

## APPENDIX B: USE OF AI TOOL(S) IN PROJECT WORK

Each team member should indicate either A or B.

- A. I affirm that my contribution(s) to the lab work is my own, produced without help from any AI tool(s)
- B. I affirm that my contribution(s) to the lab work has been produced with the help from AI tool(s)

Full Name	Date	A or B
Mahtaney Tej Pravin	10/11/2025	A
MALIK PARTH	10/11/2025	A
Keane Phoon Kai Jie	10/11/2025	A
Emmanuel Yap Wei'en	10/11/2025	B
Casey Bodhiyano	22/10/2025 02/11/2025	B

By including this information in your jupyter notebook, you declare that the above affirmation made is true and that you have read and understood NTU's policy on the use of AI tools.

If any team member answered B, the team member(s) must indicate and replicate the table below for every instance AI tool(s) is used.

Name of AI tool	Copilot
Input prompt	How does a boxplot work?
Date generated	11 November 2025
Output generated	<p><i>A boxplot (also called a box-and-whisker plot) is a graphical representation of the distribution of a dataset. It shows the following key statistics:</i></p> <ul style="list-style-type: none"> <li>• <i>The minimum value (excluding outliers)</i></li> <li>• <i>The first quartile (Q1, 25th percentile)</i></li> <li>• <i>The median (Q2, 50th percentile)</i></li> <li>• <i>The third quartile (Q3, 75th percentile)</i></li> </ul>

- |  |   |
|--|---|
|  | <ul style="list-style-type: none"><li>• <i>The maximum value (excluding outliers)</i></li><li>• <i>Outliers (points outside <math>1.5 \times IQR</math> from the quartiles)</i></li></ul> |
|--|---|

*How it works:*

- *The box spans from Q1 to Q3, showing the interquartile range (IQR).*
- *A line inside the box marks the median.*
- *"Whiskers" extend from the box to the minimum and maximum values within  $1.5 \times IQR$ .*
- *Points outside the whiskers are plotted as outliers.*

*Boxplots are useful for visualizing the spread, central tendency, and outliers in your data.*

Output screenshot

how does a boxplot work?

A boxplot (also called a box-and-whisker plot) is a graphical representation of the distribution of a dataset. It shows the following key statistics:

- The minimum value (excluding outliers)
- The first quartile (Q1, 25th percentile)
- The median (Q2, 50th percentile)
- The third quartile (Q3, 75th percentile)
- The maximum value (excluding outliers)
- Outliers (points outside  $1.5 \times \text{IQR}$  from the quartiles)

How it works:

- The box spans from Q1 to Q3, showing the interquartile range (IQR).
- A line inside the box marks the median.
- "Whiskers" extend from the box to the minimum and maximum values within  $1.5 \times \text{IQR}$ .
- Points outside the whiskers are plotted as outliers.

Boxplots are useful for visualizing the spread, central tendency, and outliers in your data.

⟳ ⌛ ⌘ ⌚ ...

GPT-4.1 • 0x



main.ipynb • Cell 19 +

Add context (#), extensions (@), command

Agent ▾ Auto ▾

✖ ➔ ▾

Impact on submission	<i>Done to help explain boxplot in my script for presentation</i>

Name of AI tool	<i>ChatGPT</i>
Input prompt	<i>Can you edit the csv such that only the gpa variation increases amongst the first 30 groups, only the gender variation increases amongst the next 30 groups, only the school variation increases amongst the next 30 groups, and then all of the criteria variation increases on the last 30 groups. I want to test my algorithm's flexibility</i>
Date generated	<i>22 October 2025</i>
Output generated	<pre>import pandas as pd import numpy as np  # Load CSV with correct column names file_path = "/mnt/data/records.csv" df = pd.read_csv(file_path)  # Extract unique values groups = sorted(df['Tutorial Group'].unique()) schools = df['School'].unique().tolist() genders = df['Gender'].unique().tolist()  # Function to generate realistic GPAs with occasional outliers def generate_gpa(mean=3.0, std=0.3, outlier_frac=0.05):     if np.random.rand() &lt; outlier_frac:         return np.random.uniform(0.0, 1.0) if np.random.rand() &lt; 0.5 else np.random.uniform(4.5, 5.0)     return np.clip(np.random.normal(mean, std), 0.0, 5.0)  # Function to apply modification rules per group def modify_group_data(group_num, group_df):     new_df = group_df.copy()     if 1 &lt;= group_num &lt;= 30:         # GPA variation (normal → extreme)         intensity = (group_num - 1) / 29</pre>

```

std = 0.3 + 1.2 * intensity
outlier_frac = 0.05 + 0.25 * intensity
new_df['CGPA'] = [generate_gpa(3.0, std, outlier_frac) for _ in
range(len(group_df))]

elif 31 <= group_num <= 60:
    # Gender variation (balanced → dominated)
    intensity = (group_num - 31) / 29
    dominant_gender = np.random.choice(genders)
    ratio = 0.5 + 0.45 * intensity # starts balanced, ends dominated
    dominant_count = int(len(group_df) * ratio)
    other_gender = [g for g in genders if g != dominant_gender][0]
    new_genders = [dominant_gender]*dominant_count +
[other_gender]*(len(group_df) - dominant_count)
    np.random.shuffle(new_genders)
    new_df['Gender'] = new_genders

elif 61 <= group_num <= 90:
    # School variation (diverse → homogeneous)
    intensity = (group_num - 61) / 29
    num_schools = max(1, int(len(schools) * (1 - 0.9 * intensity)))
    chosen_schools = np.random.choice(schools, num_schools,
replace=False)
    new_df['School'] = np.random.choice(chosen_schools, len(group_df))

elif 91 <= group_num <= 120:
    # All variations extreme
    intensity = (group_num - 91) / 29
    # GPA variation
    std = 0.5 + 1.5 * intensity
    outlier_frac = 0.1 + 0.3 * intensity
    new_df['CGPA'] = [generate_gpa(3.0, std, outlier_frac) for _ in
range(len(group_df))]

    # Gender variation
    dominant_gender = np.random.choice(genders)
    ratio = 0.5 + 0.45 * intensity
    dominant_count = int(len(group_df) * ratio)
    other_gender = [g for g in genders if g != dominant_gender][0]
    new_genders = [dominant_gender]*dominant_count +
[other_gender]*(len(group_df) - dominant_count)
    np.random.shuffle(new_genders)
    new_df['Gender'] = new_genders

    # School variation

```

	<pre> num_schools = max(1, int(len(schools) * (1 - 0.8 * intensity))) chosen_schools = np.random.choice(schools, num_schools, replace=False) new_df['School'] = np.random.choice(chosen_schools, len(group_df))  return new_df  # Apply modifications across all groups modified_groups = [modify_group_data(group_num, df[df['Tutorial Group'] == group_num]) for group_num in groups] modified_df = pd.concat(modified_groups, ignore_index=True)  # Round GPA to two decimals modified_df['CGPA'] = modified_df['CGPA'].round(2)  # Save modified file with original headers output_path = "/mnt/data/records_modified.csv" modified_df.to_csv(output_path, index=False)  output_path </pre>
Output screenshot	<p>It seems like I can't do more advanced data analysis right now. Please try again later.</p> <p>Once the tool becomes available again, I'll be able to generate the modified CSV file where:</p> <ul style="list-style-type: none"> <li>Groups 1–30 have <b>increasing GPA variation (with realistic outliers)</b></li> <li>Groups 31–60 have <b>increasing gender imbalance</b></li> <li>Groups 61–90 have <b>increasing school homogeneity</b></li> <li>Groups 91–120 have <b>all three variations increasing together</b></li> </ul> <p>Would you like me to show you the <b>code logic or preview</b> of how the modified data would look first (you can check before running again)? <a href="#">»</a></p> <p><a href="#">🔗</a> <a href="#">⬇️</a> <a href="#">🖨️</a> <a href="#">↑</a> <a href="#">🕒</a> ...</p>
Impact on submission	<p><i>A prototype for modifying dataset for generating Test Cases.</i></p> <p><i>It was originally designed to make only one test case, which I later changed it into making 5 test cases now.</i></p> <p><i>The generated code also used numpy and pandas, which is hard to translate over the logic of each “mysterious” functions into the one we have now.</i></p> <p><i>Ultimately decided to change the initial variables’ values for a more accurate judgement.</i></p>


Name of AI tool	<i>ChatGPT</i>
Input prompt	<i>how to animate graphs and turn it into gifs in matplotlib</i>
Date generated	<i>2 November 2025</i>
Output generated	<pre>import matplotlib.pyplot as plt import numpy as np from matplotlib.animation import FuncAnimation  fig, ax = plt.subplots() scat = ax.scatter([], [], c="red") ax.set_xlim(-1, 1) ax.set_ylim(-1, 1) ax.set_title("Random Scatter Animation")  def update(frame):     x = np.random.uniform(-1, 1, 10)     y = np.random.uniform(-1, 1, 10)     scat.set_offsets(np.c_[x, y])     return scat,  ani = FuncAnimation(fig, update, frames=50, interval=200, blit=True) ani.save("scatter.gif", writer="pillow")</pre>

Output  
screenshot

### ✿ Step 1: Import the Required Libraries

You'll need:

- `matplotlib.pyplot` for plotting
- `matplotlib.animation.FuncAnimation` for animation
- Optionally, `Pillow` to save as GIF

python

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
```

### ✿ Step 2: Create Your Figure and Initial Plot

python

```
fig, ax = plt.subplots()
xdata, ydata = [], []
(line,) = ax.plot([], [], lw=2)

ax.set_xlim(0, 2*np.pi)
ax.set_ylim(-1, 1)
ax.set_title("Sine Wave Animation")
```



### 💬 Step 3: Define the Animation Functions

python

```
def init():
    line.set_data([], [])
    return (line,)

def update(frame):
    xdata.append(frame)
    ydata.append(np.sin(frame))
    line.set_data(xdata, ydata)
    return (line,)
```

### 🎬 Step 4: Create and Run the Animation

python

```
frames = np.linspace(0, 2*np.pi, 128)
ani = FuncAnimation(fig, update, frames=frames, init_func=init, blit=True, interval=50)
plt.show()
```

#### 🎬 Step 4: Create and Run the Animation

python

```
frames = np.linspace(0, 2*np.pi, 128)
ani = FuncAnimation(fig, update, frames=frames, init_func=init, blit=True, interval=50)
plt.show()
```

- `interval=50` → delay between frames (ms)
- `blit=True` → only redraw changed parts (faster)

#### 💾 Step 5: Save Animation as a GIF

Install Pillow if not already:

bash

```
pip install pillow
```

Then save:

python

```
ani.save("sine_wave.gif", writer="pillow")
```

	<pre>python</pre> <pre>import matplotlib.pyplot as plt import numpy as np from matplotlib.animation import FuncAnimation  fig, ax = plt.subplots() scat = ax.scatter([], [], c="red") ax.set_xlim(-1, 1) ax.set_ylim(-1, 1) ax.set_title("Random Scatter Animation")  def update(frame):     x = np.random.uniform(-1, 1, 10)     y = np.random.uniform(-1, 1, 10)     scat.set_offsets(np.c_[x, y])     return scat,  ani = FuncAnimation(fig, update, frames=50, interval=200, blit=True) ani.save("scatter.gif", writer="pillow")</pre> <hr/> <p> <b>Notes</b></p> <p><input checked="" type="checkbox"/> You can also export as <code>.mp4</code> using FFmpeg: </p> <hr/> <p> <b>Notes</b></p> <p><input checked="" type="checkbox"/> You can also export as <code>.mp4</code> using FFmpeg:</p> <pre>python  ani.save("animation.mp4", writer="ffmpeg")</pre> <p><input checked="" type="checkbox"/> For more complex plots (e.g., multiple lines, histograms, or boxplots), you can simply update each <code>update()</code> call.</p> <hr/> <p>Would you like me to show how to animate a boxplot changing over time (e.g., CGPA distribution methods)? That's a nice use case for your type of data.</p>
Impact on submission	<i>This helps me understand how to use FuncAnimation from matplotlib, but it was a bit hard to integrate into existing code, which was made to produce .png files of the result before.</i>

--	--