COVID-19:
Forecast and Distribution Model

Math 380 Final Project

Hunter Negron and Dan Klompenhower

Table of Contents

1 INTRODUCTION

COVID-19, or Coronavirus as it's known more colloquially, has had a tenacious hold on the world's attention since its emergence in late December 2019. Starting in the eastern Chinese city of Wuhan, cases quickly spread to neighboring Asian nations, and no more than a month later, European and North American countries also reported infections. With the number of cases increasing daily, and the virus still spreading rapidly, the central question of federal governments, health organizations, and the media is what are the most efficient methods by which the pandemic can be subsided, and what can we expect if these methods are implemented successfully? Our analysis was divided into two parts to help get insight into these questions.

First, we attempted to predict the amount of new COVID-19 cases each day. To do this we used time series analysis of different independent 'regions' in order to more effectively account for the local factors playing into the number of new cases each day. We choose our regions to be specific states in the US based on the depth and availability of their data. From here, the problem of prediction becomes a matter of analyzing accurate data, and effectively simplifying the model without removing its key characteristics. We decided to tackle this using the decomposition method of time series analysis, truncating the data into three sections to simplify it without losing any of its characteristics. Then we were able to use our analysis to provide a forecast for one week beyond the data.

Second, using the above forecasts, we determine the proportion of vaccines that should be transferred to each state based on a set of criteria related to the number of new cases in that state per day, the distance that state is from a vaccine manufacturer, and the number of expected new critical cases in those states. This third condition, new critical cases per day, will be related to that state's population density. With these assumptions, the number of administered vaccines changes daily, ultimately decreasing as the rate of infection decreases.

2 PROBLEM STATEMENT

"Suppose that the world medical association announces that soon we can expect to have a new medication could stop the COVID-19 a and cure patients whose disease is not advanced. Build a realistic, sensible, and useful model that considers not only the spread of the disease, the quantity of the medicine needed, possible feasible delivery systems, locations of delivery, speed of manufacturing of the vaccine or drug, but also any other critical factors your team considers necessary as part of the model to optimize the eradication of COVID-19."

The problem is massive and fundamentally difficult. There are so many potential factors in disease spread, medicine production, and medicine delivery. To simplify the model, we chose to create a foundation for further analysis by constraining many of these variables and directly addressing two key problems:

1. Predicting the spread of COVID-19 in several states individually
2. Determining how to distribute vaccines among a system states given a prediction of the new cases each day in each state

First, we attempted to model the spread of the virus using time series analysis. We took daily data on the numbers of new confirmed COVID-19 cases in a given state and used it to predict the amount of new cases in that state every day the following week. We wanted to keep our out-of-the-sample prediction small because the volatility and underlying factors drastically affect the new cases of COVID-19 reported each day.

Second, we took our forecasted new cases and used it to model our distribution of vaccines. Our model of distribution focuses on the distance from a manufacturer, the expected new number of critical cases each day, and the state's population. This model is designed to be flexible; to expand our model we could add more states to the system, add a constraint based on mortality rate over time or add some other constraint which can be derived from the dataset.

3 THE DATASET

3.1 Selection/Characteristics

We used four datasets, one for each of the following states: Minnesota, North Dakota, South Dakota, and Kentucky. Our choice of states was based on the availability of data. These states provided data on the number of recovered cases and did so for an extended period of time. Many states in the US have not made recovered case data easily available. Amid the states that have recovered case data, very few have a reasonable sample size to work with. Due to these restraints we could not consider states of interest such as New York, California and Washington. Geographic factors, population demographics, and political status had no effect on our decision.

Our dataset included the following attributes reported on a daily basis:

- Date
- Total Cases
- Recoveries
- Active Cases
- Deaths
- New Cases

3.2 Preparation

The data originally came from each state's department of health posted on their coronavirus websites. You can access their statistics in the following links:

- Minnesota
- Kentucky
- South Dakota
- North Dakota

Our data for Minnesota, South Dakota, and North Dakota spans from 3/24/2020 to 4/28/2020. For Kentucky, the recoveries data was not available before 4/3/2020 so our data spans from 4/3/2020 to 4/28/2020.

Due to the varying formats, getting the data was by no means seamless. Often, we could not easily download a csv file and had to resort to copying and pasting data out of tables or manually entering it ourselves. From there we created our own csv file for each state in Microsoft Excel adding the dates corresponding to the data and indices. Then to update our datasets to get newly released daily data we had to repeat this process appending the new data to the previous csv file.

4 GROWTH MODEL

4.1 Description of the Model

Given the drastic fluctuation in COVID-19 cases due to new scientific discoveries, government intervention and limited testing, accurately modeling COVID-19 case growth is extremely difficult. We wanted a model that inherently includes most of these features within the data. To do this, we decided to do a time series analysis and create a model for each state, allowing our analysis to account for whatever local or regional factors that could affect the data. The models generally follow the same procedure of analysis, but certain steps involve different interpretations and processes.

We chose to fit our time series of new daily COVID-19 cases using the decomposition method. This involves taking our input data, in the form of a time series, and breaking it down into three main components: trend, seasonality, and residuals. The goal of the method is to fit a trend and seasonality component so that together they encompass most of the change between data points. In an ideal situation, after subtracting the trend and seasonality from the time series we are left with exclusively white noise.

This approach of modeling requires relatively few assumptions initially. The most important assumption being that people confirmed in a given state continue to reside in that state. Throughout the modelling process we must make several additional assumptions that our indicated test results are accurate.

5 ANALYSIS AND TESTING OF THE MODEL

Throughout this section we will use our Minnesota plots as an example, our other plots can be generated with the code provided in the appendix.

5.1 Visualizations

First, we plotted each unmodified time series to get a visualization of what we are working with, which can be seen in Figure 1 below.

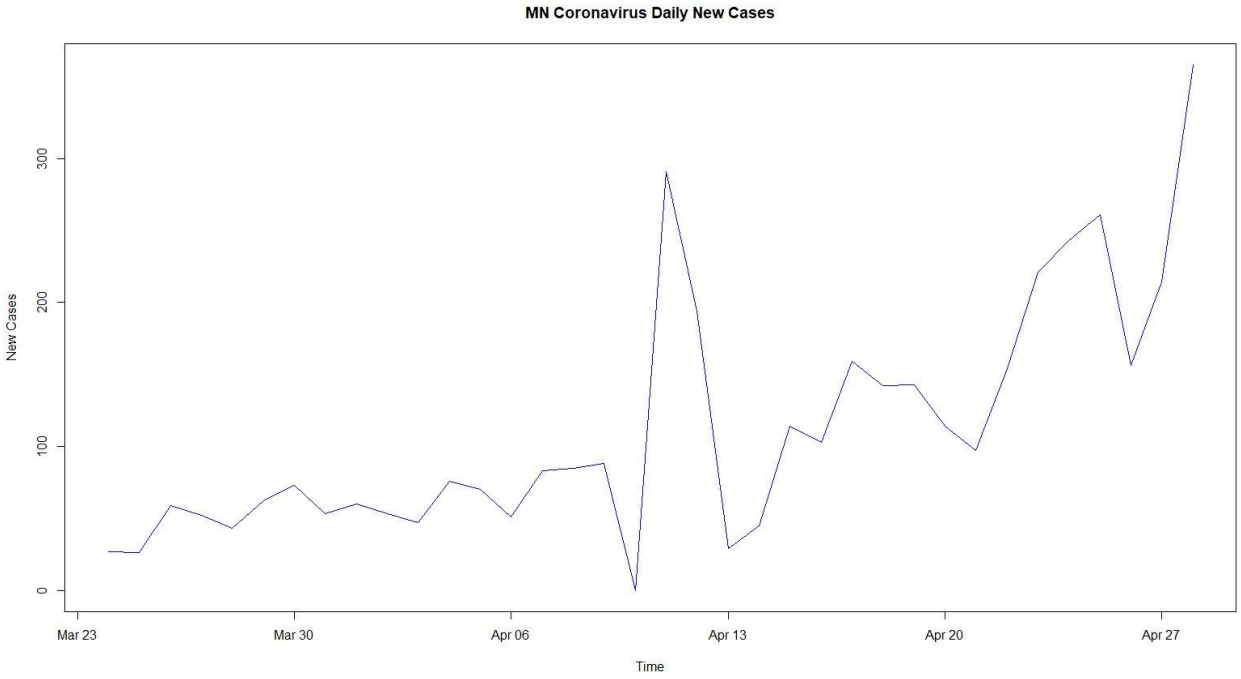

Figure 1: Plot of the Minnesota Daily New COVID-19 Cases time series

Then, we looked at the raw ACF (auto-correlation function) for the time series of new cases. It tells us how each data point is related to the data points. The ACFs (Figure 2 of the MN ACF below) all show a decreasing pattern, suggesting that we have a trend within the data.

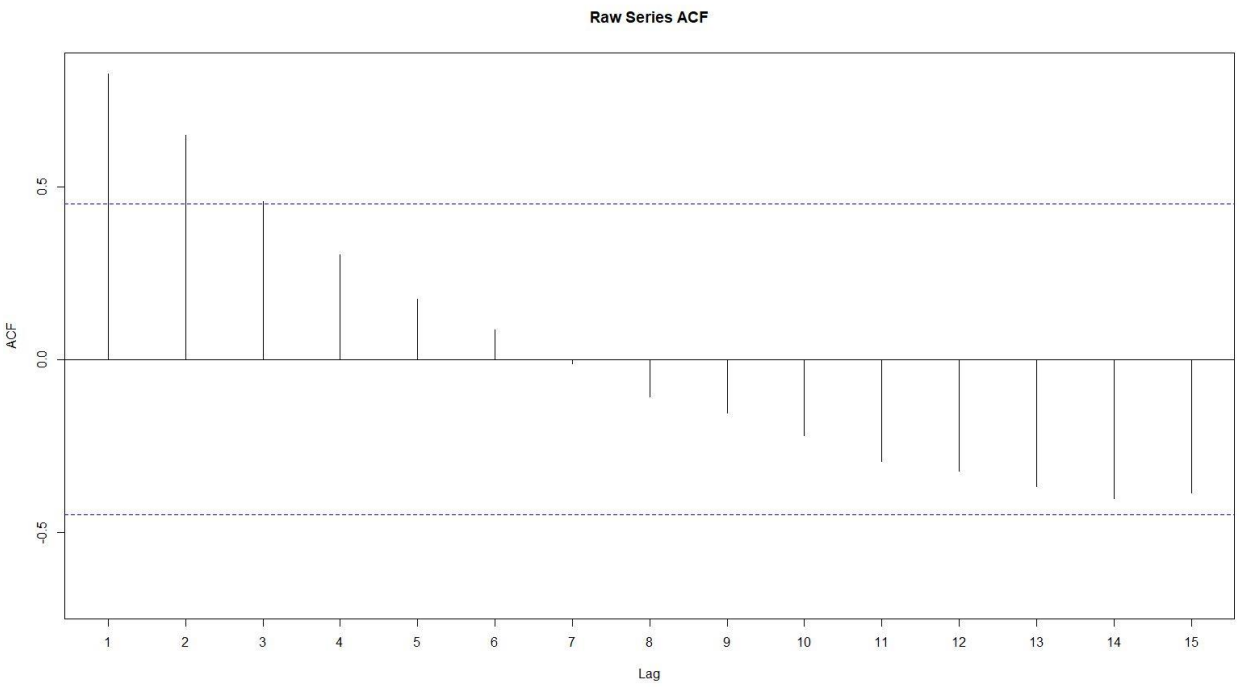

Figure 2: The raw ACF of the Minnesota time series

5.2 Decomposition

5.2.1 Trend

Next, we proceeded to decompose the model, starting with the trend. In the first step, we needed to determine which order of trend line to fit (linear, quadratic, etc.). We limited our search to linear and quadratic to prevent overfitting. From a macro-perspective, we expect a quadratic trend line to perform better as cases will swell up to a peak then decline over time as the disease is contained or eradicated. We used the trend() function from the 'itsmr' package to get a linear and quadratic trend line. Note the trend() function computes least squares regression on the data to fit the trend line. Figure 3 shows the linear and quadratic trend lines compared to the data.

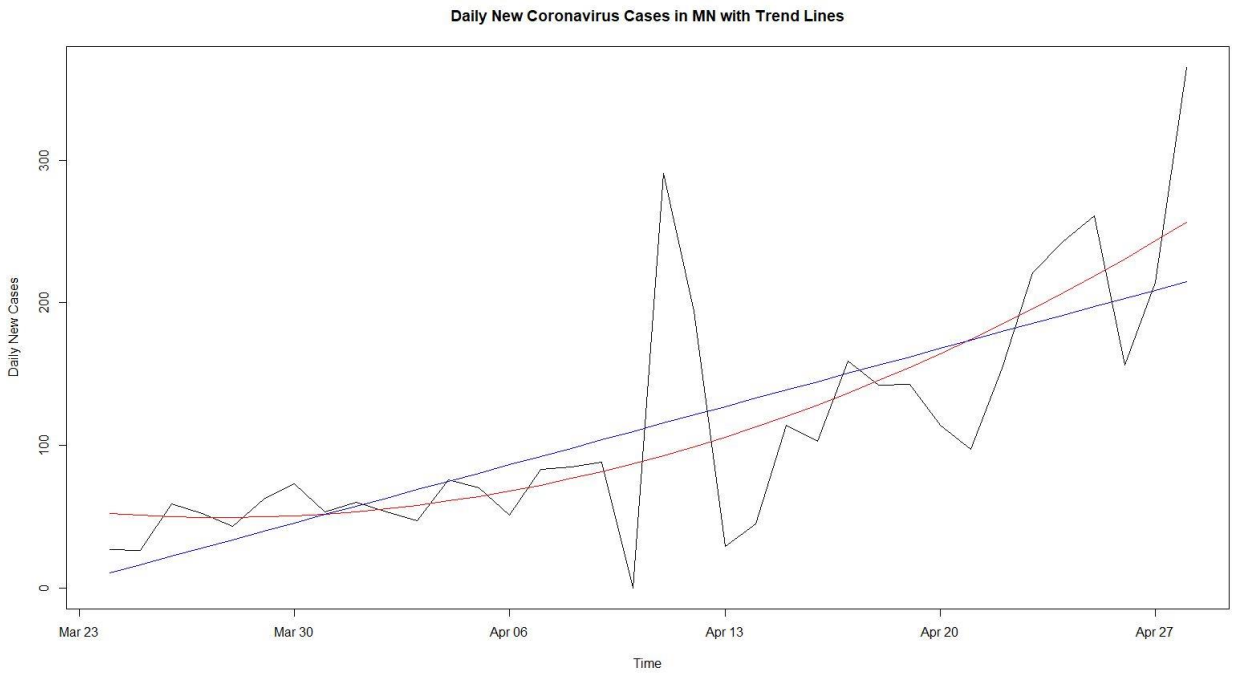

Figure 3: Plot of the linear and quadratic trend lines over the Minnesota time series

To see which is better we computed each trend line's mean squared error (MSE) with the data and took the square root of it. In every state the quadratic trend line was slightly better than the linear trend line. In our Minnesota model, the square root of the MSE for the quadratic trend line was 319.53 compared to the linear trend line's 341.90.

5.2.2 Seasonality

Now that we have a trend, we need to find the seasonal component of the time series. To do this we used itsmr's season() function. We chose a period of 7 so that our seasonality represents a week in real time. Below is Figure 4, a plot of our seasonal component.

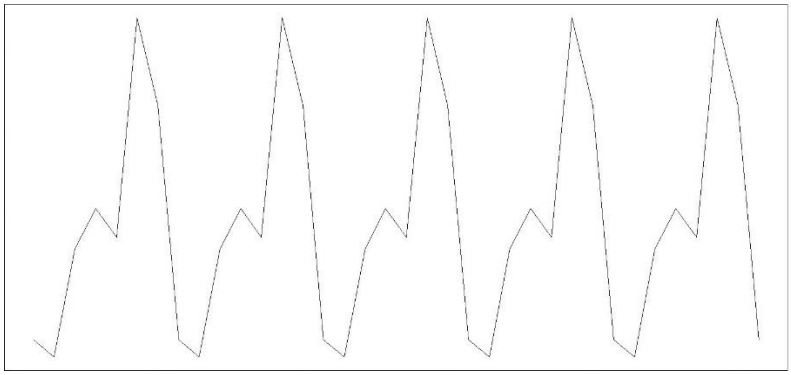

Figure 4: Plot of the seasonal component of the Minnesota time series over time

Additionally, in Figure 5 we plotted the season and trend components added together to visualize how well they encapsulate the data.

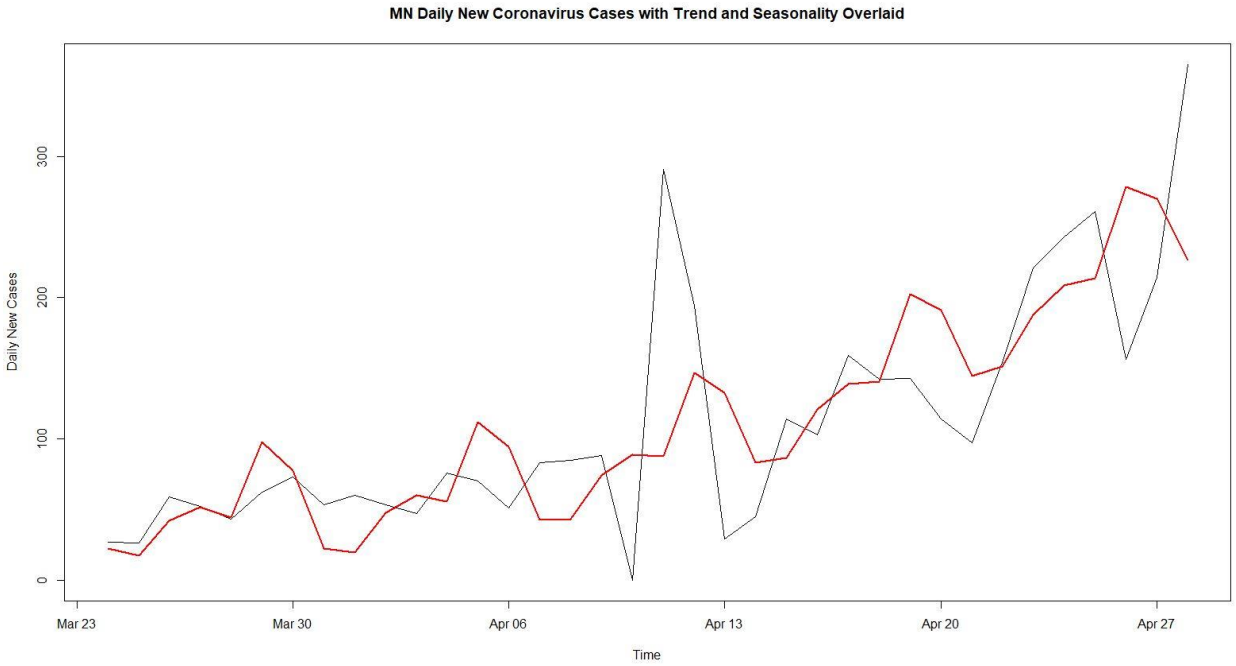

Figure 5: The seasonal and trend components together compared to the Minnesota time series

5.2.3 Residuals

Since we have decomposed the trend and seasonality of the time series, we start with the time series and subtract the trend and seasonal components to get our residuals. Now we can test our residuals to see if they are IID (independent and identically distributed) white noise. We used itsmr's test() function which performs several different tests with the null hypothesis that our residuals are IID noise. These tests gave conflicting results, however, for each state the majority of tests concluded that we could not reject the null hypothesis. Hence, we assume that residuals are IID noise. Below in Figures 6 and 7 are some of our test results.

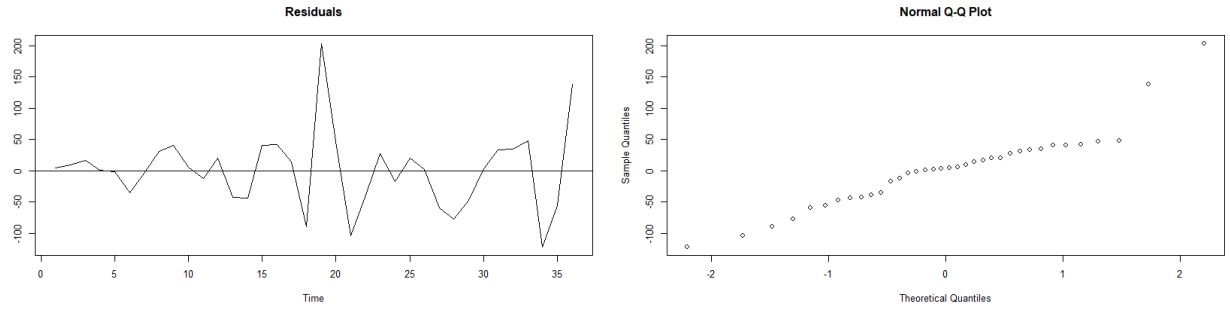

Figure 6: A plot of the residuals over time (left) and a Q-Q plot (right)

```
Null hypothesis: Residuals are iid noise.
Test                     Distribution Statistic   p-value
Ljung-Box Q                Q ~ chisq(20)     27.93     0.111
McLeod-Li Q                Q ~ chisq(20)     12.46    0.8993
Turning points T   (T-22.7)/2.5 ~ N(0,1)        16    0.0068 *
Diff signs S       (S-17.5)/1.8 ~ N(0,1)        19     0.393
Rank P             (P-315)/36.7 ~ N(0,1)       307    0.8275
>
```

Figure 7: Results of our test to see if the residuals are IID noise

Stationarity

After ensuring the residuals are IID noise the next step is to test if they are stationary. If a time series is stationary its properties are not dependent on time. To test for stationarity we used the augmented Dickey-Fuller test via the adf.test() function from the 'tseries' package. The tests for Minnesota and North Dakota indicated that we could reject the null hypothesis that the series was nonstationary at the 5% significance level. The tests for Kentucky and South Dakota indicated that the series was non-stationary and provided a lag order. Figure 8 shows the output of the adf.test() function.

```
        Augmented Dickey-Fuller Test

data:  residuals
Dickey-Fuller = -3.5677, Lag order = 3, p-value = 0.04941
alternative hypothesis: stationary
```

Figure 8: Augmented Dickey-Fuller Test results

5.3 Model Fit

Finally we fit the models using the auto.arima() function from the 'forecast' package to find the ARIMA model with the lowest AICC value for each state. AICC is the bias-corrected version of the AIC (Akaike Information Criterion) which is an estimator of out-of-sample prediction errors. One benefit of using this function to fit our model is that it prints each model tested along with its AICC value so we can also test similarly performing models if desired. Additionally, we were able to specify if the time series was stationary or non-stationary as an argument to automatically be considered. Here are the best fitting models for each state:

Minnesota	ARIMA(2,0,0) with zero mean
Kentucky	ARIMA(1,0,0) with zero mean
South Dakota	ARIMA(0,0,3) with zero mean
North Dakota	ARIMA(0,0,0) with zero mean, i.e. a white noise model

6 RESULTS AND QUALITY OF THE MODEL

6.1 Forecasting

To forecast our chosen models beyond the dataset we used the forecast() function from itsmr. We input the model we are using, the data, and the forecast length to get an output of the predictions, a 95% confidence interval, and the standard error for each forecasted value. Below we see the PLOT of the forecast for Minnesota from April 29th to May 6th. The other forecasts can be viewed in the Results & Conclusion section.

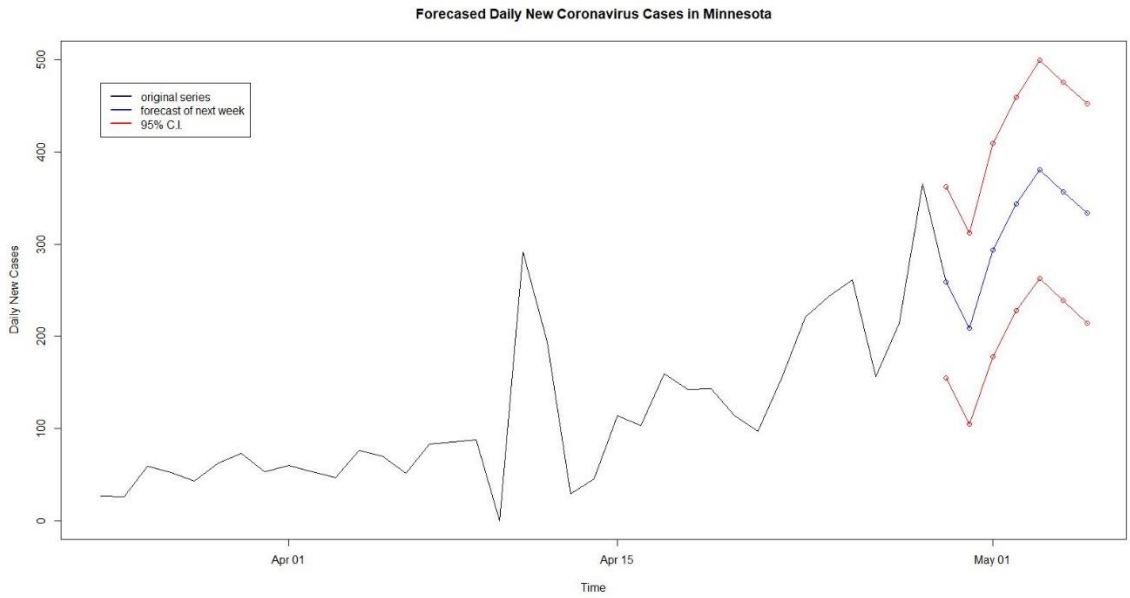

Figure 9: Minnesota forecasted daily new COVID-19 cases

6.2 Improvements

The biggest concern we had with our models was validation. To improve our model, we would like to do k-folds validation. That means truncating our dataset into independent 'folds' then forecasting from each fold using a model we seek to learn more about. Then we can compute the mean squared error of each fold's prediction and have a quantitative method to see how effective our model would have been at forecasting past data. Given we only had about 30 data points to work with, k-folds validation is sketchy as our folds used for predicting would be so small relative to their forecast size. Furthermore, our data typically had several "outliers" (based on appearance not a test) which would more negatively affect the model results in a smaller sample.

A different method worth exploring, especially for dealing with these outliers, is smoothing the data using a low pass filter. We can apply this smoothing method to filter out the most extreme values of our data. To test its value, we'd like to use k-folds, with as few as one fold if necessary. We then can compare the 'best' fitting model and a smoothed version of it directly. If we had more data to work with, we could also adjust the constant determining the proportion of data we filter. We would need to be careful in adjusting this constant as with only one or two folds it would be easy to unknowingly overfit the data.

7 DISTRIBUTION MODEL

7.1 Description of the Distribution Model

Once it was decided that this model would essentially be the composition of two smaller models, we had to determine the scope of each one. It seemed suitable at first to incorporate methods of graph theory into our vaccine distribution model since it involved efficiently connecting nodes and counting the paths taken to travel between them. Unfortunately, we could not pursue this because the amount of available data on medical and distribution networks is limited, so we altered our approach to something more feasible. Specifically, we agreed to examine solely the optimal amount of vaccine that each state should receive instead of the means by which they were shipped.

Naturally, some key assumptions were made in the framework of our vaccine distribution model. For example, we assume that the vaccines are being produced in one central facility. Making this assumption eliminates the need for considering routes and nodes in transportation, which would otherwise be a much more complicated problem. We also assume that each state receives their calculated proportion of vaccines daily, near the end of that day. That is, a shipment from the central factory is made once per day, and the quantity is determined by several pieces of data from that same day. Lastly, we assume that the central manufacturer is producing at maximum efficiency and this rate of production is a constant. Limiting our results strictly to population data instead of economic factors seemed like a more realistic task.

One of our earlier ideas was to examine county data and base our model off of results in local communities within a single state. However, not all counties record or publish their data, so this process had to be replaced with modeling state data, which was more accessible, reliable, and holistic. This led to our first real problem: solving a linear program that attempts to optimize the proportion of vaccines that each state receives under a set of mathematical constraints.

In this first version of the program, the objective function to be maximized

$$(1)\ f(x_1, x_2, x_3, x_4) = x_1 \frac{c_1}{I_1} + x_2 \frac{c_2}{I_2} + x_3 \frac{c_3}{I_3} + x_4 \frac{c_4}{I_4}$$

was subject to the following constraints:

$$(2)\ x_1 + x_2 + x_3 + x_4 = 1$$

$$(3)\ x_1, x_2, x_3, x_4 \geq 0$$

$$(4)\ x_i \leq 0.2 * \frac{P_i}{P}\ \forall\ states\ i$$

$$(5)\ x_i \leq \frac{P_i}{P}\ \forall\ states\ i$$

Where c_i is the number of *critical* cases recorded state i, I_i is the total number of infections in state i, P_i is the population of state i, P is the total population across all states, and x_i are the

decision variables representing the proportion of vaccines that should be sent to state i. The reasoning for these constraints is explained in the next section.

To our disappointment, this model was unsuccessful in maximizing the decision variables. The constraints were too specific, and this meant that each one could not be satisfied simultaneously. Additionally, we found that the percentage of critical cases was not consistently reported in each state, meaning we could no longer rely on that as one of our variables. Instead, however, we found a national percentage, averaging over all states, to be roughly 5% of total infections. We use fact in a later iteration.

Our next approach was to ease the detailed nature of the constraints, as well as modify the objective function to not include *critical* cases. The improved function for maximization was

$$(6)\ f(x_1, x_2, x_3, x_4) = x_1 \frac{c_1}{I_1} + x_2 \frac{c_2}{I_2} + x_3 \frac{c_3}{I_3} + x_4 \frac{c_4}{I_4}$$

Paired with the new, less demanding, constraints

$$(7)\ x_1 + x_2 + x_3 + x_4 = 1$$

$$(8)\ x_1, x_2, x_3, x_4 \geq 0$$

$$(9)\ x_i \geq \frac{r_i}{C}\ \forall\ states\ i$$

Where r_i is the number of *non-critical* (regular) cases in state i, P_i is the population of state i, C is the total number of cases across all states, and x_i are the decision variables representing the percentage of vaccines that should be sent to state i. For clarity, the relationship among the infection variables is $c_i + r_i = I_i$ and $\sum_{i=1}^{n} I_i = C$. That is, the number of critical cases plus the number of regular cases equals the total number of infections in each state, and the sum of all these infections equals the total number of cases across all states (with n being the number of states in the model).

For obvious reasons, the first two constraints remained unchanged. The total number of distributed vaccines must in fact equal the total number produced (the factory cannot hoard them) and these numbers must be positive (the states cannot issue out their own homemade formulas to other states). Also note that instead of using P in this model's constraints (population across all states), we used C, the total infections across all states, since this seems more relevant to the question of how vaccines ought to be handled.

While this model was capable of satisfying all imposed constraints while also returning realistic outputs, it found that the optimal solution was to distribute the vaccines according to the ratio of each state's cases to the total number of cases, $\frac{r_i}{C}$. We also tried making the ratio in the final constraint equal to the number of *new cases* to the total number of cumulative cases. These results weren't particularly exciting or insightful, so we decided we could do better. But instead of changing this existing model, we merely add to it.

One of our concerns was each state's distance from the manufacturer. This was important to consider because states nearer to a factory are more likely to receive urgent assistance in the case of an unexpected outbreak, whereas states farther away would be at risk of delayed assistance in similar events. Introducing the distance variable makes things more interesting, starting with the assumptions. First, for distance between states, we use the distance between the most populated cities in each state. This was a reasonable approximation since more crowded cities are where the highest concentration of cases are likely to be found. This is also where more shipping facilities and hospitals will be located for receiving and using the vaccines. We also assume that the vaccine manufacturer is ideally located (this will be explained more in the analysis section).

We introduce an important new variable $g_i = \frac{D_i}{D} - \frac{1}{n}$ where D is the sum distance between each state and the facility, D_i is the individual distance between state i and the facility, and n is the number of states as it was before. Notice that $\frac{D_i}{D}$ is a "traveling distance weight" in the sense that it increases for states farther away but remains normalized due to the factor of $\frac{1}{D}$. The purpose of this variable is twofold: it allows us to adjust the amount of vaccine each state received (according to the results above) by an amount *proportional* to g_i and it ensures that the total amount of vaccine produced and sent out is still equal. That is, $\sum_{i=1}^{n} g_i = 0$. Using this new value in our revised model makes it such that distance is considered in our output, and the results are not simple ratios of statistical values used in the constraints.

The new amount of vaccine sent to each state will then be calculated using the results from the linear program above by: $x_i + 0.1g_i$ where 0.1 is our scaling factor. As can be seen, the results of the previous model are still used, but an additional term is also calculated. This is what we meant when we said the original model wasn't changed, only appended to.

7.2 Analysis of the Distribution Model

The final model produced from the thought process consists of a basic linear program that optimizes the number of pharmaceuticals that ought to be delivered to each state each day by considering the number of new cases and population of the state. This process produces a set of percentages which describe how the daily production of vaccines is distributed. These levels are then adjusted when considering the distance that every state is away from the manufacturing facility.

Naturally, the inputs to this model are exactly the data that is produced by the disease growth model that is explained in detail above. Along with the "Predicted Number of Cases" and "State Population" columns of the resulting dataset, the "Distance" input to our model is a set of measurements taken between the vaccine manufacturing facility and the most populous cities of the states to which the vaccine needs to be delivered. Given these inputs, the model is then run using the algorithm that is outlined below to produce the output. The output consists of a set of n percentages, where n is the number of states, which, as stated above, are the amount of daily production given to each state.

The algorithm that produces these results is as follows.

1. Get the output from the disease growth model as a collection of datasets, each of which represents one state, wherein the columns are the "Predicted Number of Cases" and "State Population" and the rows represent each day.
2. For each day in the dataset, perform the following steps:
3. For each state i, calculate the new cases ratio, which is the number of new cases in state i divided by the total number of new cases across all states. These will be used in constructing the constraints for the linear program.
4. For each state i, calculate the severity by dividing the number of new cases by the population of that state. These will be used as the coefficients in the objective function.
5. Solve the linear program, as described in detail in the previous section by using the results of the calculations from steps 3 and 4. The utility that we used for this was the linprog() function that SciPy provides in its Optimize library.
6. The results of step 5 now represent the distribution of the vaccine *without* consideration for distance.
7. Calculate the traveling distance weight for each state by dividing the distance that the state is away from the manufacturing facility by the sum of all distances and subtracting $\frac{1}{n}$, where n is the number of states. Then multiply this by the scaling factor of 0.1. By nature of this computation, these weights will sum to 0. This ensures that when the original distribution results are adjusted, they still represent 100% of the vaccines produced.
8. Add the results from step 7 to the results for step 5. This adjusts the distribution so that it is more fair to those states that are located further away from the vaccine manufacturing facility.
9. The results of step 8 now represent the distribution of the vaccine *with* consideration for distance.

It is important to note how flexible this algorithm is with the scope of its results. For example, we only applied it to four states as a proof of concept, but the general idea can be applied to the entire country. It can be easily extended to include multiple vaccine producers. This would be a natural next step to explore with this model. Other assumptions that offer interesting avenues for future analysis include the effect of social distancing, population density, and variations to the percent of cases that are deemed "critical". These factors would play a large part in the decision to send some states more or less vaccines when needing to control this limited resource.

8 RESULTS & CONCLUSION

Using our times series models we developed forecasts for each state seen in the figures below.

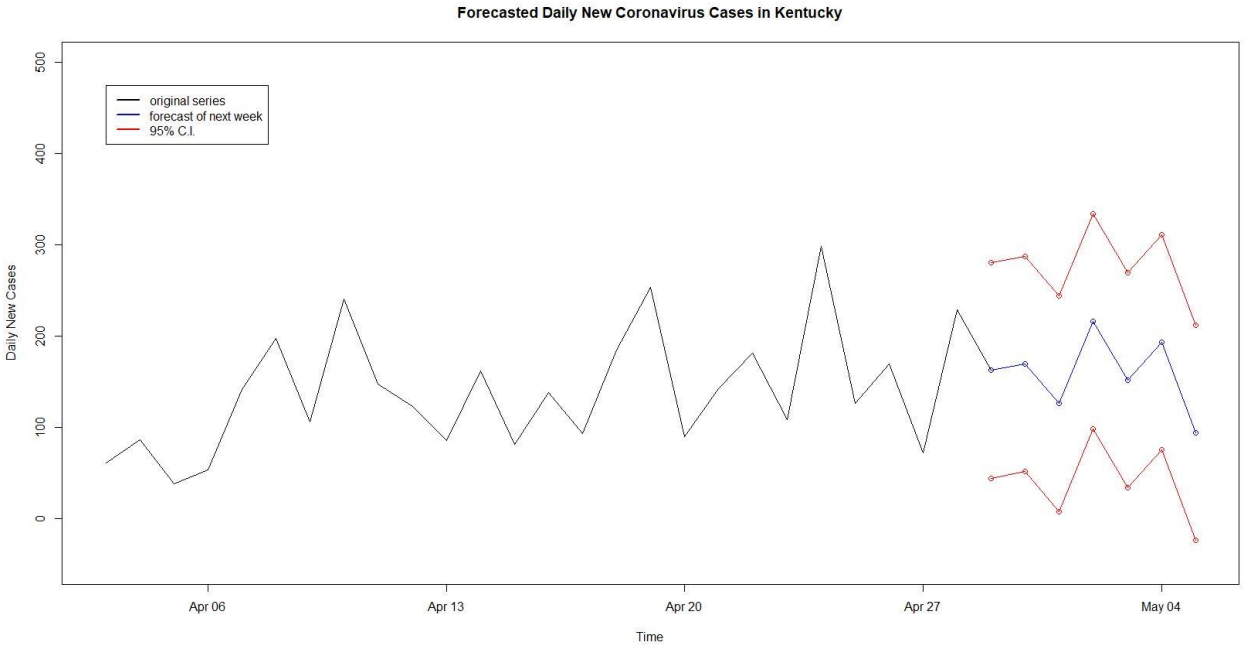

Figure 10: Forecast of daily new COVID-19 cases in Kentucky

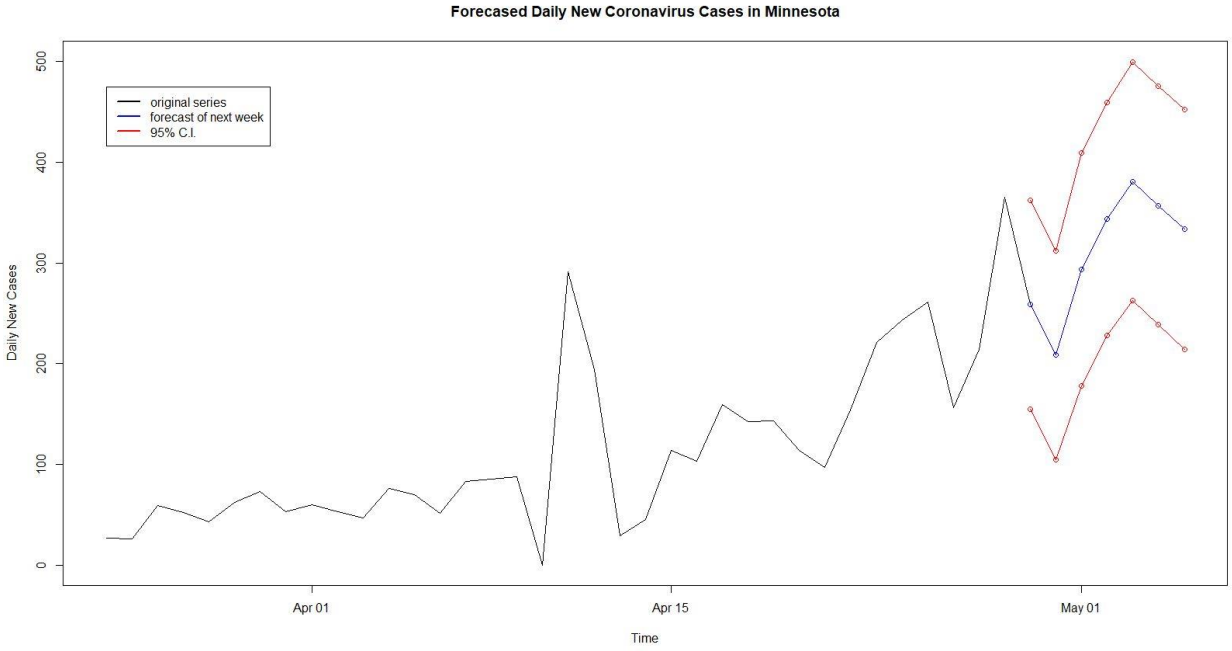

Figure 11:Forecast of daily new COVID-19 cases in Minnesota as seen previously

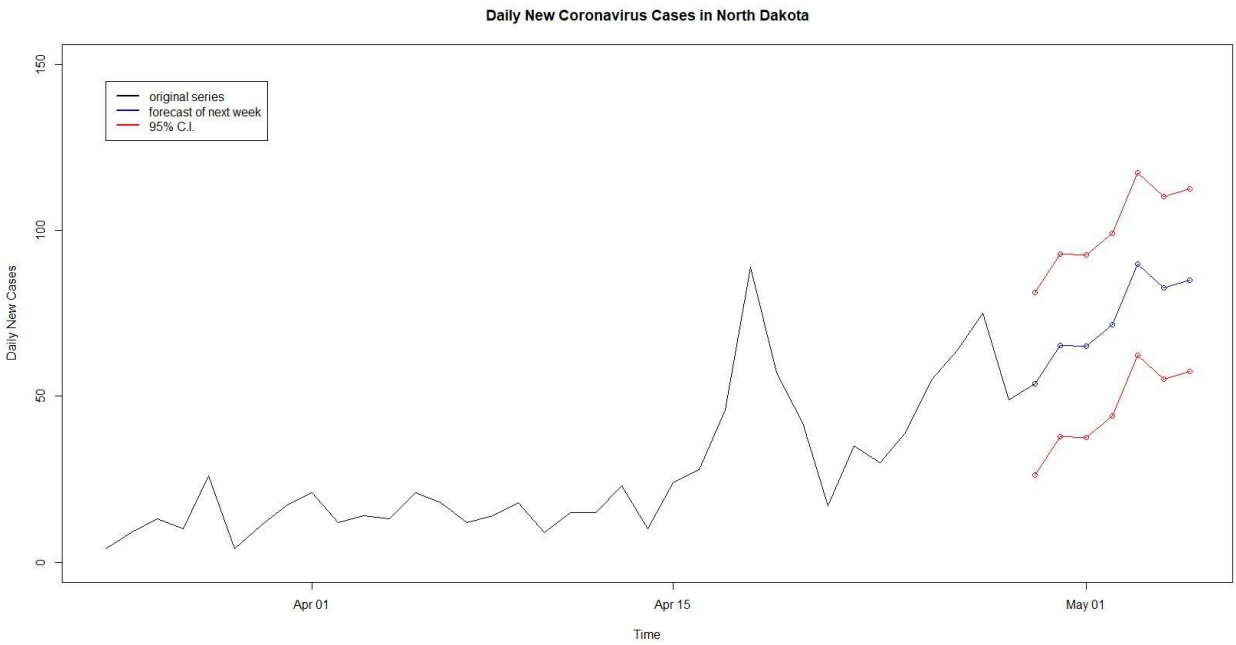

Figure 12: Forecast of daily new COVID-19 cases in North Dakota

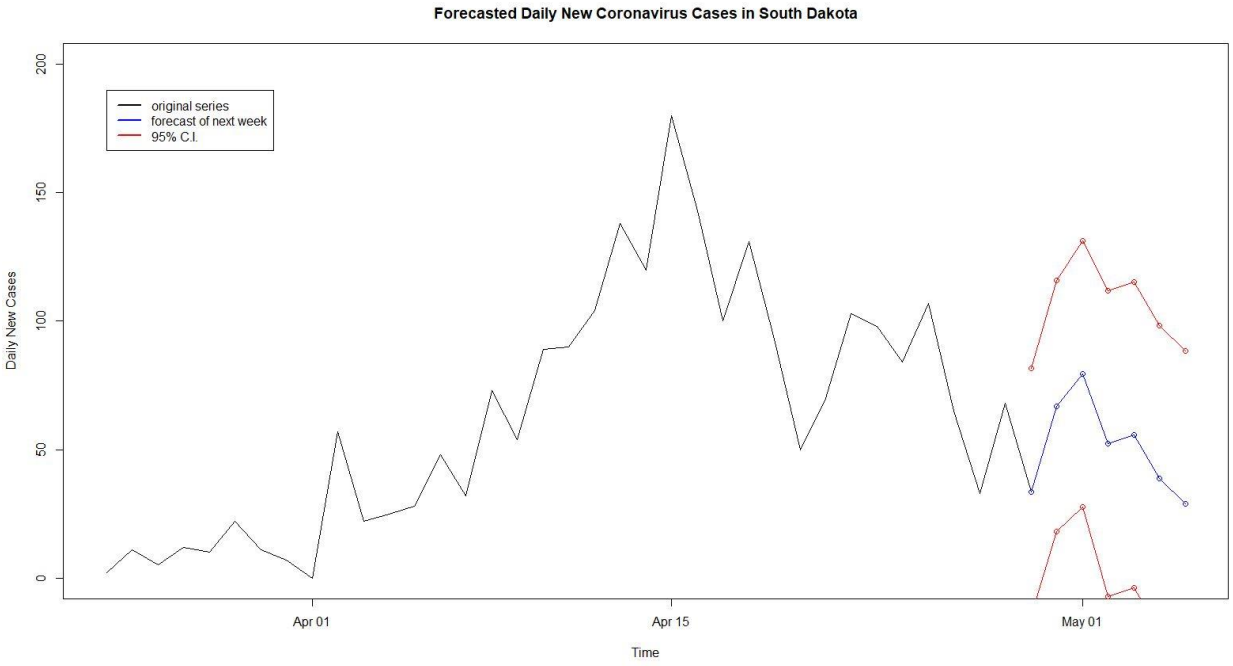

Figure 13: Forecast of daily new COVID-19 cases in South Dakota

In our predictions we get large ranges for our 95% confidence intervals. These ranges should still be carefully considered as COVID-19 cases are so volatile and many unexpected factors could cause large spikes in new cases. For example, a state could gain access to 10,000 new test kits and begin administering those tests in a very short window. This would cause large deviations from our projection. While our model using limited information can be a shortcoming, it can also

add value. In its current state our model can predict new cases based solely on the previous data of new cases. We can derive this information from the total cases reported each day. Hence, we are able to provide a simple projection for most areas affected by COVID-19. Even if our model is not perfect, by using the most recent data we can minimize our shortcomings dealing with irregularities and provide reasonable estimates of how many new cases there will be in the next week. If states provide more information, in the future we can expand our model to provide more accurate estimates and distributions constrained around other factors such as mortality rate or positive test rate.

In terms of the quality of the vaccine distribution model, it is reasonable to conclude that these results make sense given the context of many separate states all needing access to a limited resource. Given the nature of a pandemic, some areas of the country, particularly those with a high spread rate relative to population, are more likely to need the assistance of a vaccine to control that spread. Other areas, however, may be located a considerable distance away from the place where the vaccine is being manufactured. For example, not every state may have access to the necessary resources to produce the vaccine themselves. Therefore, it is reasonable to assume that vaccines are produced at one or a handful of centers and therefore divided intelligently among the states. This means that the distance between the manufacturer and the hospitals at which the vaccines will be utilized will vary greatly across the country. Therefore, this must be considered when calculating an optimal distribution. Those farther away should get a slight increase in order to provide a buffer. Those closer have the advantage of nearly instantaneous response if an unexpected scenario occurs which requires such action. In Figure 14 on the following page, notice the differences between the models that do and do not consider this factor of distance.

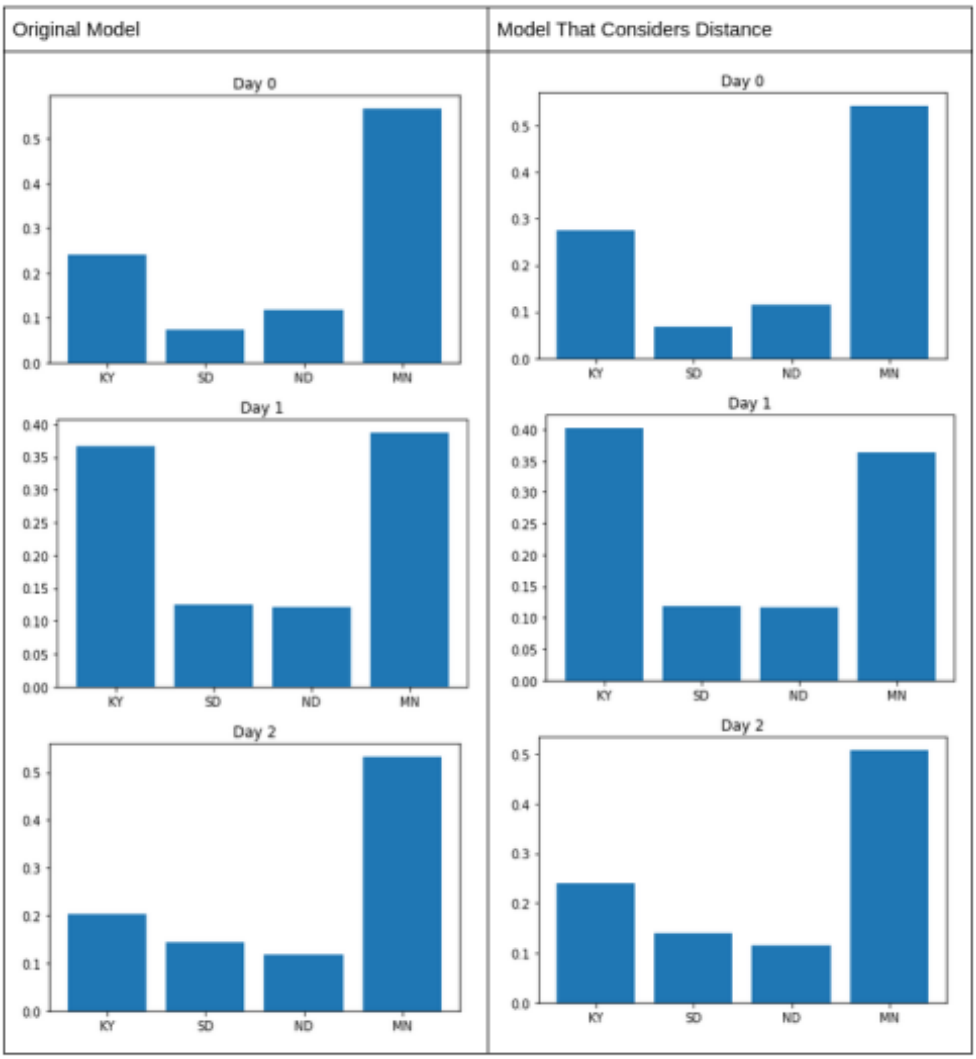

Figure 14: Histograms comparing how vaccines are distributed daily with and without distance from manufacturer as a factor.

Beyond the scope of our models that have been discussed here, there are other assumptions and modeling approaches that would be interesting to explore further. First of all, the data retrieval was extremely time consuming. The data for each state had to be manually copied off of individual webpages and there was no consistency in the formatting, which made modeling the entire country infeasible. Any alternative steps or expansions on our ideas would require a more streamlined approach to data processing. Secondly, we did not consider any data from medical testing. It would be interesting to consider the ratio of new cases in one day to the number of tests performed in one day. Using this information, we would then normalize the data for predictions and then factor the test results back in at the end of the analysis. Alternatively, the ratio of new cases to tests performed gives good insight into the severity of the pandemic in different areas, which can be a useful consideration in the model for vaccine distribution. Of course, the limitations of test kits must be accounted for in order for the results to be meaningful.

9 REFERENCES

"Coronavirus Cases." North Dakota Department of Health, May 1, 2020. https://www.health.nd.gov/diseases-conditions/coronavirus/north-dakota-coronavirus-cases.

"Coronavirus Updates And Information." South Dakota Department of Health, May 1, 2020. https://doh.sd.gov/news/Coronavirus.aspx.

"COVID-19 Map." Johns Hopkins Coronavirus Resource Center, 2020. https://corona virus.jhu.edu/map.html.

Henderson, Max. "Covid Act Now Model Reference/Assumptions." Covid Act Now, April 12, 2020. https://data.covidactnow.org/Covid_Act_Now_Model_References _and_Assumptions.pdf.

"Kentucky COVID-19." Kentucky Department of Health, May 1, 2020. kycovid19.ky.gov.

"Modeling COVID-19 Spread vs Healthcare Capacity." Shiny Apps. All Hill. Accessed May 1, 2020. https://alhill.shinyapps.io/COVID19seir/.

"Preliminary Estimates of the Prevalence of Selected Underlying Health Conditions Among Patients with Coronavirus Disease 2019." CDC.gov. Centers for Disease Control and Prevention, April 2, 2020. https://www.cdc.gov/mmwr/volumes/69/wr/ mm6913e2.htm.

Sanderson, Grant. "Simulating a Pandemic." YouTube. 3Blue1Brown, March 27, 2020. https://youtu.be/gxAaO2rsdIs.

"The SIR Epidemic Model." SciPython. Accessed May 1, 2020. https://scipython.com/ book/chapter-8-scipy/additional-examples/the-sir-epidemic-model/.

"Situation Update for Coronavirus Disease 2019." Minnesota Department of Health, May 1, 2020. https://www.health.state.mn.us/diseases/coronavirus/situation.html

10 APPENDIX (REFERENCES, CODE, PLOTS ETC)

10.1 R Code

```
# import package libraries
library(tidyverse) # for reading in data
library(dplyr) # for data manipulation/selection
library(itsmr) # for testing iid noise and forecasting
library(stlplus) # for decomposing and plotting decomposition components
library(tseries) # for stationarity testing
library(forecast) # for better ACF/PACF and auto.arima


# Reads in data
data=read_csv("KY_Data.csv")

# Creates an array of the active cases
new_cases = data[,5]

# Makes the active cases into a time series
new_cases.ts <- ts(data = new_cases, start = 1, end = 26, frequency = 1)

# Makes an array of dates to use as an x-axis
dates <- seq(as.Date(data$Date[1], format = "%Y-%m-%d"),
as.Date(data$Date[26], format = "%Y-%m-%d"), by = "days")
plot(dates,data$New_Cases, type="l",col=1,main="Daily New Kentucky Coronavirus
Cases", ylab='New Cases', xlab = 'Time')

#look at acf and pacf of raw time series
Acf(active_cases.ts,lag.max = 15,main="Raw Series ACF")
Pacf(active_cases.ts,lag.max = 15,main="Raw Series PACF")

### Decomposethe Model with ITSMR ###

#Fit a linear and quadratic trend to the time series
lintrend=itsmr::trend(data$New_Cases,1)
quadtrend=itsmr::trend(data$New_Cases,2)
#Plot them both on the same graph
plot(dates,new_cases.ts, type='l', main='Daily New Coronavirus Cases in KY
with Trend Lines',xlab='Time',ylab='Daily New Cases')
lines(dates,quadtrend, col=2)
lines(dates,lintrend, col=4)

#Test how accurate the linear trend is by calculating the MSE
linMSE=lintrend-new_cases.ts
linMSEsquared = linMSE**2
root_linMSE = sqrt(sum(linMSEsquared))
root_linMSE

#Test how accurate the quadratic trend is by calculating the MSE
quadMSE=quadtrend-new_cases.ts
quadMSEsquared = quadMSE**2
root_quadMSE = sqrt(sum(quadMSEsquared))
root_quadMSE

#Decomposition resumes now after determining our trend line.
season=itsmr::season(x=data$New_Cases,d=7)
plot(dates, season, type='l')
```

```
residuals=new_cases.ts-quadtrend-season

#Plot of trend + seasonality vs Data
ts = quadtrend + season
plot(dates,new_cases.ts,type='l',main='KY Daily New Coronavirus Cases with
Trend and Seasonality Overlaid',xlab='Time',ylab='Daily New Cases')
lines(dates,ts, col='red',lwd=2)
#Tests if the residuals are IID noise
test(residuals)
#We conclude that the residuals are IID noise as we cannot disprove the null
hypothesis

#Making an array of the trend/season to use later when we forecast.
M=c('season',7,'trend',2)

#Augmented Dickey-Fuller Test to see if the residuals are stationary (not time
dependent)
tseries::adf.test(residuals,alternative = 'stationary')

#Check how the settings change the outcome
auto.arima(residuals,trace=T,approximation = F,stepwise = F, stationary = F)

#Forecast using our model determined by the lowest AICC and best fitting
trend/seasonality
cast=itsmr::forecast(new_cases.ts,M=M,a=itsmr::arma(residuals,1,0),h=7,opt=2)

#Created for plotting the forecast
dates2 = seq(as.Date('2020-04-29'),as.Date('2020-05-05'),by="days")

#Plot of the forecast and data
plot(c(dates, dates2), c(new_cases.ts, cast$pred), type='l', main= 'Forecasted
Daily New Coronavirus Cases in Kentucky', ylab = 'Daily New Cases', xlab =
'Time', ylim = c(-50,500))
lines(dates2, cast$pred, col='blue', type='o')
lines(dates2, cast$l, col='red',type='o')
lines(dates2, cast$u, col='red',type='o')
legend(x=dates[1],y=475,legend = c('original series','forecast of next
week','95% C.I.'),col=c(1,4,2),lty=c(1,1,1),lwd=c(2,2,2))
cast
```

```
# Import package libraries
library(tidyverse) # for reading in data
library(dplyr) # for data manipulation/selection
library(itsmr) # for testing iid noise and forecasting
library(stlplus) # for decomposing and plotting decomposition components
library(tseries) # for stationarity testing
library(forecast) # for better ACF/PACF and auto.arima


# Reads in data
data=read_csv("MN_Data.csv")

# Creates an array of the active cases
new_cases = data[,5]

# Makes the active cases into a time series
new_cases.ts <- ts(data = new_cases, start = 1, end = 36, frequency = 1)

# Makes an array of dates to use as an x-axis
dates <- seq(as.Date(data$Date[1], format = "%Y-%m-%d"),
as.Date(data$Date[36], format = "%Y-%m-%d"), by = "days")

# Plots the overall time series vs the index as well as the mild, severe, and
critical cases
plot(x=dates,y=data$New_Cases, xlab='Time',type="l",col='blue',main="MN
Coronavirus Daily New Cases", ylab='New Cases')

#look at acf and pacf of raw time series
Acf(active_cases.ts,lag.max = 15,main="Raw Series ACF")
Pacf(active_cases.ts,lag.max = 15,main="Raw Series PACF")


### Decomposethe Model with ITSMR ###

#Fit a linear and quadratic trend to the time series
lintrend=itsmr::trend(data$New_Cases,1)
quadtrend=itsmr::trend(data$New_Cases,2)
#Plot them both on the same graph
plot(dates,new_cases.ts, main='Daily New Coronavirus Cases in MN with Trend
Lines',xlab='Time',ylab='Daily New Cases',type='l')
lines(dates,quadtrend, col=2)
lines(dates,lintrend, col=4)

#Test how accurate the linear trend is by calculating the MSE
linMSE=lintrend-new_cases.ts
linMSEsquared = linMSE**2
root_linMSE = sqrt(sum(linMSEsquared))
root_linMSE

#Test how accurate the quadratic trend is by calculating the MSE
quadMSE=quadtrend-new_cases.ts
quadMSEsquared = quadMSE**2
root_quadMSE = sqrt(sum(quadMSEsquared))
root_quadMSE
```

```
#The quadratic trend has a lower root MSE so we will use it for our model

#Decomposition resumes now after determining our trend line.
season=itsmr::season(x=data$New_Cases,d=7)
plot(dates, season, type='l', xlab = 'Time', ylab= 'New Cases', main =
'Minnesota New Case Seasonality')
residuals=new_cases.ts-quadtrend-season

#Plot of trend + seasonality vs Data
ts = quadtrend + season
plot(dates,new_cases.ts,type='l',main='MN Daily New Coronavirus Cases with
Trend and Seasonality Overlaid',xlab='Time',ylab='Daily New Cases')
lines(dates,ts, col='red',lwd=2)

#Tests if the residuals are IID noise
test(residuals)
#We conclude that the residuals are IID noise as we cannot disprove the null
hypothesis

#Making an array of the trend/season to use later when we forecast.
M=c('season',7,'trend',2)

#Augmented Dickey-Fuller Test to see if the residuals are stationary (not time
dependent)
tseries::adf.test(residuals,alternative = 'stationary')

#Smoothing functions to expand upon later
smoothed_residuals=smooth.exp(residuals,.75)
plot(residuals, type ='l')
lines(smoothed_residuals)

#Check how the settings change the outcome
auto.arima(residuals,trace=T,approximation = F,stepwise = F, stationary =
TRUE)

#Forecast using our model determined by the lowest AICC and best fitting
trend/seasonality
cast=itsmr::forecast(new_cases.ts,M=M,a=itsmr::arma(residuals,2,0),h=7,opt=2)

#Created for plotting the forecast
dates2 = seq(as.Date('2020-04-29'),as.Date('2020-05-05'),by="days")

#Plot of the forecast and data
plot(c(dates, dates2), c(new_cases.ts, cast$pred), type='l', main= 'Forecased
Daily New Coronavirus Cases in Minnesota', ylab = 'Daily New Cases', xlab =
'Time', ylim = c(0,500))
lines(dates2, cast$pred, col='blue', type='o')
lines(dates2, cast$l, col='red',type='o')
lines(dates2, cast$u, col='red',type='o')
legend(x=dates[1],y=475,legend = c('original series','forecast of next
week','95% C.I.'),col=c(1,4,2),lty=c(1,1,1),lwd=c(2,2,2))
cast
```

```
library(tidyverse) # for reading in data
library(dplyr) # for data manipulation/selection
library(itsmr) # for testing iid noise and forecasting
library(stlplus) # for decomposing and plotting decomposition components
library(tseries) # for stationarity testing
library(forecast) # for better ACF/PACF and auto.arima


# Reads in data
data=read_csv("ND_Data.csv")

# Creates an array of the active cases
new_cases = data[,5]

# Makes the active cases into a time series
new_cases.ts <- ts(data = new_cases, start = 1, end = 36, frequency = 1)

# Makes an array of dates to use as an x-axis
dates <- seq(as.Date(data$Date[1], format = "%Y-%m-%d"),
as.Date(data$Date[36], format = "%Y-%m-%d"), by = "days")
plot(dates,data$New_Cases, type="l",col=1,main="Daily New North Dakota
Coronavirus Cases", ylab='New Cases', xlab = 'Time')

#look at acf and pacf of raw time series
Acf(active_cases.ts,lag.max = 15,main="Raw Series ACF")
Pacf(active_cases.ts,lag.max = 15,main="Raw Series PACF")

### Decomposethe Model with ITSMR ###

#Fit a linear and quadratic trend to the time series
lintrend=itsmr::trend(data$New_Cases,1)
quadtrend=itsmr::trend(data$New_Cases,2)
#Plot them both on the same graph
plot(dates,new_cases.ts, type='l',main="Daily New North Dakota Coronavirus
Cases with Trend Lines", ylab='New Cases', xlab = 'Time')
lines(dates,quadtrend, col=2)
lines(dates,lintrend, col=4)

#Test how accurate the linear trend is by calculating the MSE
linMSE=lintrend-new_cases.ts
linMSEsquared = linMSE**2
root_linMSE = sqrt(sum(linMSEsquared))
root_linMSE

#Test how accurate the quadratic trend is by calculating the MSE
quadMSE=quadtrend-new_cases.ts
quadMSEsquared = quadMSE**2
root_quadMSE = sqrt(sum(quadMSEsquared))
root_quadMSE

#Decomposition resumes now after determining our trend line.
season=itsmr::season(x=data$New_Cases,d=7)
plot(dates, season, type='l')
residuals=new_cases.ts-quadtrend-season
```

```
#Plot of trend + seasonality vs Data
ts = quadtrend + season
plot(dates,new_cases.ts,type='l',main='ND Daily New Coronavirus Cases with
Trend and Seasonality Overlaid',xlab='Time',ylab='Daily New Cases')
lines(dates,ts, col='red',lwd=2)

#Tests if the residuals are IID noise
test(residuals)
#We conclude that the residuals are IID noise as we cannot disprove the null
hypothesis

#Making an array of the trend/season to use later when we forecast.
M=c('season',7,'trend',2)

#Augmented Dickey-Fuller Test to see if the residuals are stationary (not time
dependent)
tseries::adf.test(residuals,alternative = 'stationary')

#Check how the settings change the outcome
auto.arima(residuals,trace=T,approximation = F,stepwise = F, stationary = T)

#Forecast using our model determined by the lowest AICC and best fitting
trend/seasonality
cast=itsmr::forecast(new_cases.ts,M=M,a=itsmr::arma(residuals,0,0),h=7,opt=2)

#Created for plotting the forecast
dates2 = seq(as.Date('2020-04-29'),as.Date('2020-05-05'),by="days")

#Plot of the forecast and data
plot(c(dates, dates2), c(new_cases.ts, cast$pred), type='l', main= 'Forecasted
Daily New Coronavirus Cases in North Dakota', ylab = 'Daily New Cases', xlab =
'Time', ylim = c(0,150))
lines(dates2, cast$pred, col='blue', type='o')
lines(dates2, cast$l, col='red',type='o')
lines(dates2, cast$u, col='red',type='o')
legend(x=dates[1],y=145,legend = c('original series','forecast of next
week','95% C.I.'),col=c(1,4,2),lty=c(1,1,1),lwd=c(2,2,2))
cast
```

```
library(tidyverse) # for reading in data
library(dplyr) # for data manipulation/selection
library(itsmr) # for testing iid noise and forecasting
library(stlplus) # for decomposing and plotting decomposition components
library(tseries) # for stationarity testing
library(forecast) # for better ACF/PACF and auto.arima


# Reads in data
data=read_csv("SD_Data.csv")

# Creates an array of the active cases
new_cases = data[,5]

# Makes the active cases into a time series
new_cases.ts <- ts(data = new_cases, start = 1, end = 36, frequency = 1)

# Makes an array of dates to use as an x-axis
dates <- seq(as.Date(data$Date[1], format = "%Y-%m-%d"),
as.Date(data$Date[36], format = "%Y-%m-%d"), by = "days")
plot(dates,data$New_Cases, type="l",col=1,main="Daily New South Dakota
Coronavirus Cases", ylab='New Cases', xlab = 'Time')

#look at acf and pacf of raw time series
Acf(active_cases.ts,lag.max = 15,main="Raw Series ACF")
Pacf(active_cases.ts,lag.max = 15,main="Raw Series PACF")

### Decomposethe Model with ITSMR ###

#Fit a linear and quadratic trend to the time series
lintrend=itsmr::trend(data$New_Cases,1)
quadtrend=itsmr::trend(data$New_Cases,2)
#Plot them both on the same graph
plot(dates,new_cases.ts, type='l',main="Daily New South Dakota Coronavirus
Cases with Trend Lines", ylab='New Cases', xlab = 'Time')
lines(dates,quadtrend, col=2)
lines(dates,lintrend, col=4)

#Test how accurate the linear trend is by calculating the MSE
linMSE=lintrend-new_cases.ts
linMSEsquared = linMSE**2
root_linMSE = sqrt(sum(linMSEsquared))
root_linMSE

#Test how accurate the quadratic trend is by calculating the MSE
quadMSE=quadtrend-new_cases.ts
quadMSEsquared = quadMSE**2
root_quadMSE = sqrt(sum(quadMSEsquared))
root_quadMSE

#Decomposition resumes now after determining our trend line.
season=itsmr::season(x=data$New_Cases,d=7)
plot(dates, season, xlab = 'Time', ylab= 'New Cases', main = 'South Dakota New
Case Seasonality', type='l')
```

```
residuals=new_cases.ts-quadtrend-season

#Plot of trend + seasonality vs Data
ts = quadtrend + season
plot(dates,new_cases.ts,type='l',main='SD Daily New Coronavirus Cases with
Trend and Seasonality Overlaid',xlab='Time',ylab='Daily New Cases')
lines(dates,ts, col='red',lwd=2)

#Tests if the residuals are IID noise
test(residuals)
#We conclude that the residuals are IID noise as we cannot disprove the null
hypothesis


#Making an array of the trend/season to use later when we forecast.
M=c('season',7,'trend',2)

#Augmented Dickey-Fuller Test to see if the residuals are stationary (not time
dependent)
tseries::adf.test(residuals,alternative = 'stationary')

#Check how the settings change the outcome
auto.arima(residuals,trace=T,approximation = F,stepwise = F, stationary = F)

#Forecast using our model determined by the lowest AICC and best fitting
trend/seasonality
cast=itsmr::forecast(new_cases.ts,M=M,a=itsmr::arma(residuals,0,3),h=7,opt=2)

#Created for plotting the forecast
dates2 = seq(as.Date('2020-04-29'),as.Date('2020-05-05'),by="days")

#Plot of the forecast and data
plot(c(dates, dates2), c(new_cases.ts, cast$pred), type='l', main= 'Forecasted
Daily New Coronavirus Cases in South Dakota', ylab = 'Daily New Cases', xlab =
'Time', ylim = c(0,200))
lines(dates2, cast$pred, col='blue', type='o')
lines(dates2, cast$l, col='red',type='o')
lines(dates2, cast$u, col='red',type='o')
legend(x=dates[1],y=190,legend = c('original series','forecast of next
week','95% C.I.'),col=c(1,4,2),lty=c(1,1,1),lwd=c(2,2,2))
cast
```

10.2 Python Code

```
# Separate the CSV into multiple DataFrames
df = pd.read_csv(io.BytesIO(uploaded['forecast_new.csv']))
ky, sd, nd, mn = df[:7], df[7:14], df[14:21], df[21:]
ky.reset_index(inplace=True)
sd.reset_index(inplace=True)
nd.reset_index(inplace=True)
mn.reset_index(inplace=True)

# Important Data
states = ['KY', 'SD', 'ND', 'MN']
dist = [637.58, 201.28, 214.54, 0]
popu = [4468000, 884659, 762062, 5640000]
distProportion = [x/sum(dist) for x in dist]

# Linear Program
def find(day):
  p = [ky['State Population'][day], sd['State Population'][day],
       nd['State Population'][day], mn['State Population'][day]]
  i = [ky['Predicted New Cases'][day], sd['Predicted New Cases'][day],
       nd['Predicted New Cases'][day], mn['Predicted New Cases'][day]]

  r = [x/sum(i) for x in i] # infection ratios
  c = [-1*x/y for (x,y) in zip(i,p)] # -1 so it maximizes instead of minimizes

  A = [
      [1, 1, 1, 1],
      [-1, -1, -1, -1],

      [-1, 0, 0, 0],
      [0, -1, 0, 0],
      [0, 0, -1, 0],
      [0, 0, 0, -1]
  ]

  b = [1, -1, -1*r[0], -1*r[1], -1*r[2], -1*r[3]]

  bounds = [
            (0, None),
            (0, None),
            (0, None),
            (0, None)
  ]

  result = linprog(c, A_ub=A, b_ub=b, bounds=bounds, method='revised simplex')
  return result.x;

everyday = []
for i in range(7):
  everyday.append(find(i))

divisionOverTime = pd.DataFrame(everyday, columns=states)

# Account for Distance
distProportion = [(x/sum(dist))-0.25 for x in dist]

adjusted = []
for i in divisionOverTime.iterrows():
  adjusted.append([])
  k = 0
  for j in i[1]:
    adjusted[-1].append(j + 0.1*distProportion[k])
    k += 1

adjustedDf = pd.DataFrame(adjusted, columns=states)
```