COVID-19 Modeling: The Classic SIR Model

Leo PeBenito

6/25/2020

Introduction

Epidemiological models have a long history in the fight against infectious disease. In principle these models enable us to predict the onset of surges in the number of cases. The ability to anticipate health care needs based on this information allows medical institutions to prepare, and permits political officials to make timely policy recommendations.

Here we develop an R implementation of the classic SIR model and apply it to the COVID-19 data for the United States curated by the Johns Hopkins University (JHU) Center for Systems Science and Engineering (CSSE). The JHU time series data is stored as cumulative case counts at the city/town level (henceforth referred to as location/locale here and in the ensuing code). While the number of deaths due to COVID-19 are typically reported, data documenting recoveries is lacking including from states suffering significantly from the pandemic. Data on recoveries from COVID-19 is the complement to the data on cases from which information on the number of infectious individuals can be inferred. Without this necessary data we make a crude estimate on the time course of the infectious compartment based on the World Health Organization's estimate of the average duration of the disease (WHO 2020).

The results highlight the need for widespread and rapid testing, as well as for continuous monitoring of those infected. This code serves as a starting point for expansion of the model to include additional compartments (Hill 2020). Machine learning methods can be used to obtain better estimates for model parameters (Dandekar 2020). Interrupted time series analysis methods can be used to continue to follow the course of the pandemic and to evaluate the various measures implemented that may or may not be intended to curtail the spread of infection (Siedner et al. 2020).

The classic SIR Model

In the classic SIR model the population is compartmentalized into the susceptible, the infectious, and the recovered (which also includes the deceased). Solving the set of three simultaneous differential equations

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

$$\frac{dI(t)}{dt} = \frac{\beta\, S(t)\, I(t)}{N} - \gamma\, I(t)$$

$$\frac{dR(t)}{dt} = \gamma\, I(t)$$

, requires fitting two adjustable parameters, β and γ. β controls the transition between S and I. γ controls the transition between I and R. R_0, known as the basic reproductive number, is equal to the ratio of β/γ.

The loss function is defined as the residual sum of squares for time evolution of the the number of infected individuals

$$RSS = \sum_t \left(\hat{I}(t) - I(t) \right)^2$$

.

Exploratory Data Analysis:

Load relevant packages for data wrangling and analysis.

```
## Load packages

## Data wrangling
if(!require(tidyverse)) install.packages("tidyverse",
                                         repos = "http://cran.us.r-project.org")
## Work with dates
if(!require(lubridate)) install.packages("lubridate",
                                         repos = "http://cran.us.r-project.org")
## For web scraping
if(!require(rvest)) install.packages("rvest",
                                     repos = "http://cran.us.r-project.org")
## Has unscale(), the opposite of scale()
if(!require(DMwR)) install.packages("DMwR",
                                    repos = "http://cran.us.r-project.org")
## Get kmeans()
if(!require(caret)) install.packages("caret",
                                     repos = "http://cran.us.r-project.org")
## Perform join on imperfect match
if(!require(fuzzyjoin)) install.packages("fuzzyjoin",
                                         repos = "http://cran.us.r-project.org")
## Solve differential equations
if(!require(deSolve)) install.packages("deSolve",
                                       repos = "http://cran.us.r-project.org")
if(!require(data.table)) install.packages("data.table",
                                          repos = "http://cran.us.r-project.org")
if(!require(knitr)) install.packages("knitr",
                                     repos = "http://cran.us.r-project.org")
```

Download JHU CSSE data sets from GitHub. To run this code requires an internet connection.

```
## Download data set(s) from:
## Johns Hopkins University Center for Systems Science and Engineering (CSSE)

## CSV file of time series of confirmed cases
url_jhu_ts_confirmed_US <- paste("https://raw.githubusercontent.com/CSSEGISandData/",
                      "COVID-19/master/csse_covid_19_data/", "csse_covid_19_time_series/",
                      "time_series_covid19_confirmed_US.csv", sep="")

confirmed_US <- read_csv(url(url_jhu_ts_confirmed_US))



## CSV file of time series of COVID-19 deaths
```

```
## This data set also contains the population data
url_jhu_ts_deaths_US <- paste("https://raw.githubusercontent.com/CSSEGISandData/",
                      "COVID-19/master/csse_covid_19_data/", "csse_covid_19_time_series/",
                      "time_series_covid19_deaths_US.csv", sep="")

deaths_US <- read_csv(url(url_jhu_ts_deaths_US))
```

Put data in tidy format, give columns better names, simplify locale names, and convert dates to date class.

```
## Extract population data from deaths_US data set
population_data_US <- deaths_US[c("Province_State", "Combined_Key", "Population")]
population_data_US <- population_data_US %>%
  rename(State = "Province_State", Locale = "Combined_Key") %>% # Rename columns
  mutate(Locale = str_remove(Locale, ", US"))                  # Remove country spec


## Remove columns: UID, iso2, iso3, code3, FIPS, Admin2, Country_Region, Lat, Long
## Keep columns: Province_State, Combined_Key, and columns corresponding to dates
confirmed_US <- confirmed_US[,-c(seq(1,6), seq(8,10))]

## Keep columns: Province_State, Combined_Key, and columns corresponding to dates
## Omit column 12: Population
deaths_US <- deaths_US[,-c(seq(1,6), seq(8,10), 12)]


## Rename fields containing detailed location data
confirmed_US <- confirmed_US %>% rename(State = "Province_State", Locale = "Combined_Key")
deaths_US    <- deaths_US    %>% rename(State = "Province_State", Locale = "Combined_Key")


## Put data in tidy format
confirmed_US <- confirmed_US %>%
  pivot_longer(

    cols = -c(State, Locale),
    names_to  = "Date",
    values_to = "Cases"

  )

deaths_US <- deaths_US %>%
  pivot_longer(

    cols = -c(State, Locale),
    names_to  = "Date",
    values_to = "Deaths"

  )

## Remove string ", US" from Locale field
confirmed_US <- confirmed_US %>% mutate(Locale = str_remove(Locale, ", US"))
deaths_US    <- deaths_US    %>% mutate(Locale = str_remove(Locale, ", US"))
```

```
## Convert dates data to mode date
confirmed_US <- confirmed_US %>% mutate(Date = mdy(Date))
deaths_US    <- deaths_US    %>% mutate(Date = mdy(Date))


## Join tables by State, Locale, and Date
## note: data on deaths is not used in the following
##...this procedure is included here for completeness.
jhu_US_data <- inner_join(

  x = confirmed_US,
  y = deaths_US,
  by = c(

    "State" = "State",
    "Locale" = "Locale",
    "Date" = "Date"

    ),
  keep = FALSE

)
```

Clean data by omitting locales that lack corresponding population data, as well as those that never record any cases.

```
#####################################################################
## Omit places that do not have any cases from consideration ##
#####################################################################


## Find locales that never record any cases
locales_with_zero_cases <- jhu_US_data %>%
  group_by(Locale) %>%
  summarize(sum_cases = sum(Cases)) %>%
  filter(sum_cases == 0) %>%
  pull(Locale)


## Omit locales with no cases
jhu_US_data <- jhu_US_data %>%
  filter(!Locale %in% locales_with_zero_cases)

population_data_US <- population_data_US %>%
  filter(!Locale %in% locales_with_zero_cases)



################################################
## Omit places that have no population data ##
################################################
```

```r
## Note: some Locale listings have Population data equal to zero
locales_without_population <-
  population_data_US$Locale[which(population_data_US$Population == 0)]


## Remove entries where Population equals zero from the population data set
population_data_US <- population_data_US %>%
  filter(Population > 0)

jhu_US_data <- jhu_US_data %>%
  filter(!Locale %in% locales_without_population)
```

Define some useful functions for later use.

```r
## Get the date that cases start to appear in a given locale
get_first_cases_date <- function(location) {

  jhu_US_data %>%
    filter(Locale == location, Cases != 0) %>%
    select(Date)  %>%
    arrange(Date) %>%
    pull(Date) %>%
    head(1)

}


## Get the date of the most recently reported cases in a given locale
get_recent_cases_date <- function(location) {

  jhu_US_data %>%
    filter(Locale == location, Cases != 0) %>%
    select(Date) %>%
    arrange(desc(Date)) %>%
    pull(Date) %>%
    head(1)

}


## Get the dates between when 1st and last cases are reported in a given locale
get_cases_dates <- function(location) {

  first_cases_date  <- get_first_cases_date(location)

  recent_cases_date <- get_recent_cases_date(location)

  jhu_US_data %>%
    filter(Locale == location) %>%
    filter(between(Date, first_cases_date, recent_cases_date)) %>%
    pull(Date)

}
```

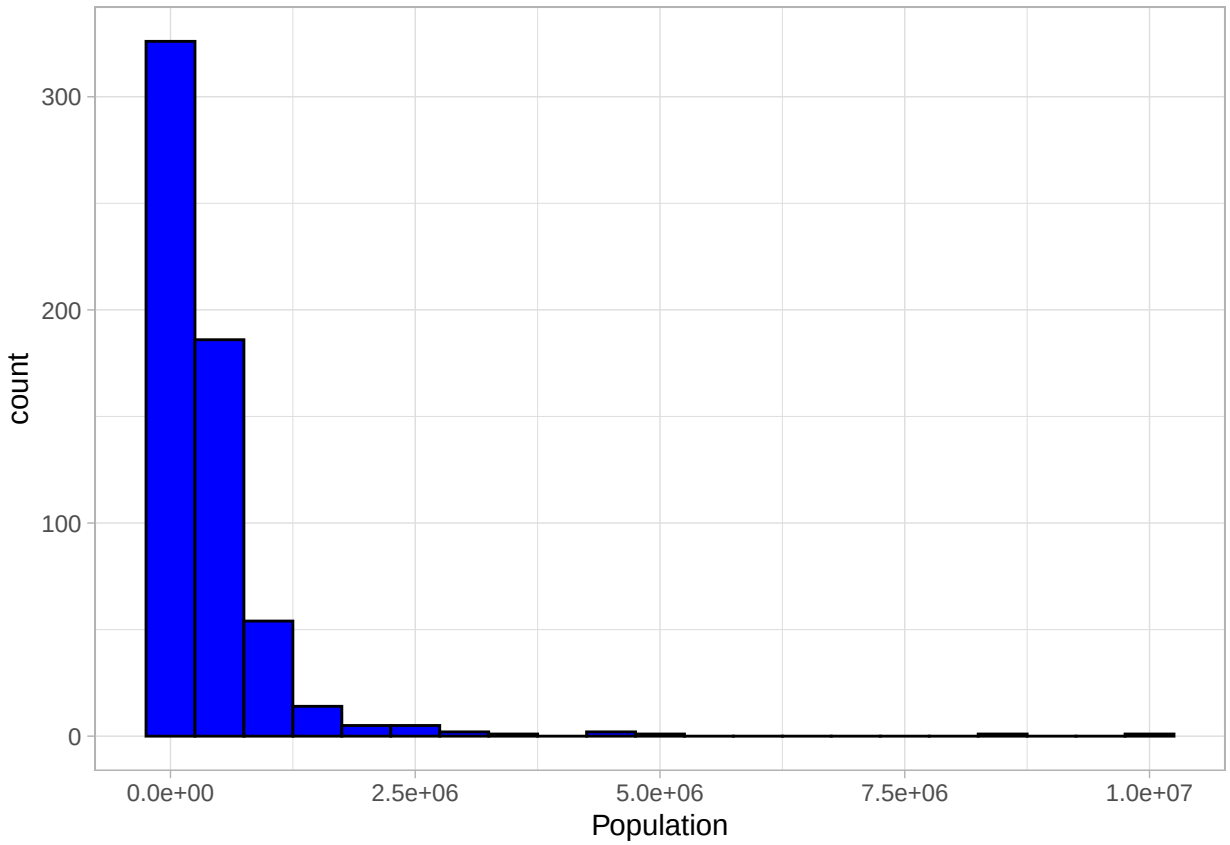
```r
## Get the date that X number of infectious individuals are observed in a given locale
## Default: X = 5000
date_of_X_infectious <- function(location, x = 5000) {

  get_raw_data(location) %>%
    filter(infected >= x) %>%
    arrange(date) %>%
    head(1) %>%
    pull(date)

}


##############################################################################################
## Get the raw data for the SIR compartments ##################################################
## Estimate the number of infectious individuals as:  Infectious = Cases - Recovered ##
##############################################################################################

get_raw_data <- function(location, lag = 0) {

  ## Date that cases start to appear
  first_cases_date <- get_first_cases_date(location)


  ## Date of the most recently reported cases
  recent_cases_date <- get_recent_cases_date(location)


  ## Dates between when the 1st and last cases are reported
  dates <- get_cases_dates(location)


  ## Cases between when the 1st and last cases are reported
  cases <- jhu_US_data %>%
    filter(Locale == location) %>%
    filter(between(Date, first_cases_date, recent_cases_date)) %>%
    pull(Cases)


  ## Use the average recovery time to estimate the no. of individuals who have recovered.
  ## WHO estimates an average recovery time of ~2 weeks (14 days).
  ## Recall: there is no distinction made between recovered and deceased.
  ## Shift data to the right changing leading positions now with missing values to 0
  ##... and truncating trailing values past the length of the input vector.
  recovery_time <- 14 # in days
  recovered <- c(

    rep(0, recovery_time),
    cases[seq(1, length(cases) - recovery_time)]

  )
```

```
  ## Estimate Infected/Infectious data
  infected <- cases - recovered


  ## Index by day since the 1st cases are seen to begin modeling
  start_index <- 1 + lag

  ## Index by day since the 1st cases are seen to stop modeling
  stop_index <- length(cases)


  return(data.frame(

    date  = dates[start_index:stop_index],
    cases = cases[start_index:stop_index],
    infected  = infected[start_index:stop_index],
    recovered = recovered[start_index:stop_index]

    )

  )

}
```

K-means Clustering

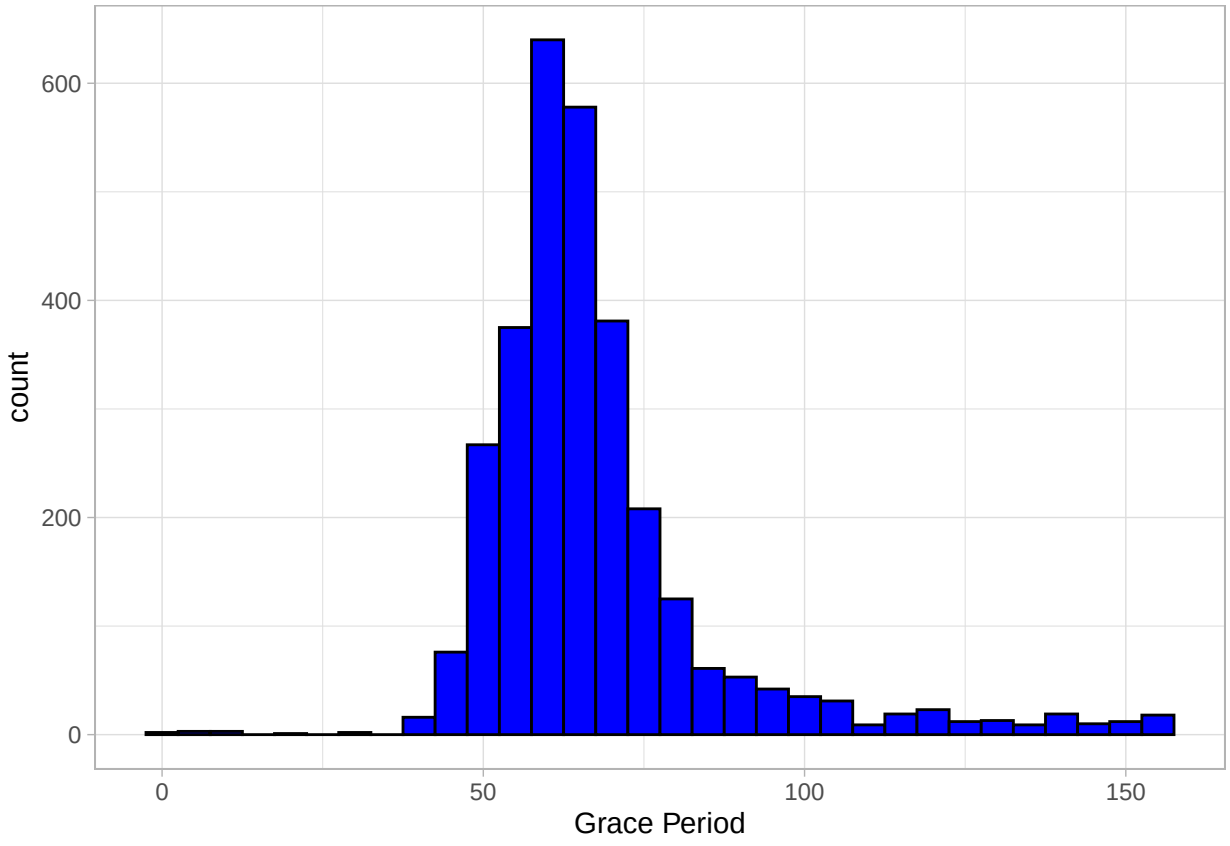
K-means clustering is used to identify a manageable subset of the data to focus on. Two parameters that can be used for clustering based on the JHU data alone are the population, and the period between when the first cases are observed in the United States and when the first cases are recorded (referred to as the grace period) for a given locale.

```
#########################################################################
## Preliminary analysis prior to K-means clustering
## Examine parameters available in JHU data set
## 1. Population of a given locale
## 2. The number of days between when cases are 1st reported in the US
##... and when cases are 1st reported in a given locale
#########################################################################

#####################################################
## Visualize variable for clustering: population ##
#####################################################

## Look at 1st variable considered for clustering: population
## Visualize locales with population over 100k
population_data_US %>%
  filter(Population > 100000) %>%
  ggplot(aes(Population)) +
  theme_light() +
  geom_histogram(binwidth = 500000, fill = "blue", col = "black")
```




```
#######################################################################
## Generate variable for clustering:  days between 1st recorded    ##
##... cases in the US and the 1st recorded cases in a given locale ##
#######################################################################

## The date of the first reported cases in the US
first_cases_date <- jhu_US_data %>%
  filter(Cases != 0) %>%
  arrange(Date) %>%
  head(1) %>%
  pull(Date)


## Days between when the 1st cases are recorded in the US and
##... when cases are 1st recorded in a locale, aka grace period
get_grace_period <- function(x) {

  jhu_US_data %>%
    filter(Locale == x, Cases != 0) %>%
    mutate(Grace_period = as.numeric(Date - first_cases_date, units = "days")) %>%
    arrange(Date) %>%
    select(Locale, Grace_period) %>%
    head(1)

}
```

```r
## Make vector of locales
locales <- unique(jhu_US_data$Locale)


## Get grace period for each locale
grace_periods <- map_dfr(locales, get_grace_period)



#########################################################
## Visualize variable for clustering: grace period ##
#########################################################

grace_periods %>%
  ggplot(aes(Grace_period)) +
  theme_light() +
  geom_histogram(binwidth = 5, fill = "blue", col = "black") +
  labs(x = "Grace Period")
```



Prepare data for K-means clustering by joining the population and grace period data into a single data frame, scaling the data by computing z-scores, and using the locale as the row name rather than as an explicit column of the data frame.

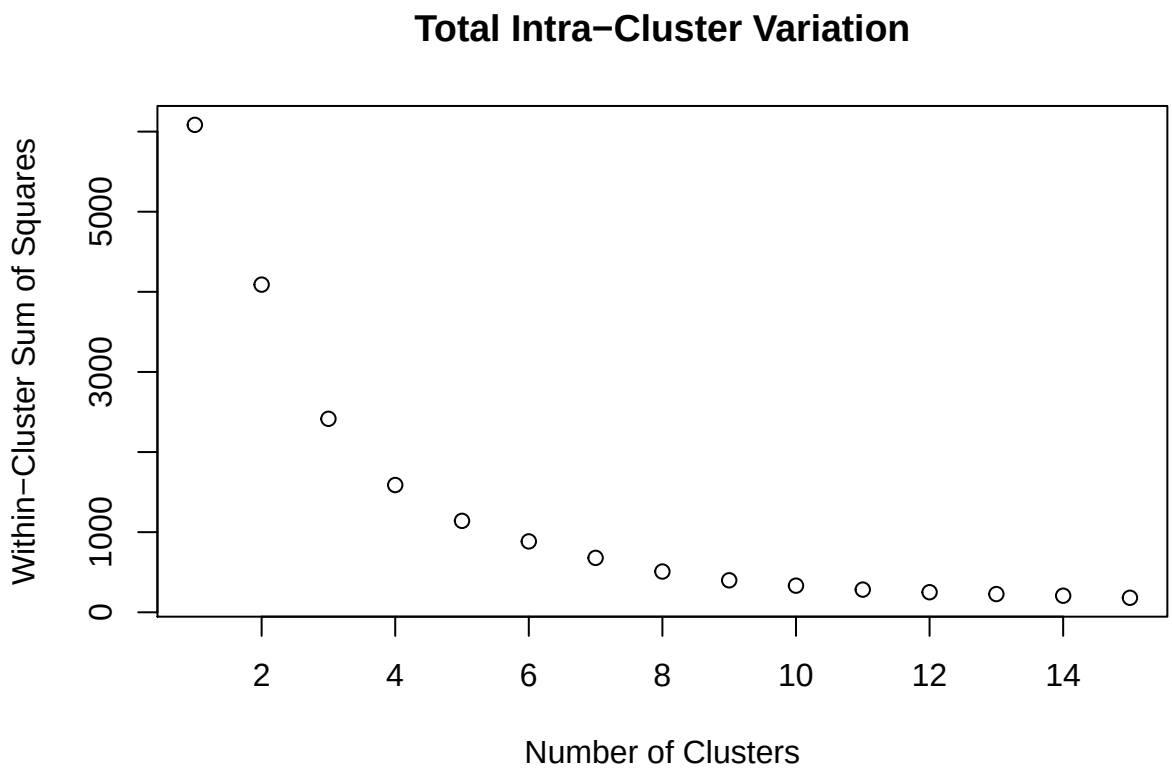
```r
########################################
## Prepare data for K-means Clustering ##
```

```
########################################

## Combine grace period information with population information
kmeans_setup <- inner_join(

  x = grace_periods,
  y = population_data_US,
  by = c("Locale" = "Locale")

)


## Omit State field
kmeans_setup <- kmeans_setup %>% select(-"State")


## Use tibble package to convert Locale field values to row names
## This is the data.frame structure for K-means
kmeans_setup <- column_to_rownames(.data = kmeans_setup, var = "Locale")


# Compute Z-scores of fields for K-means: Grace_period, Population
kmeans_setup <- scale(kmeans_setup)
```

Determine the optimum number of clusters using the "elbow method" (Hastie, Tibshirani, and Friedman 2009).

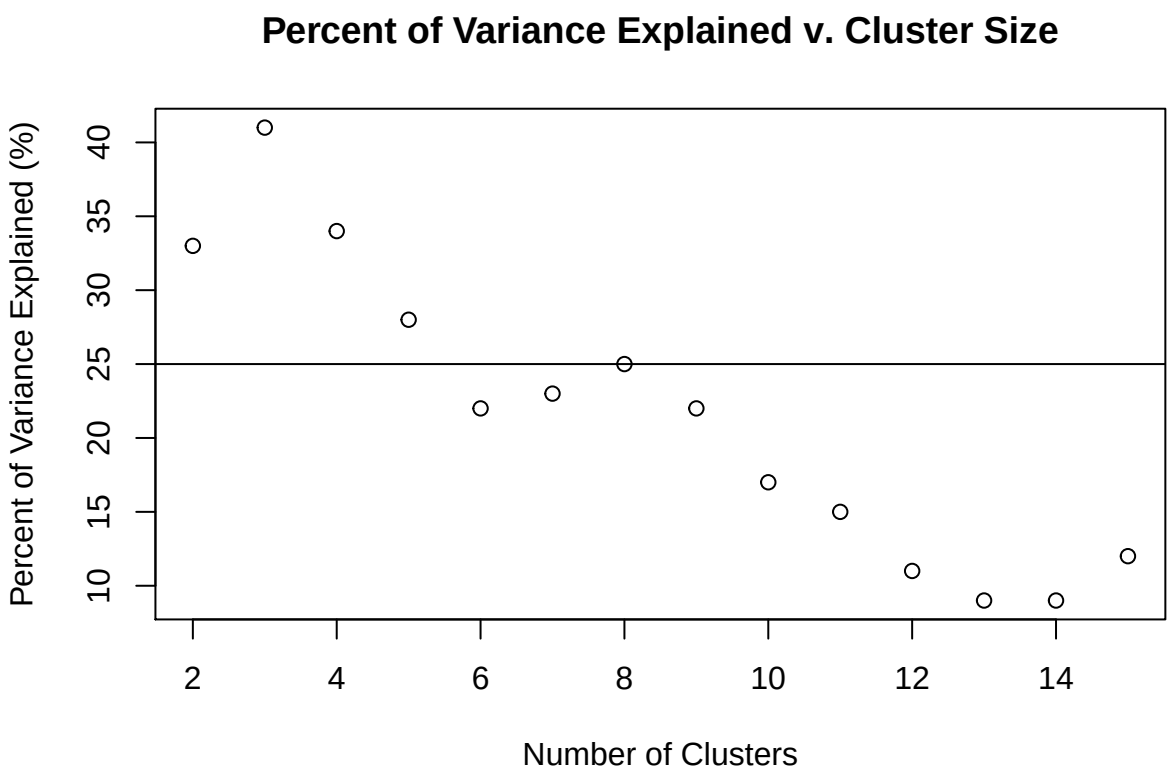
```
## K-means clustering
## Parameters: 2 parameters considered
## (1) population,
## (Prefer to use population density but land area info is difficult to obtain)
## (2) Days between when the 1st cases are documented in the US and
##... when the 1st cases are documented in a given locale


## Set seed of random number generator
set.seed(2394817)


########################################################
## Function to compute within-cluster sum of squares ##
########################################################

wss <- function(k) {

  kmeans(

    kmeans_setup,
    centers = k,
    nstart = 25,
    iter.max = 20

  )$tot.withinss
```

```
}


## Sequence of k-values to test
k_seq <- seq(1, 15)


## Get sequence of within-cluster sum of squares
wss_seq <- map_dbl(k_seq, wss)


## Elbow plot: Plot total intra-cluster variation
##... (total within-cluster sum of squares) v. the number of clusters
plot(x = k_seq, y = wss_seq,
     main = "Total Intra-Cluster Variation",
     ylab = "Within-Cluster Sum of Squares",
     xlab = "Number of Clusters")
```




```
## Plot the percent of variance explained v. the number of clusters
pve <- -diff(wss_seq)/wss_seq[1:length(wss_seq)-1]

plot(k_seq[2:length(k_seq)], round(pve, 2)*100,
     main = "Percent of Variance Explained v. Cluster Size",
     ylab = "Percent of Variance Explained (%)",
     xlab = "Number of Clusters")
```

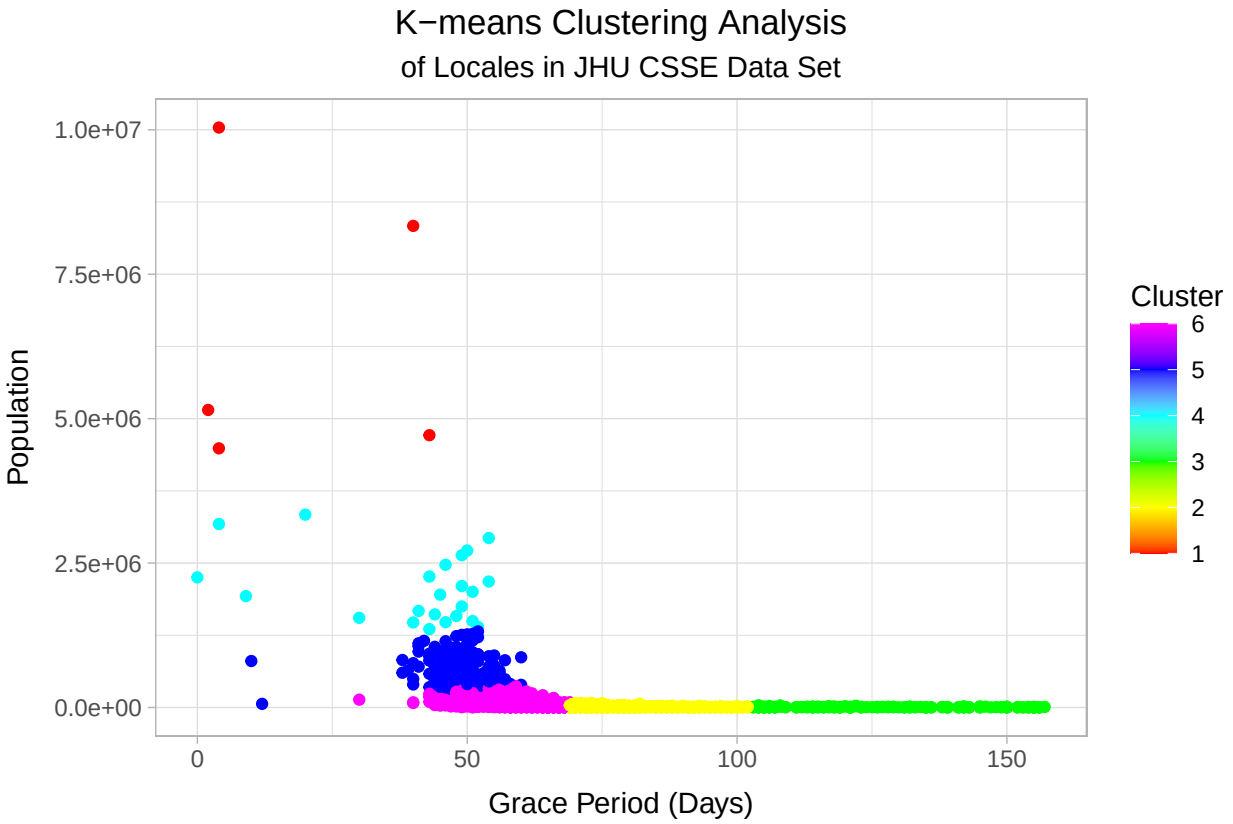
```
## Set first 0.25 (25%) value as the arbitrary cut-off
## Observation: 6 clusters brings the pve to about 0.2
abline(h = 25)
```



Examine clusters using k = 6.

```
#####################################################################
## Visualize clusters given the optimal number of clusters: 6 ##
#####################################################################

## Generate 6 clusters
kmeans_data <- kmeans(

  kmeans_setup,
  centers  = 6,
  nstart   = 25,
  iter.max = 20

)


## Put cluster data for each locale in a data.frame
kmeans_data <- data.frame(kmeans_data$cluster)
```

```r
## Rename Cluster field
kmeans_data <- kmeans_data %>%
  rename(Cluster = kmeans_data.cluster)


## Return locale info in row names to an explicit column (tidy format)
kmeans_data  <- rownames_to_column(.data = kmeans_data , var = "Locale")


## Use unscale() of DMwR package to reverse action of scale() on kmeans_setup
## Join clustering data with the variables used for clustering
## Note: here Variables used for clustering have been 'unscaled', that is they
##... have been returned to their original values from the z-scores.
clustering_data <- inner_join(

  x = kmeans_data,
  y = rownames_to_column(

    .data = data.frame(unscale(kmeans_setup, kmeans_setup)),
    var = "Locale"

  ),
  by = c("Locale" = "Locale")

)
```



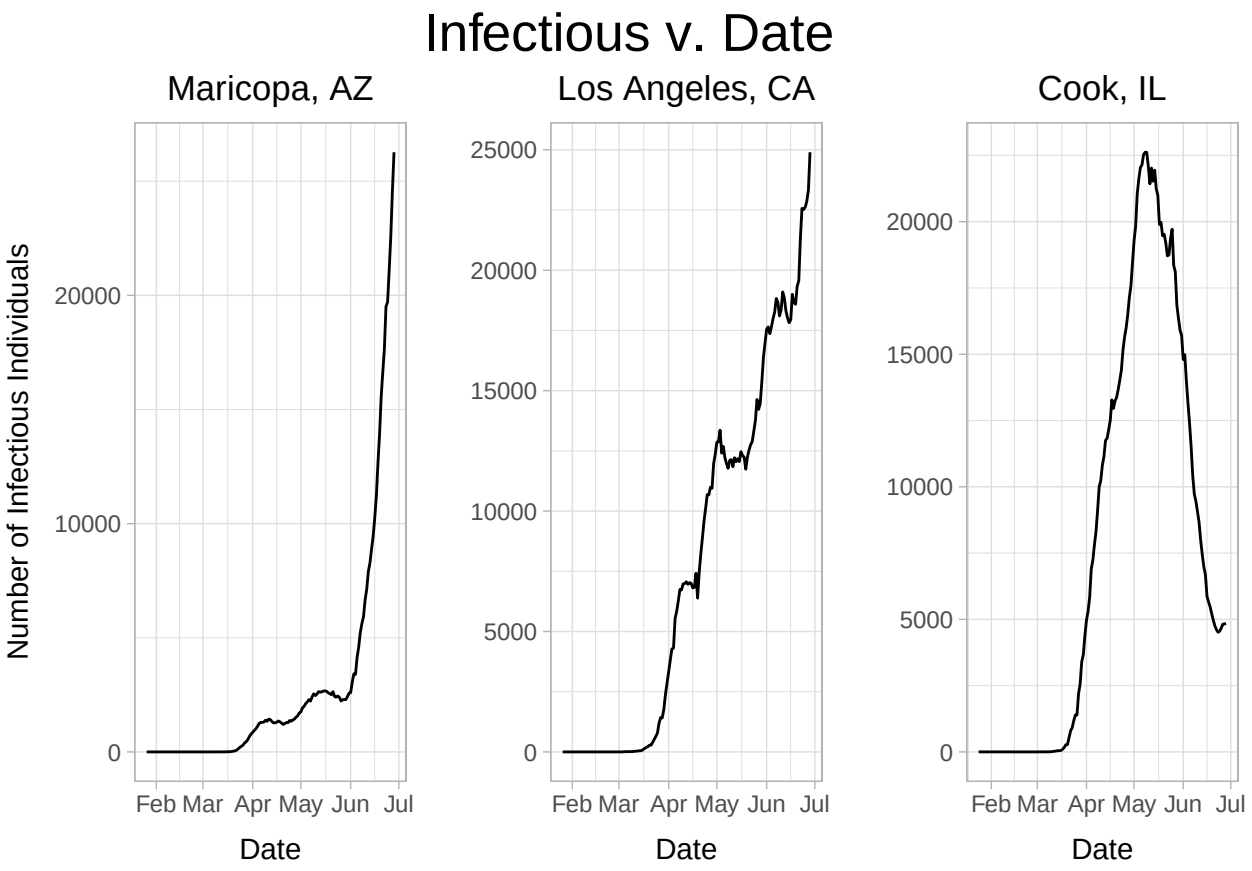
*Grace period : days between the 1st recorded cases in the US and the 1st recorded cases for a given locale

We focus on the cluster containing New York City, NY.

```
## Get cluster containing NYC, NY
nyc_cluster_id <- clustering_data %>%
  filter(Locale == "New York City, New York") %>%
  pull(Cluster)


## Get info for locales in same cluster as NYC, NY
nyc_cluster <- clustering_data %>%
  filter(Cluster == nyc_cluster_id)


# Show info for the cluster containing New York City, NY
nyc_cluster
```

```
##                   Locale Cluster Grace_period Population
## 1       Maricopa, Arizona       1            4    4485414
## 2 Los Angeles, California       1            4   10039107
## 3          Cook, Illinois       1            2    5150233
## 4 New York City, New York       1           40    8336817
## 5           Harris, Texas       1           43    4713325
```

Given that Maricopa, AZ, Los Angeles, CA, and Cook, IL, document cases well before New York City, NY, and Harris, TX, despite being in the same cluster, the former set of locales will be used to make predictions about the latter. We take a closer look at the data before proceeding with modeling.



Data for Maricopa, AZ, and Los Angeles, CA, look fairly irregular. For simplicity we therefore will use Cook, IL, as a basis for developing the SIR model before applying it to make predictions about New York City, NY.

Procedure: The SIR Model in R

Define the loss function as the residual sum of squares (RSS) and use it to evaluate the SIR model (Choisy 2018; Churches 2020). To estimate the number of infectious individuals, we assume that one is either dead or recovered after two weeks (WHO), and subtract the number of cases two weeks prior from the number of cases on a given day. This gross estimate is necessary due to a lack of data on COVID-19 recoveries.

```
## Define loss function as residual sum of squares
RSS <- function(prediction, raw_data) {

  return( sum( (prediction - raw_data)^2 ) )

}


## Define SIR model
SIR <- function(time, sir_variables, sir_parameters) {

  with(as.list(c(sir_variables, sir_parameters)), {

    dS <- - beta * S * I / N
    dI <-   beta * S * I / N - gamma * I
    dR <-  gamma * I

    return( list( c(dS, dI, dR) ) )

  })

}
```

Define functions for evaluating the SIR model based on the loss (RSS) function.

```
#######################################################
## Define function to apply SIR model to raw data ##
#######################################################

## Input parameters:
## args  <vector of doubles> - beta and gamma (in that order) coefficients in SIR model.
## Note: this function takes parameters beta and gamma as
##... a single vector for compatibility with optim().
## location  <string> - "city/town, state" format, ie: "New York City, New York".
## lag  <int> - skip this number of days since the 1st cases are seen
##... before we begin modeling the data.

fit_SIR_model <- function(args, location = "New York City, New York", lag = 0) {

  ###################################
  ## Define parameters for SIR model ##
```

```
#####################################

## Set beta and gamma parameter values from input
## Convert parameters passed as strings to numbers
parameter_values <- as.numeric(args)


## Calculate the population for the given location
N <- population_data_US %>%
  filter(Locale == location) %>%
  pull(Population)

## Append population data to SIR parameters
parameter_values <- append(parameter_values, N)


## ode() requires parameters be passed as a vector with named components
parameter_values <- setNames(parameter_values, c("beta", "gamma", "N"))



##########################################
## Get raw data for specified location ##
##########################################

raw_data <- get_raw_data(location)



##########################################
## Make a vector to increment dates/time ##
##########################################

## Note: The deSolve function ode() is old-school and does not accept date objects.
## Set a lag value to test the influence on fitting of starting at a later date
days <- 1:(length(raw_data$infected) - lag)



##################################################
## Set initial conditions for the SIR model ##
##################################################

## Set a lag value to test the influence on fitting of starting at a later date
model_start_day <- 1 + lag

initial_values <- c(

  S = N - raw_data$infected[model_start_day],  # number of susceptible people
  I = raw_data$infected[model_start_day],      # no. infectious
  R = raw_data$recovered[model_start_day]      # no. recovered (and immune), or deceased

)
```

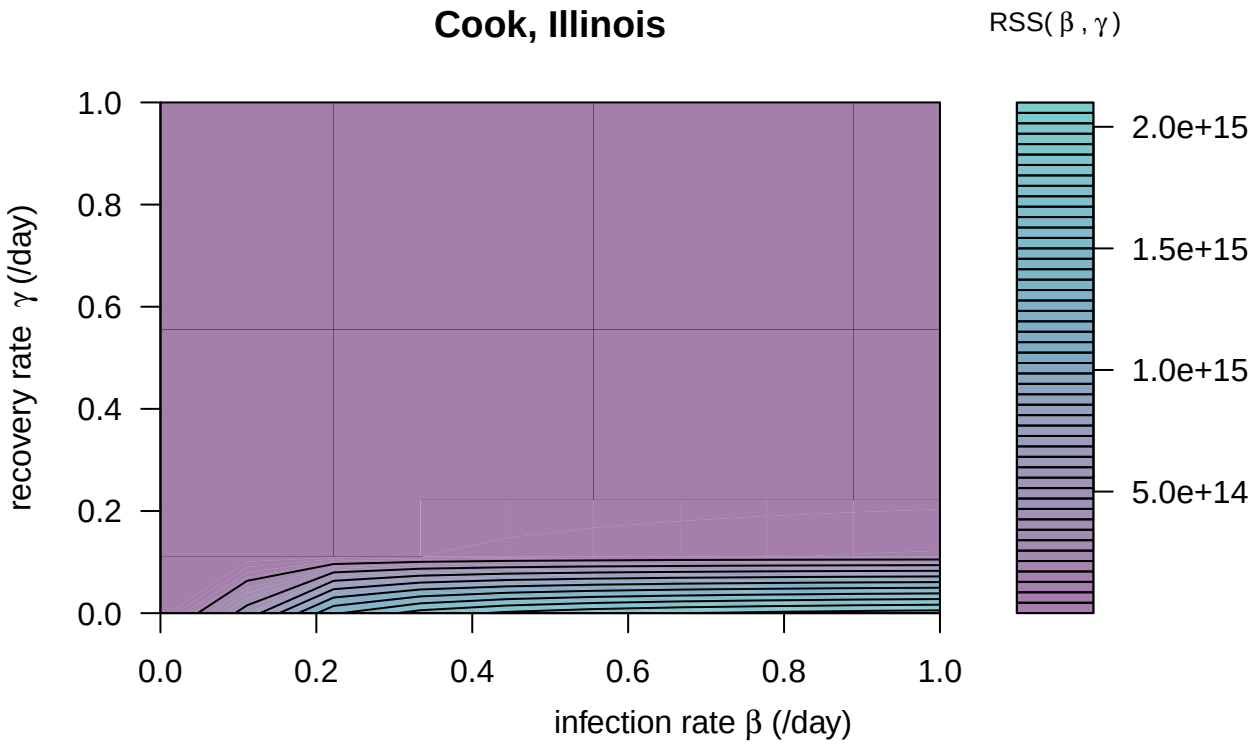
```
  ########################
  ## Evaluate SIR Model ##
  ########################

  ## Solve system of 3 simultaneous equations
  predictions <- ode(

    y = initial_values,
    times = days,
    func  = SIR,
    parms = parameter_values

  )


  return(data.frame(predictions))

}



##############################################################################
## Evaluate the fit using the SIR model based on the (RSS) loss function ##
##############################################################################

get_SIR_fit_RSS <- function(args, location, lag) {

  ## Extract raw data
  raw_data <- get_raw_data(location, lag)


  ## Compute fitted data
  fit_data <- fit_SIR_model(args, location, lag)


  ## Compute the residual sum of squares for fit v. raw for infectious data
  return( RSS(fit_data$I, raw_data$infected) )

}



#################################################################
## Evaluate the SIR model based on the (RSS) loss function ##
#################################################################

## n  - is the number of values of beta and gamma to test
## Gamma can be estimated based on the average duration of the illness,
##... as 1/(duration of illness)
## Upper boundary of 0.2 based on guess of quickest recovery of 5 days
## There is no good way to guess the value of beta
```

```r
## Guess upper boundary of 0.8 based on the ave. R_0 of ~ 2 for the 1918 Spanish Flu
## Recall: R_0 = beta / gamma

run_SIR_RSS_tests <- function(

  beta_seq = betas,
  gamma_seq = gammas,
  n = 10,
  location, lag

  ) {

  ## Create a set of betas and gammas to test.
  ## Generate all combinations of beta and gamma values.
  ## Result is an object of class: data.frame, and mode: list
  parameter_set <- expand.grid(beta_seq, gamma_seq)

  ## Rename Var1 and Var2 columns to beta and gamma
  names(parameter_set) <- c("beta", "gamma")


  ## Combine column entries into single row vector to pass elements
  ##... as single parameter vector to get_sir_fit_rss()
  ##... this is less of an aesthetic choice than a need to conform
  ##... to the same argument specifications as optim()
  ##... such that the same function that is used to evaluate the model
  ##... can be used for optimization.
  ## The result is rows of string vectors (having trouble converting to numeric vector).
  parameter_set <- parameter_set %>%
    transmute(parameters = as.vector(strsplit(paste(beta, gamma), " ")))


  ## Compute RSS for SIR model on COVID-19 data for combinations
  ##... of beta and gamma test values
  rss_tests_results <- parameter_set %>%
    mutate(

      values = map(

        parameters,
        get_SIR_fit_RSS,
        location = location,
        lag = lag

        )

      )


  return(rss_tests_results)

}
```

Set parameters and evaluate the SIR model.

```r
## Specify the name of the location to examine
## To see a list of available locations use: unique(dat$Locale)
the_location = "Cook, Illinois"


## Set number of days to skip since the 1st cases are seen before
##... we begin modeling the data
## Set lag based on when 5000 people become infected/infectious
the_lag <-
  as.numeric(date_of_X_infectious(the_location) - get_first_cases_date(the_location))


## Set the number of beta and the number of gamma values to test.
## (n) must be large enough to achieve sufficient granularity
##... such that a reasonable pair of beta and gamma values can be
##... found to begin optimization with an expectation of a descent fit.
## n = 10 in general is not large enough but runs in a reasonable amount of time.
n = 10

## Set upper boundaries for beta and gamma values
gamma_min = 0.0
gamma_max = 1.0
beta_min  = 0.0
beta_max  = 1.0

## Define sequence of beta and gamma values to test
betas  <- seq(beta_min , beta_max , len=n)
gammas <- seq(gamma_min, gamma_max, len=n)


## Evaluate SIR model
rss_tests_results <- run_SIR_RSS_tests(location = the_location, lag = the_lag)
```

It is apparent from the heat map of RSS values for the range of beta and gamma values tested that there are many possible beta and gamma pairs.



*Days since the first cases are detected before modeling starts: 69

```
## [1] "Minimum RSS:"
## [1] 8572287667
## [1] "Maximum RSS:"
## [1] 2.100227e+15
```

Optimization is performed using the L-BFGS-B method (Byrd et al. 1995) on model fitting using the pair of beta and gamma values obtained from preliminary RSS tests as starting conditions.

```
## Use an informed guess for beta and gamma to optimize fit

## Starting values for SIR parameters beta and gamma to perform optimization
starting_optimization_parameters <-
  unlist(rss_tests_results$parameters[which.min(rss_tests_results$values)])


## Perform optimization
optim_model <- optim(

  par = starting_optimization_parameters, # initial values for the parameters
  location = the_location,  # parameter to get_SIR_fit_RSS() specifying city/location
  lag = the_lag,            # parameter to get_SIR_fit_RSS() specifying model lag
  fn  = get_SIR_fit_RSS,    # Function to be minimized: RSS of SIR fit
  method = "L-BFGS-B" ,     # Gradient projection method using BFGS matrix
  lower  = c(0, 0),         # lower boundary for beta and gamma
  upper  = c(1, 1)          # upper boundary for beta and gamma
```

```
)


## Check for convergence
optim_model$message
```

```
## [1] "CONVERGENCE: REL_REDUCTION_OF_F <= FACTR*EPSMCH"
```

```
## Check convergence: 0 for L-BFGS-G method indicates successful completion
optim_model$convergence
```

```
## [1] 0
```

```
min_rss <- optim_model$value
min_rss
```

```
## [1] 75206877
```

```
## Check fitted values
params_optim <- setNames(optim_model$par, c("beta", "gamma"))
beta_optim   <- params_optim[["beta"]]
beta_optim
```

```
## [1] 0.7566687
```

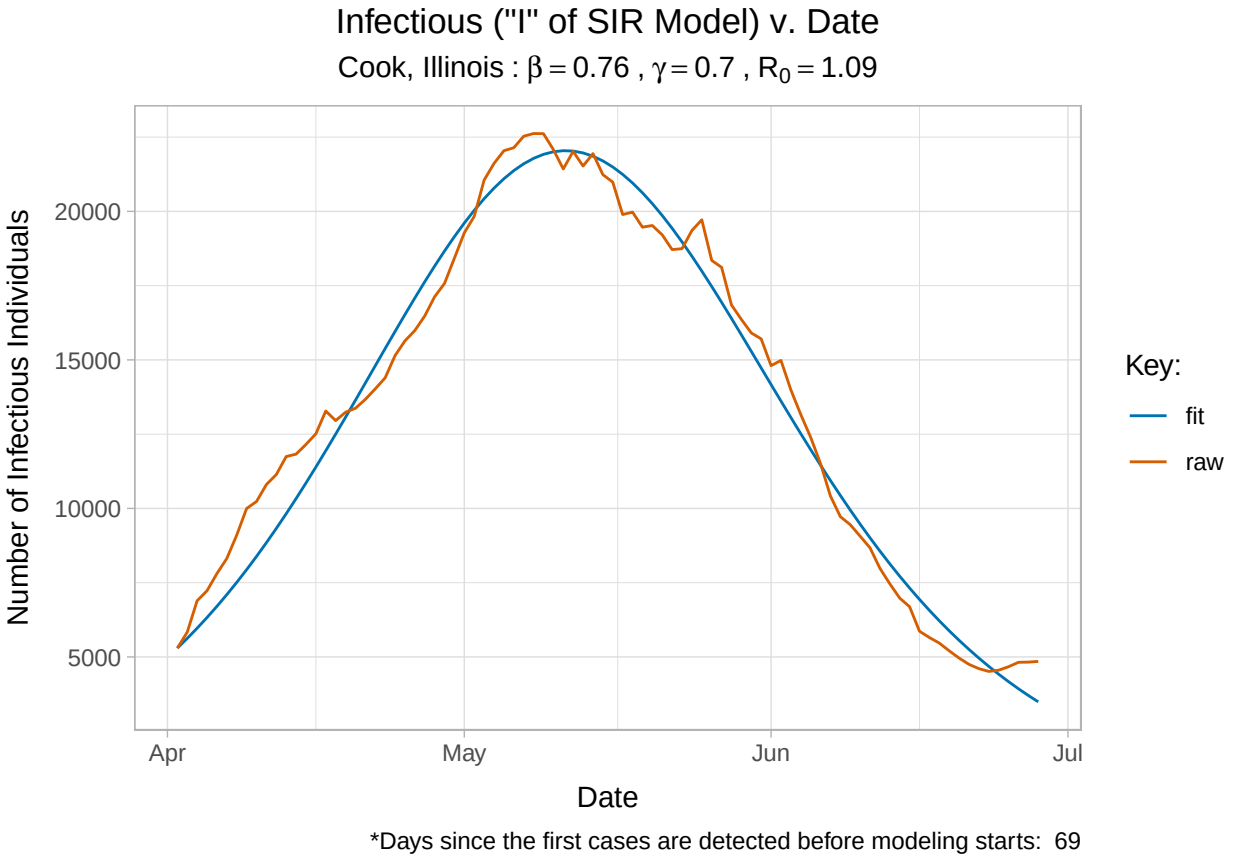
```
gamma_optim  <- params_optim[["gamma"]]
gamma_optim
```

```
## [1] 0.6957262
```

```
## Compute R_0 = beta / gamma
R0  <- params_optim[["beta"]] / params_optim[["gamma"]]
R0
```

```
## [1] 1.087596
```

Plot the optimized fit and the raw data for Cook, IL.



Results

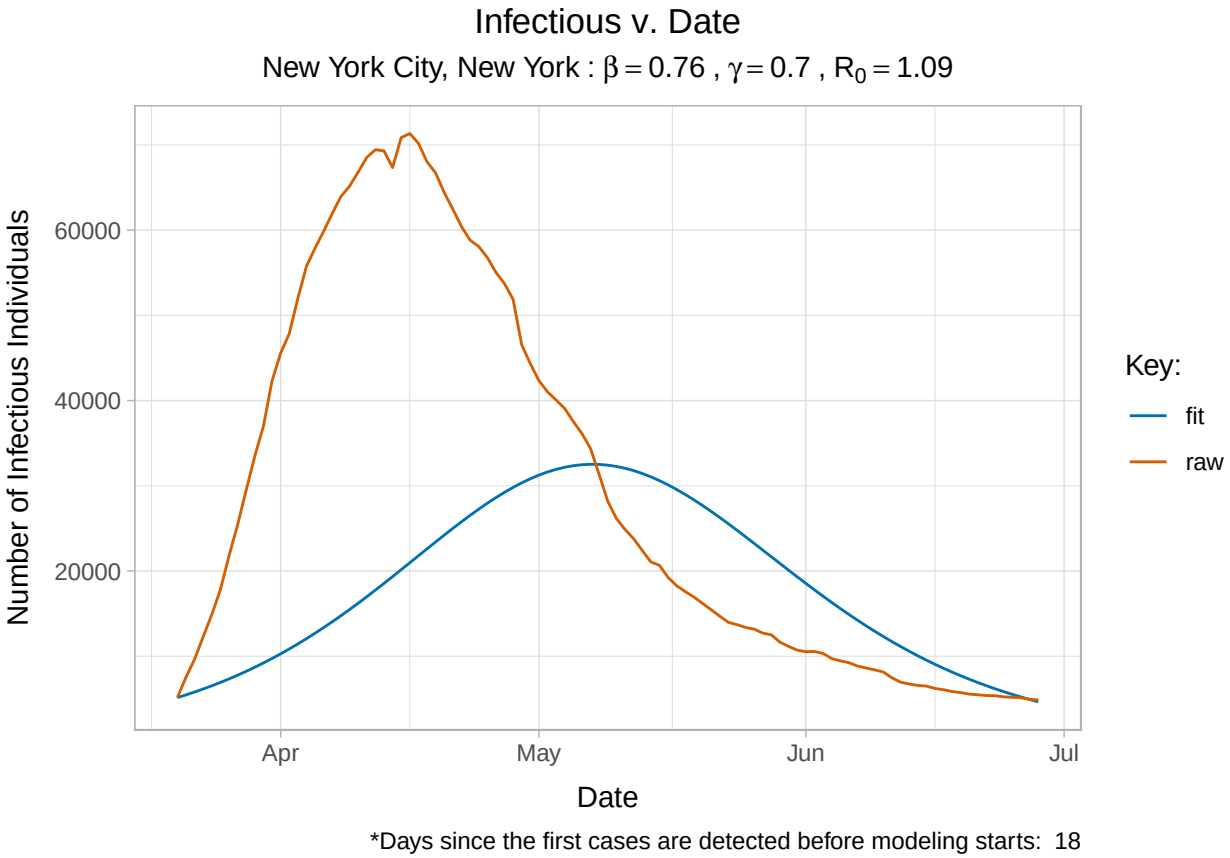
Exploratory data analysis using K-means clustering shows six clusters may be sufficient to group the data based on population and grace period. Ideally we would like to perform clustering based on population density instead of population, however the land area data for the US was not easy to obtain because the US Census data assigns different names to places than the JHU CSSE data set. Furthermore, only land area data from 1990 is available. In principle the data could be obtained using modern tools such as fuzzy join, ie:

```
## Define a vectorized-function for evaluating fuzzy logic
fuzzy <- Vectorize( function(x, y) {

  return(isTRUE(as.logical(agrep(x, y))))

})


## Perform fuzzy join
testing_df2 <- fuzzy_inner_join(
  jhu_US_data,
  census_data,
  by = c(

    "State"  = "State_Name",
```

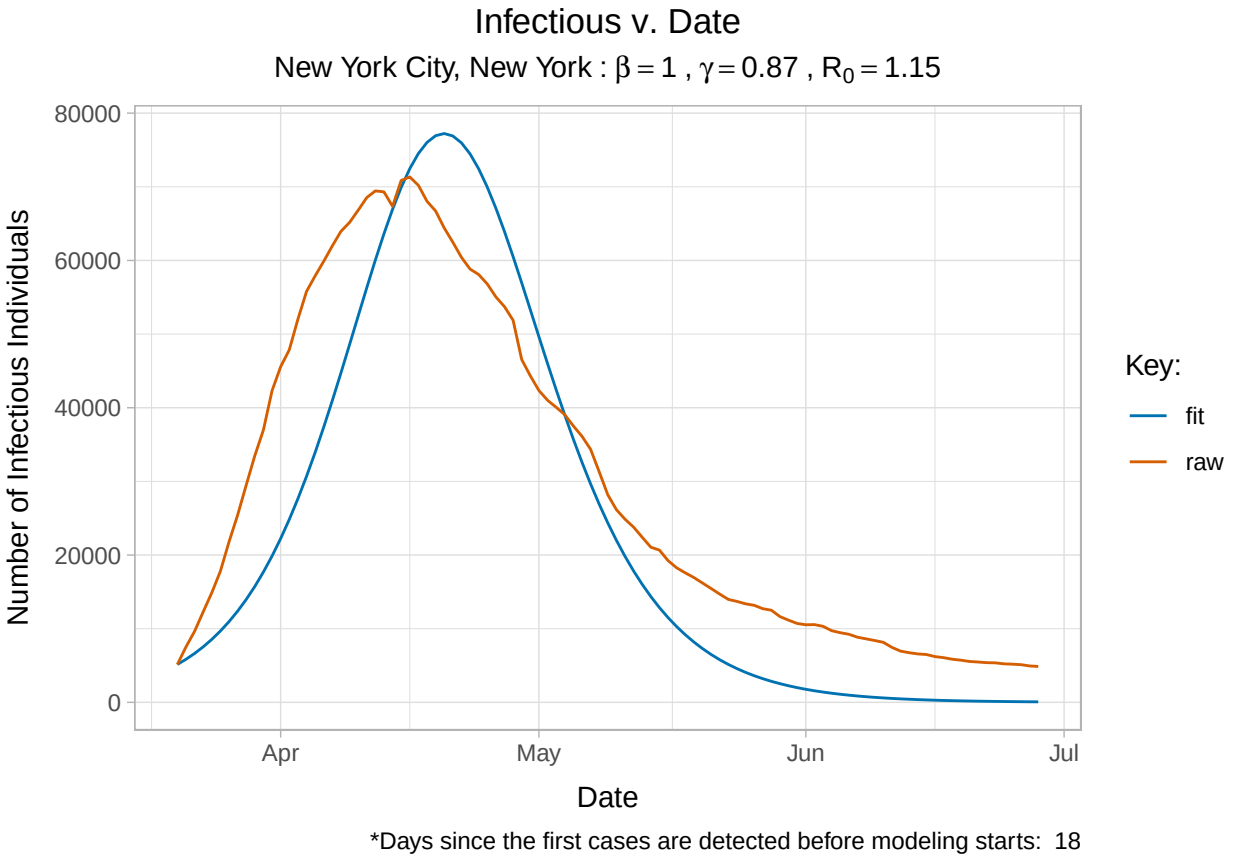
```
    "Locale" = "Locale_Name"

    ),
  match_fun = fuzzy

)
```

The fit using the SIR model is bad when staring from a low number of infectious individuals (data not shown). The fit improves when applied at a later date, when the number of cases is larger. Larger initial values for the number of infectious individuals usually results in better fits. The choice of 5000 infectious/infected is chosen because it permits good fitting of the data for Cook, IL. The heat map (shown above) of RSS values for the range of beta and gamma values tested reveals that there is a wide range of possible beta and gamma pairs. The model parameters from optimization are $\beta = 0.76$, and $\gamma = 0.7$, which produce $R_0 = 1.09$. Data for Los Angeles, CA, and Maricopa, AZ, are somewhat irregular and are not used for model development. The result of using this model to make predictions about New York City, NY, which begins to experience cases a little over a month after Cook, IL, is shown below.

```
#########################################################################
## Apply SIR Model for Prediction #######################################
## Apply model gleaned from Cook, Illinois to New York City, New York ##
#########################################################################


## Set locale
the_location = "New York City, New York"

## Set lag based on when 5000 people become infected/infectious
the_lag <-
  as.numeric(date_of_X_infectious(the_location) - get_first_cases_date(the_location))


## Use informed guess for beta and gamma based on optimized fit for Cook, Illinois
## So keep beta and gamma, and R_0 remains unchanged
fit <- fit_SIR_model(params_optim, the_location, the_lag)
raw <- get_raw_data(the_location, the_lag) %>%
  mutate(time = seq(1, length(date)))
```



It is apparent that the prediction made for NYC, NY, based on Cook, IL, is not good. It does not accurately predict the time course nor the magnitude of the infectious population. Fitting the SIR model can be applied directly to the NYC, NY, data given the outbreak in NYC is now already well under way. We optimize the fit of the SIR model to the data for NYC, NY, starting with the optimal parameters for Cook, IL. The SIR model is fairly sensitive to values of β and γ, and modest adjustments can lead to significant improvements. Convergence is achieved, which results in the following model fit.



Discussion

The basic question we are trying to answer through modeling is after the first COVID-19 cases are observed in a locale how long will it be until the surge occurs if no interventions take place? However, the SIR model does not allow us to accurately make this prediction. The SIR model is only able to fit the data once the outbreak is already in progress. Others have responded to this ambiguity by modeling the cumulative number of cases, the raw JHU data, rather than attempt to estimate the number of infectious individuals (Churches 2020). While this approach has no theoretical justification for implementing the SIR model the data may nonetheless more accurately represent the infectious population, but only in the expansion phase. As such, adequate testing would certainly improve modeling efforts. In addition, we had to make gross estimates about the infectious and recovered compartments of the SIR model due to a woeful lack of data on recoveries. States with thousands of COVID-19 cases like California, Georgia, Illinois, Florida, and Massacheusets have little or no data on recoveries in the JHU CSSE data sets. However, the SIR model in itself may not be sufficiently complex to describe disease spread due to the unique characteristics of COVID-19 and due to the complexity of actual human interactions. Its biggest drawback is that it gives predictions for the rate of infection when cases are on the rise that is initially lower than what is actually observed, as seen in the fit for NYC, NY.

Clustering is an important facet of modeling the spread of infectious disease. It is an attempt to compartmentalize (not to be confused with compartments of the SIR model) some of the epidemiological complexity by grouping entities (locales/subpopulations) that share certain key characteristics. Population and grace period are clearly not sufficient clustering parameters. Population density would likely be a step in the right direction. The parameter γ, related to the recovery time, is intrinsically related to the virus. Cohort studies, in which the population of infectious individuals is roughly controlled for, can greatly improve estimates for γ. Nonetheless, the parameter β, while related to the virus, is also dependent on the locale where the virus

is spreading. Therefore, adequate testing of the population as well as continued monitoring of those infected are crucial for applying modeling efforts to control the spread of COVID-19.

References

Byrd, Richard H., Peihuang Lu, Jorge Nocedal, and Ciyou Zhu. 1995. "A Limited Memory Algorithm for Bound Constrained Optimization." *SIAM Journal of Scientific Computing* 16 (September): 1190–1208. https://doi.org/10.1137/0916069.

Choisy, Marc. 2018. "SIR Models in R." https://rpubs.com/choisy/sir.

Churches, Tim. 2020. "Tim Churches Health Data Science Blog: Analysing Covid-19 (2019-nCoV) Outbreak Data with R - Part 1." https://timchurches.github.io/blog/posts/2020-02-18-analysing-covid-19-2019-ncov-outbreak-data-with-r-part-1/.

Dandekar, Raj. 2020. "Safe Blues: A Method for Estimation and Control in the Fight Against Covid-19." https://safeblues.org/.

Hastie, Trevor, Robert Tibshirani, and Jerome H. Friedman. 2009. *The Elements of Statistical Learning*. Springer, New York, NY, https://doi.org/10.1007/978-0-387-84858-7.

Hill, Alison. 2020. "Modeling Covid-19 Spread V. Healthcare Capacity." https://alhill.shinyapps.io/COVID19seir/.

Siedner, Mark J, Guy Harling, Zahra Reynolds, Rebecca F Gilbert, Sebastian Haneuse, Atheendar Venkataramani, and Alexander C Tsai. 2020. "Social Distancing to Slow the Us Covid-19 Epidemic: Longitudinal Pretest-Posttest Comparison Group Study." *medRxiv*. https://doi.org/10.1101/2020.04.03.20052373.

WHO. 2020. "WHO Director-General's Opening Remarks at the Media Briefing on Covid-19 - 24 February 2020." https://www.who.int/dg/speeches/detail/who-director-general-s-opening-remarks-at-the-media-briefing-on-covid-19---24-february-2020#:~:text=They%20found%20that%20for%20people,three%20to%20six%20weeks.