Info 6205 Final Project Report
Virus Transfer Simulation

Team Members:
Akash Rakesh Sinha – 001344425
Ripan Anadi Halder - 001442993

I) Introduction

Since the onset of last year, the disease COVID-19 has spread all over the world by infecting nearly a billion of people and has killed millions of them. If not avoided, it surely could have been prevented from spreading all over the world. Airports and flights connections made it possible to spread very rapidly. Government intervention, quarantine measures and lockdown were undertaken in each country to mitigate the spread. So, we planned to design our final project to understand these situations and analyze the spread of a similar virus if it arises in future. Also, we analyzed current COVID-19 data and displayed the results.

Our project is built in Python by using Jupyter Notebook. This project fetches data from public repositories and databases, processes these data and maps them for an SIR Model (Susceptible, Infected & Recovered). The project collects airport and flight details data of all the airports in USA and the air connectivity between them. We projected a viral disease whose vaccination is not present similar to COVID 19. The infection starts in Boston on Day 0 and since the virus spreads by people coming in contact with each other, it spreads all across US through domestic flights connecting all the airports with each other. Considering that, the government intervenes after a particular day and takes few more days before a lockdown and quarantine start. In such a scenario since flights are connected, the virus spreads all across US. We have considered various other factors like city population, contact rate and recovery rate. We fix these values to simulate our results. So, the spread of the disease is in two modes in our assumptions, within the city using SIR model and air traffic spread across different cities.

II) Aim of the Project:

The project aims to simulate various factors that affect the mass spreading of a viral disease. It also focuses on fetching, analyzing, and determining that possible steps that can be taken in order curtail the spread of a disease if there is an outbreak in future. To analyze visually the effects of human interventions like quarantining, flight connections and recovery rate via map and charts. Also, analyze the current COVID-19 situation using the data from John-Hopkins COVID19 dataset.

III) Project Details:

- **Data Processing and Data Cleaning**

USA Cities Population:

Implemented Beautiful Soup for parsing and extracting data about the human population for various cities in United States of America from the website (link: http://worldpopulationreview.com/us-cities/)

A function named get_soup() is defined that takes the above link as an argument. In this function the Python library Beautiful Soap is used for parsing the html data. This parsed HTML data contains the city wise population of USA stored in various HTML tags and CSS classes. This city wise data is then extracted by using the appropriate "class" attribute present in the parsed data and is stored as a table. The table data is parsed as a string which is then populated in a .csv file.

Sample output of the parsed table data:

	Rank	Name	State	2020 Pop	2010 Census	Change	Density (km²)	Area (km²)
0	1	New York City	New York	8323340	8190360	1.62%	10699	778
1	2	Los Angeles	California	4015940	3795370	5.81%	3306	1215
2	3	Chicago	Illinois	2694240	2697530	-0.12%	4575	589
3	4	Houston	Texas	2340890	2098450	11.55%	1412	1658
4	5	Phoenix	Arizona	1703080	1449300	17.51%	1270	1341
...	...	...	...	...	...	...	...	...
195	196	Miramar	Florida	143219	122299	17.11%	1917	75
196	197	Bridgeport	Connecticut	143010	144858	-1.28%	3437	42
197	198	Olathe	Kansas	142841	126249	13.14%	895	160
198	199	Denton	Texas	142173	117058	21.46%	570	249
199	200	Surprise	Arizona	142049	117674	20.71%	508	280

USA Airports Information:

This section handles the extraction, parsing and cleaning of data related to all the airports and their flight connections within the United States of America.

There are numerous airports in USA which handles the incoming and outgoing traffic of the flights from different airlines through out the year. Each airport has a unique airport code which is extracted from the website (link: https://www.flightconnections.com/airport-codes). This data is extracted and parsed using Beautiful soup python library. The airport codes are present as a list in the "ul" HTML tag which has the class attribute as "airport-list".

The above link provides the detailed information about all the airports present around the globe. Hence, this list is iterated to get the list of all the distinct airport codes. While navigating on the above webpage we observed that each and every airport code is a hyperlink which clicked upon redirects to another webpage which provides the details of the airport that have an incoming flight into the concerned airport. As our main aim is to track the spread of the disease we modified the hyperlink to extract the data related to the outgoing flights from the concerned airport.

This data now contains all the columns/fields that are related to our analysis, but since we are analyzing the spread of the disease within the United states of America we had to implement a filter on the column that provides the information about the country wise airport codes. After inspecting the website, we observed that there is class attribute that provides with our desired information and thus we implemented a find function which searches for the class attribute and extracts the city and country of the airport code in a text format. A split function is applied on the extracted data to modify it into a more organized tabular format.

Sample output while parsing the HTML page containing the airport code:


```
'\n<li>\n <a href="/flights-to-anaa-aaa" title="Direct flights to A
naa (AAA)">\n <div class="country-airports-info">\n <p class="airpo
rt-city"><span class="airport-code">AAA</span><span class="airport-
city-country">Anaa, French Polynesia</span></p>\n <p><span class="a
irport-name">Anaa Airport</span></p>\n </div>\n </a>\n </li>\n\nO/
P:\n\n{"link":"flights-from-anaa-aaa", "code":AAA, "name":"Anaa Air
port", "city":"Anaa", "country":"French Polynesia"}\n'
```


This data set is now a modified collection of information on which we have implemented a filter of “United States” on the country column. This relevant information is converted into a Data Frame and is stored as a .csv file in the data folder.

Our next step is to find and extract the relevant data by visiting every airport link in USA and find the connectivity amongst the airports within USA.

get_airport_info: This step is implemented on our modified Data Frame by creating a function that takes in two arguments i.e.. Destination (airport_codes) and source (airport_codes). In this function the data parsing is applied to extract the information such as the destination flights per month. This data is then converted to an integer for storing the int value of the count of destination flights.

Sample Output of the Data Frame:

	link	code	name	city	country
0	/flights-from-allentown-abe	ABE	Lehigh Valley International Airpo	Allentown	United States
1	/flights-from-abilene-abi	ABI	Abilene Regional Airport	Abilene	United States
2	/flights-from-ambler-abl	ABL	Ambler Airport (FAA: AFM)	Ambler	United States
3	/flights-from-albuquerque-abq	ABQ	Albuquerque International Sunport	Albuquerque	United States
4	/flights-from-aberdeen-abr	ABR	Aberdeen Regional Airport	Aberdeen	United States

get_destination_info: This function is used to extract the data related to the destination airport_codes of the flights that were originating from our concerned airports. As per the observation there are numerous destinations for a flight originating from a source including the international airports. Hence in this function we are extracting the city-country and other details of the destination airport and implementing a filter of "United States". This filter is applied so that we can extract only those paths (source – destination) which are being used within Unites States of America. This destination airport code information is then appended into the pre-existing data frame.

It is then iterated over every airport to find the list of destination airport_codes against each of them. This information is then grouped and appended in the data frame which now consists the vital information on routes such as the Source airport and its numerous destination airports. This is now extracted into .csv file and stored in the data folder as routes.csv.

Sample Output of parsing data through the website to extract destination codes:

```
/flights-from-destin-fort-walton-beach-vps
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=24.0), HTML(value='')))
/flights-from-vieques-vqs
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=4.0), HTML(value='')))
/flights-from-vero-beach-vrb
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=2.0), HTML(value='')))
/flights-from-wales-waa
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=4.0), HTML(value='')))
/flights-from-stebbins-wbb
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=3.0), HTML(value='')))
/flights-from-selawik-wlk
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=1.0), HTML(value='')))
/flights-from-meyers-chuck-wmk
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=1.0), HTML(value='')))
/flights-from-white-mountain-wmo
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=2.0), HTML(value='')))
/flights-from-napakiak-wna
```

Sample Output of the routes.csv file after data parsing and extraction:

	source_code	destination_code	flights_per_month	country
0	ABE	DTW	88	United States
1	ABE	CLT	79	United States
2	ABE	ORD	56	United States
3	ABE	ATL	42	United States
4	ABE	SFB	31	United States
...	...	...	...	...
6797	YAK	JNU	30	United States
6798	YAK	CDV	30	United States
6799	YKM	SEA	62	United States
6800	YUM	PHX	98	United States
6801	YUM	DFW	54	United States

6802 rows × 4 columns

Till now we had done an excellent job to find out the information that provided us the information on all the possible airway routes within the USA. But sadly, this was not enough for us to analyze the rate of spread of disease as we did not have enough details about these routes.

In order to fetch a detailed information about various air travel through the various airports we extracted and parsed a data dump from a website (link: https://openflights.org/data.html). This website has a airport.dat file which consists of an extensive detailed information about every airport such as the latitude, longitude, city, country, time-zone etc.

In order to extract the useful information from this data dump we considered only the important columns related to our application and dropped the unnecessary ones. It is then stored in a .csv file which is then further used to perform some pre-processing in order to filter out the information of the airports which are in United States only. This data frame is now a vital data set which helps us in determining the various informative details that can used to analyze the spread of a disease through air travel. This data set is stored in a .csv file in the data folder.

Sample output of the processed data frame:

	ID	Name	City	Country	IATA	Lat	Long
0	3411	Barter Island LRRS Airport	Barter Island	United States	BTI	70.134003	-143.582001
2	3413	Cape Lisburne LRRS Airport	Cape Lisburne	United States	LUR	68.875099	-166.110001
3	3414	Point Lay LRRS Airport	Point Lay	United States	PIZ	69.732903	-163.005005
4	3415	Hilo International Airport	Hilo	United States	ITO	19.721399	-155.048004
5	3416	Orlando Executive Airport	Orlando	United States	ORL	28.545500	-81.332901

The data that we had acquired till now was still incomplete as few more details about a flight trip such as the stops, airline etc. was still missing. In order to achieve those data, we extracted this relevant data dump from the website (link: https://openflights.org/data.html) which consisted of file name routes.dat. This data is then processed and stored on a connections_world.csv file in the data folder.

In order to visualize the spread of airports across the US we plotted the latitude and longitude of the airports from the data that we had extracted into one of our data frames.

In order to plot this, we have used the matplotlib library and have provide the colors of the land and sea along with the borders.

Sample Output of the location of various Airports across USA:

Now in order to create a complete informative data frame we took the airport routes data set that we had stored earlier and performed a left- join on the flight connections data dump that we recently created. Due to this operation, we have now achieved a data set which contains a detailed information about all the possible travel transactions that can be performed from an airport.

Sample output after the above operation:

	flights_per_month	country	airline	airline_id	source_airport	source_airport_id	dest_airport	dest_airport_id
0	88	United States	DL	2009	ABE	4355	DTW	3645
1	79	United States	AA	24	ABE	4355	CLT	3876
2	79	United States	US	5265	ABE	4355	CLT	3876
3	56	United States	UA	5209	ABE	4355	ORD	3830
4	42	United States	AF	137	ABE	4355	ATL	3682

This is a complete informative data set that we achieved after it was joined, processed and extracted from various data sources, but it still needs some data cleaning operation to be performed on it. So the first step to this is to analyze the amount of null values in our data set.

That comes upto:

```
flights_per_month     0.000000
country               0.000000
airline              17.478977
airline_id           17.478977
source_airport       17.478977
source_airport_id    17.478977
dest_airport         17.478977
dest_airport_id      17.478977
dtype: float64
```

While moving ahead with the simulation part for this project we realized that in order to create a simulation of the spread of disease we need to include the information about the latitude and longitude of the airports so that a link can be generated between the airports that perform air travel between each other. Hence we added US Airports Longitude and Latitude data to the connections table above by performing a left outer join of the latitude data on the connections table data.

So now after performing these operations we had created a base data that can perform an analysis on the spread of any disease in the United States.

But in order to determine the rate of transfer of this disease we added one more important factor that impacts the rate of transmission i.e., population of the region/country. In order to get this information on the population we had performed a data processing in the first section that provided us the with USA's city wise population which is stored in the .csv file in the data folder.

This information that we have pulled out was for top 200 USA cities This file provides the information about the population city-wise along with few other factors such as total area of the region, population density etc.

Sample output of the population data set:

	Rank	Name	State	2020 Pop	2010 Census	Change	Density (km²)	Area (km²)
0	1	New York City	New York	8323340	8190360	1.62%	10699	778
1	2	Los Angeles	California	4015940	3795370	5.81%	3306	1215
2	3	Chicago	Illinois	2694240	2697530	-0.12%	4575	589
3	4	Houston	Texas	2340890	2098450	11.55%	1412	1658
4	5	Phoenix	Arizona	1703080	1449300	17.51%	1270	1341

As we had the flight and airport information about almost all the cities in USA we fetched the population data from a bigger dataset from opendatasoft.com only for USA. After this operation we performed a left outer join on airport connections data and merged it with the population data.

The sample output after this merger is as below:

	ID	Name	City	Country	IATA	Lat	Long	population
0	3411	Barter Island LRRS Airport	Barter Island	United States	BTI	70.134003	-143.582001	NaN
1	3413	Cape Lisburne LRRS Airport	Cape Lisburne	United States	LUR	68.875099	-166.110001	NaN
2	3414	Point Lay LRRS Airport	Point Lay	United States	PIZ	69.732903	-163.005005	NaN
3	3415	Hilo International Airport	Hilo	United States	ITO	19.721399	-155.048004	43263.0
4	3416	Orlando Executive Airport	Orlando	United States	ORL	28.545500	-81.332901	270934.0

As this was a left join operation it was bound that there will be some null values in the population column that was merged with the connections data set. In order to get a brief idea on the null values present in various columns of the data set we pulled out brief statics about it as mentioned below:

```
ID             0.000000
Name           0.000000
City           0.000000
Country        0.000000
IATA           0.000000
Lat            0.000000
Long           0.000000
population    25.259792
```

As it is clearly evident that 25% of the population data is missing we came with a solution in which we will provide an estimated value of the population in the null areas by performing a Hypothesis. The Hypothesis is that the number of flights from the city is proportional to the city population.

Based on this calculation an estimate value can be predicted for the cities for which there is no data present in the data set.

In order to implement and perform this hypothesis a few data processing steps are required. The first step to this aggregating the flight_per_month column according to the source_airport code. These flights per month values are then merged with the connections data set. The merging process is a left join of the collections data on the data set containing values for flights_per_month.

Sample output for flights per month according to source aiport:

	source_airport	flights_per_month
0	ABE	485
1	ABI	266
2	ABL	100
3	ABQ	2329
4	ABR	62

Sample output after merging the flights_per_month data to connections data set:

	ID	Name	City	Country	IATA	Lat	Long	population	source_airport	flights_per_month
0	3411	Barter Island LRRS Airport	Barter Island	United States	BTI	70.134003	-143.582001	NaN	BTI	27.0
1	3413	Cape Lisburne LRRS Airport	Cape Lisburne	United States	LUR	68.875099	-166.110001	NaN	LUR	4.0
2	3414	Point Lay LRRS Airport	Point Lay	United States	PIZ	69.732903	-163.005005	NaN	PIZ	27.0
3	3415	Hilo International Airport	Hilo	United States	ITO	19.721399	-155.048004	43263.0	ITO	634.0
4	3416	Orlando Executive Airport	Orlando	United States	ORL	28.545500	-81.332901	270934.0	NaN	NaN

The brief statics of the null values in the table after this operation:

```
ID                   0.000000
Name                 0.000000
City                 0.000000
Country              0.000000
IATA                 0.000000
Lat                  0.000000
Long                 0.000000
population          27.548807
source_airport       0.000000
flights_per_month    0.000000
```

As still there are 27% null values in the population column we will go ahead with our next step in the Hypothesis process.

In our next step we are finding out the flight coefficient. This flight coefficient according to our Hypothesis is derived through a mathematical division operation in which the Total Population of US is divided by the flights per month that is operating in US.

In order to get the total flights_per_month we just aggregated the entire flights count upto the country level. The total population of the country is a dynamic and live updated value which is fetched from the website (link: https://worldpopulationreview.com/countries/united-states-population). Thus due to this the flight coef. is derived as per below:

	flights_per_month	total_population	coef
Country			
United States	982351.0	331848096	337.810107

Thus, this coef. Is now used to fill in the missing data in the population section by the calculation of:

City estimated Population: flight_coefficient * flights_per_month

	ID	Name	City	Country	IATA	Lat	Long	population	source_airport	flights_per_month	flights_coefficient
0	3411	Barter Island LRRS Airport	Barter Island	United States	BTI	70.134003	-143.582001	NaN	BTI	27.0	337.810107
1	3413	Cape Lisburne LRRS Airport	Cape Lisburne	United States	LUR	68.875099	-166.110001	NaN	LUR	4.0	337.810107
2	3414	Point Lay LRRS Airport	Point Lay	United States	PIZ	69.732903	-163.005005	NaN	PIZ	27.0	337.810107
3	3415	Hilo International Airport	Hilo	United States	ITO	19.721399	-155.048004	43263.0	ITO	634.0	337.810107
5	3417	Bettles Airport	Bettles	United States	BTT	66.913902	-151.529007	NaN	BTT	31.0	337.810107
...	...	...	...	...	...	...	...	...	...	...	...
970	8200	Toksook Bay Airport	Toksook Bay	United States	OOK	60.541401	-165.087006	NaN	OOK	0.0	337.810107
994	8314	Hilton Head Airport	Hilton Head Island	United States	HHH	32.224400	-80.697502	40512.0	HHH	198.0	337.810107
1112	9543	Ogden Hinckley Airport	Ogden	United States	OGD	41.195900	-112.012001	85444.0	OGD	7.0	337.810107
1115	9739	Napakiak Airport	Napakiak	United States	WNA	60.690300	-161.979004	NaN	WNA	54.0	337.810107
1116	9744	Napaskiak Airport	Napaskiak	United States	PKA	60.702900	-161.778000	NaN	PKA	54.0	337.810107

The values that will be filled in the empty slots of the population column is an estimate value, but it is still enough for our application to perform operations and simulations.

These entire series of steps starting from data parsing through various websites, data extraction and data processing through various functions has provided us with the desired data set which are now divided and stored in two separate .csv files:

1: airport_data_preprocessed.csv & 2: connections_preprocessed.csv

The sample output for both these .csv files are:

Connections:

	flights_per_month	country	airline	airline_id	source_airport	source_airport_id	dest_airport	dest_airport_id	latitude_source	longitude_source	latitude
	88	United States	DL	2009	ABE	4355	DTW	3645	40.652100	-75.440804	42.2123
	90	United States	DL	2009	ALB	3864	DTW	3645	42.748299	-73.801697	42.2123
	53	United States	DL	2009	APN	5720	DTW	3645	45.078098	-83.560303	42.2123
	337	United States	AF	137	ATL	3682	DTW	3645	33.636700	-84.428101	42.2123
	337	United States	AM	321	ATL	3682	DTW	3645	33.636700	-84.428101	42.2123

Airport Data:

	ID	Name	City	Country	IATA	Lat	Long	population	source_airport	flights_per_month	flights_coefficient
0	3411	Barter Island LRRS Airport	Barter Island	United States	BTI	70.134003	-143.582001	9121.0	BTI	27.0	337.810107
1	3413	Cape Lisburne LRRS Airport	Cape Lisburne	United States	LUR	68.875099	-166.110001	1352.0	LUR	4.0	337.810107
2	3414	Point Lay LRRS Airport	Point Lay	United States	PIZ	69.732903	-163.005005	9121.0	PIZ	27.0	337.810107
3	3415	Hilo International Airport	Hilo	United States	ITO	19.721399	-155.048004	43263.0	ITO	634.0	337.810107
5	3417	Bettles Airport	Bettles	United States	BTT	66.913902	-151.529007	10473.0	BTT	31.0	337.810107

- **Data Modelling and Simulation / Mathematical analysis/evidence**

This is the most important department of our project where in we have implemented all our logics, formulas, algorithms, data-plotting, simulation-code, etc.

Disease-Spread:

We have designed and implemented this section to understand and analyze the spread of the disease through air -travel.

To start with the implementation of this section we first tried to read and print the .csv files that we had created in our earlier sections. The two important .csv files that we are working on are:

- connections_preprocessed.csv
- airport_data_preprocessed.csv

Our entire application's model is dependent on analyzing the spread of the disease through air-travel. In-order to achieve that, we have implemented it in a step-by-step analysis process.

Probability of infected plane: This function provides us with the information on what will be the probability that an airplane is infected that is originating from an infected city and how will it impact the spread of a disease from an infected city via an infected airplane.

Probability_of_infected_plane: I / N

Where, I - the number of people infected in the city

N - the total population in the city

Probability of Infected City: In this we are trying to find the probability of a city to get infected from an incoming plane. The calculation and working of this function are totally dependent on factors like the infectious_sources, population_surces and daily_flight_sources. To start with, we have considered that the probability of all the planes is healthy is equal to 1 and have implemented a zip function which performs the parallel iteration. By considering the above factors we can compute that,

P(new city infected) = 1 - P(all incoming planes are healthy)

1 - [P(all planes from city A are healthy) * P(all planes from city B are healthy) * ...]

1 - [(1 - I_A/N_A) ^ f_A * (1 - I_B/N_B) ^ f_B * ...]

Calculate Reproduction Number: The reproduction number is dependent on the following factors:

- maxR: maximal possible R value during the simulation
- minR: minimal possible R value during the simulation
- simulationDays: number of days in the simulation run
- interventionStart: number of days after which R starts going down
- interventionPeriod: number of days it takes from the intervention start to drive R value to its minimal values

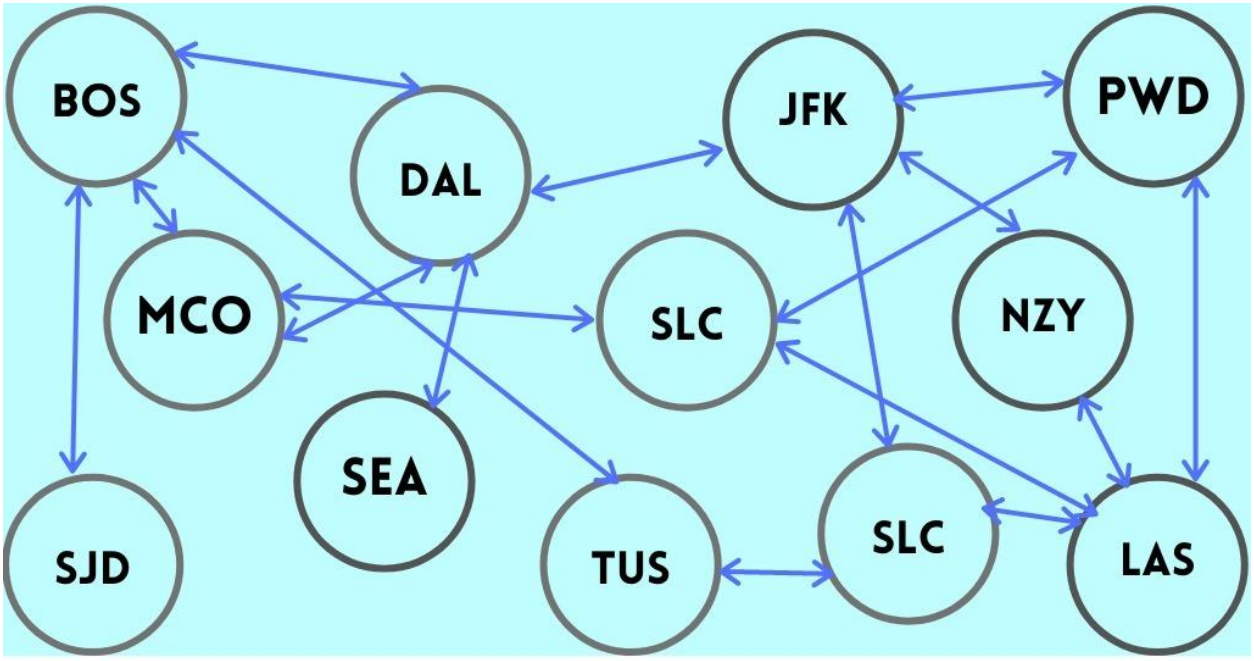
In order to determine the final output, we have concatenated the following three terminologies:

- reproductionIntervention = expit(np.linspace(-5, 3, num=interventionPeriod))[::-1]
- reproductionIntervention = reproductionIntervention * (maxR - minR) + minR
- reproductionNumber = np.concatenate((reproductionNumber, reproductionIntervention, np.repeat(minR, simulationDays)))

In order to take our study forward, we implemented "**NetworkX".** This is a Python package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks. Through this package we were able to create a store our complex information on this

package's inbuilt data structure named DiGraph(). A DiGraph stores nodes and edges with optional data, or attributes and holds directed edges

To steer this ahead we pulled out the unique source-destination pairs data from our connection data frame in order to create a connections_graph. This connections_graph is created using the function from our above python package's function named from_pandas_edgelist() which is vital in determining the neighboring cities.



DiGraph of US Aiports

Storing a directed graph of source and destination airport. To find a neighbor, call graph.neighbor(city) function and returns all nearby neighbors (Uses BFS). This graph represents the connections between two airports.

City Neighbors: This is used to find the neighboring cities for a given city by searching it in the DiGraph. This is one of the most important functions of our algorithm as it provides the information on the neighbors that a corresponding city has.

To pull out this information we need these two main arguments which are the city_name and the connections_graph which are then used to perform a Breadth-First-Search operation using the package's inbuilt function named neighbors(). The Digraph.neighbors returns all the neighbors of a given node.

In order to proceed ahead with our further analysis, we have done few computations in order to cater the information according to our formulas.

Starting with pulling out the Dictionary of Airport Code and their corresponding cities.

Eg: {
'BOS': 'Boston',
'PWM': 'Portland',
'PDX': 'Portland'
}

Dictionary of City and their corresponding airport codes (Few Cities have more than 1 airports).

Eg: {
'Boston': ['BOS'],
'Portland': ['PDX', 'PWM']
}

This above data is further refined by implementing a unique() function on it so that we can pull out the unique values and then it is stored as a data frame for further computations.

Next, we pull out the dictionary of Cities and their respective populations.

Eg: {
'Boston': 667137.0,
'Portland': 632309.0
}

This is now further computed to pull out dictionary of Cities and their resp neighbors (Direct Flight connections)

Eg: {

'Boston': ['ATL', 'BWI', 'CLT', 'DCA', 'DEN','DFW', 'DTW', 'EWR', 'FLL', 'JFK', 'LAX', 'LEB', 'MCO', 'MIA', 'ORD','PHL', 'RDU', 'RSW', 'SEA', 'SFO', 'TPA'],

'Portland': ['ATL', 'BOI', 'DEN', 'DFW', 'GEG', 'LAS', 'LAX', 'MSP', 'ORD', 'PDT', 'PHX', 'PSP', 'RDM', 'RNO', 'SAN', 'SEA', 'SFO', 'SJC', 'SLC', 'SMF', 'SNA', 'BWI', 'JFK']

}

Number of flights between source & destination every month:

Eg: {
('BOS', 'ATL'): 58.8,
('BOS', 'BWI'): 14.8,

```
    ('BOS', 'CLT'): 21.400000000000002,
    ('BOS', 'DCA'): 17.1,
    ('BOS', 'DEN'): 13.6,
    ('BOS', 'DFW'): 20.4,
    ('BOS', 'DTW'): 8.8
}
```

Using the above information, a parallel iteration is performed using zip() to determine the number of flights between the source and destination airports.

find_neighbours_of_a_city: This function is used to pull out the neighboring cities from the City_Neighbors data frame.

find_all_current_healthy_airports: This provides the list of healthy airports by removing the set of infected airports from overall list.

find_all_current_infected_airports: This provides the list of infected airports by creating an intersection between the overall list of airports with the set of Infected airports.

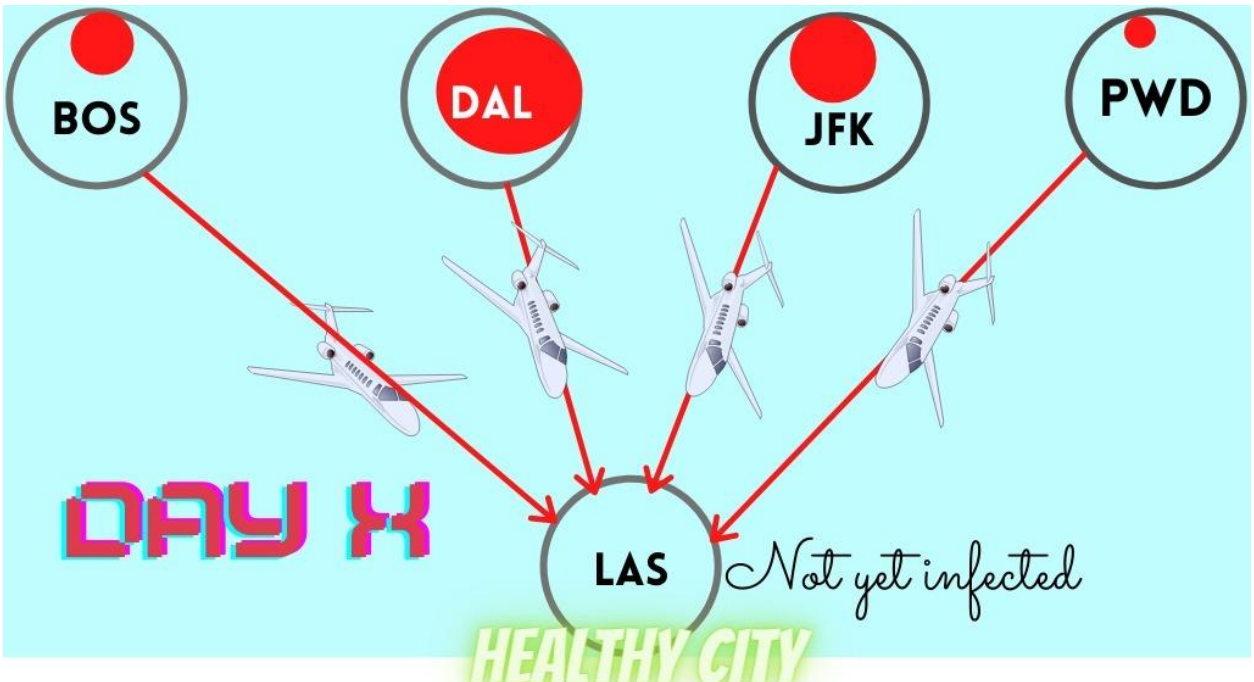
find_total_number_of_flights: This function pulls out the number of flights operating between a source and destination.

get_infected_number: It is used to extract the information that provides us with the infections on a day according to the infections happening in the cities on a particular day.

calculate_infection_prob: This function performs the operation of calculating the probability of the spread of infection based on important factors like current susceptible city, the list of infected population, the infected number, the infected city, etc.

run_neighbor_simulation: This is used to create a simulation based on the S.I.R model that we have implemented. This simulation helps in understanding the spread of the disease on a day-to-day basis based on some key factors like day, infected, susceptible, recovered, etc. This SIR model is processed by providing the city population and an assumed figure of the first infected number (which we have considered as 100).

run_infectious_city_simulation: In this function we are able to pull out the information about the infectious cities based on the commutes that are happening through air travel. The information needed to perform this simulation is the neighboring airports, the susceptible airports, the susceptible cities, the current infection city source etc.



Spreading Infection

I - number of infected in the city,
N - total population of the city and
f - Flights per day then,

Probability of a Plane Infected = I / N
Probability of a City Infected by Incoming Planes =
1 - [(1 - Ia/Na)fa * (1 - Ib/Nb)fb * (1 - Ic/Nc)fc * ...]

Our S.I.R model is quite efficient in providing the detailed overview on how the disease will perform over a period of days and how quickly will it spread across the country from the day this disease has been detected. In order to get the best of our S.I.R model we need to provide some pre-defined values so that it can perform its computation in order to simulate the result.

Sample values that are pre provided:

NUMBER_OF_SIMULATIONS = 3
SIMULATION_DAYS = 100
VERBOSE = True

max_R = 7
min_R = 1
GAMMA = 0.034
intervention_start=30
intervention_period=14

These assumed values are now fed into our model through various functions starting with equating the reproduction number of the disease through calculate_reproduction_number() function that takes 5 arguments namely max_R, min_R, simulation_days, intervention_start, intervention_period.

The next function to be executed is the main function of our S.I.R model. This function will return the S, I & R values of the disease which will be used in the next function to perform the appropriate simulation. This function takes in values of city_population, first_infected_number, reproduction_number, gamma, simulation_days.

The final task of this section is to now perform a simulation based on the computations that we have done so far. In order to execute that it will need the following information infected cities, infected airports, new infections, the value of S, I, R, susceptible cities, infected cities, current infection source etc. As we have already computed the above information, we have used TQDM_Notebook to perform the simulation task. TQDM notebook is a Jupyter Notebook progress bar decorator for iterators.

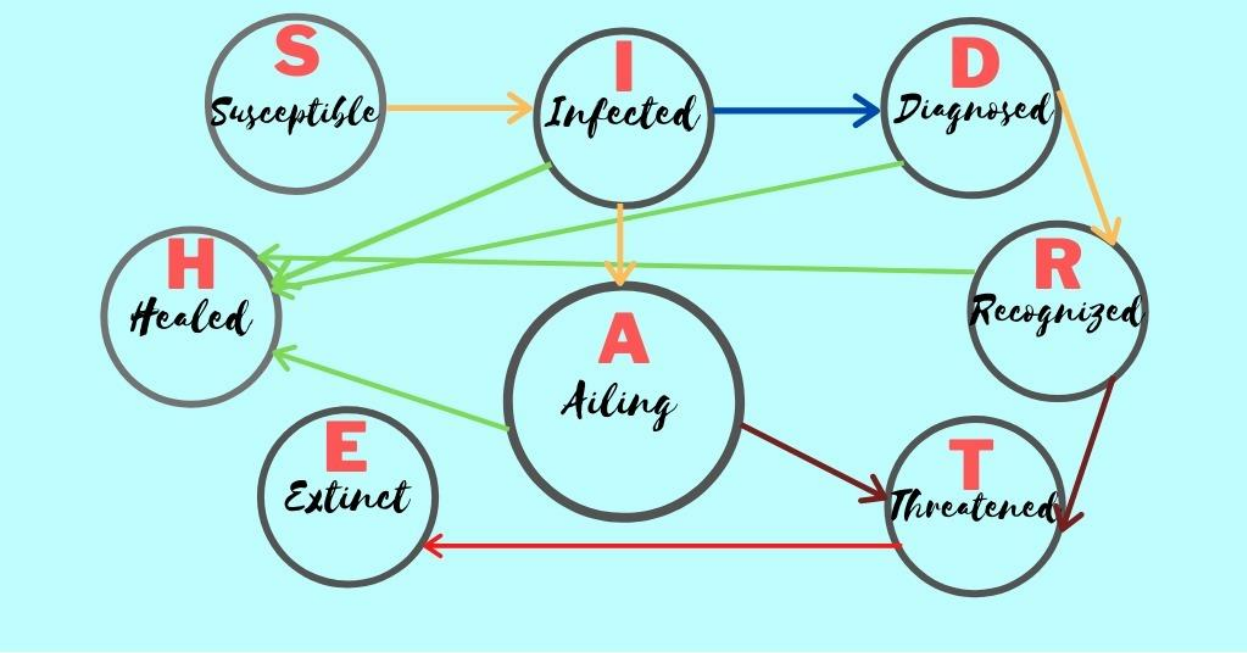
S I R model:

This is the model on which the result and performance of our entire application is based on. While browsing and researching upon the details on the key factors related to the spread of a disease we had stumbled upon this term known as SIR model. This SIR model is a small subset or a simpler version of a model named SIDARTHE Model. The SIDARTHE model is a complex model which covers a wide variety of factors in determining the overall performance and impact of the disease. The factors considered by the SIDARTHE model is based on how the disease transmission behaves in humans. Thus, this model categorizes the population in different categorizes depending upon their interaction with the virus over a period of days.

The infected category consists of the population that have contracted the disease but are either asymptotic or symptotic. The population that are showing the symptoms and are tested with the disease are moved into the Diagnosed Category and the population with asymptotic behavior are considered in the Ailing category. The Susceptible population are those which are the people who are not infected with the disease out of the entire available population. The part of the population from the Diagnosed category can either be moved to the Recognized category or the Healed Category based on how the individual is treated.

The individual can be moved to the threatened section if the health condition is deteriorating and if it then stops responding then it will be finally moved to the Extinct category

.



SIDARTHE Model

Graphical scheme representing the interactions among different stages of infection in the mathematical model SIDARTHE: S, susceptible (uninfected); I, infected (asymptomatic or pauci-symptomatic infected, undetected); D, diagnosed (asymptomatic infected, detected); A, ailing (symptomatic infected, undetected); R, recognized (symptomatic infected, detected); T, threatened (infected with life-threatening symptoms, detected); H, healed (recovered); E, extinct (dead).

Thus, we observed that this model is quite complex and is covering a wide range of categories in order to analyze the impact and performance of a disease. But according to our scalability of the project we decided to pick the key factors that can be used to analyze the impact and performance of the disease.

Thus, the categories that we have considered are :

S = S(t) is the number of susceptible individuals,

I = I(t) is the number of infected individuals, and

R = R(t) is the number of recovered individuals.

The SIR model functions by creating a class with the first function that is defined is an Init function which works as a constructor function in which we pass along the information of

N is the total population

I0 is the initial number of infected individuals,

S0 - Everyone else is susceptible to infection initially

b is contacts per day that are sufficient to spread the disease, and

k is our assumption that a fixed fraction k of the infected group will recover during any given day i.e., mean recovery rate, (in 1/days)

Upon initialization the model moves on to the next function named _compute() which computes the differential values of S, I and R over a period of days. To compute such a differential equation, this function takes in the following arguments:

The SIR model differential equations

y - SIR vector
t - Time in days
N - Population
b - Contacts per day
k - Mean Recovery Rate

The equations are referred from https://www.maa.org/press/periodicals/loci/joma/the-sir-model-for-spread-of-disease-the-differential-equation-model

The values of the S, I & R can be extracted from y(SIR vector) and are used in the computations ahead.

The Susceptible Equation
dSdt = -b[int(t)] * S * I / N

The Infected Equation = -The Susceptible Equation - The Recovered Equation
dIdt = b[int(t)] * S * I / N - k * I

The Recovered Equation
dRdt = k * I

After computing the differential values, we can now move ahead onto the main function of this section that provides us with the actual value of S,I & R for the disease that we are computing.

In order to do that we take into consideration few factors related to the formulas such as:

y0 - Initial conditions vector
S0 - Initial Susceptible (N - I0)
I0 - Initial Infected
R0 - zero recovered at initial stage

In order to calculate the final values of S,I & R we have imported the scipy python package from which we can use the odeint function that will provide us with the desired result.

Visualize Spread:

Now we move on to the next section of our project and this section handles the visualization of all the functions and the model that we have built so far. The python package used in the section includes pandas, numpy, matplotlib, seaborn, tqdm, ipywidgets etc.

Starting with this section we have created a function that reads the data from various data files that are created and uses it as a data frame to plot the graphs. In this we group the count of infected data day wise.

The function named "plot_with_std", plots the data according to mean and average values. Then we read the data from the "Infected_cities" file that we had created earlier which consists of day wise count of the infected cities.

Sample output of the above file:

```
day
0       1
2       2
3      44
4      96
5     126
6     148
7     168
8     187
9     211
10    235
11    248
12    262
13    272
14    289
15    305
16    315
17    332
18    343
```

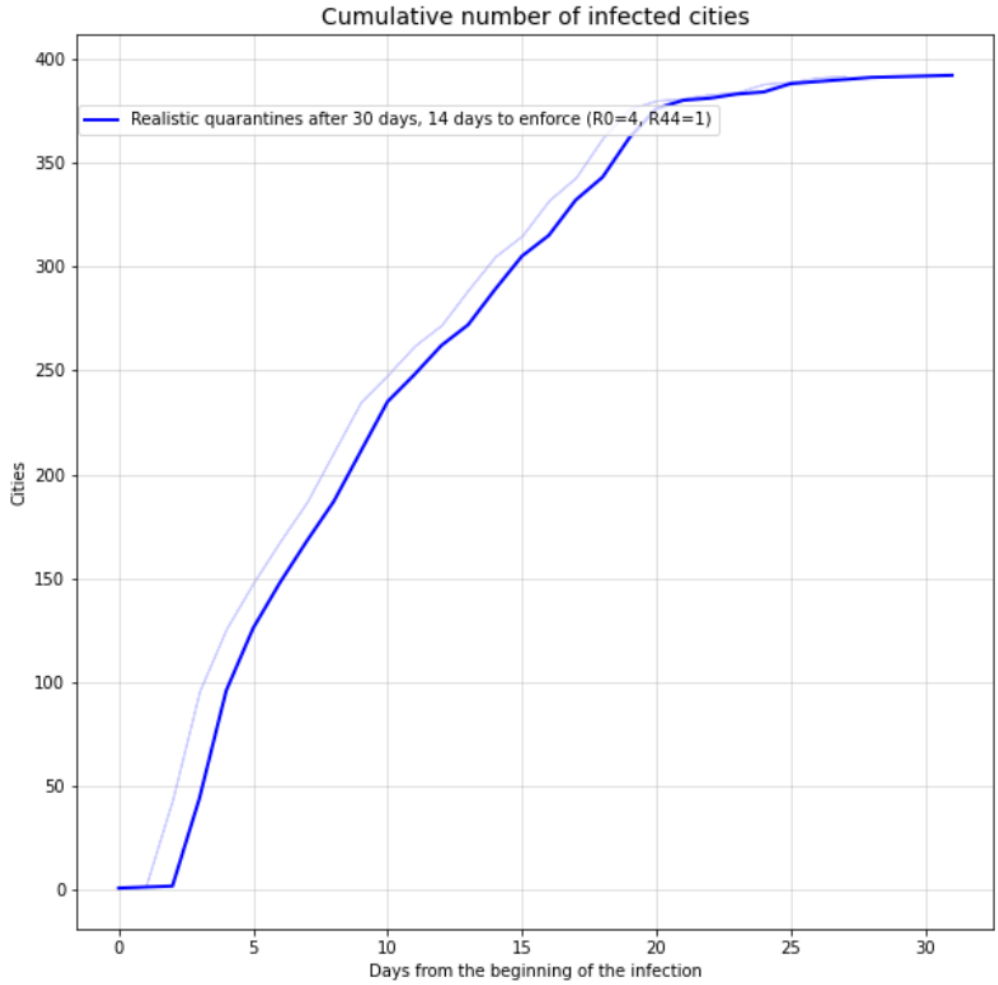
Now we move ahead to read the data from "airport_data_preprocessed" file that has the sample output as below:

	ID	Name	City	Country	IATA	Lat	Long	population	source_airport	flights_per_month	flights_coefficient
0	3411	Barter Island LRRS Airport	Barter Island	United States	BTI	70.134003	-143.582001	9121.0	BTI	27.0	337.810107
1	3413	Cape Lisburne LRRS Airport	Cape Lisburne	United States	LUR	68.875099	-166.110001	1352.0	LUR	4.0	337.810107
2	3414	Point Lay LRRS Airport	Point Lay	United States	PIZ	69.732903	-163.005005	9121.0	PIZ	27.0	337.810107
3	3415	Hilo International Airport	Hilo	United States	ITO	19.721399	-155.048004	43263.0	ITO	634.0	337.810107
5	3417	Bettles Airport	Bettles	United States	BTT	66.913902	-151.529007	10473.0	BTT	31.0	337.810107

After reading the data from the above two data frames, we move on to plotting them in this section. The first graph that we have plotted is dependent on the specific condition in which it take 14 days to enforce the quarantine rules after the 30 days of infection spread. This is a line graph of the

Cumulative number of infected cities which are plotted against the number of cities and the days from the beginning of the infection.

Sample output of the line graph:



As we go next, we move on to the final part of our disease spread visualization section where in we have implemented few functions that will process the spread data so that it can be merged, which will assist in creating a Map view of the disease spread between the cities in USA due to air travel. The functions consider all the important data frames that we have created earlier to plot a map which provides a strong visual information on the connectivity of the flights and a representation about the spread of the disease throughout the cities across USA.

For bringing this into effect, we utilized a python package that can create a basic map of USA. On top of that we considered the Latitude and Longitude information of the airports that are located in the various cities across USA. Then we plotted the visual connectivity between the source and destination airports. Then to sum up our visualization we considered our Infected_cities data frame to pull out the cities that have been infected over a period of time due to the transmission of disease occurring due to air travel.

Hence our final representation of the spread of disease within USA is as below:

John-Hopkins COVID19:

In this section we have visualized the data generated through the research done by John-Hopkins University on the spread of Covid19 disease. To go ahead with these operations, we have imported few python libraries namely pandas, matplotlib, seaborn, plotly, Ipython & datetime.

To start with this section, we read and store the disease data from the websites and divide them into the following three categories:

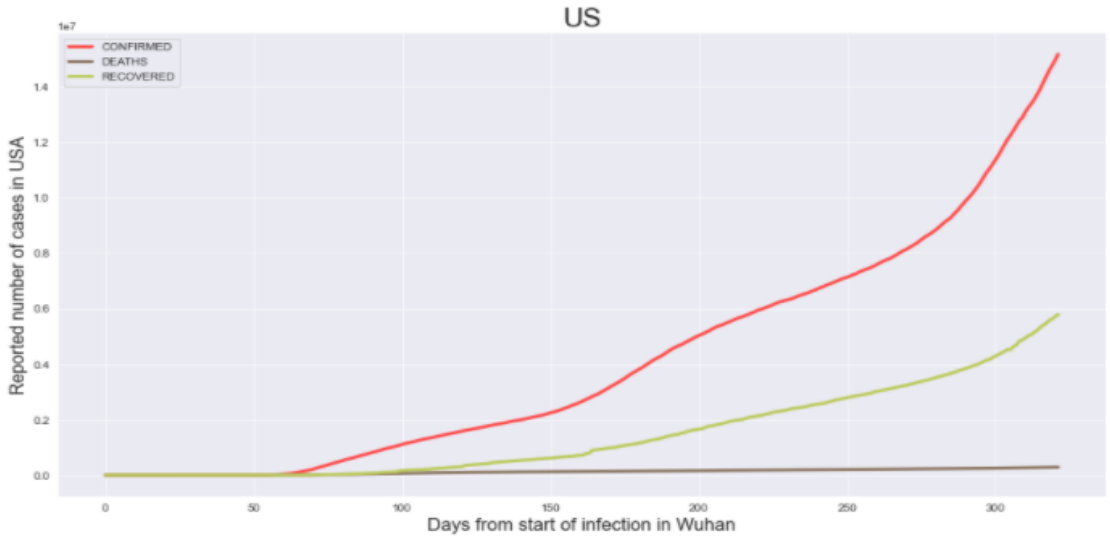
Confirmed – Link: https://raw.githubusercontent.com/CSSEGISandData/COVID-19/master/csse_covid_19_data/csse_covid_19_time_series/time_series_covid19_confirmed_global.csv

Deaths – Link: https://raw.githubusercontent.com/CSSEGISandData/COVID-19/master/csse_covid_19_data/csse_covid_19_time_series/time_series_covid19_deaths_global.csv

Recovered – Link: https://raw.githubusercontent.com/CSSEGISandData/COVID-19/master/csse_covid_19_data/csse_covid_19_time_series/time_series_covid19_recovered_global.csv

The data dump pulled out from these websites contain the information about all the countries across the globe. But, for our analysis we have implemented the filter on country column in order to restrict and store the data for United States of America only.

In the next step we plot the graph using the matplotlib library, wherein we plot the lines for Confirmed, Deaths and Recovered cases that were reported in USA against the series of days from the start of infection in Wuhan.



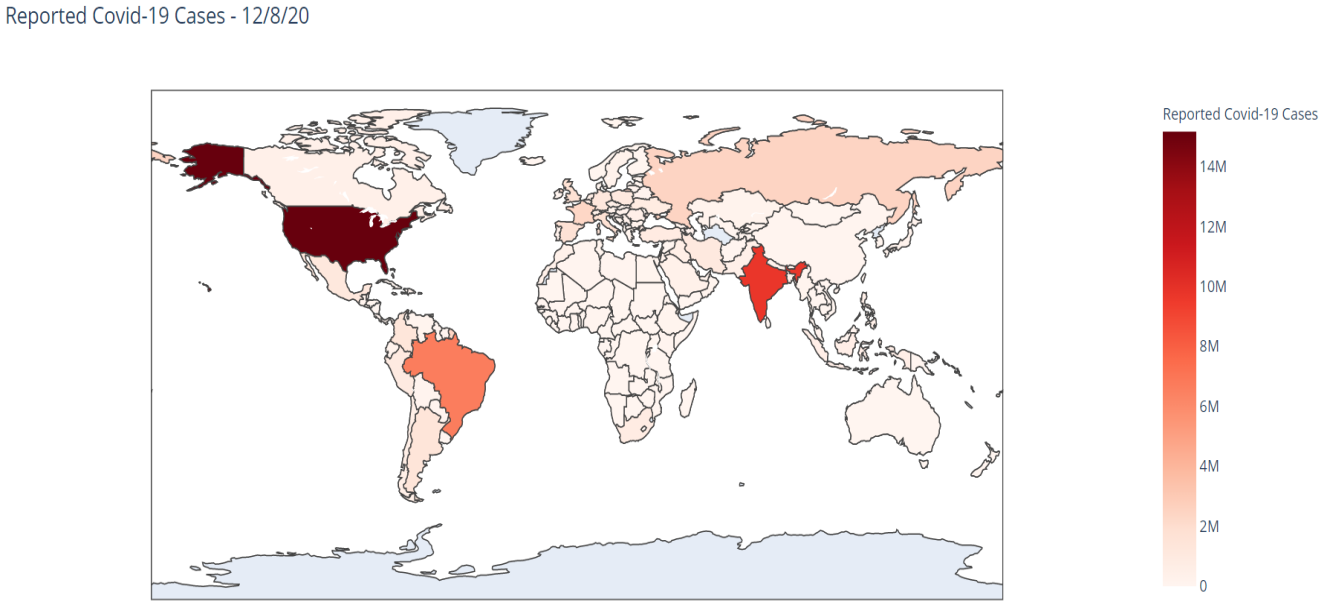
Now looking at a bigger picture on how the disease impacted the entire world, we started with processing the data for the total number of cases for every country across the globe.

The sample data for such a scenario is:

	Country	TotalCases
0	Afghanistan	47716
1	Albania	44436
2	Algeria	89416
3	Andorra	7127
4	Angola	15729

The next step is to plot this data on a map to provide us with a visualization for better understanding and also we extracted this map in a HTML format which is stored in the output folder with the name “total_cases_figure.html”.

Map for total number of cases worldwide:



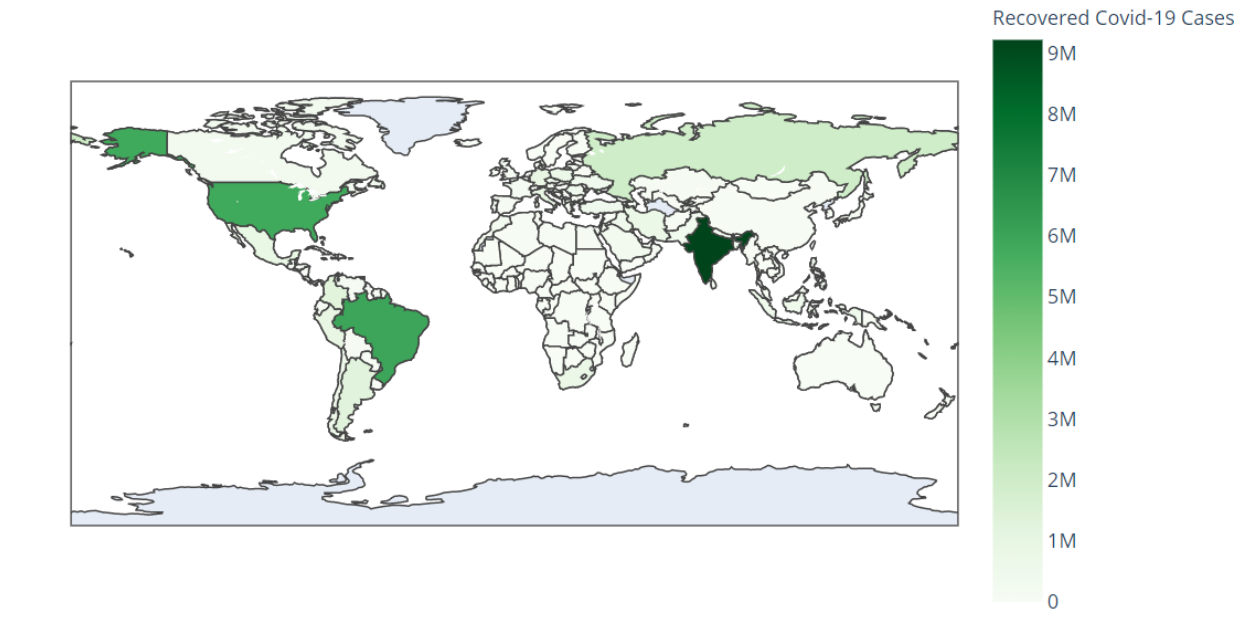
Similarly, we move on to our next part where we analyze and plot the map for the recovered cases across the globe. For this data processing is applied on the recovered data dump and the relevant information containing the country wise recovered cases is extracted.

Sample Output:

	Country	TotalRecovered
0	Afghanistan	37920
1	Albania	22527
2	Algeria	58146
3	Andorra	6367
4	Angola	8470

Using this we can plot the above data on a map to provide us with a visualization for better understanding and also we extracted this map in a HTML format which is stored in the output folder with the name “total_recoverd_figure.html”.

Map for total number of recovered cases worldwide:



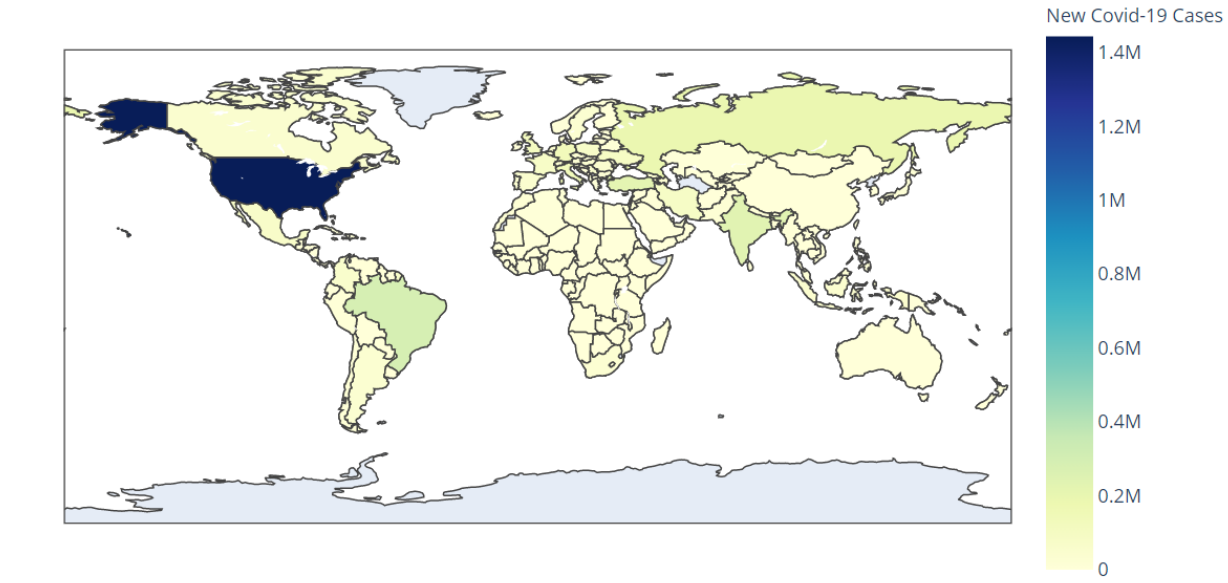
Moving onto the final map of our project, wherein we demonstrate the new cases of Covid19 that are emerging across the globe. For this, we process the data in order to extract the country wise count of the new cases that have emerged.

Sample output:

	Country	New Cases
0	Afghanistan	1200
1	Albania	5422
2	Algeria	5264
3	Andorra	337
4	Angola	478

Using this we can plot the above data on a map to provide us with a visualization for better understanding and also we extracted this map in a HTML format which is stored in the output folder with the name “new_cases_figure.html”.

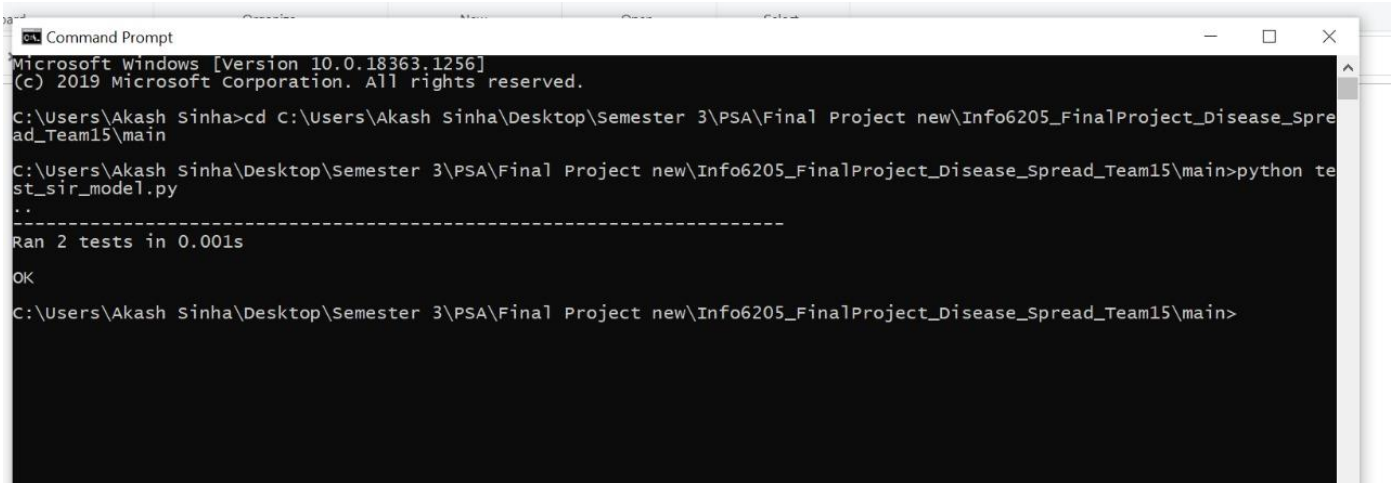
Map for new cases worldwide:



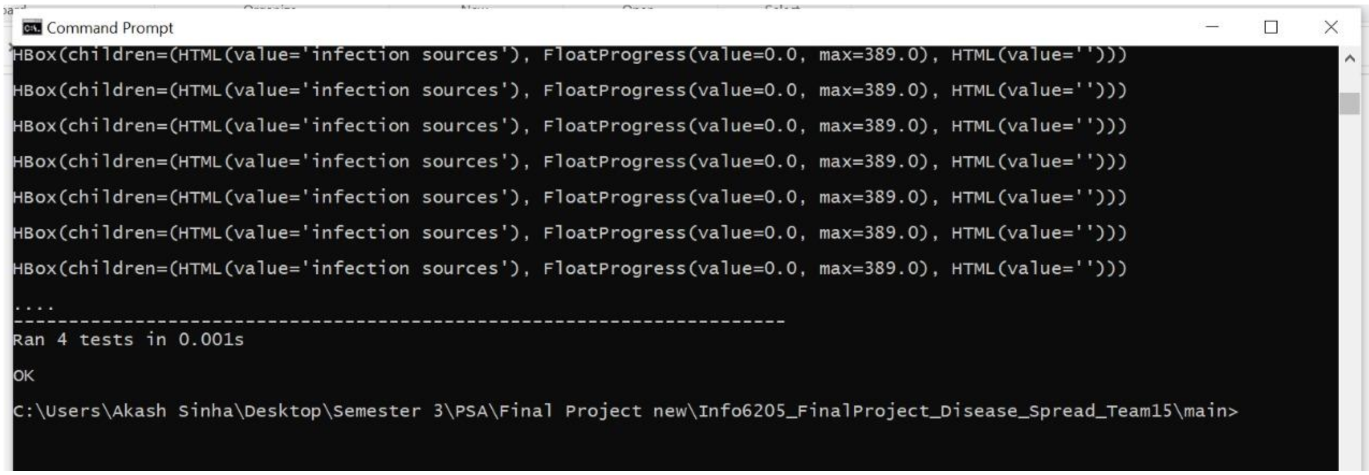
IV) Unit Testing:

We have unit testing for every python and jupyter notebook files. The successful tetsing of each file is as below:

SIR_Model_Unit_Testing



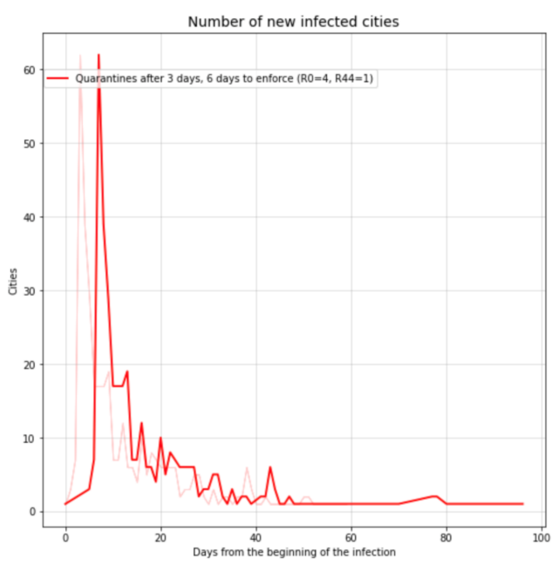
Disease_Spread_Unit_Testing

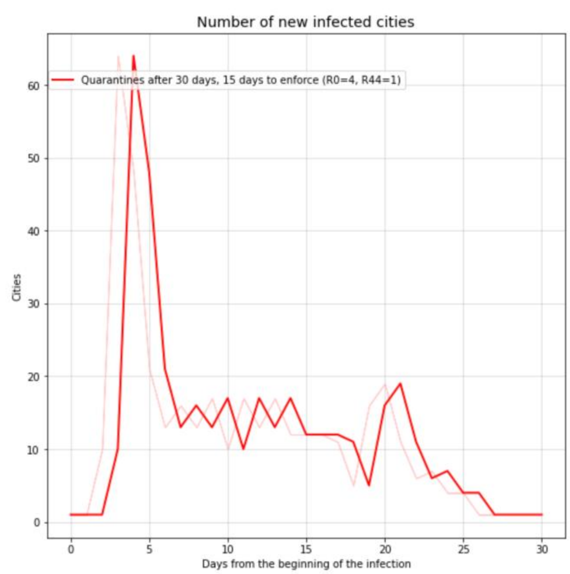


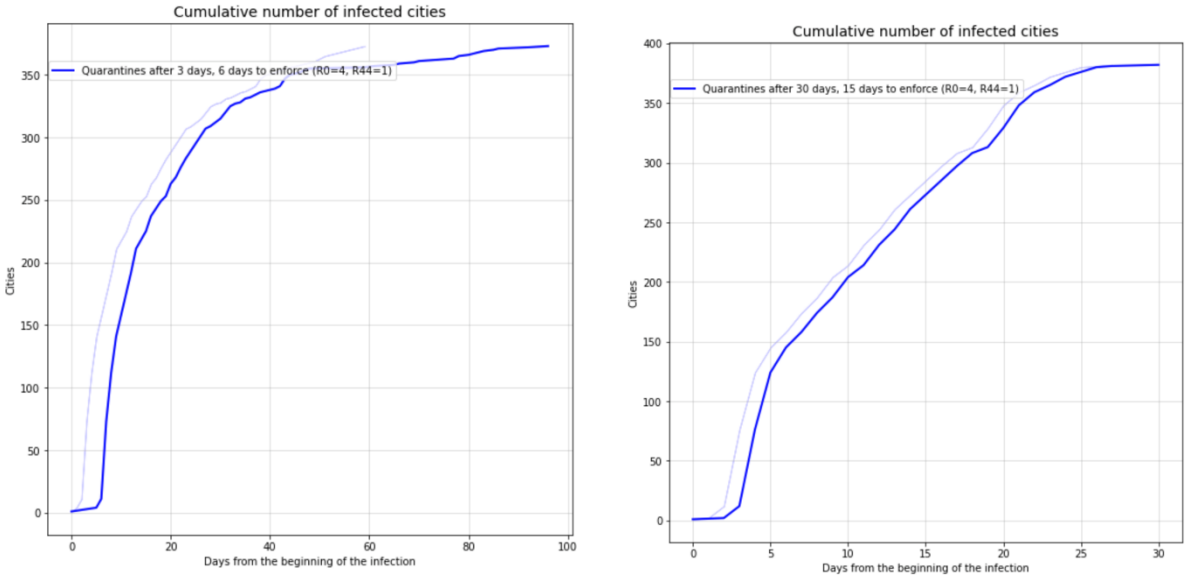
```
HBox(children=(HTML(value='infection sources'), FloatProgress(value=0.0, max=389.0), HTML(value='')))
HBox(children=(HTML(value='infection sources'), FloatProgress(value=0.0, max=389.0), HTML(value='')))
HBox(children=(HTML(value='infection sources'), FloatProgress(value=0.0, max=389.0), HTML(value='')))
HBox(children=(HTML(value='infection sources'), FloatProgress(value=0.0, max=389.0), HTML(value='')))
HBox(children=(HTML(value='infection sources'), FloatProgress(value=0.0, max=389.0), HTML(value='')))
HBox(children=(HTML(value='infection sources'), FloatProgress(value=0.0, max=389.0), HTML(value='')))
HBox(children=(HTML(value='infection sources'), FloatProgress(value=0.0, max=389.0), HTML(value='')))
....
----------------------------------------------------------------------
Ran 4 tests in 0.001s
OK
C:\Users\Akash Sinha\Desktop\Semester 3\PSA\Final Project new\Info6205_FinalProject_Disease_Spread_Team15\main>
```

V) Observations and Conclusions:

1. When the quarantine interventions are at following days and quarantine starts after the following days, the disease spreads differently.







So, it takes significantly more time to spread if quarantine measures are taken sooner than later.

2. If recovery rate of a country in general is low, the virus spreads faster. Below observations are examples of low k and high k.

Fig. High Recovery Rate – Day 5

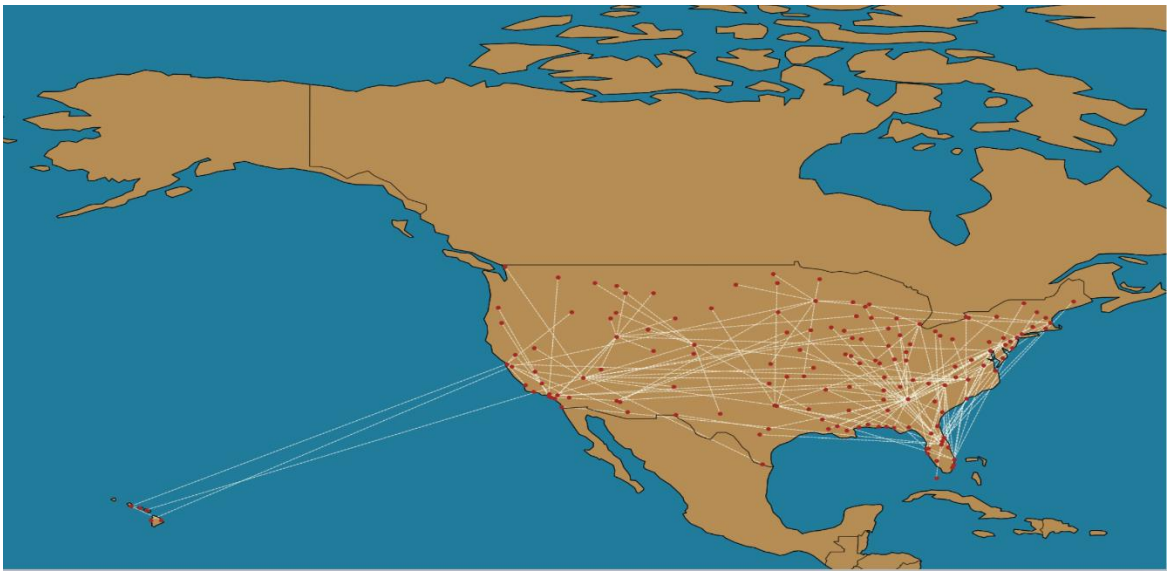


Fig. Low Recovery Rate – Day 5

3. According to the dataset of John Hopkins on COVID-19, the disease is still spreading rapidly in some countries while is in great control in others as shown below for the past week new cases.