Student name: Truong Khang Thinh Nguyen

Student ID: 223446545

TASK 5.1P – Store Data to Cloud

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

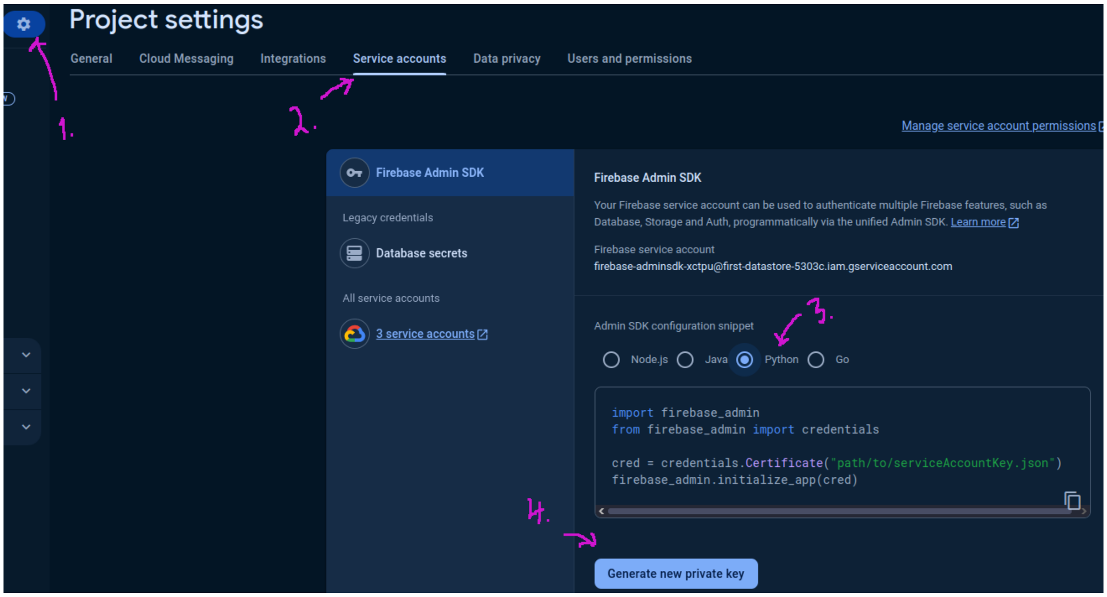
Steps

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.



Firebase Admin SDK

Your Firebase service account can be used to authenticate multiple Firebase features, such as Database, Storage and Auth, programmatically via the unified Admin SDK. Learn more

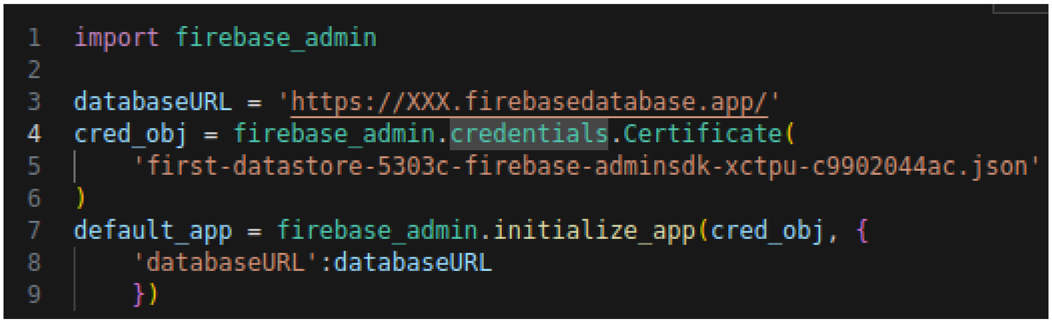
Firebase service account
firebase-adminsdk-xctpu@first-datastore-5303c.iam.gserviceaccount.com

Admin SDK configuration snippet

○ Node.js ○ Java ● Python ○ Go

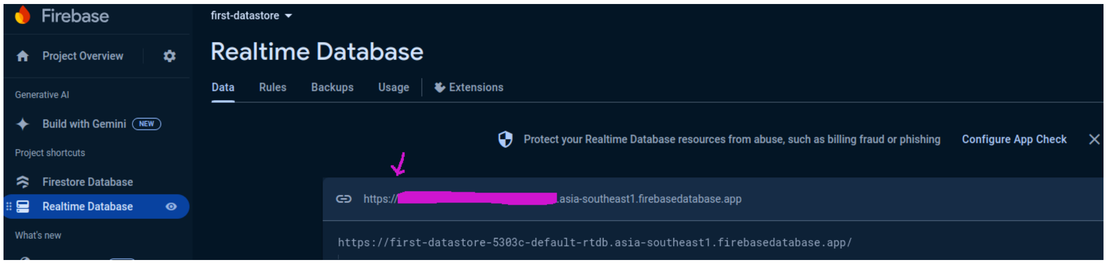
```
import firebase_admin
from firebase_admin import credentials

cred = credentials.Certificate("path/to/serviceAccountKey.json")
firebase_admin.initialize_app(cred)
```

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.



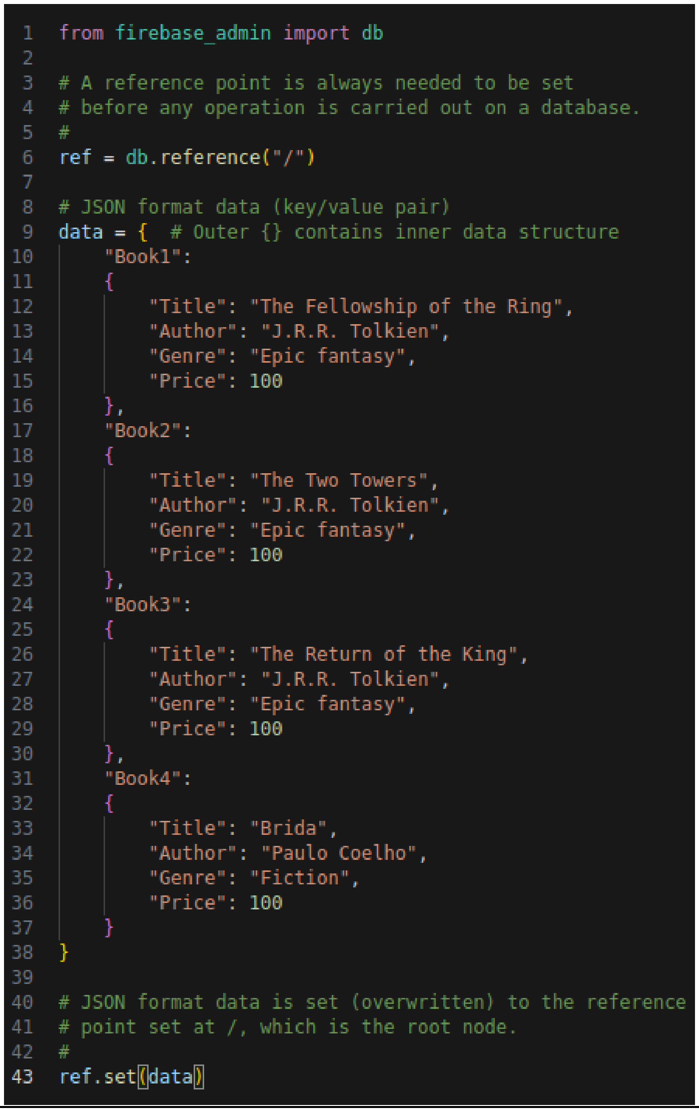
https://[redacted].asia-southeast1.firebasedatabase.app

https://first-datastore-5303c-default-rtdb.asia-southeast1.firebasedatabase.app/

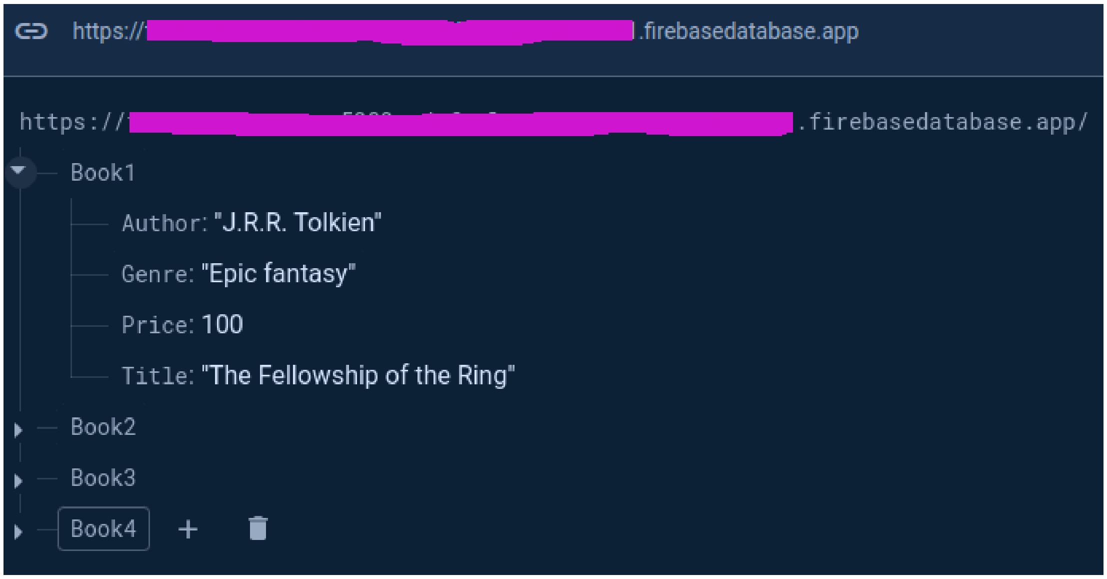
If you compile the code snippet above, it should do with no error.

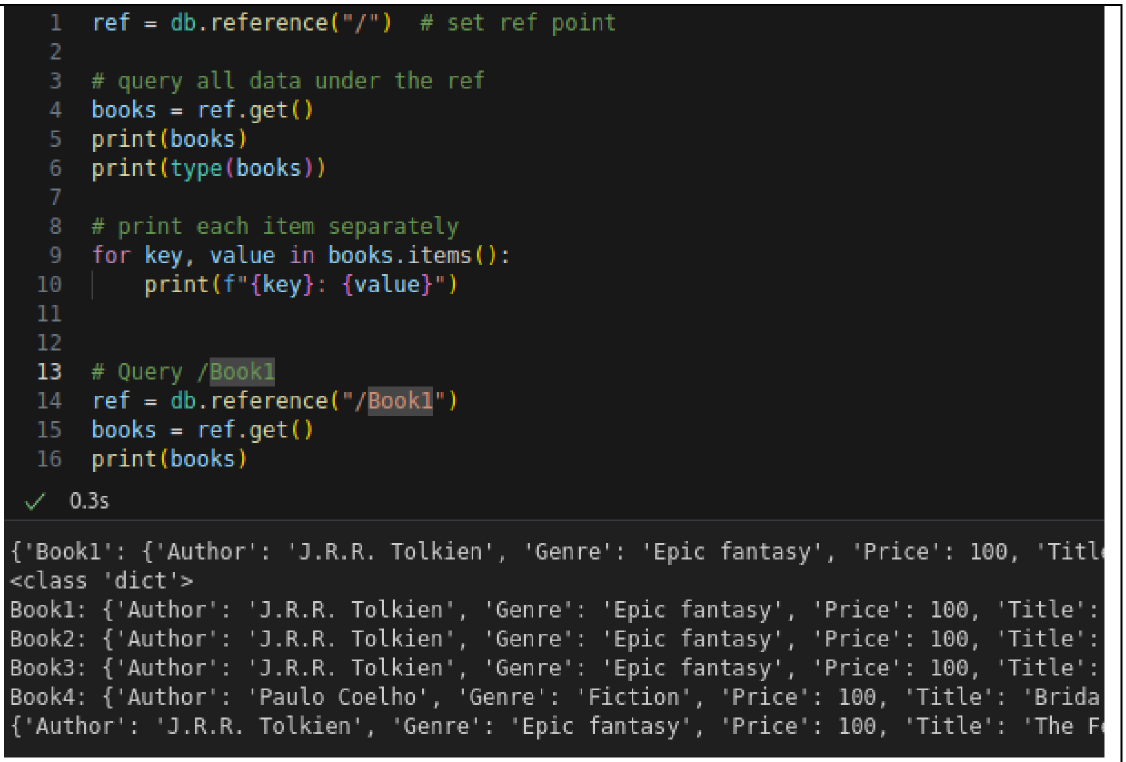
5 **Write to database Using the set() Function**:

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.



```python
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 -  https://[redacted].firebasedatabase.app https://[redacted].firebasedatabase.app/ Book1 Author: "J.R.R. Tolkien" Genre: "Epic fantasy" Price: 100 Title: "The Fellowship of the Ring" Book2 Book3 Book4
6	**Read data using get() function**: Data can be read using get() function on the reference set beforehand, as shown below.



```
 1  ref = db.reference("/")  # set ref point
 2
 3  # query all data under the ref
 4  books = ref.get()
 5  print(books)
 6  print(type(books))
 7
 8  # print each item separately
 9  for key, value in books.items():
10      print(f"{key}: {value}")
11
12
13  # Query /Book1
14  ref = db.reference("/Book1")
15  books = ref.get()
16  print(books)
```

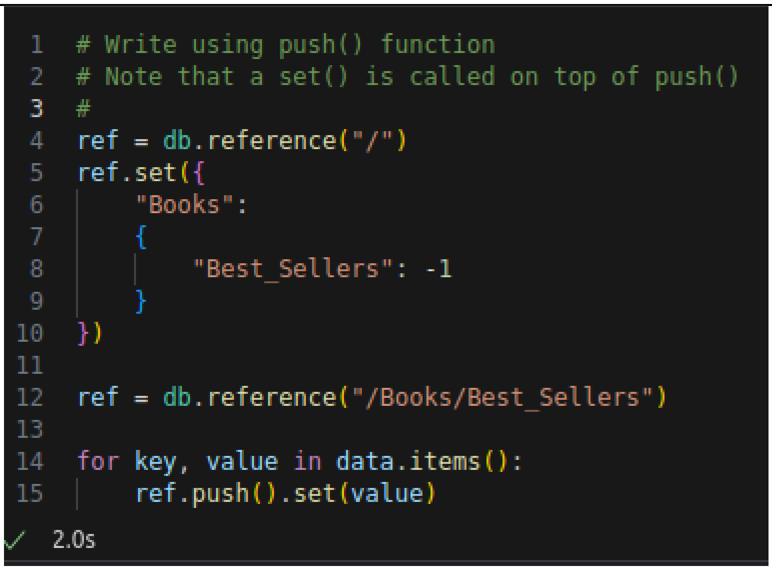
✓ 0.3s

```
{'Book1': {'Author': 'J.R.R. Tolkien', 'Genre': 'Epic fantasy', 'Price': 100, 'Titl
<class 'dict'>
Book1: {'Author': 'J.R.R. Tolkien', 'Genre': 'Epic fantasy', 'Price': 100, 'Title':
Book2: {'Author': 'J.R.R. Tolkien', 'Genre': 'Epic fantasy', 'Price': 100, 'Title':
Book3: {'Author': 'J.R.R. Tolkien', 'Genre': 'Epic fantasy', 'Price': 100, 'Title':
Book4: {'Author': 'Paulo Coelho', 'Genre': 'Fiction', 'Price': 100, 'Title': 'Brida
{'Author': 'J.R.R. Tolkien', 'Genre': 'Epic fantasy', 'Price': 100, 'Title': 'The F
```

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

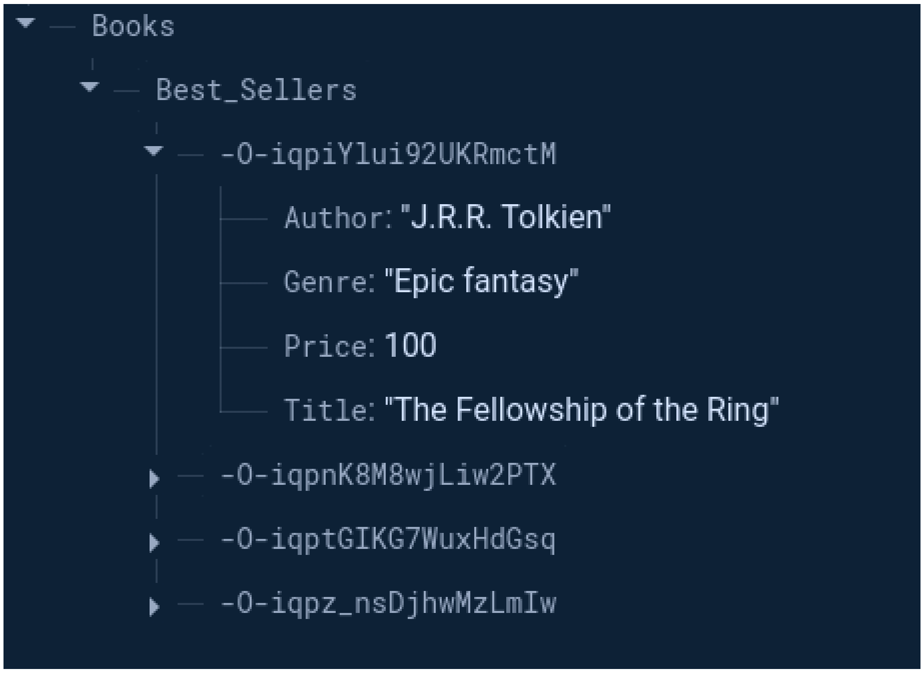
Write to database Using the push() Function:
The push() function saves data under a *unique system generated key*. This is different than set() where you set the keys such as Book1, Book2, Book3 and Book4 under which the content (author, genre, price and title) appears. Let's try to push the same data 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.



```python
# 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)
```

✓ 2.0s

The output will reset the previous data set in / node. The current data is shown below.

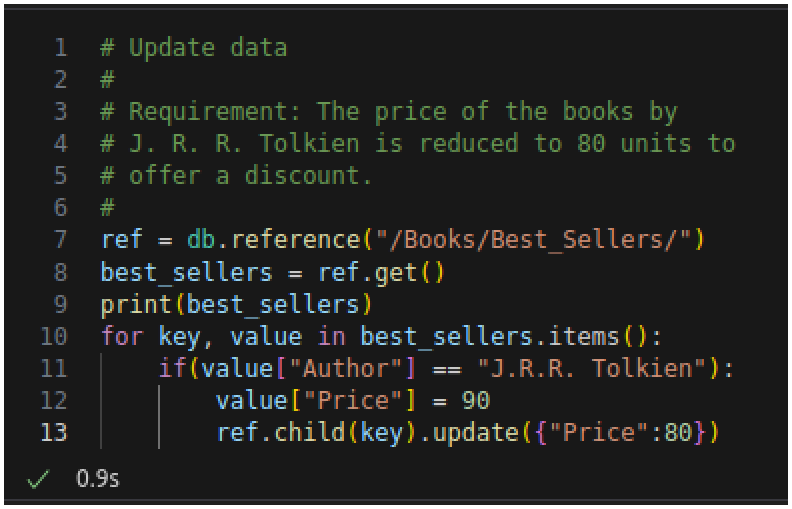


```
▼ — Books
      ▼ — Best_Sellers
            ▼ — -O-iqpiYlui92UKRmctM
                    — Author: "J.R.R. Tolkien"
                    — Genre: "Epic fantasy"
                    — Price: 100
                    — Title: "The Fellowship of the Ring"
            ▶ — -O-iqpnK8M8wjLiw2PTX
            ▶ — -O-iqptGIKG7WuxHdGsq
            ▶ — -O-iqpz_nsDjhwMzLmIw
```

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.

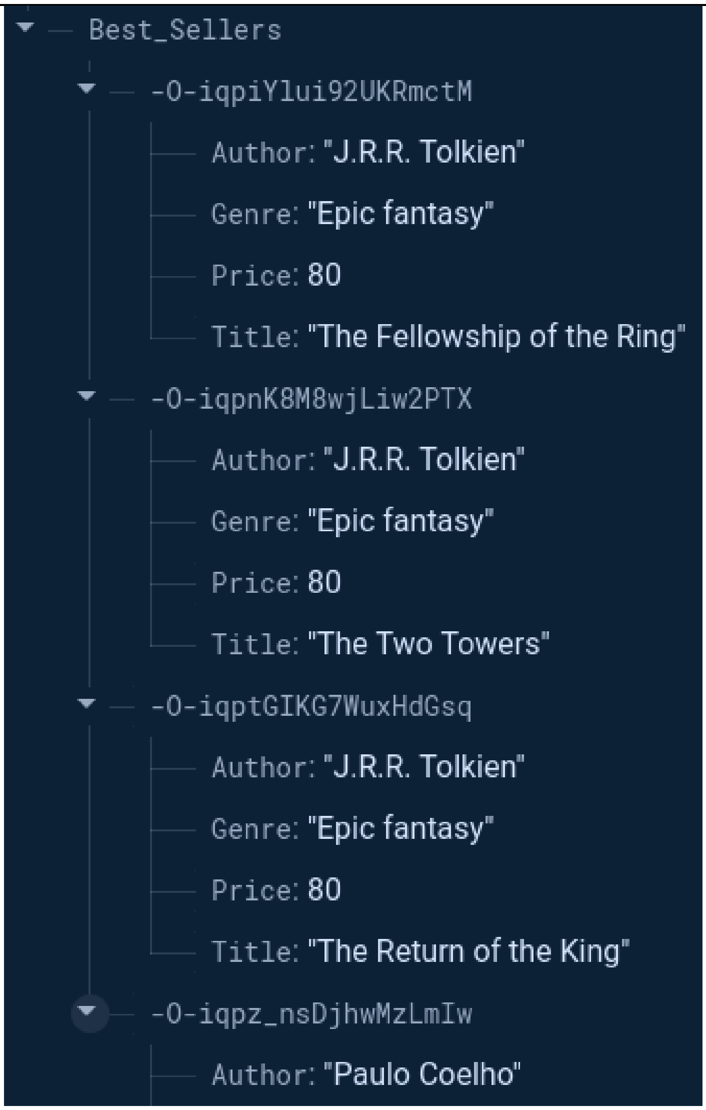
8 **Update data**:

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.

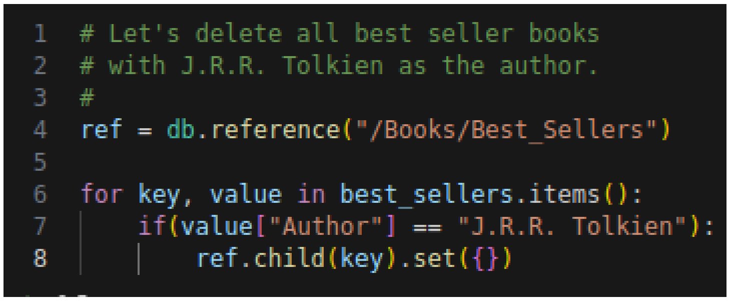


```
Best_Sellers
    -O-iqpiYlui92UKRmctM
        Author: "J.R.R. Tolkien"
        Genre: "Epic fantasy"
        Price: 80
        Title: "The Fellowship of the Ring"
    -O-iqpnK8M8wjLiw2PTX
        Author: "J.R.R. Tolkien"
        Genre: "Epic fantasy"
        Price: 80
        Title: "The Two Towers"
    -O-iqptGIKG7WuxHdGsq
        Author: "J.R.R. Tolkien"
        Genre: "Epic fantasy"
        Price: 80
        Title: "The Return of the King"
    -O-iqpz_nsDjhwMzLmIw
        Author: "Paulo Coelho"
```

9

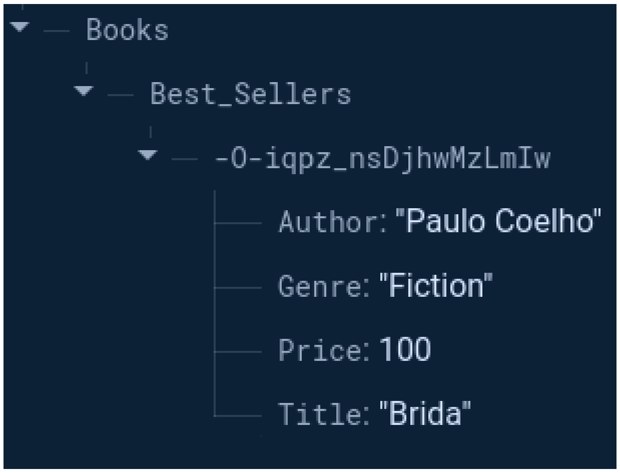
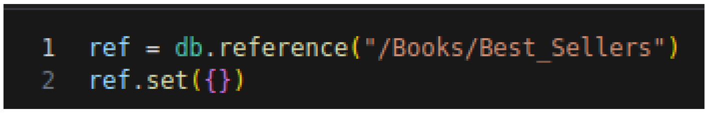
Delete data:

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.

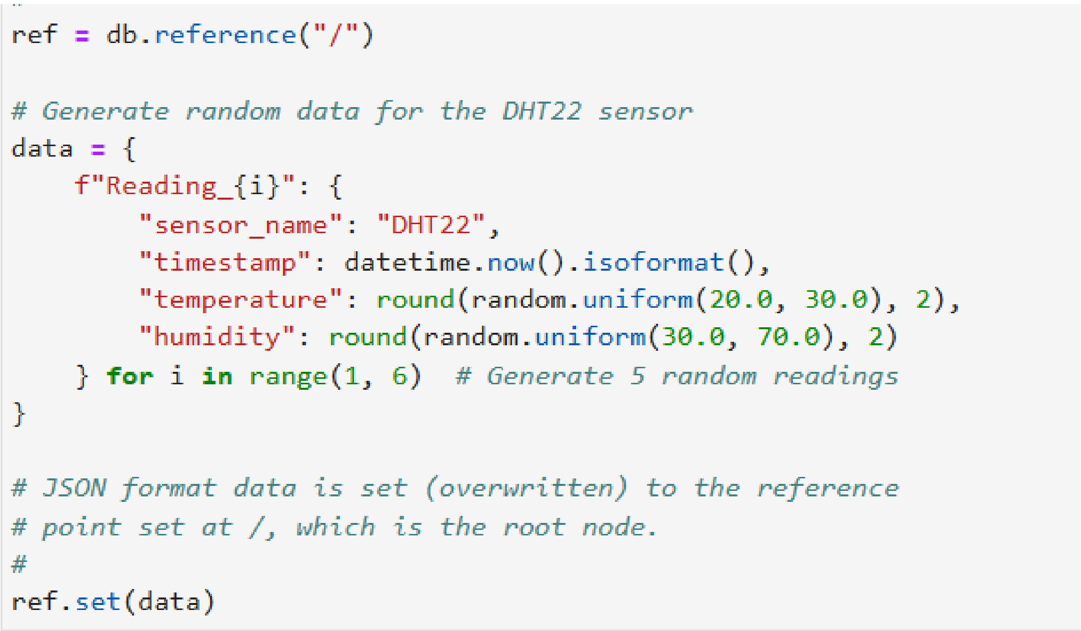
	
	Books
	Best_Sellers
	-O-iqpz_nsDjhwMzLmIw
	Author: "Paulo Coelho"
	Genre: "Fiction"
	Price: 100
	Title: "Brida"
	If ref.child() not used, as shown the code below, all data will be removed.
	
	`ref = db.reference("/Books/Best_Sellers")`
	`ref.set({})`
	Now in Firebase console you will see no data exists.

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: <Your answer> The top level objects will be the types of sensors such as DHT22, Accelerometer, Ultrasonic sensors, ... And the inside key value will be the data and its value will be smaller keys with the **timestamp** with its corresponding datetime value and for example the DHT will be the **temperature** and **humidity** values. For Accelerometer will be the **x, y, z** as well as theirs correspondig values. Adn for Ultrasonic sensor the key will be **distance** be the key and the value will be the distance it has measured.

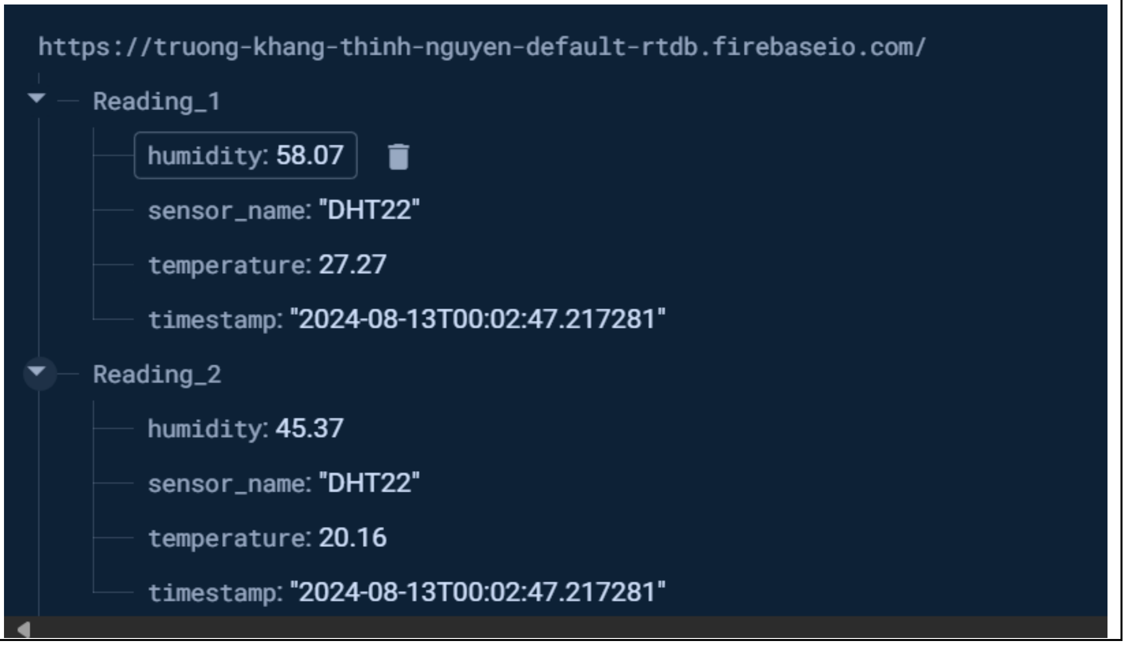
11

Question: Generate some random data for DHT22 sensor, insert data to database, query all data and screenshot the output here.

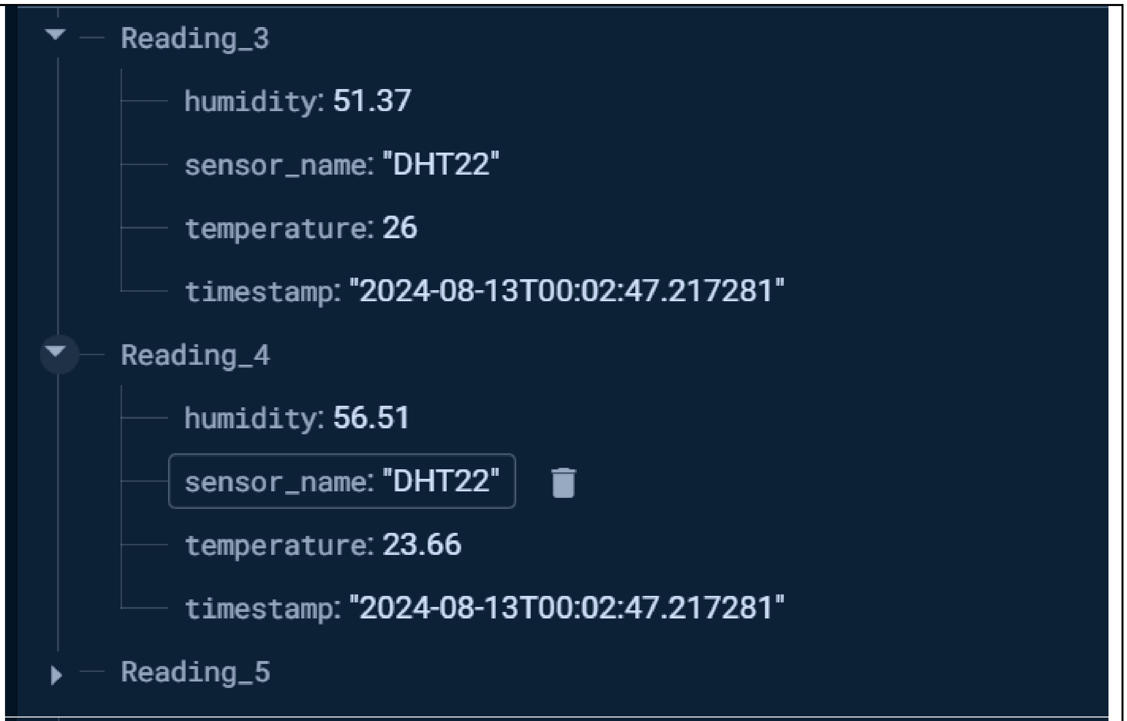
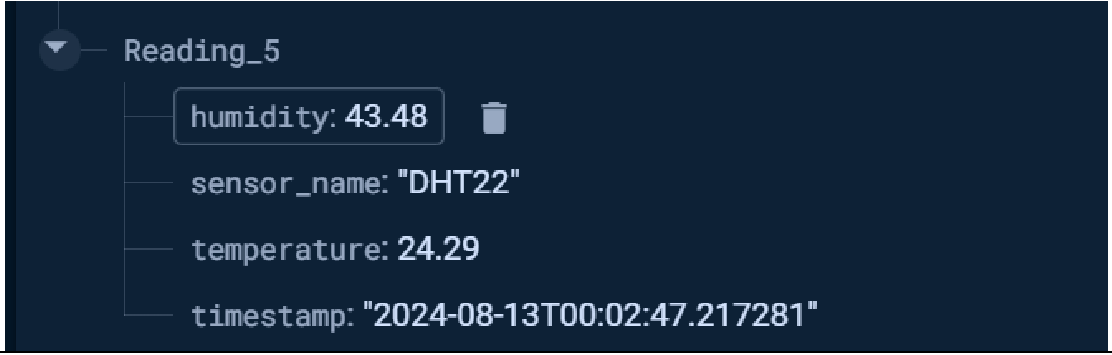
Answer: <Your answer>

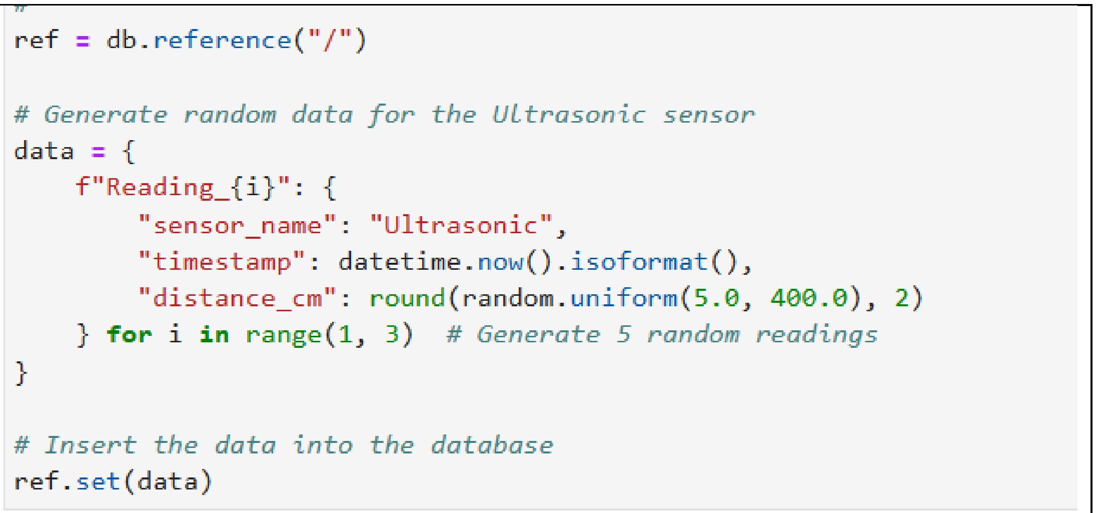


```
ref = db.reference("/")

# Generate random data for the DHT22 sensor
data = {
    f"Reading_{i}": {
        "sensor_name": "DHT22",
        "timestamp": datetime.now().isoformat(),
        "temperature": round(random.uniform(20.0, 30.0), 2),
        "humidity": round(random.uniform(30.0, 70.0), 2)
    } for i in range(1, 6)  # Generate 5 random readings
}

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

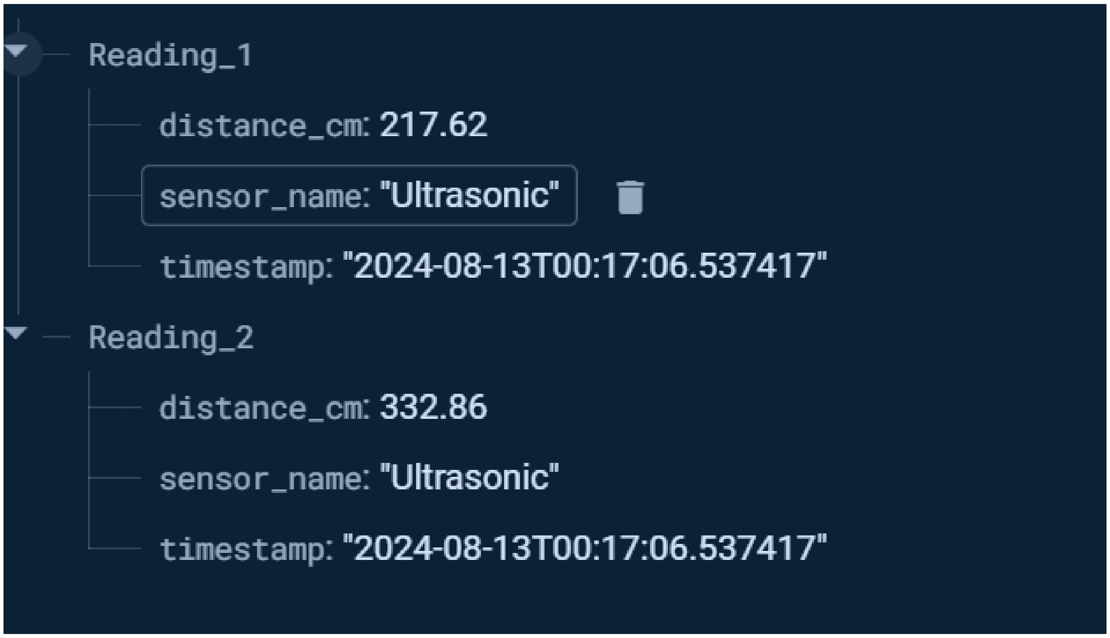


```
https://truong-khang-thinh-nguyen-default-rtdb.firebaseio.com/
▾ — Reading_1
      humidity: 58.07
      sensor_name: "DHT22"
      temperature: 27.27
      timestamp: "2024-08-13T00:02:47.217281"
▾ — Reading_2
      humidity: 45.37
      sensor_name: "DHT22"
      temperature: 20.16
      timestamp: "2024-08-13T00:02:47.217281"
```

	 Reading_3 humidity: 51.37 sensor_name: "DHT22" temperature: 26 timestamp: "2024-08-13T00:02:47.217281" Reading_4 humidity: 56.51 sensor_name: "DHT22" temperature: 23.66 timestamp: "2024-08-13T00:02:47.217281" Reading_5  Reading_5 humidity: 43.48 sensor_name: "DHT22" temperature: 24.29 timestamp: "2024-08-13T00:02:47.217281"
12	Question: Generate some random data for the SR04 Ultrasonic sensor, insert data to database, query all data and screenshot the output here. Answer: <Your answer>



```
ref = db.reference("/")

# Generate random data for the Ultrasonic sensor
data = {
    f"Reading_{i}": {
        "sensor_name": "Ultrasonic",
        "timestamp": datetime.now().isoformat(),
        "distance_cm": round(random.uniform(5.0, 400.0), 2)
    } for i in range(1, 3)  # Generate 5 random readings
}

# Insert the data into the database
ref.set(data)
```



```
Reading_1
    distance_cm: 217.62
    sensor_name: "Ultrasonic"
    timestamp: "2024-08-13T00:17:06.537417"
Reading_2
    distance_cm: 332.86
    sensor_name: "Ultrasonic"
    timestamp: "2024-08-13T00:17:06.537417"
```

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>

	Firebase Realtime Database events allow automatic triggering of actions based on data operations like creation, updates, or deletions. ➔ Database Events. The Python SDK primarily performs CRUD operations, and event handling is typically done using Firebase Cloud Functions. You can simulate real-time monitoring with Python by polling the database. ➔ Python SDK Handling. When sensors (e.g., Arduino) store data directly to Firebase, these events are useful for triggering actions, such as updating a monitoring dashboard. ➔ Use Case. ⇨ We can actually use Firebase Cloud Functions to handle real-time events and then use Firebase Admin SDK in Python to retrieve and process data as needed. This combination leverages the strengths of both platforms, enabling efficient real-time monitoring and data handling.

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.

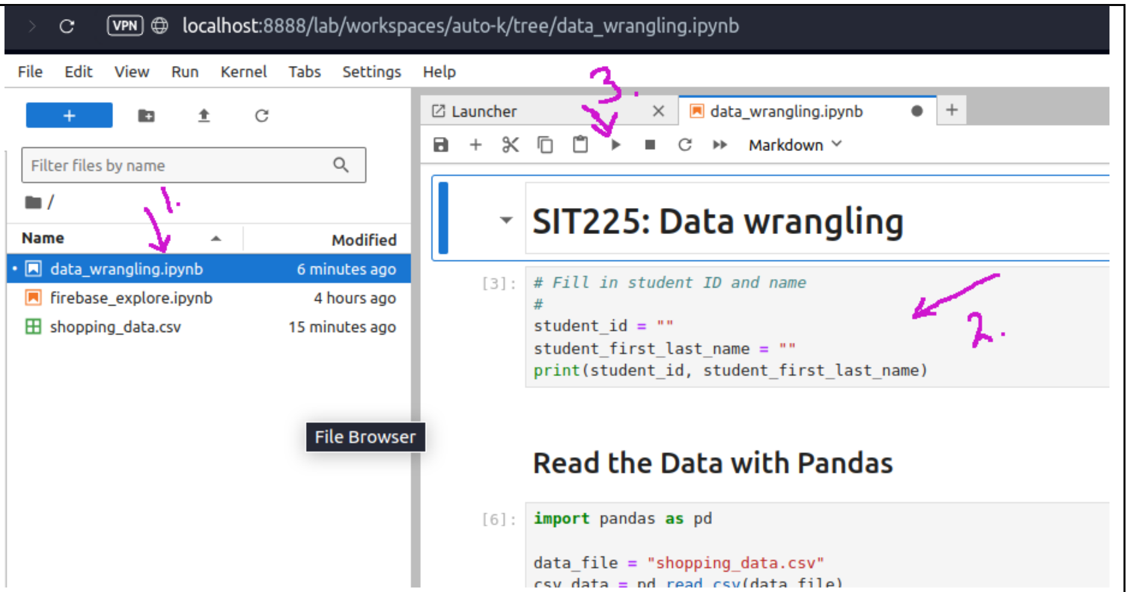
Software Required

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).



SIT225: Data wrangling

```
[3]: # Fill in student ID and name
     #
     student_id = ""
     student_first_last_name = ""
     print(student_id, student_first_last_name)
```

Read the Data with Pandas

```
[6]: import pandas as pd

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

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: <Your answer>

Of course that I can actually use Pandas in reading sensor CSV data that I have generated before. I think in terms of modification, first thing need considering is handling missing values, data type correction (ensure all the sensor readings are in the correct format), or data filtering ,etc.

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: <Your answer>

	- Removing missing data : applicable when the amount of missing data is minimal and when applying it does not affect significantly representativeness of the dataset. ➔ Loss of valuable information, especially in small datasets. - Mean / Median / Mode imputation: Mean and Median are appropriate when dealing with numerical data and Mode is for categorical data type ➔ Reduce variability and perhaps introduce bias if the missing data is not random. - Forward and Backward fill : commonly used in time series data at which the values are expected to be continuous or similar in the course of short periods ➔ it has to be in assumption that the missing data has a logical continuation from the adjacent values, and it might not be always the case. - Interpolation: Continuous data where a trend or pattern between surrounding data points has been observed and assumed ➔ Can be misleading if the data has sudden jumps or non-linear patterns.

firebase-explore

August 12, 2024

[ ]:

```
# Fill in student ID and name
#
student_id = "223446545"
student_first_last_name = "Truong Khang Thinh Nguyen"
print(student_id, student_first_last_name)
```

[ ]:

```
# Install libraris, if not yet.
! pip install firebase_admin pandas

import firebase_admin

databaseURL = 'https://truong-khang-thinh-nguyen-default-rtdb.firebaseio.com/'
cred_obj = firebase_admin.credentials.Certificate(
    'truong-khang-thinh-nguyen-firebase-adminsdk-mne9g-2ae8218f25.json'
)
default_app = firebase_admin.initialize_app(cred_obj, {
        'databaseURL':databaseURL
        })
```

[13]:

```
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)
```

[14]:

```
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'}
```

[15]:

```python
print(data)
```

```
{'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}}
```

[16]:

```python
# 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)
```

[17]:

```python
# 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})
```

```
{'-O45dqWXMwb_JudJ9X3c': {'Author': 'J.R.R. Tolkien', 'Genre': 'Epic fantasy',
'Price': 100, 'Title': 'The Fellowship of the Ring'}, '-O45dqcz5UwutpQYFqXS':
{'Author': 'J.R.R. Tolkien', 'Genre': 'Epic fantasy', 'Price': 100, 'Title':
'The Two Towers'}, '-O45dqkFGYxDTtamQkbm': {'Author': 'J.R.R. Tolkien', 'Genre':
'Epic fantasy', 'Price': 100, 'Title': 'The Return of the King'},
'-O45dqrnhcMlTw_yYwjL': {'Author': 'Paulo Coelho', 'Genre': 'Fiction', 'Price':
100, 'Title': 'Brida'}}
```

[18]:

```
# 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({})
```

[19]:

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

```
{'-O45dqrnhcMlTw_yYwjL': {'Author': 'Paulo Coelho', 'Genre': 'Fiction', 'Price':
100, 'Title': 'Brida'}}
<class 'dict'>
```

[20]:

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

data-wrangling

August 12, 2024

1 SIT225: Data wrangling

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

[1]:

```
# Fill in student ID and name
#
student_id = "223446545"
student_first_last_name = "Truong Khang Thinh Nguyen"
print(student_id, student_first_last_name)
```

```
223446545 Truong Khang Thinh Nguyen
```

1.1 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.

[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
```

1.2 Access the Column

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

[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
```

1.3 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.

[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 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
```

1.4 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

```
[9]: print("Shows data to 5th to less than 20th in a row:")
     print(csv_data.iloc[5:20])
```

```
Shows data to 5th to less than 20th 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
10          11    Male   67                  19                      14
11          12  Female   35                  19                      99
12          13  Female   58                  20                      15
13          14  Female   24                  20                      77
14          15    Male   37                  20                      13
15          16    Male   22                  20                      79
16          17  Female   35                  21                      35
17          18    Male   20                  21                      66
18          19    Male   52                  23                      29
19          20  Female   35                  23                      98
```

1.5 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.

[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  
```

1.6 Handling Missing Value

[7]:

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

```
False
```

[8]:

```
# We will use another data source with missing values to practice this part.
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
```

[ ]:

```

```

1.6.1 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)

1.6.2 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

[10]:

```
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: <your answer>
Because in the data_missing dataframe, we are currently having the non-numeric␣
 ↪column which is the Genre so it will terminate the program. Instead
make sure to exclude that column.
"""
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
File ~\Documents\ANACONDA\Lib\site-packages\pandas\core\nanops.py:1680, in␣
 ↪_ensure_numeric(x)
   1679 try:
-> 1680     x = x.astype(np.complex128)
   1681 except (TypeError, ValueError):

ValueError: complex() arg is a malformed string

During handling of the above exception, another exception occurred:
```

```
ValueError                                Traceback (most recent call last)
File ~\Documents\ANACONDA\Lib\site-packages\pandas\core\nanops.py:1683, in␣
 ↪_ensure_numeric(x)
   1682 try:
-> 1683     x = x.astype(np.float64)
   1684 except ValueError as err:
   1685     # GH#29941 we get here with object arrays containing strs

ValueError: could not convert string to float:␣
 ↪'MaleMaleFemaleFemaleFemaleFemaleFemaleFemaleMaleFemaleMaleFemaleFemaleFemaleMaleMaleFemale

The above exception was the direct cause of the following exception:

TypeError                                 Traceback (most recent call last)
Cell In[10], 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 ~\Documents\ANACONDA\Lib\site-packages\pandas\core\generic.py:11556, in␣
 ↪NDFrame._add_numeric_operations.<locals>.mean(self, axis, skipna,␣
 ↪numeric_only, **kwargs)
  11539 @doc(
  11540     _num_doc,
  11541     desc="Return the mean of the values over the requested axis.",
   (…)
  11554     **kwargs,
  11555 ):
> 11556     return NDFrame.mean(self, axis, skipna, numeric_only, **kwargs)

File ~\Documents\ANACONDA\Lib\site-packages\pandas\core\generic.py:11201, in␣
 ↪NDFrame.mean(self, axis, skipna, numeric_only, **kwargs)
  11194 def mean(
  11195     self,
  11196     axis: Axis | None = 0,
   (…)
  11199     **kwargs,
  11200 ) -> Series | float:
> 11201     return self._stat_function(
  11202         "mean", nanops.nanmean, axis, skipna, numeric_only, **kwargs
  11203     )
```

```
File ~\Documents\ANACONDA\Lib\site-packages\pandas\core\generic.py:11158, in␣
 ↪NDFrame._stat_function(self, name, func, axis, skipna, numeric_only, **kwargs
  11154     nv.validate_stat_func((), kwargs, fname=name)
  11156 validate_bool_kwarg(skipna, "skipna", none_allowed=False)
> 11158 return self._reduce(
  11159     func, name=name, axis=axis, skipna=skipna, numeric_only=numeric_onl
  11160 )

File ~\Documents\ANACONDA\Lib\site-packages\pandas\core\frame.py:10519, in␣
 ↪DataFrame._reduce(self, op, name, axis, skipna, numeric_only, filter_type,␣
 ↪**kwds)
  10515     df = df.T
  10517 # After possibly _get_data and transposing, we are now in the
  10518 #  simple case where we can use BlockManager.reduce
> 10519 res = df._mgr.reduce(blk_func)
  10520 out = df._constructor(res).iloc[0]
  10521 if out_dtype is not None:

File ~\Documents\ANACONDA\Lib\site-packages\pandas\core\internals\managers.py:
 ↪1534, in BlockManager.reduce(self, func)
   1532 res_blocks: list[Block] = []
   1533 for blk in self.blocks:
-> 1534     nbs = blk.reduce(func)
   1535     res_blocks.extend(nbs)
   1537 index = Index([None])  # placeholder

File ~\Documents\ANACONDA\Lib\site-packages\pandas\core\internals\blocks.py:339 ␣
 ↪in Block.reduce(self, func)
    333 @final
    334 def reduce(self, func) -> list[Block]:
    335     # We will apply the function and reshape the result into a single-r w
    336     #  Block with the same mgr_locs; squeezing will be done at a higher
 ↪level
    337     assert self.ndim == 2
--> 339     result = func(self.values)
    341     if self.values.ndim == 1:
    342         # TODO(EA2D): special case not needed with 2D EAs
    343         res_values = np.array([[result]])

File ~\Documents\ANACONDA\Lib\site-packages\pandas\core\frame.py:10482, in␣
 ↪DataFrame._reduce.<locals>.blk_func(values, axis)
  10480     return values._reduce(name, skipna=skipna, **kwds)
  10481 else:
> 10482     return op(values, axis=axis, skipna=skipna, **kwds)

File ~\Documents\ANACONDA\Lib\site-packages\pandas\core\nanops.py:96, in␣
 ↪disallow.__call__.<locals>._f(*args, **kwargs)
     94 try:
```

```
     95     with np.errstate(invalid="ignore"):
---> 96         return f(*args, **kwargs)
     97 except ValueError as e:
     98     # we want to transform an object array
     99     # ValueError message to the more typical TypeError
    100     # e.g. this is normally a disallowed function on
    101     # object arrays that contain strings
    102     if is_object_dtype(args[0]):

File ~\Documents\ANACONDA\Lib\site-packages\pandas\core\nanops.py:158, in␣
 ↪bottleneck_switch.__call__.<locals>.f(values, axis, skipna, **kwds)
    156         result = alt(values, axis=axis, skipna=skipna, **kwds)
    157 else:
--> 158     result = alt(values, axis=axis, skipna=skipna, **kwds)
    160 return result

File ~\Documents\ANACONDA\Lib\site-packages\pandas\core\nanops.py:421, in␣
 ↪_datetimelike_compat.<locals>.new_func(values, axis, skipna, mask, **kwargs)
    418 if datetimelike and mask is None:
    419     mask = isna(values)
--> 421 result = func(values, axis=axis, skipna=skipna, mask=mask, **kwargs)
    423 if datetimelike:
    424     result = _wrap_results(result, orig_values.dtype, fill_value=iNaT)

File ~\Documents\ANACONDA\Lib\site-packages\pandas\core\nanops.py:727, in␣
 ↪nanmean(values, axis, skipna, mask)
    724     dtype_count = dtype
    726 count = _get_counts(values.shape, mask, axis, dtype=dtype_count)
--> 727 the_sum = _ensure_numeric(values.sum(axis, dtype=dtype_sum))
    729 if axis is not None and getattr(the_sum, "ndim", False):
    730     count = cast(np.ndarray, count)

File ~\Documents\ANACONDA\Lib\site-packages\pandas\core\nanops.py:1686, in␣
 ↪_ensure_numeric(x)
   1683         x = x.astype(np.float64)
   1684     except ValueError as err:
   1685         # GH#29941 we get here with object arrays containing strs
-> 1686         raise TypeError(f"Could not convert {x} to numeric") from err
   1687 else:
   1688     if not np.any(np.imag(x)):

TypeError: Could not convert␣
 ↪['MaleMaleFemaleFemaleFemaleFemaleFemaleFemaleMaleFemaleMaleFemaleFemaleFemaleMaleMaleFemal
 ↪to numeric
```

[11]:

```
# 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
```

[12]:

```
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
```

[13]:

```
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
```

1.6.3 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.

[14]:
```
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 Median :")
print(data_filling2.head(10))

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

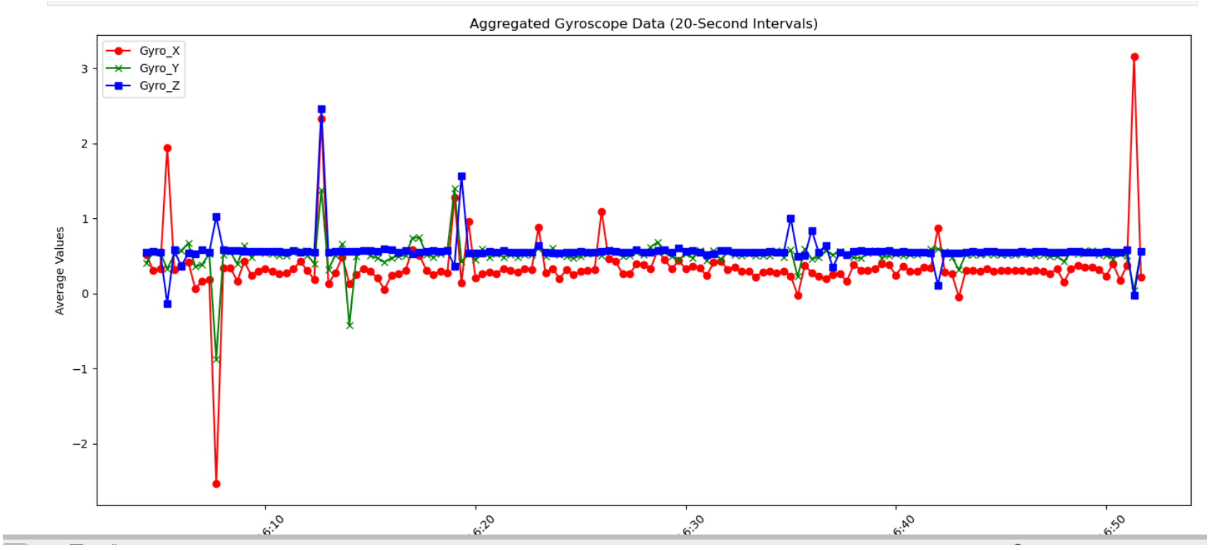
```
CustomerID                100.5
Age                        36.0
Annual Income (k$)         62.0
Spending Score (1-100)     50.0
dtype: float64
Dataset with empty values! :
   CustomerID   Age  Annual Income (k$)  Spending Score (1-100)
0           1  19.0                15.0                    39.0
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
```

TASK 5.1P – SIT225

TRUONG KHANG THINH NGUYEN

Hypothesis : If the Arduino Nano 33 IoT with the gyroscope sensor is placed on a stable table for one hour, the x, y, and z readings from the gyroscope will exhibit minor fluctuations primarily due to sensor noise and drift. Despite these fluctuations, the overall values for the x ,y and z axes will remain relatively stable, with any significant deviations likely attributed to sensor bias or external environmental factors.



As we can see from our generated graph from our recorded data, our assumption is true. We can see in some specific point of time, the x,y,z values did drop or increase significantly but then remained stable throughout the period of 1 hour.

Another point to know that , since during the time of implementing the work, I did put the Arduino align a little bit, which cause the x values remain smaller compared to other 2 y and z values. (Since y represents up and down, z represents forward and backward so they didn't change much is obvious since I kept the Arduino stayed the same on my table).

Python Program :

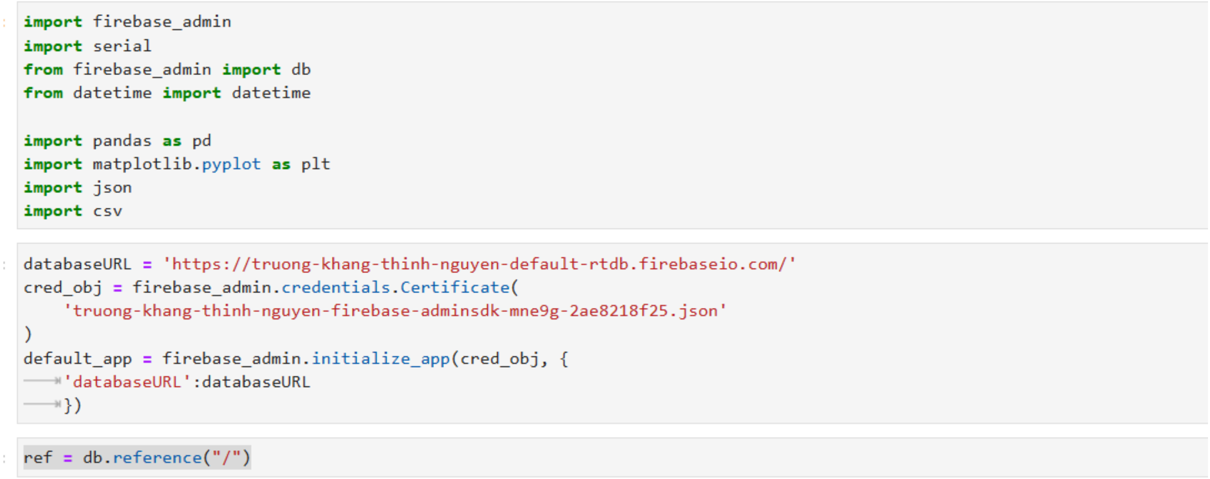
```
import firebase_admin

import serial

from firebase_admin import db

from datetime import datetime


import pandas as pd

import matplotlib.pyplot as plt

import json

import csv
```

```
databaseURL = 'https://truong-khang-thinh-nguyen-default-rtdb.firebaseio.com/'
cred_obj = firebase_admin.credentials.Certificate(
    'truong-khang-thinh-nguyen-firebase-adminsdk-mne9g-2ae8218f25.json'
)
default_app = firebase_admin.initialize_app(cred_obj, {
            'databaseURL':databaseURL
            })
ref = db.reference("/")
# Set the baud rate
baud_rate = 9600

# Create a serial port communication
s = serial.Serial(port = "COM7", baudrate = baud_rate, timeout=20)

while True:
    gyro_data = s.readline().decode("utf-8").strip()
    # Timestamp
    timestamp = datetime.now().strftime("%Y-%m-%d  %H:%M:%S")

    gyro_x , gyro_y , gyro_z = gyro_data.split(",")
    # Send data to Firebase using push
    ref.push({
    "Timestamp" : timestamp ,
    "Gyro_X": gyro_x ,
    "Gyro_Y" : gyro_y ,
    "Gyro_Z" : gyro_z
    })
# Load JSON data from a file
with open('Task 5.1P.json', 'r') as file:
    data = json.load(file)
```

```
# Convert the dictionary to a pandas DataFrame
df = pd.DataFrame.from_dict(data, orient='index')


# Save the DataFrame to a CSV file
df.to_csv('gyro_data.csv', index=False)
# Load converted CSV file from JSON file
df = pd.read_csv("gyro_data.csv")
df
# Check null valuesdf
df.info()
df['Timestamp'] = pd.to_datetime(df['Timestamp'])


# Set the Timestamp column as the index
df.set_index('Timestamp', inplace=True)


# Aggregate data to 20-second intervals, calculating the mean for each interval
df_aggregated = df.resample('20S').mean()


plt.figure(figsize=(14, 7))


plt.plot(df_aggregated.index, df_aggregated['Gyro_X'], label='Gyro_X', color='r', linestyle='-',
marker='o')
plt.plot(df_aggregated.index, df_aggregated['Gyro_Y'], label='Gyro_Y', color='g', linestyle='-',
marker='x')
plt.plot(df_aggregated.index, df_aggregated['Gyro_Z'], label='Gyro_Z', color='b', linestyle='-',
marker='s')


# Formatting the plot
plt.xlabel('Timestamp')
plt.ylabel('Average Values')
plt.title('Aggregated Gyroscope Data (20-Second Intervals)')
plt.legend()
```

plt.xticks(rotation=45)

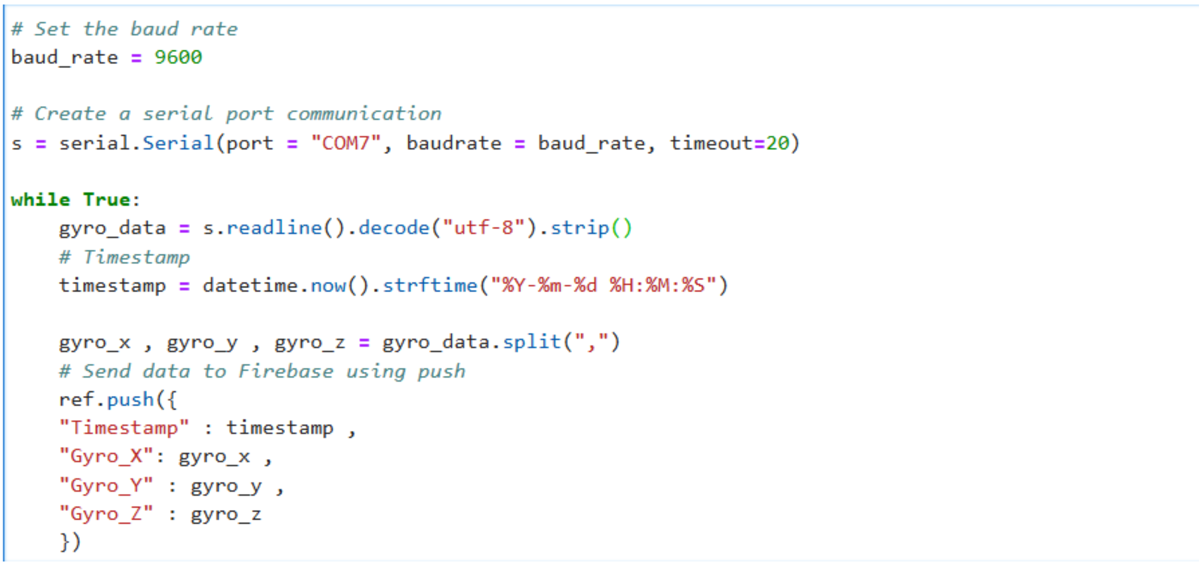
plt.tight_layout()



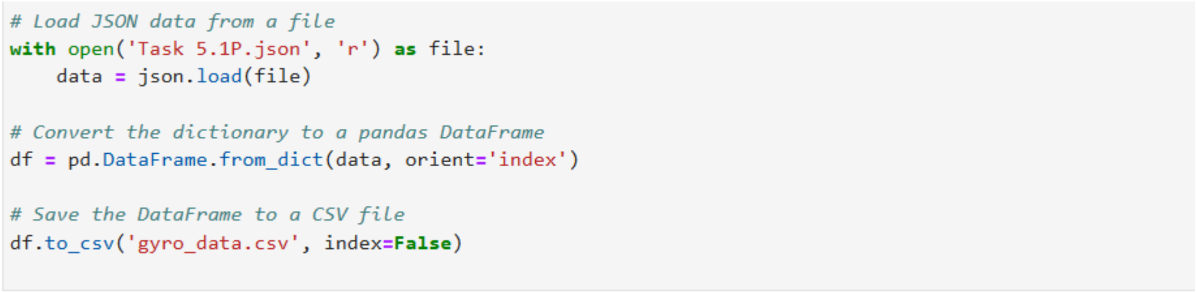
```python
import firebase_admin
import serial
from firebase_admin import db
from datetime import datetime

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

```python
databaseURL = 'https://truong-khang-thinh-nguyen-default-rtdb.firebaseio.com/'
cred_obj = firebase_admin.credentials.Certificate(
    'truong-khang-thinh-nguyen-firebase-adminsdk-mne9g-2ae8218f25.json'
)
default_app = firebase_admin.initialize_app(cred_obj, {
    'databaseURL':databaseURL
    })
```

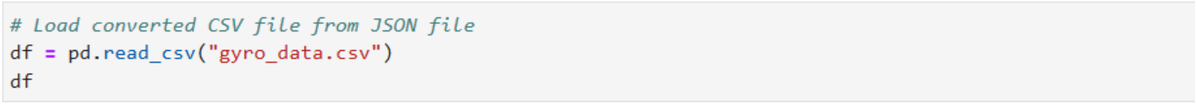
```python
ref = db.reference("/")
```



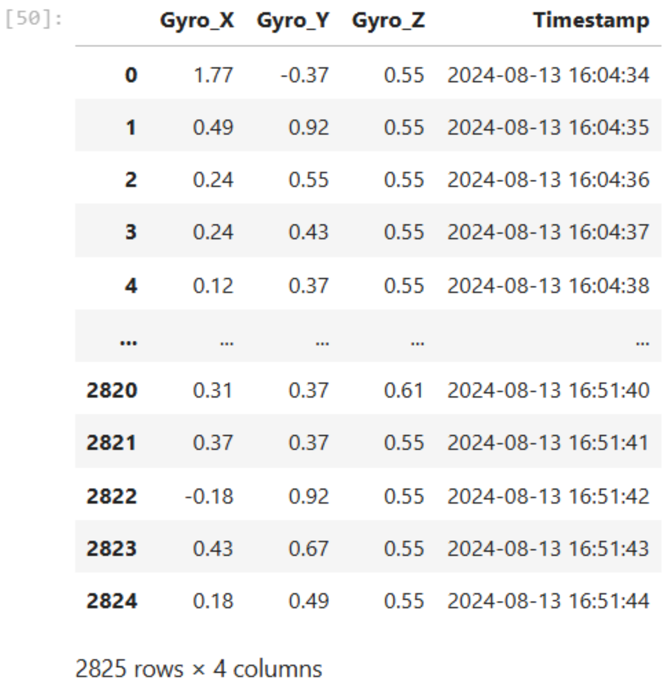
```python
# Set the baud rate
baud_rate = 9600

# Create a serial port communication
s = serial.Serial(port = "COM7", baudrate = baud_rate, timeout=20)

while True:
    gyro_data = s.readline().decode("utf-8").strip()
    # Timestamp
    timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")

    gyro_x , gyro_y , gyro_z = gyro_data.split(",")
    # Send data to Firebase using push
    ref.push({
    "Timestamp" : timestamp ,
    "Gyro_X": gyro_x ,
    "Gyro_Y" : gyro_y ,
    "Gyro_Z" : gyro_z
    })
```



```python
# Load JSON data from a file
with open('Task 5.1P.json', 'r') as file:
    data = json.load(file)

# Convert the dictionary to a pandas DataFrame
df = pd.DataFrame.from_dict(data, orient='index')

# Save the DataFrame to a CSV file
df.to_csv('gyro_data.csv', index=False)
```



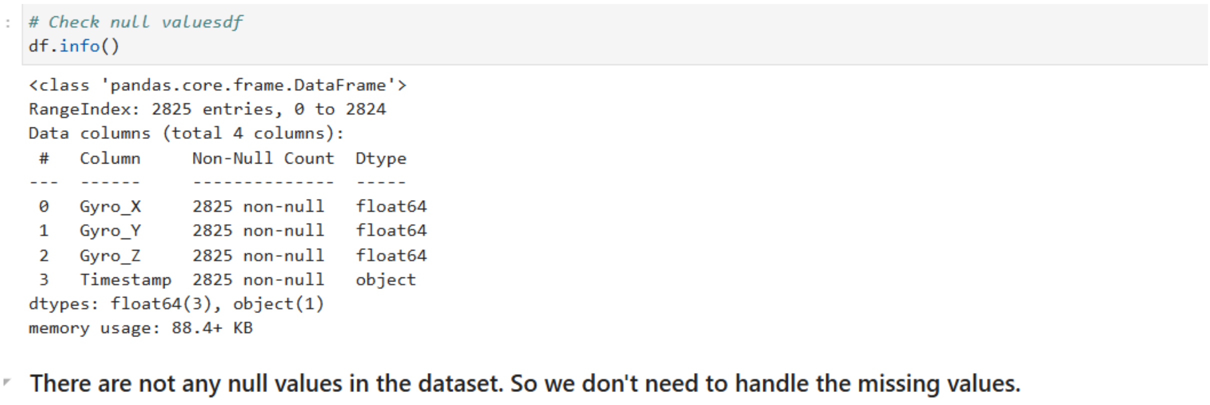
```python
# Load converted CSV file from JSON file
df = pd.read_csv("gyro_data.csv")
df
```



[50]:

	Gyro_X	Gyro_Y	Gyro_Z	Timestamp
0	1.77	-0.37	0.55	2024-08-13 16:04:34
1	0.49	0.92	0.55	2024-08-13 16:04:35
2	0.24	0.55	0.55	2024-08-13 16:04:36
3	0.24	0.43	0.55	2024-08-13 16:04:37
4	0.12	0.37	0.55	2024-08-13 16:04:38
...	...	...	...	...
2820	0.31	0.37	0.61	2024-08-13 16:51:40
2821	0.37	0.37	0.55	2024-08-13 16:51:41
2822	-0.18	0.92	0.55	2024-08-13 16:51:42
2823	0.43	0.67	0.55	2024-08-13 16:51:43
2824	0.18	0.49	0.55	2024-08-13 16:51:44

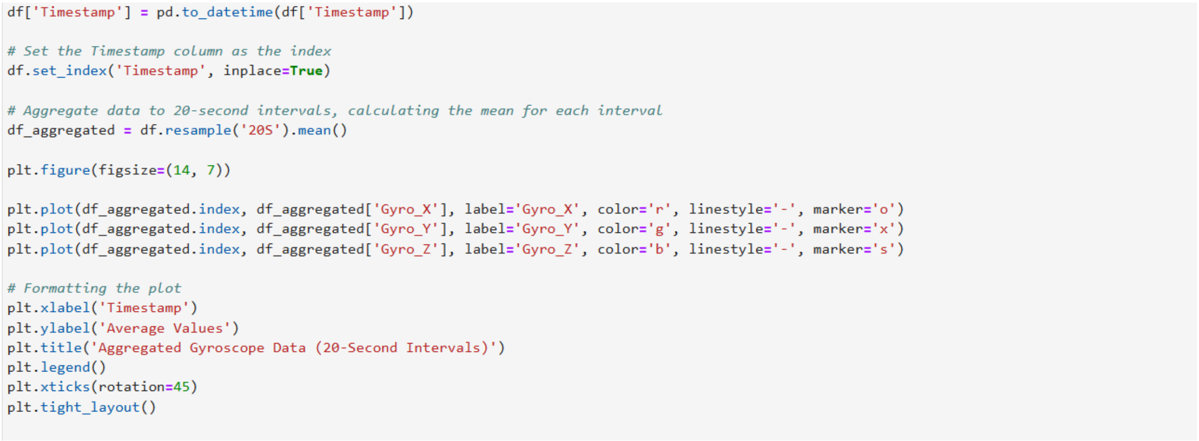
2825 rows × 4 columns



```
# Check null valuesdf
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2825 entries, 0 to 2824
Data columns (total 4 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   Gyro_X     2825 non-null   float64
 1   Gyro_Y     2825 non-null   float64
 2   Gyro_Z     2825 non-null   float64
 3   Timestamp  2825 non-null   object 
dtypes: float64(3), object(1)
memory usage: 88.4+ KB
```

There are not any null values in the dataset. So we don't need to handle the missing values.



```
df['Timestamp'] = pd.to_datetime(df['Timestamp'])

# Set the Timestamp column as the index
df.set_index('Timestamp', inplace=True)

# Aggregate data to 20-second intervals, calculating the mean for each interval
df_aggregated = df.resample('20S').mean()

plt.figure(figsize=(14, 7))

plt.plot(df_aggregated.index, df_aggregated['Gyro_X'], label='Gyro_X', color='r', linestyle='-', marker='o')
plt.plot(df_aggregated.index, df_aggregated['Gyro_Y'], label='Gyro_Y', color='g', linestyle='-', marker='x')
plt.plot(df_aggregated.index, df_aggregated['Gyro_Z'], label='Gyro_Z', color='b', linestyle='-', marker='s')

# Formatting the plot
plt.xlabel('Timestamp')
plt.ylabel('Average Values')
plt.title('Aggregated Gyroscope Data (20-Second Intervals)')
plt.legend()
plt.xticks(rotation=45)
plt.tight_layout()
```

Basically the Python script tried to open the connection between the Python script and the Firebase database. Then it connected with the Arduino IDE via the Serial Communication and then used the data recorded from the Gyroscope sensor from the Arduino Iot33

The next stage is that after it received the data from the Arduino it appended the row data to the firebase and just from that after 1 hour of executing. I stopped the program to receive any further data.

In terms of converting the CSV file, firstly I downloaded the JSON file from the Firebase and then wrote a program to convert it into CSV file. Then I plotted the x,y,z variables together with checking null values and interpreting the generated graph.

Arduino Code:

```
#include <Arduino_LSM6DS3.h>


float gyro_x, gyro_y, gyro_z;


void setup() {
  Serial.begin(9600); // set baud rate
  while (!Serial);  // wait for port to init


  if (!IMU.begin()) {
    Serial.println("Failed to initialize IMU!");
    while (1);
  }
}


void loop() {
  // read accelero data
  if (IMU.gyroscopeAvailable()) {
    IMU.readGyroscope(gyro_x, gyro_y, gyro_z);
  }


  Serial.println(
    String(gyro_x) + ", " + String(gyro_y) + ", " + String(gyro_z));


  delay(1000);
}
```

For the Arduino IDE, after including the necessary packages for measuring the Gryroscope data and then it sent the recorded data to the Serial communication. And its sampling rate is 1, I think this one is the appropriate sampling rate since it provides a good balance between capturing detailed rotational data and maintaining manageable data volumes. This rate is suitable to detect meaningful changes in movement and avoids any redundant data that has already been recorded.

Link Video Demonstrating

https://deakin.au.panopto.com/Panopto/Pages/Viewer.aspx?id=5435cc4c-66ce-4202-ba8c-b1cb00992fa7