Crazy Coders: https://github.com/serenashen1/FinalProject-CrazyCoders.git

Charlotte Parent, Nicolina Moore, Serena Shen

1. The goals for your project include what APIs/websites you planned to work with and what data you planned to gather (10 points)

In our Project Plan, we had discussed using four APIs in order to compare different children's shows and networks from our childhood. The APIs we had planned to use were the following: Disney API, PBS Kids API, and Cartoon Network API.

We wanted to gather data regarding the different television shows on each network, the hours they operated during the day, the amount of views each show/network received, the rating of each show, and how long each show ran for.

With this data, we were going to calculate the most popular show per channel, the least popular show per channel, and the most popular time to watch each show/channel.

We had planned to create visualizations on Matplotlib of this data by using…

- A graph/scatter plot to compare each network and the times people watched the shows
- A graph showing change in popular/most-watched show over time for each channel
- A chart depicting most popular and least popular shows on each channel
- A bar chart showing average rating per channel

After realizing we did not have enough items to meet the requirements through these APIs, we decided to change the direction of our project and create a new plan with different APIs and goals in mind.

2. The goals that were achieved including what APIs/websites you actually worked with and what data you did gather (10 points)

The first goal we achieved was to find our new APIs. Since we switched the direction of our project, we looked on the website API Ninjas and ended up finding APIs we could work with. We used an Airport API, Air Quality API, Country API, and for extra credit, we used a fourth API called Weather API for the extra credit.

Air Quality API: `https://api.api-ninjas.com/v1/airquality`
Airport API: `https://api.api-ninjas.com/v1/airports`
Country API: `https://api.api-ninjas.com/v1/country`
Weather API: `https://api.api-ninjas.com/v1/weather`

The next goal we achieved was to create relationships based off of the data gathered, as we were interested in the averages within the air and travel economy, as they are closely related. From there, we made three calculations based on the data we collected:

1) Our first calculation was to determine the average number of refugees in each country based on the airport's timezone.

We wanted to determine the best time for volunteers and people around the world who dedicate their time and have interest in helping refugees through the community and help provide resources, so determining the airport with the highest number of refugees at a given time helped us achieve our goal – as these volunteers now know the best time/time zone to help.

2) Our second calculation was to determine the number of tourists (more than 200 per country) and the average elevation of that country.

We wanted to determine the number of tourists per country and the average elevation of that country, as it would help us determine what kinds of destinations/vacations people are most interested in. If there are a lot of tourists traveling to higher elevations, this would help us conclude that travelers enjoy more mountainous vacations, where if the elevation is lower we can assume travelers enjoy visiting warmer locations closer to sea level.

3) Our third calculation was to determine the average air quality in a country and the country's average population size.

We wanted to determine the average air quality in countries and determine their average population as it will help us conclude how certain countries use their resources such as natural gas, gas, and wood – and could help us conclude how much resources, waste, or fuel is used based on how healthy or unhealthy the air is.

We used Google Sheets to organize the content within each API and our databases before we took the next step of physically calculating the data and creating the visualizations.

The data we evaluated for each calculation:

airports
airports
airport_locations
weather
country_info
air_quality

airports *Airport API*

Name	Type	Description	Key
ID	int	Airport ID	Primary key
city_ID	int of city		
IATA_code			
name	name of airport	name of airport	

airport_locations *Airport API*

Name	Type	Description	Key
ID	int	Airport location ID	primary key
country_ID	int	ID of the country-- link to country table	
city_name	text	City of the airport	
region	text	Region of the airport	
timezone	date	Timezone of the airport	
latitude	int	Lat. of the airport	
longitude	int	Long. of the airport	
elevation	int	Elevation of the airport	

weather *Weather API*

Name	Type	Description	Key
ID	int	ID	Primary key
city_ID	int	ID of city	
country_ID	int	ID of country	
wind_speed	int	wind speed	
wind_degrees	int	Temperature (deg) of wind	
temperature	int	Temperature of location	
humidity	int	Humidity %	
sunset	string	Sunset time	
sunrise	string	Sunrise time	
cloud_pct	int	Cloud cover percent	
feels_like	int	"Feels like" temperature	
max_temp	int	Max temperature	
min_temp	int	Min temperature	

country *Country API*

Name	Type	Description	Key
ID	Int	Country ID	Primary key
name	text		
gdp	int		
surface_area	int		
life_expectancy_male	int		
imports	text	Number of imports	
currency_name	text		
urban_population_growth	int		
capital	text	Capital	
tourists	int		
life_expectancy_female	int		
threatened_species	text		
population	int		
urban_population	int		
pop_growth	int		
region	text		
pop_density	int		
refugees	int		

air_quality *Air Quality API*

Name	Type	Description	Key
ID	int	ID	Primary key
city_ID	int	city ID	
country_ID	int	country_ID	
AQI	int	Air Quality Index	
CO	int	CO percentage	

Visualizations
Average

3. The problems that you faced (10 points)

The first problem we encountered was changing our entire project plan and coming up with a new project that would allow more connections and conclusions to be made within the data given.

The second problem we encountered was merging our work on VSCode so all three of us could make edits and changes. After several attempts, we finally figured it out and set up a system that forced us to all stop and push and pull our code together.

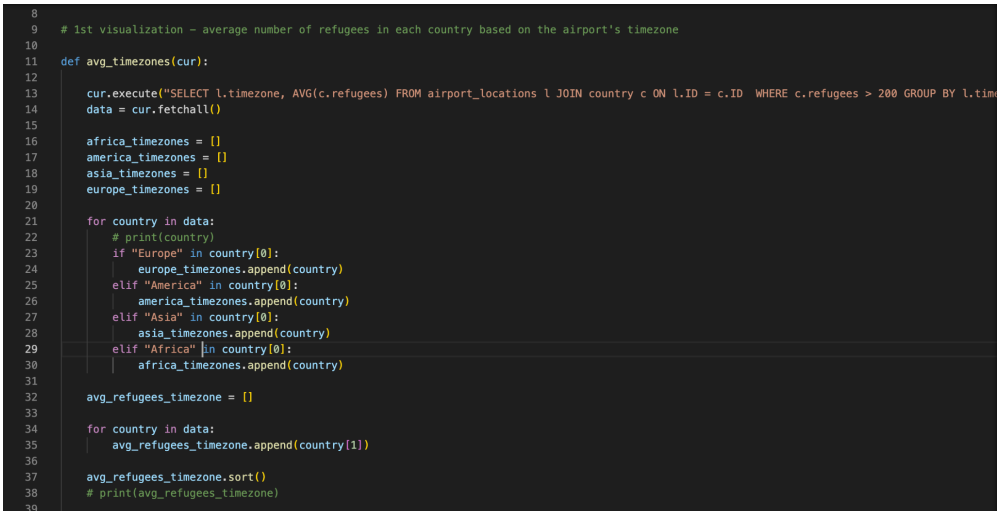
The third problem we encountered was with the foreign keys. Everytime we added the foreign keys when creating each table, we kept getting errors and started to overcomplicate our code.

The next problem we faced is when inputting the cities and countries into airport.py, there were multiple airports in countries that had multiple cities of the same name. Therefore, we had to go through the output data and see which cities were duplicated and what we already had in our databases to ensure there were no repeats again. As a result, we had to hardcode some of our cities to ensure they only appear once.

The next problem we encountered was when we created our databases, we couldn't get all 100 rows to show up, only the first 25. We ended up forgetting to add SELECT and VSCode kept giving us errors for the cities and countries we had inputted, and every time we ran it the ones that were problematic would change. It took a very long time to debug this, but we ended up figuring it out.

4. The calculations from the data in the database (i.e. a screenshot) (10 points)

1)



```
8
9   # 1st visualization - average number of refugees in each country based on the airport's timezone
10
11  def avg_timezones(cur):
12
13      cur.execute("SELECT l.timezone, AVG(c.refugees) FROM airport_locations l JOIN country c ON l.ID = c.ID  WHERE c.refugees > 200 GROUP BY l.tim
14      data = cur.fetchall()
15
16      africa_timezones = []
17      america_timezones = []
18      asia_timezones = []
19      europe_timezones = []
20
21      for country in data:
22          # print(country)
23          if "Europe" in country[0]:
24              europe_timezones.append(country)
25          elif "America" in country[0]:
26              america_timezones.append(country)
27          elif "Asia" in country[0]:
28              asia_timezones.append(country)
29          elif "Africa" in country[0]:
30              africa_timezones.append(country)
31
32      avg_refugees_timezone = []
33
34      for country in data:
35          avg_refugees_timezone.append(country[1])
36
37      avg_refugees_timezone.sort()
38      # print(avg_refugees_timezone)
39
```

```
    x = avg_refugees_timezone
    y = ["Africa/Bujumbura", "Africa/Conakry", "Africa/Johannesburg", "Africa/Khartoum", "Africa/Maputo", "Africa/Nairobi", "America/Chicago", "A
    plt.barh(y, x, color = "pink")
    plt.xlabel("Average Number of Refugees per Time Zone")
    plt.ylabel("Time Zone of Country")
    plt.title('Average # of Refugees per Country vs. Time Zone of Country)')
    plt.show()

    highest_ref = max(avg_refugees_timezone)
    lowest_ref = min(avg_refugees_timezone)

    total_ref= 0
    for num  in avg_refugees_timezone:
        total_ref += num
    total_ref = total_ref/len(avg_refugees_timezone)

    highest_tz= ""
    lowest_tz = ""

    for country in data:
        if country[1] == highest_ref:
            highest_tz = country[0]
        if country[1] == lowest_ref:
            lowest_tz = country[0]

    with open('avg_timezones.txt', 'w') as f:
        f.write("In the " + highest_tz + " timezone, the average number of refugees was " + str(round(highest_ref)) + ". This is the highest avera
        f.write("In the " + lowest_tz + " timezone, the average number of refugees was " + str(round(lowest_ref)) + ". This is the lowest average
        f.write("Across all timezones, the average number of refugees is " + str(round(total_ref)) + ".")

    f.close()
```

avg_timezones.txt

```
In the Europe/Vilnius timezone, the average number of refugees was 6387. This is the highest average number of refugees in a timezone.
In the Africa/Conakry timezone, the average number of refugees was 226. This is the lowest average number of refugees in a timezone.
Across all timezones, the average number of refugees is 1318.
```

2)

```
# 2nd calculation - number of tourists more than 200 vs elevation (scatterplot)
#reasoning: tourists wanna go to places with higher elevations/mountains maybe??

def avg_tourists(cur):

    elevations = []
    num_tourists = []

    cur.execute("SELECT c.tourists , l.elevation, l.city FROM country c JOIN airport_locations l ON c.ID = l.ID WHERE l.elevation > 200  ORDER BY
    # cur.execute("SELECT MAX(c.tourists) , l.elevation, l.city FROM country c JOIN airport_locations l ON c.ID = l.ID WHERE l.elevation > 200  OR
    data = cur.fetchall()

    for country in data:
        num_tourists.append(country[0])
        elevations.append(country[1])

    # print(elevations)
    # print(num_tourists)

    plt.scatter(num_tourists, elevations)
    plt.xlabel("Number of Tourists (more than 200) per Country")
    plt.ylabel("Country Elevation (ft)")
    plt.title('Number of Tourists per Country vs. Country Elevation (ft)')
    plt.show()

    cur.execute("SELECT AVG(c.tourists) , l.elevation, l.city FROM country c JOIN airport_locations l ON c.ID = l.ID WHERE l.elevation > 9000  GR
    over_9k = cur.fetchall()

    with open('average_tourists_elevation.txt', 'w') as a:
        a.write("The average number of tourists where the elevation is greater than 9,000 feet is " + str(over_9k)+".")
```

average_tourists_elevation.txt

```
In countries where the elevation is high, the average number of tourists is 4221.
In countries where the elevation is in the middle, the average number of tourists is 14694.
In countries where the elevation is low, the average number of tourists is 12210.
```

3)

```python
def avg_AQI(cur):

    cur.execute("SELECT q.AQI, c.name, c.population FROM air_quality q JOIN country c ON q.ID = c.ID")
    data = cur.fetchall()

    good = []
    moderate = []
    unhealthy_s = []
    unhealthy = []
    v_unhealthy = []
    no_data = []

    for i in range(len(data)):
        # print(data[i][0])
        # print(data[i][0])
        if data[i][0] == -1:
            no_data.append(data[i])
            # print(no_data)
        elif data[i][0] > 0 and data[i][0] <= 50:
            good.append(data[i])
        elif data[i][0] >= 51 and data[i][0] <= 100:
            moderate.append(data[i])
        elif data[i][0] >= 101 and data[i][0] <= 150:
            unhealthy_s.append(data[i])
        elif data[i][0] >= 151 and data[i][0] <= 200:
            unhealthy.append(data[i])
        elif data[i][0] >= 201 and data[i][0] <= 250:
            v_unhealthy.append(data[i])

    good_total = 0
    mod_total = 0
    unhealthy_s_total = 0
    unhealthy_total = 0
```

```python

    # edit the data, make it rounded

    for each_country in good:
        # print(each_country[2] * 1000)
        good_total += each_country[2] * 1000
    good_avg = round(good_total/len(good))

    for each_country in moderate:
        mod_total += each_country[2] * 1000
    mod_avg = round(mod_total/len(good))

    for each_country in unhealthy_s:
        unhealthy_s_total += each_country[2] * 1000
    unhealthy_s_avg = round(unhealthy_s_total/len(good))

    for each_country in unhealthy:
        unhealthy_total += each_country[2] * 1000
    unhealthy_total_avg = round(unhealthy_total/len(good))

    x = ["Good", "Moderate", "Unhealthy for Some", "Unhealthy for All"]
    y = [good_avg, mod_avg, unhealthy_s_avg, unhealthy_total_avg]

    plt.bar(x, y, color = "red")
    plt.xlabel('AQI (Air Quality Index) Category')
    plt.ylabel('Average Population Size of Country (in 100 millions)')
    plt.title('AQI vs. Average Country Population Size')
    plt.show()

```

avg_pop_AQI.txt

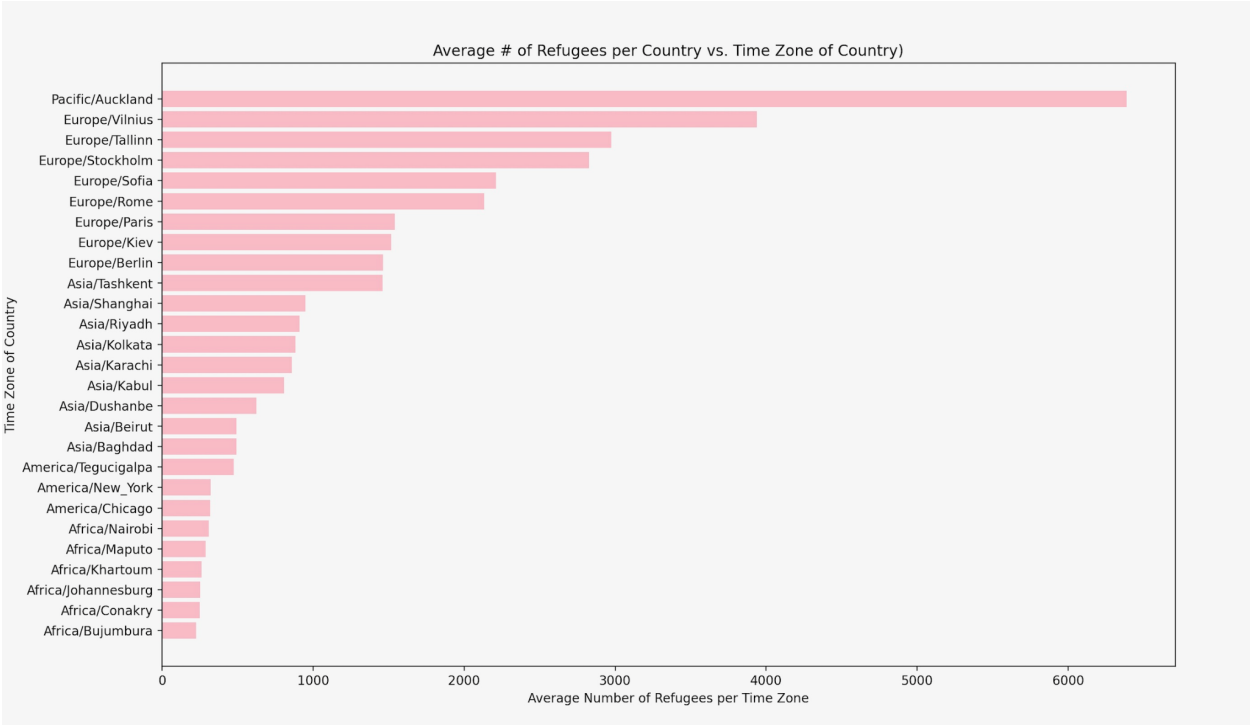
```
The average population of a country with a 'Good' Air Quality Index is 50192120.
The average population of a country with a 'Moderate' Air Quality Index is 74414720.
The average population of a country with a 'Unhealthy (for some)' Air Quality Index is 18739760.
The average population of a country with a 'Unhealhty (for all)' Air Quality Index is 34201600.

```

```
213
214 def ec_weather(cur):
215
216     humidity = []
217     species = []
218
219     cur.execute("SELECT w.city, w.humidity, c.threatened_species FROM weather w JOIN country c ON w.ID = c.ID WHERE w.humidity!= -1 AND w.humidity
220     data = cur.fetchall()
221
222     for each in data:
223         humidity.append(each[1])
224         species.append(each[2])
225     #     if each[1] == -1:
226     #         continue
227     # print
228     # species.sort()
229     plt.barh(species, humidity, color = "plum")
230     plt.ylabel("Number of Threatened Species")
231     plt.xlabel("Humidity in Country (in countries where humidity is less than 50 F) (deg. F)")
232     plt.title("Number of Threatened Species vs. Humidity (deg. F)")
233     plt.show()
```

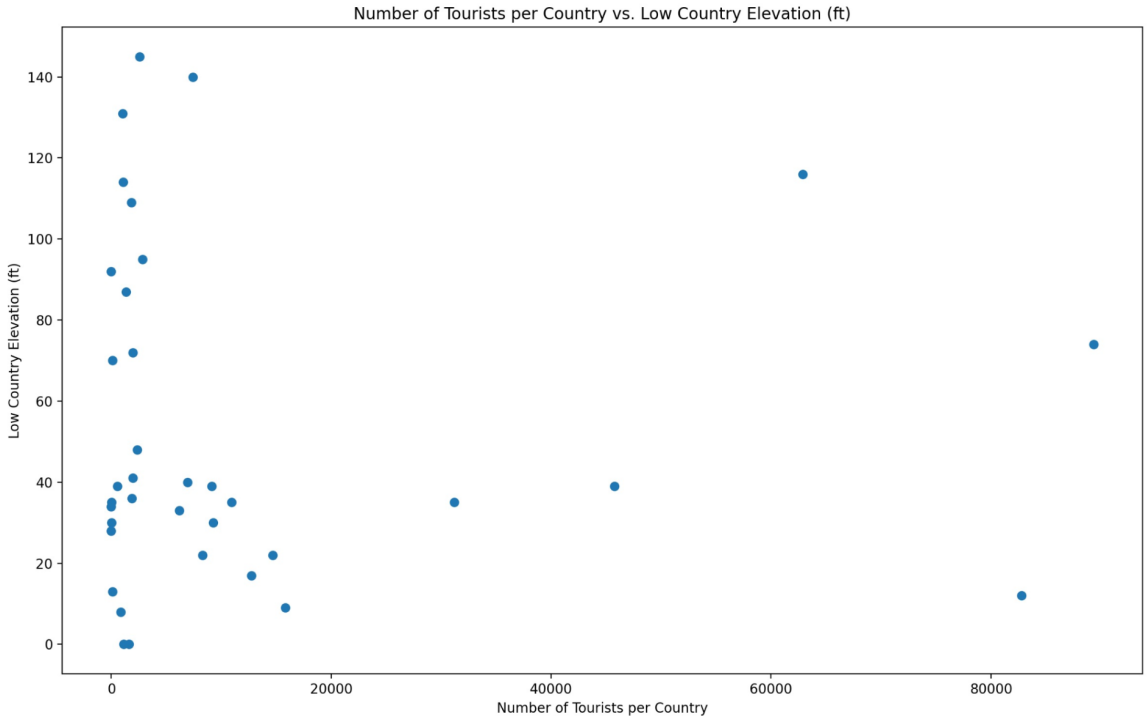
```
234
235     max_species = max(species)
236     min_species = species[4]
237     max_humidity = 0
238     min_humidity = 0
239
240     for i in range(len(humidity)):
241         if species[i] == max_species:
242             max_humidity = humidity[i]
243         elif species[i] == min_species:
244             min_humidity = humidity[i]
245
246     with open('weather_ec_calc.txt', 'w') as f:
247         f.write("The largest number of threatened species, " + str(max_species) + ", is located in country with " + str(max_humidity) + " c
248         f.write("The smallest number of threatened species, " + str(min_species) + ", is located in country with " + str(min_humidity) + "
249     f.close()
250
```

5. The visualization that you created (i.e. screenshot or image file) (10 points)

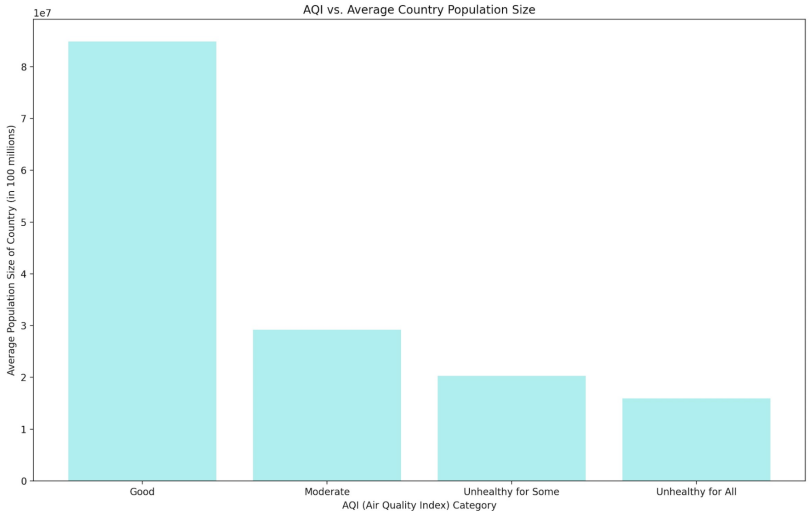
1)



2)

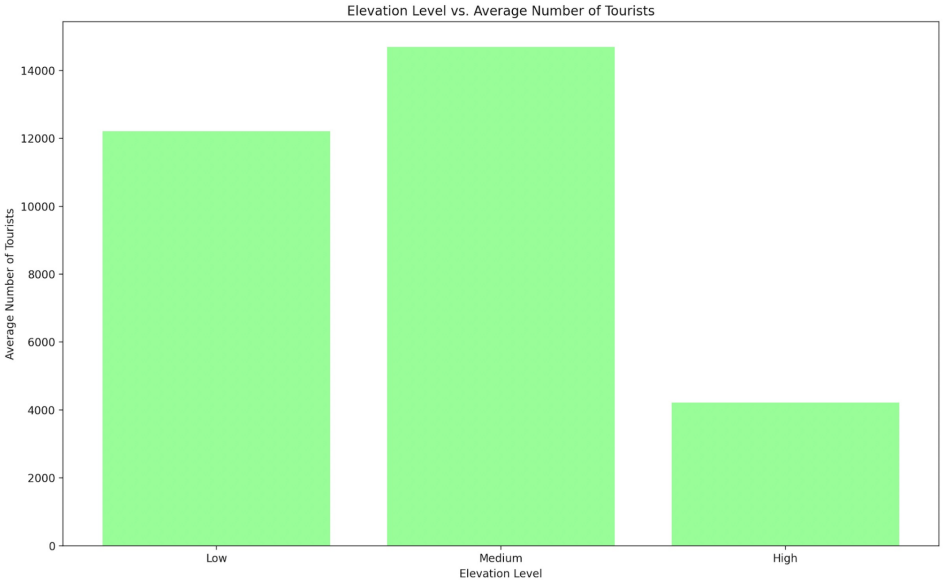


3)

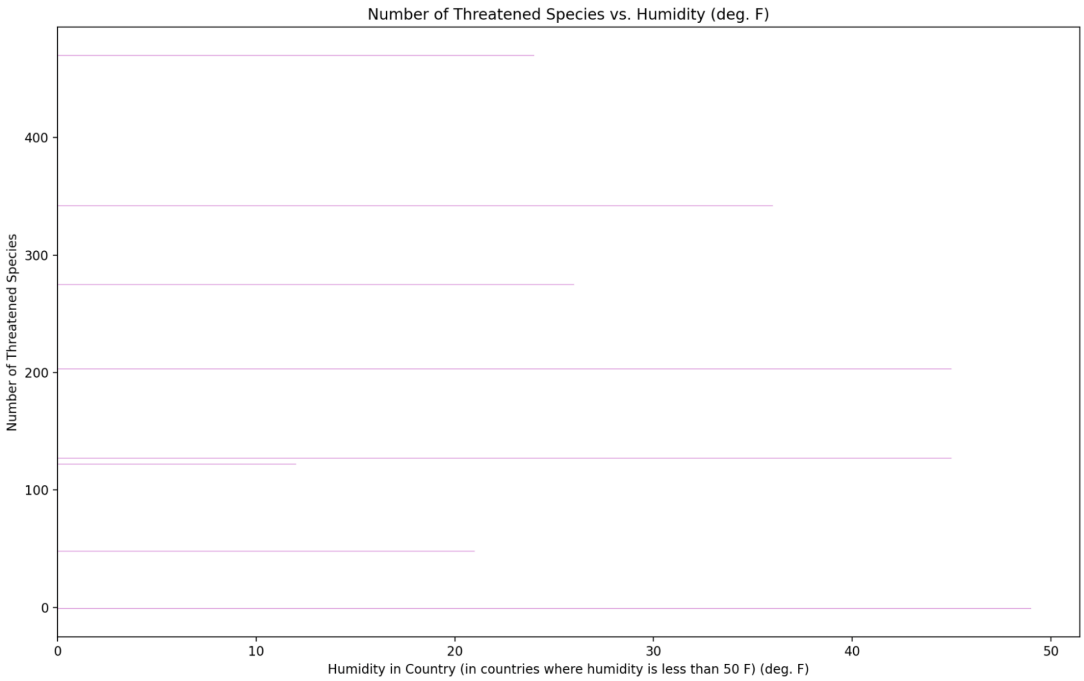


4) AQI quality categories: 0-50 = good, 51-100 = moderate, 101-150 = Unhealthy for some, 151-200 = Unhealthy, 201-300 = Very Unhealthy

4) EXTRA CREDIT VISUALIZATIONS




```
Elevation categories: low elevation = 0-199, medium elevation: 200-999, high
elevation: 1,000+
```



6. Instructions for running your code (10 points)

First, run air_quality.py 4 times in order to input the air_quality table to the database
- Joined with country.py to create average_tourists_elevation.txt

Next, run airport.py 4 times in order to input the airport table to the database

Next, run locations.py 4 times in order to create the airport location table

Next, run country.py 4 times in order to input the country table to the database
- Joined with air_quality.py to create average_tourists_elevation.txt

Next, run weather.py 4 times in to input the weather tablet to the database

Finally, run visualizations.py to see the graphs and visualizations we created

7. Documentation for each function that you wrote. This includes describing the input and output for each function (20 points)

AIR_QUALITY.PY

1) **def get_data_air_quality(country):**

INPUT: (country)
OUTPUT: A dictionary

PURPOSE: This function takes in data from the air_quality API URL and country and converts it into JSON, returning as a dictionary.

2) **def create_air_quality_table(cities, cur, conn):**

INPUT: List of cities, cur, conn
OUTPUT: Creates air_quality table in airports.db

PURPOSE: To create a table

3) **def main():**

INPUT: None
OUTPUT: None

PURPOSE: To run all the functions

AIRPORT.PY

1) **def get_airport_data(city):**

INPUT: (city)
OUTPUT: A dictionary

PURPOSE: This function takes in data from the airport API URL and city and converts it into JSON, returning as a dictionary.

2) **def create_airport_table(cities, cur, conn):**

INPUT: List of cities, cur, conn
OUTPUT: Creates airport table in airports.db

PURPOSE: To create a table

3) **def main():**

INPUT: None
OUTPUT: None

PURPOSE: To run all the functions

COUNTRY.PY

1) **def getdata(country):**

INPUT: (country)

OUTPUT: A dictionary

PURPOSE: This function takes in data from the country API URL and converts it into JSON, returning as a dictionary.

2) **def createtable1(countries1, cur, conn):**

INPUT: List of countries, cur, conn
OUTPUT: Creates country table in airports.db

PURPOSE: To create a table

3) **def main():**

INPUT: None
OUTPUT: None

PURPOSE: To run all the functions

LOCATION.PY

1) **def get_location_data(city):**

INPUT: (city)
OUTPUT: A dictionary

PURPOSE: This function takes in data from the air_quality API URL and country and converts it into JSON, returning as a dictionary.

2) **def create_airport_loc_table(cities, cur, conn):**

INPUT: List of cities, cur, conn
OUTPUT: Creates airport location table in airports.db

PURPOSE: To create a table

3) **def main():**

INPUT: None
OUTPUT: None

PURPOSE: To run all the functions

VISUALIZATIONS.PY

1) **def avg_timezones(cur):**

INPUT: (cur)
OUTPUT: A visualization

PURPOSE: To calculate the average number of refugees in each country based on the airport's timezone

2) **def avg_tourists(cur):**

INPUT: (cur)
OUTPUT: A visualization

PURPOSE: To calculate the average number of tourists (more than 200 per country) vs the average elevation of that country

3) **def avg_AQI(cur):**

INPUT: (cur)
OUTPUT: A visualization

PURPOSE: To calculate the average population of countries grouped by the AQI category

4) **def ec_weather(cur):**

INPUT: (cur)

OUTPUT: A visualization

PURPOSE: To calculate the average number of threatened species and the average humidity in a country

5) **def main():**

INPUT: None
OUTPUT: None

PURPOSE: To run all the functions

WEATHER.PY

1) **def get_data_weather(city):**

INPUT: (city)
OUTPUT: A dictionary

PURPOSE: This function takes in data from the country API URL and converts it into JSON, returning as a dictionary.

2) **def create_weather_table(cities, cur, conn):**

INPUT: List of cities, cur, conn
OUTPUT: Creates weather table in airports.db

PURPOSE: To create a table

3) **def main():**

INPUT: None
OUTPUT: None

PURPOSE: To run all the functions

8. You must also clearly document all resources you used. The documentation should be of the following form (20 points)

	Date	Issue Description	Location of Resource	Result (did it solve the issue?)
Airport API	04/16/2023	Changed APIs last minute and needed to find new one	API Ninjas https://api-ninjas.com/	Yes
Weather API	04/16/2023	Changed APIs last minute and needed to find new one	API Ninjas https://api-ninjas.com/	Yes
Country API	04/16/2023	Changed APIs last minute and needed to find new one	API Ninjas https://api-ninjas.com/	Yes
Air Quality API	04/16/2023	Changed APIs last minute and needed to find new one	API Ninjas https://api-ninjas.com/	Yes
Google Sheets Databases	04/16/2023	We needed a way to normalize our data and lay out how we are going to utilize the databases	Google Sheets Database - Entit…	Yes
Matplotlib	04/16/2023	Needed a way to create the visuals	https://matplotlib.org/	Yes

Changes made after grading session:

- All statements that included “drop table” are deleted
- We had duplicate string data in all tables in the form of the name of the cities; to change this, we stored the city name column in only the airport location table and used a city_ID foreign key in other tables
- Created a new foreign key called country_ID to serve as a foreign key for the name of the countries in other tables in the database
- Ran weather.py file one more time to get all 100 items in there instead of 75