Pandemic Flu Spread (Q10)

Team MK: Kevin Cooper, Matteo Zullo

Abstract

This project looks at the spread of the flu within a congregate setting given ten initial infection vectors a conditional probability of infection based on a number of factors like masking, social distancing, and vaccination status. We looked at four different scenarios:

- Masking, social distancing, and a two dose vaccine strategy.
- Masking, social distancing, and a one dose vaccine strategy.
- “Lockdown” with masking, and a two dose vaccine strategy.
- “Herd immunity” in which no masking or social distancing is observed along with a one dose strategy.

We found that the utilization of masking and social distancing led to a prolonged but lower intensity pandemic, while the “Herd immunity” strategy led to a sharp increase in cases and deaths that quickly subsided as the flu ran out of individuals to infect. The “Lockdown” strategy was an extreme version of masking and social distancing with relatively few cases and deaths per day but a very prolonged presence of the virus. The major difference between the one and two dose approaches was that it led to slightly fewer cases on the front end and relatively more in the latter stages, but large differences were not seen because the conditional probability of infection with one dose was only slightly higher.

Background and Problem Description

Infectious disease spread is obviously an area of intense interest right now. This problem builds upon our previous work in Mini Project 1 and looks at a more realistic scenario in which the probability of infection is not static and is governed by conditional probabilities associated with a number of factors. The probability of infection for people who encounter an infectious person under various conditions is as follows:

- 0.60 for an unvaccinated, unmasked individual
- 0.30 for an unvaccinated, masked individual
- 0.20 for a person who is 10 days out from their 1st vaccine dose.
- 0.10 for a person who is 10 days out from their 2nd dose.

Additionally, we looked at various social distancing scenarios. We used log-normal distributions and three sets of modes and medians to represent different levels of daily social interactions:

- Default: Median 10, Mode 8
- Limited: Median 5, Mode 4
- Lockdown: Median 2, Mode 1

We began with a population of 10,000 people with 10 infected and simulated a scenario over the course of 50 days and a vaccine introduction after 10 days. Incubation time before an infected person becomes contagious was 2 days. The amount of days that a person was sick was represented as a log-normal with a median of 9 and mode of 7. The probability of death was a constant 0.03. Lastly, the time period between vaccines was 15 days, and the lag between vaccination and immunity -- or partial immunity in the case of the 1st dose -- was 10 days. Our Jupyter notebook in which we built out the simulation and its parameters can be found here.

Stats Note

Importantly, our setup is defined in three pieces and fashions multiple nested code functions to allow for maximum modeling flexibility. Individuals have a daily number of social interactions, each one representing a Bernoulli trial for the probability of meeting an infected individual. Therefore, the chance of a healthy individual encountering a virus carrier each day is given by:

Bin(*x* = 1; *n* = no. social interactions; *P1* = no. infected individuals / total no. of individuals)

Please note that social distancing measures only modulate the *P1* parameter in the above distribution but have no effect on the subsequent conditional probability (say, *P2*) of getting infected upon meeting an infected individual. This probability is given by:

P2(getting infected | meeting a sick individual) = *f*(masking, vaccine)

Hence, *P2* is parametrized by mask-wearing and, more importantly, vaccinations. Our code allows for parametrization of this probability at the individual level which might be expanded out to account for environmental variates (e.g., variants, weather, etc.) as well as individual variates (e.g., immune response capabilities, etc.). Lastly, individuals who get infected have a nonzero chance of dying upon infection (say, *P3*):

P3(dying | getting infected) = *f*(vaccine)

Importantly, neither social distancing nor mask-wearing parametrize *P3* while vaccines have demonstrated in multiple peer-reviewed studies to give very high protection against hospitalization and death for COVID-19, which is so important for public health. Much in the same way that *P2* has the potential to account for more information at the individual and environmental level, P3 might be further parametrized through expert validation.

Code Note

Among the infinite ways to keep the time in a programming language like Python, we decided to define daily updates of a population instantiated through the `initialize_population()` function. The population information is self-contained in a Python dictionary which keeps track of each individual status through time-varying and time-invariant variables. Vaccine rollouts are updated first through the `vaccine_update()` function and then infections are updated through the `update_population()` function. All the parameters can be conveniently set at the beginning of the script along with any policy combinations (including and not limited to severity of containment policies) and probability of infection (including and not limited to vaccine efficacy) might be passed. `simulate_pandemic()` is the main function running the simulation with the chosen parameters. Functions have been thoroughly commented with parameter definitions and code explanations. Again, our main goal was to provide a flexible framework which is able to accommodate the vagaries of disease outbreaks and policy responses keeping the modeling assumptions (e.g., vaccines happening at the beginning of the day) to a minimum.

Main Findings

We found that limited social distancing coupled with masking and a two dose strategy led to a prolonged but low intensity pandemic. Cases and deaths peak at around 2,800 and 80, respectively, on day 8, and the pandemic is largely over by day 13 (Appendix A). The results for limited social distancing with masking and a one dose strategy were generally the same. Cases and deaths peak at around 2,800 and 70, respectively, on day 8, and the pandemic is largely over by day 13. The major difference in the two scenarios is that there is a slight lag in intensity around the day 8 peak in which cases are lower in the leadup but higher after the peak compared to the two dose strategy (Appendix B). This is because with a one dose strategy a greater number people are gaining at least partial immunity at an earlier stage in the pandemic.

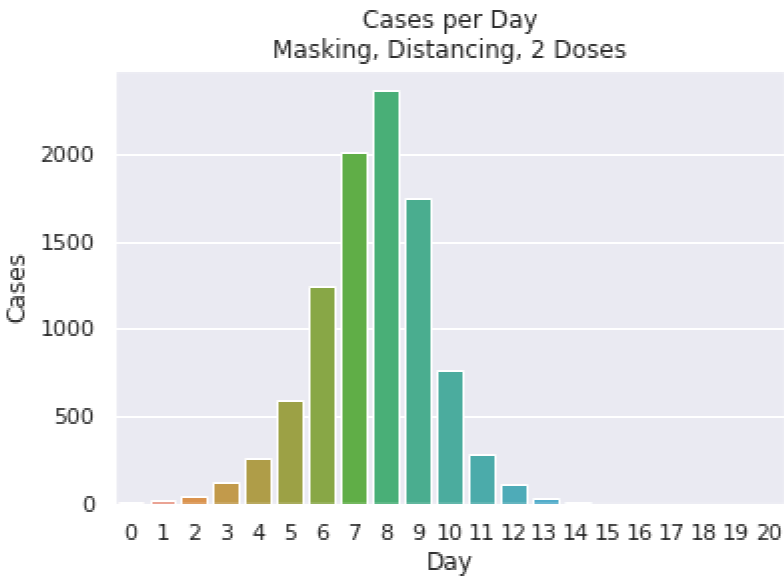
The Lockdown scenario came the closest to “flattening the curve.” Cases and deaths only peaked at 800 and 31, respectively, at around day 16 (Appendix C). The major difference in this scenario was that this doubled the length of the pandemic from ~16 days to ~31 days. Lastly, the “Herd Immunity” scenario in which no masking or social distancing was observed along with a one dose strategy led to the most dramatic results. Cases and deaths peaked at 5,500 and 160, respectively, on day 4, and the pandemic was over by day 6 (Appendix D).

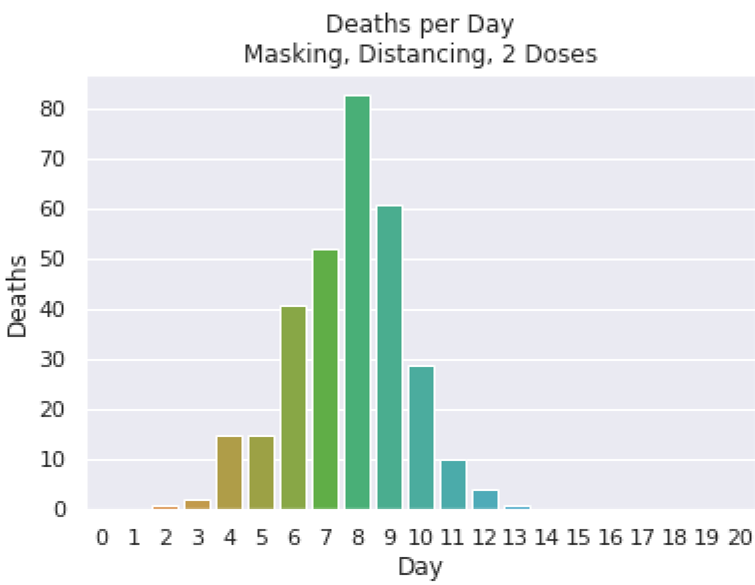
Conclusion

Our project shows the very dramatic differences in case loads, deaths, and duration that can result from different approaches to managing a pandemic. Even modest measures like masking and limited social distancing led to a far lower intensity pandemic. One factor that we didn’t consider that would have been particularly valuable in the “Herd Immunity” scenario was that the extremely high intensity of the pandemic after only four days would very likely have led to the local hospital system being overwhelmed. This in turn would have increased the probability of death from infection. One way we could have accounted for this would have been to include logic accounting for overall case loads. If case loads approached a certain threshold then the probability of death from infection would increase.

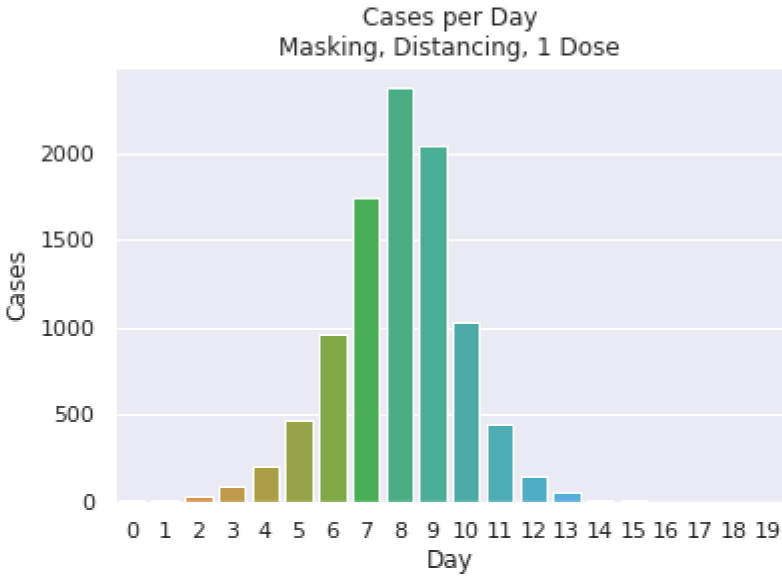
Appendix

Appendix A

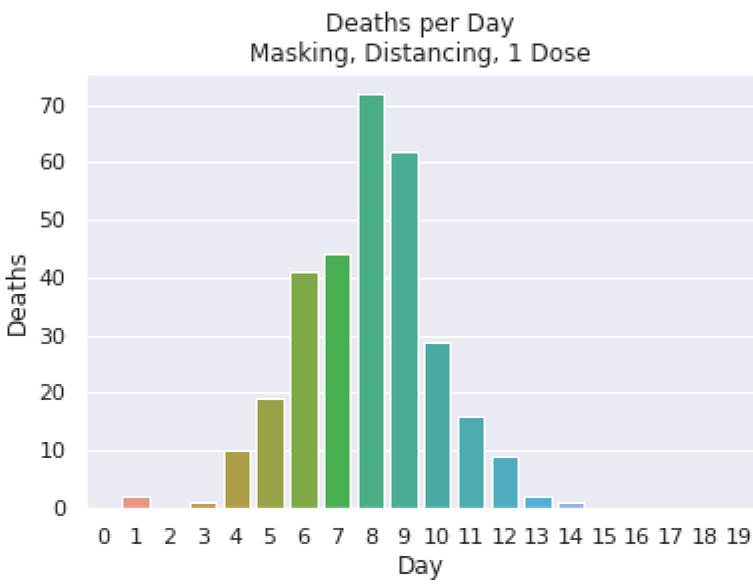




Appendix B

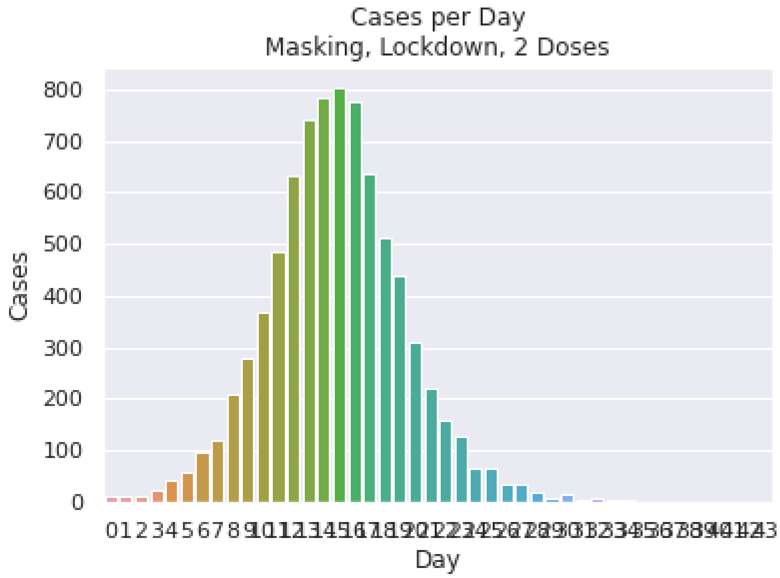


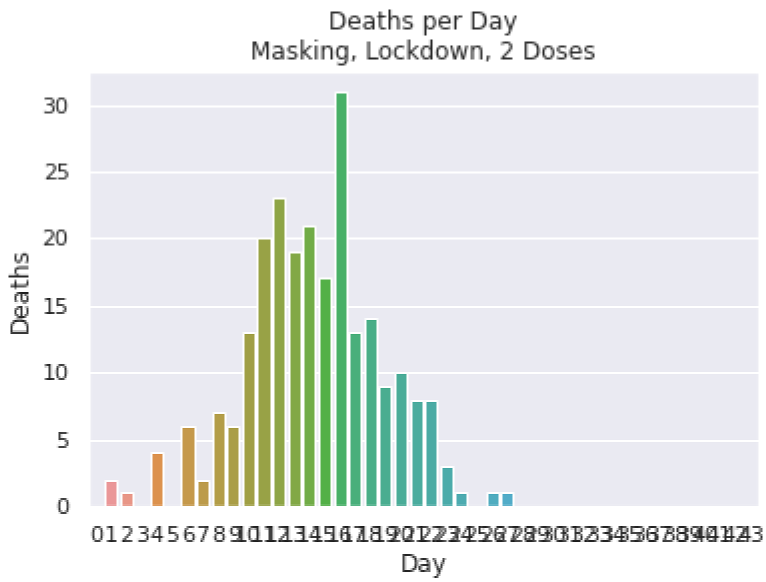
Cases per Day
Masking, Distancing, 1 Dose
Cases
2000
1500
1000
500
0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
Day



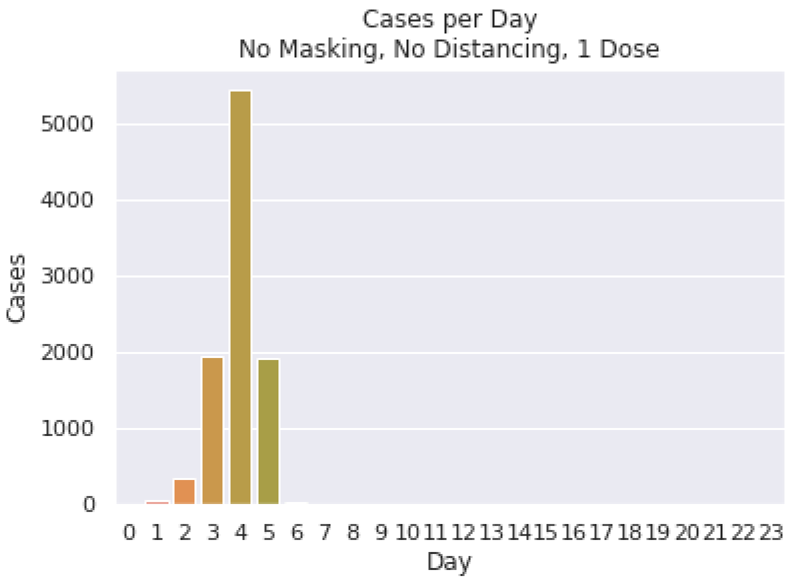
Deaths per Day
Masking, Distancing, 1 Dose
Deaths
70
60
50
40
30
20
10
0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
Day

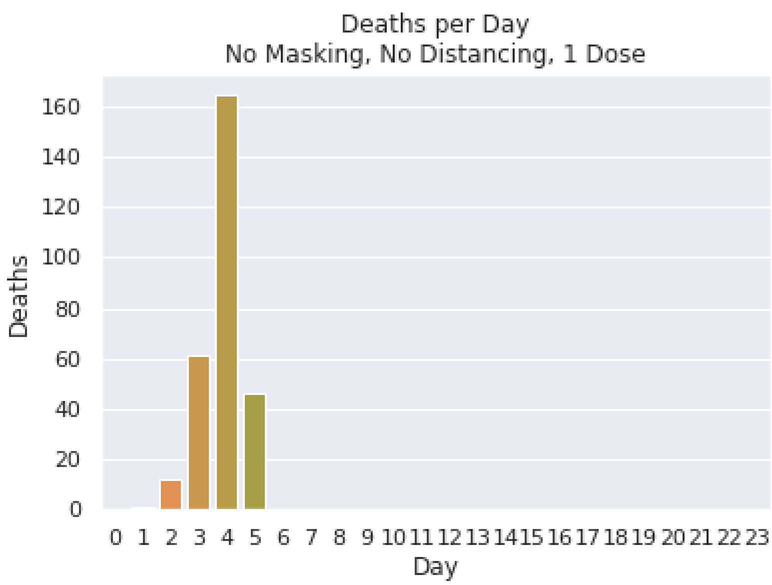
Appendix C


Cases per Day
Masking, Lockdown, 2 Doses
Cases
Day
0
100
200
300
400
500
600
700
800


Deaths per Day
Masking, Lockdown, 2 Doses
Deaths
Day
0
5
10
15
20
25
30

Appendix D





Appendix E

```
import random
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_theme(style="darkgrid")
import numpy as np
from collections import defaultdict
from google.colab import files
import math
```

```
# set parameters
random.seed(249469394)
N_POP = 10000  # no. of population
N_SICK = 10  # no. of sick
DAYS_INCUB = 2 + 1  # days of incubation (add 1 b/c counter starts at 1)
SICK_SPELL = [7,9]  # length of infection (mode and median of lognormal)
SOCIAL = [[8,10], [4,5], [1,2]]  # number of social interactions w/o and
w/ social distancing (mode and median of lognormals)
PROBS = [0.6,0.3,0.2,0.1]  # probabilities of infection when meeting sick
PROB_DEATH = 0.03  # probability of dying when infected
VAX_BETWEEN = 15  # days between 1st and 2nd injection
VAX_LAG = 10  # days between injection and immunity


def logN(mode_med: list) -> int:
 """
 Samples from a lognormal and floors.

 :param mode: mode of lognormal distribution
 :param median: median of lognormal distribution
 :return: floored lognormal(mu, sigma) observation
 """
 mode, median = mode_med
 assert mode < median, "Mode might not be greater than the median. Set the
parameters accordingly."

 mu = math.log(median)  # mu parameter
 sigma = math.sqrt(mu-math.log(mode))  # sigma parameter
 return math.floor(np.random.lognormal(mu, sigma))


def initialize_population(n: int, n_sick: int) -> dict:
 """
 Initializes a population.

 :param n: number of individuals
 :param n__: number of sick individuals
 :return: dictionary with individual features
 """
```

```
 # infection
 id = ['id_' + str(i) for i in range(n)]  # IDs
 sick = [1 for i in range(n_sick)] + [0 for i in range(n-n_sick)]  #
status {'healthy': 0, , 'sick': 1, 'recovered': 2, 'dead': 3}
 sick_counter = [DAYS_INCUB + 1 for i in range(n_sick)] + [0 for i in
range(n-n_sick)]  # days from infection
 dead = [0 for i in range(n)]  # dead dummy
 sick_spell = [logN(SICK_SPELL) for i in range(n_sick)] + [0 for i in
range(n-n_sick)]  # length of infection

 # vaccine
 vax = [0 for i in range(n)]  # vaccine {'no': 0, '1st dose': 1, '2nd
dose': 2}
 vax1_counter = [0 for i in range(n)]  # days from 1st dose
 vax2_counter = [0 for i in range(n)]  # days from 2nd dose

 # policies
 mask = [0 for i in range(n)]  # masking dummy
 socdist = [0 for i in range(n)]  # social distancing dummy

 keys = ['id', 'sick', 'sick_counter', 'sick_spell', 'vax',
'vax1_counter', 'vax2_counter', 'dead', 'mask', 'socdist']
  return {k:v for k,v in locals().items() if k in keys}


def count_sick(sick: list) -> int:
 """
 Counts number of sick individuals.
 """
 return int(len([i for i in sick if i == 1]))


def subset_population(pop: dict):
 """
 Returns lists of indexes for the different health status.
 """

 idx_healthy, idx_incubation, idx_recovered, idx_dead, idx_sick = [], [],
[], [], []
```

```
 # extract features
 ids = [i for i in range(len(pop['id']))]
 statuses = [s for s in pop['sick']]
 counters = [c for c in pop['sick_counter']]

 # subset
 for idx, status, counter in zip(ids, statuses, counters):
   if (status == 0): idx_healthy.append(idx)
   elif (status == 1) and (counter <=DAYS_INCUB):
idx_incubation.append(idx)
   elif (status == 2): idx_recovered.append(idx)
   elif (status == 3): idx_dead.append(idx)
   else: idx_sick.append(idx)

 return idx_healthy, idx_incubation, idx_recovered, idx_dead, idx_sick


def calculate_conditional(pop: dict, idx: int) -> float:
 """
 Returns the conditional probability of an infection given a contact with
a carrier.

 :param pop: dictionary holding population
 :param idx: individual index
 :return: probability of infection
 """

 # extract individual features
 status = [pop['mask'][idx],  # masking
           pop['vax'][idx],  # vaccine dummy
           pop['vax1_counter'][idx] > VAX_LAG,  # more than 10 days from
1st vaccine
           pop['vax2_counter'][idx] > VAX_LAG]  # more than 10 days from
1st vaccine

 # calculate probability
 if status == ([1,0,0,0] or [1,1,0,0]):  p = PROBS[1] # masking
 elif status == ([1,1,1,0] or [0,1,1,0]):  p = PROBS[2] # vaccine 1st dose
 elif status == ([1,2,1,0] or [0,2,1,0]):  p = PROBS[2] # vaccine 2nd dose
(not yet immune)
```

```
    elif status == ([0,2,1,1] or [1,2,1,1]):  p = PROBS[3] # vaccine 2nd dose
    else: p = PROBS[0]
    return p


def day_report(sick__: list, sick: list) -> list:
    """
    Returns a summary of new cases, deaths, and recovered patients.

    :param pop__: population at prior time-period
    :param pop: population at current time-period
    :return: list holding summary stats
    """
    cases, recovered, deaths = 0,0,0

    # for each individual
    for before, after in zip(sick__, sick):
      # if before-after statuses differ
      if before != after:
        if after == 3: deaths += 1
        if after == 2: recovered += 1
        if after == 1: cases += 1
    return [cases, recovered, deaths]


def update_population(pop: dict, masking: int, socdist: int) -> dict:
    """
    Updates the population for the next time-period.

    :param pop: population
    :param p_death: probability of death conditional on infection
    :param masking: mask-wearing dummy
    :param socdist: social distancing dummy
    :return: updated population
    """

    # containment policies
    for idx in range(len(pop['id'])):
      pop['mask'][idx] = 1 if masking else 0
      if socdist == 0:
```

```
    pop['socdist'][idx] = 0
  elif socdist == 1:
    pop['socdist'][idx] = 1
  else:
    pop['socdist'][idx] = 2
  # subset population
 idx_healthy, idx_incubation, idx_recovered, idx_dead, idx_sick =
subset_population(pop)
  # calculate probabilities
 p1 = len(idx_incubation + idx_sick)/len(idx_healthy + idx_recovered +
idx_incubation + idx_sick)  # P(meeting someone sick)
 p3 = PROB_DEATH  # P(death | infection)

 # update healthy

 # for each individual
 for idx in idx_healthy:

   # number of social interactions depends on social distancing
   if pop['socdist'][idx] == 0:
     social = logN(SOCIAL[0])
   elif pop['socdist'][idx] == 1:
     social = logN(SOCIAL[1])
   else:
     social = logN(SOCIAL[2])
   # for each interaction, Bernoulli(p) trials with prob p1 of meeting
someone sick
   s = 0
   while s < social:
     s += 1
     # if meets someone sick
     if random.random() <= p1:
       # if gets infected
       p2 = calculate_conditional(pop, idx)  # P(infection | meeting
someone sick)
       if random.random() <= p2:
         pop['sick'][idx] += 1  # update status to sick
         pop['sick_counter'][idx] += 1  # update counter
         pop['sick_spell'][idx] == logN(SICK_SPELL)  # assign length of
infection
```

```
        if random.random() <= p3: # assign fatality with prob p3
          pop['dead'][idx] == 1
          pop["sick"][idx] += 2
        else:
          0

        # stop Bernoulli trials if gets infected
        break


 # update sick

 # for each individual
 for idx in idx_sick + idx_incubation:
   pop['sick_counter'][idx] += 1

   # if last day of infection
   if pop['sick_counter'][idx] == (pop['sick_spell'][idx] + DAYS_INCUB +
1):
     # recovers
     pop['sick'][idx] += 1
  return pop


def vaccine_update(pop: dict, vax_n: int, vax_strat = 0)  -> dict:
 """
 Updates the vaccine rollout.
 Two rollout strategies are possible, maximizing first dose injections or
second dose injections.

 :param pop: population
 :param vax_n: number of vaccines / day
 :param days_between: days between 1st and 2nd dose
 :param vax_strat strategy dummy (1 if prioritize first dose, else
complete vaccination)
 :return: updated population
 """

 # vaccines require individuals not to be sick
 # 2nd dose eligible
```

```
 idx_elig2 = [idx for idx in range(len(pop['id'])) if
(pop['sick'][idx]==0) and (pop['vax1_counter'][idx]> VAX_BETWEEN)]
 # 1st dose eligible
 idx_elig1 = [idx for idx in range(len(pop['id'])) if
(pop['sick'][idx]==0) and (pop['vax'][idx]==0)]

 # prioritization
 idx_elig = idx_elig2 + idx_elig1 if vax_strat else idx_elig1 + idx_elig2

 # rollouts
 idx_injections = idx_elig[:vax_n]
 for idx in idx_injections:
   pop['vax'][idx] += 1
  # update counters
 # for each individual
 for idx in range(len(pop['id'])):
   # if vaccinated, update counters
   if pop['vax'][idx] > 0:
     if pop['vax'][idx] == 2: pop['vax2_counter'][idx] += 1
     if pop['vax'][idx] == 1: pop['vax1_counter'][idx] += 1
 return pop


def simulate_pandemic(day_stop: int = 365, day_vax: int = 100, vax_rate:
float=0.01, mask: int=1, dist: int=1, vax_strat:int=0):
 """
 Simulates a pandemic and returns daily cases, death tolls, and
recoveries.

 :param day_stop: day simulation ends
 :param day_vax: day of vaccine distribution
 :param vax_rate: percent of population vaccinated / day
 :param mask: whether masking is followed. 1 if yes
 :param dist: whether social distancing is followed. 0 if no, 1 if yes, 2
if lockdown
 :param vax_strat: one or two dose vax strategy. 1 if one dose, 0 if two
dose
 :return: records
 """
```

```python
assert day_stop > day_vax, "Vaccine must be distributed before simulation
ends."

# initialize population
population = initialize_population(N_POP, N_SICK)
n_pop = len(population['id'])
n = count_sick(population['sick'])
 # initialize counters
day = 0
days, cases, recovered, deaths = [0],[n],[0],[0]

# until vaccine
while (day < day_vax) and (n > 0):
  day += 1

  # sick at begin
  sick__ = [s for s in population['sick']]

  # update population
  population = update_population(population, mask, dist)
  sick = [s for s in population['sick']]
  n = count_sick(population['sick'])

  # update records
  c, r, d = day_report(sick__, sick)
  days.append(day)
  cases.append(c)
  recovered.append(r)
  deaths.append(d)

 # after vaccine
vax_n = math.floor(vax_rate * n_pop)  # vaccines per day

# until end or no more sick individuals
while (day < day_stop) and (n > 0):
  day += 1

  # sick at begin
  sick__ = [s for s in population['sick']]
```

```
    # update vaccines
    population = vaccine_update(population, vax_n, vax_strat)

    # update population
    population = update_population(population, mask, dist)
    sick = [s for s in population['sick']]
    n = count_sick(population['sick'])

    # update records
    c, r, d = day_report(sick__, sick)
    days.append(day)
    cases.append(c)
    recovered.append(r)
    deaths.append(d)

  return days, cases, recovered, deaths


def bar_plot(x_vals, x_lab, y_vals, y_lab, title, file_name):
  ax = sns.barplot(x=x_vals, y=y_vals)
  plt.xlabel(x_lab)
  plt.ylabel(y_lab)
  plt.title(title)
  plt.savefig(file_name)
  files.download(file_name)
  return plt.show()


# Two Dose Strategy
days, cases, recovered, deaths = simulate_pandemic(50,10,0.01,1,1,0)

bar_plot(days, "Day", cases, "Cases", "Cases per Day\nMasking, Distancing,
2 Doses", "cases_per_day_mask_dist_two.png")

bar_plot(days, "Day", deaths, "Deaths", "Deaths per Day\nMasking,
Distancing, 2 Doses", "deaths_per_day_mask_dist_two.png")


# One Dose Strategy
days, cases, recovered, deaths = simulate_pandemic(50,10,0.01,1,1,1)
```

```
bar_plot(days, "Day", cases, "Cases", "Cases per Day\nMasking, Distancing,
1 Dose", "cases_per_day_mask_dist_one.png")

bar_plot(days, "Day", deaths, "Deaths", "Deaths per Day\nMasking,
Distancing, 1 Dose", "deaths_per_day_mask_dist_one.png")


# Two Dose Strategy with Lockdown
days, cases, recovered, deaths = simulate_pandemic(50,10,0.01,1,2,0)

bar_plot(days, "Day", cases, "Cases", "Cases per Day\nMasking, Lockdown, 2
Doses", "cases_per_day_lockdown.png")

bar_plot(days, "Day", deaths, "Deaths", "Deaths per Day\nMasking,
Lockdown, 2 Doses", "deaths_per_day_lockdown.png")


# Herd Immunity Strategy
days, cases, recovered, deaths = simulate_pandemic(50,10,0.01,0,0,1)

bar_plot(days, "Day", cases, "Cases", "Cases per Day\nNo Masking, No
Distancing, 1 Dose", "cases_per_day_herd_immunity.png")

bar_plot(days, "Day", deaths, "Deaths", "Deaths per Day\nNo Masking, No
Distancing, 1 Dose", "deaths_per_day_herd_immunity.png")
```