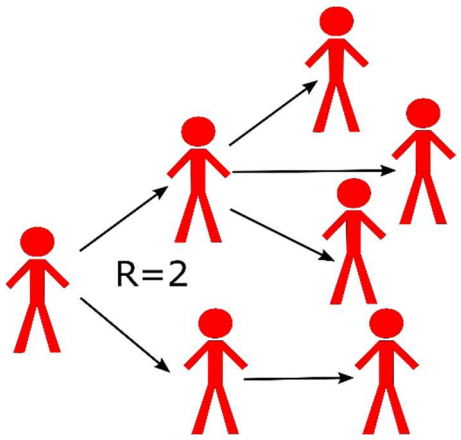
## Introduction

"The deeper understanding Faust sought,
Could not from the Devil be bought,
But now we are told,
By theorists bold,
All we need to know is R0"

*Sir Robert May, Australian scientist*

Since the start of the novel coronavirus pandemic, a popular summary statistic used to describe the progress of the pandemic is the reproduction number R. R usually comes in two forms: the basic reproduction number ($R_0$, pronounced "R Naught") and the effective reproduction number ($R_{eff}$). $R_0$ is defined as the average number of new infections caused by an infectious individual in a wholly susceptible population. $R_{eff}$ is defined as the average number of new infections caused by an infectious individual in a population consisting of both susceptible and non-susceptible hosts. (Wuh et al, 2023). The primary appeal of R is that it offers a single number that indicates whether the transmission of the pathogen is increasing or decreasing, depending on whether R is above or below one. (Vegvari et a., 2021). Below is a schematic showing the definition of the reproductive number. (Handel, 2021). Each person `produces' *on average* 2 other infectious individuals.




The value of R depends on the duration of the infectious period, the probability of infecting a susceptible individual during a direct or indirect contact, and the number of new susceptible individuals contacted per unit of time. (Dietz, 1993). Thus, the value R for Covid-19 is based not only on the nature of the virus but also on people's contact behavior. For respiratory viruses such as the influenza virus, there is ample evidence that the value of R is heavily affected by season changes. [Ali et al., 2022] Epidemiological studies show that low temperature and/or humidity improve the stability of influenza virus [Polozov] and impair the human innate immune system [kudo].

Some studies suggest that there is also a possible association between temperature and COVID-19 reproduction number, at least for the earliest part of the pandemic [landier et al, 2021] Laboratory research also indicates that similar to the influenza virus, the structure of SARS-CoV-2, the virus that causes Covid-19, is more stable at lower temperatures. For example, one study

found that the virus remained highly stable at 4°C (39.2°F), with no significant reduction in infectivity even after 14 days, suggesting that cold temperatures can preserve the virus's infectivity for extended periods. [Chin et al., 2020]. Other studies, however, suggest a weak association between temperature and COVID-19 reproduction number. (Yu, 2020). Because the value of R is critical for understanding the contagiousness of the virus and the progression of disease transmission, any seasonal variation in this number could have significant implications for public health.

This experiment tests whether seasonal differences after 2020, when vaccines are available and people are no longer staying at home, affect the values of $R_{eff}$ for COVID-19 across regions in the DC Metropolitan area. The experiment uses mathematical modeling and nonlinear optimization methods, thereby offering a data-driven approach to identifying when the COVID-19 virus may become more dangerous. If it is demonstrated that R for COVID-19 changes due to season, local governments and health organizations can use this insight to optimize resource allocation, prepare hospitals, and make timely policy decisions. Additionally, it could empower local communities and individuals to be more aware of potential seasonal risks, guiding their behaviors to reduce transmission.

Overall, this experiment bridges the gap between theoretical mathematical models and practical application, providing a tool for more effective disease management in real-world scenarios.

## Question

In recent years, do colder weather conditions during the fall to winter months in the DC Metropolitan area cause changes to the effective reproduction number ($R_{eff}$) of the COVID-19 virus?

## Hypothesis

*If* $R_{eff}$ for COVID-19 is estimated for various regions in the DC Metropolitan area for the warmer six months from May through October, for the years 2021-2024 (after the beginning of the COVID-19 pandemic when vaccines are available), using a compartmental mathematical model and nonlinear regression methods, *and if* the reproduction number is also estimated for the colder six months from November through April of the same years in the same regions using the same methods, *then* there will be an increase in the values of $R_{eff}$ between the warmer season and the subsequent colder season, for each of the years and for each of the regions.

## Materials: Machine Requirements and Instructions for Downloading Requisite Software and Data Science Packages:

1. *Machine Requirements*: A modern laptop capable of running Jupyter Notebook comfortably. Jupyter Notebook is an interactive web-based interface where you can write, run, and visualize Python code in real-time, making it popular for data science and machine learning. The laptop should have at least a mid-range processor like an Intel Core i5 or AMD Ryzen 5, which ensures sufficient performance for data analysis,

scripting, and coding tasks. The laptop should also have a minimum of 8 GB of RAM, though 16 GB is recommended for smoother multitasking due to the complexity of the algorithms involved in this project (constrained optimizations and nonlinear problems). The laptop should come with a solid-state drive (SSD) with at least 256 GB of storage to assure faster load times and overall system responsiveness, which is essential for handling data files efficiently. Jupyter Notebook is compatible with Windows, macOS, and Linux operating systems, so the choice of OS is flexible.

2. *Software, Programming Language and Data Science Packages*: Jupyter Notebook itself does not come with Python pre-installed. One convenient way to install both Python and Jupyter together is through the Anaconda distribution, an open-source distribution that comes bundled with Python, Jupyter Notebook, and other data science packages.

   To install both Jupyter Notebook and Python together through Anaconda, first download and install the Anaconda distribution from https://www.anaconda.com. Once Anaconda is installed, open the Anaconda Navigator or use the command "jupyter notebook" in the terminal or command prompt to launch Jupyter Notebook. This approach simplifies setup, as everything needed for Python programming and data analysis is packaged together.

## Experiment Procedure:

### Step 1: Gather COVID-19 Infection Data for Montgomery, PG and Howard Counties

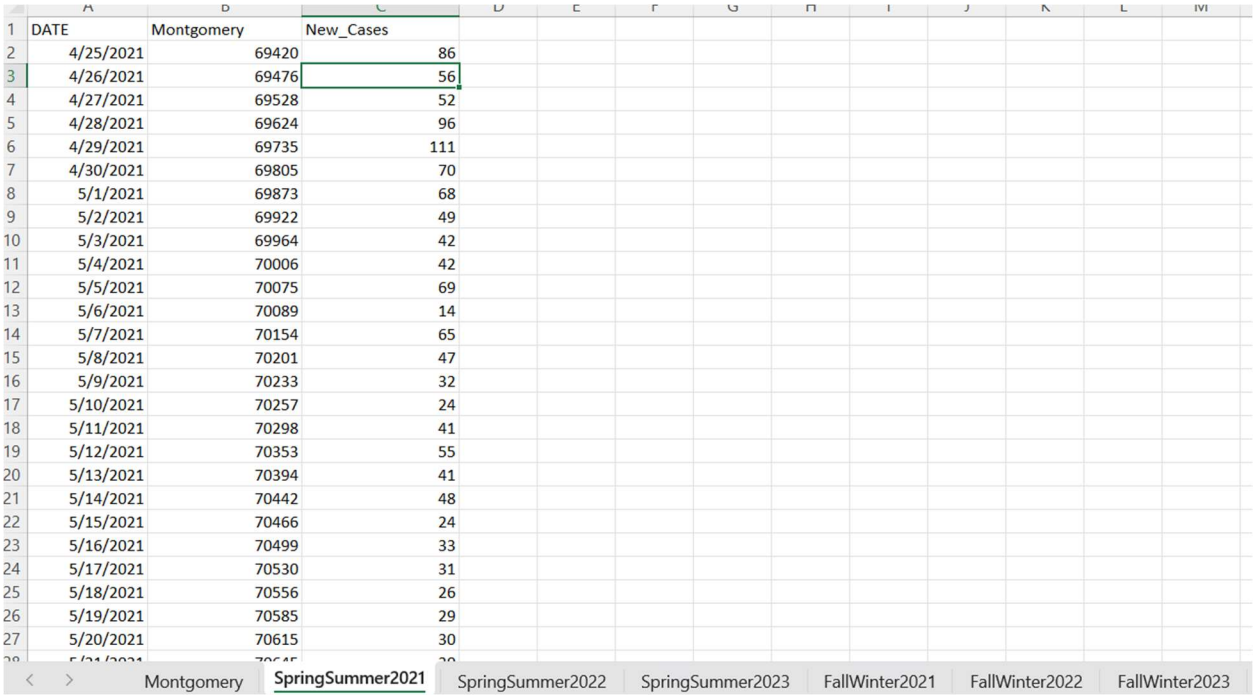
At the outset, we must gather detailed COVID-19 infection data for various regions between 2021 and 2024, when vaccines have become available and the most of the United States was no longer under stay-at-home orders. In most regions, daily infection data is readily available between 2021 through the first half of 2023. However, on May 11, 2023, the U.S. COVID-19 Public Health Emergency declaration expired, and many regions, including D.C. and Virginia no longer continuously reported daily infection rates after that date. Maryland, however, continued to report daily infection rates through 2023 and 2024. Thus, this experiment is based on Maryland's infection data between 2021 and 2024 for the counties of Montgomery, Prince George, and Howard.

Maryland provides county-level COVID-19 infection data at the following link:

https://data.imap.maryland.gov/datasets/maryland::mdcovid19-casesbycounty/about

It includes collection of cumulative positive COVID-19 test results that have been reported each day by the local health departments via the state's electronical health reporting system. The table, however, does not provide explicit data for new reported cases each day. In order to calculate the number of new cases for each date, a formula is applied to the spreadsheet to subtract the previous day's cumulate total from the current day's cumulative total.

The new data table is then processed to isolate daily new cases for each county: Montgomery, Prince William, and Howard. For each county, further sub-spreadsheets are created to isolate cases for: Summer and Fall 2021, 2021/22 winter season date range, and 2022/23 winter season date range. This creates a new column of "New Cases" by day. Below is an example of the data file for Montgomery county, with sub-sheets for each of the time ranges:



| | A | B | C |
|---|---|---|---|
| 1 | DATE | Montgomery | New_Cases |
| 2 | 4/25/2021 | 69420 | 86 |
| 3 | 4/26/2021 | 69476 | 56 |
| 4 | 4/27/2021 | 69528 | 52 |
| 5 | 4/28/2021 | 69624 | 96 |
| 6 | 4/29/2021 | 69735 | 111 |
| 7 | 4/30/2021 | 69805 | 70 |
| 8 | 5/1/2021 | 69873 | 68 |
| 9 | 5/2/2021 | 69922 | 49 |
| 10 | 5/3/2021 | 69964 | 42 |
| 11 | 5/4/2021 | 70006 | 42 |
| 12 | 5/5/2021 | 70075 | 69 |
| 13 | 5/6/2021 | 70089 | 14 |
| 14 | 5/7/2021 | 70154 | 65 |
| 15 | 5/8/2021 | 70201 | 47 |
| 16 | 5/9/2021 | 70233 | 32 |
| 17 | 5/10/2021 | 70257 | 24 |
| 18 | 5/11/2021 | 70298 | 41 |
| 19 | 5/12/2021 | 70353 | 55 |
| 20 | 5/13/2021 | 70394 | 41 |
| 21 | 5/14/2021 | 70442 | 48 |
| 22 | 5/15/2021 | 70466 | 24 |
| 23 | 5/16/2021 | 70499 | 33 |
| 24 | 5/17/2021 | 70530 | 31 |
| 25 | 5/18/2021 | 70556 | 26 |
| 26 | 5/19/2021 | 70585 | 29 |
| 27 | 5/20/2021 | 70615 | 30 |



The Virginia Department of Health (VDH) provides Virginia COVID-19 data by county, including Arlington County, Alexandria County, and Loudon County. (Data for nearby Fairfax County, however, is unavailable). The following link provides public access to a master data table that can be downloaded as an Excel spreadsheet:

https://data.virginia.gov/dataset/vdh-covid-19-publicusedataset-cases-by-district-death-hospitalization

The Virginia Department of Health (VDH) provides Virginia COVID-19 data by county, including Arlington County, Alexandria County, and Loudon County. (Data for nearby Fairfax County, however, is unavailable). The following link provides public access to a master data table that can be downloaded as an Excel spreadsheet:

https://data.virginia.gov/dataset/vdh-covid-19-publicusedataset-cases-by-district-death-hospitalization

This table includes the cumulative (total) number of COVID-19 cases, hospitalizations, and deaths for each county in Virginia by report date. The table, however, does not provide explicit data for new reported cases each day. In order to calculate the number of new cases for each date, a formula is applied to the spreadsheet to subtract the previous day's cumulate total from the current day's cumulative total. This creates a new column of "New Cases" by day.

The new data table is then processed to isolate daily new cases for each county: Arlington, Alexandria, and Loudon. For each county, further spreadsheets are created to isolate cases in the 2021 prewinter season date range, 2022 prewinter season date range, 2021/22 winter season date range, and 2022/23 winter season date range.

The resulting 12 data files are:

(1) Arlington new cases from June 25, 2021 through October 31, 2021;

(2) Arlington new cases from June 25, 2022 through October 31, 2022;

(3) Arlington new cases from October 26, 2021 through February 28, 2022

(4) Arlington new cases from October 26, 2022 through February 28, 2023

(5) Alexandria new cases from June 25, 2021 through October 31, 2021;

(6) Alexandria new cases from June 25, 2022 through October 31, 2022;

(7) Alexandria new cases from October 26, 2021 through February 28, 2022

(8) Alexandria new cases from October 26, 2022 through February 28, 2023

(9) Loudon new cases from June 25, 2021 through October 31, 2021;

(10) Loudon new cases from June 25, 2022 through October 31, 2022;

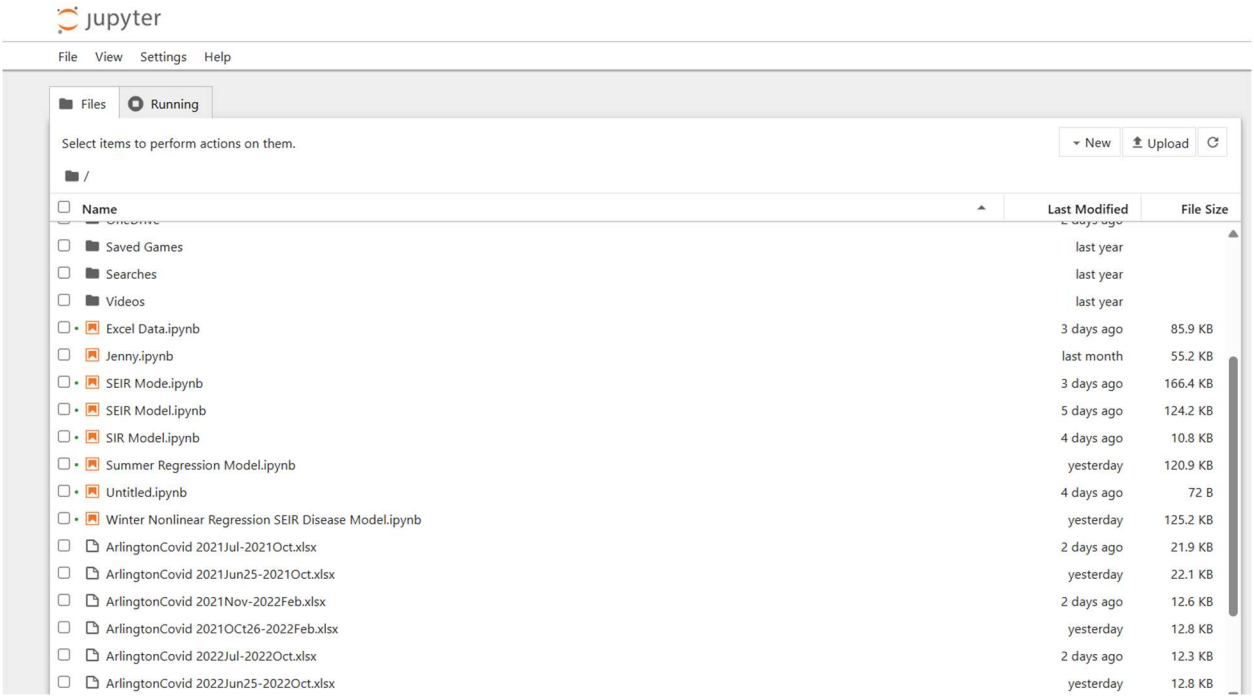
(11) Loudon new cases from October 26, 2021 through February 28, 2022

(12) Loudon new cases from October 26, 2022 through February 28, 2023

The winter ranges cover the four winter months from November through February. This range is selected because it includes all the holidays between Thanksgiving through New Years when the population is most likely to gather indoors. The pre-winter date ranges are set to the four months immediately before, i.e., from July through October. Six days prior to July 1$^{st}$ (June 25$^{th}$ through

July 1st) and six days prior to November 1st (October 26th through November 1st) are included in the date ranges because they are needed in subsequent steps to calculate the number of currently infectious individuals per day (with an average recovery rate set to 7 days. This will be discussed later in detail.)

Once we have the updated data table files for each location and time range, upload these files to the Jupyter workspace by clicking on the "Upload Files" button on the Jupyter Dashboard. The data tables are now ready to be read in using Python code and to be processed using mathematical modeling.




## Step 2: Set Up Disease Model to Describe the Trajectory of Covid-19 Transmission

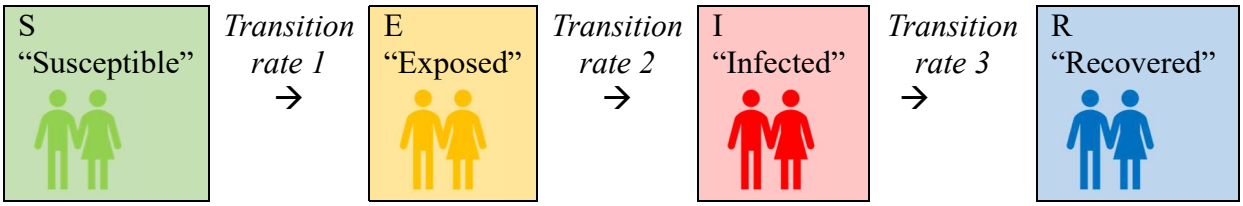
We cannot simply obtain the COVID19 basic reproduction number by looking at the actual infection data. We must apply a mathematical model to the data to help us make the estimate.

This experiment uses the SEIR compartment disease model. In the SEIR model, a population $N$ is divided into four distinct subpopulations (*i.e.*, "compartments") with respect to a disease (e.g., Covid19) that is capable of infecting anyone in the population:

(1) susceptible individuals $S$ who have not contracted the disease,
(2) exposed individuals $E$ who have been exposed to the disease but are not yet infectious
(3) infectious individuals $I$ who are capable of infecting others, and
(4) recovered individuals $R$ who recovered from the disease and are no longer able to infect others.

The task of the SEIR model is to predict the trajectory of an epidemic as individuals transition from one subpopulation to another. In subsequent steps of this experiment, we will fit this trajectory to the actual data to estimate the COVID19 reproduction number.

The relationship between *S*, *E, I*, and *R* can be visualized using a flowchart, in which arrows indicate the direction of movement between subpopulations:




**Step 2(a)** ***State Initial Assumptions for the SEIR Model***

In any mathematical modeling of real-world phenomenon, certain assumptions must be made for the system. The assumptions chosen define the particular details of the model. Below are the assumptions for the particular SEIR model used in this experiment:

- Assume that the population *N* is constant with no vital dynamics. In other words, within the duration of time that we are studying, we do not consider inflows of new births and outflows of natural death and that the population stays the same. This is a reasonable assumption for this experiment because COVID19 is a fast disease. In the short period that we are studying (i.e., four months), the total population does not significantly change.
- Interactions between individuals in the population are homogenous, meaning that individuals within the population have equal probability of coming into contact with each other and interacting. This is known as "mass incidence" (Mickens).
- We assume that in the *R* compartment, recovered individuals cannot become reinfected. For diseases with a limited period of immunity from reinfection such as Covid-19, assuming no reinfection of the recovered population is reasonable if the time period under study is short (Cooper, 2022).
- No distinction is made between fully, partially recovered, or those who die from the disease, as long as the individual can no longer transmit the disease to others. Fully vaccinated individuals also fall under the *R* compartment, as they are assumed to be incapable of contracting and transmitting the disease to others.

The equations set up below depend on these assumptions.

**Step 2(b)** ***Set up Mathematical Equations, Define Parameters***

The four compartments for a population *N* can be denoted by the following functions:

*S(t)*: the number of susceptible individuals at time t

*E(t)*: the number of exposed individuals at time t

*I(t)*: the number of infectious individuals at time t

*R(t)*: the number of recovered individuals at time t

These four groups add up to the total population *N*, which is assumed to be constant:

$$N = S(t) + E(t) + I(t) + R(t).$$

We need to set up equation to describe how fast are the number of susceptible people *S(t)* changing with respect to time, how fast is the number of exposed people *E(t)* changing with respect to time, how fast is the number of infected people *I(t)* changing with respect to time, and how fast the number of recovered people *R(t)* is changing with respect to time. In calculus terms, this means analyzing the derivatives of *S(t)*, of *E(t)*, of *I(t)*, and of *R(t)*, respectively. The calculus Leibniz notations used to denote each of these derivative functions are: $\frac{dS(t)}{dt}$, $\frac{dE(t)}{dt}$, $\frac{dI(t)}{dt}$and $\frac{dR(t)}{dt}$.

*Rate of Change for Susceptibles:*

Let us first construct an equation to describe the rate of change from the susceptibles category *S* to the exposed category *E*. The function $\frac{dS(t)}{dt}$ for the equation we want to set up should define the rate at which new infected cases are generated per unit time. We will use days as the unit time.

When a susceptible individual comes into contact with an infectious individual, that susceptible individual becomes infected with a certain probability and transitions from the susceptible category into the infected category. Recall that in this model, there is homogenous mixing of the population *N* where there is any one individual has equal chance of meeting any other individual. Thus, for any one individual susceptible person, we assume there is an average number of contacts *b* the individual has per day. Also, for each of those interactions, there is an unbiased chance that a particular contact is an infected individual. Specifically, the probability that a susceptible individual comes into contact with an infected individual is $\frac{I(t)}{N}$. We assume that each susceptible individual meets on average a fraction of the total population and thereby meets a fraction of the total infected individuals. For example, if 10% of the population is infected, then 10% of the people a susceptible individual comes into contact with will be infected. Multiplying the two terms together, we get $b * \frac{I(t)}{N}$, which is the average number of interactions between individuals in the S and I categories on any given day. But we are not done. We also need to take into account the transmissivity of the particular disease being modeled. Depending on the nature of the pathogen, there is a percentage chance *p* that the interaction between a susceptible individual and an infected individual actually leads to infection. In other words, some diseases or more infectious than others.

To summarize, the probability of infection on any given day in a population *N* is:

*B* (the average number of contacts the individual has per day) ×

$\frac{I(t)}{N}$ (probability that the contact is infected) × 1(a)

$p$ (probability that interaction between I and S result in transmission)

We can use the coefficient $\beta$ to denote the first and third aspects above. This constant parameter $\beta$ can be thought of as the "transmission rate constant" of the population. (Kauf; math 2120 SIS Model Part 1):

transmission rate constant $\beta$ = number of contacts $b$ × disease transmissivity $p$

It measures the probability per day that an infectious individual can infect a susceptible individual. The unit for $\beta$ is 1/day. Because the value of $\beta$ depends on the particular disease being studied as well as social and behavioral factors, estimating $\beta$ is not an easy task and is the subject of many epidemiology studies. Estimation can only be done retrospectively after the epidemic has run its course. (Brauer, Chavez).

Rewriting expression 1(a), we have the following expression for the probability of infection on any given day:

$$\beta \frac{I(t)}{N} \qquad 1(b)$$

To get the number of infections for the total susceptible population per day, we multiply $\beta \frac{I(t)}{N}$ by *S(t)*. Thus, we have the following differential equation to describe the rate of change of susceptible individuals:

$$\frac{dS(t)}{dt} = -\beta \frac{S(t)I(t)}{N} \qquad 2(a)$$

Equation 2(a) is a function of the number of susceptables *S*, the number of infected *I*, and the total population *N*. The physical unit is number of individuals over unit time (day). The equation has a negative sign because when infection occurs, the infected individuals leave the *S* category to enter the *E* category. Together, this equation tells us that the higher the number of interactions between those in the *S* category and those in the *I* category, or the more contagious the disease is, the higher the likelihood that susceptible people will become exposed to the virus and thereby leave the *S* category to enter the *E* category.

*Rate of Change for Exposed*

We next construct an equation to describe how the Exposed category *E* changes with respect to time. While the susceptible subpopulation decreases per day by the number of individuals who become infected, the Exposed subpopulation increases by the same number of newly infected individuals in that time. While people are entering the category, others who have already been exposed for a period of time and have now become infectious leave the category to enter the Infectious category *I.*

While people are entering the *E* category, others who have already been exposed for a period of time and have now infectious leave the category to enter the *I* category at a rate of $\alpha E(t)$. The

coefficient α can be thought of as the "latent rate." (CITE)). It is the reciprocal of the average number of days an exposed person stays in the *E* category. In other words, it denotes the fraction of exposed individuals who become infectious per day. The unit for $\gamma$ is 1/day.

The differential equation can be written as follows:

$$\frac{dE(t)}{dt} = \frac{\beta}{N} S(t)I(t) - \alpha E(t) \qquad 2(\text{b})$$

This equation has both a positive and a negative component, thereby telling us that there are both people coming into the category as well as people leaving the category. As explained above, the positive component $\frac{\beta}{N} S(t)I(t)$ is the same $\frac{\beta}{N} S(t)I(t)$ component as in equation 1(a) for $\frac{dS(t)}{dt}$, describing the rate in which people enter the *E* category from the *S* category. The negative component $-\alpha E(t)$ describes the rate in which exposed individuals become infectious and leave the *E* category.

*Rate of Change for Infectious*

We next construct an equation to describe how the infected category *I* changes with respect to time. While the exposed subpopulation decreases per day by the number of individuals who become infectious, the infectious subpopulation increases by the same number of newly infected individuals in that time. While people are entering the *I* category, others who have already been infexious for a period of time and have now recovered leave the category to enter the *R* category at a rate of $\gamma I(t)$. The coefficient γ can be thought of as the "recovery rate." (Maultsby). It is the reciprocal of the average number of days an infected person stays in the *I* category. In other words, it denotes the fraction of infectious individuals who recover per day. The unit for $\gamma$ is 1/day.

The equation can be written as follows:

$$\frac{dI(t)}{dt} = \alpha E(t) - \gamma I(t) \qquad 2(\text{b})$$

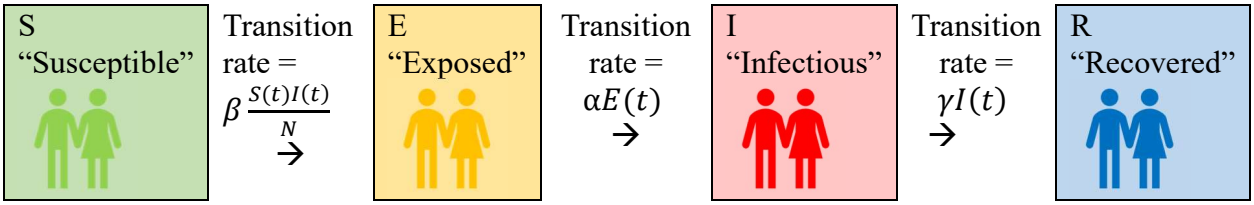
This equation has both a positive and a negative component, thereby telling us that there are both people coming into the category as well as people leaving the category. As explained above, the positive component $\frac{\beta}{N} S(t)I(t)$ is the same $\frac{\beta}{N} S(t)I(t)$ component as in equation 1(a) for $\frac{dS(t)}{dt}$, describing the rate in which people enter the *I* category from the *S* category. The negative component $-\gamma I(t)$ describes the rate in which infected individuals recover and leave the category.

*Rate of Change for Recovered*

Lastly, we construct an equation to describe how the recovered category changes with respect to time. The differential equation can be written us:

$$\frac{dR(t)}{dx} = \gamma I(t) \qquad 2(\text{c})$$

This fourth equation has only one positive component, $\gamma I(t)$, which is the same negative component $\gamma I(t)$ in equation 1(b) describing the rate at which infected individuals leave the infected category to enter the recovered category. The following flowchart shows these rates of transition as individuals move from S, to E, to I, then to the R category:




Combining the four equations, we have the following coupled system of nonlinear ordinary first order differential equations:

$$\frac{dS(t)}{dt} = -\beta \frac{S(t)I(t)}{N} \qquad 2(a)$$

$$\frac{dE(t)}{dt} = \beta \frac{S(t)I(t)}{N} - \alpha E(t) \qquad 2(b)$$

$$\frac{dI(t)}{dt} = \alpha E(t) - \gamma I(t) \qquad 2(c)$$

$$\frac{dR(t)}{dt} = \gamma I(t) \qquad 2(d)$$

The parameter $N$ is the total population number of individuals where $N = S + E + I + R$. The parameters $\beta$, $\alpha$, and $\gamma$ – the transmission rate constant, the latent rate constant, and the recovery rate constant -- all have units 1/day. The independent variable time $t$ has units in days. The units for all four differential equations are number of individuals per day.

***Step 2(c): Define Initial Conditions and Account for Vaccination***

Differential equations typically have infinite solutions. To solve for a particular solution, we need to define initial conditions, *i.e.*, what happens to the four functions at time $t = 0$. We can denote the following:

$$S_o = S(0)$$

$$I_o = I(0)$$

$$E_o = E(0)$$

$$R_0 = R(0)$$

In our model, the values $S_o$ is positive because at the very beginning, most of the population has not yet been exposed. We can also take into account the impact of vaccination here. If at time $t = 0$, a number $V$ of the population (where V < N) is vaccinated, then that number of individuals

moves directly from the S category to the R category. This changes $S_o$ to $S_o - V$ and $R_0$ to $R_0 + V$.

***Step 2(c): Derive Equation for Basic Reproduction Number by Analyzing Mathematical Properties of the SEIR Equations at Time t=0***

The system of nonlinear ordinary first order differential equations 3(a)-3(d) does not have an explicit analytical solution, and we must use a numerical scheme to solve for *S(t), E(t), I(t),* and *R(t)*. We can, however, determine analytically the growth rate of infection *I(t)* at the beginning of the outbreak. Having a solution for *I(t)* at $t = 0$ will allow us to find a threshold ratio between β and γ (*i.e.,* reproduction number) that will predict whether an epidemic will occur.

In our scenario, at $t = 0$, a positive number *V* of the population *N* is vaccinated, and $E_o$ and $I_o$ are close to 0, we have $\frac{S_o}{N} \approx \frac{N-V}{N} = 1 - \frac{V}{N}$. This is a scalar, and we can rewrite equations 2(a) and 2(b) as the following:

$$\frac{dS(t)}{dt} = -\beta\left(\frac{S_o}{N}\right)I(t) \qquad 2(a)$$

$$\frac{dE(t)}{dt} = \beta\left(\frac{S_o}{N}\right)I(t) - \alpha E(t) \qquad 2(b)$$

$$\frac{dI(t)}{dt} = \alpha E(t) - \gamma I(t) \qquad 2(c)$$

$$\frac{dR(t)}{dt} = \gamma I(t) \qquad 2(d)$$

We only need to consider equations 4(b) and 4(c) because they alone drive the solutions to 4(a) and 4(d). Equations 4(b) and 4(c) is a linear system and can be solved analytically using linear algebra and performing an eigenvalues analysis. First, we write equations 4(b) and 4(c) into the matrix notation form *X' = **A** *X*, where *X* is the vector containing E(t) and I(t) and ***A*** is a 2 x 2 matrix containing the parameters $\beta, \alpha,$ and $\gamma$:

$$\frac{d}{dt}\begin{bmatrix}E\\I\end{bmatrix} = \begin{bmatrix}-\alpha & \beta\frac{S_o}{N}\\ \alpha & -\gamma\end{bmatrix}\begin{bmatrix}E\\I\end{bmatrix}$$

To solve for $\begin{bmatrix}E\\I\end{bmatrix}$, we must look for eigenvalues λ for the matrix $\boldsymbol{A} = \begin{bmatrix}-\alpha & \beta\left(1-\frac{S_o}{N}\right)\\ \alpha & -\gamma\end{bmatrix}$. We ask, is there values λ for which the following is true:

$$\boldsymbol{A}\begin{bmatrix}E\\I\end{bmatrix} = \lambda\begin{bmatrix}E\\I\end{bmatrix}$$

We can rewrite the above equation as a homogenous system:

$$(\boldsymbol{A} - \lambda \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}) \begin{bmatrix} E \\ I \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Then, to find a non-zero solution for eigenvector $\begin{bmatrix} E \\ I \end{bmatrix}$, we need to set the determinant $\boldsymbol{D}$ of matrix $(\boldsymbol{A} - \lambda \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix})$ to 0:

$$det \begin{bmatrix} -\alpha - \lambda & \beta \frac{S_o}{N} \\ \alpha & -\gamma - \lambda \end{bmatrix} = 0$$

Applying determinant formula for a 2x2 matrix, we get the following expression for λ:

$$\lambda^2 + (\alpha + \gamma)\lambda - \alpha\left(\beta \frac{S_o}{N} - \gamma\right) = 0$$

Solving for the polynomial result in two eigenvalues, one positive $\lambda_+$, and one negative $\lambda_-$. For the positive eigenvalue $\lambda_+$, we have:

$$\lambda_+ = -\frac{1}{2}[\alpha + \gamma] + \frac{1}{2}\sqrt{(\alpha - \gamma)^2 + 4\alpha\beta\frac{S_o}{N}}$$

Recall from above that we have $\frac{d}{dt}\begin{bmatrix} E \\ I \end{bmatrix} = \boldsymbol{A}\begin{bmatrix} E \\ I \end{bmatrix} = \lambda \begin{bmatrix} E \\ I \end{bmatrix}$. Using $\lambda_+$, we can solve for $\frac{dI(t)}{dt}$:

$$\frac{dI(t)}{dt} = \left(-\frac{1}{2}[\alpha + \gamma] + \frac{1}{2}\sqrt{(\alpha - \gamma)^2 + 4\alpha\beta\frac{S_o}{N}}\right) I(t)$$

Integrating on both sides and using the initial condition $I_o$ we can approximate the initial growth rate at $t = 0$ as:

$$I(t) \approx I_o e^{(-\frac{1}{2}[\alpha+\gamma]+\frac{1}{2}\sqrt{(\alpha-\gamma)^2+4\alpha\beta\frac{S_o}{N}})t}$$

This means that in the beginning, the number of infectives grows exponentially and the initial exponential growth rate *G* is approximately:

$$G \approx -\frac{1}{2}[\alpha + \gamma] + \frac{1}{2}\sqrt{(\alpha - \gamma)^2 + 4\alpha\beta\frac{S_o}{N}}$$

We see that for I(t) to be positive and increase over time, *G* must be positive. Through algebraic manipulation, we find that $G > 0$ when $\beta\frac{S_o}{N} > \gamma$ , i.e., $\frac{\beta}{\gamma}\frac{S_o}{N} > 1$ . **Thus, we define the reproduction number at $t = 0$ to be:**

$$\boldsymbol{r_0 \equiv \frac{\beta}{\gamma}\frac{S_o}{N}} \qquad (3)$$

At t = 0, if $\frac{\beta}{\gamma}\frac{S_o}{N} > 1$ , our model predicts an epidemic. On the other hand, at time t = 0, if $\frac{\beta}{\gamma}\frac{S_o}{N} \leq 1$ , no epidemic occurs. One way to think about this is, if the rate at which susceptible individuals become infected $\beta$ exceeds the rate at which infected individuals recover $\gamma$ , there will be an accumulation $\beta$ of individuals in the infected category and the infection curve will rise.

## Step 3: Find Numerical Solution Using Python

Very few nonlinear systems can be solved analytically in terms of elementary functions or even special functions. (Kuhl) We must typically rely on computer processes to numerically approximate the solution. The system of nonlinear ordinary differential equations of 2(a) – 2(c) is one in which we cannot use analytical proof to come up with an exact solution. We can, however, write a computer program using a language such as Python to help us find the numerical solutions for *S(t)*, *E(t), I(t)*, and *R(t)*. Python contains a library SciPy used for mathematics and science contains modules for solving advanced computation and scientific tasks. In order to solve a set of ordinary differential equations, we use the Integrate module and the ODEINT function used for solving ordinary deferential equations.

Here again are the differential equations for the SIR model:

$$\frac{dS(t)}{dt} = -\beta\frac{S(t)I(t)}{N} \qquad 2(a)$$

$$\frac{dE(t)}{dt} = \beta\frac{S(t)I(t)}{N} - \alpha E(t) \qquad 2(b)$$

$$\frac{dI(t)}{dt} = \alpha E(t) - \gamma I(t) \qquad 2(c)$$

$$\frac{dR(t)}{dt} = \gamma I(t) \qquad 2(d)$$

Before using Python's ODEINT function to solve for the above differential equations, we have to first rewrite them in vector form. Letting the vector $\vec{V}$ = (S, E, I, R), we need to write a function that returns: $\frac{dV(t)}{dt} = (\frac{dS(t)}{dt}, \frac{dE(t)}{dt}, \frac{dI(t)}{dt}, \frac{dR(t)}{dt})$. The function $\frac{dV(t)}{dt}$ can take in vector $\vec{V}$ = (S, E, I, R) and time t as parameters.

$$\vec{V} = \begin{bmatrix} S \\ E \\ I \\ R \end{bmatrix} \Rightarrow \frac{d\vec{V}}{dt} = \vec{f}\left(t, \vec{V}\right) = \vec{f}\,(t, S, I, R) = \begin{bmatrix} S' \\ E' \\ I' \\ R' \end{bmatrix} = \begin{bmatrix} -\beta\frac{S(t)I(t)}{N} \\ \beta\frac{S(t)I(t)}{N} - \alpha E(t) \\ \alpha E(t) - \gamma I(t) \\ \gamma I(t) \end{bmatrix}$$

We then write the following function for $\vec{V}$ in Python:

```
def dVdt(vector, t, N, alpha, beta, gamma):
    S, E, I, R = vector
    dSdt = -beta * S * I / N
    dEdt = beta * S * I / N - alpha * E
    dIdt = alpha * E - gamma * I
    dRdt = gamma * I
    return dSdt, dEdt, dIdt, dRdt
```

We can now use Python's ODEINT function to solve for the set of differential equations contained in vector $\vec{V}$. The initial conditions are S0, E0, I0, and R0. We also take into account the number of vaccinated individuals at t=0. We then use the ODEINT function to integrate over the values of time t (in days) between t= 0 to t=120 (or whatever length of days that matches the actual infection data obtained from VDH), where *t* is implemented as an array data structure.

```
  # Define initial number of exposed, infected, and recovered individuals: E0, I0, R0.
    # take into account 75,000 fully vaccinated as of May 2021
    E0, I0, R0 = 0, dataI[0], 75000

 # Set up initial conditions vector for S0, E0, I0, R0
    initial_conditions = S0, E0, I0, R0

    # Create an array of evenly spaced time values(in days)
    t = np.linspace(0, 120, 120)

    # Solve for the differential equations for the SIR model (in vector form) by integrating over
the time array t.
    solution = odeint(dVdt, initial_conditions, t, args=(N, alpha_beta[0], alpha_beta[1],
gamma))
```

Lastly, the ODEINT function returns the solution vector in the following form:

Array ([S0, E0, I0, R0], [S1, E1, I1, R1], [S2, E2, I2, R2] … , [S120, E120, I120, R120])

Thus, the array has to be transposed in order to extract the numerical solutions for S, E, I, R:

```
S, I, R = solution.T
```

## Step 4: Process Real-life Infectious Data Using Python

Although we can now graph the solution curve for the modeled infectious data, we cannot yet compare this data to the real-life data that we obtained in step 1. This is because the real-life data contains only the number of new infections each day, not the cumulative number of people who are still in the "infectious" category per day. We must process the data table to sum up the infection count from a previous range of days (the recovery period), because those positive

individuals have not yet recovered and left the “infectious” category. In this experiment, we assume that the average recovery period for COVID-19 is seven days. (Kuhn).

Using Python, we first read in a data table containing entries for the number of new infectious per date. For example, in the Python code below, we read in am Excel file containing Arlington infection data entries from October 26, 2021 through February 28, 2022:

```
# read in Arlington Covid data for new infections
# read in Arlington Covid data for new infections
winter21_data = pd.read_excel('ArlingtonCovid 2021Oct26-2022Feb.xlsx')
winter22_data = pd.read_excel('ArlingtonCovid 2022Oct26-2023Feb.xlsx')
winter21_cases = list(winter21_data.New_Cases)
winter22_cases = list(winter22_data.New_Cases)
```

The read-in data table in the code above contains the four months from November through February, as well as six days prior to October first, so that we can calculate a 7-day running total infection count for each day. We process the read-in data into a new list containing the current number of infectious individuals per day (7-day running total) for the specific date range between November 1, 2021 and February 28, 2022, wherein November 1, 2021 is the first day and is represented by index 0 of the list:

```
# Estimate infectious popluation for all read in data - assumes infectious period of 7 days
def update_list(lst):
    updated_lst = []

    for i in range(len(lst)):
        # Get the last 6 entries (or fewer) before the current entry
        last_six = lst[max(0, i-6):i]
        updated_entry = lst[i] + sum(last_six)
        updated_lst.append(updated_entry)

    updated_lst = updated_lst[6:]
    return updated_lst

winter21_infectious_data = update_list(winter21_cases)
winter22_infectious_data = update_list(winter22_cases)
```

---

Another issue we have to take into account is underreporting. Underreporting of COVID-19 cases and death is a commonly acknowledged phenomenon. [Lau et. al, 2020] It can arise due to several reasons. First, cases and deaths are not counted if they occur outside public health surveillance systems. Individuals may remain outside such systems due to lack of access, fear or personal choice (e.g., McCulloh et al., 2020, p. 2). Second, a shortage of information, a lack of tests and testing inaccuracies during the early stages of the pandemic may have led to many cases and/or deaths being misdiagnosed or missed altogether (Manski & Molinari, 2021; Reese et al., 2021). Third, the lack of universal testing protocols means that the significant number of asymptomatic but infectious individuals evade official counts. (e.g. Reese et al., 2021). The most

recent systematic study on underreporting estimates a multiplication factor for the true number of cases between two and nine. [Millimet et al., 2022]

In this experiment, we take the conservative approach and use a multiplication factor of two to account for underreporting. Thus, for each entry in the list of infectious number of individuals per day, we multiply the value by two. This can be accomplished by the following code:

```
# account for 50% underreporting - multiply by 2
winter21data = [x * 2 for x in winter21_infectious_data]
winter22data = [x * 2 for x in winter22_infectious_data]

summer21data = [x * 2 for x in summer21_infectious_data]
summer22data = [x * 2 for x in summer22_infectious_data]
```

The real-life infection data is now ready to be used to compare with the modeled infection data.

## Step 5: Implement Nonlinear Regression Using Python to Find Best-Fit Infection Curve

Having set up a method to model the progression of COVID-19 transmission over time and having processed the real-life COVID-19 infectious data, we now use nonlinear regression methods to fit our mathematical model to our real-life data. Nonlinear regression is a statistical technique used to model complex relationships between variables where the relationship is not a straight line (i.e., not linear). The aim of using nonlinear regression in our experiment is to find a modeled infection curve that most closely aligns with the real-life data and thus obtain a reproduction number that more accurately represents the real-life disease dynamics. Specifically, we want to find a curve having parameters -- transmission rate β and latent rate α -- that best describe the real-life data's behavior. The transmission rate β and latent rate α of the best fit curve will then be used to estimate the best-fit reproduction number.

We first must define a function that gives the differences or "loss" between the modeled infection curves and the real-life curve. This can be calculated using a **least-squares loss function.** For each data point, the squared differences between the observed values and the predicted values from the model are computed as such,

$$loss = \sum_{i=1}^{n} (I_{predicted}(t_i, \alpha, \beta) - I_{observed}(t_i))^2$$

where $I_{predicted}$ is the modeled infectious count at time $t_i$, based on parameters α and β, $I_{observed}$ is the real-life infectious count at time $t_i$, and $n$ is the number of time points (days) in the data. Below is the python implementation of the least-squares loss function:

```
# Define loss function using SEIR Model ODEs
def loss_function(alpha_beta, N, gamma, dataI):

    # Define initial number of exposed, infected, and recovered individuals: E0, I0, R0.
    # take into account 75,000 fully vaccinated as of May 2021
    E0, I0, R0 = 0, dataI[0], 75000

    # Define initial number of susceptible individuals.
    S0 = N - E0 - I0 - R0

    # Set up initial conditions vector for S0, E0, I0, R0
    initial_conditions = S0, E0, I0, R0

    # Create an array of evenly spaced time values(in days)
    t = np.linspace(0, 120, 120)

    # Solve for the differential equations for the SEIR model (in vector form) by integrating over the time array t.
    solution = odeint(dVdt, initial_conditions, t, args=(N, alpha_beta[0], alpha_beta[1], gamma))
    # transpose solution array to extract data
    S, E, I, R = solution.T

    loss = 0

    for i in range(120):
        data_I = dataI[i]
        model_I = I[i]
        res = (model_I - data_I)**2
        loss += res
    return (loss)
```

Note that before making the loss calculation, our "loss_function" definition calls the "odeint" function of Python's SciPy library (see step 3 above) to solve for $I_{predicted}$. For each of the evenly spaced time values, the difference between $I_{predicted}$ and $I_{observed}$ is calculated.

Having defined the loss function, we now use a nonlinear optimization algorithm to adjust the parameters α and β in order to minimize the loss function (*i.e.*, minimize the difference between the modeled curve and the observed data curve). The optimization algorithm iteratively updates the values of α and β to improve the fit between the two curves.

To avoid calling needless iterative steps in our optimization algorithm, we want to set realistic constraints for α and β to reflect the realistic biological range of latent rates and transmission rates. Without constraints, the optimization algorithm might converge to extreme parameter values that reduce the fitting error but produce a model that is not biologically meaningful.

This experiment uses the **Limited-memory Broyden–Fletcher–Goldfarb–Shanno with Box constraints** (**L-BFGS-B)** optimization method, which is particularly suited for problems where the variables are subject to boundary constraints for parameters. (Zhu et al., 1997). The algorithm uses a limited-memory approach, making it computationally efficient. (Stanford, 2004) L-BFGS-B is also robust in handling non-linear relationships between model parameters, which are typical in infectious disease models. (Dewi et al., 2017)

There are several other methods in SciPy's minimize function that allow for optimization with restricted bounds besides the L-BFGS-B method. Each of these methods, including TNC (Truncated Newton), SLSQP (Sequential Least Squares Programming), trust-constr (Trust

Region Constrained), and Powell, were attempted. However, the L-BFGS-B method came closest to approximating the infectious curve.

The **SciPy library** in Python provides a convenient implementation of L-BFGS-B in its "optimize.minimize" function. To find the best fit parameters α and β, we call the "optimize.minimize" function, passing in our defined loss function, setting the initial guesses for α and β, setting bounds for α and β, and setting the method to 'L-BFGS-B':

```
# Initial guess for alpha and beta
initial_guess = [1./5.5, 0.4]  # Initial guess for alpha and beta

# Set bounds for alpha and beta
bounds = [(1./6, 1./2), (0.2, 2)]  # alpha between 1/(6 days) and 1/(2 days), beta between 0.2 and 2

# Use scipy.optimize.minimize to minimize the loss function for winter 2021 Arlington data and winter 2022 Arlington data
winter21_result = spo.minimize(loss_function, initial_guess, args=(N, gamma, winter21data), method='L-BFGS-B', bounds=bounds)
winter22_result = spo.minimize(loss_function, initial_guess, args=(N, gamma, winter22data), method='L-BFGS-B', bounds=bounds)

# Print the optimal parameters and minimum loss for winter 2021 Arlington and winter 2022 Arlington
print("Optimal alpha and beta for Winter 2021:", winter21_result.x)
print("Optimal alpha and beta for Winter 2022:", winter22_result.x)
```

```
Optimal alpha and beta for Winter 2021: [0.5        0.35879849]
Optimal alpha and beta for Winter 2022: [0.5        0.24240437]
```

```
#best fit alpha
alpha_fit_winter21 = winter21_result.x[0]
alpha_fit_winter22 = winter22_result.x[0]
```

```
#best fit beta
beta_fit_winter21 = winter21_result.x[1]
beta_fit_winter22 = winter22_result.x[1]
```

The sample code above calculates and returns the best-fit parameters α and β corresponding to Arlington's infection data for the months of November 2021 through February 2022, and for the months of November 2022 through February 2023. The resulting α and β are stored in an array.

## Step 6: Calculate Best-Fit Reproduction Number

We revisit reproduction number equation (3) determined in step (2):

$$\boldsymbol{r_0} \equiv \frac{\beta}{\gamma}\frac{S_o}{N} \qquad (3)$$

Now that we have the best-fit parameter β, we can easily determine $\boldsymbol{r_0}$ by plugging this best-fit β into equation (3), alongside our predefined values for recovery rate γ, initial number of susceptible individuals $S_0$ and population count $N$.

For example, in the sample code above for Arlington's winter 2021/22 data, we determined the best-fit parameter β to be 0.35879849. Using this β, the code below calculates the reproduction number $\boldsymbol{r_0}$ corresponding to Arlington's 2021/22 winter data to be 1.566:

```
#Calculate reproduction number for winter 21
rep0_winter21 = (beta_fit_winter21 * S0_winter21) / (gamma * N)
print("reproduction number for winter 21:", rep0_winter21)
```

```
reproduction number for winter 21: 1.566001128851172
```

## Step 7: Graph Results Using Python

Having determined the best-fit infectious curve, we can now graph the curve alongside the real-life infection curve on one plot to compare the two.

First, we must explicitly define the best-fit curve using our best-fit α and β determined in step 5 above. Recall from step 3 above that we defined a function called "dVdt" that take in vector $\vec{V}$ = (S, E, I, R) and time t as parameters, and returns a new vector $\vec{V}$ containing the differential equations (S', E', I', R') of equations 2(a)-2(d). We then use SciPy's ODEINT function to solve for this set of differential equations. The initial conditions are S0, E0, I0, and R0, which takes into account the number of vaccinated individuals at t=0. The best-fit α and β are given as parameters, along with the total population *N* and the average recovery rate γ. The result is transposed and the extracted data represents the best-fit infection curve. The sample code below defines the best-fit curve corresponding to Arlington's data from October 2022 through February 2023, using the corresponding best-fit α and β:

```
#Find best fit I(t) for Winter 2022-23 Arlington Data
# Define initial number of exposed, infected, and recovered individuals: E0, I0, R0.
# take into account %60 vaccination
E0_winter22, I0_winter22, R0_winter22 = 0, winter22data[0], 75000
# Define initial number of susceptible individuals.
S0_winter22 = N - E0_winter22 - I0_winter22 - R0_winter22

# Set up initial conditions vector for S0, E0, I0, R0
initial_conditions = S0_winter22, E0_winter22, I0_winter22, R0_winter22
# Create an array of evenly spaced time values(in days)
t = np.linspace(0, 120, 120)
# Solve for the differential equations for the SIR model (in vector form) by integrating over the time array t.
solution_winter22 = odeint(dVdt, initial_conditions, t, args=(N, alpha_fit_winter22, beta_fit_winter22, gamma))
# transpose solution array to extract data
S_winter22, E_winter22, I_winter22, R_winter22 = solution_winter22.T
```

Next, we use the graphing functions of Python's matplotlib library to plot the best-fit infection curve and the real-life curve on the same graph.

Weather covariates were introduced as mean, minimum or maximum value over the estimated transmission period.