Modelling Influenza in Chicago and Los Angeles During the 2023-2024 Flu Season

A Systems Dynamics Report

Section 001 ME EN 335

Zander Gonzales

December 2025

Abstract

This report presents a system dynamics analysis of influenza transmission in Chicago and Los Angeles during the 2023–2024 flu season using a classic SIR (Susceptible–Infected–Recovered) model. Public health data were scaled for underreporting and used to estimate optimal transmission (β) and recovery (γ) parameters through error minimization. The fitted models accurately reproduced observed infection peaks and were used to assess the effects of vaccination rates and initial infection levels. While effective at capturing overall epidemic trends, the model is limited by assumptions of homogeneous mixing, perfect immunity, and single-strain dynamics.

Contents

List of Figures

1 Introduction

This project models the dynamics of influenza transmission during the 2023–2024 flu season in Chicago and Los Angeles. The goal was to develop a model that closely aligns with real epidemiological data from this period. This would allow us to use our model to predict how the flu would behave in the future, and how modifying key variables, such as vaccination rate, affects the peak number of infections. Accurate outbreak forecasts provide public health practitioners with essential information for resource allocation, intervention planning, and timely communication with the public [8].

The influenza virus exemplifies a dynamic system where the number of susceptible, infected, and recovered individuals change over time based on interactions between people and the characteristics of the disease. This system can be modelled mathematically using the SIR model, which is a set of coupled, first-order differential equations that describe how disease diffuses through populations.

2 Methods

2.1 Modelling Assumptions

To develop a feasible dynamic model, the following assumptions were made:

1. Closed population: Not enough people died from the flu to significantly change our total population. The percentage of deaths due to the flu in the United States is estimated to be 0.0018% [5], making this a realistic assumption.
2. Permanent immunity: Individuals who recovered from influenza or who were vaccinated were considered removed from the susceptible group for the remainder of the season.
3. Homogeneous mixing: Every individual has an equal probability of interacting with every other individual. This simplifies the contact structure and is consistent with the classical SIR model.
4. Underreporting rate: Approximately 20% of influenza cases are reported [2], so we multiplied the reported case numbers by the reciprocal of this proportion to estimate the total number of infections.

2.2 Governing Equations

The influenza dynamics for each city were modeled using the classical SIR system of ordinary differential equations:

$$\frac{dS_t}{dt} = -\frac{\beta I_t S_t}{N} \tag{1}$$

$$\frac{dI_t}{dt} = \frac{\beta I_t S_t}{N} - \gamma I_t \tag{2}$$

$$\frac{dR_t}{dt} = \gamma I_t \tag{3}$$

where:

- S(t) = number of susceptible individuals at time t
- I(t) = number of infectious individuals at time t

- R(t) = number of recovered individuals at time t
- N = total population after subtracting the vaccinated fraction
- β = infection rate, representing how quickly the flu spreads through contact
- γ = recovery rate, equal to $1/T_{infectious}$ where $T_{infectious}$ is the average infectious period

This system is first-order, nonlinear, and coupled, and was solved numerically.

2.3 Data Processing and Parameter Estimation

Public data from the Illinois Department of Public Health (Chicago) and the Los Angeles Department of Public Health were collected. The reported weekly influenza cases were scaled by 1/0.2 to estimate the total number of infections (Figure 1).

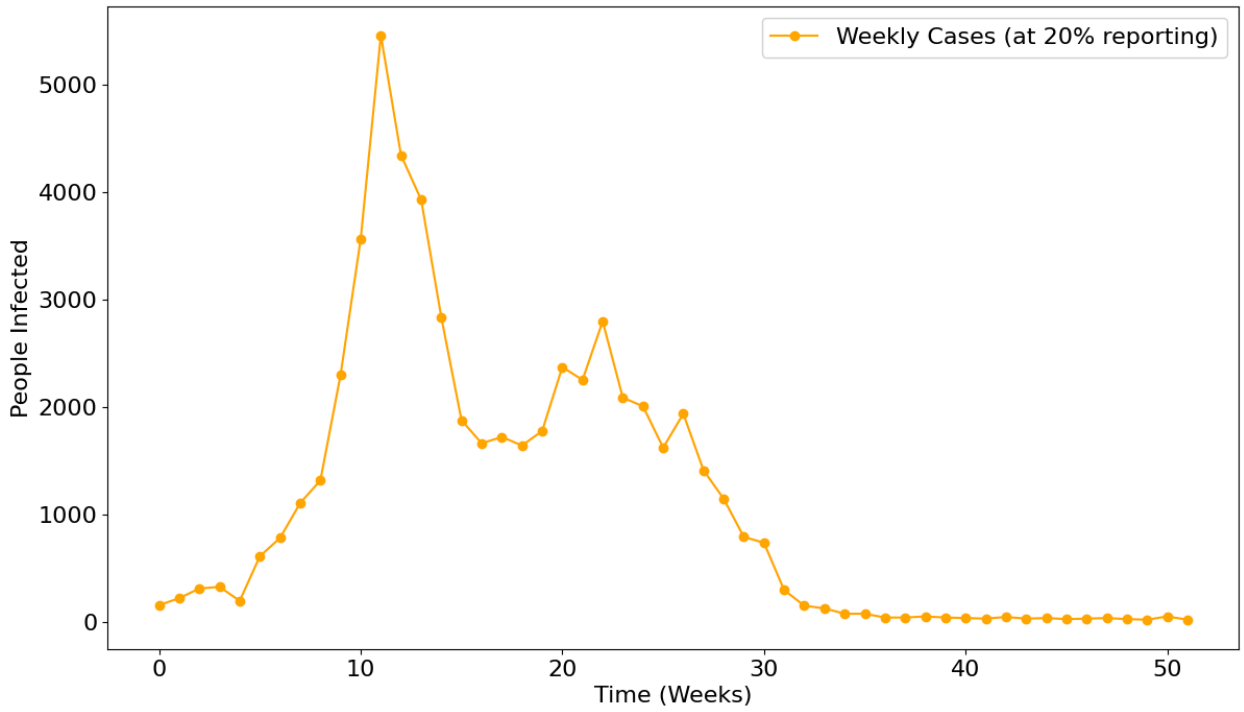

Figure 1. Number of individuals infected with influenza in Chicago from October 2023 to October 2024, assuming 20% of cases are reported to the Illinois Department of Public Health.

The primary unknown parameters were β and γ, the parameters representing infection and recovery rate for this specific strain of influenza. A Python implementation using solve_ivp in SciPy solved the SIR differential equations. To estimate the optimal β and γ parameters, a mean squared error function compared model predictions to actual data. A grid search over arrays of β and γ values, the ranges taken from online research of what common values were, produced a 2D error matrix, visualized using a contour plot (Figure 2). The global minimum in this matrix identified the β and γ values that best fit the observed data.

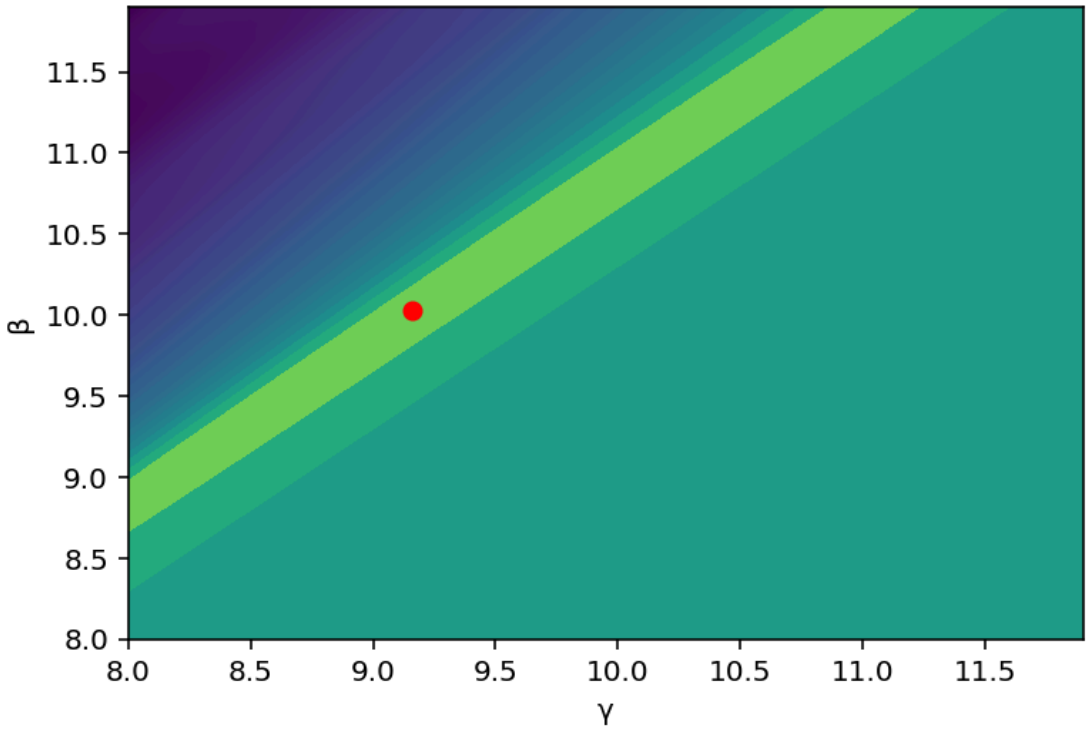

Figure 2. A contour map highlighting combinations of values for β and γ that minimize the prediction error of infected individuals over time, with the red dot indicating the absolute minimum error.

3 Results

As shown in Figure 3, the model was able to closely reproduce the infection data reported by the Illinois Department of Public Health. Similar results were achieved with the Los Angeles dataset, incorporating each city's vaccination rates [4] [6]. The additional curves on the graph illustrate how higher vaccination coverage influences infection dynamics. As modeled in Figure 3, increasing vaccination rates does not change the rate at which the virus spreads, but it does reduce the height of the infection peak by a predictable amount. Furthermore, comparing the blue and purple curves in Figure 3 reveals that although the maximum number of infected individuals remains the same, the peak occurs earlier in the flu season. This indicates that higher early-season infection numbers do not significantly alter the total number of new weekly infections later in the season, but they do shift the timing of the peak forward.

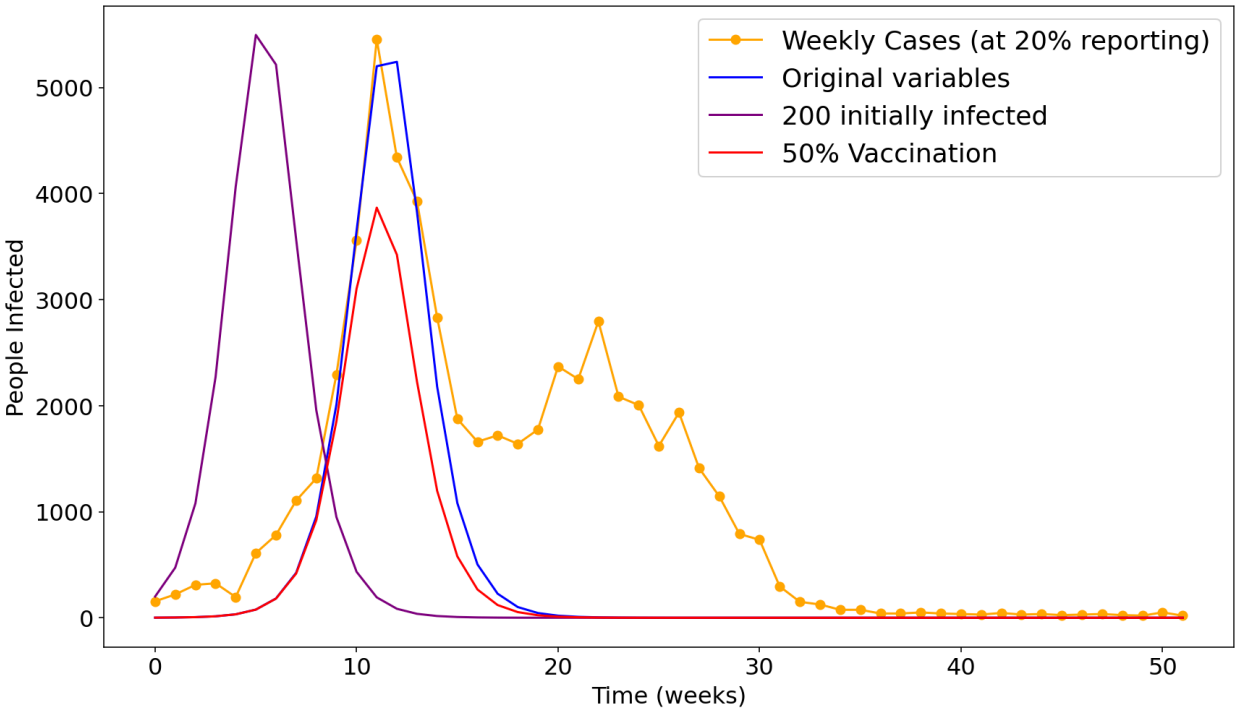

Figure 3. Number of influenza infections in Chicago from October 2023 to October 2024, assuming 20% of cases are reported, compared with model predictions for different vaccination rates and initial infections; the original model used 1 initially infected person and a 30% vaccination rate.

When looking at Figure 3, the two different peaks that occur on the graph of the data given are the result of two different strains of influenza. As a result, it can be seen that the number of infected individuals and the rate of new infections per week will rise and peak at different times based on the difference in the rate of infection between the two strains. Therefore, the model would have to be adjusted based on new data from the first few weeks of each flu season.

4 Discussion

4.1 Interpretation of Model Accuracy

The application of the SIR (Susceptible-Infected-Recovered) model demonstrated a high degree of accuracy in tracking the 2023-2024 influenza season for both Chicago and Los Angeles. As illustrated in the results, the optimized model closely tracks the peak and duration of the infection curves. The effectiveness of the model relied heavily on the optimization of the transmission rate (β) and recovery rate (γ). The use of a Mean Squared Error (MSE) function allowed for the quantification of fit quality, and the generation of a contour map confirmed the existence of a global minimum. This suggests that for a specific strain and location, distinct and constant values for transmissibility and recovery can reasonably approximate the complex dynamics of a flu season.

4.2 Impact of Reporting Rate Assumptions

A critical variable in this study was the assumption that only 20% of actual influenza cases are reported to health officials. By multiplying the recorded data by the reciprocal of this reporting rate, the model was

able to simulate the "true" scale of the epidemic rather than just the clinical surveillance data. This adjustment is validated by the model's output; without scaling the infected population, the interaction between Susceptible and Infected populations would likely have yielded artificially low transmission rates (β), failing to capture the speed at which the virus actually spread through the populace.

4.3 Comparison of Demographics and Strain Types

The model proved robust across two distinct metropolitan areas, Chicago and Los Angeles. Despite differences in population density and climate, the SIR framework adapted well to both datasets when parameters were tuned. However, it is important to note that the 2023-2024 season consisted of a mixture of viral strains, specifically Influenza A(H1N1), Influenza A(H3N2), and Influenza B (Victoria lineage)[2]. Our model aggregates these into a single "Infected" class. While the aggregate fit was strong, the distinct peaks observed in the raw data—and the slight deviations in the model's tail—may be attributed to the different transmission dynamics of these concurrent strains.

4.4 Limitations and Future Work

While the results are promising, several limitations exist in this simulation:

- Population Dynamics: The model assumes a closed system where the total population remains constant, assuming that deaths and births during the flu season were negligible regarding the total population count.
- Vaccination Integration: The graphs incorporated data regarding the percentage of the population vaccinated. However, the standard SIR model treats immunity as binary (Susceptible or Recovered/Removed). A more complex model could account for the varying efficacy of the vaccine against specific strains, which was reported to be roughly 46% effective overall for this season [2].
- Homogeneity: The SIR model assumes homogeneous mixing, meaning any individual has an equal probability of contacting any other. In large cities like Chicago and LA, social clustering significantly affects spread.

Future iterations of this project could implement an SEIR (Susceptible-Exposed-Infected-Recovered) model to account for the incubation period of the virus. Additionally, separating the model into sub-populations (e.g., by age or neighborhood) could provide a more granular view of how the flu moves through a metropolitan area.

5 References

[1] Ben Moussa, Myriam, et al. “National Influenza Annual Report 2023–2024: A Focus on Influenza B and Public Health Implications.” *Canada Communicable Disease Report*, 7 Nov. 2024, pmc.ncbi.nlm.nih.gov/articles/PMC11542548/.

[2] “Influenza Activity in the United States during the 2023–2024 Season and Composition of the 2024–2025 Influenza Vaccine.” *Centers for Disease Control and Prevention*, 25 Sept. 2024, www.cdc.gov/flu/whats-new/flu-summary-2023-2024.html#:~:text=During%20the%202023%E2%80%932024%20season%2C%20influenza%20activity%20began%20to%20increase,2024%20in%20the%20Southern%20Hemisphere.

[3] “Influenza Vaccination Coverage Dashboard.” *Illinois Department of Public Health*, dph.illinois.gov/topics-services/prevention-wellness/immunization/coverage-dashboards/flu-vaccination.html. Accessed 11 Nov. 2025.

[4] Ige, Olusimbo. *Chicago 2023–2024 Influenza Season Surveillance Summary.* Chicago Department of Public Health, 31 May 2024, www.chicago.gov/content/dam/city/depts/cdph/H1N1_swine_flu/FluUpdate/2024/Influenza-Season-Surveillance-Summary-2023-2024-05.31.2024.pdf.

[5] “How CDC Estimates the Burden of Seasonal Flu in the United States.” *Centers for Disease Control and Prevention*, www.cdc.gov/flu-burden/php/about/how-cdc-estimates.html. Accessed 11 Nov. 2025.

[6] “Preliminary Estimated Flu Disease Burden 2023–2024 Flu Season.” *Centers for Disease Control and Prevention*, www.cdc.gov/flu-burden/php/data-vis/2023-2024.html. Accessed 11 Nov. 2025.

[7] “Respiratory Vaccines.” *LA County Flu Vaccine Dashboard*, ph.lacounty.gov/media/FluSeason/vaccine/#/flu. Accessed 11 Nov. 2025.

[8] Osthus, Dave, et al. “Forecasting Seasonal Influenza with a State-Space SIR Model.” *The Annals of Applied Statistics*, Mar. 2017, pmc.ncbi.nlm.nih.gov/articles/PMC5623938/#S16.

6 Appendix A - Chicago Model Code

```
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
import scipy.integrate as scint
import numpy as np

#--------------------------------Constants / Plotting Sample
Data-------------------------------#
vaccination_percent = 0.3
infected_init = 1
percent_reported = 0.20
total_pop = 2_000_000

def plot_influenza_data(file_path, reported_percentage, total_population):
#Plotting Sample Data
   """
   Reads the influenza surveillance CSV and plots an *estimated* number of
   susceptible, infected, and recovered people over time.
   """
   # Load the dataset from the specified CSV file
   df = pd.read_csv(file_path)

   # --- Data Preparation ---
   df['WEEK_START'] = pd.to_datetime(df['WEEK_START'])
   df = df.sort_values(by='WEEK_START')

   # --- Apply Case Multiplier ---
   if not (0 < reported_percentage <= 1.0):
       print(f"Error: reported_percentage must be between 0 (exclusive) and 1.0
(inclusive).")
       return

   case_multiplier = 1.0 / reported_percentage
   df['ESTIMATED_TRUE_CASES'] = df['LAB_FLU_POSITIVE'] * case_multiplier

   # --- Calculate Recovered & Infected totals ---
   df['CUMULATIVE_INFECTED'] = df['ESTIMATED_TRUE_CASES'].cumsum()
   df['CUMULATIVE_RECOVERED'] = df['CUMULATIVE_INFECTED'].shift(1).fillna(0)

   # --- NEW: Calculate Susceptible Population ---
```

```
   # This is the total population minus everyone who has ever been infected.
   df['SUSCEPTIBLE'] = total_population - df['CUMULATIVE_INFECTED']

   # --- Plotting ---
   plt.figure(1, figsize=(14, 8))

   # # Plot the weekly positive cases ("infected")
   # plt.plot(df['WEEK_START'], df['ESTIMATED_TRUE_CASES'],
   #              label=f'Weekly Cases (at {reported_percentage * 100:.0f}%
reporting)',
   #              marker='o',
   #              linestyle='-',
   #              color='blue')

   # Plot the weekly positive cases ("infected")
   plt.plot(np.linspace(0, 51, 52), df['ESTIMATED_TRUE_CASES'],
                label=f'Weekly Cases (at {reported_percentage * 100:.0f}%
reporting)',
                marker='o',
                linestyle='-',
                color='orange')

   #     # Plot the cumulative recovered cases
   # plt.plot(df['WEEK_START'], df['CUMULATIVE_RECOVERED'],
   #          label='Cumulative Recovered',
   #          linestyle='--',
   #          color='green')

   # # --- NEW: Plot the Susceptible population ---
   # plt.plot(df['WEEK_START'], df['SUSCEPTIBLE'],
   #          label='Susceptible Population',
   #          linestyle=':',
   #          color='orange',
   #          linewidth=2.5)  # Make it a bit thicker to stand out

   # --- Formatting the Plot ---

   # --- UPDATED: Title to reflect all data ---
   # title_text = f'Influenza Model (Pop: {total_population:,.0f}, Reporting:
{reported_percentage * 100:.0f}%)'
   # plt.title(title_text, fontsize=16)
```

```
   # plt.xlabel('Date (Week Start)', fontsize=12)
   # plt.ylabel('Number of People', fontsize=12)

   # Add a legend to identify the lines
   # plt.legend()

   # ax = plt.gca()
   # ax.xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m-%d'))
   # ax.xaxis.set_major_locator(mdates.MonthLocator())

   # plt.gcf().autofmt_xdate()
   # plt.tight_layout()

   # Display the plot
   # print(f"Displaying plot for a total population of
{total_population:,.0f}...")
   #plt.show #Comment out to put both data and model on the same graph

   x = 1 #This just makes sure the commented out sections fold into the
function properly
for i in [1]: #Constants
   '''This for loop allows a foldable section for all of the constants'''
   csv_file = 'Influenza_Surveillance_Weekly_2023-2024-_Historical.csv'

   # --- Adjustable Variables ---

   # 1. Percentage of cases you assume are reported (e.g., 0.20 = 20%)
   PERCENT_REPORTED = percent_reported

   # 2. Total population for the simulation
   #    (Using underscores for readability, e.g., 3_000_000 for 3 million)
   TOTAL_POPULATION = total_pop
   TOTAL_POPULATION = TOTAL_POPULATION - (vaccination_percent*TOTAL_POPULATION)
#Original is 0.3

   # Pass the variables into the function
   plot_influenza_data(csv_file, PERCENT_REPORTED, TOTAL_POPULATION)

   #Read in the collected data and extract the infected numbers
   df = pd.read_csv(csv_file)
   case_multiplier = 1.0 / PERCENT_REPORTED #Multiplies up reported numbers to
get an estimate of total infected
```

```
    data_cases = df['LAB_FLU_POSITIVE'] * case_multiplier

    data_cases_16 = data_cases[0:16] #Useful for fitting only to the first curve

#----------------------------------Create Model for
Influenza---------------------------------#
def infect_model(t, y, beta, gamma):
    '''
    Uses a SIR model to find the susceptible, infected, and recovered numbers
over a given time
    t = Time span over which to find the data
    y = Initial conditions of [S0, I0, R0]
    beta = People / week infected
    gamma = People / week recovered
    '''

    #Create equations
    S, I, R = y
    dS = -beta * S * I / TOTAL_POPULATION
    dI = (beta * S * I / TOTAL_POPULATION) - (gamma * I)
    dR = gamma * I

    return dS, dI, dR

#----------------------------Find Ideal Beta / Gamma Values for
Model-------------------------#
def calc_error(model, data):
    '''Finds the mean-squared error between two arrays that are input'''
    mse = np.mean((model - data)**2) #Compute mean-squared error
    return mse

def get_errors():
    '''Go through and create an error value for every combination of beta and
gamma'''
    #Define Parameters for running the ODE
    init_cond = [TOTAL_POPULATION, infected_init, 0] #Susceptible, Infected,
Recovered (S,I,R)
    t_span = [0,51]
    t_points = np.linspace(t_span[0], t_span[1], 52)

    #Provide ranges of beta and gamma over which to calculate the errors
    beta_array = np.arange(8, 12, .1) #Guesses for rate of spread
```

```
    gamma_array = np.arange(8, 12, .1) #Guesses for rate of recovery

    #Instantiate an empty list for the error values
    error_array = np.zeros((len(beta_array),len(gamma_array)))

    for i in range(len(beta_array)): #Go through every value of gamma for each
value of beta
        for j in range(len(gamma_array)):
            sol = scint.solve_ivp(infect_model, t_span, init_cond, t_eval =
t_points, args = ((beta_array[i], gamma_array[j]))) #Solves with given combo of
values
            model_cases = sol.y[1] #Extracts the number of infected from the
solution
            error = calc_error(model_cases, data_cases) #Find error between
model and data at the given combo of variables
            error_array[i,j] = error #Add the error value in the error matrix,
indexed at the variables

    return beta_array, gamma_array, error_array

def error_map(betas, gammas, errors):
    '''Creates a contour map mapping all of the error values from the matrix
returned by get_errors.
    Meant to be run through running ideal_variables'''
    plt.figure(2)
    plt.contourf(betas, gammas, errors, levels=120, norm='log',
vmin=np.min(errors), vmax=np.max(errors), cmap='viridis_r') #Create contour map

    plt.ylabel('β')
    plt.xlabel('γ')
    plt.scatter([9.16],[10.03], marker='o', color='red') #Plot the ideal beta
and gamma previously found on the contour map
    plt.show()

def ideal_variables():
    '''Runs the get_errors and error_map functions and prints out the ideal
beta/gamma and their associated error'''
    beta_m, gamma_m, error_m = get_errors() #Get the beta, gamma, and error
values from get_errors
    error_map(beta_m, gamma_m, error_m) #Map the beta, gamma, and error values
on a contour map through error_map
```

```
   min_i, min_j = np.unravel_index(np.argmin(error_m), error_m.shape) #Extract
the indices of the minimum error value
   print(f"({round(beta_m[min_i],3)} {round(gamma_m[min_j],3)}),
{round(np.min(error_m))}") #Print the ideal beta, gamma, and error

#Uncomment to find the ideal variables for the model, and plot error values on
a contour map
# ideal_variables()

#---------------------------------Solve / Plot Model with
Guesses-----------------------------#
def solve_model(beta = 10.03, gamma = 9.16):
   '''Solve the influenza model with a set guess for beta and gamma, and then
graph the result'''
   #Define Parameters
   init_cond = [TOTAL_POPULATION, infected_init, 0] #Susceptible, Infected,
Recovered (S,I,R)
   t_span = [0,51]
   t_points = np.linspace(t_span[0], t_span[1], 52)

   #Solve using solve_ivp
   sol = scint.solve_ivp(infect_model, t_span, init_cond, t_eval = t_points,
args = (beta, gamma))

   #Plot results
   plt.figure(1)
   plt.plot(sol.t,sol.y[1],label = 'Infected Model', color='blue')
   # plt.plot(sol.t,sol.y[0],label = 'Susceptible', color='blue' ) #Uncomment
these to graph susceptible and recovered
   # plt.plot(sol.t,sol.y[2],label = 'Recovered', color='orange',
linestyle='--' )

   plt.xlabel('Time (weeks)')
   plt.ylabel('People')
   plt.legend(fontsize = 18)
   plt.grid()
   plt.show()

   return sol


#Comment out to not graph the data or the model with the variable guesses
```

```
model_sol = solve_model()
test_error = calc_error(model_sol.y[1], data_cases)

#Past beta/gamma combos and error values
#10.03, 9.16 = 222_273, actual pop, etc
```

7 Appendix B - Los Angeles Model Code

```
    # --- Adjustable Variables ---

    # 1. Percentage of cases you assume are reported (e.g., 0.20 = 20%)
    PERCENT_REPORTED = percent_reported

    # 2. Total population for the simulation
    #    (Using underscores for readability, e.g., 3_000_000 for 3 million)
    TOTAL_POPULATION = total_pop
    TOTAL_POPULATION = TOTAL_POPULATION - (vaccination_percent*TOTAL_POPULATION)
#Original is 0.3

    # Pass the variables into the function
    plot_influenza_data(csv_file, PERCENT_REPORTED, TOTAL_POPULATION)

    #Read in the collected data and extract the infected numbers
    df = pd.read_csv(csv_file)
    case_multiplier = 1.0 / PERCENT_REPORTED #Multiplies up reported numbers to
get an estimate of total infected
    df["LAB_FLU_POSITIVE"] =
df["LAB_FLU_POSITIVE"].iloc[::-1].reset_index(drop=True) #Reverse the order of
the column entries, necessary because it is given in reverse chronological
order
    data_cases = df['LAB_FLU_POSITIVE'] * case_multiplier

    data_cases_16 = data_cases[0:16] #Useful for fitting only to the first curve

#---------------------------------Create Model for
Influenza--------------------------------#
def infect_model(t, y, beta, gamma):
    '''
    Uses a SIR model to find the susceptible, infected, and recovered numbers
over a given time
    t = Time span over which to find the data
    y = Initial conditions of [S0, I0, R0]
    beta = People / week infected
    gamma = People / week recovered
    '''

    #Create equations
    S, I, R = y
```

```
    dS = -beta * S * I / TOTAL_POPULATION
    dI = (beta * S * I / TOTAL_POPULATION) - (gamma * I)
    dR = gamma * I

    return dS, dI, dR

#----------------------------Find Ideal Beta / Gamma Values for
Model-------------------------#
def calc_error(model, data):
    '''Finds the mean-squared error between two arrays that are input'''
    mse = np.mean((model - data)**2) #Compute mean-squared error
    return mse

def get_errors():
    '''Go through and create an error value for every combination of beta and
gamma'''
    #Define Parameters for running the ODE
    init_cond = [TOTAL_POPULATION, infected_init, 0] #Susceptible, Infected,
Recovered (S,I,R)
    t_span = [0,55]
    t_points = np.linspace(t_span[0], t_span[1], 56)

    #Provide ranges of beta and gamma over which to calculate the errors
    beta_array = np.arange(5, 9, .1) #Guesses for rate of spread
    gamma_array = np.arange(4, 8, .1) #Guesses for rate of recovery

    #Instantiate an empty list for the error values
    error_array = np.zeros((len(beta_array),len(gamma_array)))

    for i in range(len(beta_array)): #Go through every value of gamma for each
value of beta
        for j in range(len(gamma_array)):
            sol = scint.solve_ivp(infect_model, t_span, init_cond, t_eval =
t_points, args = ((beta_array[i], gamma_array[j]))) #Solves with given combo of
values
            model_cases = sol.y[1] #Extracts the number of infected from the
solution
            error = calc_error(model_cases, data_cases) #Find error between
model and data at the given combo of variables
            error_array[i,j] = error #Add the error value in the error matrix,
indexed at the variables
```

```
    return beta_array, gamma_array, error_array

def error_map(betas, gammas, errors):
    '''Creates a contour map mapping all of the error values from the matrix
returned by get_errors.
    Meant to be run through running ideal_variables'''
    plt.figure(2)
    plt.contourf(betas, gammas, errors, levels=120, norm='log',
vmin=np.min(errors), vmax=np.max(errors), cmap='viridis_r') #Create contour map

    plt.xlabel('β')
    plt.ylabel('γ')
    plt.scatter([6.8],[5.8], marker='o', color='red') #Plot the ideal beta and
gamma previously found on the contour map
    plt.show()

def ideal_variables():
    '''Runs the get_errors and error_map functions and prints out the ideal
beta/gamma and their associated error'''
    beta_m, gamma_m, error_m = get_errors() #Get the beta, gamma, and error
values from get_errors
    error_map(beta_m, gamma_m, error_m) #Map the beta, gamma, and error values
on a contour map through error_map
    min_i, min_j = np.unravel_index(np.argmin(error_m), error_m.shape) #Extract
the indices of the minimum error value
    print(f"({round(beta_m[min_i],3)} {round(gamma_m[min_j],3)}),
{round(np.min(error_m))}") #Print the ideal beta, gamma, and error

#Uncomment to find the ideal variables for the model, and plot error values on
a contour map
# ideal_variables()

#----------------------------------Solve / Plot Model with
Guesses-----------------------------#
def solve_model(beta = 6.8, gamma = 5.8):
    '''Solve the influenza model with a set guess for beta and gamma, and then
graph the result'''
    #Define Parameters
    init_cond = [TOTAL_POPULATION, infected_init, 0] #Susceptible, Infected,
Recovered (S,I,R)
    t_span = [0,55]
    t_points = np.linspace(t_span[0], t_span[1], 56)
```

```
    #Solve using solve_ivp
    sol = scint.solve_ivp(infect_model, t_span, init_cond, t_eval = t_points,
args = (beta, gamma))

    #Plot results
    plt.figure(1)
    plt.plot(sol.t,sol.y[1],label = 'Infected Model', color='blue')
    # plt.plot(sol.t,sol.y[0],label = 'Susceptible', color='blue' ) #Uncomment
these to graph susceptible and recovered
    # plt.plot(sol.t,sol.y[2],label = 'Recovered', color='orange',
linestyle='--' )

    plt.xlabel('Time (weeks)')
    plt.ylabel('People')
    plt.legend(fontsize = 18)
    plt.grid()
    plt.show()

    return sol

#Comment out to not graph the data or the model with the variable guesses
model_sol = solve_model()
test_error = calc_error(model_sol.y[1], data_cases)

# Past beta/gamma combos and error values
#6.8, 5.8 = 31_373_698, half of time, fixed reverse order
```