Hypothesis

The gyroscope readings (X, Y, Z axes) should display consistent patterns, particularly when the device undergoes smooth, repetitive motions. If there are rapid spikes or dips, they might indicate abrupt movements or me getting tired moving the chair.

Data Collection

I collected gyroscope data (X, Y, Z axes) to analyze the rotational movement of the device while spinning myself and moving sitting in a desk chair. This can be used to detect orientation changes, motion patterns, or even sudden movements like shakes or twists.

I Collected the data continuously over a period of around 10 minutes. Sampling at 1-second intervals should provide enough granularity to identify meaningful changes in motion.

Observation and Analysis

Pattern Observation:

Repeating Patterns: I moved the device in a cyclic motion so you'll see repeating patterns but i made sure to change directing and move around to get various data. You may observe downward trends in the axes if the device is slowly tilted in a single direction. Relative Changes: By plotting all three axes together, you can see how the X, Y, and Z gyroscope readings change relative to one another.

If the hypothesis is that there will be a repeating pattern during a circular or repetitive motion, and the data supports this, the hypothesis holds. If the hypothesis doesn’t hold, potential reasons could be inconsistencies in the motion, errors in the sensor readings, or environmental factors.

In [5]:
```
import firebase_admin
from firebase_admin import credentials, db
import pandas as pd
import csv
import time

# Initialize Firebase
cred_obj = firebase_admin.credentials.Certificate(
    r'C:\Users\ramma\Downloads\sit225-5-firebase-adminsdk-o982n-f4f8bea959.j
)
firebase_admin.initialize_app(cred_obj, {
```

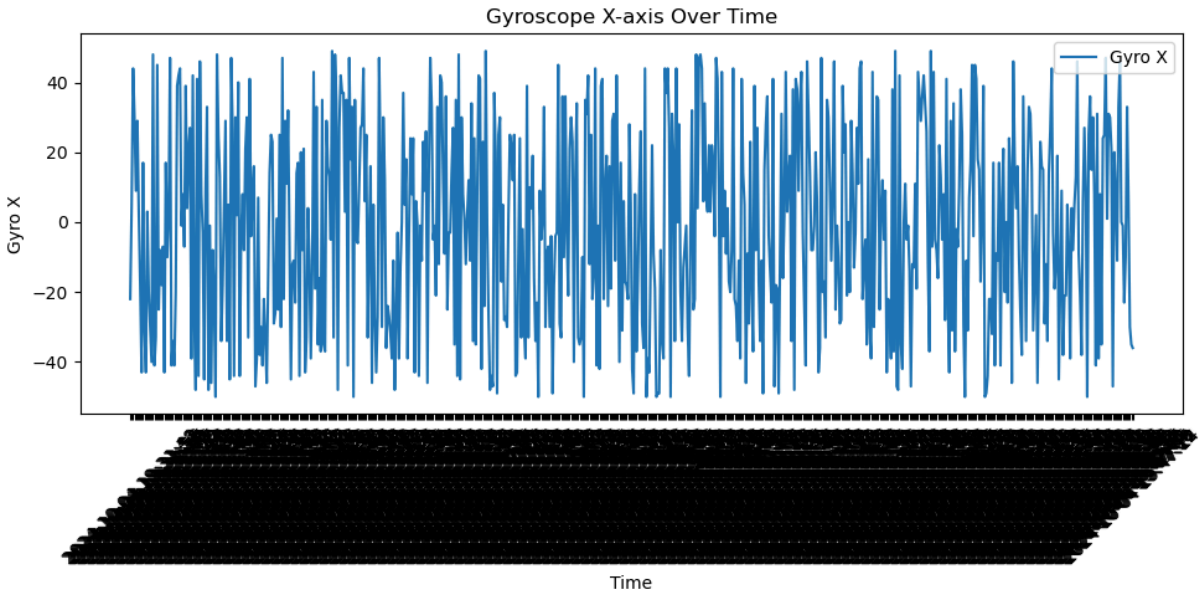
```
    'databaseURL': 'https://sit225-5-default-rtdb.firebaseio.com/'
})

# Reference to the Firebase data
ref = db.reference('/mpu6050_data')

# Get all data
all_data = ref.get()

# Convert data to CSV format and save as a CSV file
with open('gyro_data.csv', mode='w', newline='') as file:
    writer = csv.writer(file)
    # Write the header
    writer.writerow(["timestamp", "gyro_x", "gyro_y", "gyro_z"])

    # Write the data
    for key, value in all_data.items():
        writer.writerow([value["timestamp"], value["gyro_x"], value["gyro_y"

# Load the CSV into a Pandas DataFrame for cleaning and analysis
df = pd.read_csv('gyro_data.csv')

# Cleaning the data (removing non-number and empty fields)
df.dropna(inplace=True)  # Removes rows with empty fields

# Save the cleaned data back to CSV
df.to_csv('cleaned_gyro_data.csv', index=False)

print("Cleaned data saved to 'cleaned_gyro_data.csv'")
```

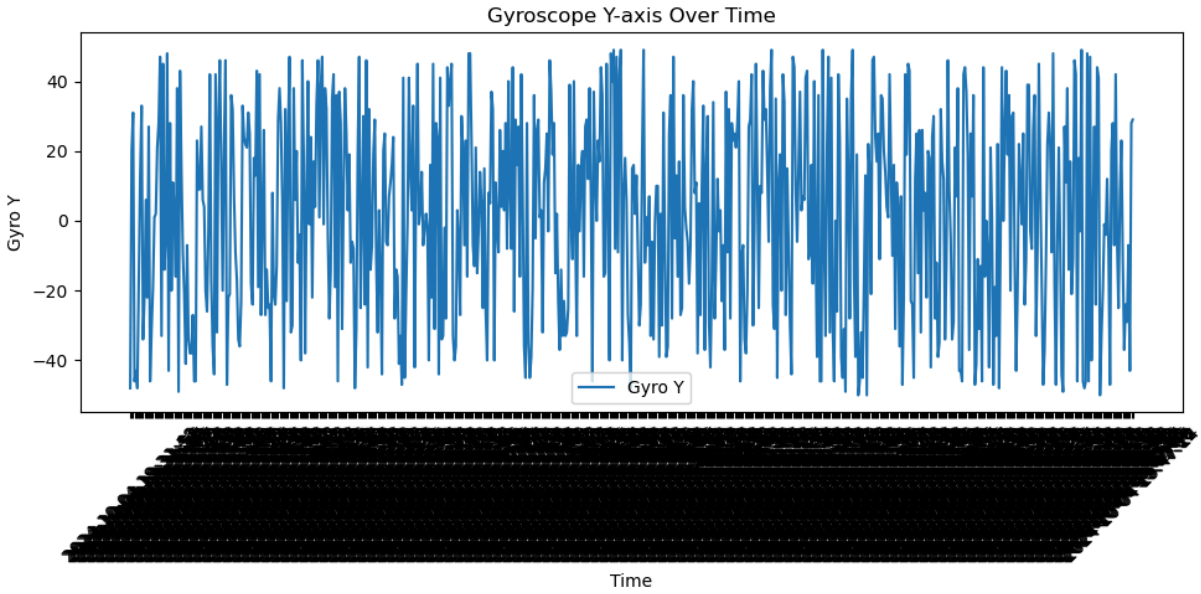
```
Cleaned data saved to 'cleaned_gyro_data.csv'
```

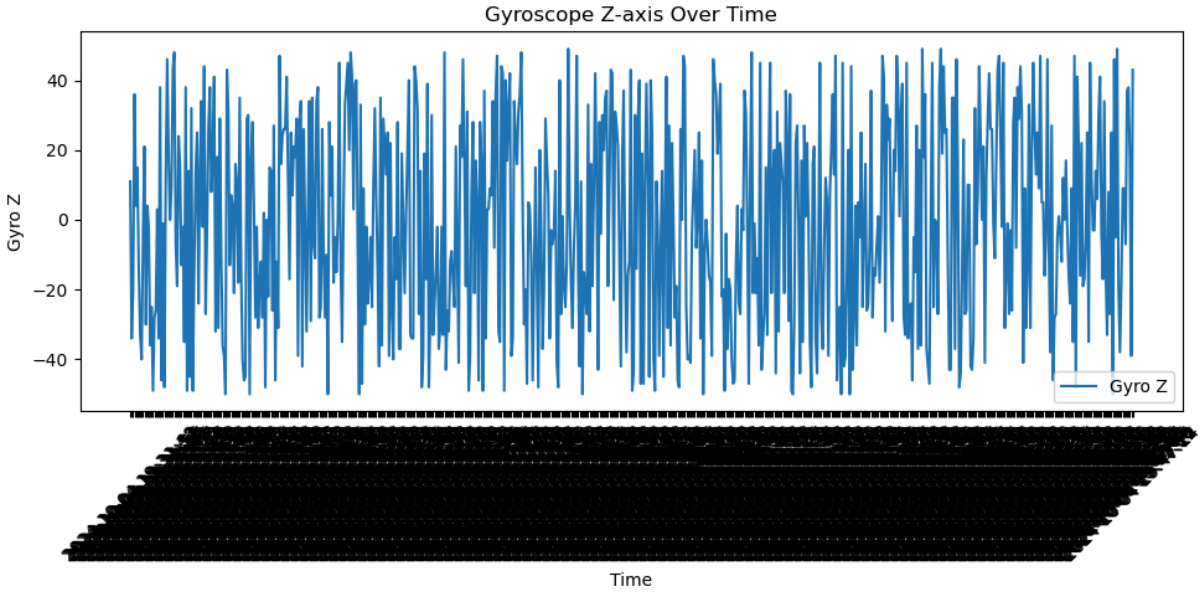
In [8]:

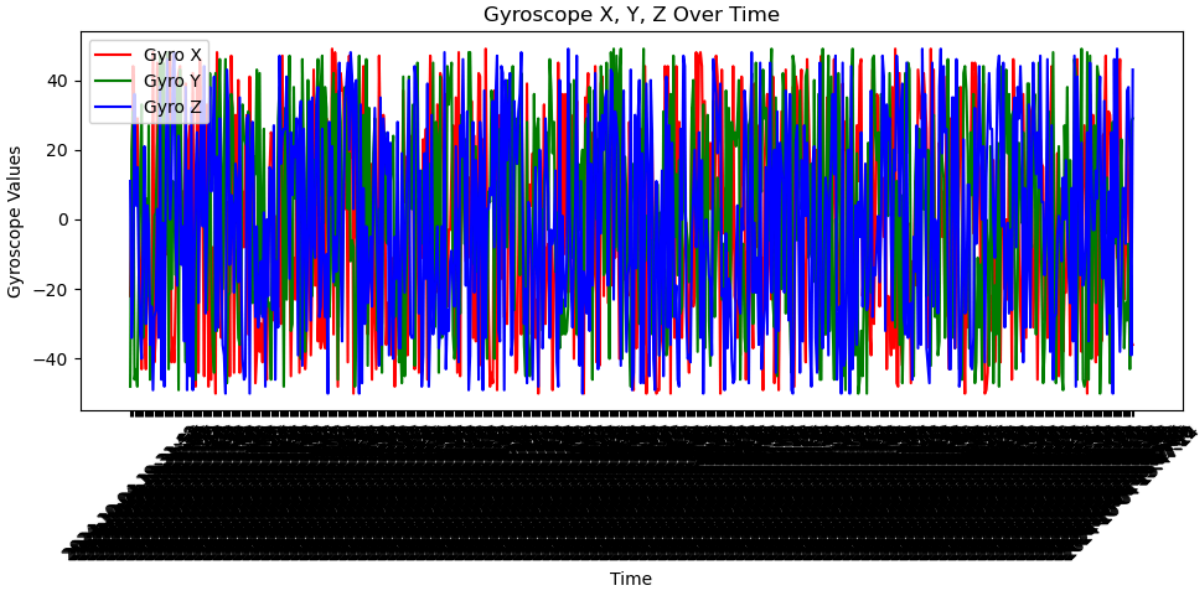
```
import matplotlib.pyplot as plt

# Load the cleaned data
df = pd.read_csv('cleaned_gyro_data.csv')

# Plot X-axis Gyroscope data
plt.figure(figsize=(10,5))
plt.plot(df['timestamp'], df['gyro_x'], label='Gyro X')
plt.xlabel('Time')
plt.ylabel('Gyro X')
plt.title('Gyroscope X-axis Over Time')
plt.xticks(rotation=45)
plt.tight_layout()
plt.legend()
plt.show()

# Plot Y-axis Gyroscope data
plt.figure(figsize=(10,5))
plt.plot(df['timestamp'], df['gyro_y'], label='Gyro Y')
plt.xlabel('Time')
plt.ylabel('Gyro Y')
plt.title('Gyroscope Y-axis Over Time')
plt.xticks(rotation=45)
plt.tight_layout()
```

```
plt.legend()
plt.show()

# Plot Z-axis Gyroscope data
plt.figure(figsize=(10,5))
plt.plot(df['timestamp'], df['gyro_z'], label='Gyro Z')
plt.xlabel('Time')
plt.ylabel('Gyro Z')
plt.title('Gyroscope Z-axis Over Time')
plt.xticks(rotation=45)
plt.tight_layout()
plt.legend()
plt.show()

# Plot all three axes in one graph for comparison
plt.figure(figsize=(10,5))
plt.plot(df['timestamp'], df['gyro_x'], label='Gyro X', color='red')
plt.plot(df['timestamp'], df['gyro_y'], label='Gyro Y', color='green')
plt.plot(df['timestamp'], df['gyro_z'], label='Gyro Z', color='blue')
plt.xlabel('Time')
plt.ylabel('Gyroscope Values')
plt.title('Gyroscope X, Y, Z Over Time')
plt.xticks(rotation=45)
plt.tight_layout()
plt.legend()
plt.show()
```




Gyroscope Y-axis Over Time
Gyro Y
40
20
0
−20
−40
Gyro Y
Time


Gyroscope Z-axis Over Time
Gyro Z
40
20
0
−20
−40
Gyro Z
Time


Gyroscope X, Y, Z Over Time
Gyro X
Gyro Y
Gyro Z
Gyroscope Values
40
20
0
−20
−40
Time

Student name: Rammaka Iddamalgoda

Student ID: s223496576

SIT225: Data Capture Technologies

Video link: https://deakin.au.panopto.com/Panopto/Pages/Viewer.aspx?id=d64650d9-319a-4ffa-96b4-b1ff009e7626

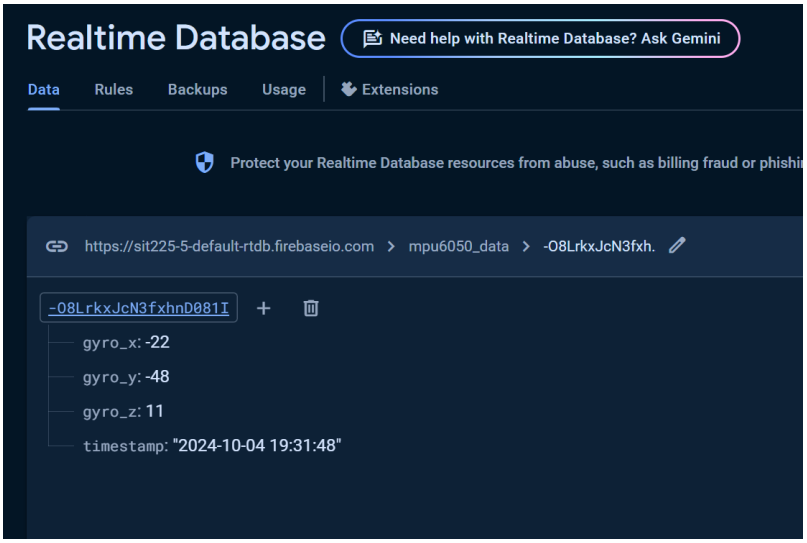
Py script

```
import serial
import time
import json
import firebase_admin
from firebase_admin import credentials, db

# Initialize Firebase with the correct credentials
cred_obj = firebase_admin.credentials.Certificate(
  r'C:\Users\ramma\Downloads\sit225-5-firebase-adminsdk-o982n-f4f8bea959.json'
)
firebase_admin.initialize_app(cred_obj, {
  'databaseURL': 'https://sit225-5-default-rtdb.firebaseio.com//'
})

# Set up Serial communication
ser = serial.Serial('COM3', 9600, timeout=1)
time.sleep(2)  # Allow some time for Arduino to initialize

# Firebase database reference
ref = db.reference('mpu6050_data')

while True:
  data = ser.readline().decode('utf-8').strip()
  if data:
    try:
      # Parse the labeled data
      if "Gyro X:" in data:
        parts = data.replace("Gyro X: ", "").replace("Y: ", "").replace("Z: ", "").split(",")
        gyro_x, gyro_y, gyro_z = map(float, parts)
        timestamp = time.strftime('%Y-%m-%d %H:%M:%S')

        # Prepare data in JSON format
        gyro_data = {
          "timestamp": timestamp,
          "gyro_x": gyro_x,
          "gyro_y": gyro_y,
          "gyro_z": gyro_z
        }

        # Push data to Firebase
        ref.push(gyro_data)
        print(f"Data sent: {gyro_data}")
    except ValueError:
      print(f"Error parsing data: {data}")

  time.sleep(1)  # Adjust the data rate
```

Arduino Sketch

```
#include <Wire.h>
#include <Adafruit_MPU6050.h>
#include <Adafruit_Sensor.h>
Adafruit_MPU6050 mpu;

void setup() {
 Serial.begin(9600);
 Wire.begin();

 // Initialize the MPU6050 sensor
 if (!mpu.begin()) {
  Serial.println("Failed to find MPU6050 chip");
  while (1) {
   delay(10);
  }
 }

 // Setup the gyro and accelerometer ranges
 mpu.setGyroRange(MPU6050_RANGE_250_DEG);
 mpu.setAccelerometerRange(MPU6050_RANGE_2_G);
 mpu.setFilterBandwidth(MPU6050_BAND_21_HZ);

 Serial.println("MPU6050 Initialized");
}

void loop() {
 sensors_event_t a, g, temp;
 mpu.getEvent(&a, &g, &temp);

 // Print gyroscope data to the Serial Monitor
 Serial.print("Gyro X: "); Serial.print(g.gyro.x);
 Serial.print(", Y: "); Serial.print(g.gyro.y);
 Serial.print(", Z: "); Serial.println(g.gyro.z);

 delay(500); // Adjust the data rate
}
```

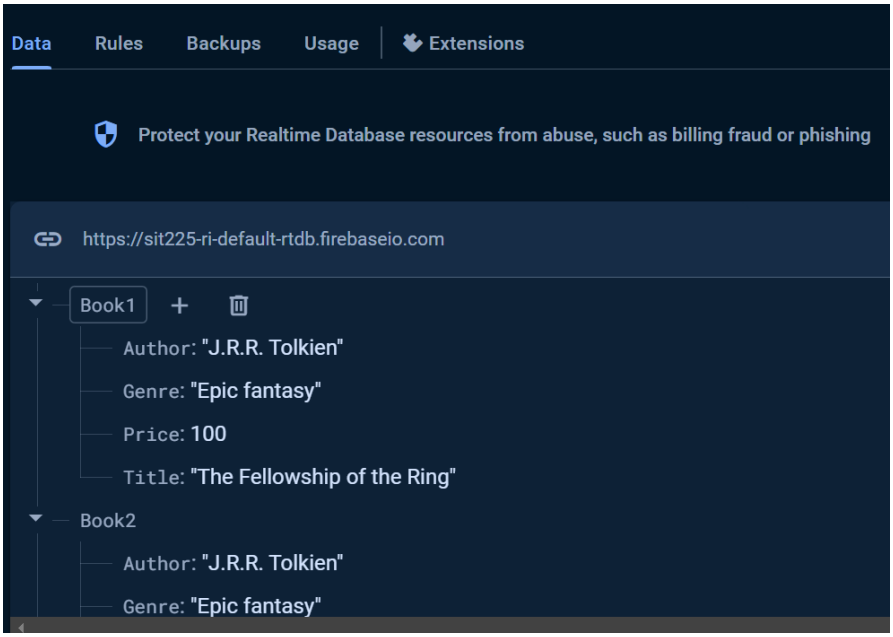
Realtime database: Gyro data

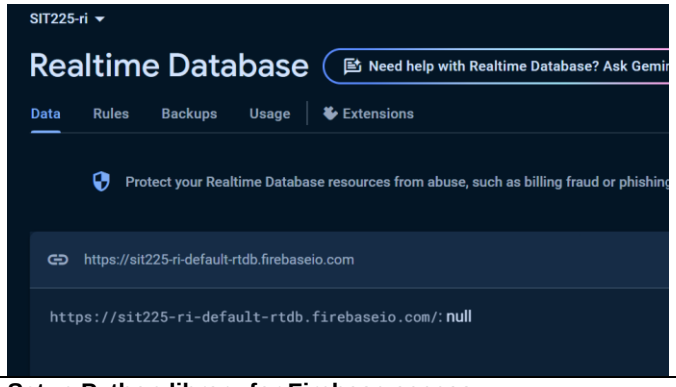


Activity 5.1: Firebase Realtime database

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.

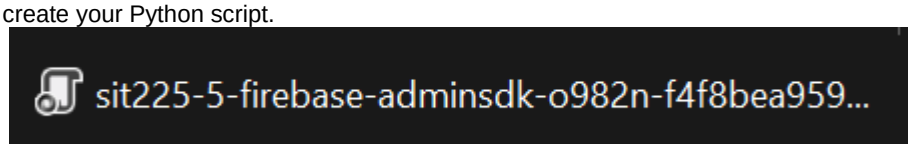
My realtime database



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**: 
3	**Setup Python library for Firebase access**: We will be using Admin Database API, which is available in *firebase_admin* library. pip install firebase_admin\

```
Downloading cachecontrol-0.14.0-py3-none-any.whl (22 kB)
Downloading google_cloud_firestore-2.19.0-py2.py3-none-any.whl (336 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 336.8/336.8 kB 10.5 MB/s eta 0:00:00
Downloading google_cloud_storage-2.18.2-py2.py3-none-any.whl (130 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 130.5/130.5 kB 3.8 MB/s eta 0:00:00
Downloading PyJWT-2.9.0-py3-none-any.whl (22 kB)
Downloading google_cloud_core-2.4.1-py2.py3-none-any.whl (29 kB)
Downloading google_crc32c-1.6.0-cp311-cp311-win_amd64.whl (33 kB)
Downloading google_resumable_media-2.7.2-py2.py3-none-any.whl (81 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 81.3/81.3 kB 503.6 kB/s eta 0:00:00
Downloading grpcio-1.66.2-cp311-cp311-win_amd64.whl (4.3 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 4.3/4.3 MB 14.4 MB/s eta 0:00:00
Downloading grpcio_status-1.62.3-py3-none-any.whl (14 kB)
Downloading protobuf-4.25.5-cp310-abi3-win_amd64.whl (413 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 413.4/413.4 kB 8.6 MB/s eta 0:00:00
Installing collected packages: pyjwt, protobuf, grpcio, google-crc32c, google-resumable-media, c
us, google-cloud-core, google-cloud-storage, google-cloud-firestore, firebase_admin
  Attempting uninstall: pyjwt
    Found existing installation: PyJWT 2.4.0
    Uninstalling PyJWT-2.4.0:
      Successfully uninstalled PyJWT-2.4.0
  Attempting uninstall: protobuf
    Found existing installation: protobuf 3.20.3
    Uninstalling protobuf-3.20.3:
      Successfully uninstalled protobuf-3.20.3
Successfully installed cachecontrol-0.14.0 firebase_admin-6.5.0 google-cloud-core-2.4.1 google-c
ogle-cloud-storage-2.18.2 google-crc32c-1.6.0 google-resumable-media-2.7.2 grpcio-1.66.2 grpcio-
.25.5 pyjwt-2.9.0
```

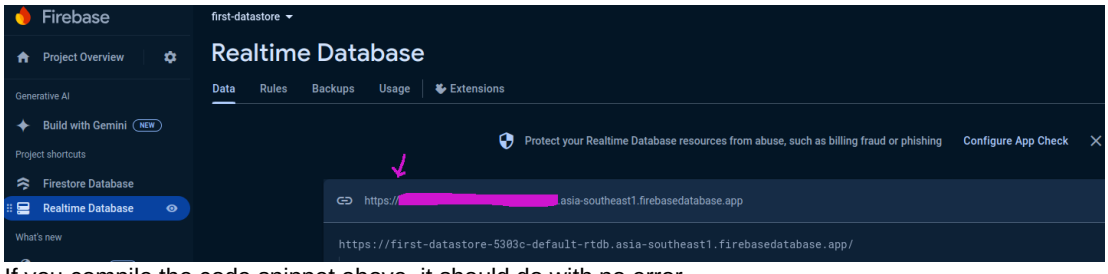
Firebase will allow access to Firebase server APIs from Google Service Accounts. To authenticate the Service Account, we require a private key in JSON format. To generate the key, go to project settings, click Generate new private key, download the file, and place it in your current folder where you 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).

```python
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.

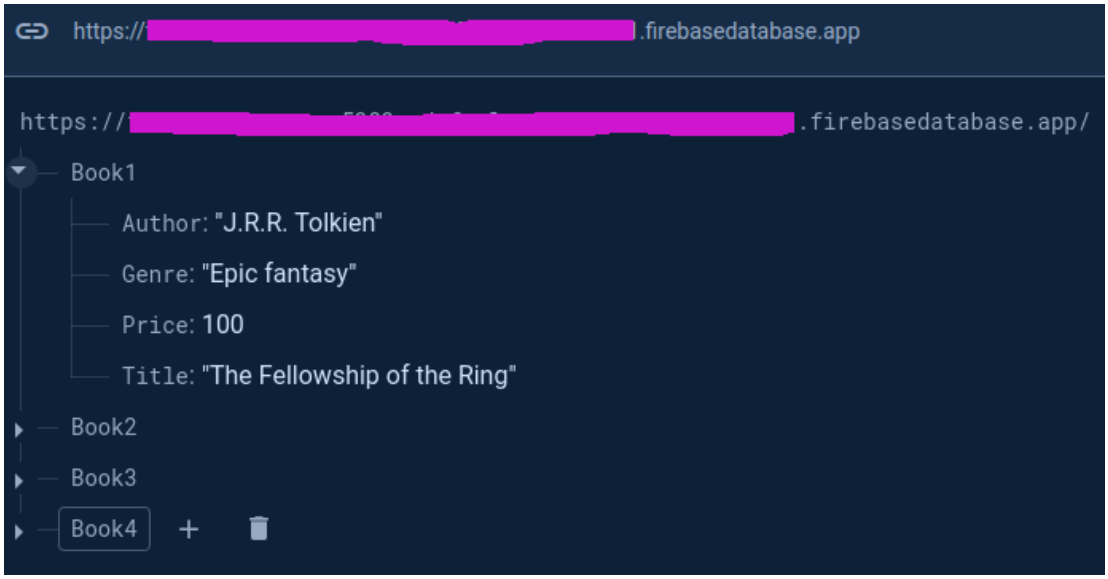


If you compile the code snippet above, it should do with no error.

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.

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

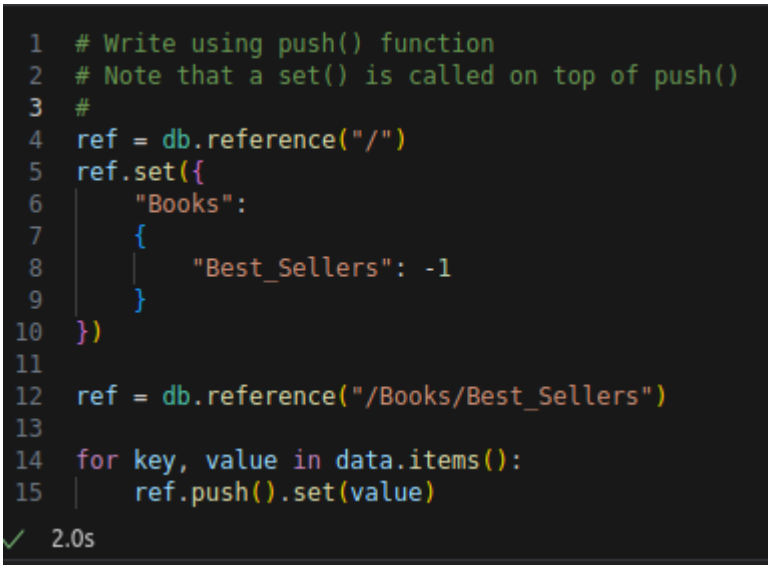
Read data using get() function:
Data can be read using get() function on the reference set beforehand, as shown below.

```
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.  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.

```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

```
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.

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

Now in Firebase console you will see no data exists.

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:

```
{
  "sensor_name": "DHT22",
  "timestamp": "2024-10-03 15:01:23",
  "temperature": 24.5,
  "humidity": 60
}
```

to handle multiple data values (like temperature and humidity) in the same JSON object, you can store data for multiple sensors in the database under different nodes. For example:

json
Copy code

```
{
  "DHT22": {
    "sensor_name": "DHT22",
    "timestamp": "2024-10-03 15:01:23",
    "temperature": 24.5,
    "humidity": 60
  },
  "SR04": {
    "sensor_name": "SR04",
```

<table>
<tr><td></td><td>"timestamp": "2024-10-03 15:01:23",
"distance": 150
}
}</td></tr>
<tr><td>11</td><td></td></tr>
<tr><td>12</td><td>Question: Generate some random data for DHT22 sensor, insert data to database, query all data and screenshot the output here.

Question: Generate some random data for the SR04 Ultrasonic sensor, insert data to database, query all data and screenshot the output here.

Answer:

For both 11 and 12

<pre>
default_app = firebase_admin.initialize_app(cred_obj, {
    'databaseURL': 'https://sit225-ri-default-rtdb.firebaseio.com/'
})

# Generate random data for DHT22 sensor
dht_data = {
    "sensor_name": "DHT22",
    "timestamp": time.strftime('%Y-%m-%d %H:%M:%S'),
    "temperature": round(random.uniform(20, 30), 2),
    "humidity": random.randint(40, 70)
}

# Push DHT22 data to Firebase
ref = db.reference("/DHT22_data")
ref.push(dht_data)

# Generate random data for SR04 sensor (Ultrasonic sensor)
sr04_data = {
    "sensor_name": "SR04",
    "timestamp": time.strftime('%Y-%m-%d %H:%M:%S'),
    "distance": random.randint(100, 200)
}

# Push SR04 data to Firebase
ref = db.reference("/SR04_data")
ref.push(sr04_data)
</pre>
<pre>
Data successfully pushed to Firebase
</pre></td></tr>
<tr><td>13</td><td>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: Firebase Realtime Database generates events on data operations (insert, update, delete). These events can be handled using listeners in Python to trigger actions when data changes.</td></tr>
</table>

	In Python, you can use the listen() function to monitor changes in real-time, but Firebase Functions (in JavaScript) are more commonly used for handling database events directly on the server side.

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.

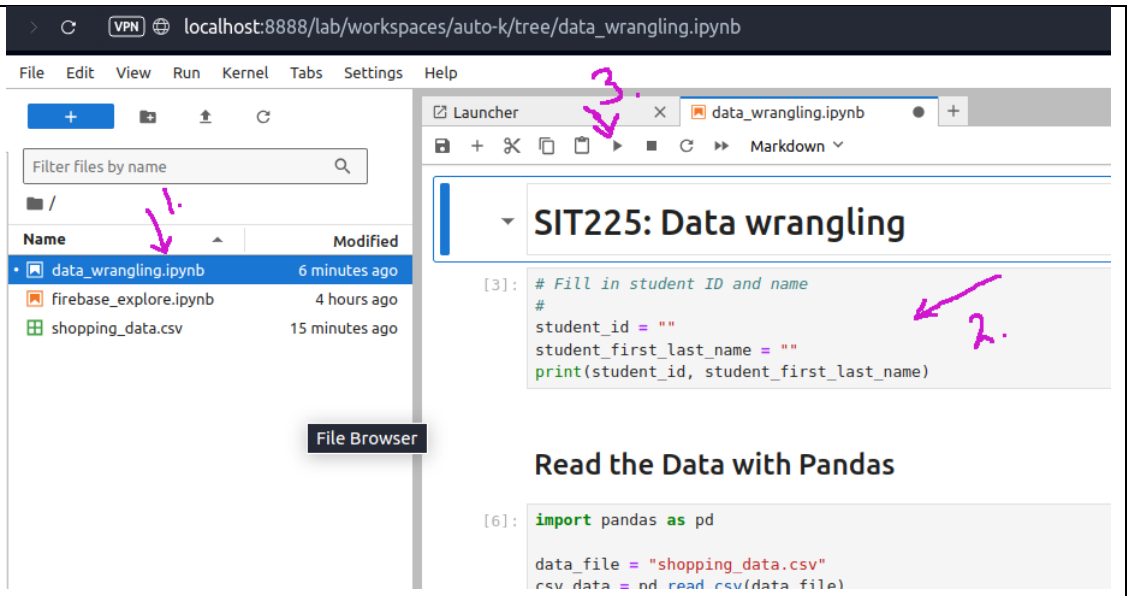
Software Required

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, I can use Pandas to read the sensor data that I generated earlier. The pd.read_csv() function is useful for loading the CSV file. If there are missing or noisy values, I can handle those by dropping them. Or, I could use methods like fillna() to fill missing values with the mean or median of the data to ensure consistency. If the CSV file has inconsistent formatting or extra columns, I might need to specify the correct delimiter or column names while loading the file with pd.read_csv(). Additionally, it may be necessary to check the data types of each column and convert them appropriately, especially for the timestamp and numeric data.
4	Question: What do you understand of the Notebook section called Handling Missing Value? Discuss in group and briefly summarise different missing value imputation methods and their applicability on different data conditions. Answer: The Notebook section on Handling Missing Values discusses various imputation techniques that help fill in missing data points in a dataset. The common methods include: Mean Imputation: This method fills missing values with the mean of the available data. It is suitable for data with no major outliers or when data points are evenly distributed around the mean.

	Median Imputation: This method fills missing values with the median of the dataset. It works well for skewed data or when the data has significant outliers, as the median is less affected by extreme values. Mode Imputation: This is used for categorical data where the most frequent value (mode) is used to replace missing values. Linear Interpolation: This is applied for time-series data where the trend and continuity are important. It estimates missing values by interpolating between known data points. Dropping Missing Values: Sometimes, if there are only a few missing data points, it is better to drop the rows or columns with missing values, especially if their absence does not significantly affect the analysis. These methods are applied depending on the type of data and how much missing data exists. For example, mean or median imputation is suitable for numerical data, while linear interpolation is preferred in time-series data.

In [11]:
```
# Fill in student ID and name
#
student_id = "s223496576"
student_first_last_name = "Rammaka Iddamalgoda"
print(student_id, student_first_last_name)
```

```
s223496576 Rammaka Iddamalgoda
```

In [13]:
```
import firebase_admin
from firebase_admin import credentials, db

# Initialize Firebase Admin SDK with the correct certificate path
cred_obj = firebase_admin.credentials.Certificate(
    r'C:\Users\ramma\Downloads\sit225-ri-firebase-adminsdk-octoc-af69f7a7ab.
)

# Initialize Firebase app (ensure only one initialization)
if not firebase_admin._apps:
    default_app = firebase_admin.initialize_app(cred_obj, {
        'databaseURL': 'https://sit225-ri-default-rtdb.firebaseio.com/'
    })

# A reference point is always needed to be set before any operation is carri
ref = db.reference("/")

# JSON format data (key/value pair)
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
    }
}

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

```
# 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 a specific book (/Book1)
ref = db.reference("/Book1")
book1 = ref.get()
print(book1)

# Write data using push()
ref = db.reference("/Books/Best_Sellers")
for key, value in data.items():
    ref.push().set(value)

# Update data (reducing price for Tolkien books)
ref = db.reference("/Books/Best_Sellers/")
best_sellers = ref.get()
for key, value in best_sellers.items():
    if value["Author"] == "J.R.R. Tolkien":
        ref.child(key).update({"Price": 80})

# Delete all Tolkien's books from the Best Sellers list
ref = db.reference("/Books/Best_Sellers/")
for key, value in best_sellers.items():
    if value["Author"] == "J.R.R. Tolkien":
        ref.child(key).set({})
```

```
{'Book1': {'Author': 'J.R.R. Tolkien', 'Genre': 'Epic fantasy', 'Price': 10
0, 'Title': 'The Fellowship of the Ring'}, 'Book2': {'Author': 'J.R.R. Tolki
en', 'Genre': 'Epic fantasy', 'Price': 100, 'Title': 'The Two Towers'}, 'Boo
k3': {'Author': 'J.R.R. Tolkien', 'Genre': 'Epic fantasy', 'Price': 100, 'Ti
tle': 'The Return of the King'}, 'Book4': {'Author': 'Paulo Coelho', 'Genr
e': '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'}
```

In [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': 10
0, 'Title': 'The Fellowship of the Ring'}, 'Book2': {'Author': 'J.R.R. Tolki
en', 'Genre': 'Epic fantasy', 'Price': 100, 'Title': 'The Two Towers'}, 'Boo
k3': {'Author': 'J.R.R. Tolkien', 'Genre': 'Epic fantasy', 'Price': 100, 'Ti
tle': 'The Return of the King'}, 'Book4': {'Author': 'Paulo Coelho', 'Genr
e': 'Fiction', 'Price': 100, 'Title': 'Brida'}, 'Books': {'Best_Sellers':
{'-O8LLuZGE4Tvg7RdVbAC': {'Author': 'Paulo Coelho', 'Genre': 'Fiction', 'Pri
ce': 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'}
Books: {'Best_Sellers': {'-O8LLuZGE4Tvg7RdVbAC': {'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'}
```

In [15]:

```
# 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})
```

```
{'-O8LLuZGE4Tvg7RdVbAC': {'Author': 'Paulo Coelho', 'Genre': 'Fiction', 'Pri
ce': 100, 'Title': 'Brida'}}
```

In [16]:
```
# Delete all best_seller data.
#
ref = db.reference("/Books/Best_Sellers/")
best_sellers = ref.get()
print(best_sellers)
print(type(best_sellers))
```

```
{'-O8LLuZGE4Tvg7RdVbAC': {'Author': 'Paulo Coelho', 'Genre': 'Fiction', 'Pri
ce': 100, 'Title': 'Brida'}}
<class 'dict'>
```

In [ ]:
```

```

In [6]:
```python
import random
import time
import firebase_admin
from firebase_admin import credentials, db

# Initialize Firebase Admin SDK with the correct certificate path
cred_obj = firebase_admin.credentials.Certificate(
    r'C:\Users\ramma\Downloads\sit225-ri-firebase-adminsdk-octoc-af69f7a7ab.
)

# Initialize Firebase app (ensure only one initialization)
if not firebase_admin._apps:
    default_app = firebase_admin.initialize_app(cred_obj, {
        'databaseURL': 'https://sit225-ri-default-rtdb.firebaseio.com/'
    })

# Generate random data for DHT22 sensor
dht_data = {
    "sensor_name": "DHT22",
    "timestamp": time.strftime('%Y-%m-%d %H:%M:%S'),
    "temperature": round(random.uniform(20, 30), 2),
    "humidity": random.randint(40, 70)
}

# Push DHT22 data to Firebase
ref = db.reference("/DHT22_data")
ref.push(dht_data)

# Generate random data for SR04 sensor (Ultrasonic sensor)
sr04_data = {
    "sensor_name": "SR04",
    "timestamp": time.strftime('%Y-%m-%d %H:%M:%S'),
    "distance": random.randint(100, 200)
}

# Push SR04 data to Firebase
ref = db.reference("/SR04_data")
ref.push(sr04_data)
```

```
Data successfully pushed to Firebase
```

In [ ]:
```python

```

SIT225: Data wrangling

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

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

Read the Data with Pandas

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

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

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

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

print(csv_data)

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

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

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

Access the Column

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

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

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

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

Access the Row

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

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

print(csv_data.iloc[5])

print()

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

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

21
```

Show Data Based on Range

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

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

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

Using Numpy to Show the Statistic Information

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

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

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

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

Handling Missing Value

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

```
False
```

In [8]:
```
# We will use another data source with missing values to practice this 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
```

In [ ]:
```

```

Ways to deal with missing values.

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

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

Filling with Mean Values

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

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 occurs because the mean() function in Pandas is attempting to calculate the mean for columns that contain non-numeric data, specifically the 'Genre' column, which contains strings such as 'Male' and 'Female'. Since these values are not numeric, Pandas cannot compute a mean for this column, leading to the TypeError.

Solution

To solve this issue, I think dropping the column before applying would work, but in the case that non numeric data is also required it can just be excluded from

numeric calculations

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

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

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

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

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

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

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

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

Filling with Median

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

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

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

In [ ]: