Student name: Aditya Suhag

Student ID: 224001686

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

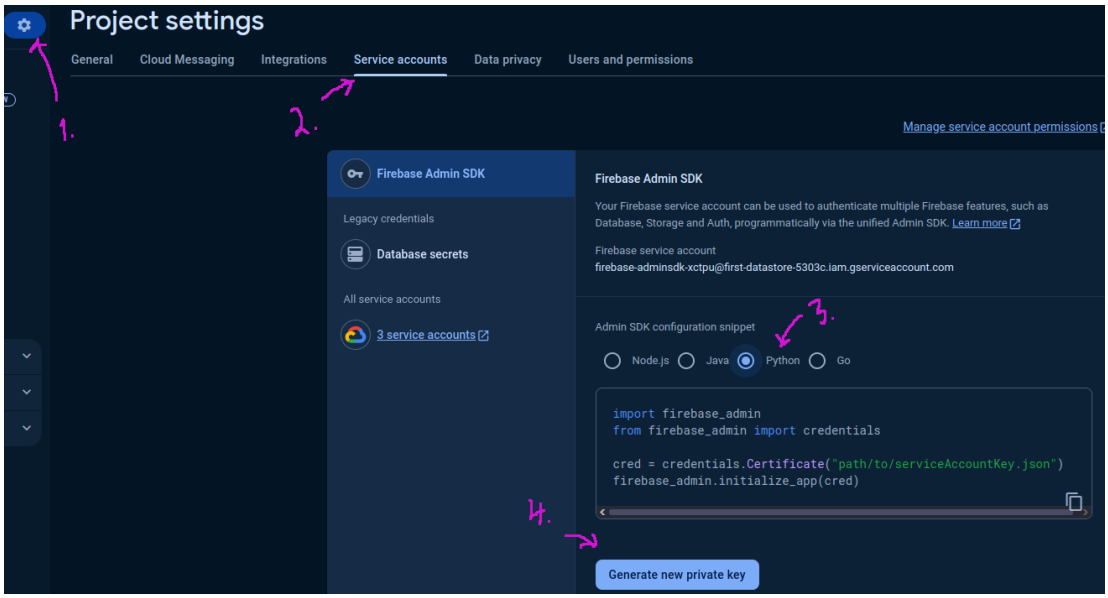
No hardware is required.

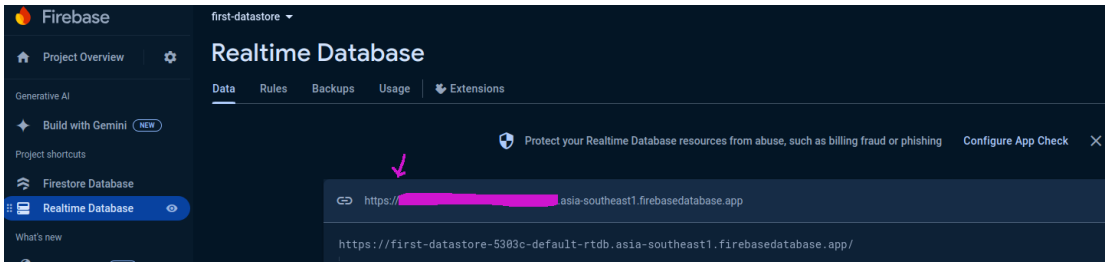
Software Required

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.



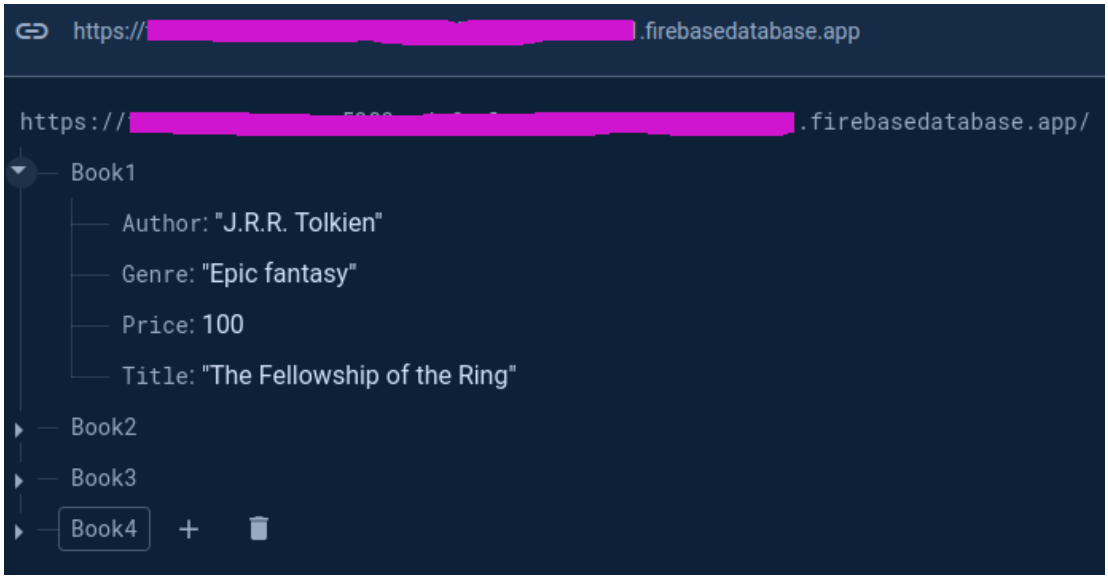
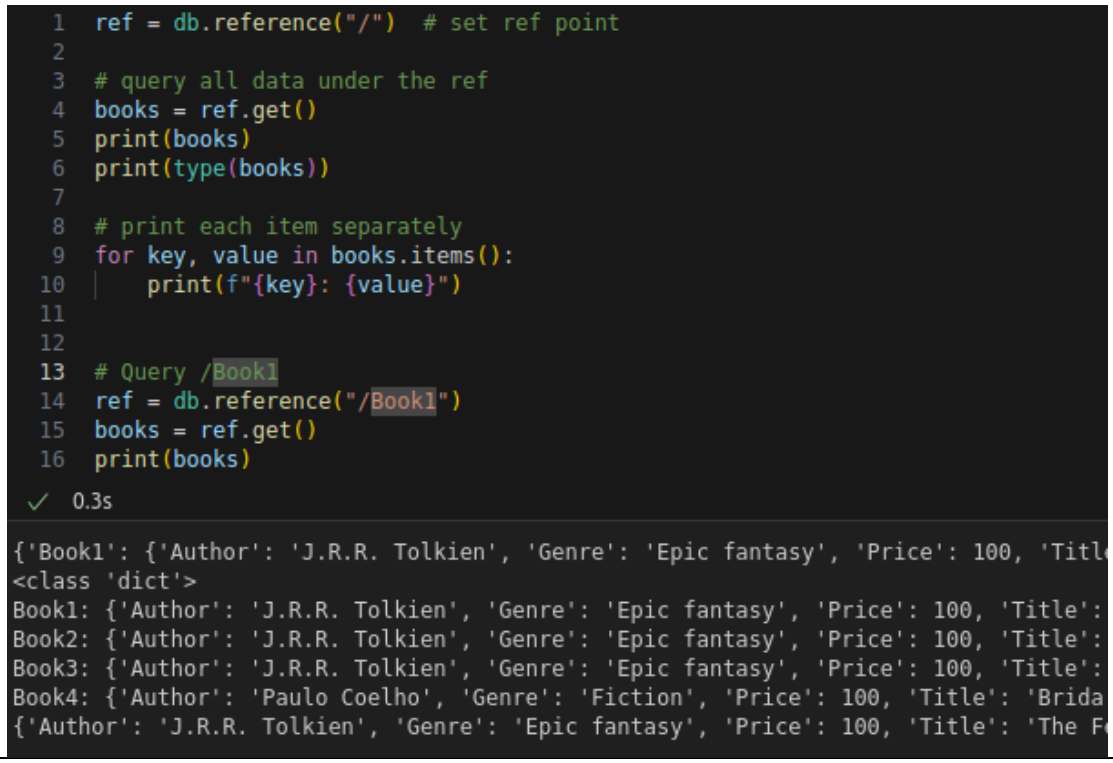
If you compile the code snippet above, it should do with no error.

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.

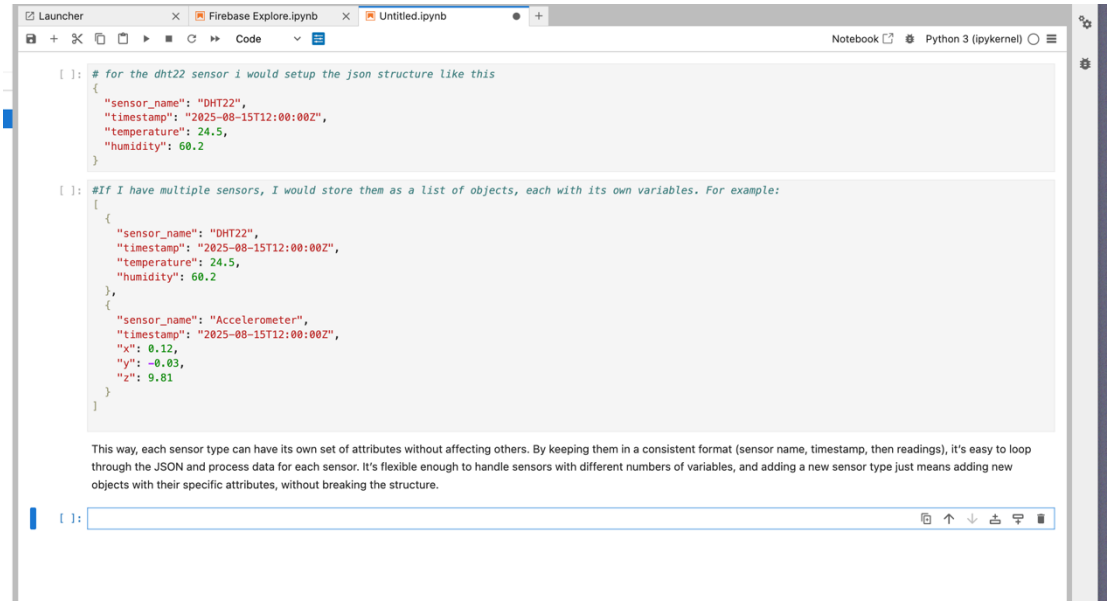
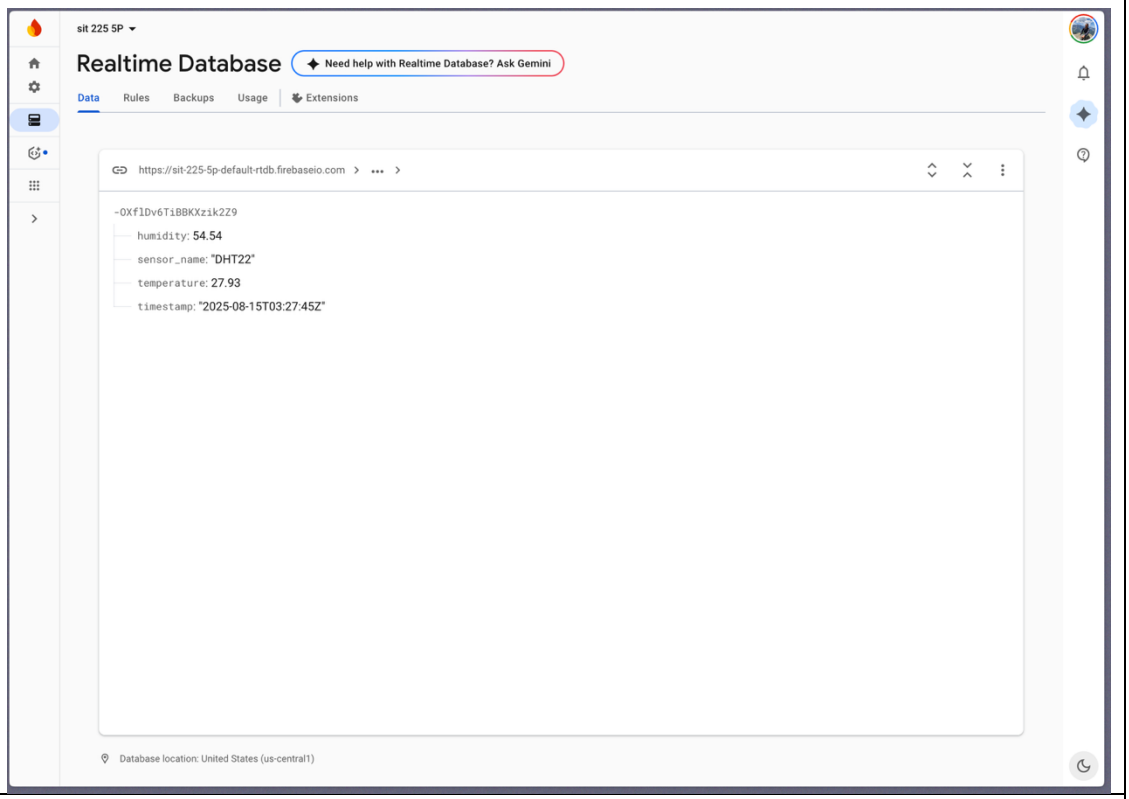
8 **Update data**:
Let's say the price of the books by J. R. R. Tolkien is reduced to 80 units to offer a discount. The first 3 books are written by this author, and we want to apply for a discount on all of them.

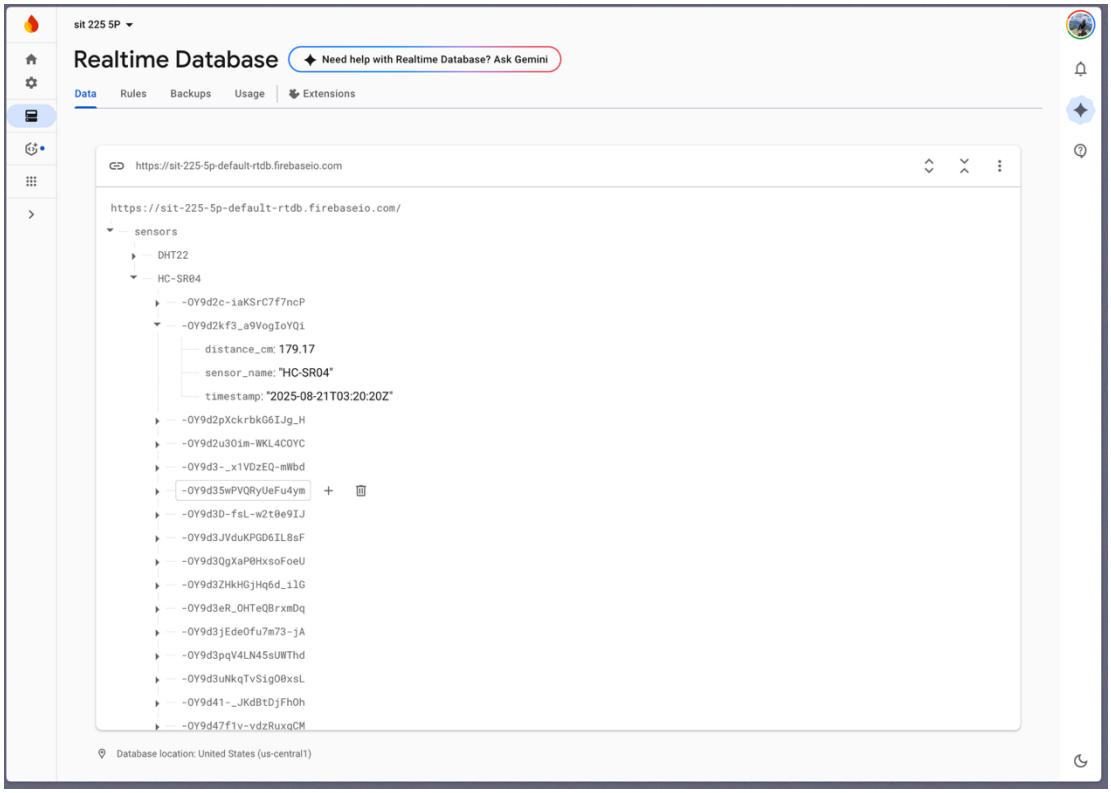
```
1  # Update data
2  #
3  # Requirement: The price of the books by
4  # J. R. R. Tolkien is reduced to 80 units to
5  # offer a discount.
6  #
7  ref = db.reference("/Books/Best_Sellers/")
8  best_sellers = ref.get()
9  print(best_sellers)
10 for key, value in best_sellers.items():
11     if(value["Author"] == "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: 
11	Question: Generate some random data for DHT22 sensor, insert data to database, query all data and screenshot the output here. Answer: 
12	Question: Generate some random data for the SR04 Ultrasonic sensor, insert data to database, query all data and screenshot the output here.

	Answer: 
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: The Realtime Database of Firebase produces events whenever any data is created, updated, or deleted, and the application reacts to these changes instantaneously. Such events are useful when sensors, for example, Arduino devices, send readings directly into the database, and we want a centralized monitoring system to report on updates in real-time. Using the **Firebase Python Admin SDK**, we can add a listener to a particular path within the database so that whenever new data comes in or some existing data changes, the callback of the listener is invoked automatically. Potentially, we can manipulate or display the data view in the callback on a live dashboard or write it to a CSV file for further analysis. This kind of approach eliminates the need to poll the database all the time while recording the results of sensor data in real time.

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

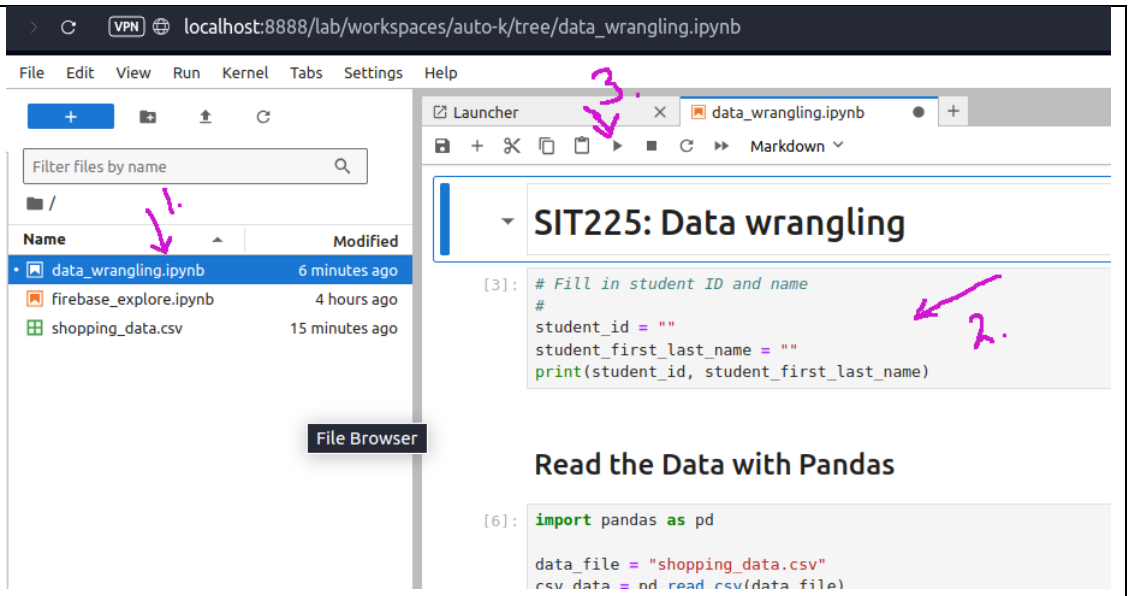
No hardware is required.

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, we may definitely use Pandas to read in the sensor CSV data we generated earlier. The use of Pandas makes it straightforward to load the file and proceed with manipulations, but some minor changes may be required. One need, when reading the CSV, is to ensure the timestamp column is parsed properly so it becomes a datetime object, thereby facilitating sorting and analysis later on. We may then wish to fill or drop any missing or invalid sensor values using `fillna()` as one option, dependent on the analytic objectives or dataset requirements. Lastly, make sure any numeric columns like temperature, humidity, or distance are in the appropriate format so any further calculations work. These small modifications will clean the data and

	make it easier to analyze when using Pandas for visualization or further processing.
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: From the Handling Missing Values section, we learned about different ways to handle missing data using Pandas. There are certain major approaches based on the situation. If there are only very few missing values and they are insignificant, we can simply drop corresponding rows or columns using dropna(). In case there are many, and their presence in data is important, we can go ahead and fill them with some constant value using fillna(value). For example, you set all missing humidity values to 0. Another common approach is imputation using mean, median, or mode. So, we use fillna(data.mean()) when it is numeric data with a normal distribution, fillna(data.median()) when it is data with outliers, and fillna(data.mode()[0]) when it is categorical or repeated values. While considering time series data, often ffill (forward fill) or bfill (backward fill) methods are used to fill missing values by carrying the last known value backward or forward. This method would vary by the kind of data and the pattern of missing values, so one's choice will be very important to maintain data quality and accuracy.

```
# Fill in student ID and name
#
student_id = "224001686"
student_first_last_name = "Aditya_Suhag"
print(student_id, student_first_last_name)
```

```
224001686 Aditya_Suhag
```

```
import firebase_admin
from firebase_admin import credentials

databaseURL = 'https://sit-225-5p-default-rtdb.firebaseio.com/'
service_account_dict = {
"type": "service_account",
  "project_id": "sit-225-5p",
  "private_key_id": "9830377238ccdf1ed2558cd7cfad0cdeeeaec253",
  "private_key": "-----BEGIN PRIVATE KEY-----\
nMIIEvQIBADANBgkqhkiG9w0BAQEFAASCBKcwggSjAgEAAoIBAQCQ6QdvH8fyZS2L\
nNhTg9Bx19oLPSrDHpjDZalVth+/Zzs3Tc0Ti/ze22iFKN9MygHkBNFloL84+yIcw\
nzOVNFYGdijcosDFQNnQdvzZyPw3q7k/KSW6uBSd2fMEQ2jiy4J1Gc7qzyasF5iky\
numzCOHr34d3i5Tx7kXvVnHx3xaYOoSc7JyqcUkHmaXi1Kc/8JJofmzrU6FgrEekW\
nf0shWgYJ4b/Tr7YBiovpl2M1nTE6fhgxAUUPLTbszPlb1D/B0tvwZPrp+36xSuWF\
nYnQd13PjAyvJA0Dsv9bZd40Mxp1UoWD+0aS15t6hFUlp+nAklCgQZkX4Vm2HgwLf\
nEQV2R+1dAgMBAAECggEAH/DVXCefrWyi0MwPZxQ3j8LRy3o1pBQcMVgQU2/CrPo6\
nARENzncd4zaKXu8WJMwPn/XXfTEIX2tY7SdEpayHYN+9z8CiZgFSr9ndWOpQGYgE\
n21w7Qq1NKP9PmPTYyYjb8dMBhK5/fXHhg9Z2sYq8hBXrkB8iKUSEjK2M++N7ej6z\
nTQ1UGrgm487De46uzqTz6JfVQcqGPVw/vPJ8ChJCclV0q0yMyOHPafZoaLaKr2yM\
nQmlgIQ5n6IIBSm/E3qhA+b+EzY4joaRkpUAevICLefk6RHY//yZKPohcV84AHXzT\
nV+vKfVmV5WO5tCamwwT0vaK2GaAen7Ro6KcJM3Gr8QKBgQDH2wyOVrrrj8Mvs8Yq\
nFEt5zYen/l26WCUz4lk5IUWVvAp5sstVMAoruYiwO/wTK/UmNt9g3RR6V4DlNjas\
nu282wUdYO9y0asvgQ9m2BY9dkS8SI7HgUkSMO6mpgFylIYsSaeadMwxIjIo11L/j\
na7aaPIaJz35w4bMADEtv9py4CQKBgQC5nn2+Sbc+Tvble6qnHSIIYK8Mp8cOW/nq\
n7H7f2L/58o7IbqbCEV58h2S9myWExdhM1BlbjZfz/etUh/zykHvpskicIXIfnlVw\
nrI0hAb5M+25j/oCf1ieVosg/JkJvA/UFA6QybASb7crdSRt8JjR5iNmCroz3V9hP\
nybVGD6kXtQKBgQCv0DaSfR/AnF6z9y1r1SC0jSc6lR11EZfgtc2R4YAqghqhBrWz\
nQiigYH4rswlFAFWQAZOuza33FkVQ3sloWRc0DwGpUDgPCyhBKM88PlXplQ/YoEgZ\
ns6WTy+SsNTzLE+AZGcqDHeY/aEspofgQo7AmrPoZ/8wWl6Z1c1tlqoghqQKBgF6F\
ni+jDpxGOqoGPqQYyeqsjeOCL63yvb6OmzLD2skP4tabxWe/HYoMteypmgAONsCjn\
ndQY/I1r3Ocgw+6kV1FPcGseO7IuXWFDpPXDKb3rKKpu29Q7vwfacDzbBJ+FadYIZ\
nKZsXkH1K6RwoZ1ugJOaivoiilFw8GoX+5FWKEgRlAoGABqpv4CGxzXfvgsoFo7e3\
nSlR0D4oL7dPwM6FJKnfC//8Khif2/JSxIW4GvuxssHe122bCs7vMoWnQNRskhHDj\
nOrvwZEFRoF2v2eT0dmKQ5FHK36vKsnknT4i0xO4gYiSOcxTp1mE1OG9x9OGrg9t5\
nGJEDjLu+2solG52bPtzecZo=\n-----END PRIVATE KEY-----\n",
  "client_email": "firebase-adminsdk-fbsvc@sit-225-
5p.iam.gserviceaccount.com",
  "client_id": "112142944362554566299",
  "auth_uri": "https://accounts.google.com/o/oauth2/auth",
  "token_uri": "https://oauth2.googleapis.com/token",
  "auth_provider_x509_cert_url":
"https://www.googleapis.com/oauth2/v1/certs",
```

```
  "client_x509_cert_url":
"https://www.googleapis.com/robot/v1/metadata/x509/firebase-adminsdk-
fbsvc%40sit-225-5p.iam.gserviceaccount.com",
  "universe_domain": "googleapis.com"}
cred_obj = credentials.Certificate(service_account_dict)
firebase_admin.initialize_app(cred_obj, {
    'databaseURL': databaseURL
})
```

```
<firebase_admin.App at 0x10832c050>
```

```
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)

ref = db.reference("/")  # set ref point

# query all data under the ref
books = ref.get()
print(books)
print(type(books))

# print each item separately
for key, value in books.items():
    print(f"{key}: {value}")


# Query /Book1
ref = db.reference("/Book1")
books = ref.get()
print(books)
```

```
{'Book1': {'Author': 'J.R.R. Tolkien', 'Genre': 'Epic fantasy',
'Price': 100, 'Title': 'The Fellowship of the Ring'}, 'Book2':
{'Author': 'J.R.R. Tolkien', 'Genre': 'Epic fantasy', 'Price': 100,
'Title': 'The Two Towers'}, 'Book3': {'Author': 'J.R.R. Tolkien',
'Genre': 'Epic fantasy', 'Price': 100, 'Title': 'The Return of the
King'}, 'Book4': {'Author': 'Paulo Coelho', 'Genre': 'Fiction',
'Price': 100, 'Title': 'Brida'}}
<class 'dict'>
Book1: {'Author': 'J.R.R. Tolkien', 'Genre': 'Epic fantasy', 'Price':
100, 'Title': 'The Fellowship of the Ring'}
Book2: {'Author': 'J.R.R. Tolkien', 'Genre': 'Epic fantasy', 'Price':
100, 'Title': 'The Two Towers'}
Book3: {'Author': 'J.R.R. Tolkien', 'Genre': 'Epic fantasy', 'Price':
100, 'Title': 'The Return of the King'}
Book4: {'Author': 'Paulo Coelho', 'Genre': 'Fiction', 'Price': 100,
'Title': 'Brida'}
{'Author': 'J.R.R. Tolkien', 'Genre': 'Epic fantasy', 'Price': 100,
'Title': 'The Fellowship of the Ring'}
```

```
# Write using push() function
# Note that a set() is called on top of push()
#
ref = db.reference("/")
ref.set({
      "Books":
      {
            "Best_Sellers": -1
      }
```

```
})

ref = db.reference("/Books/Best_Sellers")

for key, value in data.items():
     ref.push().set(value)

# Update data
#
# Requirement: The price of the books by
# J. R. R. Tolkien is reduced to 80 units to
# offer a discount.
#
ref = db.reference("/Books/Best_Sellers/")
best_sellers = ref.get()
print(best_sellers)
for key, value in best_sellers.items():
     if(value["Author"] == "J.R.R. Tolkien"):
          value["Price"] = 90
          ref.child(key).update({"Price":80})
```

```
{'-OXfWsFtvX85siQtxGrK': {'Author': 'J.R.R. Tolkien', 'Genre': 'Epic
fantasy', 'Price': 100, 'Title': 'The Fellowship of the Ring'}, '-
OXfWsPOY_b14Fx9OdPz': {'Author': 'J.R.R. Tolkien', 'Genre': 'Epic
fantasy', 'Price': 100, 'Title': 'The Two Towers'}, '-OXfWshL-
iumEMjOltiO': {'Author': 'J.R.R. Tolkien', 'Genre': 'Epic fantasy',
'Price': 100, 'Title': 'The Return of the King'}, '-
OXfWswVR8zOaGI79eMP': {'Author': 'Paulo Coelho', 'Genre': 'Fiction',
'Price': 100, 'Title': 'Brida'}}
```

```
# Let's delete all best seller books
# with J.R.R. Tolkien as the author.
#
ref = db.reference("/Books/Best_Sellers")

for key, value in best_sellers.items():
     if(value["Author"] == "J.R.R. Tolkien"):
          ref.child(key).set({})

# Delete all best_seller data.
#
ref = db.reference("/Books/Best_Sellers/")
best_sellers = ref.get()
print(best_sellers)
print(type(best_sellers))
```

```
{'-OXfWswVR8zOaGI79eMP': {'Author': 'Paulo Coelho', 'Genre':
'Fiction', 'Price': 100, 'Title': 'Brida'}}
<class 'dict'>
```

```
ref = db.reference("/Books/Best_Sellers")
ref.set({})
```

SIT225: Data wrangling

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

```
# Fill in student ID and name
#
student_id = "224001686"
student_first_last_name = "Aditya_Suhag"
print(student_id, student_first_last_name)
```

```
224001686 Aditya_Suhag
```

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.

```
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.

```
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.

```
# 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 row
# (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

```
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.

```
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

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

```
False
```

```
# We will use another data source with missing values to practice this
part.
data_missing = pd.read_csv("Shopping Data Missing Values.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
```

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

```
print(data_missing.mean())

"""

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

Answer: The error happens because data_missing most likely contains
both numeric and non-numeric columns (like strings or object types).
Pandas tries to calculate the mean for all columns, but it fails on
non-numeric columns since mean can only be computed for numbers.The
```

```
solution for this
is to only calculate mean for the numerical values.

"""

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[12], 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 /opt/anaconda3/lib/python3.13/site-packages/pandas/core/frame.py:11693, in DataFrame.mean(self, axis, skipna, numeric_only, **kwargs)
  11685 @doc(make_doc("mean", ndim=2))
  11686 def mean(
  11687     self,
   (...)
  11691     **kwargs,
  11692 ):
> 11693     result = super().mean(axis, skipna, numeric_only, **kwargs)
  11694     if isinstance(result, Series):
  11695         result = result.__finalize__(self, method="mean")

File /opt/anaconda3/lib/python3.13/site-packages/pandas/core/generic.py:12420, in NDFrame.mean(self, axis, skipna, numeric_only, **kwargs)
  12413 def mean(
  12414     self,
  12415     axis: Axis | None = 0,
   (...)
  12418     **kwargs,
  12419 ) -> Series | float:
> 12420     return self._stat_function(
  12421         "mean", nanops.nanmean, axis, skipna, numeric_only, **kwargs
  12422     )

File /opt/anaconda3/lib/python3.13/site-packages/pandas/core/generic.py:12377, in NDFrame._stat_function(self, name, func, axis, skipna,
```

```
numeric_only, **kwargs)
  12373 nv.validate_func(name, (), kwargs)
  12375 validate_bool_kwarg(skipna, "skipna", none_allowed=False)
> 12377 return self._reduce(
  12378     func, name=name, axis=axis, skipna=skipna,
numeric_only=numeric_only
  12379 )

File
/opt/anaconda3/lib/python3.13/site-packages/pandas/core/frame.py:11562
, in DataFrame._reduce(self, op, name, axis, skipna, numeric_only,
filter_type, **kwds)
  11558     df = df.T
  11560 # After possibly _get_data and transposing, we are now in the
  11561 #  simple case where we can use BlockManager.reduce
> 11562 res = df._mgr.reduce(blk_func)
  11563 out = df._constructor_from_mgr(res, axes=res.axes).iloc[0]
  11564 if out_dtype is not None and out.dtype != "boolean":

File
/opt/anaconda3/lib/python3.13/site-packages/pandas/core/internals/
managers.py:1500, in BlockManager.reduce(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
/opt/anaconda3/lib/python3.13/site-packages/pandas/core/internals/
blocks.py:404, in Block.reduce(self, func)
    398 @final
    399 def reduce(self, func) -> list[Block]:
    400     # We will apply the function and reshape the result into a
single-row
    401     #  Block with the same mgr_locs; squeezing will be done at
a higher level
    402     assert self.ndim == 2
--> 404     result = func(self.values)
    406     if self.values.ndim == 1:
    407         res_values = result

File
/opt/anaconda3/lib/python3.13/site-packages/pandas/core/frame.py:11481
, in DataFrame._reduce.<locals>.blk_func(values, axis)
  11479         return np.array([result])
  11480 else:
> 11481     return op(values, axis=axis, skipna=skipna, **kwds)

File
```

```
/opt/anaconda3/lib/python3.13/site-packages/pandas/core/nanops.py:147,
in bottleneck_switch.__call__.<locals>.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
/opt/anaconda3/lib/python3.13/site-packages/pandas/core/nanops.py:404,
in _datetimelike_compat.<locals>.new_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,
**kwargs)
    406 if datetimelike:
    407     result = _wrap_results(result, orig_values.dtype,
fill_value=iNaT)

File
/opt/anaconda3/lib/python3.13/site-packages/pandas/core/nanops.py:720,
in nanmean(values, axis, skipna, mask)
    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
/opt/anaconda3/lib/python3.13/site-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
['MaleMaleFemaleFemaleFemaleFemaleFemaleFemaleMaleFemaleMaleFemaleFema
leFemaleMaleMaleFemaleMaleMaleFemaleMaleFemaleMaleFemale
MaleFemaleFemaleMaleFemaleMaleMaleFemaleFemaleFemaleFemaleFemale
FemaleMaleMaleFemaleFemaleFemaleFemaleFemaleFemaleFemaleMaleFema
leMaleFemaleMaleFemaleMaleMaleMaleFemaleFemaleMaleMaleFemale
FemaleMaleFemaleMaleFemaleFemaleFemaleMaleMaleFemaleMaleFemaleFemaleMa
leMaleMaleFemaleFemaleMaleFemaleFemaleFemaleFemaleFemaleMaleMaleFemale
FemaleMaleFemaleFemaleMaleMaleFemaleFemaleMaleMaleMaleFemaleFemaleMale
MaleMaleMaleFemaleFemaleMaleFemaleFemaleFemaleFemaleFemaleFemaleMaleFe
maleFemaleMaleFemaleFemaleMaleMaleMaleMaleMaleMaleFemaleFemaleMaleFema
```

```
leFemaleMaleMaleFemaleFemaleMaleFemaleFemaleMaleMaleMaleFemaleFemaleMa
leMaleMaleFemaleFemaleFemaleFemaleMaleFemaleMaleFemaleFemaleFemaleMale
FemaleMaleFemaleMaleFemaleFemaleMaleMaleMaleMaleFemaleFemaleMaleMa
leMaleMaleFemaleFemaleMaleFemaleFemaleFemaleMaleFemaleMaleFemaleFemale
FemaleMaleFemaleFemaleFemaleFemaleMaleMaleMale'] to numeric
```

```
# 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
```

```
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
```

```
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.

```
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.media
n())
print("Dataset that has been processed Handling Missing Values with
Median :")
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
```

```python
# collect_and_upload.py
import os
import csv
import time
from pathlib import Path
from datetime import datetime, timezone

import serial
import firebase_admin
from firebase_admin import credentials, db


PORT = "/dev/cu.usbmodem1101"
BAUD = 115200
DB_URL = "https://sit-225-5p-default-rtdb.firebaseio.com"
SERVICE_KEY = "firebase-key.json"


def ensure_firebase():
    """Initialize Firebase once per process (safe for notebooks)."""
    try:
        firebase_admin.get_app()
    except ValueError:
        if not Path(SERVICE_KEY).exists():
            raise FileNotFoundError(
                f"Service account JSON not found at: {SERVICE_KEY}"
            )
        cred = credentials.Certificate(SERVICE_KEY)
        firebase_admin.initialize_app(cred, {"databaseURL": DB_URL})

def parse_line(line: str):
    """
    Accepts either:
      - 'timestamp_ms,gx,gy,gz'
      - 'gx,gy,gz'
    Returns tuple: (dev_ms_or_None, gx, gy, gz) or None if bad line.
    """
    if not line:
        return None
    s = line.strip()
    if not s or s.startswith("#"):
        return None
    parts = [p.strip() for p in s.split(",")]
    if len(parts) == 4 and parts[0].isdigit():
        try:
            return int(parts[0]), float(parts[1]), float(parts[2]),
float(parts[3])
        except ValueError:
            return None
    elif len(parts) == 3:
```

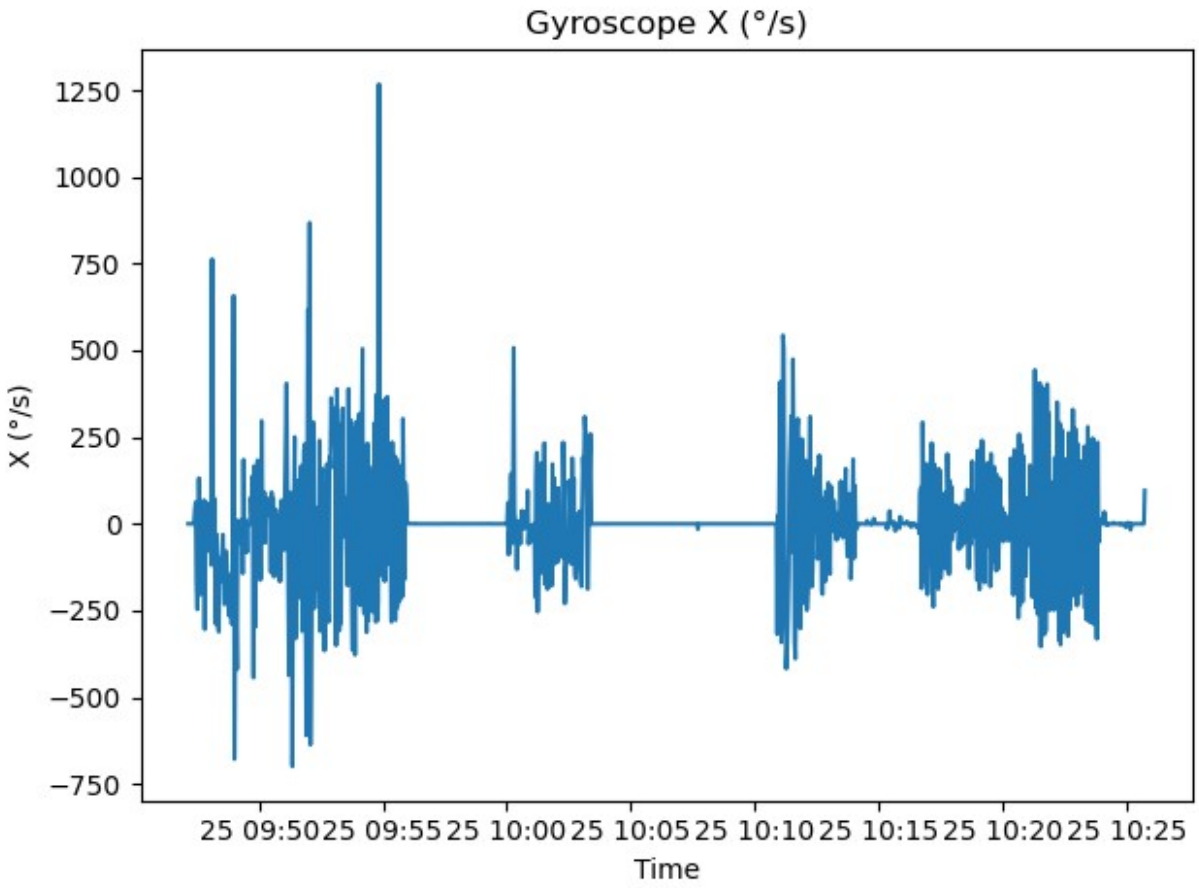
```
        try:
            gx, gy, gz = map(float, parts)
            return None, gx, gy, gz
        except ValueError:
            return None
    return None

def open_serial(port: str, baud: int, timeout: float = 1.0) ->
serial.Serial:
    try:
        ser = serial.Serial(port, baud, timeout=timeout)
        # small delay helps some OSes stabilize the port after open
        time.sleep(0.5)
        return ser
    except serial.SerialException as e:
        raise RuntimeError(
            f"Could not open serial port '{port}'. "
            f"Check Tools→Port in Arduino IDE and update PORT.\n{e}"
        )

def main():
    ensure_firebase()
    root = db.reference("/gyroscope")

    session_id = datetime.now().strftime("%Y%m%d_%H%M%S")
    session_ref =
root.child("sessions").child(session_id).child("samples")

    csv_path = Path(f"live_session_{session_id}.csv")
    csv_exists = csv_path.exists()

    # Open CSV once; write header only if file is new
    csv_file = csv_path.open("a", newline="")
    writer = csv.writer(csv_file)
    if not csv_exists:
        writer.writerow(["pc_timestamp_iso", "dev_timestamp_ms",
"x_dps", "y_dps", "z_dps"])

    print(f"[INFO] Session: {session_id}")
    print(f"[INFO] Writing local CSV: {csv_path.resolve()}")
    print("[INFO] Press Ctrl+C to stop.")

    ser = open_serial(PORT, BAUD, timeout=1)

    try:
        while True:
            raw = ser.readline().decode(errors="ignore")
            parsed = parse_line(raw)
            if not parsed:
                continue
```

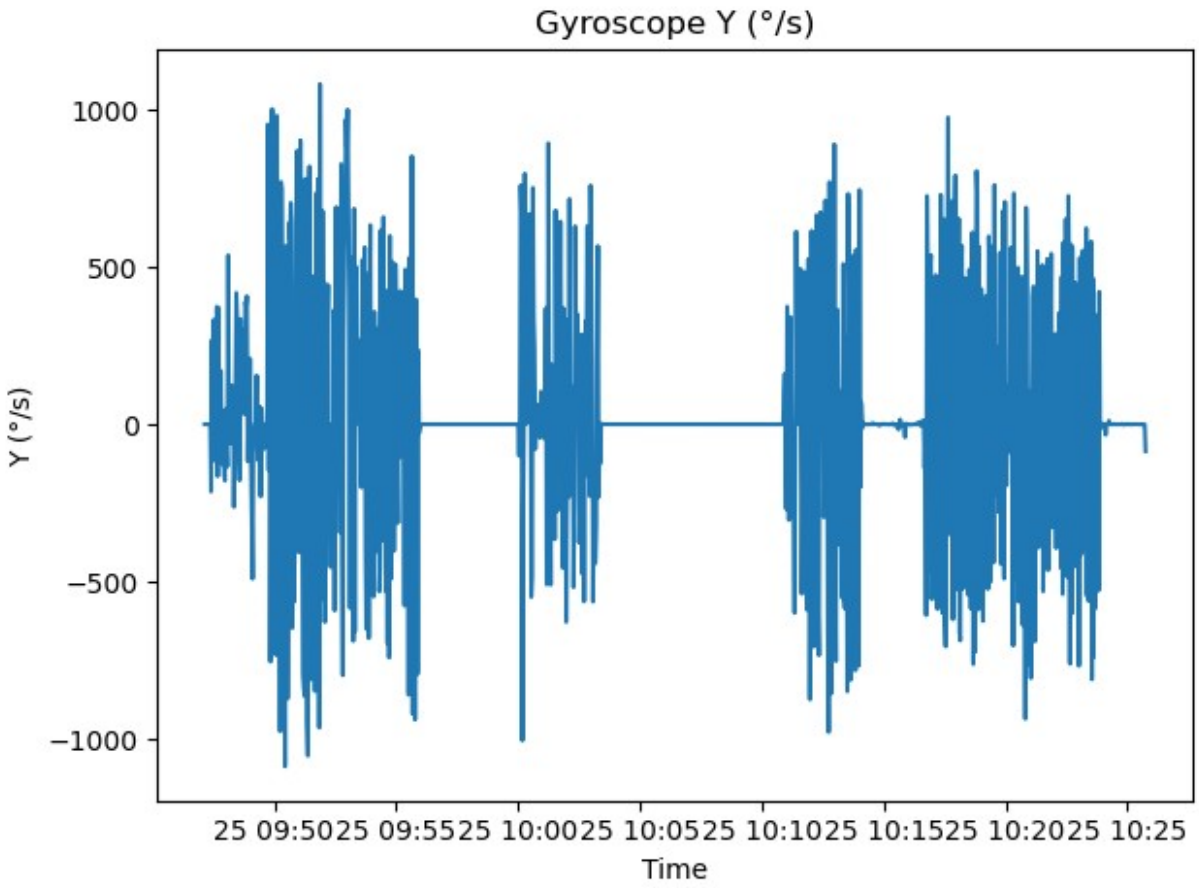
```
            dev_ms, gx, gy, gz = parsed  # dev_ms may be None if
Arduino didn't print timestamp
            now_iso = datetime.now(timezone.utc).isoformat()

            payload = {"timestamp": now_iso, "x": gx, "y": gy, "z":
gz}
            if dev_ms is not None:
                payload["dev_timestamp_ms"] = dev_ms

            # Push to Firebase
            session_ref.push(payload)

            # Append to local CSV
            writer.writerow([now_iso, dev_ms if dev_ms is not None
else "", gx, gy, gz])

            # Be nice to CPU/network; adjust if you need full
throughput
            time.sleep(0.002)

    except KeyboardInterrupt:
        print("\n[INFO] Stopped by user.")
    finally:
        try:
            ser.close()
        except Exception:
            pass
        csv_file.flush()
        csv_file.close()

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

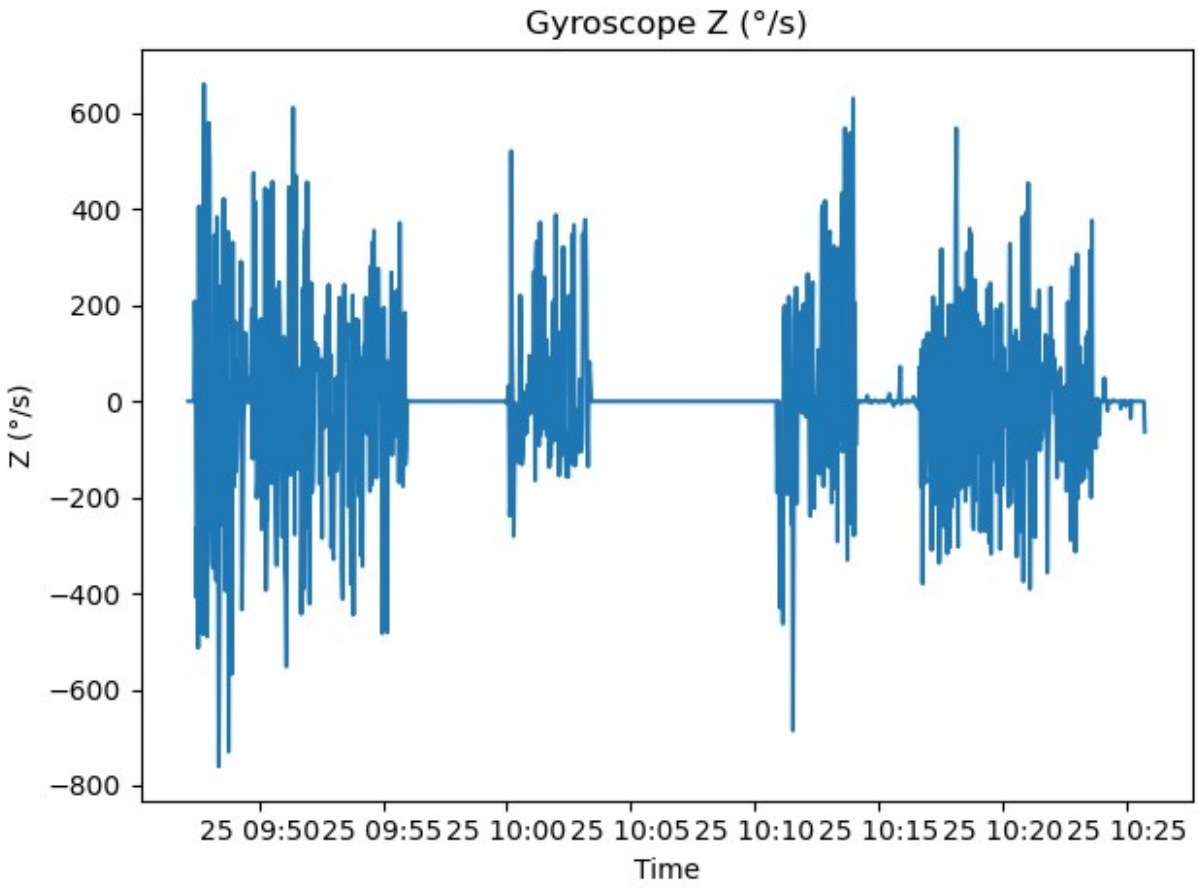
```
[INFO] Session: 20250825_194706
[INFO] Writing local CSV: /Users/adityasuhag/SIT225_2024T2/week 5
content /live_session_20250825_194706.csv
[INFO] Press Ctrl+C to stop.
```

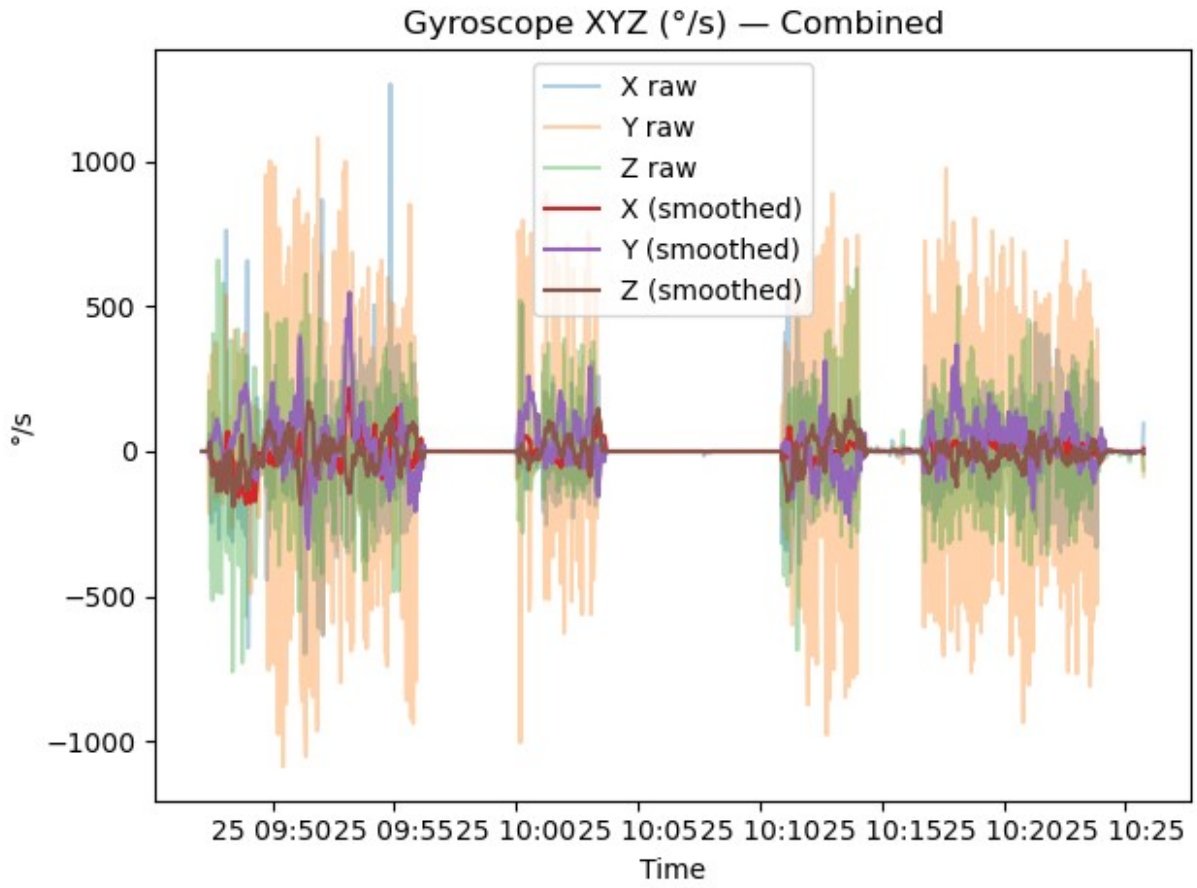
```
import glob, pandas as pd, matplotlib.pyplot as plt


paths = sorted(glob.glob("live_session_*.csv"))
assert paths, "No live_session_*.csv found."
csv_path = paths[-1]
df = pd.read_csv(csv_path)


df["pc_timestamp_iso"] = pd.to_datetime(df["pc_timestamp_iso"],
errors="coerce")
for col in ["x_dps", "y_dps", "z_dps"]:
```

```
    df[col] = pd.to_numeric(df[col], errors="coerce")
df = df.dropna(subset=["pc_timestamp_iso", "x_dps", "y_dps",
"z_dps"]).sort_values("pc_timestamp_iso")


sm = df[["x_dps","y_dps","z_dps"]].rolling(window=10,
min_periods=1).mean()

t = df["pc_timestamp_iso"]

# A) Separate plots
plt.figure()
plt.plot(t, df["x_dps"]); plt.title("Gyroscope X (°/s)");
plt.xlabel("Time"); plt.ylabel("X (°/s)")
plt.tight_layout(); plt.savefig("gyro_x.png"); plt.show()

plt.figure()
plt.plot(t, df["y_dps"]); plt.title("Gyroscope Y (°/s)");
plt.xlabel("Time"); plt.ylabel("Y (°/s)")
plt.tight_layout(); plt.savefig("gyro_y.png"); plt.show()

plt.figure()
plt.plot(t, df["z_dps"]); plt.title("Gyroscope Z (°/s)");
plt.xlabel("Time"); plt.ylabel("Z (°/s)")
plt.tight_layout(); plt.savefig("gyro_z.png"); plt.show()

# B) Combined plots
plt.figure()
plt.plot(t, df["x_dps"], alpha=0.35, label="X raw")
plt.plot(t, df["y_dps"], alpha=0.35, label="Y raw")
plt.plot(t, df["z_dps"], alpha=0.35, label="Z raw")
plt.plot(t, sm["x_dps"], label="X (smoothed)")
plt.plot(t, sm["y_dps"], label="Y (smoothed)")
plt.plot(t, sm["z_dps"], label="Z (smoothed)")
plt.title("Gyroscope XYZ (°/s) — Combined")
plt.xlabel("Time"); plt.ylabel("°/s"); plt.legend()
plt.tight_layout(); plt.savefig("gyro_xyz_combined.png"); plt.show()
```


Gyroscope X (°/s)
X (°/s)
1250
1000
750
500
250
0
−250
−500
−750
25 09:50
25 09:55
25 10:00
25 10:05
25 10:10
25 10:15
25 10:20
25 10:25
Time


Gyroscope Y (°/s)
1000
500
0
−500
−1000
Y (°/s)
25 09:5025 09:5525 10:0025 10:0525 10:1025 10:1525 10:2025 10:25
Time


Gyroscope Z (°/s)
600
400
200
0
−200
−400
−600
−800
Z (°/s)
25 09:50
25 09:55
25 10:00
25 10:05
25 10:10
25 10:15
25 10:20
25 10:25
Time



Takeaways from the graphs-

1. All graphs seperately - Looking into such individual gyroscope plots for X, Y, and Z axes, the respective axis responds differently depending on the type of movement. The X-axis tends to have big spikes during rotations or tilts in that direction, whereas during the resting period, its value comes back near-zero. The Y-axis shows the largest and most consistent swings because walking tends to cause forward-backward movements of the body that heavily influence this axis. The Z-axis shows fluctuations, especially when there might be turning or twisting movements, but with an amplitude or behavior that is mostly smaller compared to Y. Flat segments in all three graphs mark the resting phases during which almost no rotation could be observed. These observations show that the gyroscope can separate stillness from activity and also point out which axis is most heavily engaged in the kind of motion at hand.

2. All graphs together - It can be observed from the combined gyroscope graph that the sensor sensed two different states-walking and resting. During the rest states, the readings in the three axes of measurement (x, y, z) remain very near zero thereby exhibiting only minor fluctuations with the sensor mostly being stationary. On the contrary, during walking, rhythmic spikes and fluctuations are visible mainly on the y-axis, suggesting that most of the axial body rotation exists in this direction during

locomotion. The x-axis and the z-axis also show some presence of movement albeit their modifications are small, possibly associated with side-to-side sway and slight twisting. As compared to the raw noisy data, the smoothed curves provide a better avenue for observing several activity patterns. This goes to further assert that the gyroscope helps identify motion while walking and a state of rest during inactivity, as well as further noting which axis is more affected during such activity.