

A dark blue vertical bar runs along the left edge of the page. A blue arrow points to the right from this bar, containing the date 7/25/2022.

7/25/2022

# Pandemic Flu Spread

Project Work for ISYE-6644 (SIMULATION)

Several thin, curved lines in dark blue and light gray originate from the bottom left corner and sweep upwards and to the right.

Group -112, Yoages Kumar Mantri, Saurabh Sinha, Arjun Jagini  
GEORGIA TECH

## Contents

Title: Pandemic flu spread .....	2
Abstract.....	2
Background and Description of Problem .....	2
Problem Statement.....	2
Background .....	2
Main Findings.....	3
Application Area.....	3
Models and Methods used .....	3
Data Collection.....	4
Approach.....	4
Solving the Problem .....	4
Implementation Details .....	5
Comparative Analysis of different models.....	8
Conclusion and Future work .....	10
References .....	10
Appendix .....	11

## Title: Pandemic flu spread

*Group Members – Yoages Kumar Mantri, Saurabh Sinha, Arjun Jagini*

### Abstract

The objective of this project is to simulate the spread of a pandemic. The focus was to predict how long the pandemic would last and estimate the number of people that would get infected.

To do so, we have implemented the Reed-Frost model (SIR) and its extensions. This model was developed to understand disease propagation and is based on the chain binomial model. There are multiple variations of this model, each making different assumptions with respect to susceptibility, immunity, incubation period, and more. The intention was to study and compare how the different variations of the Reed-Frost model affect the simulated pandemic spread.

To model any pandemic, it's important to understand how the disease spreads and then select the correct model. In general, we found that the pandemic increases the most in the initial stages. Hence preventive measures in the initial days of pandemic (like vaccination, isolation etc) are crucial to contain its overall impact.

## Background and Description of Problem

### Problem Statement

Consider a classroom of 21 elementary school kids. 20 of the kids are healthy (and susceptible to flu) on Day 1. Tommy (the 21st kid) walks in with the flu and starts interacting with his potential victims. To keep things simple, let's suppose that Tommy comes to school every day (whether or not he's sick) and will be infectious for 3 days. Thus, there are 3 chances for Tommy to infect the other kids - Days 1, 2, and 3. Suppose that the probability that he infects any individual susceptible kid on any of the three days is  $p = 0.02$ ; and suppose that all kids and days are independent (so that you have i.i.d. Bern(p) trials).

If a kid gets infected by Tommy, he will then become infectious for 3 days as well, starting on the next day.

### Background

We will simulate the spread of the flu in the classroom using the Reed-Frost model. This is a set of models developed by Lowell Reed and Wade Hampton Frost of Johns Hopkins University in the 1920s. The epidemiological models were designed to study the propagation of acute, infectious diseases.

It is based on the chain binomial model. Chain binomial models are developed from simple binomial model by assuming that infection spreads from individual to individual in discrete time units, producing chains of infection governed by binomial distribution.

Inputs to the model include the size of the population, how many people are infected, how many are immune, and the probability of an infected person spreading the disease to a susceptible person.

The probability of a susceptible person becoming infected can be denoted as:

$$1-(1-p)^{C_t}$$

where:

- $C_t$  = Number of infections at time  $t$ , and
- $p$  = Probability that an infected individual will spread the disease to a susceptible individual

## Main Findings

### Application Area

The findings from this problem statement have real world applications. In the current context, Covid spread is a good example. If we know the variant of Reed Frost model that is applicable, and the initial values of  $S$ ,  $I$  and  $R$ , then we can predict the flu spread. Understanding the right model can help in formulating vaccination distribution strategy and policies, which can greatly help in ending the pandemic sooner.

### Models and Methods used

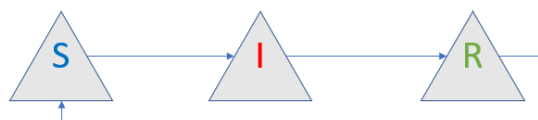
The models we have considered have 3 main states - Susceptible, Infected and Recovered. A Susceptible person can get infected and move to Infected stage. After getting Infected, the person develops either temporary immunity and moves to Recovered stage for some time or develops permanent immunity and stays in the recovered stage. There are also cases, where no immunity is developed like that of cough & cold, flu in which case the person becomes susceptible again immediately post infection.

Various combinations of above are detailed below:

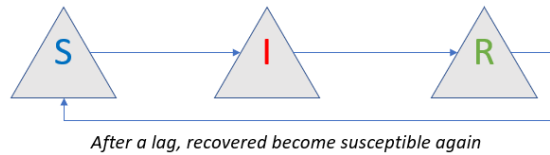
1. **SIR (Susceptible – Infectious – Recovered)** model: This model assumes that once you recover you are not susceptible to infection anymore, i.e., permanent immunity is developed. Another variation is in case of fatal infection where Recovered state is replaced with Deceased.



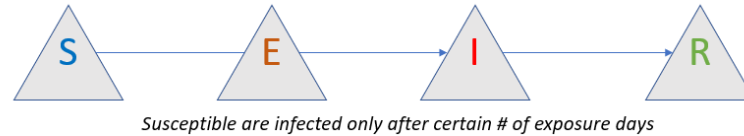
2. **SIS (Susceptible – Infectious – Susceptible)** model: This model assumes no immunity build so an individual is susceptible to infection immediately post recovery, i.e., no immunity is developed.



3. **SIRS (Susceptible – Infectious – Recovered - Susceptible)** model: This model assumes that once you recover you are susceptible to infection but only after the immunity you built wears out i.e., susceptible after some time lag.



4. **SEIR** (Susceptible – Exposed – Infectious – Recovered) model: This model has a significant incubation period for infection i.e., people are infected, but they can't spread infection for some time lag. Rest of transitions are like **SIR** model.



## Data Collection

Data collection was not required for this project.

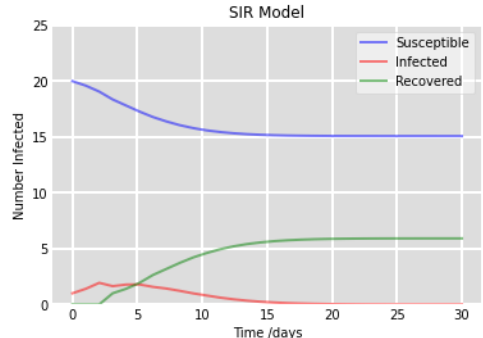
## Approach

1. We applied the basic SIR model on the classroom problem statement and verified that the answers through stochastic approach are very close to the analytical answers.
2. We also implemented SIS, SIRS and SEIR models with varying probabilities on the same problem statement and analysed the differences in results.

## Solving the Problem

The answers via simulation method (SIR) are very close to the analytical answer.

Question	Analytical Method	Simulation
Q1- What is the distribution of the number of kids that Tommy infects on Day 1?	Binomial (20,0.02)	
Q2- What is the expected number of kids that Tommy infects on Day 1?	Expected value for Binomial (20, 0.02) = $20 \times 0.02 = 0.4$	<b>0.4037</b>
Q3 - What is the expected number of kids that are infected by Day 2 (you can count Tommy if you want)?	Number of kids that infected will be following Binomial (20 – 0.4, 1- (1- 0.02) (1+0.4)) or Binomial (19.6, 0.0279). So expected number of kids infected = <b>0.547</b>	<b>0.5442</b>

Q4 - What are the (estimated) expected numbers of kids that are infected by Day 1?	N/A	<p>The expected number of kids infected on Day <math>i</math>, is denoted by red line in below plot. Pandemic is expected to last for <b>15 days</b> under <b>SIR</b> model. (Plot from <math>N = 30000</math>, <math>p = 0.02</math>)</p> 
--	-----	---

## Implementation Details

We have coded the 4 models in python. We could leverage the power of numpy to do faster data manipulation over multiple simulations.

The high-level approach that we have adopted is as follows:

1. Maintain a transition dictionary to track the state of kids on a given day. Details:

State	Description	Possible Transitions
I1, I2, I3	Infected for 3 days i.e., 1,2 and 3	Transition to R for SEIR and SIR models. Transition to S for SIS model. Transition to R1 for SIRS model.
S	Susceptible to infection	Transition E1 for SEIR and I1 for all other models
R	Once Recovered stays recovered. Permanent Immunity.	Stays in R only. Used in SIR and SEIR model
R1, R2, R3	Recovered for 3 days. Loses immunity after 3 days	Transition to S. Used only in SIRS model.

```

Next_State_ModelSIR = dict({"I1": "I2", "I2": "I3", "I3": "R", "R": "R"})
Next_State_ModelSIRS = dict({"I1": "I2", "I2": "I3", "I3": "R1", "R1": "R2", "R2": "R3", "R3": "S"})
Next_State_ModelSIS = dict({"I1": "I2", "I2": "I3", "I3": "S"})
Next_State_ModelSEIR = dict({"E1": "E2", "E2": "I1", "I1": "I2", "I2": "I3", "I3": "R", "R": "R"})

```

2. The status of every kid across different days is tracked in a dataframe named *kids*. A representation snapshot for 10 days for SIR model is as follows:

names	Day 1	Day 2	Day 3	Day 4	Day 5	Day 6	Day 7	Day 8	Day 9	Day 10
A	S	S	S	S	I1	I2	I3	R	R	R
B	S	S	S	I1	I2	I3	R	R	R	R
C	S	S	S	S	S	S	S	S	S	S
D	S	S	S	S	S	S	S	S	S	S
E	S	S	S	S	S	S	S	S	S	S
F	S	S	S	S	S	S	S	S	S	S
G	S	S	S	S	S	S	S	S	S	S
H	S	S	S	S	S	S	I1	I2	I3	R
I	S	S	S	S	S	S	S	S	S	S
J	S	I1	I2	I3	R	R	R	R	R	R
K	S	S	S	I1	I2	I3	R	R	R	R
L	S	S	S	S	S	S	S	S	S	S
M	S	S	S	S	S	S	S	S	S	S
N	S	S	S	S	S	S	S	S	S	S
O	S	S	S	S	S	I1	I2	I3	R	R
P	S	S	S	S	S	S	S	S	S	S
Q	S	S	S	S	S	S	S	S	S	S
R	S	S	S	S	S	S	S	S	S	S
S	S	S	S	S	S	S	S	S	S	S
T	S	S	I1	I2	I3	R	R	R	R	R
Tommy	I1	I2	I3	R	R	R	R	R	R	R

3. Infection spreading is simulated as below:
- The idea used here is that every infected kid has the potential to spread infection to all the susceptible kids. So, we assume on any given day, all infected kids interact with all the susceptible kids. Thus, we generate *#Infected* UNIF (0,1) random number for each susceptible kid. If any of these numbers is less than the infection threshold, then we assume that a susceptible kid got infected. It may happen that multiple infected kids pass infection to same susceptible kid i.e., multiple UNIF (0,1) random number  $\leq$  probability threshold. Hence, we use a minimum function to see if at all infection has been spread.
  - If the count of both infected (I1, I2, I3) and susceptible (S) is greater than 0 on any given day, then we generate a random matrix *random\_mat* with the size (*#Susceptible* \* *#Infected*).

```
#Check if there are kids who can infect or there any susceptible population
if (kids.loc[kids["day" + str(i)].isin(["I1", "I2", "I3"])].shape[0] > 0) & (kids.loc[kids["day" + str(i)]=="S"].shape[0] > 0):

    # Generate a random matrix of size #Susceptible by #Infected
    random_mat = np.random.rand(kids.loc[kids["day" + str(i)]=="S"].shape[0], kids.loc[kids["day" + str(i)].isin(["I1", "I2", "I3"])].shape[0])
```

Example of a random matrix for 10 susceptible and 4 infected kids. Here only, 1 susceptible kid was infected.

	0	1	2	3
0	0.736157	0.661514	0.879729	0.188215
1	0.298239	0.871743	0.750827	0.667874
2	0.984215	0.58334	0.73415	0.865799
3	0.416841	0.791076	0.26005	0.375342
4	0.565803	0.604391	0.377464	0.865244
5	0.637257	0.434624	0.973494	0.792441
6	0.262766	0.756959	0.946317	0.164171
7	0.12413	0.0694789	0.0983329	0.955501
8	0.0745559	0.972868	0.748836	0.510292
9	0.59246	0.509171	0.0106987	0.219409

- c) We then take row-wise minimum to check if any susceptible got infected as explained above. We then take of count of total susceptible kids who got infected in *NumSusceptible2Infected* variable and randomly update next day status of *NumSusceptible2Infected* kids to I1/E1 based on the underlying model.

```
NumSusceptible2Infected = random_mat[random_mat<=p_infect].size

if NumSusceptible2Infected > 0:
    kids.sort_values("day" + str(i),ascending=False, inplace=True)

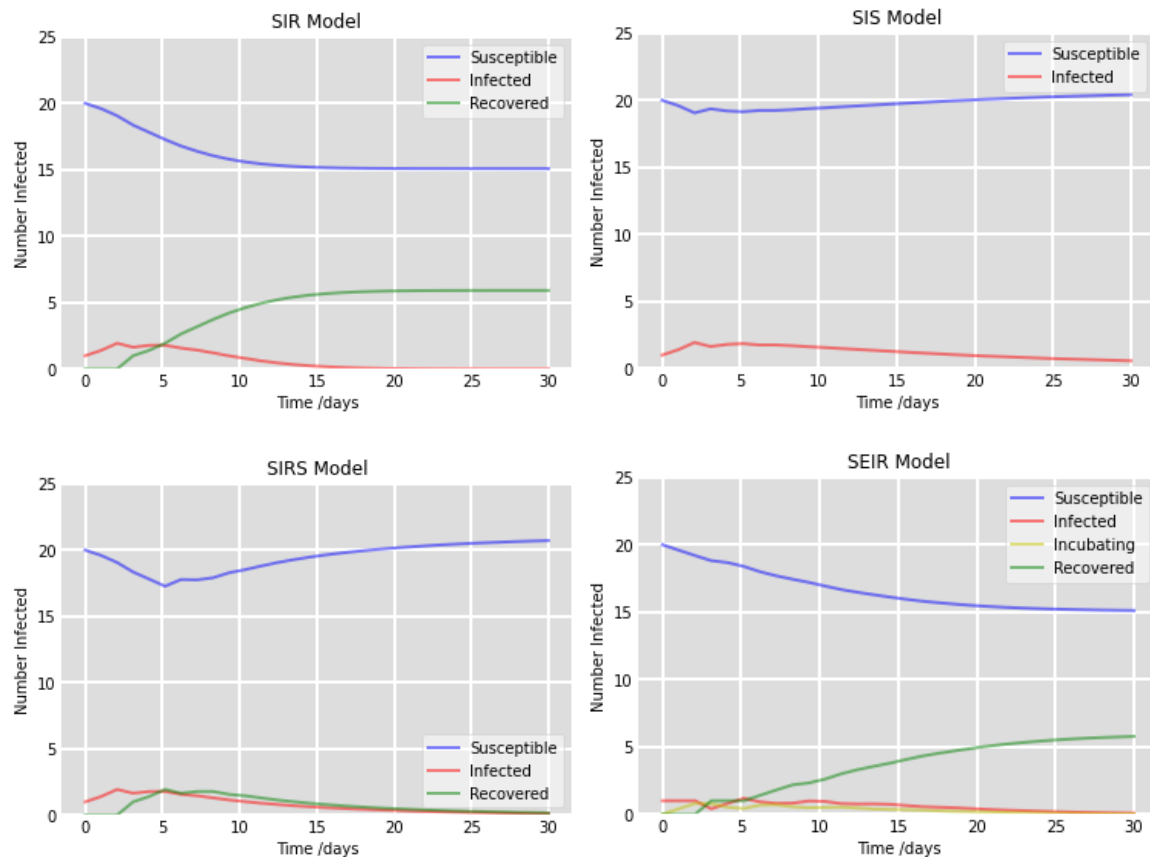
    for j in range(NumSusceptible2Infected):
        if kids.iloc[j,i] == "S":
            kids.iloc[j,i+1] = TransitionFromS
```

4. After simulating infection spread as mentioned above, the kids dataframe is updated based on the transition dictionary. For **each day**, we do the following:
  - a. For susceptible kids who got the infection, we update the status to I1 or E1.
  - b. For susceptible kids who didn't get the infection, we keep the status to S.
  - c. For remaining kids, we update the next day status based on transition matrix discussed above.
5. The above spread simulation is run for 30000 times, to generate an overall view of how infection could potentially spread. We then taken average of number of Susceptible, Exposed, Infected and Recovered kids for each day across the number of simulations and plot the results accordingly for using `plot_graphs` function in the code. Details of the outputs are discussed below.



## Comparative Analysis of different models

**30000** runs with  $p = 0.02$

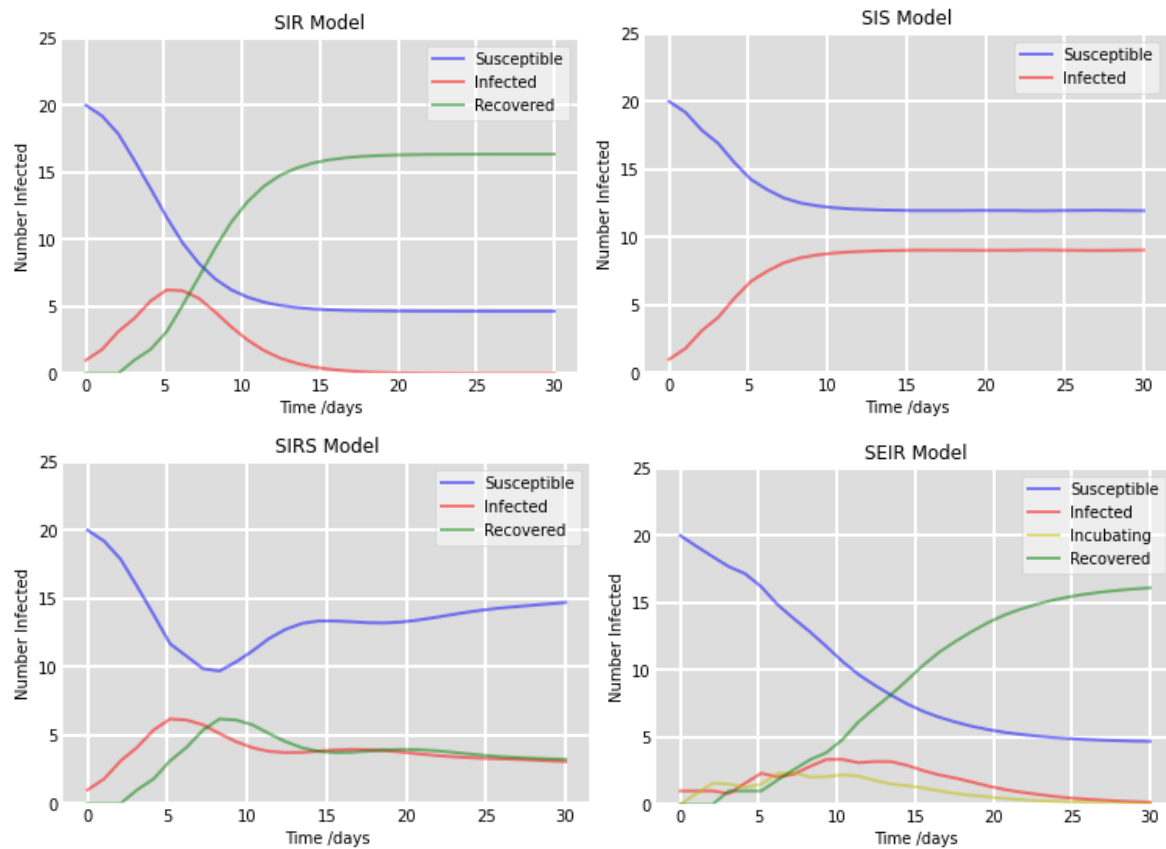


**Observations (30000 runs with  $p = 0.02$ )**

1. For SIR model, no infections are observed after 15 days.
2. SIS lasts the longest ( $> 30$  days) as there is no recovery. Infected people become susceptible again.
3. SIRS has an additional phase after SIR. After recovery, infected people join back in the susceptible group. Hence the epidemic lasts slightly longer than SIR.
4. In SEIR, susceptible people need to have longer exposure for multiple days before getting infected. Hence the number of infections is the lowest. Pandemic ends at around 25 days.

We also, tried another combination of the above 4 models, where we increased the probability of infection to 0.04. Results are detailed below:

**30000 runs with  $p = 0.04$**



Observations (**30000 runs with  $p = 0.04$** )

In general, the infections are peaking and declining at a faster rate.

1. For SIR, the number of infections increase at a faster rate in the initial days, which is expected as  $p$  is twice as compared to previous case. However, the epidemic dies down at roughly 15 days (same as  $p = 0.02$ )
2. For SIS, steady state is reached at around 10 days, when the count of both susceptible and infected stabilizes
3. For SIRS, infections start to come down from day 5, but last longer than 30 days.
4. For SEIR, infections take time to peak due to longer incubation period. This has the lowest number of infections because it takes longer for the infection to spread. Infections come down to 0 at around 27 days.

## Conclusion and Future work

Through this course work, we have gained a good basic understanding of pandemic models. While analyzing the various versions of Reed-Frost model, we saw how different modelling assumptions and factors can significantly change the predicted outcome of the pandemic. This drove home the importance of ensuring a model should represent real-world conditions as closely as possible.

While the Reed-Frost model is one of the original disease propagation models, there are some limitations to it such as:

1. Probability of infection may change during pandemic. In the classroom example, we have assumed a fixed probability of 0.02.
2. Unlike a class where we assumed a fixed number of students, the population of a given area changes during pandemic (due to births and deaths).
3. The model assumes that every person comes into contact with every other person on every day. While this may be a reasonable assumption for a small classroom, it would be unrealistic to make this assumption for a larger population.

In the future, we would like to apply this knowledge to the Covid-19 dataset and explore advanced epidemiological models.

## References

1. Ohio Supercomputer Center, *Reed-Frost Epidemic Model*, <https://www.osc.edu/education/si/projects/epidemic>
2. Lowell Reed and Wade Hampton Frost (Johns Hopkins Science Review), *Epidemic theory: what is it?*, <https://www.youtube.com/watch?v=OR5uzbPajzM>
3. Mark, *The Reed-Frost Model and its Application to Real Life*, [http://www.warwickmaths.com/wp-content/uploads/2020/07/72\\_-The-Reed-Frost-Model-and-its-Applications-to-Real-Life.pdf](http://www.warwickmaths.com/wp-content/uploads/2020/07/72_-The-Reed-Frost-Model-and-its-Applications-to-Real-Life.pdf)
4. Aniruddha A, Devdatt D, Bryan L, Madhav M, Srinivasan V and Anil V, *Mathematical models for Covid-19 Pandemic: A Comparative Analysis*, <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7523122/>

## Appendix

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Prepare the initial dataframe
names = ['Tommy', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T']

##### Parameters #####
days = 30
p = 0.02

Next_State_ModelSIR = dict({"I1": "I2", "I2": "I3", "I3": "R", "R": "R"})
Next_State_ModelSIRS = dict({"I1": "I2", "I2": "I3", "I3": "R1", "R1": "R2", "R2": "R3", "R3": "S"})
Next_State_ModelSIS = dict({"I1": "I2", "I2": "I3", "I3": "S"})
Next_State_ModelSEIR = dict({"E1": "E2", "E2": "I1", "I1": "I2", "I2": "I3", "I3": "R", "R": "R"})

NumSim = 30000
AnswerQs2 = 0
AnswerQs3 = 0
AnswerQs1 = pd.DataFrame({'Probability': np.zeros(21, dtype='float')})
Next_State_Model = Next_State_ModelSIR

##### Answer for Qs 1 to 3 #####

for n in range(NumSim):
    if (n%1000) == 0 : print(n)

    kids = pd.DataFrame({'names': names})
    for i in range(days):
        kids["day" + str(i+1)] = "S"

    kids.loc[kids["names"]=="Tommy", "day1"] = "I1"

    for i in range(1, days):

        #Update status for Infected kids
        kids["day" + str(i+1)] = np.where(kids["day" + str(i)]=="S", "S", kids["day" + str(i)]\
                                          .map(Next_State_Model))

        #Check if there are kids who can infect or there any susceptible population
        if (kids.loc[kids["day" + str(i)].isin(["I1", "I2", "I3"])].shape[0] > 0) & \
            (kids.loc[kids["day" + str(i)]=="S"].shape[0] > 0):
            # Generate a random matrix of size #Susceptible by #Infected
            random_mat = np.random.rand(kids.loc[kids["day" + str(i)]=="S"].shape[0], \
                                         kids.loc[kids["day" + str(i)].isin(["I1", "I2", "I3"])].shape[0])
            #Take row-wise minimum to see if any susceptible got infected
            random_mat = np.min(random_mat, axis = 1)
            NumSusceptible2Infected = random_mat[random_mat<=p].size

            if NumSusceptible2Infected > 0:
                kids.sort_values("day" + str(i), ascending=False, inplace=True)

                for j in range(NumSusceptible2Infected):
                    if kids.iloc[j,i]=="S":
                        kids.iloc[j,i+1] = "I1"

    AnswerQs1.iloc[kids.loc[kids["day2"] != "S"].shape[0]-1,0] += 1
    AnswerQs2 += kids.loc[kids["day2"] == "I1"].shape[0]
    AnswerQs3 += kids.loc[kids["day3"] == "I1"].shape[0]

AnswerQs1 = AnswerQs1/NumSim
AnswerQs2 = AnswerQs2/NumSim
AnswerQs3 = AnswerQs3/NumSim
```

```
##### Answer for Qs 4 #####
#Add code for plots
def plot_graphs(S,E,I,R,DAYS,model_name):

    t = np.linspace(0, DAYS,DAYS)

    fig = plt.figure(facecolor='w')
    ax = fig.add_subplot(111, facecolor='#dddddd', axisbelow=True)

    ax.plot(t, S, 'b', alpha=0.5, lw=2, label='Susceptible')
    ax.plot(t, I, 'r', alpha=0.5, lw=2, label='Infected')
    if E.shape[0] == DAYS:
        ax.plot(t, E, 'y', alpha=0.5, lw=2, label='Incubating')
    if R.shape[0] == DAYS:
        ax.plot(t, R, 'g', alpha=0.5, lw=2, label='Recovered')

    ax.set_xlabel('Time /days')
    ax.set_ylabel('Number Infected')
    ax.set_ylim(0,25)
    ax.yaxis.set_tick_params(length=0)
    ax.xaxis.set_tick_params(length=0)
    ax.grid(b=True, which='major', c='w', lw=2, ls='-')
    legend = ax.legend()
    legend.get_frame().set_alpha(0.5)

    plt.title("{} Model".format(model_name))

    for spine in ('top', 'right', 'bottom', 'left'):
        ax.spines[spine].set_visible(False)
    plt.show()

def RunPandemicSim(days,NumSim,Next_State_Model,names,p_infect,TransitionFromS):
    col = []

    for i in range(days):
        col.append("day" + str(i+1))

    SimResults_S = pd.DataFrame(columns=col)
    SimResults_E = pd.DataFrame(columns=col)
    SimResults_I = pd.DataFrame(columns=col)
    SimResults_R = pd.DataFrame(columns=col)

    for n in range(NumSim):
        if(n%1000) == 0 : print(n)

        kids = pd.DataFrame({'names':names})
        for i in range(days):
            kids["day" + str(i+1)] = "S"
            kids.loc[kids["names"]=="Tommy", "day1"] = "I1"

        for i in range(1,days):

            #Update status for Infected kids
            kids["day" + str(i+1)] = np.where(kids["day" + str(i)]=="S", "S", kids["day" + str(i)] \
                .map(Next_State_Model))

            #Check if there are kids who can infect or there any susceptible population
            if (kids.loc[kids["day" + str(i)].isin(["I1","I2","I3"])].shape[0] > 0) \
                & (kids.loc[kids["day" + str(i)]=="S"].shape[0] > 0):
```

```

# Generate a random matrix of size #Susceptible by #Infected
random_mat = np.random.rand(kids.loc[kids["day" + str(i)]=="S"].shape[0],\
                             kids.loc[kids["day" + str(i)].isin(["I1", "I2", "I3"])]).shape[0])

#Take row-wise minimum to see if any susceptible got infected
random_mat = np.min(random_mat, axis = 1)
NumSusceptible2Infected = random_mat[random_mat<=p_infect].size

if NumSusceptible2Infected > 0:
    kids.sort_values("day" + str(i),ascending=False, inplace=True)

    for j in range(NumSusceptible2Infected):
        if kids.iloc[j,i] == "S":
            kids.iloc[j,i+1] = TransitionFromS

data_to_append_S = {}
data_to_append_E = {}
data_to_append_I = {}
data_to_append_R = {}

arr_S = np.where(kids.iloc[:,1:] == "S",1,0).sum(axis = 0)
arr_E = np.where(kids.iloc[:,1:] == "E1",1,0).sum(axis = 0) + \
        np.where(kids.iloc[:,1:] == "E2",1,0).sum(axis = 0)
arr_I = np.where(kids.iloc[:,1:] == "I1",1,0).sum(axis = 0) + \
        np.where(kids.iloc[:,1:] == "I2",1,0).sum(axis = 0) + \
        np.where(kids.iloc[:,1:] == "I3",1,0).sum(axis = 0)
arr_R = np.where(kids.iloc[:,1:] == "R",1,0).sum(axis = 0) + \
        np.where(kids.iloc[:,1:] == "R1",1,0).sum(axis = 0) + \
        np.where(kids.iloc[:,1:] == "R2",1,0).sum(axis = 0) + \
        np.where(kids.iloc[:,1:] == "R3",1,0).sum(axis = 0)

for i in range(len(SimResults_S.columns)):
    data_to_append_S[SimResults_S.columns[i]] = arr_S[i]
    data_to_append_E[SimResults_E.columns[i]] = arr_E[i]
    data_to_append_I[SimResults_I.columns[i]] = arr_I[i]
    data_to_append_R[SimResults_R.columns[i]] = arr_R[i]

SimResults_S = SimResults_S.append(data_to_append_S, ignore_index = True)
SimResults_E = SimResults_E.append(data_to_append_E, ignore_index = True)
SimResults_I = SimResults_I.append(data_to_append_I, ignore_index = True)
SimResults_R = SimResults_R.append(data_to_append_R, ignore_index = True)

return SimResults_S, SimResults_E, SimResults_I, SimResults_R

```



```

SimResults_SIR_S,_,SimResults_SIR_I,SimResults_SIR_R = RunPandemicSim(30,30000,Next_State_ModelSIR,names,p,"I1")

plot_graphs(SimResults_SIR_S.mean(axis = 0),\
             pd.DataFrame(),SimResults_SIR_I.mean(axis = 0),\
             SimResults_SIR_R.mean(axis = 0),30,"SIR")

SimResults_SIS_S,_,SimResults_SIS_I,_ = RunPandemicSim(30,30000,Next_State_ModelSIS,names,p,"I1")

plot_graphs(SimResults_SIS_S.mean(axis = 0),\
             pd.DataFrame(),SimResults_SIS_I.mean(axis = 0),\
             pd.DataFrame(),30,"SIS")

SimResults_SIRS_S,_,SimResults_SIRS_I,SimResults_SIRS_R = RunPandemicSim(30,30000,Next_State_ModelSIRS,names,p,"I1")
plot_graphs(SimResults_SIRS_S.mean(axis = 0),\
             pd.DataFrame(),SimResults_SIRS_I.mean(axis = 0),\
             SimResults_SIRS_R.mean(axis = 0),30,"SIRS")

SimResults_SEIR_S, SimResults_SEIR_E, SimResults_SEIR_I, SimResults_SEIR_R = RunPandemicSim(30,30000,Next_State_ModelSEIR,names,p,"E1")
plot_graphs(SimResults_SEIR_S.mean(axis = 0), \
             SimResults_SEIR_E.mean(axis = 0), \
             SimResults_SEIR_I.mean(axis = 0), SimResults_SEIR_R.mean(axis = 0),30,"SEIR")

SimResults_SIR_S,_,SimResults_SIR_I,SimResults_SIR_R = RunPandemicSim(30,30000,Next_State_ModelSIR,names,p*2,"I1")
plot_graphs(SimResults_SIR_S.mean(axis = 0),\
             pd.DataFrame(),SimResults_SIR_I.mean(axis = 0),\
             SimResults_SIR_R.mean(axis = 0),30,"SIR")

SimResults_SIS_S,_,SimResults_SIS_I,_ = RunPandemicSim(30,30000,Next_State_ModelSIS,names,p*2,"I1")
plot_graphs(SimResults_SIS_S.mean(axis = 0),\
             pd.DataFrame(),SimResults_SIS_I.mean(axis = 0),\
             pd.DataFrame(),30,"SIS")

SimResults_SIRS_S,_,SimResults_SIRS_I,SimResults_SIRS_R = RunPandemicSim(30,30000,Next_State_ModelSIRS,names,p*2,"I1")
plot_graphs(SimResults_SIRS_S.mean(axis = 0),\
             pd.DataFrame(),SimResults_SIRS_I.mean(axis = 0),\
             SimResults_SIRS_R.mean(axis = 0),30,"SIRS")

SimResults_SEIR_S, SimResults_SEIR_E, SimResults_SEIR_I, SimResults_SEIR_R = RunPandemicSim(30,30000,Next_State_ModelSEIR,names,p*2,"E1")
plot_graphs(SimResults_SEIR_S.mean(axis = 0), \
             SimResults_SEIR_E.mean(axis = 0), \
             SimResults_SEIR_I.mean(axis = 0), SimResults_SEIR_R.mean(axis = 0),30,"SEIR")

```