Ariana Namei

Zanesha Chowdhury

SI 201

12/15/25

Ctrl Alt Elite Final Project Report

Our original goals for the project were quite different from what we actually ended up doing. At first we wanted to get data on the prevalence of Pokémon in pop culture like song lyrics and anime. We also wanted to see how common Pokémon names were in various word clouds, as well as data on Pokémon color to see if it had any correlation to common colors in songs. We originally planned to use 4 APIs: PokeAPI, Jikan anime API, Ksoft lyrics API, and Word Cloud API. The data we planned to collect from PokeAPI was the first 150 Pokémon. From those Pokémon, we planned to collect Pokémon ID, Pokémon name, Base experience, Pokémon height/weight, and Pokémon primary type. From the KSoft lyrics API, we planned to collect song lyric metadata containing keywords about Pokémon, like names of Pokémon names. From those songs with keywords, we would collect song ID, song title, artist, URL, and what keyword we matched. From the Jikan Anime API, we planned to collect Pokémon anime episode data. We planned to collect episode ID, title, air date, episode number, and series. For the Word Cloud API, our plan was to collect Word Cloud metadata. This includes word cloud ID, source text, source text used to generate the cloud, category of words, timestamp, and image URL from the API. On an individual word level, we planned to collect data about word frequency, font size, whether the word is a Pokémon name, or whether it is an anime-related keyword.

However, those plans were found to be quite difficult, so we decided to change our approach. We only ended up using three API's instead of four, because our group ended up shrinking down to two, which will be touched on in the next paragraph. Our APIs changed because we realized that many of the APIs we were using previously did not work, and we realized that we did not need an overarching theme that is shared between all APIs. The APIs we used for our final project were PokeAPI, Spotipy API, and Weather API. Since we decided to have individual goals for each API, we had different data collected. For PokeAPI we wanted to know the average base experience and average weight per Pokémon type. From the Spotipy API, we wanted to know the average track popularity per artist. Finally, for the Weather API, we wanted to know wind speed against temperature for a few major cities within the United States.

In order to accomplish these goals, we collected different data from each API. We had two main tables for PokeAPI: pokemon and pokemon_stats. For the pokemon table, we collected Pokémon names, base experience, height, weight, and primary type. In the pokemon_stats table, we collected health, attack, defence, and speed points for each Pokémon. For the Spotipy API, we collected track titles, artist names, and popularity scores. Finally, for the Weather API, the data we collected were city names, the dates the weather was collected, temperature, wind speed, and forecasts. From all the data that we collected, we successfully completed our individual goals for each API.

While completing this project, we faced many problems. The biggest one was the decision to reach out to the teaching team about one of our group members not contributing and causing issues the night before we were originally scheduled to present. Another problem was finding APIs that actually worked. Many of the APIs on the GitHub webpage were not working, and finding 4 APIs that worked proved to be a bit challenging. We originally thought that we

needed to relate all of our APIs to one another and create some kind of overarching storyline, which made it more challenging at first, but then we eventually realized that was not necessary. We also struggled with removing the duplicate string data from our tables, separating our program into 3 separate files, and pulling and pushing from GitHub. The fact that we could not code at the same time also proved to be a challenge for collaboration.

For our final project, we had four total calculations. Half of them calculated data from PokeAPI, and the other two were from Spotipy API and Weather API. For PokeAPI, we calculated average base experience by Pokémon type and the average weight per Pokémon type. For Spotipy, we calculated average popularity per artist, and finally for Weather API, we calculated temperature against wind speed.

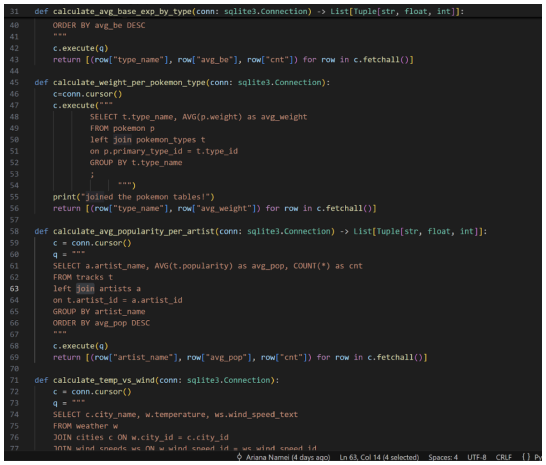
```
import os
import sqlite3
from typing import List, Tuple
import spotipy
import csv


# spotify
import spotipy
from spotipy.oauth2 import SpotifyClientCredentials
from database import DB_PATH

# API keys
SPOTIPY_CLIENT_ID = os.getenv("SPOTIPY_CLIENT_ID", "fc80ead3b4f0410da95885d93e837534")
SPOTIPY_CLIENT_SECRET = os.getenv("SPOTIPY_CLIENT_SECRET", "2535128eadda464c8890983d1ac28786")

#spotify client
spotify_client = None
if SPOTIPY_CLIENT_ID and SPOTIPY_CLIENT_SECRET:
    auth_manager = SpotifyClientCredentials(client_id=SPOTIPY_CLIENT_ID, client_secret=SPOTIPY_CLIENT_SECRET)
    spotify_client = spotipy.Spotify(auth_manager=auth_manager)
else:
    print("Spotify credentials not set. Skipping Spotify fetch.")

# calculations
def calculate_avg_base_exp_by_type(conn: sqlite3.Connection) -> List[Tuple[str, float, int]]:
    c = conn.cursor()
    q = """
    SELECT t.type_name, AVG(p.base_experience) AS avg_be, COUNT(*) as cnt
    FROM pokemon p
    left join pokemon_types t
    on p.primary_type_id = t.type_id
    WHERE primary_type_id IS NOT NULL
    GROUP BY t.type_name
```



```
def calculate_avg_base_exp_by_type(conn: sqlite3.Connection) -> List[Tuple[str, float, int]]:
    ORDER BY avg_be DESC
    """
    c.execute(q)
    return [(row["type_name"], row["avg_be"], row["cnt"]) for row in c.fetchall()]

def calculate_weight_per_pokemon_type(conn: sqlite3.Connection):
    c=conn.cursor()
    c.execute("""
            SELECT t.type_name, AVG(p.weight) as avg_weight
            FROM pokemon p
            left join pokemon_types t
            on p.primary_type_id = t.type_id
            GROUP BY t.type_name
            ;
                """)
    print("joined the pokemon tables!")
    return [(row["type_name"], row["avg_weight"]) for row in c.fetchall()]

def calculate_avg_popularity_per_artist(conn: sqlite3.Connection) -> List[Tuple[str, float, int]]:
    c = conn.cursor()
    q = """
    SELECT a.artist_name, AVG(t.popularity) as avg_pop, COUNT(*) as cnt
    FROM tracks t
    left join artists a
    on t.artist_id = a.artist_id
    GROUP BY artist_name
    ORDER BY avg_pop DESC
    """
    c.execute(q)
    return [(row["artist_name"], row["avg_pop"], row["cnt"]) for row in c.fetchall()]

def calculate_temp_vs_wind(conn: sqlite3.Connection):
    c = conn.cursor()
    q = """
    SELECT c.city_name, w.temperature, ws.wind_speed_text
    FROM weather w
    JOIN cities c ON w.city_id = c.city_id
```

```python
def calculate_temp_vs_wind(conn: sqlite3.Connection):
    JOIN wind_speeds ws ON w.wind_speed_id = ws.wind_speed_id
    WHERE w.temperature IS NOT NULL
      AND ws.wind_speed_text IS NOT NULL
    """
    c.execute(q)

    results = []
    for row in c.fetchall():

        wind_raw = row["wind_speed_text"]
        try:
            wind_value = int(wind_raw.split()[0])
        except:
            continue

        results.append((row["city_name"], row["temperature"], wind_value))

    return results
def write_csv(filename: str, headers: List[str], rows: List[Tuple]):
    """Writes rows of tuples to a CSV file with given headers."""
    with open(filename, "w", newline="", encoding="utf-8") as f:
        writer = csv.writer(f)
        writer.writerow(headers)
        writer.writerows(rows)
    print(f"Wrote {filename} ({len(rows)} rows)")


def example_run():

    conn = sqlite3.connect(DB_PATH)
    conn.row_factory = sqlite3.Row

    # Pokemon calculations
    avg_be = calculate_avg_base_exp_by_type(conn)
    write_csv(
        "pokemon_base_exp_by_type.csv",
        ["type_name", "avg_base_experience", "count"],
        avg_be
    )

    weights = calculate_weight_per_pokemon_type(conn)
    write_csv(
        "pokemon_weight_by_type.csv",
        ["type_name", "avg_weight"],
        weights
    )

    # spotify calculations
    avg_pop = calculate_avg_popularity_per_artist(conn)
    write_csv(
        "spotify_avg_popularity.csv",
        ["artist_name", "avg_popularity", "count"],
        avg_pop
    )

    # weather calculations
    temp_wind = calculate_temp_vs_wind(conn)
    write_csv(
        "temp_vs_wind.csv",
        ["city_name", "temperature", "wind_speed_mph"],
        temp_wind
    )

    conn.close()
    print("All CSV files written.")



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

pokemon_base_exp_by_type.csv

```
type_name,avg_base_experience,count
fairy,165.0,2
ghost,143.0,3
grass,142.55555555555554,9
fire,133.55555555555554,9
psychic,132.4,5
fighting,130.0,5
water,127.61111111111111,18
electric,124.8,5
rock,124.25,4
poison,123.58333333333333,12
normal,118.07142857142857,14
ground,105.0,4
bug,99.6,10
```

pokemon_weight_by_type.csv

```
type_name,avg_weight
bug,164.9
electric,224.8
fairy,237.5
fighting,560.0
fire,496.44444444444446
ghost,135.66666666666666
grass,198.22222222222223
ground,189.0
normal,233.14285714285714
poison,309.9166666666667
psychic,464.0
rock,1587.5
water,515.3333333333334
```

spotify_avg_popularity.csv

```
artist_name,avg_popularity,count
Taylor Swift,76.33333333333333,39
Gunna,76.0,1
Drake,75.93333333333334,15
Rick Ross,75.5,2
Meek Mill,75.0,1
Gracie Abrams,75.0,1
Lil Baby,74.0,1
GP Explorer,68.0,2
Adele,58.10344827586207,29
Adèle Castillon,48.0,4
ADÉLA,45.0,3
Adelitas Way,36.0,2
```

temp_vs_wind.csv

```
city_name,temperature,wind_speed_mph
"Chicago, IL",45.0,10
"New York, NY",33.0,8
"Los Angeles, CA",54.0,0
"Chicago, IL",24.0,10
"Chicago, IL",44.0,15
"Ann Arbor, MI",30.0,2
"New York, NY",40.0,13
"Detroit, MI",6.0,12
"New York, NY",22.0,17
"Los Angeles, CA",55.0,0
"New York, NY",50.0,6
"Ann Arbor, MI",2.0,10
"Chicago, IL",37.0,5
"Detroit, MI",17.0,6
"Chicago, IL",14.0,15
"Chicago, IL",14.0,10
"Los Angeles, CA",77.0,5
"New York, NY",36.0,6
"Detroit, MI",23.0,8
"Ann Arbor, MI",8.0,6
"New York, NY",26.0,10
"Chicago, IL",3.0,15
"Detroit, MI",32.0,5
"Los Angeles, CA",53.0,5
"Detroit, MI",12.0,6
"Ann Arbor, MI",10.0,2
"Los Angeles, CA",53.0,5
"Chicago, IL",45.0,10
"Ann Arbor, MI",14.0,5
"Detroit, MI",33.0,8
"Chicago, IL",3.0,15
"Los Angeles, CA",54.0,0
"Detroit, MI",46.0,16
"New York, NY",31.0,9
"Chicago, IL",25.0,10
"Los Angeles, CA",74.0,0
"New York, NY",22.0,17
"New York, NY",31.0,12
```

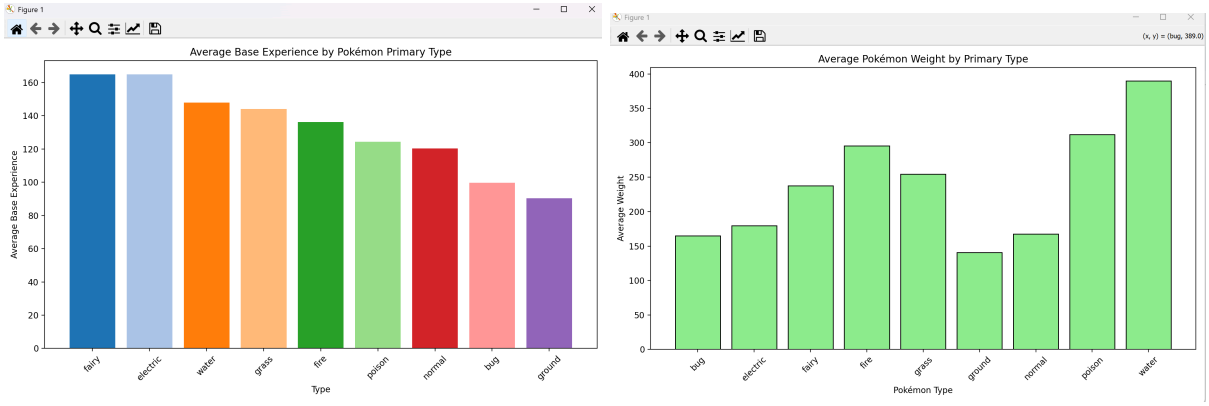
temp_vs_wind.csv

```
"Ann Arbor, MI",28.0,7
"New York, NY",40.0,13
"Los Angeles, CA",76.0,0
"New York, NY",42.0,12
"Chicago, IL",32.0,10
"Ann Arbor, MI",44.0,15
"Detroit, MI",22.0,10
"Los Angeles, CA",80.0,0
"Detroit, MI",6.0,12
"Chicago, IL",24.0,10
"Detroit, MI",29.0,7
"Chicago, IL",30.0,10
"New York, NY",31.0,9
"New York, NY",40.0,6
"Chicago, IL",37.0,5
"Los Angeles, CA",81.0,0
"Detroit, MI",24.0,12
"Chicago, IL",30.0,10
"Los Angeles, CA",55.0,5
"Los Angeles, CA",53.0,5
"New York, NY",35.0,14
"Ann Arbor, MI",2.0,10
"Ann Arbor, MI",21.0,8
"Ann Arbor, MI",10.0,2
"New York, NY",29.0,9
"Detroit, MI",46.0,16
"Chicago, IL",24.0,10
"Ann Arbor, MI",14.0,5
"Ann Arbor, MI",20.0,12
"Detroit, MI",23.0,8
"Ann Arbor, MI",19.0,9
"Chicago, IL",32.0,10
"New York, NY",50.0,6
"Detroit, MI",21.0,8
"Detroit, MI",12.0,6
"Los Angeles, CA",55.0,0
"Ann Arbor, MI",23.0,12
"New York, NY",36.0,6
"Detroit, MI",23.0,8
```

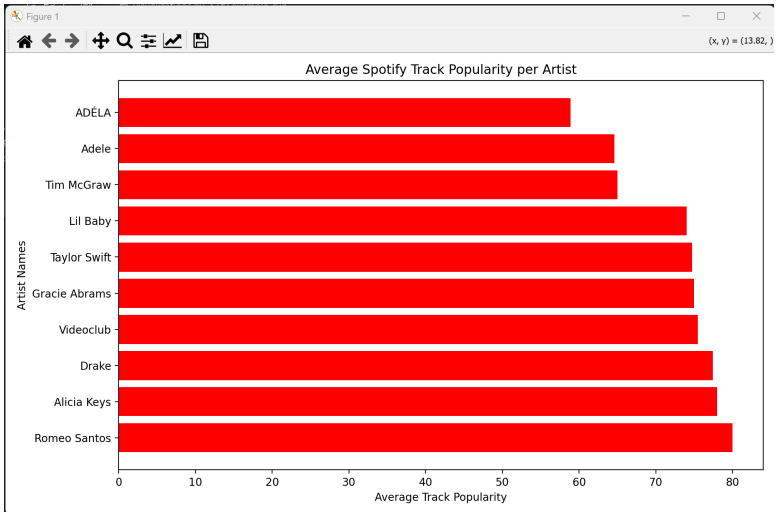
temp_vs_wind.csv

```
"Detroit, MI",23.0,8
"Los Angeles, CA",54.0,0
"New York, NY",26.0,10
"Los Angeles, CA",55.0,5
"New York, NY",28.0,9
"Detroit, MI",33.0,8
"Ann Arbor, MI",2.0,10
"Ann Arbor, MI",8.0,6
"Ann Arbor, MI",21.0,8
"Ann Arbor, MI",21.0,10
"Los Angeles, CA",77.0,5
"New York, NY",31.0,9
"Detroit, MI",32.0,5
"New York, NY",40.0,13
"Ann Arbor, MI",32.0,7
"Detroit, MI",26.0,12
"Los Angeles, CA",54.0,0
"Detroit, MI",24.0,12
"Detroit, MI",6.0,12
"Chicago, IL",20.0,5
"Ann Arbor, MI",30.0,2
"Detroit, MI",22.0,10
"Detroit, MI",29.0,7
"Chicago, IL",37.0,5
```

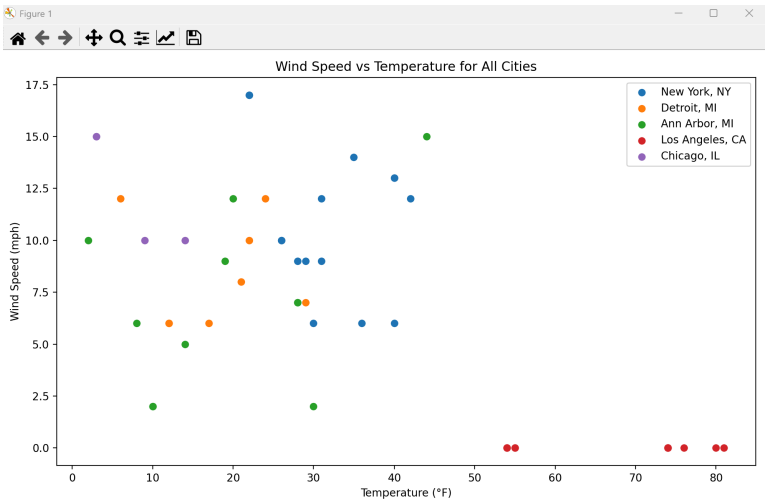
Each visualization correlated to a specific calculation. We graphed PokeAPI data into bar graphs, Spotipy data into a horizontal bar graph, and Weather API into a scatter plot.



Bar graphs for PokeAPI: Left graph calculates average base experience and the right graph calculates average pokemon weight



Horizontal bar graph for Spotipy API: calculates average Spotify track popularity per artist

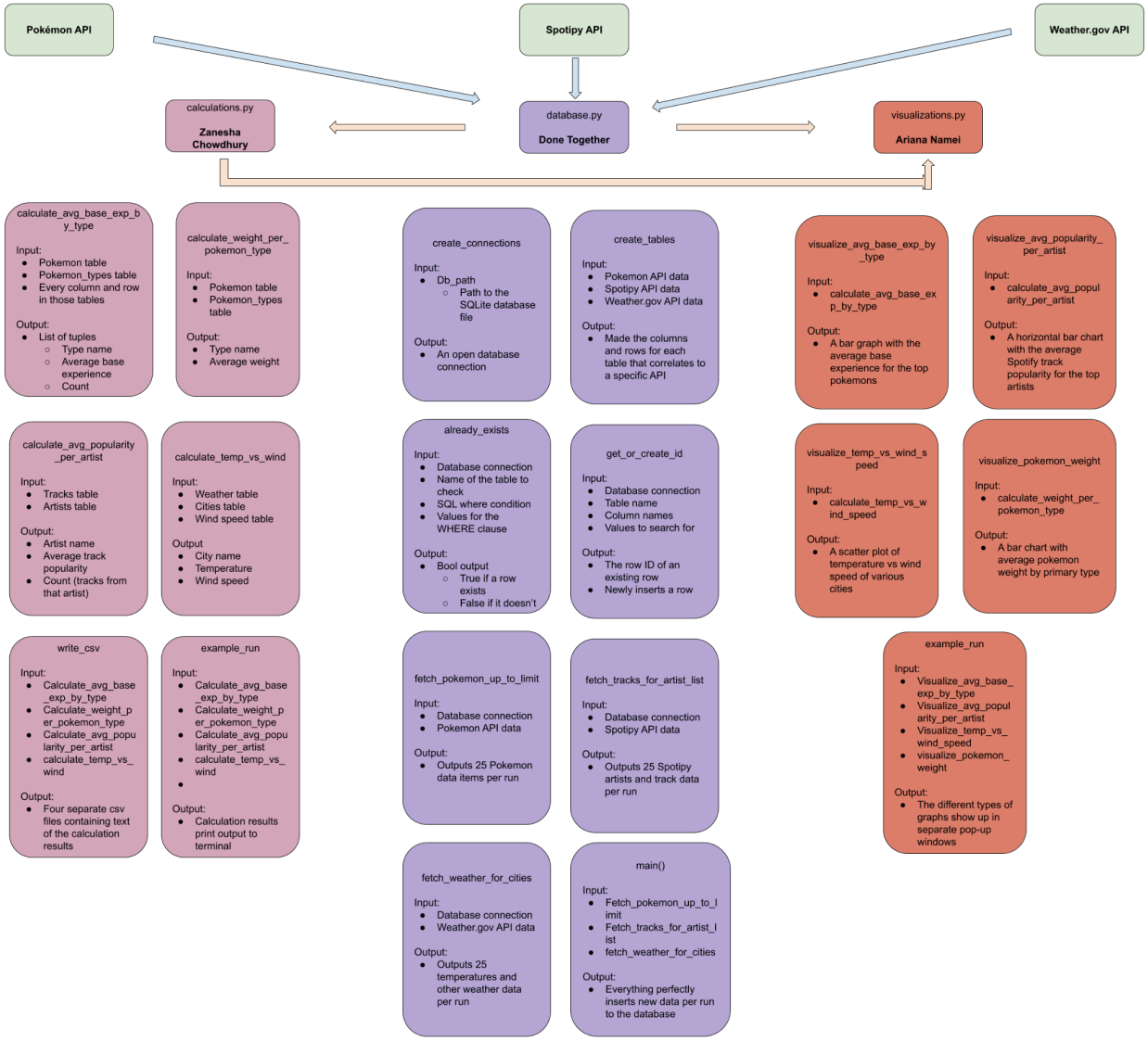


Scatter plot for Weather API: tracks wind speed versus temperature for major cities

In order to run our code, the first file to run is the database file. This should be run four times to get 100 entries in the table. Afterwards, the calculation file should be run next. The last file that should be run is the visualizations file. However, before running any files, Spotipy's package should be downloaded before running any of these files.

In terms of splitting up the work, we decided to split it up quite evenly. Since 80% of the work went under the database file and picking out what APIs to use, we decided to work on that

together. Additionally, the calculations and visualization files did not take up nearly as much time as the database, so we decided to assign those files separately. Zanesha took care of the calculations and Ariana took care of the visualizations. In order to save space, and for cleanliness, we decided to omit some arrows in our function diagram below. Every box with a designated color is grouped together and part of one specific file.



Github Repository Link: https://github.com/zaneshac/SI-201-Duo-Final-Project.git

Date	Issue Description	Location of Resource	Result (did it solve the issue?)
12/4 (first day of working on project as duo)	Visualizations and everything being in one file	Collegian Building (office hours)	Visualizations were removed
12/5	String duplicates and removal resulting in nulls	Collegian building (office hours)	Some string duplicates were removed but not all nulls
12/7	String duplicate removals resulted in table nulls	Spandan's office hours (virtual)	Some nulls were removed but not all
12/8	String duplicate removal nulls	Dana (office hours)	Yes, all were removed
12/8	Duplicate songs, not getting enough weather results	UMSI lounge (peer tutoring)	Yes, it solved the issue
12/9	Join issues, join needed to be moved	Peer tutoring (virtual) Peer tutoring (UMSI	All issues were solved except for the

	to calculations, the same temperature data was in more than one column, visualizations and calculations did not match the removal of duplicate string data	lounge) Spandan's office hours (virtual)	same temperature data appearing twice
12/10	Same temperature data appearing twice, graph was not complicated enough	Peer tutoring (virtual)	Yes, all issues were solved