age" at the first
# (remember that the row starts at 0)
#
print(csv_data["Age"].iloc[1])
```

```
CustomerID                    6
Genre                    Female
Age                          22
Annual Income (k$)           17
Spending Score (1-100)       76
Name: 5, dtype: object

21
```

Show Data Based on Range

After displaying a data set, what if you want to display data from rows 5 to 20 of a dataset? To anticipate this, pandas can also display data within a certain range, both ranges for rows only, only columns, and ranges for rows and columns

In [5]:
```
print("Shows data to 5th to less than 10th in a row:")
print(csv_data.iloc[5:10])
```

```
Shows data to 5th to less than 10th in a row:
   CustomerID   Genre  Age  Annual Income (k$)  Spending Score (1-100)
5           6  Female   22                  17                      76
6           7  Female   35                  18                       6
7           8  Female   23                  18                      94
8           9    Male   64                  19                       3
9          10  Female   30                  19                      72
```

Using Numpy to Show the Statistic Information

The describe() function allows to quickly find statistical information from a dataset. Those information such as mean, median, modus, max min, even standard deviation. Don't forget to install Numpy before using describe function.

In [6]:
```
print(csv_data.describe(include="all"))
```

```
        CustomerID   Genre         Age  Annual Income (k$)  \
count   200.000000     200  200.000000          200.000000
unique         NaN       2         NaN                 NaN
top            NaN  Female         NaN                 NaN
freq           NaN     112         NaN                 NaN
mean    100.500000     NaN   38.850000           60.560000
std      57.879185     NaN   13.969007           26.264721
min       1.000000     NaN   18.000000           15.000000
25%      50.750000     NaN   28.750000           41.500000
50%     100.500000     NaN   36.000000           61.500000
75%     150.250000     NaN   49.000000           78.000000
max     200.000000     NaN   70.000000          137.000000

        Spending Score (1-100)
count               200.000000
unique                     NaN
top                        NaN
freq                       NaN
mean                 50.200000
std                  25.823522
min                   1.000000
25%                  34.750000
50%                  50.000000
75%                  73.000000
max                  99.000000
```

Handling Missing Value

In [7]:
```
# For the first step, we will figure out if there is missing value.
print(csv_data.isnull().values.any())
print()
```

```
False
```

In [8]:
```
# We will use another data source with missing values to practice this pa
data_missing = pd.read_csv("shopping_data_missingvalue.csv")
print(data_missing.head())

print()

print("Missing? ", data_missing.isnull().values.any())
```

```
   CustomerID   Genre   Age  Annual Income (k$)  Spending Score (1-100)
0           1    Male  19.0                15.0                    39.0
1           2    Male   NaN                15.0                    81.0
2           3  Female  20.0                 NaN                     6.0
3           4  Female  23.0                16.0                    77.0
4           5  Female  31.0                17.0                     NaN

Missing?  True
```

In [ ]:
```

```

Ways to deal with missing values.

Follow the tutorial (https://deepnote.com/app/rickyharyanto14-3390/Data-Wrangling-w-Python-e5d1a23e-33cf-416d-ad27-4c3f7f467442). It includes -

1. Delete data
   - deleting rows
   - pairwise deletion
   - delete column
2. imputation
   - time series problem
     - Data without trend with seasonality (mean, median, mode, random)
     - Data with trend and without seasonality (linear interpolation)
   - general problem
     - Data categorical (Make NA as multiple imputation)
     - Data numerical or continuous (mean, median, mode, multiple imputation and linear regression)

Filling with Mean Values

The mean is used for data that has a few outliers/noise/anomalies in the distribution of the data and its contents. This value will later fill in the empty value of the dataset that has a missing value case. To fill in an empty value use the fillna() function

In [ ]:
```
print(data_missing.mean())

"""
```

```
Question: This code will generate error. Can you explain why and how it c
Move on to the next cell to find one way it can be solved.

Answer: Because there is a string value in the dataset, so the mean opera
it can be solved by letting pandas ignore the string values. ##print(data

"""
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[9], line 1
----> 1 print(data_missing.mean())
      3 """
      4
      5 Question: This code will generate error. Can you explain why and how it can be solved?
   (...)      9
     10 """

File ~/Desktop/SIT225_2025_T2/week-5/data-wrangling/.venv/lib/python3.13/site-packages/pandas/core/frame.py:11700, in DataFrame.mean(self, axis, skipna, numeric_only, **kwargs)
  11692 @doc(make_doc("mean", ndim=2))
  11693 def mean(
  11694     self,
   (...)  11698     **kwargs,
  11699 ):
> 11700     result = super().mean(axis, skipna, numeric_only, **kwargs)
  11701     if isinstance(result, Series):
  11702         result = result.__finalize__(self, method="mean")

File ~/Desktop/SIT225_2025_T2/week-5/data-wrangling/.venv/lib/python3.13/site-packages/pandas/core/generic.py:12439, in NDFrame.mean(self, axis, skipna, numeric_only, **kwargs)
  12432 def mean(
  12433     self,
  12434     axis: Axis | None = 0,
   (...)  12437     **kwargs,
  12438 ) -> Series | float:
> 12439     return self._stat_function(
  12440         , nanops.nanmean, axis, skipna, numeric_only, **kwargs
  12441     )

File ~/Desktop/SIT225_2025_T2/week-5/data-wrangling/.venv/lib/python3.13/site-packages/pandas/core/generic.py:12396, in NDFrame._stat_function(self, name, func, axis, skipna, numeric_only, **kwargs)
  12392 nv.validate_func(name, (), kwargs)
  12394 validate_bool_kwarg(skipna, "skipna", none_allowed=False)
> 12396 return self._reduce(
  12397     func, name=name, axis=axis, skipna=skipna, numeric_only=numeric_only
  12398 )

File ~/Desktop/SIT225_2025_T2/week-5/data-wrangling/.venv/lib/python3.13/site-packages/pandas/core/frame.py:11569, in DataFrame._reduce(self, op, name, axis, skipna, numeric_only, filter_type, **kwds)
  11565     df = df.T
  11567 # After possibly _get_data and transposing, we are now in the
  11568 #  simple case where we can use BlockManager.reduce
> 11569 res = df._mgr.reduce(blk_func)
  11570 out = df._constructor_from_mgr(res, axes=res.axes).iloc[0]
  11571 if out_dtype is not None and out.dtype != "boolean":

File ~/Desktop/SIT225_2025_T2/week-5/data-wrangling/.venv/lib/python3.13/site-packages/pandas/core/internals/managers.py:1500, in BlockManager.reduc
```

```
e(self, func)
   1498 res_blocks: list[Block] = []
   1499 for blk in self.blocks:
-> 1500     nbs = blk.reduce(func)
   1501     res_blocks.extend(nbs)
   1503 index = Index([None])  # placeholder

File ~/Desktop/SIT225_2025_T2/week-5/data-wrangling/.venv/lib/python3.13/s
ite-packages/pandas/core/internals/blocks.py:406, in Block.reduce(self, fu
nc)
    400 @final
    401 def reduce(self, func) -> list[Block]:
    402     # We will apply the function and reshape the result into a sin
gle-row
    403     #  Block with the same mgr_locs; squeezing will be done at a h
igher level
    404     assert self.ndim == 2
--> 406     result = func(self.values)
    408     if self.values.ndim == 1:
    409         res_values = result

File ~/Desktop/SIT225_2025_T2/week-5/data-wrangling/.venv/lib/python3.13/s
ite-packages/pandas/core/frame.py:11488, in DataFrame._reduce.<locals>.blk
_func(values, axis)
  11486         return np.array([result])
  11487 else:
> 11488     return op(values, axis=axis, skipna=skipna, **kwds)

File ~/Desktop/SIT225_2025_T2/week-5/data-wrangling/.venv/lib/python3.13/s
ite-packages/pandas/core/nanops.py:147, in bottleneck_switch.__call__.<loc
als>.f(values, axis, skipna, **kwds)
    145         result = alt(values, axis=axis, skipna=skipna, **kwds)
    146 else:
--> 147     result = alt(values, axis=axis, skipna=skipna, **kwds)
    149 return result

File ~/Desktop/SIT225_2025_T2/week-5/data-wrangling/.venv/lib/python3.13/s
ite-packages/pandas/core/nanops.py:404, in _datetimelike_compat.<locals>.n
ew_func(values, axis, skipna, mask, **kwargs)
    401 if datetimelike and mask is None:
    402     mask = isna(values)
--> 404 result = func(values, axis=axis, skipna=skipna, mask=mask, **kwarg
s)
    406 if datetimelike:
    407     result = _wrap_results(result, orig_values.dtype, fill_value=i
NaT)

File ~/Desktop/SIT225_2025_T2/week-5/data-wrangling/.venv/lib/python3.13/s
ite-packages/pandas/core/nanops.py:720, in nanmean(values, axis, skipna, m
ask)
    718 count = _get_counts(values.shape, mask, axis, dtype=dtype_count)
    719 the_sum = values.sum(axis, dtype=dtype_sum)
--> 720 the_sum = _ensure_numeric(the_sum)
    722 if axis is not None and getattr(the_sum, "ndim", False):
    723     count = cast(np.ndarray, count)

File ~/Desktop/SIT225_2025_T2/week-5/data-wrangling/.venv/lib/python3.13/s
ite-packages/pandas/core/nanops.py:1686, in _ensure_numeric(x)
   1683 inferred = lib.infer_dtype(x)
   1684 if inferred in ["string", "mixed"]:
```

```
   1685     # GH#44008, GH#36703 avoid casting e.g. strings to numeric
-> 1686     raise TypeError(f"Could not convert {x} to numeric")
   1687 try:
   1688     x = x.astype(np.complex128)

TypeError: Could not convert ['MaleMaleFemaleFemaleFemaleFemaleFemaleFemal
eMaleFemaleMaleFemaleFemaleFemaleMaleMaleFemaleMaleMaleFemaleMaleMaleFemal
eMaleFemaleMaleFemaleMaleFemaleFemaleMaleFemaleMaleMaleFemaleFemaleFemaleF
emaleFemaleFemaleFemaleMaleMaleFemaleFemaleFemaleFemaleFemaleFemaleFemaleF
emaleMaleFemaleMaleFemaleMaleFemaleMaleFemaleMaleMaleMaleFemaleFemaleMaleM
aleFemaleFemaleMaleFemaleFemaleMaleFemaleFemaleMaleMaleMaleFemaleFemaleFem
aleMaleMaleMaleFemaleFemaleMaleFemaleFemaleFemaleFemaleMaleMaleFemaleMaleFemal
eFemaleMaleFemaleFemaleMaleMaleFemaleFemaleMaleMaleMaleFemaleFemaleMaleMal
eMaleMaleFemaleFemaleMaleFemaleFemaleFemaleFemaleFemaleFemaleMaleFemaleFem
aleMaleFemaleFemaleMaleMaleMaleMaleMaleMaleFemaleFemaleMaleFemaleFemaleMal
eMaleFemaleFemaleMaleFemaleFemaleMaleMaleMaleFemaleFemaleMaleMaleMaleFemal
eFemaleFemaleFemaleMaleFemaleMaleFemaleFemaleFemaleMaleFemaleMaleFemaleMal
eFemaleFemaleMaleMaleMaleMaleMaleFemaleFemaleMaleMaleMaleMaleFemaleFemaleM
aleFemaleFemaleMaleFemaleMaleFemaleFemaleFemaleFemaleMaleFemaleFemaleFemal
eFemaleMaleMaleMale'] to numeric
```

In [10]:
```
# Genre column contains string values and numerial operation mean fails.
# Lets drop Genre column since for numerial calculation.
#
data_missing_wo_genre = data_missing.drop(columns=['Genre'])
print(data_missing_wo_genre.head())
```

```
   CustomerID   Age  Annual Income (k$)  Spending Score (1-100)
0           1  19.0                15.0                    39.0
1           2   NaN                15.0                    81.0
2           3  20.0                 NaN                     6.0
3           4  23.0                16.0                    77.0
4           5  31.0                17.0                     NaN
```

In [11]:
```
print(data_missing_wo_genre.mean())
```

```
CustomerID              100.500000
Age                      38.939698
Annual Income (k$)       61.005051
Spending Score (1-100)   50.489899
dtype: float64
```

In [12]:
```
print("Dataset with empty values! :")
print(data_missing_wo_genre.head(10))

data_filling=data_missing_wo_genre.fillna(data_missing_wo_genre.mean())
print("Dataset that has been processed Handling Missing Values with Mean
print(data_filling.head(10))

# Observe the missing value imputation in corresponding rows.
#
```

```
Dataset with empty values! :
   CustomerID   Age  Annual Income (k$)  Spending Score (1-100)
0           1  19.0                15.0                    39.0
1           2   NaN                15.0                    81.0
2           3  20.0                 NaN                     6.0
3           4  23.0                16.0                    77.0
4           5  31.0                17.0                     NaN
5           6  22.0                 NaN                    76.0
6           7  35.0                18.0                     6.0
7           8  23.0                18.0                    94.0
8           9  64.0                19.0                     NaN
9          10  30.0                19.0                    72.0
Dataset that has been processed Handling Missing Values with Mean :
   CustomerID        Age  Annual Income (k$)  Spending Score (1-100)
0           1  19.000000           15.000000               39.000000
1           2  38.939698           15.000000               81.000000
2           3  20.000000           61.005051                6.000000
3           4  23.000000           16.000000               77.000000
4           5  31.000000           17.000000               50.489899
5           6  22.000000           61.005051               76.000000
6           7  35.000000           18.000000                6.000000
7           8  23.000000           18.000000               94.000000
8           9  64.000000           19.000000               50.489899
9          10  30.000000           19.000000               72.000000
```

Filling with Median

The median is used when the data presented has a high outlier. The median was chosen because it is the middle value, which means it is not the result of calculations involving outlier data. In some cases, outlier data is considered disturbing and often considered noisy because it can affect class distribution and interfere with clustering analysis.

In [13]:
```
print(data_missing_wo_genre.median())
print("Dataset with empty values! :")
print(data_missing_wo_genre.head(10))

data_filling2=data_missing_wo_genre.fillna(data_missing_wo_genre.median()
print("Dataset that has been processed Handling Missing Values with Media
print(data_filling2.head(10))

# Observe the missing value imputation in corresponding rows.
#
```

```
CustomerID                100.5
Age                        36.0
Annual Income (k$)         62.0
Spending Score (1-100)     50.0
dtype: float64
Dataset with empty values! :
   CustomerID   Age  Annual Income (k$)  Spending Score (1-100)
0           1  19.0                15.0                    39.0
1           2   NaN                15.0                    81.0
2           3  20.0                 NaN                     6.0
3           4  23.0                16.0                    77.0
4           5  31.0                17.0                     NaN
5           6  22.0                 NaN                    76.0
6           7  35.0                18.0                     6.0
7           8  23.0                18.0                    94.0
8           9  64.0                19.0                     NaN
9          10  30.0                19.0                    72.0
Dataset that has been processed Handling Missing Values with Median :
   CustomerID   Age  Annual Income (k$)  Spending Score (1-100)
0           1  19.0                15.0                    39.0
1           2  36.0                15.0                    81.0
2           3  20.0                62.0                     6.0
3           4  23.0                16.0                    77.0
4           5  31.0                17.0                    50.0
5           6  22.0                62.0                    76.0
6           7  35.0                18.0                     6.0
7           8  23.0                18.0                    94.0
8           9  64.0                19.0                    50.0
9          10  30.0                19.0                    72.0
```