Final Project Report:

Team: KAPPA (Giselle Ciulla, Grace Coleman, Zita Jameson)
Topic: Covid-19
GitHub: https://github.com/zjameson110600/Final-Project

Goals:

Initial:

- Utilize three various music APIs to calculate the popularity of each track on the different platforms.

Final:

- Utilize three different COVID-19 APIs
  - One containing data on cases and deaths for each country
  - One containing data on population, continent, and latitude/longitude for each country
  - One containing data on tests per country
- Calculations:
  - We calculated the death rate per country
  - We calculated the infection rate per country
  - We calculated the testing rate per country

Problems Faced:

- The music APIs proved difficult to work with
  - After receiving feedback from the teaching team we realized …
    - Our initial project plan did not have anything new being calculated
- It was difficult to find APIs that covered different parts of the Covid-19 pandemic
  - We ended up extracting different types of data from APIs that had some similar information and some different information.
  - We figured working with such "new" information could be a little difficult, but we found solutions to those difficulties.
- Finding common key between different APIs was difficult
- Adding unique data into the database was not working at the beginning
- Map visualization was more difficult to work with with our dataset.

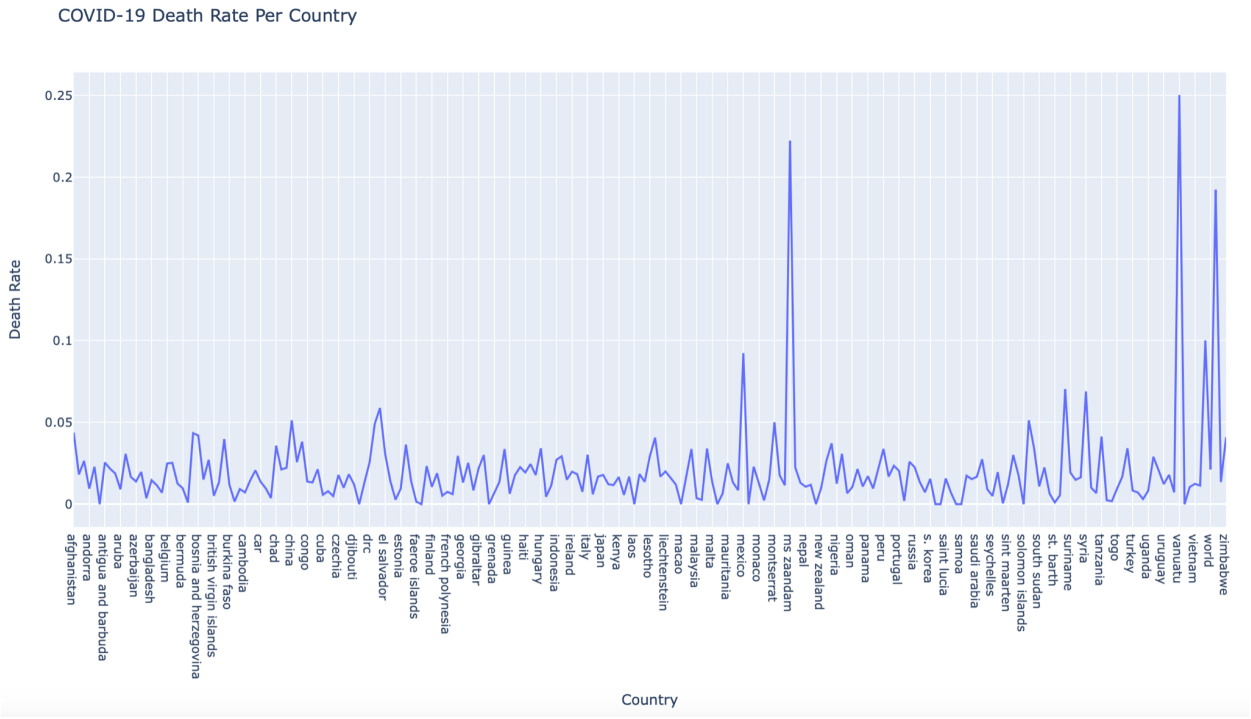
Calculations:

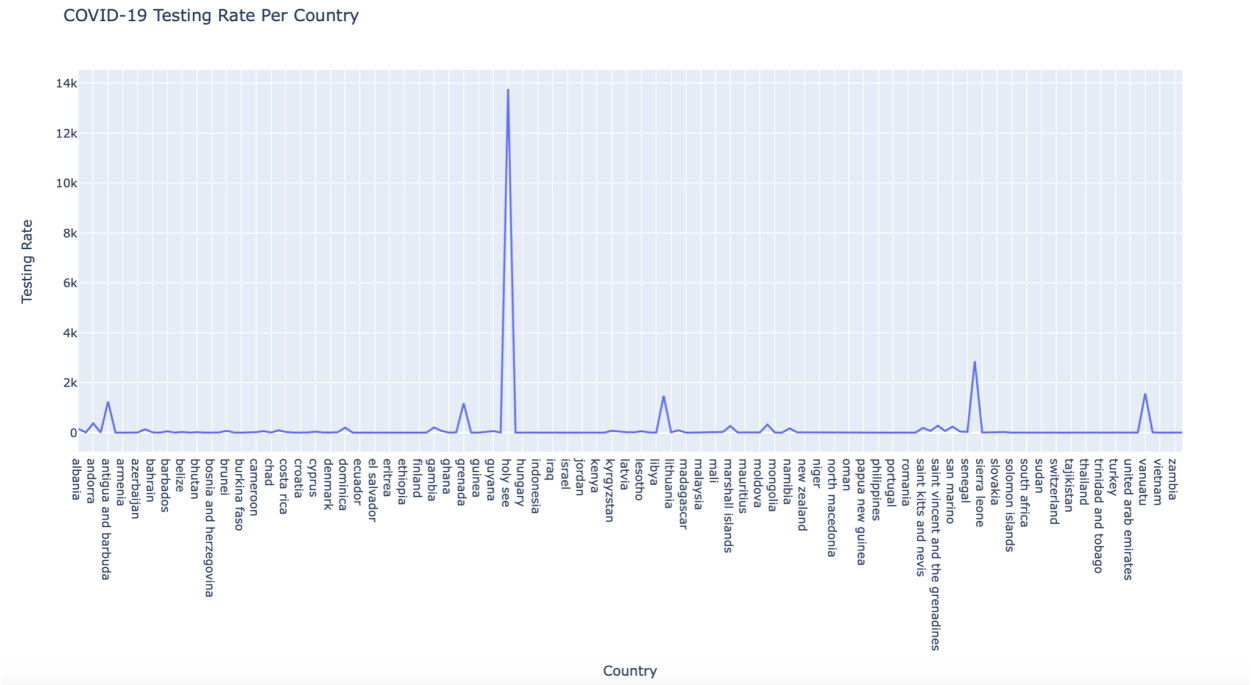
https://docs.google.com/spreadsheets/d/1yUEocChO_jCikEL1caBesCdU5Mg9wOkVLiDQwm9R94Y/edit?usp=sharing [Calculating Death Rate]

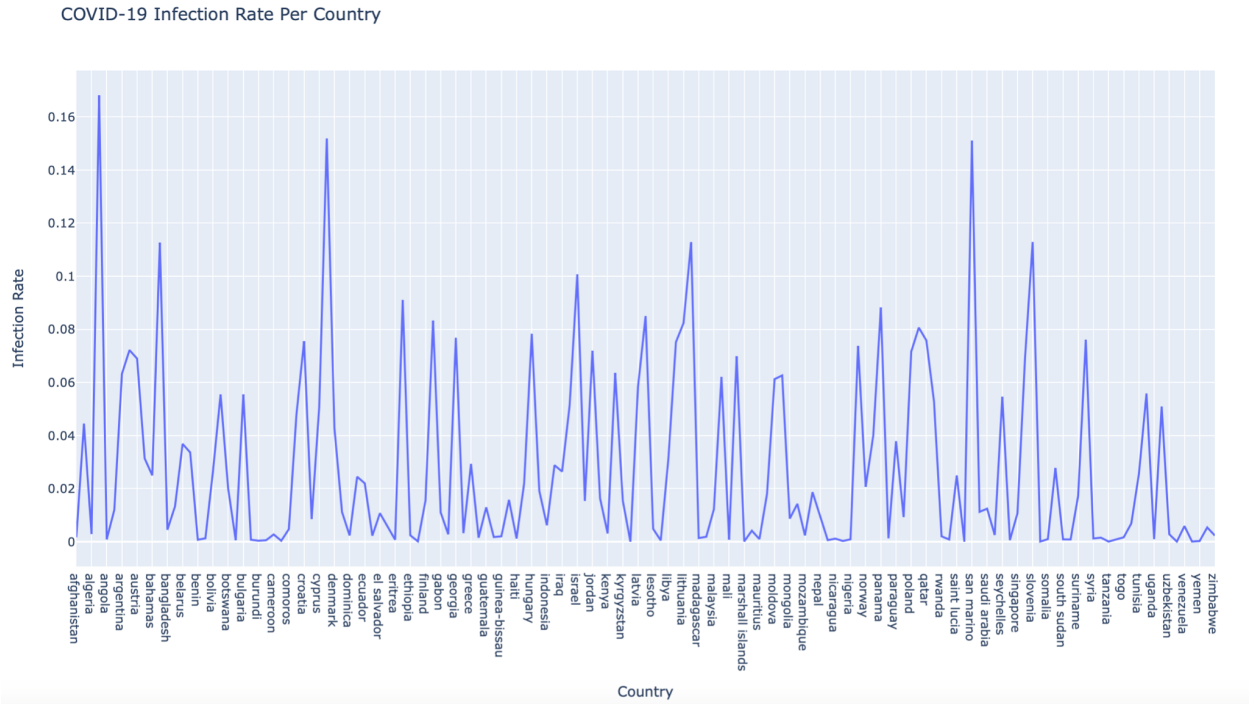
https://docs.google.com/spreadsheets/d/1Q-DA0T53LQcgdyIQvBf2rlGqV9U_VvP7YQsP54ZsfsM/edit?usp=sharing [Calculating Infection Rate]

https://docs.google.com/spreadsheets/d/1wAmMK83Bd0VRjsb6eKz1oHqSaJ2zyHNzaKdS3fZQj6k/edit?usp=sharing [Calculating Testing Rate]

Visualizations:




COVID-19 Testing Rate Per Country
Testing Rate
14k
12k
10k
8k
6k
4k
2k
0
albania
andorra
antigua and barbuda
armenia
azerbaijan
bahrain
barbados
belize
bhutan
bosnia and herzegovina
brunei
burkina faso
cameroon
chad
costa rica
croatia
cyprus
denmark
dominica
ecuador
el salvador
eritrea
ethiopia
finland
gambia
ghana
grenada
guinea
guyana
holy see
hungary
indonesia
iraq
israel
jordan
kenya
kyrgyzstan
latvia
lesotho
libya
lithuania
madagascar
malaysia
mali
marshall islands
mauritius
moldova
mongolia
namibia
new zealand
niger
north macedonia
oman
papua new guinea
philippines
portugal
romania
saint kitts and nevis
saint vincent and the grenadines
san marino
senegal
sierra leone
slovakia
solomon islands
south africa
sudan
switzerland
tajikistan
thailand
trinidad and tobago
turkey
united arab emirates
vanuatu
vietnam
zambia
Country


COVID-19 Infection Rate Per Country
Infection Rate
0.16
0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
afghanistan
algeria
angola
argentina
austria
bahamas
bangladesh
belarus
benin
bolivia
botswana
bulgaria
burundi
cameroon
comoros
croatia
cyprus
denmark
dominica
ecuador
el salvador
eritrea
ethiopia
finland
gabon
georgia
greece
guatemala
guinea-bissau
haiti
hungary
indonesia
iraq
israel
jordan
kenya
kyrgyzstan
latvia
lesotho
libya
lithuania
madagascar
malaysia
mali
marshall islands
mauritius
moldova
mongolia
mozambique
nepal
nicaragua
nigeria
norway
panama
paraguay
poland
qatar
rwanda
saint lucia
san marino
saudi arabia
seychelles
singapore
slovenia
somalia
south sudan
suriname
syria
tanzania
togo
tunisia
uganda
uzbekistan
venezuela
yemen
zimbabwe
Country

Code Instructions:

1. Make sure the only files in the project folder end in ".py" or are the README.
    a. No database, image, or csv files in folder
2. Visual Studio Code, SQLite Database Browser, Plotly and Matplotlib should be fully installed on your computer. The API does not need to be pulled up or accessed before running the code.
3. Open up the zipped file. The file you should open first is "FinalProject.py".
    a. Run this file code four times.
    b. It will create a database called 'FinalCovids.db' and will create the tables "Countries", "Populations", and "Tested" within the database.
    c. You will now see the 'FinalCovids.db' file in the project folder.
4. With SQLite DataBase Browser installed, the tables within the database will be viewable.
    a. At the end, there should be 100 rows in each table
5. Allow the code time to pull up the visualizations.
    a. Ignore pop-up visualizations until you have run it four times in order to get the most accurate results.
6. Code will create CSV files that can be opened in Google Sheets or Excel that shows our three calculations (these are also linked above).

Code Documentation:

First Function:

```
def cases_deaths(cur, conn):

   # gets data from api and puts into table

   url = 'https://coronavirus-19-api.herokuapp.com/countries'

   request = requests.get(url)

   result = request.json()

   country_count = 0

   cur.execute("CREATE TABLE IF NOT EXISTS Countries (country TEXT PRIMARY KEY, cases
INTEGER, deaths INTEGER)")

   for country in result:

       countries = (country['country']).lower()

       cases = country['cases']
```

```
        deaths = country['deaths']

        if country_count == 25:

            break

        if cur.execute("SELECT country FROM Countries WHERE country = ?",
(countries,)).fetchone() == None:

            cur.execute("INSERT OR REPLACE INTO Countries (country,cases,deaths) VALUES
(?,?,?)", (countries, cases, deaths))

            country_count += 1

            continue

    conn.commit()
```

Output: Returns nothing, function adds country, cases, and deaths to database.

Second Function:

```
def population_location(cur, conn):

    # gets data from api and puts into table

    url = 'https://covid-api.mmediagroup.fr/v1/cases'

    request = requests.get(url)

    result = request.json()

    continents = {}

    pop_count = 0

    count = 0

    cur.execute("CREATE TABLE IF NOT EXISTS Populations (country TEXT PRIMARY KEY,
country_id INTEGER, continent TEXT, population INTEGER, latitude FLOAT, longitude
FLOAT)")

    conn.commit()
```

```
   for country in result:

       countries = country

       try:

           continue

       try:

           lat = float(result[countries]["All"]["lat"])

           long = float(result[countries]["All"]["long"])

       except:

           continue

       if pop_count == 25:

           break

       if cur.execute("SELECT population FROM Populations WHERE population = ?",
(population,)).fetchone() == None:

           cur.execute("INSERT OR REPLACE INTO Populations (country, country_id,
continent, population, latitude, longitude) VALUES (?,?,?,?,?,?)",(countries.lower(),
count, continent, population, lat, long))

           pop_count += 1

           count += 1

           continue

   conn.commit()
```

Output: Returns nothing, function adds country, country ID, continent, population, latitude, and longitude to database.

Third Function:

```
def testing(cur, conn):

   # returns the amount of people tested in each country

   url = 'https://api.quarantine.country/api/v1/summary/latest'

   request = requests.get(url)

   result = request.json()

   continent_count = 0

   cur.execute("CREATE TABLE IF NOT EXISTS Tested (id INTEGER PRIMARY KEY,tested
INTEGER)")

   conn.commit()

   for x in result:

       test = result['data']['regions']

       for x in test:

           continents = cur.execute("SELECT continent FROM Populations").fetchall()

           conn.commit()

           countries = x.lower()

           tested = test[x]['tested']

           if continent_count == 25:

               break

           if cur.execute("SELECT country FROM Populations WHERE country = ?",
(x,)).fetchone() == None:

               country = cur.execute("SELECT country_id FROM Populations WHERE country
= ?", (x,)).fetchone()

               cur.execute("INSERT OR REPLACE INTO Tested (id,tested) VALUES (?,?)",
(country,tested))
```

```
                conn.commit()

                continent_count += 1

            continue

    conn.commit()
```

Output: Returns nothing, function adds country ID and tested into database.

Calculation 1:

```
def calculate_countries(cur, conn, filepath):

    #calculates death rate and inserts data into csv

    source_dir = os.path.dirname(os.path.abspath(__file__))

    file_path = os.path.join(source_dir, filepath)



    data = cur.execute("SELECT country,cases,deaths FROM Countries ORDER BY country
ASC").fetchall()

    conn.commit()



    with open(filepath, 'w') as f:

        f = csv.writer(f, delimiter = ',')

        f.writerow(['Country', 'Cases', 'Deaths', 'Death Rate'])

        for x in data:

            try:

                death_rate = (x[2]/x[1])

            except:

                death_rate = 0
```

```
            all_data = (x[0], x[1], x[2], death_rate)

            f.writerow(all_data)
```

Output: Returns nothing, adds country, cases, deaths, and death rate into CSV file.

Calculation 2:

```
def calculate_populations(cur, conn, filepath):

    #calculates infection rate and inserts data into csv

    source_dir = os.path.dirname(os.path.abspath(__file__))

    file_path = os.path.join(source_dir, filepath)

     distinct = cur.execute("SELECT Countries.cases, Populations.country,
Populations.population FROM Countries INNER JOIN Populations ON Countries.country =
Populations.country ORDER BY Populations.country ASC").fetchall()

    conn.commit()

    #cases, country, pop

    with open(filepath, 'w') as f:

        f = csv.writer(f, delimiter = ',')

        f.writerow(['Country', 'Population', 'Cases', 'Infection Rate'])

        for x in distinct:

            infection_rate = x[0]/x[2]

            all_data = (x[1], x[2], x[0], infection_rate)

            f.writerow(all_data)
```

Output: Returns nothing, adds country, population, cases, and infection rate to CSV file.

Calculation 3:

```
def calculate_testing(cur, conn, filepath):
```

```
   #calculates testing rate per country

   source_dir = os.path.dirname(os.path.abspath(__file__))

   file_path = os.path.join(source_dir, filepath)



   testing = cur.execute("SELECT Tested.tested, Tested.id, Populations.country,
Populations.population FROM Tested INNER JOIN Populations ON Tested.id =
Populations.country_id").fetchall()

   conn.commit()

   #tested, id, country, population

   with open(filepath, "w") as f:

       f = csv.writer(f, delimiter= ",")

       f.writerow(['Id', 'Country', 'Tested', 'Population','Testing Rate'])

       for x in testing:

           try:

               testing_rate = x[0]/x[3]

           except:

               testing_rate = 0

           all_data= (x[1], x[2], x[0], x[3], testing_rate)

           f.writerow(all_data)
```

Output: Returns nothing, adds country ID, country, tested, population, and testing rate to CSV file.

VISUALIZATION FUNCTIONS:

```
def countries_plot():

   df = pd.read_csv('calculation_countriess.csv')
```

```
    fig = px.line(df, x = 'Country', y = 'Death Rate', title = 'COVID-19 Death Rate Per Country')

    fig.update_xaxes(categoryorder='category ascending')

    fig.show()









def populations_plot():

    df = pd.read_csv('calculation_populationss.csv')

    fig = px.line(df, x = 'Country', y = 'Infection Rate', title = 'COVID-19 Infection Rate Per Country')

    fig.update_xaxes(categoryorder='category ascending')

    fig.show()



def testing_plot():

    df = pd.read_csv('calculation_testings.csv')

    fig = px.line(df, x = 'Country', y = 'Testing Rate', title = 'COVID-19 Testing Rate Per Country')

    fig.update_xaxes(categoryorder='category ascending')

    fig.show()
```

Output: Returns nothing, creates line graph for each calculation per country.

Documentation of Resources:

Date	Issue Description	Location of Resource	Result

April 14th 2021	Music APIs are too hard to work with and our original project plan was poor (we didn't have an idea of new things to calculate).	Found new APIs to work with and redesigned our project plan. Gathered new APIs from GitHub: ● https://github.com/javieravilles/covidAPI ● https://github.com/M-Media-Group/Covid-19-API ● https://github.com/Yatko/Coronavirus-API	This solved our issue! These new APIs proved easy to work with and our new project plan was precise.
April 17th 2021	No common key between two databases.	Went to office hours to figure out the shared key (country id between populations and testing databases).	Figured out a shared key that worked for our project.
April 22nd, 2021	Adding unique data into the database was not working.	Went to office hours to figure out the problem with adding data into the database.	GSI's were able to help us to solve the problem with a few small fixes in our code.
April 22nd, 2021	We wanted to make a map visualization, and it was more complicated and difficult than anticipated.	Went to matplotlib's website and looked through visualization examples.	Decided to make a line graph instead of map, and successfully made a different graph for each API.