SIR Modeling of COVID-19

Leo PeBenito

12/24/2020

Preliminary Information

The documentation presented provides **R** access to the JHU CSSE COVID-19 time series data sets, and introduces **R** utilities suitable for epidemiological analysis. Heavy emphasis is placed on the **R** programming language. R markdown is used often with R-chunks displayed without concealing the underlying code as a means of self-documentation. In this spirit, explanatory text is often relegated to the comments within code chunks, and is customarily terse. A forewarning, functional programming encourages composition of functions, which facilitates a modular framework that is simple to modify, such that compartmentalization of independent tasks makes it easy to recycle and swap out code. However, from the perspective of documentation, breaking up a single task into many small sub-routines can make it feel like one has to hang on for some time before reaching the punch-line. Considering this caveat, codes that are related are kept as close together and in as natural an order as possible. With rare exceptions, every attempt is made to "stay DRY" (Don't Repeat Yourself). Hopefully the flow is logical enough that related code is within close proximity so that finding relevant dependencies and documentation is relatively quick and straight forward.

Introduction

Epidemiological models have a long history in the fight against infectious disease. An important ability of such models is to enable prediction of 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.

R implementation of the classic SIR model is applied 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 CSSE time series data is stored as cumulative case counts at the county/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 (in addition to deaths) is the complement to the cases data from which the number of infectious individuals can be roughly inferred. Without this necessary data a crude estimation of the infectious period is made based on the report from the CDC (Center for Disease Control and Prevention) (CDC 2020) to approximate the number of infectious individuals.

The goal of the project is to provide a framework for predicting the spread of COVID-19. At the outset, exploratory data analysis is performed to chart the spread and general characteristics of the pandemic across U.S. states in North America. Next, COVID-19 spread is considered at the locale level, and optimization of the SIR model is explored.

Two approaches to predicting spread of COVID-19 at the locale level are considered. The first approach considered is to use one locale, which exhibits cases earlier, to make a prediction about another locale that witnesses cases later on. K-means clustering is used as the basis for comparison between locales. For availability, the clustering variables are chosen based on their presence within the JHU CSSE COVID-19

data set. Two locales from the same cluster are considered similar such that there is a basis for comparison between them. Two locales are chosen that are consistently in the same cluster over a range of k-values (number of clusters). Beyond this the choice of locales is made on a geographical basis, one from the West Coast and the other from the East Coast. The SIR parameters from one North West locale are used to predict the spread of COVID-19 in a North East locale, and the prediction is compared with the actual data.

The second approach for predicting COVID-19 spread is to focus on a single locale and to use data from the recent past to make a short-term forecast for the immediate future. SIR parameters are obtained for a segment of the data and the model is continued into the future using these parameters. Confidence intervals for the fitted segment are used to estimate the error associated with the forecast. Finally, the forecast is compared with the actual data.

Methods

The tidyverse is used for data wrangling. K-means clustering is performed using `kmeans` from the `stats` package. The system of differential equations is solved using `ode` of the `deSolve` package. Optimization is performed using the L-BFGS-B algorithm with `optim` from the `stats` package. Confidence intervals are computed using maximum likelihood estimation (MLE) with `mle2` from the `bbmle` package.

R Environment

```
##                _
## platform       x86_64-w64-mingw32
## arch           x86_64
## os             mingw32
## system         x86_64, mingw32
## status
## major          3
## minor          6.3
## year           2020
## month          02
## day            29
## svn rev        77875
## language       R
## version.string R version 3.6.3 (2020-02-29)
## nickname       Holding the Windsock
```

R Packages

```
##  [1] "furrr"       "future"      "ggrepel"     "sf"          "viridis"
##  [6] "viridisLite" "bbmle"       "stats4"      "deSolve"     "DMwR"
## [11] "grid"        "lattice"     "data.table"  "lubridate"   "forcats"
## [16] "stringr"     "dplyr"       "purrr"       "readr"       "tidyr"
## [21] "tibble"      "ggplot2"     "tidyverse"   "gridExtra"   "knitr"
## [26] "stats"       "graphics"    "grDevices"   "utils"       "datasets"
## [31] "methods"     "base"
```

Parallelization Setup (optional)

If the `furrr` package is installed this code will setup `future_map`-functions to run in parallel. Comment out this section if you do not want to run the code in parallel.

```r
# Parallelization setup (optional, un-comment to activate)
future::plan(multiprocess) # One-time start up cost
#future::plan(sequential) # restore serial setup
```

Load COVID Data

```r
# Download data sets for the United States from the GitHub repository maintained
# by Johns Hopkins University Center for Systems Science and Engineering (CSSE)
# To run this code requires an internet connection.

# CSV file of time series of confirmed cases in the United States
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 in the United States
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))
```

Load Geospatial Data

These files are needed for generating maps.

```r
# Download shapefiles for the USA from the US Census Bureau web page.
URL <- "https://www2.census.gov/geo/tiger/TIGER2019/STATE/tl_2019_us_state.zip"

# Setup directory and download shapefile zip.
subDir  <- "Shapefiles/"

ifelse(!dir.exists(file.path(mainDir, subDir)),
       dir.create(file.path(mainDir, subDir)), FALSE)

shape_zip <- paste0(mainDir, subDir, "tl_2019_us_state.zip")
download.file(url = URL, destfile = shape_zip)

# Unpack zip file
shape_dir <- paste0(mainDir, subDir, "tl_2019_us_state")
unzip(zipfile = shape_zip, exdir = shape_dir)

# Define Area of interest: read shapefile
```

```
shape_file <- paste0(mainDir, subDir,
                     "tl_2019_us_state/", "tl_2019_us_state.shp")
aoi_boundary <- st_read(shape_file, quiet = TRUE)
```

Extract COVID-19 Data

```
# 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 data from cruise ships
# Presumably these people were quarantined upon arrival and did not contribute
# to spread of COVID19
confirmed_US <- confirmed_US %>%
  filter(!(Province_State == "Grand Princess" |
             Province_State == "Diamond Princess"))
deaths_US <- deaths_US %>%
  filter(!(Province_State == "Grand Princess" |
             Province_State == "Diamond Princess"))


# 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 field 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 ", US" from Locale
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
jhu_US_data <- inner_join(
  x = confirmed_US,
  y = deaths_US,
  by=c(
    "State" = "State",
    "Locale" = "Locale",
    "Date" = "Date"
    ),
  keep=FALSE)
```

Clean COVID-19 Data

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

# Find Locales that never record any cases
# Side Note: entries with bad Locale/Combined_Key:
# "Weber,Utah,US", "District of Columbia,District of Columbia,US"
locales_with_zero_cases <- jhu_US_data %>%
  group_by(Locale) %>%
  summarize(sum_cases = sum(Cases), .groups = 'drop') %>%
  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 #
```

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

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


# Remove entries for which the population is equal to zero from population data
population_data_US <- population_data_US %>%
  filter(Population > 0)

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

Explore JHU CSSE COVID-19 Data

Data exploration begins with the temporal and spatial evolution of the COVID-19 pandemic at the state level. Geographical maps are used to plot the succession of recorded cases among U.S. states in North America as well as basic cases and deaths data. A heat map is used to chart the progression of COVID-19 cases over time since the beginning of the outbreak to the present. Given this bird's eye view of the pandemic's origins and trajectory, as well as of what regions to consider first, the focus is then narrowed down to the locale level.

Two approaches to utilizing the SIR model based on realistic scenarios are considered. The first is to use data from one locale to make a prediction about the outcome for another locale that experiences cases at a later date. K-means clustering is used to identify locales for training and testing data sets. Population and grace period are used as two variables for performing clustering. Grace period is the number of days since the first cases are reported in the U.S. and when a locale begins reporting cases. Two locales that are consistently in the same cluster over a range of k-values (number of clusters) are chosen for analysis. The locale that experiences cases earlier is used as the training data set, while the latter is treated as the testing data set.

A case study is performed on the training set to explore optimization of the SIR model. The influence of the initial values for the compartments of the SIR model are examined by modifying the start date for fitting the data for an increasing number of the infectious individuals. Optimization of the model fit is performed with the L-BFGS-B algorithm using the residual sum of squares (RSS) as the loss function. To supply an initial guess for the SIR parameters for fit-optimization first a scan of the parameter space is performed, and the set of parameters that minimize the RSS is chosen. The case study produces an optimized SIR model from the training data, which is then used to predict the time course of the infectious population for the testing data.

The second approach focuses on a single locale and makes a forecast about the immediate future based on the recent past. In this scenario, the data from the recent past is the training data and the future data that the forecast aims to predict is the testing data. The SIR parameters obtained from fitting two weeks of training data are used to make a one week forecast. Confidence intervals for the fit of the training data are obtained using maximum likelihood estimation (MLE), and are used to estimate the error associated with the forecast. The forecast is evaluated based on the RSS and whether the real data remains within the forecast error.

Analysis Setup

R codes for analysis of the raw JHU CSSE COVID-19 data are presented here.

```r
# Purpose: define the variables and functions needed to explore the
# Johns Hopkins COVID-19 time series data set.
# A location is specified at the state or locale level.
# A locale is the location (place) associated with a corresponding time series.
# Locale specifications include the state name separated by a comma.


# Create stamp (function) for stamping figures with date info.
# Note: using words with numbers throws off date-format detection, ie: COVID-19.
stampf <- stamp(
  x = "(JHU CSSE data obtained on 1 January 1970)",
  orders = "dmy",
  quiet = TRUE
)


# List of all states, commonwealths, and territories
states  <- levels(factor(jhu_US_data$State))

# List of all locales (cities/towns)
locales <- levels(factor(jhu_US_data$Locale))


# Get population
get_pop <- function(location) {

  # Calculate the population for the given location
  if (location %in% states) {

    N <- population_data_US %>%
      group_by(State) %>%
      summarize(state_pop = sum(Population), .groups = "drop") %>%
      filter(State == location) %>%
      pull(state_pop)

  }

  else if (location %in% locales) {

    N <- population_data_US %>%
      filter(Locale == location) %>%
      pull(Population)

  }

  else {

    stop("Error: get_pop() - Invalid location spec")

  }


  return(N)
```

```
}


# Date that cases start to appear
get_init_cases_date <- function(location) {

  if (location %in% states) {

    date <- jhu_US_data  %>%
      filter(State == location, Cases != 0) %>%
      select(Date)  %>%
      arrange(Date) %>%
      pull(Date) %>%
      head(1)

  }

  else if (location %in% locales) {

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

  }

  else {

    stop("Error: get_init_cases_date() - Invalid location spec")

  }


  return(date)

}


# Date of the most recently reported cases
get_last_cases_date <- function(location) {

  if (location %in% states) {

    date <- jhu_US_data %>%
      filter(State == location, Cases != 0) %>%
      select(Date) %>%
      arrange(desc(Date)) %>%
      pull(Date) %>%
      head(1)

  }
```

```
  else if (location %in% locales) {

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

  }

  else {

    stop("Error: get_last_cases_date() - Invalid location spec")

  }


  return(date)

}


# Dates between when 1st and last cases reported
# If dates are supplied check their validity
get_COVID_dates <- function(location = test_locale) {

  init_cases_date <- get_init_cases_date(location)
  last_cases_date <- get_last_cases_date(location)

  if (location %in% states) {

    dates <- jhu_US_data %>%
      filter(State == location) %>%
      filter(between(Date, init_cases_date, last_cases_date)) %>%
      select(Date)   %>%
      distinct(Date) %>%
      arrange(Date)  %>%
      pull(Date)

  }

  else if (location %in% locales) {

    dates <- jhu_US_data %>%
      filter(Locale == location) %>%
      filter(between(Date, init_cases_date, last_cases_date)) %>%
      select(Date)  %>%
      arrange(Date) %>%
      pull(Date)

  }
```

```
  else {

    stop("Error: get_COVID_dates() - Invalid location spec")

  }


  return(dates)

}


# Check date validity and return indices.
# Arguments are 8 digit dates in ymd format.
# Returns indices for date specified for a given location.
get_date_indices <- function(location = test_locale,
                             delay_days = 0,
                             lag_days = 0,
                             start_date = NULL,
                             stop_date = NULL) {

  dates <- get_COVID_dates(location = location)

  first_date <- dates[1]
   last_date <- dates[length(dates)]

  # If start_date is set and is not NULL
  if (!missing(start_date) & !is.null(start_date)) {

    # Remove punctuation and check format
    if (nchar(str_remove_all(start_date, "[[:punct:]]")) != 8) {

      stop("Error: invalid start_date format")

    }

    # Check range
    else if ((ymd(start_date) < first_date) |
             (ymd(start_date) >= last_date)) {

      stop("Error: start_date outside valid range")

    }

    else {

      # Get index
      start_index <- which(dates == ymd(start_date))

    }

  }
```

```
  # Default starting index
  else if (missing(start_date) | is.null(start_date)) {

    start_index <- 1 + delay_days

  }

  # If stop_date is set and is not NULL
  if (!missing(stop_date) & !is.null(stop_date)) {

    # Remove punctuation and check format
    if (nchar(str_remove_all(stop_date, "[[:punct:]]")) != 8) {

      stop("Error: invalid stop_date format")

    }

    # Check range
    else if ((ymd(stop_date) <  dates[1]) |
             (ymd(stop_date) >= dates[length(dates)])) {

      stop("Error: stop_date outside valid range")

    }

    else {

      # Get index
      stop_index <- which(dates == ymd(stop_date))

    }

  }

  # Default stop index
  else if (missing(stop_date) | is.null(stop_date)) {

    stop_index <- length(dates) - lag_days

  }


  # Final dates/indices range check: auto-correct order start/stop specs
  indices <- sort(c(start_index, stop_index))
  indices <- setNames(indices, c("start_index", "stop_index"))


  return(indices)

}

########################################################################
# Grace period: days between the 1st recorded cases in the US and #
```

```r
# 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)


# Grace period for a given locale
get_grace_period <- function(location) {

  if (location %in% locales) {

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

  }

  else if (location %in% states) {

    grace_period <- jhu_US_data %>%
      group_by(State) %>%
      filter(State == location, Cases != 0) %>%
      mutate(Grace_period = as.numeric(Date - first_cases_date,
                                       units = "days")) %>%
      arrange(Date) %>%
      select(State, Grace_period) %>%
      head(1)

  }

  else {

    stop("Error: get_grace_period() - Invalid location spec")

  }


  return(grace_period)

}


# Get the raw COVID data
# Set the start and stop dates with a single 8 digit number in year-month-day
```

```r
# (ymd) format as single number padding single digit months and days with 0's.
get_COVID_data <- function(location = test_locale,
                           start_date = NULL,
                           stop_date  = NULL) {

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

  indices <- get_date_indices(location   = location,
                              start_date = start_date,
                              stop_date  = stop_date)

  start_index <- indices["start_index"]
  stop_index  <- indices[ "stop_index"]

  # Cases between when 1st and last cases reported
  if (location %in% states) {

    cases <- jhu_US_data %>%
      filter(State == location) %>%
      filter(between(Date, dates[start_index], dates[stop_index])) %>%
      group_by(Date) %>%
      summarize(state_cases = sum(Cases), .groups = "drop") %>%
      pull(state_cases)

    deaths <- jhu_US_data %>%
      filter(State == location) %>%
      filter(between(Date, dates[start_index], dates[stop_index])) %>%
      group_by(Date) %>%
      summarize(state_deaths = sum(Deaths), .groups = "drop") %>%
      pull(state_deaths)

  }

  else if (location %in% locales) {

    cases <- jhu_US_data %>%
      filter(Locale == location) %>%
      filter(between(Date, dates[start_index], dates[stop_index])) %>%
      pull(Cases)

    deaths <- jhu_US_data %>%
      filter(Locale == location) %>%
      filter(between(Date, dates[start_index], dates[stop_index])) %>%
      pull(Deaths)

  }

  else {

    stop("Error: get_COVID_data() - Invalid location spec")

  }
```

```
  return(
    data.frame(
      location = location,
      index = seq(1, stop_index - start_index + 1),
      steps = seq(start_index, stop_index),
      dates = dates[start_index:stop_index],
      cases = cases,
      deaths= deaths
    )
  )

}


# Get total number of COVID cases for a state or locale
get_COVID_totals <- function(location) {

  if ((location %in% states) | (location %in% locales)) {

    data <- get_COVID_data(location = location)
    total_cases <- data$cases[length(data$cases)]
    total_deaths<- data$deaths[length(data$deaths)]

  }

  else {

    stop("Error: total_COVID_cases() - Invalid location spec")

  }

  N <- get_pop(location)


  return(
    data.frame(
      "Location"  = location,
      Total_cases = total_cases,
      Total_dead  = total_deaths,
      Percent_cases = total_cases/N*100,
      Percent_dead  = total_deaths/N*100,
      Mortality  = total_deaths/total_cases*100
    )
  )

}


# Plot raw data
plot_COVID_cases <- function(location,
                             start_date = NULL,
                              stop_date = NULL) {
```

```
  raw <- get_COVID_data(location   = location,
                        start_date = start_date,
                         stop_date =  stop_date)

  raw %>%
    ggplot(aes(x = dates, y = cases)) +
    geom_line() +
    theme_light() +
    labs(
      title = "Raw Cases Data v. Date",
      subtitle = bquote(.(location)),
      x = "Date",
      y = "Number of Recorded Cases"
    ) +
    theme(
      plot.title    = element_text(hjust =  0.5),
      plot.subtitle = element_text(hjust =  0.5),
      axis.title.y  = element_text(vjust =  3.0),
      axis.title.x  = element_text(vjust = -1.0),
      plot.caption  = element_text(vjust = -2.0, hjust = 1.0)
    )

}


# Map of US States in North America color coded according to variable.
# Variables:
# data - is the data frame
# join_column - this column should contain state names.
# variable - is the observable used to generate color gradient scale.
# title - is a string for the plot title
# legend - is a string for the title of the gradient scale.
# caption - is a string for the plot caption.
# Prerequisites:
# aoi_boundary - obtained with sf package
# stampf - defined with lubridate::stamp()
plot_map <- function(data,
                     join_column,
                     variable,
                     title = NULL,
                     legend = NULL,
                     caption = NULL) {

  data %>%
    rename(NAME = {{join_column}}) %>%
    full_join(aoi_boundary,
              by = "NAME") %>%
    filter(!(NAME == "Alaska"
             | NAME == "Hawaii"
             | NAME == "Northern Mariana Islands"
             | NAME == "Virgin Islands"
             | NAME == "District of Columbia"
             | NAME == "Puerto Rico"
```

```
            | NAME == "United States Virgin Islands"
            | NAME == "Commonwealth of the Northern Mariana Islands"
            | NAME == "Guam"
            | NAME == "American Samoa")) %>%
    mutate(NAME = state.abb[match(NAME, state.name)]) %>%
    ggplot(
      aes(
        geometry = geometry,
        fill  = {{variable}},
      )
    ) +
    scale_fill_gradientn(
      colors = c(
        "red", "orange", "yellow", "green", "cyan", "blue", "purple", "magenta"
      )
    ) +
    geom_sf(size = 1, color = "black") +
    geom_label_repel(
      aes(label = NAME, geometry = geometry),
      stat = "sf_coordinates",
      min.segment.length = 0,
      color = "black",
      segment.colour = "black"
    ) +
    coord_sf() +
    theme_void() +
    ggtitle(title) +
    theme(
      plot.title = element_text(hjust = 0.5),
      plot.subtitle = element_text(hjust = 0.5),
      plot.caption = element_text(hjust = 0.5, vjust = -1.0)
    ) +
    labs(
      subtitle = "(U.S. States in North America)",
      caption = bquote(atop(.(caption) ~ "\n", ~ .(stampf(today()))))
    ) +
    guides(
      fill = guide_colourbar(
        title = legend,
        barheight = 9,
        barwidth = 0.5
      )
    )

}
```

State Maps

Geographical maps are used to get a spatial perspective on the spread and impact of the COVID-19 pandemic across U.S. states in North America. Heat maps are used to display summary data, each state is assigned a color along a gradient scale associated with each geospatial map.

Cases appear on the West Coast, in the Midwest, and on East Coast at the start of the pandemic (Fig.1).

Washington and Illinois are the first states to record cases followed by Arizona, California, and Massachusetts. Following these initial recordings there is a significant gap between when other states start recording cases, after which the rest of the country begins recording cases in quick succession.

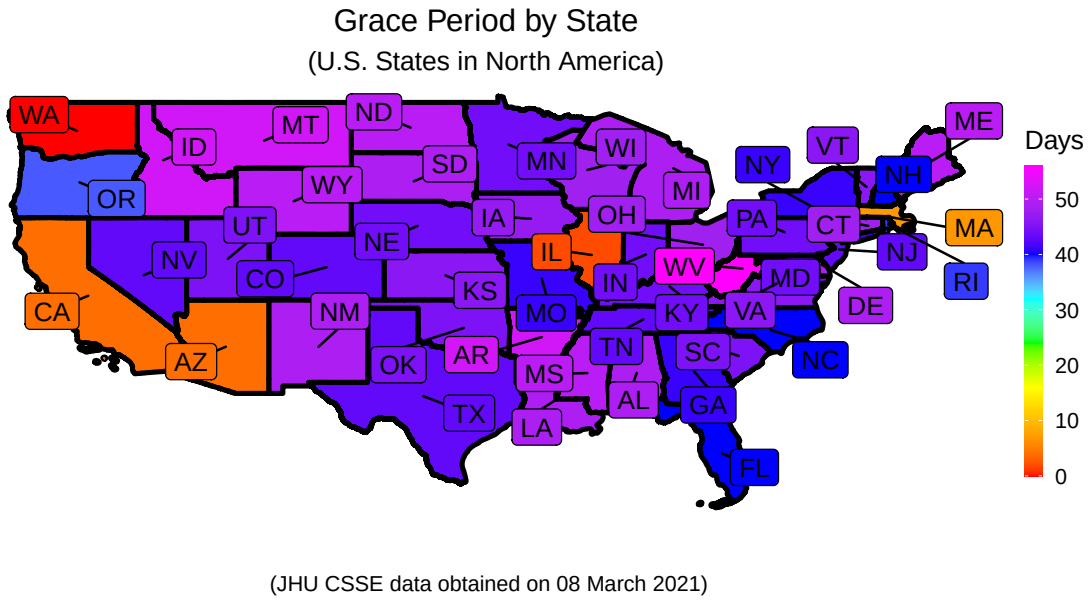


Figure 1: Map of U.S. states in North America color coded along a gradient scale by grace period. Grace period is the number of days since the first COVID-19 cases were recorded in the USA and when the first cases were recorded in a given state.

In general, the states with the greatest number of cases are southern (Fig.2). The North East and the Midwest (particularly Illinois) also tend to have higher case numbers. In the Western U.S. there is a slight red shift toward the northern states indicating fewer cases. In the Eastern U.S. cases are a bit more evenly distributed among the states, but again the most northern states are reddish.

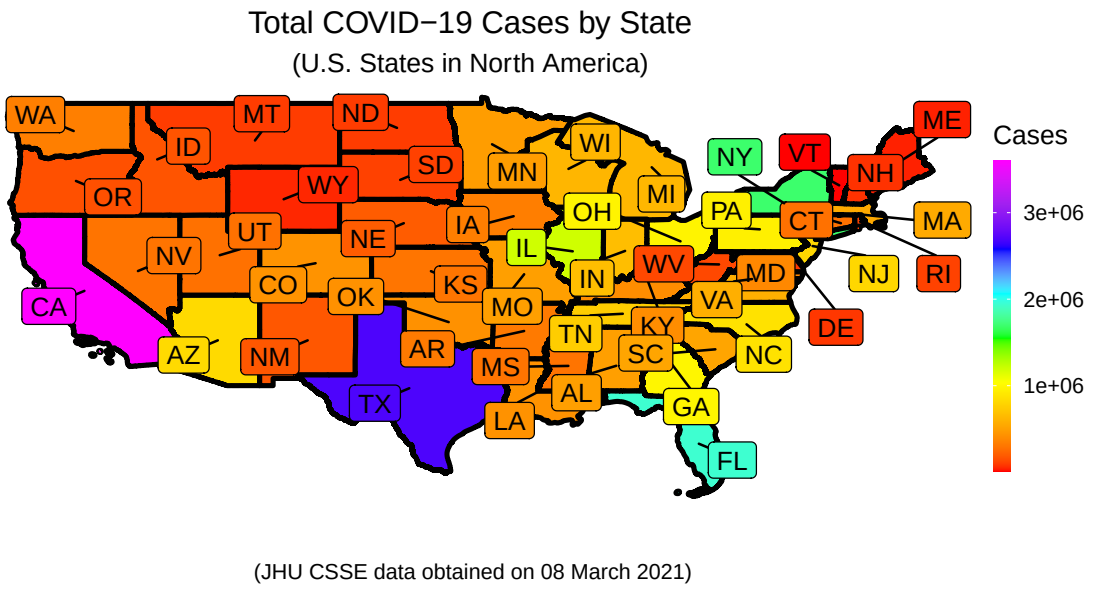


Figure 2: Map of U.S. states in North America color coded along a gradient scale by the number of COVID-19 cases.

The pattern in deaths due to COVID-19 among U.S. states roughly follows that for the number of cases (Fig.3). The North East stands out a little more in terms of the number of deaths.

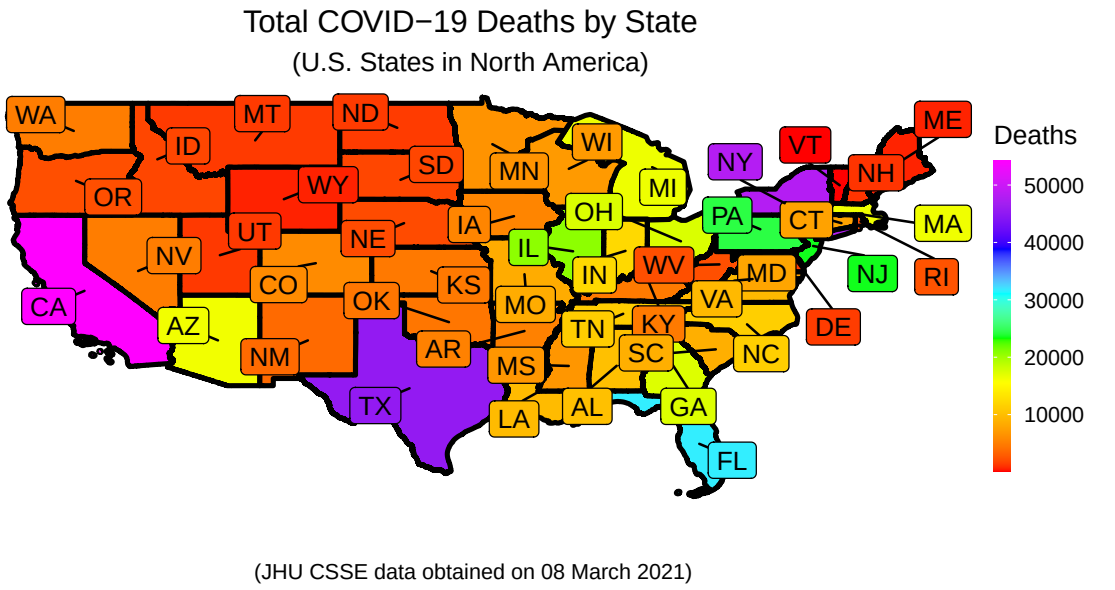


Figure 3: Map of U.S. states in North America color coded along a gradient scale by the number of COVID-19 deaths.

While the north central states do not show high case numbers it is apparent that they have not been less affected than their state counterparts. Based on the percent of the state population that has become infected these states have suffered the worst (Fig.4).

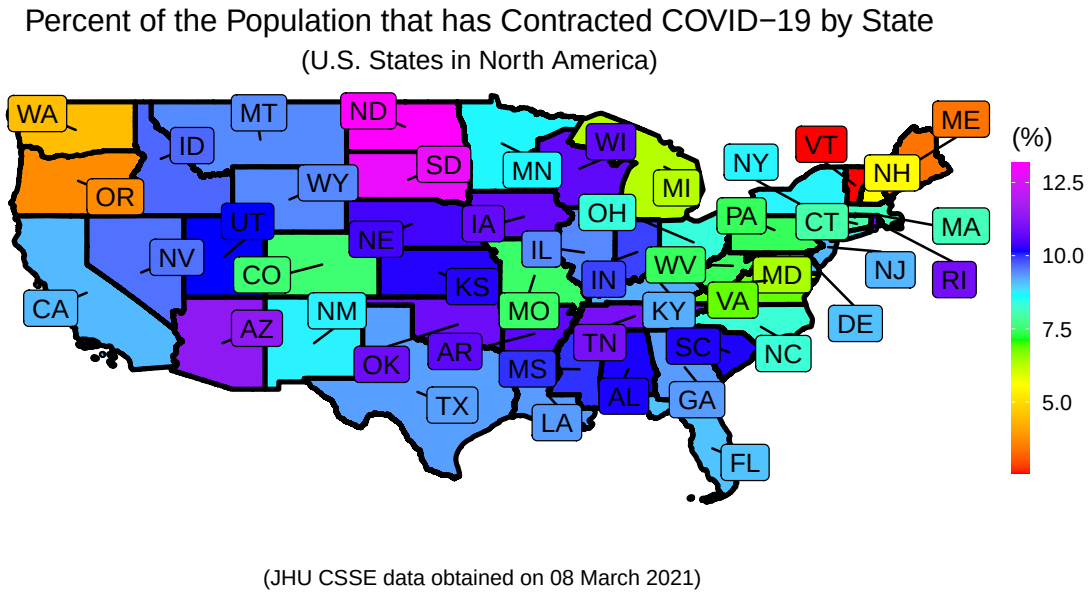


Figure 4: Map of U.S. states in North America color coded along a gradient scale by the percent of the population that has been infected by COVID-19.

In terms of the percentage of the population of the state that has died from COVID-19 most states are at around 0.1 % or below (Fig.5). The North East as well as North and South Dakota are clear exceptions. Louisiana and Mississippi do not generally present as having been significantly affected but are also on the higher end of the spectrum in terms of reduction of state population.

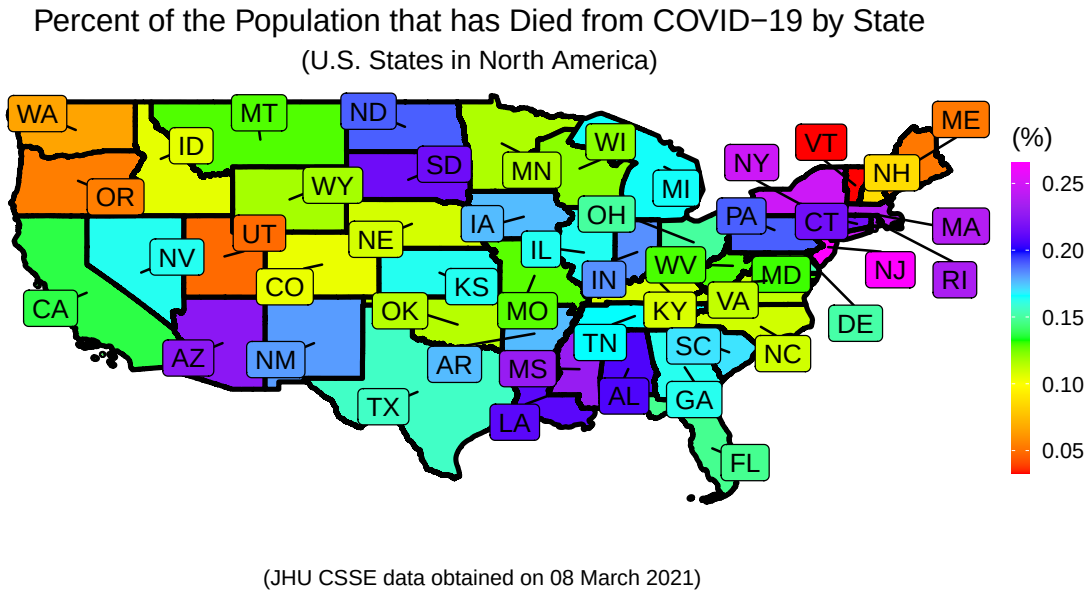


Figure 5: Map of U.S. states in North America color coded along a gradient scale by the percent of the population that has died from COVID-19.

Mortality due to COVID-19 is calculated based on the WHO's definition of case fatality ratio (CFR, in %) as (WHO 2020)

$$\text{Mortality} = \frac{\text{Total No. Deaths}}{\text{Total No. Cases}} * 100$$

The North Eastern U.S. stands out as having the worst outcomes for those who become infected, particularly New York, New Jersey, Massachusetts, and Connecticut (Fig.6).

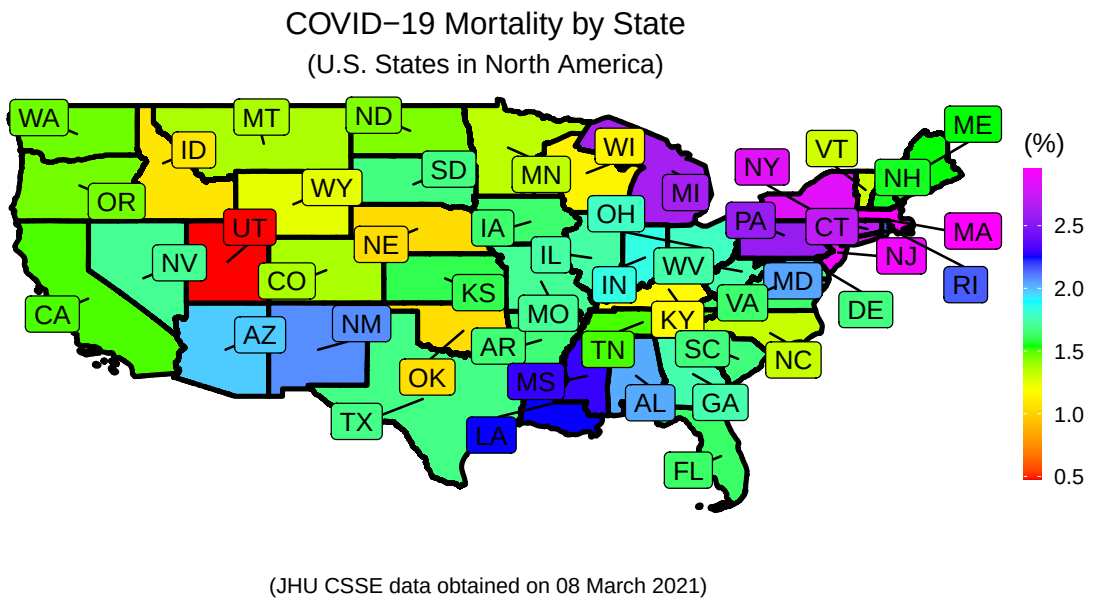


Figure 6: Map of U.S. states in North America color coded along a gradient scale by COVID-19 mortality.

Time Series

A heat map is used to represent time series data of COVID-19 cases for U.S. states in North America (Fig.7). Colors along the gradient scale correspond to the number of cases. This facilitates comparison between the states, which can be given distinct rows along the y-axis as each time series proceeds along the x-axis. The heat map shows coming up on a year since the start of the pandemic that the number of cases continues to rise across the United States. As of December 2020, Texas, California, Florida, New York, and Illinois have not seen a plateau in the number of cases since July 2020. Regular blurred vertical columns may appear across the plot, which are artifacts of the PDF file rendering using SumatraPDF, and do not appear with other programs, ie: Microsoft Edge.

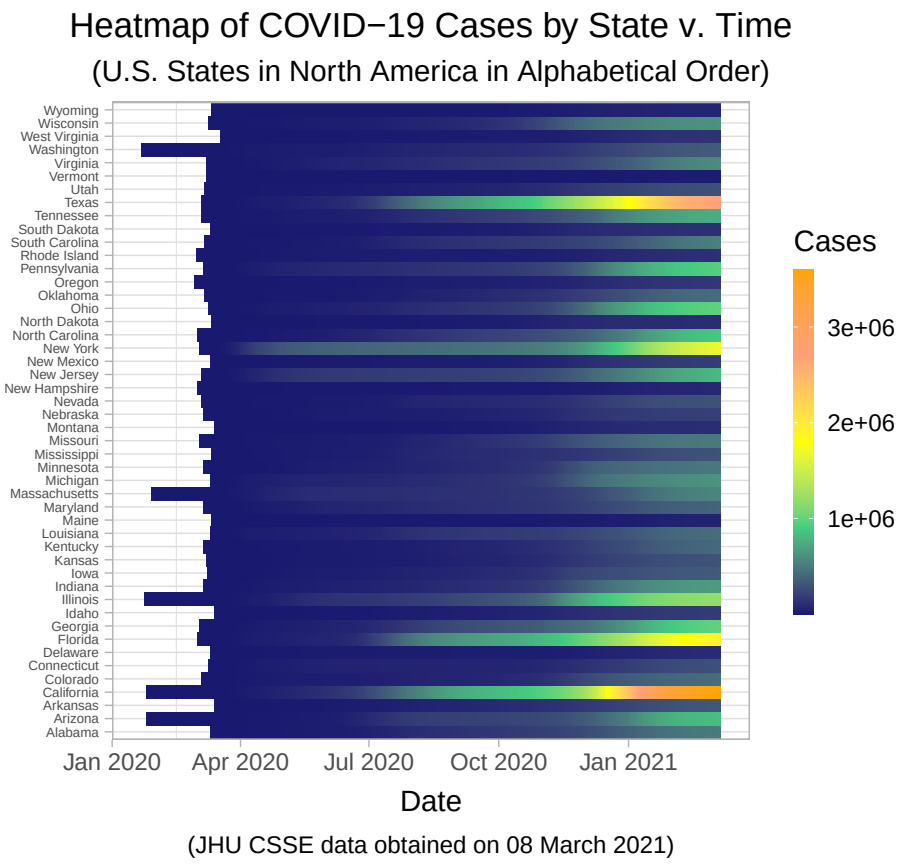


Figure 7: Heat map of the time series of COVID-19 cases across U.S. states in North America.

K-means Clustering

In the first approach considered for predicting the course of the pandemic at the locale level the SIR model applied to one locale is used to anticipate the results for another locale that experiences cases at a later date. The former data set provides the training data set and the latter furnishes the testing data set. However, locales cannot simply be arbitrarily chosen for comparison. Population and grace period are used to perform K-means clustering of locales. Grace period is the number of days between the start of the pandemic and when cases are first recorded in a given locale. Z-scores are computed for each variable to normalize the distance calculation. The choice of variables is limited to those available within the JHU CSSE COVID-19 data set for convenience. A more elaborate clustering scheme is likely to yield better results.

Explore K-means Clustering

Similarity between clusters tends to improve with an increasing number of clusters. The intra-cluster variation is used to examine the similarity between locales within a cluster. The percent of variance explained is used to determine how much of an improvement is made with an increase in the number of clusters. Both are plotted versus increasing k-value below (Fig.8). The results indicate that little information is gained beyond six clusters.

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

```r
# Combine grace period data with population data
kmeans_setup <- inner_join(x = locale_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)

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

# Parameters:
# (1) population
# (2) Grace period
# Note: the cluster ID can change from one K-means run to the next

set.seed(2394817)

wss <- function(k) {

  kmeans(
    x = kmeans_setup,
    centers = k,
    nstart = 25,
    iter.max = 20
  )$tot.withinss

}

# Sequence of k-values to test
k_seq <- seq(1, 10)


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

# Plot total intra-cluster variation
# (total within-cluster sum of squares) v. the number of clusters
data.frame(x = k_seq, y = wss_seq) %>%
  ggplot(aes(x, y)) +
  geom_point() +
  theme_classic() +
  theme(
```

```
    aspect.ratio = 1/1,
    plot.title = element_text(size = 10, face = "bold", hjust = 0.5)
  ) +
  labs(
    title = "Total Intra-Cluster Variation",
    x = "k-value (No. clusters)",
    y = "Within-Cluster Sum of Squares"
  ) -> p1


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

data.frame(x = k_seq[2:length(k_seq)], y = round(pve, 2)*100) %>%
  ggplot(aes(x, y)) +
  geom_point() +
  geom_hline(yintercept = 25, linetype = "dashed", color = "red", size = 1) +
  theme_classic() +
  theme(
    aspect.ratio = 1/1,
    plot.title = element_text(size = 10, face = "bold", hjust = 0.5)
    ) +
  labs(
    title = "Percent of Variance Explained v. K",
    x = "k-value (No. clusters)",
    y = "Percent of Variance Explained (%)"
  ) -> p2
```

```
grid.arrange(p1, p2, ncol = 2)
```

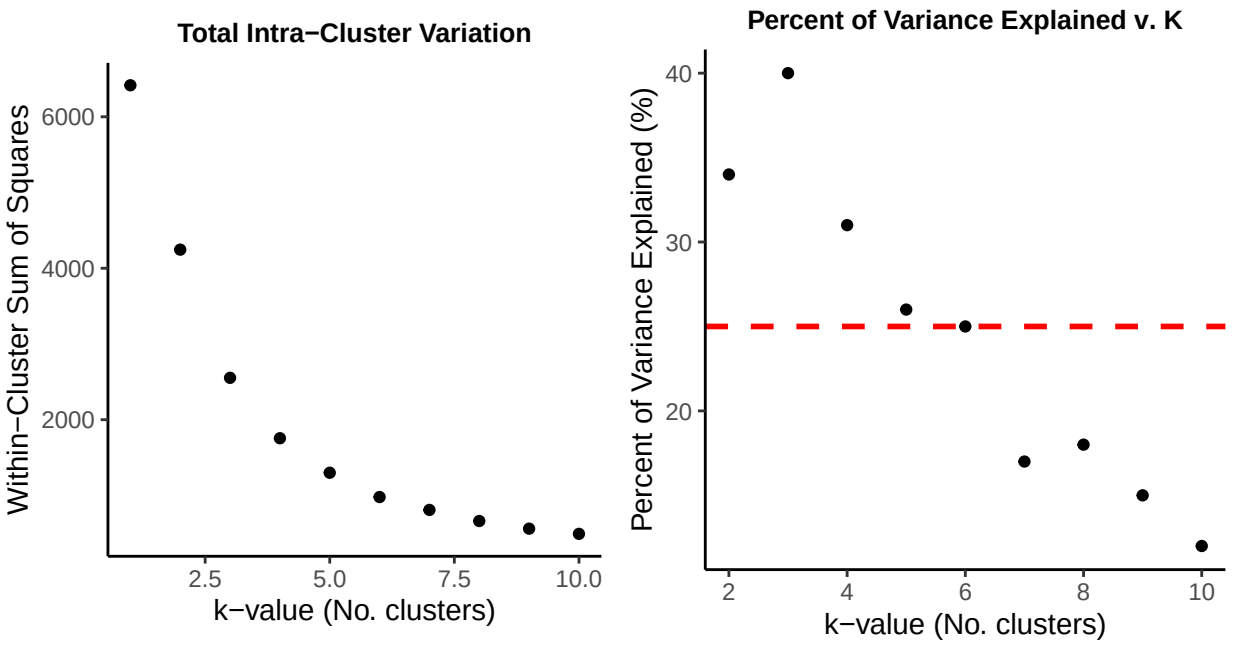


Figure 8: The intra-cluster sum of squares (Left) and the percent of variance explained are examined for increasing k-values (an increasing number of clusters).

K-means Clustering Plots

King, Washington is the first locale to begin recording cases, so it is the natural choice for the training data set. Queens, New York is consistently in the same cluster as King, Washington for the range of k-values (2-7) examined (Fig.9). Queens, New York is on the opposite coast but at at a similar latitude so it provides a useful contrast for defining the testing data set.

Illinois is the second state to record COVID-19 cases and based on the cases map (Fig.2) could be the origin for the pandemic in the Midwest. Based on the clustering data, Cook is representative of large cities. For this reason, as well as to extend the geography of the analysis to an important region, Cook, Illinois is selected as the locale for forecasting analysis.

```
# Note: the cluster ID can change from one K-means run to the next
kmeans_clustering <- function(data = kmeans_setup,
                              k,
                              n_start = 25,
                              iter_max = 20) {

  # Use unscale of DMwR package to reverse action of scale() on kmeans_setup
  require("DMwR")

  # Generate 6 clusters
  kmeans_data <- kmeans(
```

```
    x = data,
    centers  = k,
    nstart   = n_start,
    iter.max = iter_max
  )


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

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

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

  # Join clustering data with the variables used for clustering
  # *** Variables used for clustering have been 'unscaled',
  # (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")
  )


  return(clustering_data)

}


# Get clustering data for range of k values (number-of-clusters)
base_name <- "kmeans_clusters"

for (i in 2:7) {

  cluster_name <- paste0(base_name, i)
  assign(cluster_name, kmeans_clustering(k = i))

}
```

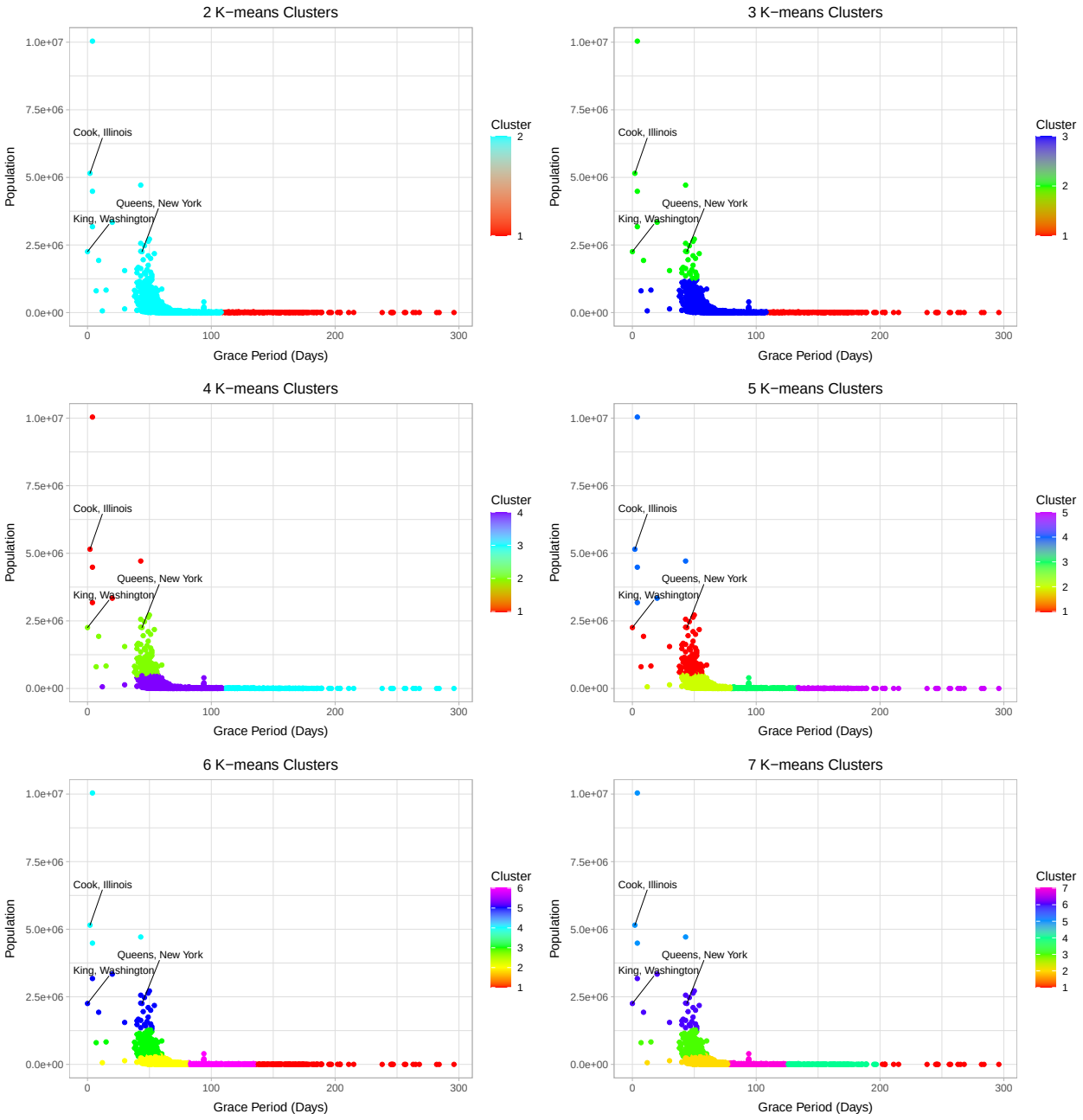


Figure 9: Plots of K-means clustering of U.S. locales in the JHU CSSE data set by population and grace period for successive k-values. Grace period is the number of days between when the 1st cases were recorded in the U.S. and when the 1st cases are recorded for a given locale.

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. Beta (β) controls the transition between S and I. Gamma (γ) controls the transition between I and R. The basic reproductive number, known as R_0, is equal to the ratio of β/γ.

To evaluate the efficacy of the model the loss function is chosen as the residual sum of squares for time evolution of the the number of infectious individuals

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

, where $\hat{I}(t)$ is the SIR estimate for the infectious population over time and $I(t)$ is the estimate for the infectious population based on the raw data.

SIR Data Extraction

The JHU CSSE COVID-19 time series data stores cumulative case counts for each locale. However, the SIR model is largely based on the number of infectious individuals. To compute the number of infectious individuals over time is complicated. A simple approach is adopted based on the CDC's estimate for the infectious period of between 10 to 20 days (CDC 2020). An even number of weeks in this range, 14 days is used as the infectious period, for simplicity. To determine the number of infectious individuals over time, two weeks into the time series the cumulative number of cases two weeks prior is subtracted from the value on a given day. Following the infectious period an individual enters the recovered compartment, which includes deaths. The susceptible compartment is the total population minus the recovered compartment as well as the current number of infectious people. **R** codes for SIR analysis are presented here.

```
# Purpose: generate SIR data from the raw cases data using the simplest set
# of assumptions.
# Estimate the no. infectious individuals as:  Infectious = Cases - Recovered
# A simplifying assumption applied is that one remains infectious while ill.
# Use the ave. recovery time to estimate the no. individuals who have recovered.
# WHO estimates one is infectious for 2 weeks, while symptoms can last up to 6+.
# Recall: there is no distinction made between recovered and deceased.
# Shifts data to the right changing leading positions now with missing values to
# 0 and truncating trailing values past the length of the input vector.
default_infectious_period = 14

# Note: infectious_period = 0 is unacceptable.  According to the current
# implementation this would lead to cases = recovered, which results in the
# population of infectious = 0 for all time! Therefore, infectious_period <= 0
# should be interpreted as no recoveries, which amounts to simply working with
# the raw cases data with an arbitrary start date.
# delay (default: 0) skip no. days after the 1st cases were observed.
# lag   (default: 0) omit no. days prior to current day.
# NOTE: date-range specs override delay/lag specs,
# ie: compute boundary indices using dates if they are supplied.
get_SIR_data <- function(location = test_locale,
                         infectious_period = default_infectious_period,
                         delay_days = 0,
```

```
                          lag_days = 0,
                        start_date = NULL,
                         stop_date = NULL) {

# Dates and cases between when 1st and last cases reported
raw_data <- get_COVID_data(location)

# Raw COVID data
dates <- raw_data$dates
cases <- raw_data$cases

# Check start_date and stop_date (used to subset data)
indices <- get_date_indices(location   = location,
                            delay_days = delay_days,
                              lag_days =   lag_days,
                            start_date = start_date,
                             stop_date =  stop_date)

# Final dates/indices range check: auto-correct order start/stop specs
start_index <- indices["start_index"]
stop_index  <- indices[ "stop_index"]


# Get population of given locale
N <- get_pop(location)


# Estimate evolution of the recovered, infectious, and susceptible populations
if (infectious_period <= 0) {

  recovered <- rep(0, length(cases))

}

else {

  recovered <- c(
    rep(0, infectious_period),
    cases[seq(1, length(cases) - infectious_period)]
  )

}


infectious  <- cases - recovered

susceptible <- N - infectious


return(
  data.frame(
    index = seq(1, stop_index - start_index + 1),
    steps = seq(1, length(cases))[start_index:stop_index],
```

```
      dates = dates[start_index:stop_index],
      cases = cases[start_index:stop_index],
      raw_S = susceptible[start_index:stop_index],
      raw_I = infectious[ start_index:stop_index],
      raw_R = recovered[  start_index:stop_index]
    )
  )

}

# Plot SIR Infectious data
plot_SIR_I_data <- function(location,
                            infectious_period = default_infectious_period,
                            delay_days = 0,
                            lag_days = 0,
                            start_date = NULL,
                            stop_date  = NULL) {

  raw_data <- get_SIR_data(location = location,
                           infectious_period = infectious_period,
                           delay_days = delay_days,
                           lag_days = lag_days,
                           start_date = start_date,
                           stop_date  = stop_date)

  raw_data %>%
    ggplot(aes(x = dates, y = raw_I)) +
    geom_line() +
    theme_light() +
    labs(
      title = "Infectious Data v. Date",
      subtitle = bquote(.(location)),
      x = "Date",
      y = "Number of Infectious People"
    ) +
    theme(
      plot.title    = element_text(hjust =  0.5),
      plot.subtitle = element_text(hjust =  0.5),
      axis.title.y  = element_text(vjust =  3.0),
      axis.title.x  = element_text(vjust = -1.0)
    )

}

########################################
# Functions for Computing Model Delay #
########################################

# Get the date that X no. infectious individuals are "observed" for a locale
# infectious_period is an argument to get_SIR_data() hence it can be passed here
date_of_X_infectious <- function(location,
                                 X = 300,
                                 infectious_period = default_infectious_period
```

```
                                ) {

  sir_data <- get_SIR_data(location = location,
                           infectious_period = infectious_period)

  # Stop if x > max no. infectious
  if (X > max(sir_data$raw_I)) {

    stop("Error: date_of_X_infectious() -
         X value is above the max observed for ", paste(location))

  }

  # Get date of X no. infectious ("I" of SIR model)
  date <- sir_data %>%
    filter(raw_I >= X) %>%
    arrange(dates) %>%
    head(1) %>%
    pull(dates)


  return(date)

}
```

SIR Model Fitting

The `ode` function in the `deSolve` package is used to solve the system of differential equations defined by the SIR model. The infectious compartment of the SIR model is of paramount interest. Therefore, the loss function for evaluating the model fit of the raw data is applied solely to the infectious compartment. The residual sum of squares (RSS) is chosen as the loss function. A contour plot can be used to visualize the RSS values for a range of beta and gamma values. The `optim` function of the `stats` package is used to optimize the SIR fit to the raw infectious data. **R** codes for fitting and optimization of the SIR model are presented below.

```
##############################
# Define SIR model for ode() #
##############################

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) ) )

  })

}
```

```r
#######################################################
# Define SIR fit function by composition with ode() #
#######################################################

# Input parameters:

# params <vector of doubles> - beta and gamma (in that order)
# Note: this function takes parameters beta and gamma as a single vector for
# compatibility with optim().
# Gamma can be estimated based on the average duration of the infectious period
# as 1/(infectious duration)
# 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 other than perhaps by
# comparison to other pandemics,
# ie: deduction from the ave. R_0 of ~ 2 for the 1918 Spanish Flu.
# Recall: R_0 = beta / gamma

# location  <string> - "city/town, state" format, ie: "New York City, New York".

# There are two ways to specify a region over which to apply fitting.
# 1st is to specify a delay time
# delay_days  <int> - skip this number of days since the 1st cases are seen.
# 2nd is to specify a date-range in ymd format, which overrides delay_days.

fit_SIR_model <- function(params,
                          location = test_locale,
                          delay_days = 0,
                          infectious_period = default_infectious_period,
                          start_date = NULL,
                          stop_date  = NULL) {

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

  N <- get_pop(location)

  # 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_SIR_data(location = location,
                           delay_days = delay_days,
                           infectious_period = infectious_period,
                           start_date = start_date,
                           stop_date  = stop_date)
```

```
  # Get initial values for SIR model
  initial_values <- c(
    S = raw_data$raw_S[1],  # number of susceptible people
    I = raw_data$raw_I[1],  # no. infectious
    R = raw_data$raw_R[1]   # no. recovered (and immune), or deceased
  )


  # Evaluate SIR Model
  days <- 1:length(raw_data$raw_I)

  predictions <- ode(
    y     = initial_values,
    times = days,
    func  = SIR,
    parms = parameter_values
  )


  predictions <- data.frame(predictions) %>%
    rename(
      index = time,
      fit_S = S,
      fit_I = I,
      fit_R = R
    )


  return(predictions)

}

##############################################################################
# Evaluate the Fit using the SIR Model based on the (RSS) Loss Function #
##############################################################################

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

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

}

# Function for composition with optim()
get_SIR_RSS <- function(params,
                        location = test_locale,
                        delay_days = 0,
                        infectious_period = default_infectious_period,
                        start_date = NULL,
                        stop_date = NULL) {

  # Extract raw data
  raw_data <- get_SIR_data(location = location,
```

```
                            delay_days = delay_days,
                            infectious_period = infectious_period,
                            start_date = start_date,
                            stop_date = stop_date)

  # Compute fitted data
  fit_data <- fit_SIR_model(params = params,
                            location = location,
                            delay_days = delay_days,
                            infectious_period = infectious_period,
                            start_date = start_date,
                            stop_date = stop_date)

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

}

############################################################
# Evaluate the SIR Model based on the (RSS) Loss Function #
############################################################

# beta_min and beta_max set the lower and upper boundaries of beta to scan
# gamma_min and gamma_max set the lower and upper boundaries of gamma to scan
# n_betas and n_gammas are the number of values of beta and gamma to scan
# (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 generally is not large enough.

# Set default scan values
beta_min  = 0.1
beta_max  = 1.0
n_betas   = 25
gamma_min = 0.1
gamma_max = 1.0
n_gammas  = 25
betas     = seq(beta_min , beta_max , len = n_betas)
gammas    = seq(gamma_min, gamma_max, len = n_gammas)

scan_SIR_RSS <- function(beta_seq  = betas,
                         gamma_seq = gammas,
                         location  = test_locale,
                         delay_days = 0,
                         infectious_period = default_infectious_period,
                         start_date = NULL,
                         stop_date = NULL) {

  # Parallelization option
  require("furrr")

  # Create a set of betas and gammas to test
  # generate all combinations of beta and gamma values
```

```
  # Object is 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_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 for all combinations of beta and gamma test values
  results_rss_scan <- parameter_set %>%
    mutate(

      #values = map(        # Normal version
      values = future_map(  # Parallel version
        parameters,
        get_SIR_RSS,
        location = location,
        delay_days = delay_days,
        infectious_period = infectious_period,
        start_date = start_date,
        stop_date  = stop_date
      )
    )


  return(results_rss_scan)

}

# Heat map with contour lines of SIR model fit RSS for beta and gamma pairs
SIR_RSS_contour_plot <- function(location  = test_locale,
                           beta_seq  = betas,
                           gamma_seq = gammas,
                           rss_values_matrix) {

  require(grDevices)

  lvls <- seq(min(rss_values_matrix), max(rss_values_matrix), le = 25)

  filled.contour(
    x = beta_seq,
    y = gamma_seq,
    z = rss_values_matrix,
    xlim = c(min(beta_seq), max(beta_seq)),
```

```
    ylim = c(min(gamma_seq), max(gamma_seq)),
    zlim = c(min(rss_values_matrix), max(rss_values_matrix)),
    col  = hcl.colors(
      n = 100,
      palette = "Viridis",
      alpha = 0.5,
      rev = FALSE,
      fixup = TRUE
    ),
    levels = lvls,
    main = bquote(.(location)),
    ylab = expression(paste(gamma)),
    xlab = "",
    xaxs = "i", yaxs = "i",
    key.title = title(
      main = bquote("RSS("~beta~","~gamma~")"), cex.main = 0.85),
    plot.axes = {
      contour(
        x = beta_seq,
        y = gamma_seq,
        z = rss_values_matrix,
        add  = TRUE,
        axes = FALSE,
        drawlabels = FALSE
      );
      axis(1); axis(2) # must add back axes explicitly
    }

  )
  # Adjust x-axis label only: shift up from subtitle
  mtext(
    text = expression(paste(beta)),
    side = 1,
    line = 2.5
  )

}

#################
# Optimization #
#################

# Wrapper for optim() wrapper that uses the SIR model
optim_SIR <- function(init_params,
                      location = test_locale,
                      delay_days = 0,
                      start_date = NULL,
                      stop_date = NULL) {

  model <- optim(
    par = init_params,          # initial values
    location = location,        # optional parameter to get_SIR_fit_RSS()
    delay_days = delay_days,    # optional parameter to get_SIR_fit_RSS()
```

```
    start_date = start_date,    # optional parameter to get_SIR_fit_RSS()
    stop_date  = stop_date,     # optional parameter to get_SIR_fit_RSS()
    fn  = get_SIR_RSS,          # Function to be minimized: RSS of SIR fit
    method = "L-BFGS-B" ,       # Gradient projection method using BFGS matrix
    lower  = c(0.1, 0.1),       # lower boundary for beta and gamma
    upper  = c(1, 1)            # upper boundary for beta and gamma
  )


  return(
    tibble(
      "location" = location,
      "delay_days" = delay_days,
      "start_date" = ymd(start_date),
      "stop_date" = ymd(stop_date),
      "beta"  = model$par[1],
      "gamma" = model$par[2],
      "R0"  = model$par[1]/model$par[2],
      "RSS" = model$value,
      "convergence" = model$convergence,
      "message" = model$message
    )
  )

}

######################################################################
# Plot SIR Fit of Infectious Compartment and Raw Infectious Data #
######################################################################

# Tidy infectious data for raw data and SIR fit data
tidy_SIR_I <- function(raw, fit, join_column = "index") {

  raw_and_fit <- inner_join(raw, fit, by = {{join_column}})

  tidy_data <- raw_and_fit %>%
    select(dates, fit_I, raw_I) %>%
    rename(

      raw = raw_I,
      fit = fit_I

    ) %>%
    pivot_longer(

      cols      = -c("dates"),
      names_to  = "code",
      values_to = "infectious"

    )


  return(tidy_data)
```

```
}


#  Plot infectious data for raw data and SIR fit data
plot_SIR_fit <- function(location,
                         data, x, y, key,
                         delay_days = NULL,
                         start_date = NULL,
                         beta, gamma) {

  # If start_date is set and is not NULL
  if (!missing(start_date) & !is.null(start_date)) {

    # Check date validity
    indices <- get_date_indices(location = location, start_date = start_date)

    delay <- indices["start_index"]

  }

  # If start_date is not given set start_index to 1
  else if (missing(start_date) | is.null(start_date)) {

    delay <- 1

  }


  R0 <- beta/gamma

  ggplot(data, aes({{x}}, {{y}}, color = {{key}})) +
    geom_line() +
    theme_light() +
    labs(
      title = "Infectious (SIR model) v. Date",
      subtitle = bquote(.(location) ~ ":"
                        ~beta  == .(round(beta,  2))~","
                        ~gamma == .(round(gamma, 2))~","
                        ~ R[0] == .(round(R0, 2)) ),
      x = "Date",
      y = "Infectious",
      color = "Key:"
    ) +
    scale_color_manual(values=c("#0072B2", "#D55E00")) +
    theme(
      plot.title    = element_text(hjust =  0.5),
      plot.subtitle = element_text(hjust =  0.5),
      axis.title.y  = element_text(vjust =  3.0),
      axis.title.x  = element_text(vjust = -1.0),
      plot.caption  = element_text(vjust = -2.0, hjust = 0.5)
    )

}
```

SIR Analysis - Pt. 1: Inference using K-means Clustering

The aim is to infer the outcome for a given locale based on another. Comparison between locales is motivated by K-means clustering based on population and grace period. Locales within the same cluster are deemed similar. In general multiple locales can be used to infer the result of others. The locales that experiences COVID-19 cases earlier supply the training data set while those that do later furnish the testing data. An average of the parameters obtained from the training data can then be used to infer those of locales in the testing set. Here, for simplicity, only two locales are chosen and one is used to supply the training data while the other supplies the testing set.

Case Study of King, Washington (*the training data set*)

A case study is first performed using the training data to optimize application of the SIR model. K-means clustering suggests King, Washington and Queens, New York might experience the pandemic similarly. King, Washington was the first locale to report COVID-19 cases on 2020-01-22, and has multiple waves of infection (Fig. 10). The first wave of infections is considered to try and gain insights into COVID-19 dynamics and parameterization of the SIR model in the absence of a public health response, which might perturb the model. The peak of the first wave occurs around the 14^{th} of April.

```
case_study_locale <- "King, Washington"

# Date 1st cases are recorded
case_study_start_date <- get_init_cases_date(case_study_locale)

# Plot of infectious-SIR data
plot_SIR_I_data(case_study_locale)
```

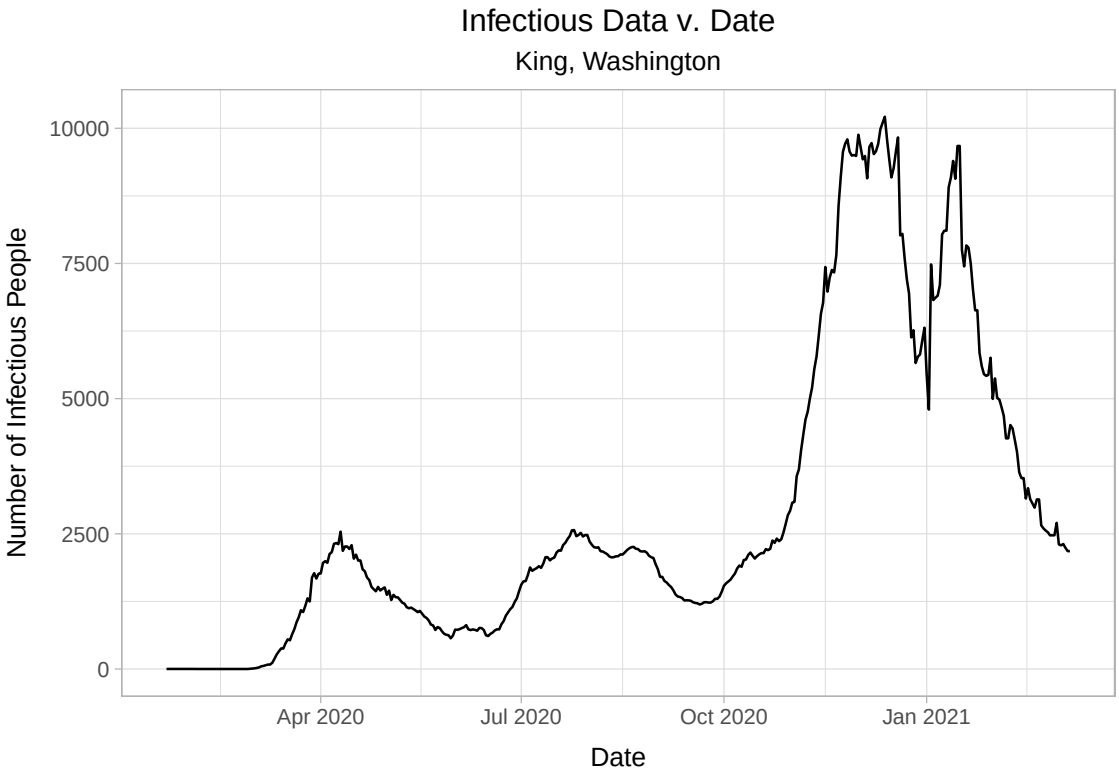


Figure 10: Plot of the infectious population for King, Washington.

```
# Obs. Peak of 1st wave for "King, WA" is around 2 weeks into April 2020.
case_study_stop_date <- 20200414
```

Initial Value Determination for SIR Compartments

The initial value for the infectious population determines those of the remaining SIR compartments and establishes the modeling start date. The plot of the fit RSS against the initial number of infectious individuals shows the SIR model does not perform well when the initial number of infectious individuals is less than 200 (Fig. 11). The threshold number of infectious people before the SIR model can be applied is chosen as 300. Improvement of the SIR model's performance begins to taper beyond this value. Further improvement can be sought via greater values for the initial number of infectious individuals but at the expense of delaying model application (Fig. 12).

```
# Purpose: study the influence of the initial number of infectious people on the
# fitted model.  The initial no. infectious people is used to determine the
# delay time prior to model fitting.
# A higher number obviously puts a limit on how soon one can begin obtaining
# predictions using the SIR model.  But the SIR model predicts a significantly
# delayed outbreak compared to what is typically observed for a
# small initial no. infectious people.

# Instantiate table for collating case study RSS data
case_study_rss_data <- tibble("I(init)" = integer(), "RSS" = double())


# Get SIR fit (and RSS) for specific value of the initial no. infectious people.
# Specifying the initial no. infectious people determines modeling start date.
fit_SIR_I_init <- function(I_init = 0,
                           location  = case_study_locale,
                           stop_date = case_study_stop_date) {

  # 0 is a natural initial value for generating sequences but is meaningless
  # in terms of an initial value for the no. infectious people.
  # Protocol: if 0 is passed convert it to the minimum I_0 value = 1.
  if (I_init == 0) {I_init <- 1}

  start_date <- date_of_X_infectious(location = location,
                                     X = I_init)

  scan <- scan_SIR_RSS(location = location,
                       start_date = start_date,
                       stop_date  = stop_date)

  fit <- fit_SIR_model(params = unlist(scan$parameters[which.min(scan$values)]),
                       location = location,
                       start_date = start_date,
                       stop_date  = stop_date)

  raw <- get_SIR_data(location = location,
                      start_date = start_date,
                      stop_date  = stop_date)
```

```
  rss <- RSS(prediction = fit$fit_I, raw_data = raw$raw_I)


  # Collate I_0 and RSS to global variable
  case_study_rss_data <<- bind_rows(case_study_rss_data,
                                    tibble("I(init)" = I_init,
                                           "RSS" = rss))

  # Label fit-column according to the I_init value used in fitting.
  # This facilitates tidy-formatting and plotting components with color=key.
  df <- tibble(
    dates = raw$dates,
    fit = fit$fit_I
  ) %>% rename(!!paste(I_init) := fit)


  return(df)

}


# Build starting data frame to join to based on the raw SIR data
case_study_SIR_raw <- get_SIR_data(location   = case_study_locale,
                                   start_date = case_study_start_date,
                                   stop_date  = case_study_stop_date)
case_study_fits <- case_study_SIR_raw[c("dates", "raw_I")]

# Define sequence of initial SIR-I values
I0_seq <- seq(0, 500, 100)

# Loop over initial SIR-I values
#time_init <- Sys.time()  # Start timer

for (i in I0_seq) {

  # Join (v. row/column binding)
  case_study_fits <- full_join(x = case_study_fits,
                               y = fit_SIR_I_init(i),
                               by = "dates",
                               keep = FALSE)
}

#time_elapsed <- Sys.time() - time_init
#time_elapsed                             # Run time


# Plot RSS v. I_0
# Zoom in by omitting the values for I(init)=1
case_study_rss_data[-1,] %>%
  ggplot(aes(x = `I(init)`, y = RSS)) +
  geom_line() +
  theme_light() +
  labs(
```

```
    title = "SIR Fit RSS v. Starting Number of Infectious Individuals",
    subtitle = "",
    caption = "",
    x = "Initial Number of Infectious Individuals"
  ) +
  theme(
    plot.title = element_text(hjust =  0.5)
  )
```

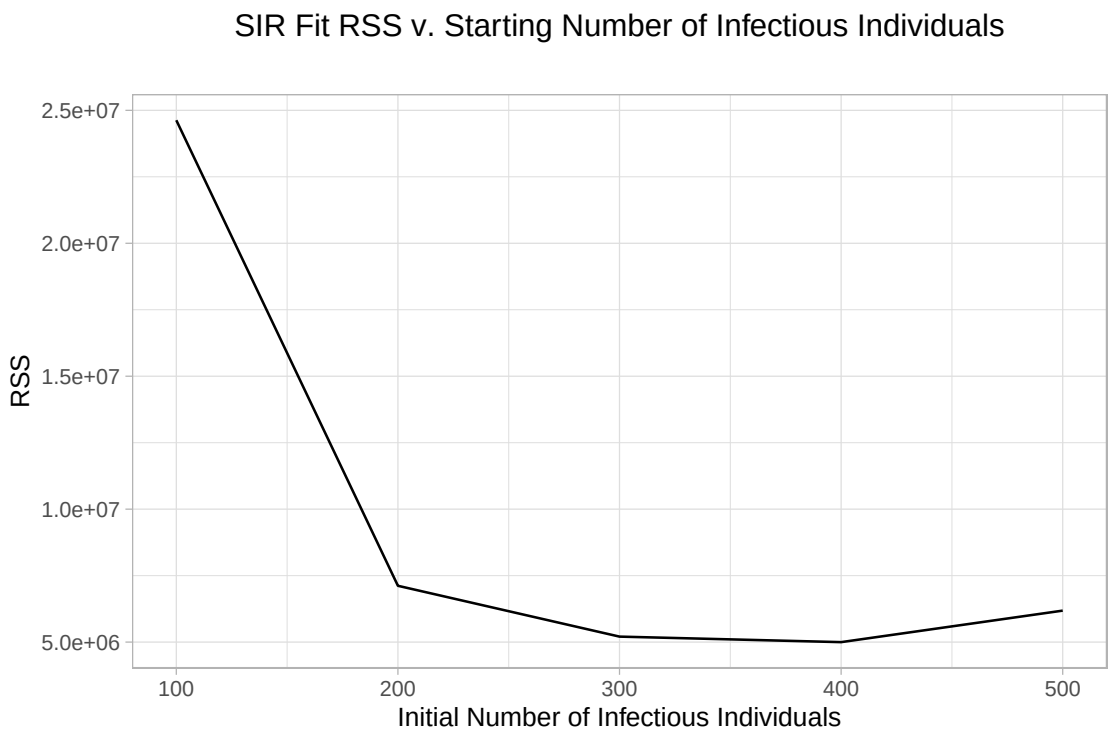


Figure 11: Plot of RSS for the SIR model fit using increasing values for the initial number of infectious individuals.

```
# Put raw and model data for multiple fits in tidy format
# RECALL: the sequence starts from 0 for looping purposes, but is modified to 1
# for modeling purposes (I_0 cannot be zero)
# Modify vector in-place (element-wise)
I0_seq[1] <- 1

# Subset date, raw, and fit columns
case_study_fits <-
  case_study_fits[c("dates",
                    "raw_I",
                    as.character(I0_seq))] %>%
  rename(raw = raw_I)

case_study_tidy <- case_study_fits %>% pivot_longer(
  cols      = -c("dates", "raw"),
  names_to  = "code",
```

```
  values_to = "infectious")


# Plot successive fitted curves to show that fit improves as the initial value
# Specify factor levels: ensure numeric ordering of factor levels in plot legend
# Uses viridis color palette
case_study_tidy$code <- factor(case_study_tidy$code, levels = I0_seq)

case_study_tidy %>%
  ggplot(aes(x = dates, y = raw)) +
  geom_line() +
  geom_line(aes(x = dates, y = infectious, group = code, color = code)) +
  scale_color_viridis(discrete = TRUE, option = "D", direction = -1) +
  theme_light() +
  labs(
    title = "Influence of Initial Infectious SIR Value on Model Fit",
    subtitle = "Locale: King, Washington",
    caption = "*Raw infectious SIR data is shown by the black line.",
    x = "Date",
    y = "Population",
    color = "I(init)"
  ) +
  theme(
    plot.title    = element_text(hjust =  0.5),
    plot.subtitle = element_text(hjust =  0.5),
    axis.title.y  = element_text(vjust =  3.0),
    axis.title.x  = element_text(vjust = -1.0),
    plot.caption  = element_text(vjust = -2.0, hjust = 0.50)
  )
```

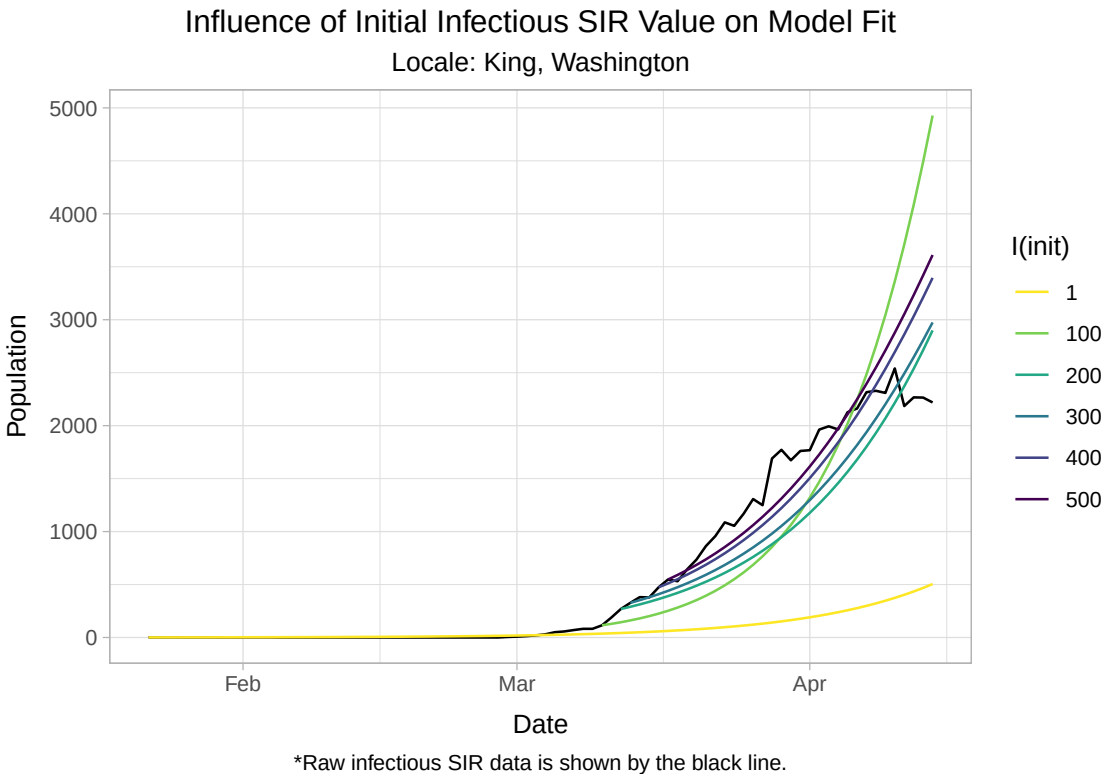


Figure 12: Plot of the SIR fits for the infectious population using an increasing value for the initial number of infectious individuals.

Initial Value Determination for SIR Parameters

In addition to initial values for the compartments the SIR model requires supplying parameters beta (β) and gamma (γ). A contour plot of the SIR fit RSS for a scan of β and γ values shows there is a large region of parameter space that yields comparable RSS values (Fig.13). A fine detailed scan of the parameter space is an inefficient approach to finding these parameters. A coarse scan is used instead to supply starting SIR parameters for optimization using the L-BFGS-B algorithm (suitable for multiple parameters) in the `optim` function from the `stats` package. The optimized SIR parameters for the first wave in King, Washington and the fit are shown in figure 14.

```
train_case_locale <- "King, Washington"

case_study_init_I <- 300


# Begin modeling when the no. infectious people reaches: 250
train_case_start_date <- date_of_X_infectious(location = train_case_locale,
                                              X = case_study_init_I)

# Consider 1st phase of pandemic
# Once 300 cases have been observed
# and until 2020 April 14 when the 1st peak occurs.
train_case_stop_date <- 20200414

# Get raw SIR data
```

```
train_case_SIR_raw <- get_SIR_data(location = train_case_locale,
                                   start_date = train_case_start_date,
                                   stop_date  = train_case_stop_date)

# Scan SIR fit parameters
train_case_SIR_scan <- scan_SIR_RSS(location = train_case_locale,
                                    start_date = train_case_start_date,
                                    stop_date  = train_case_stop_date)

# Contour plot of SIR fit RSS
train_case_rss_scan_matrix <- matrix(unlist(train_case_SIR_scan$values),
                                     sqrt(length(train_case_SIR_scan$values)))

SIR_RSS_contour_plot(location = train_case_locale,
                     rss_values_matrix = train_case_rss_scan_matrix)
```

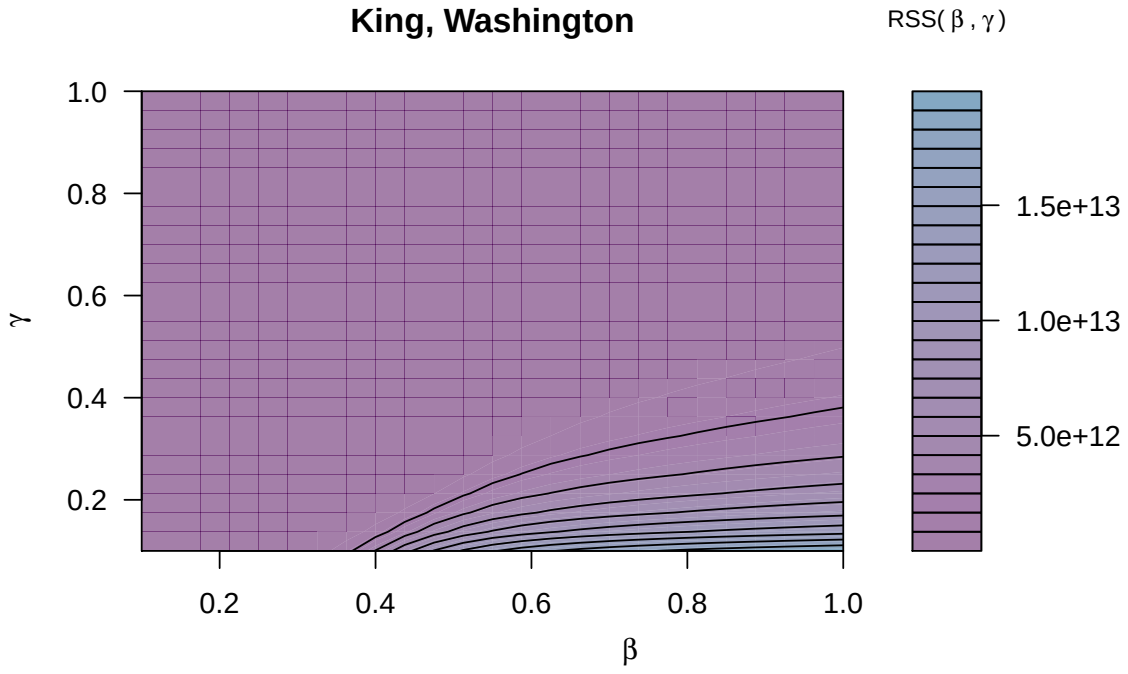


Figure 13: Heat map and contour plot of the SIR fit RSS values for a scan of beta and gamma values for the first wave of infections in King, Washington.

```
# Purpose:
# Determine if knowledge about "King, WA" could potentially have
# provided some insight into the outbreak in "Queens, NY" by using the
# parameters obtained for "King, WA" on "Queens, NY".
# "King, WA" and "Queens, NY" are in the same K-means cluster when k = 4.
# They have similar population size but "King, WA" began recording cases over
# a month before "Queens, NY" and they are on opposite sides of the country.
```

```r
# Get parameters that minimize the SIR fit based on RSS scan
train_case_starting_optim_params <-
  unlist(train_case_SIR_scan$parameters[which.min(train_case_SIR_scan$values)])


# Run optimization
train_case_SIR_optim <-
  optim_SIR(init_params = train_case_starting_optim_params,
  location   = train_case_locale,
  start_date = train_case_start_date,
  stop_date  = train_case_stop_date)

# Apply optimized model
train_case_SIR_ofit <- fit_SIR_model(
  params     = c(train_case_SIR_optim$beta, train_case_SIR_optim$gamma),
  location   = train_case_locale,
  start_date = train_case_start_date,
  stop_date  = train_case_stop_date)

# Put raw and fit data in tidy format
train_case_SIR_tidy_ofit <- tidy_SIR_I(raw = train_case_SIR_raw,
                                       fit = train_case_SIR_ofit, "index")

# Plot
plot_SIR_fit(location = train_case_locale,
             data = train_case_SIR_tidy_ofit,
             x = dates,
             y = infectious,
             key = code,
             start_date = train_case_start_date,
             beta  = train_case_SIR_optim$beta,
             gamma = train_case_SIR_optim$gamma)
```

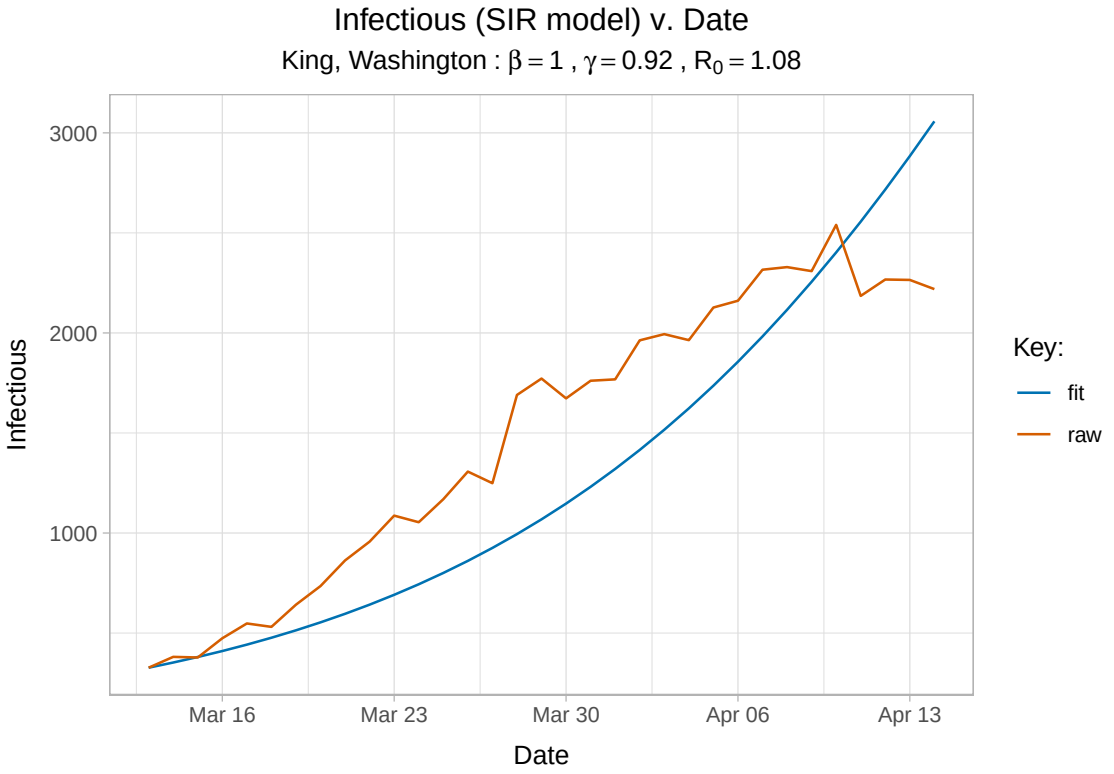


Figure 14: Plot of the optimized SIR fit and the raw data for the first wave of COVID-19 infections in King, Washington.

Queens, New York (*the testing data set*)

The SIR modeling protocol developed using King, Washington as a case study can be adopted to examine other locales. The first wave of cases in King, Washington is also used as the training data for developing a model that can then be tested elsewhere. The SIR parameters obtained from the first wave in King, Washington are applied to the first wave in Queens, New York, which began over a month later on 2020-03-06. The SIR parameters for King, Washington massively underestimate the magnitude of the pandemic in Queens, New York. (Fig.15).

```
# Purpose: use optimized parameters for King, WA to model data for Queens, NY.

# Consider 1st phase of the pandemic
# Once 300 cases have been observed
# and until 2020 April 14 when the 1st peak occurs in "King, WA"

# Question: given what was known about King, WA, could one predict what followed
# in Queens, NY when the first 300 cases were observed.
# Ans.: No, King, WA model underestimates the Queens, NY outbreak.
# Results: K-means using population and grace period is not sufficient for
# for comparison between locales on opposite sides of the country.

test_case_locale <- "Queens, New York"


# Begin modeling when there are 300 infectious people
```

```r
test_case_start_date <- date_of_X_infectious(location = test_case_locale,
                                             X = case_study_init_I)

# Get raw SIR data
test_case_SIR_raw <- get_SIR_data(location   = test_case_locale,
                                  start_date = test_case_start_date,
                                  stop_date  = train_case_stop_date)

# Apply "King, Washington" parameters to model "Queens, New York"
test_case_SIR_fit <- fit_SIR_model(
  params = c(train_case_SIR_optim$beta, train_case_SIR_optim$gamma),
  location = train_case_locale,
  start_date = test_case_start_date,
  stop_date  = train_case_stop_date)

# Put raw and fit data in tidy format
test_case_SIR_tidy_fit <- tidy_SIR_I(raw = test_case_SIR_raw,
                                     fit = test_case_SIR_fit,
                                     join_column = "index")

# Plot
plot_SIR_fit(location = test_case_locale,
             data = test_case_SIR_tidy_fit,
             x = dates,
             y = infectious,
             key = code,
             start_date = test_case_start_date,
             beta  = train_case_SIR_optim$beta,
             gamma = train_case_SIR_optim$gamma)
```

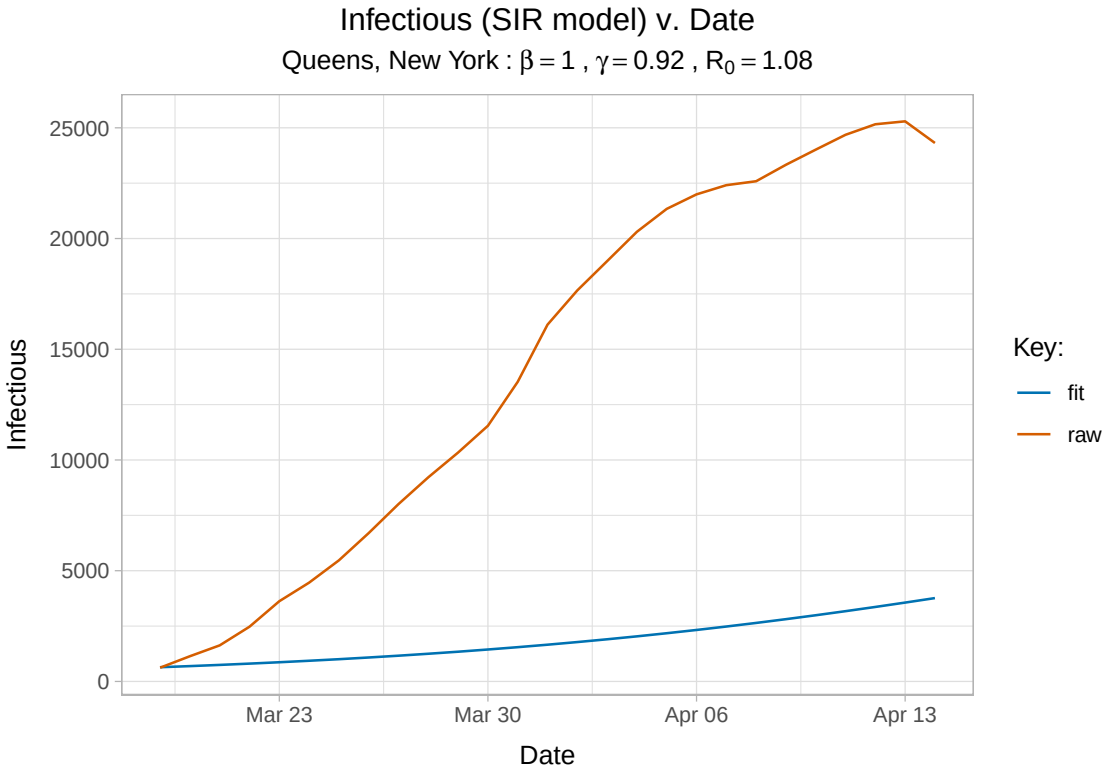


Figure 15: Plot of the SIR fit using parameters obtained from King, Washington, to model the first wave of COVID-19 infections in Queens, New York.

SIR Analysis - Pt. 2: Forecasting

It is difficult to make comparisons between locales such that accurate predictions can be made for one based on another. Moreover, once the pandemic reaches a locale it is important to use endogenous data for making predictions, as well. The SIR model data can be extended into the future using the parameters derived from fitting. Available data provides the training data, while the testing data becomes accessible in the future. A weekly forecast is proposed based on the previous two weeks. Confidence intervals from model fitting are used to estimate the forecast error. A definite period is chosen for reproducibility of the results shown.

Cook County, Illinois

Illinois is located in the Midwestern U.S. and is the second state to record COVID-19 cases, on 2020-01-24, only two days after the first recorded cases. Cook, Illinois is consistently associated with clusters at the upper end of the population spectrum (Fig.9). The heat map for time series (Fig.7) shows that COVID-19 cases are on the rise in Illinois during the month leading into the 2020 presidential election. Given the circumstances this is an important period during which to provide forecasting for Cook, Illinois.

```
# Note: the middle of October suggested the pandemic was not over
start_date_spec = 20201007
stop_date_spec  = 20201021

# Set locale variable
test_locale <- "Cook, Illinois"
```

```r
# Get raw SIR data for Cook, IL
Cook_IL_SIR_raw <- get_SIR_data(location = test_locale,
                                start_date = start_date_spec,
                                 stop_date = stop_date_spec)

# Scan SIR parameters
Cook_IL_SIR_scan <- scan_SIR_RSS(location   = test_locale,
                                 start_date = start_date_spec,
                                 stop_date  = stop_date_spec)

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

# Run optimization using date range.
Cook_IL_SIR_optim <- optim_SIR(init_params = starting_optimization_parameters,
                               location    = test_locale,
                               start_date  = start_date_spec,
                               stop_date   =  stop_date_spec)

# Fit optimized model
Cook_IL_SIR_ofit <- fit_SIR_model(
  c(Cook_IL_SIR_optim$beta, Cook_IL_SIR_optim$gamma),
  Cook_IL_SIR_optim$location,
  Cook_IL_SIR_optim$lag,
  start_date = start_date_spec,
  stop_date  = stop_date_spec,
)

# Put data in tidy format
Cook_IL_SIR_tidy_ofit <- tidy_SIR_I(raw = Cook_IL_SIR_raw,
                                    fit = Cook_IL_SIR_ofit,
                                    join_column = "index")
```

Confidence Intervals

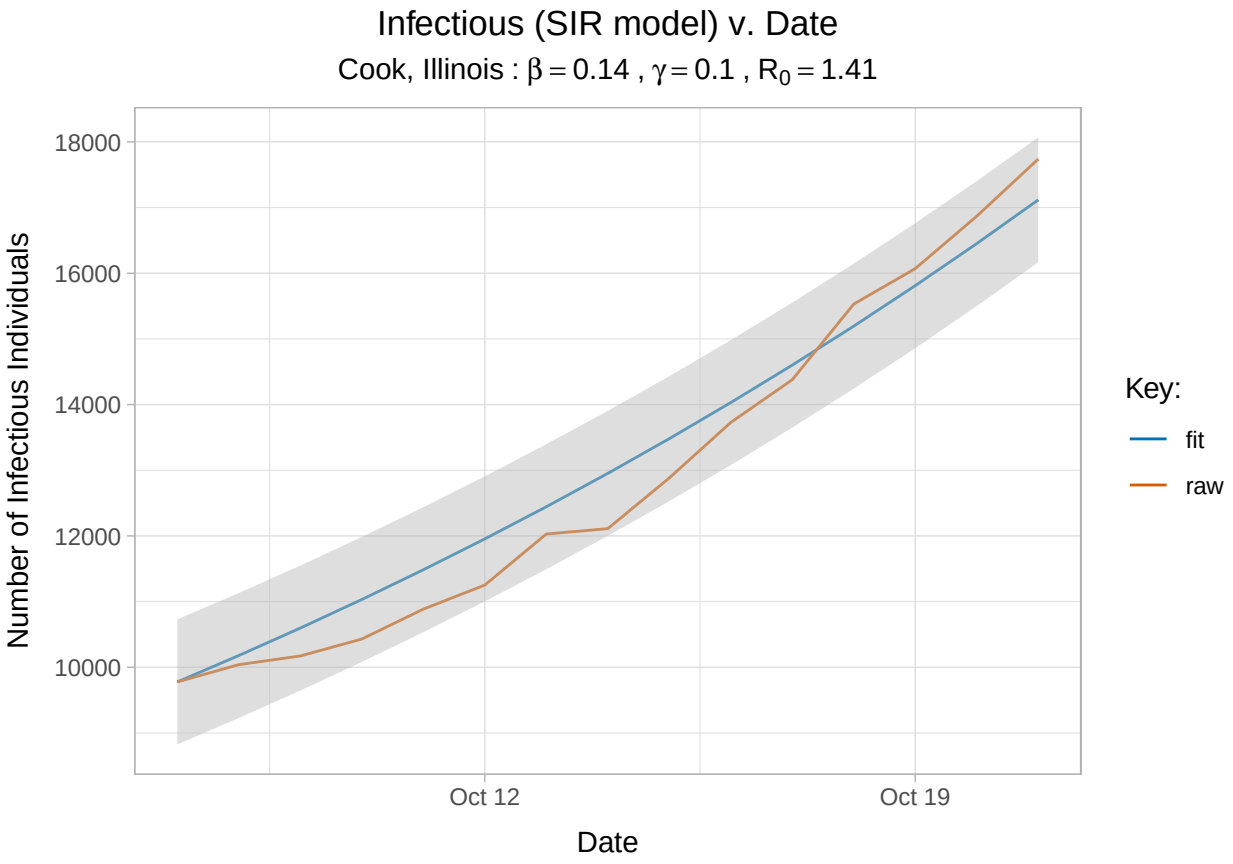
Confidence intervals for the SIR fit are calculated using maximum likelihood estimation (MLE) by minimizing the negative log-likelihood, assuming a Gaussian distribution of the errors. **R** codes for computing confidence intervals based on MLE are presented below.

```r
# Define likelihood function
mLL <- function(sigma = 1, observations, predictions) {

  # Minus log-likelihood
  return(
    -sum(
      dnorm(                # Normal density
        x = observations,
        mean = predictions,
        sd = sigma,
        log = TRUE          # Probability (p) given as log(p)
      )
```

```
    )
  )

}

# Use MLE to obtain sigma for the distribution of residuals
mle_mll <- function(raw, fit) {

  estimates <- mle2(
    minuslogl = mLL,
    start = list(sigma = 1),  # parameters to minuslogl function
    method = "L-BFGS-B",
    data = list(observations = raw,
                predictions  = fit)
  )


  return(estimates)

}

Cook_IL_SIR_ofit_mle <- mle_mll(raw = Cook_IL_SIR_raw$raw_I,
                                fit = Cook_IL_SIR_ofit$fit_I)

# Parameter estimates
Cook_IL_SIR_ofit_mle_sigma_hat <- coef(Cook_IL_SIR_ofit_mle)["sigma"]

# Get confidence intervals for the fitted curve
get_SIR_CIs <- function(data, sigma) {

  # confidence level
  cl <- 0.95
  cl <- (1 - cl)/2

  low_CI <- qnorm(p = cl,     mean = data, sd = sigma)
  upp_CI <- qnorm(p = 1 - cl, mean = data, sd = sigma)


  return(data.frame(lower = low_CI, upper = upp_CI))

}

Cook_IL_SIR_ofit_CIs <- get_SIR_CIs(data  = Cook_IL_SIR_ofit$fit_I,
                                    sigma = Cook_IL_SIR_ofit_mle_sigma_hat)


# Convert wide data to tidy data
tidy_SIR_I_dataCI <- function(raw, x1, fit, x2, join_col, lower, upper) {

  raw_and_fit <- inner_join(raw, fit, by = {{join_col}})

  raw_and_fit <- cbind(raw_and_fit, lower, upper)
```

```
  fig_data <- raw_and_fit %>%
    select(dates, {{x1}}, {{x2}}, lower, upper) %>%
    rename(
      raw = {{x1}},
      fit = {{x2}},
      lower_CI = lower,
      upper_CI = upper
    ) %>%
    pivot_longer(              # Only tidy what needs to be color-coded
      cols = -c(
        "dates",
        "lower_CI",
        "upper_CI"
      ),
      names_to  = "code",
      values_to = "infectious"
    )

}

Cook_IL_SIR_tidyCI <- tidy_SIR_I_dataCI(raw = Cook_IL_SIR_raw,  x1 = raw_I,
                                        fit = Cook_IL_SIR_ofit, x2 = fit_I,
                                        join_col = "index",
                                        lower = Cook_IL_SIR_ofit_CIs$lower,
                                        upper = Cook_IL_SIR_ofit_CIs$upper)


# Plot data + fit + CIs
plot_SIR_fitCI <- function(location,
                           data, x, y, code,
                           lower, upper,
                           beta, gamma) {

  R0 <- beta/gamma

  ggplot(data, aes({{x}}, {{y}}, color = {{code}})) +
    geom_line() +
    geom_ribbon(
      aes(ymin = {{lower}}, ymax = {{upper}}, fill = "band"),
      fill = "grey",
      alpha = 0.50,
      color = "NA"
    ) +
    theme_light() +
    labs(
      title = "Infectious (SIR model) v. Date",
      subtitle = bquote(.(location) ~ ":"
                        ~beta  == .(round(beta, 2)) ~ ","
                        ~gamma == .(round(gamma, 2)) ~ ","
                        ~R[0]  == .(round(R0, 2))),
      x = "Date",
      y = "Number of Infectious Individuals",
      color = "Key:"
```

```
    ) +
    scale_color_manual(values = c("#0072B2", "#D55E00")) +
    theme(
      plot.title    = element_text(hjust =  0.5),
      plot.subtitle = element_text(hjust =  0.5),
      axis.title.y  = element_text(vjust =  3.0),
      axis.title.x  = element_text(vjust = -1.0),
      plot.caption  = element_text(vjust = -2.0, hjust = 0.5)
    )

}

plot_SIR_fitCI(location = test_locale,
               data  = Cook_IL_SIR_tidyCI,
               x = dates,
               y = infectious,
               code  = code,
               lower = lower_CI,
               upper = upper_CI,
               beta  = Cook_IL_SIR_optim$beta,
               gamma = Cook_IL_SIR_optim$gamma)
```



Weekly Forecast

R codes for weekly forecasting using the SIR model are presented below.

```r
# Compute forecast
forecast_SIR <- function(start_date,
                         day_forecast = 7,
                         N, beta, gamma,
                         S, I, R) {

  # Instantiate dataframe
  forecast = tibble("steps" = 0,
                    "dates" = start_date,
                    "S" = S, "I" = I, "R" = R)

  for (steps in 1:day_forecast) {

    # Increment compartments
    dS  <- - beta * S * I / N
    gI  <-  gamma * I
    S_t <- S + dS
    I_t <- I - dS - gI
    R_t <- R + gI

    # Update values
    S <- S_t
    I <- I_t
    R <- R_t

    dates <- ymd(start_date) + days(steps)
    forecast <- bind_rows(forecast,
                          tibble(steps,
                                 dates,
                                 S, I, R))
  }


  return(forecast)

}

Cook_IL_SIR_forecast <- forecast_SIR(
  start_date   = tail(Cook_IL_SIR_raw$date, n = 1),
  day_forecast = 7,
  N = get_pop("Cook, Illinois"),
  beta  = Cook_IL_SIR_optim$beta,
  gamma = Cook_IL_SIR_optim$gamma,
  S = tail(Cook_IL_SIR_raw$raw_S, n = 1),
  I = tail(Cook_IL_SIR_raw$raw_I, n = 1),
  R = tail(Cook_IL_SIR_raw$raw_R, n = 1)
)

# Get confidence intervals for forecast
Cook_IL_SIR_forecast_CIs <- get_SIR_CIs(data  = Cook_IL_SIR_forecast$I,
                                        sigma = Cook_IL_SIR_ofit_mle_sigma_hat)

# Put raw infectious data and forecast in tidy format separately and row_bind
```

```r
Cook_IL_SIR_forecast_tidy_CIs <- bind_rows(
  Cook_IL_SIR_raw %>%
    rename(raw = raw_I) %>%
    select(dates, raw),

    cbind(Cook_IL_SIR_forecast, Cook_IL_SIR_forecast_CIs) %>%
    select(dates, I, lower, upper)
  ) %>%
  rename(forecast = I) %>%
  pivot_longer(
    cols = -c(dates, lower, upper),
    names_to  = "code",
    values_to = "infectious"
  )


# Plot function for raw + forecast + confidence intervals
plot_SIR_forecastCI <- function(location,
                                data, x, y, code,
                                lower, upper,
                                beta, gamma) {

  R0 <- beta/gamma

  ggplot(data, aes({{x}}, {{y}}, color = {{code}})) +
    geom_line() +
    geom_ribbon(
      aes(ymin = {{lower}}, ymax = {{upper}}, fill = "95% C.I.s"),
      alpha = 0.50,
      color = "NA"
    ) +
    scale_color_manual(values = c("#000000", "#D55E00")) +
    scale_fill_manual(values = c("#999999"), name = "Confidence Intervals") +
    theme_light() +
    labs(
      title = "Infectious (SIR model) v. Date",
      subtitle = bquote(.(location) ~ ":"
                        ~beta  == .(round(beta, 2)) ~ ","
                        ~gamma == .(round(gamma, 2)) ~ ","
                        ~R[0]  == .(round(R0, 2))),
      x = "Date",
      y = "Number of Infectious Individuals",
      color = "Key:"
    ) +
    theme(
      plot.title    = element_text(hjust =  0.5),
      plot.subtitle = element_text(hjust =  0.5),
      axis.title.y  = element_text(vjust =  3.0),
      axis.title.x  = element_text(vjust = -1.0),
      plot.caption  = element_text(vjust = -2.0, hjust = 1.0)
    )

}
```

```
plot_SIR_forecastCI(location = test_locale,
                    data  = Cook_IL_SIR_forecast_tidy_CIs,
                    x = dates,
                    y = infectious,
                    code  = code,
                    lower = lower,
                    upper = upper,
                    beta  = Cook_IL_SIR_optim$beta,
                    gamma = Cook_IL_SIR_optim$gamma)
```

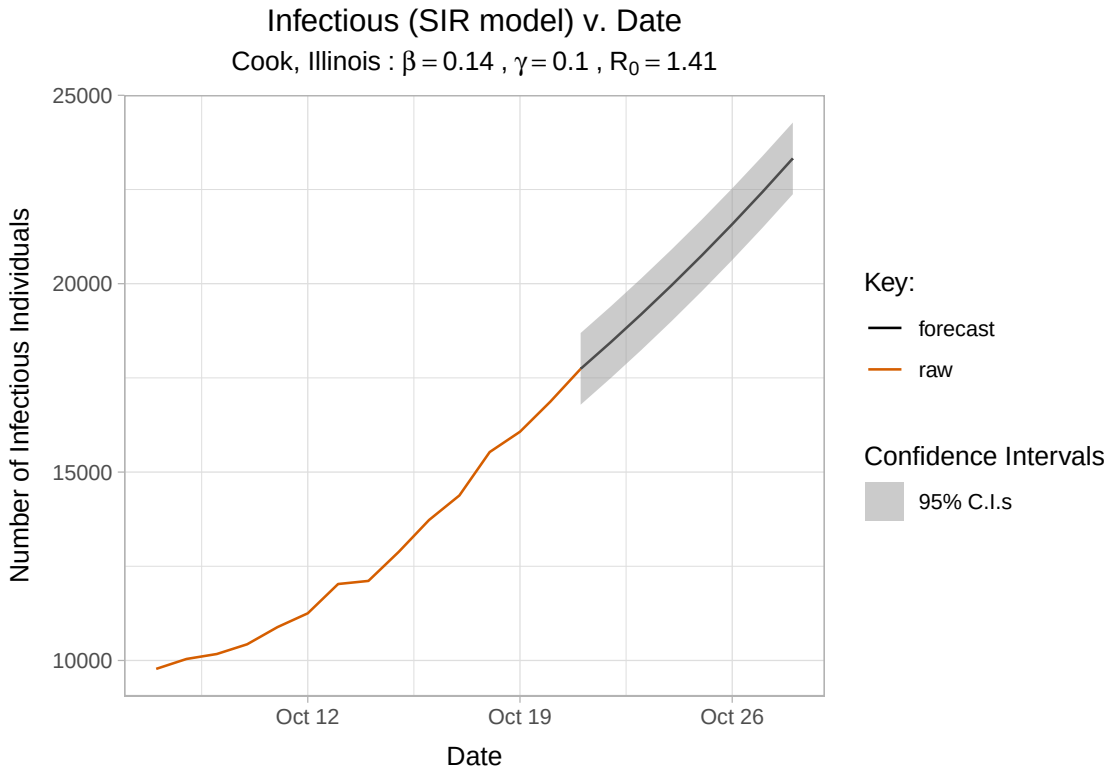


Figure 16: Plot of SIR forecast for Cook, Illinois.

Results and Discussion

Raw case counts show that in general major cities have been hit the hardest by the COVID-19 pandemic (Table 1).

Table 1: The top 10 U.S. locales with the greatest number of COVID-19 cases by state (JHU CSSE data obtained on 08 March 2021).

	State	No. Cases
1	Los Angeles, California	1203152
2	Maricopa, Arizona	516216
3	Cook, Illinois	478884
4	Miami-Dade, Florida	419479
5	Harris, Texas	359382
6	Clark, Nevada	228665
7	Queens, New York	218410
8	Salt Lake, Utah	139779
9	Philadelphia, Pennsylvania	119874
10	Middlesex, Massachusetts	113553

The large number of deaths due to COVID-19 in Kings, New York, in Essex, New Jersey, and in Middlesex, Massachusetts show that the areas surrounding these major cities have suffered a devastating impact as well (Table 2).

Table 2: The top 10 U.S. locales with the greatest number of COVID-19 deaths by state (JHU CSSE data obtained on 08 March 2021).

	State	No. Deaths
1	Los Angeles, California	22029
2	Cook, Illinois	9453
3	Maricopa, Arizona	9316
4	Kings, New York	9258
5	Miami-Dade, Florida	5558
6	Harris, Texas	5301
7	Wayne, Michigan	4172
8	Clark, Nevada	3931
9	Middlesex, Massachusetts	3493
10	Philadelphia, Pennsylvania	3170

The first cases are recorded in Washington and Illinois (Table 3). Soon after cases are recorded in California, Arizona, and Massachusetts. Following this initial detection there is at least a four week delay before the rest of the country begins recording cases in quick succession.

Table 3: The first 10 U.S. states to record COVID-19 cases (JHU CSSE data obtained on 08 March 2021).

	State	Grace Period
1	Washington	0
2	Illinois	2
3	Arizona	4
4	California	4
5	Massachusetts	7
6	Oregon	38
7	Rhode Island	39
8	Florida	40
9	New Hampshire	40
10	North Carolina	40

Since its arrival the pandemic has overall evolved differently across U.S. states with some broad regional trends as seen by the color patterns in the states maps shown above (Figs. 2, 3, 4, 5, 6). States that were the first to witness COVID-19 cases have not necessarily been the ones to suffer the worst. The top five states with the greatest total number of cases are California, Texas, Florida, Illinois, and New York (Table 4).

Table 4: The top 10 U.S. states with the highest COVID-19 cases count (JHU CSSE data obtained on 08 March 2021).

	State	Total No. Cases
1	California	3599250
2	Texas	2695558
3	Florida	1941496
4	New York	1694490
5	Illinois	1198255
6	Georgia	989975
7	Ohio	978471
8	Pennsylvania	950773
9	North Carolina	872176
10	Arizona	826454

Those states with the highest number of cases have generally also seen the greatest number of deaths (Fig. 3). New Jersey is a bit of an exception, although it does not make the top ten in number of cases it is in the top ten for number of COVID-19 deaths (Table 5).

Table 5: The top 10 U.S. states with the highest COVID-19 death count (JHU CSSE data obtained on 08 March 2021).

	State	Total No. Deceased
1	California	54220
2	New York	48042
3	Texas	45315
4	Florida	31683
5	Pennsylvania	24334
6	New Jersey	23574
7	Illinois	20764
8	Ohio	17501
9	Georgia	17337
10	Michigan	16494

Interestingly the states that have seen the greatest percentage of the population become infected are geographically distinct from those with high case counts. North Dakota, South Dakota, Iowa, Wisconsin, Nebraska have seen the greatest proportion of their respective populations become infected (Table 6). These states are all northern and inland (Fig.4).

Table 6: The top 10 U.S. states with the highest percentage of the population becoming infected with COVID-19 (JHU CSSE data obtained on 08 March 2021).

	State	Percent Infected
1	North Dakota	13.17360
2	South Dakota	12.83986
3	Arizona	11.35439
4	Tennessee	10.98745
5	Rhode Island	10.94481
6	Oklahoma	10.84142
7	Iowa	10.73488
8	Wisconsin	10.67688
9	Arkansas	10.65457
10	Nebraska	10.40339

The worst outcomes have been largely located in the North Eastern U.S.. New Jersey, New York, Massachusetts, North Dakota, and Connecticut have had the greatest reduction to their respective state populations due to COVID-19 (Table 7).

Table 7: Top 10 U.S. states with the highest percentage of the population dying from COVID-19 (JHU CSSE data obtained on 08 March 2021).

	State	% Population Deceased
1	New Jersey	0.2654075
2	New York	0.2469574
3	Massachusetts	0.2390231
4	Rhode Island	0.2359913
5	Mississippi	0.2287520
6	Arizona	0.2243252
7	Connecticut	0.2158031
8	South Dakota	0.2147720
9	Louisiana	0.2096888
10	Alabama	0.2069675

This is also reflected in COVID-19 mortality. As of this writing the mortality rate in most states is under 2.5% (Fig.6). New York, New Jersey, Massachusetts, and Connecticut are clear exceptions (Table 8).

Table 8: Top 10 U.S. states with the highest COVID-19 mortality (JHU CSSE data obtained on 08 March 2021).

	State	Mortality (CFR, in %)
1	Massachusetts	2.953035
2	New Jersey	2.905270
3	New York	2.835189
4	Connecticut	2.705839
5	Michigan	2.621245
6	Pennsylvania	2.559391
7	District of Columbia	2.486781
8	Mississippi	2.287781
9	Louisiana	2.249369
10	Rhode Island	2.156195

The static depictions of the data shown in the states maps do not give any indication of the trajectory of the pandemic. The heat map depicting the time series of the cases data (Fig. 7) shows that by October of 2020 the pandemic is not over and picking up momentum across nearly all U.S. states. The canonical SIR model is used to make predictions about disease spread. An even number of weeks (14 days) between the lower and upper bounds of the CDC's estimate for the infectious period (CDC 2020) is used to determine the number of infectious individuals over time from the JHU CSSE raw cases time series data. This data is used to fit the SIR model.

One approach to predicting the evolution of the pandemic is to look at areas that have already begun to witness cases. As the SIR model suggests, how a virus spreads is influenced by features specific to a given locale. K-means clustering is one approach to motivate comparison between distinct locations. Distances, similarity between locales, are calculated based on population and grace period (the number of days between when the first cases are reported in the U.S. and when cases begin to be recorded for the given locale). The within-cluster sum of squares continues to decrease with the formation of additional clusters, however beyond six clusters not much additional information can be gained (Fig. 8). As shown by the scatter plots clusters can be formed by subdivision along population or grace period, with additional clusters involving further subdivision along one of these variables (Fig. 9).

King, Washington and Queens, New York are consistently in the same cluster for all of the clustering values performed, up to $k = 7$ (Fig. 9). They have comparative populations and record cases during the start of the pandemic, but are geographically distinct; both are northern but King, Washington, is on the West Coast whereas Queens, New York, is on the East Coast of the U.S.. At the start of the pandemic, if people in Queens, New York wanted to get a sense of how the pandemic was going to impact them it would not have been unreasonable at the time to look at how things were evolving in King, Washington. From a data perspective, King, Washington serves as the training set and Queens, New York serves as the testing set. The SIR parameters obtained from fitting data for the first wave of infections at the start of the pandemic in King, Washington, when applied to Queens, New York, vastly underestimate the scope of the pandemic (Fig. 15).

Thus, the simple clustering scheme using the variables population and grace period is not sufficient to motivate comparison between locales for predicting COVID-19 dynamics. Drawing comparison with others is perhaps the only option available involving real data when trying to anticipate the COVID-19 pandemic before it reaches a particular locale. Once the virus is established the SIR model can be applied directly to real data associated with a given locale. The SIR model is the simplest possible and there are restrictions on its use. At least 300 infectious individuals are required before the model begins to provide a reasonable fit to the data (Figs. 11, 12). It is apparent from the heat map with contour lines of RSS values for the range of beta (β) and gamma (γ) values tested that the SIR model is unstable, that is there are many possible beta and gamma pairs that bring the RSS down to a level comparable with the global minimum (Fig.13). Optimization can partially address this but confidence intervals for the fit are essential for using this data for forecasting. Long term predictions with any model should be taken with a grain of salt as fundamental assumptions associated with virtually any model are continuously violated. The geography of COVID-19 analysis presented here is extended with Cook, Illinois, a major city in the Midwestern U.S.. Only a one week forecast is made based on data from the past two weeks.

The middle of October is chosen as the time period for consideration for reproducibility of the results, this also happens to be the month leading up to the 2020 U.S. presidential election. Confidence intervals using MLE based on the SIR fit between October 7 and October 21 are used to estimate the error associated with the forecast for the following week. The prediction is smoother than the actual data but quite accurate (Fig. 17).

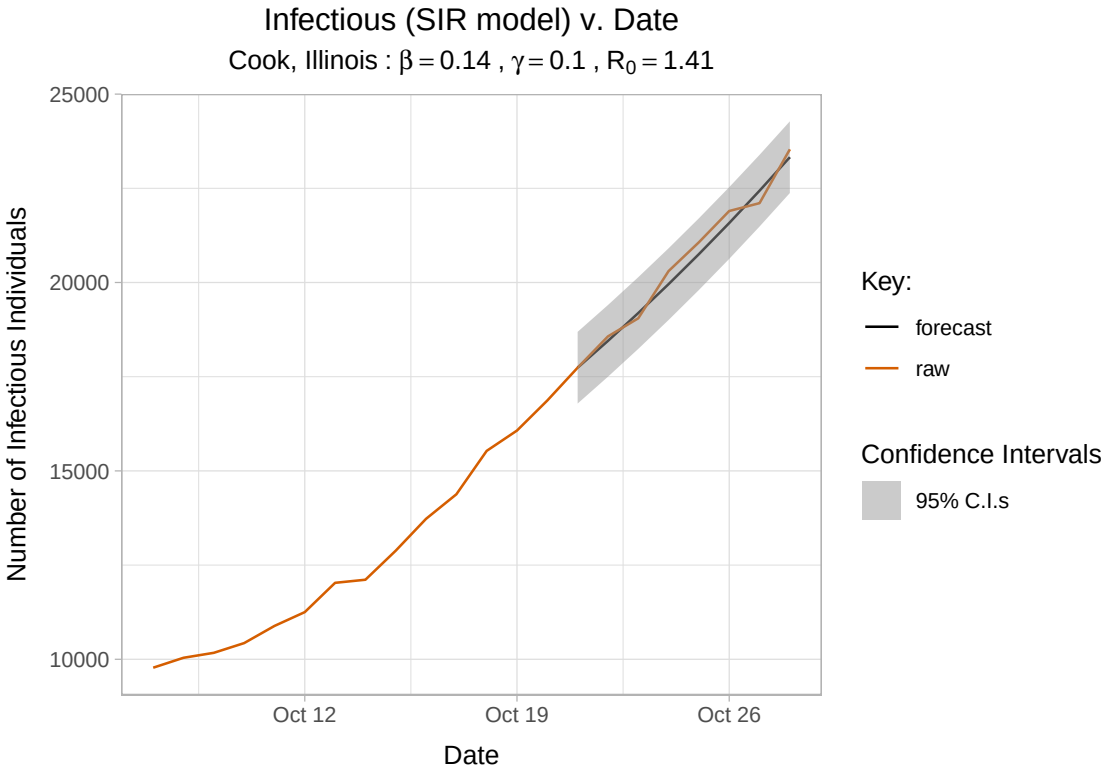


Figure 17: Late October SIR forecast for Cook, Illinois.

The RSS for the forecast is 5.0759143×10^5, a comparatively low value. Furthermore, the actual data the forecast is intended to predict is well within the forecast error.

Conclusions

Two approaches to predicting the spread of COVID-19 based on realistic scenarios are considered. The first, based on using one locale to infer the outcome for another is anticipatory. The second, which uses recent data to forecast about the immediate future is reactionary. The results show that a more robust clustering criteria than that presented here is needed to make reliable comparison between locales such that one can be predictive for another. It is easier to make accurate predictions based on short-term forecasts for a given locale.

Notwithstanding the simple approach to calculating the number of infectious individuals and the simple (SIR) model employed, this framework has practical application for monitoring the spread of COVID-19 once it reaches a threshold prevalence. Future work may focus on both improving the calculation for the population of infectious individuals, and expanding the SIR model to include additional compartments. The analysis presented here reiterates the need for rapid and wide-spread testing for two reasons. Firstly, the short-term nature of accurate forecasts relies on real-time data. Secondly, the case numbers can surge rapidly, overwhelming the health care system, which leads to increased mortality. The simple approach taken here serves as a firm foundation upon which to build a more complex predictive framework.

References

CDC. 2020. "Transmission - When Is Someone Infectious?" https://www.cdc.gov/coronavirus/2019-ncov/hcp/faq.html#Transmission.

WHO. 2020. "Estimating Mortality from COVID 19." https://www.who.int/news-room/commentaries/detail/estimating-mortality-from-covid-19.