Modeling the Spread of Norovirus on Commercial Cruise Voyages in R

Robert Sholl, Kelechi Igwe, Stephen Edache

2024-04-17

Contents

About

The following document details a step-by-step explanation for how to utilize R and data collected from the CDC Vessel Sanitation Program (VSP) to determine the rate of transmission of Norovirus on commercial cruise ships.

This will include shallow overviews of the mathematical justifications behind why certain functions and data cleaning measures will taken, but for a more rigorous synopsis please refer to our article.

Part I

Introduction

Background

Commercial cruise voyages are a multi-billion dollar tourism industry. With any large population sourcing from numerous different networks, (regardless of their level of prior connectivity), and an extended period of high volume interaction there are natural concerns for disease spread.

The VSP is primarily concerned with outbreaks of acute gastroenteritis, (AGE). Norovirus is the primary source of AGE across the world and thus makes up the vast majority of the outbreaks tracked by the VSP.

Project Intent

This project's intent is to utilize publicly available disease outbreak data from cruise ship passenger voyages to predict the likelihood of any one person getting norovirus based on what specific cruise ship they're traveling on, the number of people traveling on the same ship, where they are going, how long the trip is, and the timeframe in which the cruise will occur.

Structure

The structure of this document will be split into the functional units of the R scripts used in the project:

- Data Acquisition and Preparation
  - Infection Data
  - Temporal Data
  - Spatial Data
- Model Fitting

  - Program Validation

- Model Checking

  - Regressing Diagnostics
  - Graphical Diagnostics
  - Within-Sample Validation
  - Forecasting

- Interpretation

  - Closing Remarks
  - Source Code

Simplifying Assumptions

Simplifying assumptions are made throughout in order to achieve the end goal of this project. Aside from time and resource limitations, we have found three major factors that we believe justify the use of these assumptions.

VSP Definition of Outbreak The VSP only collects data on what is considered to be an outbreak, which they define on their website.

- Are on ships under VSP jurisdiction, defined as:

  - Carry 13 or more passengers
  - Have a foreign itinerary with U.S. ports.

- Are on ships carrying 100 or more passengers.
- Are on voyages from 3-21 days long.
- Are on voyages where 3% or more of passengers or crew report symptoms of gastrointestinal illness to the ship's medical staff.

We will use this as a baseline for our simplifying assumptions, and justify all of them within this definition.

Spread of Norovirus Simplifying for the sake of brevity, R_0 is a measure of how many additional infections are caused by each existing infection, with $R_0 > 1$ being seen as a rising outbreak, $R_0 = 1$ being a stable presence of disease, and $R_0 < 1$ being a decline towards eventual die out of infection.

Norovirus, along with all other AGE agents, is problematic from the perspective of generating a consistent and realistic number for an average R_0 due to the prevalence of under reporting associated with this category of illness. Reported values show the range of $1 < R_0 < 8$ with a general average of $R_0 \approx 2$. Compared with seasonal influenza $1 < R_0 < 2$ and recent observations of the highest value for SARS-CoV-19 Omicron $R_0 = 1.9$, Norovirus is a highly infectious virus.

We can make the statement then, that any existence of a single case of Norovirus on a cruise vessel will result in an outbreak given that the virus travels fast enough that traditional outbreak control measures won't prevent the VSP definition of an outbreak from occurring by the time symptoms become noticeable enough to warrant action being taken.

The Jones Act and The Passenger Vessel Services Act These maritime laws paint two restrictions on cruise ships simultaneously.

- Any vessel carrying merchandise between U.S. ports must be built, owned, and operated by U.S. Citizens, as well as constructed out of U.S. owned materials.
- Foreign flagged ships must start and end their journey at the same port provided they began in a U.S. port, and must have a distant foreign port at part of their journey.

These restrictions in practice result in a system used by most cruise liners.

- Almost all cruise ships have U.S. ports.
- Almost all cruise ships have foreign ports.
- Almost all cruise ships start and end at the same location.

We believe that this justifies the sole use of VSP data for modeling the totality of Norovirus outbreaks on cruise vessels, due to a negligibly small population of cruise voyages having no interaction with U.S. ports at any given time.

Part II

Data

Data Acquisition and Preparation

This chapter will cover all data acquisition, preparation, and organization. Each step in this process will be "over-detailed" to an extent. This is done to act more as guide of the thought process and intent of the code in the event that:

> The R packages and syntax used have lost support and become outdated.
>
> The individual reading this intends to use an alternate programming language for replication of this project.

Despite this, there will still be a component of acting as an in-depth R tutorial. The following packages will be needed for this book's content, so their install commands have been included below:

Final Note: The entirety of the original code is riddled with "shower thought" markdowns. These have been left out of this tutorial in lieu of a more "professionally acceptable" documentation style, but they remain in the original code which can be viewed via a download link in the references portion of this book. This is just an acknowledgement, not an apology.

Acquisition

We first start with the acquisition of the VSP data. This can be found on the CDC website: https://www.cdc.gov/nceh/vsp/surv/gilist.htm

The data is in descending chronological order, with the most recent outbreak on the current year shown first, and each year separated into its own block.

We chose to use Excel to query the data directly from the website. It should be noted that there is a massive variety of data scraping techniques available, all of which are more efficient at the process than Excel. We chose Excel because it was the most user-friendly and readily accessible method at the time.

The method for performing a query can be seen in the screenshots below:

Figure 1: 1. Select the 'Data' tab 2. Select 'From Other Sources' 3. Select 'From Web'

Once the data is in the Excel document all that needs to be done is save the individual sheets in .csv format. We chose to do minor cleaning before-hand, and the steps associated with saving individual sheets is a needlessly cumbersome burden to place on the reader, so a dropbox link is included below containing the .csv files used in the proceeding code. Downloading the files is preferable to using the dropbox URL to reduce computing burden every time troubleshooting has to occur or a new environment is being loaded up.

Figure 2: Input the URL containing your data.

Figure 3: Check the ‘Select Multiple Items’ box if you wish you pull more than one, otherwise select one and press load.

Dropbox is used over github for end-user friendliness.

URL

https://www.dropbox.com/scl/fo/7d2e8ro0e8mbkoxkbi6c3/AMT0sQ86B27OIEZopkIMXhg?rlkey=e5otlpcp0rr7931h3j1u5r3cp&dl=0

Infection Data

Place the files into an accessible folder of the RStudio environment you're using. The code string below will read them into R as named, accessible units.

In order to get the code to function, you will have to rename the file paths. An easy way to ensure the file path is correct and readable from your project environment you can write the read.csv(" ") command and press 'Tab' while the space between the two quotations is selected. This should show a list of files available to the environment that you can parse through to find what you're trying to connect to the command.

You will need the following packages for all data preparation, the code block below will make them available to your project environment post-intallation:

```
cru.noro.14 <- read.csv("_book/_main_files/.csv files/Cruise_2014.csv")
cru.noro.15 <- read.csv("_book/_main_files/.csv files/Cruise_2015.csv")
cru.noro.16 <- read.csv("_book/_main_files/.csv files/Cruise_2016.csv")
cru.noro.17 <- read.csv("_book/_main_files/.csv files/Cruise_2017.csv")
cru.noro.18 <- read.csv("_book/_main_files/.csv files/Cruise_2018.csv")
cru.noro.19 <- read.csv("_book/_main_files/.csv files/Cruise_2019.csv")
```

```
head(cru.noro.19)
```

```
##                 Column1       Column2              Column3            Column4
## 1           Cruise Line   Cruise Ship Sailing_Dates(Start) Sailing_Dates(End)
## 2 Norwegian Cruise Line Norwegian Joy            11/8/2019         11/24/2019
## 3          Aida Cruises      AIDAdiva            10/3/2019         10/13/2019
## 4          Aida Cruises      AIDAluna            9/28/2019         10/12/2019
## 5          Aida Cruises      AIDAdiva             9/5/2019          9/23/2019
## 6  Carnival Cruise Line      Conquest             9/7/2019          9/14/2019
##           Column5                                      Column6
## 1 Causative Agent Proportion_Case_Counts_Numerator(Passengers)
## 2         Unknown                                         127
```

```
## 3       Norovirus                                          83
## 4       Norovirus                                          85
## 5       Norovirus                                         117
## 6       Norovirus                                          17
##                                          Column7                    Column8
## 1 Proportion_Case_Counts_Denominator(Passengers) %_Case_Counts(Passengers)
## 2                                          3,602                      3.53
## 3                                          2,251                      3.70
## 4                                          2,166                      3.90
## 5                                          2,055                      5.70
## 6                                          3,241                      0.52
##                                    Column9
## 1 Proportion_Case_Counts_Numerator(Crew)
## 2                                      6
## 3                                     10
## 4                                     10
## 5                                      8
## 6                                     35
##                                   Column10            Column11
## 1 Proportion_Case_Counts_Denominator(Crew) %_Case_Counts(Crew)
## 2                                  1,769               0.34
## 3                                    610               1.60
## 4                                    612               1.60
## 5                                    610               1.30
## 6                                   1158               3.02
```

Reading through the files you will find that there is a large amount of excess information, we want to isolate the numerator, (confirmed infected), and denominator, (total population), columns in each file so that we can begin working with the rates of infection.

It's convenient to place the data we need into data frames as well, so we can accomplish two goals with one action per sheet by defining the columns we want as columns in a new data frame:

```
df.cru.14 <- data.frame(pcn = cru.noro.14[,6],
                        pcd = cru.noro.14[,7],
                        ccn = cru.noro.14[,10],
                        ccd = cru.noro.14[,11])
df.cru.15 <- data.frame(pcn = cru.noro.15[,6],
                          pcd = cru.noro.15[,7],
                          ccn = cru.noro.15[,9],
                          ccd = cru.noro.15[,10])
df.cru.16 <- data.frame(pcn = cru.noro.16[,6],
                        pcd = cru.noro.16[,7],
                        ccn = cru.noro.16[,9],
```

```
                         ccd = cru.noro.16[,10])
df.cru.17 <- data.frame(pcn = cru.noro.17[,6],
                         pcd = cru.noro.17[,7],
                         ccn = cru.noro.17[,9],
                         ccd = cru.noro.17[,10])
df.cru.18 <- data.frame(pcn = cru.noro.18[,6],
                         pcd = cru.noro.18[,7],
                         ccn = cru.noro.18[,9],
                         ccd = cru.noro.18[,10])
df.cru.19 <- data.frame(pcn = cru.noro.19[,6],
                         pcd = cru.noro.19[,7],
                         ccn = cru.noro.19[,9],
                         ccd = cru.noro.19[,10])

head(df.cru.14)
```

```
##                                 pcn                                 pcd
## 1 Case_Counts_Numerator(Passengers) Case_Counts_Denominator(Passengers)
## 2                                158                               3,009
## 3                                122                               3,161
## 4                                 97                               2,120
## 5                                111                               2,122
## 6                                 65                               1,096
##                             ccn                           ccd
## 1 Case_Counts_Numerator(Crew) Case_Counts_Denominator(Crew)
## 2                            14                         1,160
## 3                            30                         1,176
## 4                             8                           808
## 5                             6                           790
## 6                             8                           569
```

The top row of each column contains the descriptions of the columns rather than any usable data, so we'll remove those:

```
df.cru.14c <- df.cru.14[-c(1),]
df.cru.15c <- df.cru.15[-c(1),]
df.cru.16c <- df.cru.16[-c(1),]
df.cru.17c <- df.cru.17[-c(1),]
df.cru.18c <- df.cru.18[-c(1),]
df.cru.19c <- df.cru.19[-c(1),]

head(df.cru.14c)
```

```
##   pcn   pcd ccn   ccd
```

```
## 2 158 3,009  14 1,160
## 3 122 3,161  30 1,176
## 4  97 2,120   8   808
## 5 111 2,122   6   790
## 6  65 1,096   8   569
## 7 114 1,273  10   575
```

The numeric data includes commas to denote 1000s, while we can declare prefixes before working with the data to ensure these aren't a problem, it's more convenient for future troubleshooting of errors to not have them involved in our data frames.

We'll remove them by using the `gsub` command and "replacing" our commas with nothing.

```
df.cru.14c$pcd <- gsub(",","", df.cru.14c$pcd)
df.cru.14c$ccd <- gsub(",","", df.cru.14c$ccd)

df.cru.15c$pcd <- gsub(",","", df.cru.15c$pcd)
df.cru.15c$ccd <- gsub(",","", df.cru.15c$ccd)

df.cru.16c$pcd <- gsub(",","", df.cru.16c$pcd)
df.cru.16c$ccd <- gsub(",","", df.cru.16c$ccd)

df.cru.17c$pcd <- gsub(",","", df.cru.17c$pcd)
df.cru.17c$ccd <- gsub(",","", df.cru.17c$ccd)

df.cru.18c$pcd <- gsub(",","", df.cru.18c$pcd)
df.cru.18c$ccd <- gsub(",","", df.cru.18c$ccd)

df.cru.19c$pcd <- gsub(",","", df.cru.19c$pcd)
df.cru.19c$ccd <- gsub(",","", df.cru.19c$ccd)

head(df.cru.19c)
```

```
##   pcn  pcd ccn  ccd
## 2 127 3602   6 1769
## 3  83 2251  10  610
## 4  85 2166  10  612
## 5 117 2055   8  610
## 6  17 3241  35 1158
## 7  36 1066   4  763
```

Infection Totals and Rate

Our basic infection data is now sufficiently cleaned up to begin organizing it into more useful formats.

Next we'll compute the infection rate, which, as our labels of the data suggests, follows a simple rate of change format:

$$Rate = \frac{Cases}{Total}$$

The code below generates a total count column across all voyages for each individual sheet:

```
df.cru.14c <- df.cru.14c %>% mutate(tcn = as.numeric(df.cru.14c$pcn) + as.numeric(df.cru.14c$ccn)
df.cru.14c <- df.cru.14c %>% mutate(tcd = as.numeric(df.cru.14c$pcd) + as.numeric(df.cru.14c$ccd)

df.cru.15c <- df.cru.15c %>% mutate(tcn = as.numeric(df.cru.15c$pcn) + as.numeric(df.cru.15c$ccn)
df.cru.15c <- df.cru.15c %>% mutate(tcd = as.numeric(df.cru.15c$pcd) + as.numeric(df.cru.15c$ccd)

df.cru.16c <- df.cru.16c %>% mutate(tcn = as.numeric(df.cru.16c$pcn) + as.numeric(df.cru.16c$ccn)
df.cru.16c <- df.cru.16c %>% mutate(tcd = as.numeric(df.cru.16c$pcd) + as.numeric(df.cru.16c$ccd)

df.cru.17c <- df.cru.17c %>% mutate(tcn = as.numeric(df.cru.17c$pcn) + as.numeric(df.cru.17c$ccn)
df.cru.17c <- df.cru.17c %>% mutate(tcd = as.numeric(df.cru.17c$pcd) + as.numeric(df.cru.17c$ccd)

df.cru.18c <- df.cru.18c %>% mutate(tcn = as.numeric(df.cru.18c$pcn) + as.numeric(df.cru.18c$ccn)
df.cru.18c <- df.cru.18c %>% mutate(tcd = as.numeric(df.cru.18c$pcd) + as.numeric(df.cru.18c$ccd)

df.cru.19c <- df.cru.19c %>% mutate(tcn = as.numeric(df.cru.19c$pcn) + as.numeric(df.cru.19c$ccn)
df.cru.19c <- df.cru.19c %>% mutate(tcd = as.numeric(df.cru.19c$pcd) + as.numeric(df.cru.19c$ccd)
```

This code block computes the rate of infection:

```
df.cru.14c <- df.cru.14c %>% mutate(pc = as.numeric(df.cru.14c$pcn) / as.numeric(df.cru.14c$pcd))
df.cru.14c <- df.cru.14c %>% mutate(cc = as.numeric(df.cru.14c$ccn) / as.numeric(df.cru.14c$ccd))
df.cru.14c <- df.cru.14c %>% mutate(tc = as.numeric(df.cru.14c$tcn) / as.numeric(df.cru.14c$tcd))

df.cru.15c <- df.cru.15c %>% mutate(pc = as.numeric(df.cru.15c$pcn) / as.numeric(df.cru.15c$pcd))
df.cru.15c <- df.cru.15c %>% mutate(cc = as.numeric(df.cru.15c$ccn) / as.numeric(df.cru.15c$ccd))
df.cru.15c <- df.cru.15c %>% mutate(tc = as.numeric(df.cru.15c$tcn) / as.numeric(df.cru.15c$tcd))

df.cru.16c <- df.cru.16c %>% mutate(pc = as.numeric(df.cru.16c$pcn) / as.numeric(df.cru.16c$pcd))
df.cru.16c <- df.cru.16c %>% mutate(cc = as.numeric(df.cru.16c$ccn) / as.numeric(df.cru.16c$ccd))
df.cru.16c <- df.cru.16c %>% mutate(tc = as.numeric(df.cru.16c$tcn) / as.numeric(df.cru.16c$tcd))
```

```
df.cru.17c <- df.cru.17c %>% mutate(pc = as.numeric(df.cru.17c$pcn) / as.numeric(df.cru
df.cru.17c <- df.cru.17c %>% mutate(cc = as.numeric(df.cru.17c$ccn) / as.numeric(df.cru
df.cru.17c <- df.cru.17c %>% mutate(tc = as.numeric(df.cru.17c$tcn) / as.numeric(df.cru

df.cru.18c <- df.cru.18c %>% mutate(pc = as.numeric(df.cru.18c$pcn) / as.numeric(df.cru
df.cru.18c <- df.cru.18c %>% mutate(cc = as.numeric(df.cru.18c$ccn) / as.numeric(df.cru
df.cru.18c <- df.cru.18c %>% mutate(tc = as.numeric(df.cru.18c$tcn) / as.numeric(df.cru

df.cru.19c <- df.cru.19c %>% mutate(pc = as.numeric(df.cru.19c$pcn) / as.numeric(df.cru
df.cru.19c <- df.cru.19c %>% mutate(cc = as.numeric(df.cru.19c$ccn) / as.numeric(df.cru
df.cru.19c <- df.cru.19c %>% mutate(tc = as.numeric(df.cru.19c$tcn) / as.numeric(df.cru
```

Sectioning Off Data

Now we'll pull the relevant columns out of our data frames and place them into vectors for later use in our models.

It should be noted that this is not an "absolute" methodology for working with data in R, so long as the data ends up in the intended places and serves the intended purpose any method for getting it to that point can be used.

Below you'll see us first pull our infection rates, then our "sizes" which is our nomenclature of sample population for the purpose of this project (total passenger and crew counts), and our total cases.

```
# vector of infection rates
cru.rate <- c(tc.14 = df.cru.14c$tc,
              tc.15 = df.cru.15c$tc,
              tc.16 = df.cru.16c$tc,
              tc.17 = df.cru.17c$tc,
              tc.18 = df.cru.18c$tc,
              tc.19 = df.cru.19c$tc)

# vector of cruise sizes in the usp data
Ac.size <- c(tc.14 = df.cru.14c$tcd,
             tc.15 = df.cru.15c$tcd,
             tc.16 = df.cru.16c$tcd,
             tc.17 = df.cru.17c$tcd,
             tc.18 = df.cru.18c$tcd,
             tc.19 = df.cru.19c$tcd)

# vector of case counts in the usp data
Ac.cases <- c(tc.14 = df.cru.14c$tcn,
              tc.15 = df.cru.15c$tcn,
              tc.16 = df.cru.16c$tcn,
```

```
              tc.17 = df.cru.17c$tcn,
              tc.18 = df.cru.18c$tcn,
              tc.19 = df.cru.19c$tcn)
```

With this we can say that our disease data is set up for our model. We'll move on to our other model components.

Temporal Data

As the goal of this project is to produce a model that incorporates both space and time it's best we clean up and prepare our temporal component of the data.

Voyage Duration

We'll follow a similar system as with our infection data and place our start and end dates into data frames.

```
df.t14 <- data.frame(start = cru.noro.14[,3],
                     end = cru.noro.14[,4])
df.t15 <- data.frame(start = cru.noro.15[,3],
                      end = cru.noro.15[,4])
df.t16 <- data.frame(start = cru.noro.16[,3],
                      end = cru.noro.16[,4])
df.t17 <- data.frame(start = cru.noro.17[,3],
                      end = cru.noro.17[,4])
df.t18 <- data.frame(start = cru.noro.18[,3],
                      end = cru.noro.18[,4])
df.t19 <- data.frame(start = cru.noro.19[,3],
                      end = cru.noro.19[,4])

head(df.t14)
```

```
##                   start                 end
## 1 Sailing_Dates(Start) Sailing_Dates(End)
## 2           10/18/2014         11/16/2014
## 3             4/5/2014          4/12/2014
## 4             4/5/2014          4/12/2014
## 5            3/28/2014           4/5/2014
## 6             3/2/2014          3/28/2014
```

While this isn't a situation of actual messy data, it's appropriate to say the data's format is very "messy". We see once again that we have a descriptive row at the top of each column, using the same technique we'll remove it.

```
df.t14c <- df.t14[-c(1),]
df.t15c <- df.t15[-c(1),]
df.t16c <- df.t16[-c(1),]
df.t17c <- df.t17[-c(1),]
df.t18c <- df.t18[-c(1),]
df.t19c <- df.t19[-c(1),]
```

We'll now reformat the dates into Month/Day/Year using the `dplyr mutate` function.

```
df.t14c <- df.t14c %>% mutate(start = mdy(start))
df.t14c <- df.t14c %>% mutate(end = mdy(end))

df.t15c <- df.t15c %>% mutate(start = mdy(start))
df.t15c <- df.t15c %>% mutate(end = mdy(end))

df.t16c <- df.t16c %>% mutate(start = mdy(start))
df.t16c <- df.t16c %>% mutate(end = mdy(end))

df.t17c <- df.t17c %>% mutate(start = mdy(start))
df.t17c <- df.t17c %>% mutate(end = mdy(end))

df.t18c <- df.t18c %>% mutate(start = mdy(start))
df.t18c <- df.t18c %>% mutate(end = mdy(end))

df.t19c <- df.t19c %>% mutate(start = mdy(start))
df.t19c <- df.t19c %>% mutate(end = mdy(end))
```

Our goal is to use this as a way to get the duration these cruise ships are at sea. We also want to retain these start and end dates for future use in potential variables so we'll be sure to declare a new column rather than eliminate any currently cleaned up data.

Taking the difference between the start and end will give us a number of days. We have to declare our column data `as.Date` in order for it to work in this function.

It's more logical to subtract the end from the start rather than what we do with the code below, but this does allow an opportunity to detail the syntax for taking absolute values so it's useful to the overall tutorial to retain this method regardless of redundancy.

```
df.t14c$d_t <- c(as.Date(as.character(df.t14c$start))-as.Date(as.character(df.t14c$end)
df.t15c$d_t <- c(as.Date(as.character(df.t15c$start))-as.Date(as.character(df.t15c$end)
df.t16c$d_t <- c(as.Date(as.character(df.t16c$start))-as.Date(as.character(df.t16c$end)
```

```
df.t17c$d_t <- c(as.Date(as.character(df.t17c$start))-as.Date(as.character(df.t17c$end)))
df.t18c$d_t <- c(as.Date(as.character(df.t18c$start))-as.Date(as.character(df.t18c$end)))
df.t19c$d_t <- c(as.Date(as.character(df.t19c$start))-as.Date(as.character(df.t19c$end)))
```

```
# convert negative values to positive

df.t14c$d_t <- abs(df.t14c$d_t)
df.t15c$d_t <- abs(df.t15c$d_t)
df.t16c$d_t <- abs(df.t16c$d_t)
df.t17c$d_t <- abs(df.t17c$d_t)
df.t18c$d_t <- abs(df.t18c$d_t)
df.t19c$d_t <- abs(df.t19c$d_t)
```

Now we merge all of our cleaned up time data into one convenient data frame.

```
df.tt <- bind_rows(df.t14c, df.t15c, df.t16c, df.t17c, df.t18c, df.t19c)

head(df.tt)
```

```
##        start        end     d_t
## 1 2014-10-18 2014-11-16 29 days
## 2 2014-04-05 2014-04-12  7 days
## 3 2014-04-05 2014-04-12  7 days
## 4 2014-03-28 2014-04-05  8 days
## 5 2014-03-02 2014-03-28 26 days
## 6 2014-02-08 2014-02-22 14 days
```

Voyage Seasons

We want to categorize our data into seasons to gain another discrete temporal variable. It's easier to do this with our new data frame constructed out of our cleaned up time data, so please remember to follow these steps in the order they are presented to get proper results.

It isn't a particularly clean technique to create a new data frame every time you want to create more variables.

In an environment where your code may inevitably be passed along to someone further ahead in production than yourself you will want to clear all of the excess vectors and data frames we make use of here.

However, it cannot be understated the value of using this practice when testing unfamiliar programming techniques. By separating variables of interest from their sources you can avoid a lot of unnecessary back tracking through your data preparation scripts whenever an error is made. This practice can also make

troubleshooting errors a smooth, piece-wise operation, rather than a gauntlet of interpreting error logs.

We'll create a new data frame to isolate the columns we want to work with.

```
df.seasons <- data.frame(start = df.tt$start,
                         end = df.tt$end)
```

We'll make use of the `dplyr` package again to reformat our M/D/Y format to just months. `lubridate` is currently a useful replacement for this package, in this context, but I found `dplyr` to be sufficient for this specific task and far less prone to errors.

The syntax below simply declares the columns for the start and end dates as just the month component of each current column. If you receive errors after this point you will likely have to rebuild the `df.seasons` data frame.

```
df.seasons <- df.seasons %>% mutate(start = month(start))
df.seasons <- df.seasons %>% mutate(end = month(end))
```

Since we are categorizing our start and end dates by seasons, we have to declare what these seasons will look like in our final product. When looking at both the data available and the function being used, it's most convenient to organize them in the following way:

> 1 Winter = December to February (12 to 2) 2 Spring = March to May (3 to 5) 3 Fall = September to November (9 to 11) 4 Summer = June to August (6 to 8)

`findInterval` works well for this, each number in `c(...)` has to be the cut-off for the category. Since winter includes a month labeled with a number greater than our cut-off value in it's category, we'll use `gsub` once again to change anything that is found to be in the `"0"` interval into the `"1"` interval.

```
df.seasons$start <- findInterval(df.seasons$start, c(2,5,8,11))
df.seasons$end <- findInterval(df.seasons$end, c(2,5,8,11))

# converting all 0s to 1s
df.seasons$start <- gsub("0","1",df.seasons$start)
df.seasons$end <- gsub("0","1",df.seasons$end)

head(df.seasons)
```

```
##   start end
## 1     3   4
```

```
## 2     1   1
## 3     1   1
## 4     1   1
## 5     1   1
## 6     1   1
```

We now have all of our time data prepared and organized for our model.

Spatial Data

For the purpose of this project, the cruise ships themselves will be treated as a "Spatial" component. There will be three major steps to this process.

- Isolating Cruise Liners (Corporate ownership) and Cruise Ships (The named vessel).
- Creating "size" intervals (reminder that size references sample population).
- Adding in their port of origin as a column.

Liners and Ships

First, we go back to our original data and place it all into one unified data frame.

```
df.raw.all <- bind_rows(cru.noro.14,cru.noro.15,cru.noro.16,cru.noro.17,cru.noro.18,cru.noro.19)
```

Then we remove all of the rows that are our spreadsheet headers.

```
df.raw.all <- df.raw.all[-c(1),]
df.raw.all <- df.raw.all[-c(10),]
df.raw.all <- df.raw.all[-c(22),]
df.raw.all <- df.raw.all[-c(35),]
df.raw.all <- df.raw.all[-c(46),]
df.raw.all <- df.raw.all[-c(57),]
```

To finish this step we separate our liner and ship IDs into a new data frame.

```
df.las <- data.frame(liner = df.raw.all[,1],
                     ship = df.raw.all[,2])

head(df.las)
```

```
##                          liner                ship
## 1            Princess Cruises       Crown Princess
## 2            Princess Cruises       Crown Princess
## 3 Royal Caribbean Cruise Line Grandeur of the Seas
## 4 Royal Caribbean Cruise Line Grandeur of the Seas
## 5        Holland America Line           ms Maasdam
## 6        Holland America Line           ms Veendam
```

Step one of our process is complete.

Size Intervals

This step uses the same technique as the season categorization. First we declare our categories:

> 1 small cruise: 50-800 2 small-mid cruise: 801-1500 3 mid-sized cruise: 1501-2500 4 large cruise: 2501-3500 5 mega cruise: 3501+

Then we apply our interval function and place the sizes into a column vector.

```
size.interv <- findInterval(Ac.size, c(50,801,1501,2501,3501))

head(size.interv)
```

```
## [1] 5 5 4 4 3 3
```

Step two is complete.

Ports of Origin

Gathering the port of origin for these ships is a more complicated act of research and data scraping than the scope of this tutorial allows. If you would like to learn more about making API calls, please refer to this website:

https://learn.microsoft.com/en-us/aspnet/web-api/overview/advanced/calling-a-web-api-from-a-net-client

Data of the itineraries of the exact voyages for these cruises was found, and condensed into a .csv file.

This file can be downloaded at the below URL:

https://www.dropbox.com/scl/fi/2efm7ugkhjanzqc0opiz6/Cleanedstartendcru.csv?rlkey=gpnynbxwlllzrb2v8gaynyc7e&dl=0

```
cru.strt_lat_long <- read.csv("_book/_main_files/.csv files/Cleanedstartendcru.csv")

head(cru.strt_lat_long)
```

```
##                   Ship Start.Destination End.Destination Year Start.Lat
## 1       Crown Princess       Los Angeles      California 2014   33.7405
## 2       Crown Princess       Los Angeles      California 2014   33.7405
## 3 Grandeur of the Seas         Baltimore         Bahamas 2014   39.2876
## 4 Grandeur of the Seas         Baltimore         Bahamas 2014   39.2876
## 5           ms Maasdam    Rio de Janeiro Fort Lauderdale 2014  -22.8955
## 6           ms Veendam         San Diego Fort Lauderdale 2014   32.7157
##   Start.Long  X X.1 X.2 X.3 X.4 X.5 X.6 X.7 X.8 X.9 X.10 X.11
## 1  -118.2786 NA  NA  NA  NA  NA  NA  NA  NA  NA  NA   NA   NA
## 2  -118.2786 NA  NA  NA  NA  NA  NA  NA  NA  NA  NA   NA   NA
## 3   -76.6108 NA  NA  NA  NA  NA  NA  NA  NA  NA  NA   NA   NA
## 4   -76.6108 NA  NA  NA  NA  NA  NA  NA  NA  NA  NA   NA   NA
## 5   -43.1822 NA  NA  NA  NA  NA  NA  NA  NA  NA  NA   NA   NA
## 6  -117.1611 NA  NA  NA  NA  NA  NA  NA  NA  NA  NA   NA   NA
```

There's a significant amount of NAs in columns of NAs. These columns are likely just errors in reading the .csv, as no data from the original .csv is missing. We'll clean those up by changing `cru.strt_lat_long` to a data frame with only the columns we want to retain.

```
cru.strt_lat_long <- data.frame(Ship = cru.strt_lat_long$Ship,
                                Start_dest = cru.strt_lat_long$Start.Destination,
                                End_dest = cru.strt_lat_long$End.Destination,
                                Start_lat = cru.strt_lat_long$Start.Lat,
                                Start_long = cru.strt_lat_long$Start.Long)

head(cru.strt_lat_long)
```

```
##                   Ship     Start_dest        End_dest Start_lat Start_long
## 1       Crown Princess    Los Angeles      California   33.7405  -118.2786
## 2       Crown Princess    Los Angeles      California   33.7405  -118.2786
## 3 Grandeur of the Seas      Baltimore         Bahamas   39.2876   -76.6108
## 4 Grandeur of the Seas      Baltimore         Bahamas   39.2876   -76.6108
## 5           ms Maasdam Rio de Janeiro Fort Lauderdale  -22.8955   -43.1822
## 6           ms Veendam      San Diego Fort Lauderdale   32.7157  -117.1611
```

Final Preparations

When we inspect this data frame now we'll notice there are three missing cruise ships from the list. These ships did not have itinerary data available. Case

deletion will be used for this situation, with the assumption that the loss of three data points due to lack of available associated spatial data will not cause significant error in our models.

Understand, with every ounce of truth I can weigh against this statement, that it is far easier to locate the three missing cruise vessels manually than it is to use any amount of programming to automate the task. I retract that statement in the event there's a machine learning expert reading this tutorial for reasons far beyond comprehension.

We'll remove the rows containing these cruise vessels. In order to do this lets make our first unified data frame.

```
cruisedata.df <- data.frame(cases = Ac.cases,
                            size = Ac.size,
                            infection = cru.rate,
                            liner = df.las[,1],
                            ship = df.las[,2],
                            size.cat = factor(size.interv),
                            start_date = df.tt[,1],
                            end_date = df.tt[,2],
                            start_season = df.seasons[,1],
                            end_season = df.seasons[,2],
                            total_days = df.tt[,3])
```

Then delete the rows of interest.

```
{
  cruisedata.df <- cruisedata.df[-c(19),]
  cruisedata.df <- cruisedata.df[-c(37),]
  cruisedata.df <- cruisedata.df[-c(42),]
}
```

Finally we add our latitude and longitudes to the unified data frame.

```
cruisedata.df <- data.frame(cases = cruisedata.df$cases,
                            size = cruisedata.df$size,
                            infection = cruisedata.df$infection,
                            liner = cruisedata.df$liner,
                            ship = cruisedata.df$ship,
                            size.cat = cruisedata.df$size.cat,
                            start_date = cruisedata.df$start_date,
                            end_date = cruisedata.df$end_date,
                            start_season = cruisedata.df$start_season,
                            end_season = cruisedata.df$end_season,
                            total_days = cruisedata.df$total_days,
```

```
                              Start_dest = cru.strt_lat_long$Start_dest,
                              End_dest = cru.strt_lat_long$End_dest,
                              Start_lat = cru.strt_lat_long$Start_lat,
                              Start_long = cru.strt_lat_long$Start_long)
```

We've completed the tedious portion of this project. The end result is a large data frame with all of the variables we've deemed necessary and appropriate for our analytics.

Part III

Model Fitting

Model selection

The objective of this R program is to build a viable model for making predictions on probability of infection for cruise voyages based on set parameters. Two models will be used and checked against one another:

- An intercept-only linear model (serving as a reference)
- A binomial regression model

Results of each model will be shown, but analysis on what they provide will not be covered in this section.

For the analytics, please refer to Analysis chapter.

Intercept-Only Model

The intercept-only regression model contains no predictor variables. For our purpose, it serves as a baseline reference to the efficacy of our other models.

The general model is shown below:

$$Y_{ij} = \beta_0 * \epsilon_{i,j} \tag{1}$$

We're using the duration of our voyages as the intercept, and plotting the results of the model against the reported rate of infection and the duration of the voyages.

The `lm` function is used, where we declare our `infection` variable, (our reported rates), to `total_days`, (our voyage duration), referencing our master data frame.

```
m1 <- lm(infection ~ size, data=cruisedata.df)

summary(m1)
```

```
##
## Call:
## lm(formula = infection ~ size, data = cruisedata.df)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.034107 -0.012933 -0.004742  0.002294  0.155384
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 4.295e-02  8.011e-03   5.361 1.34e-06 ***
## size        6.776e-07  2.529e-06   0.268     0.79
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.02882 on 61 degrees of freedom
## Multiple R-squared:  0.001176,   Adjusted R-squared:  -0.0152
## F-statistic: 0.07181 on 1 and 61 DF,  p-value: 0.7896
```

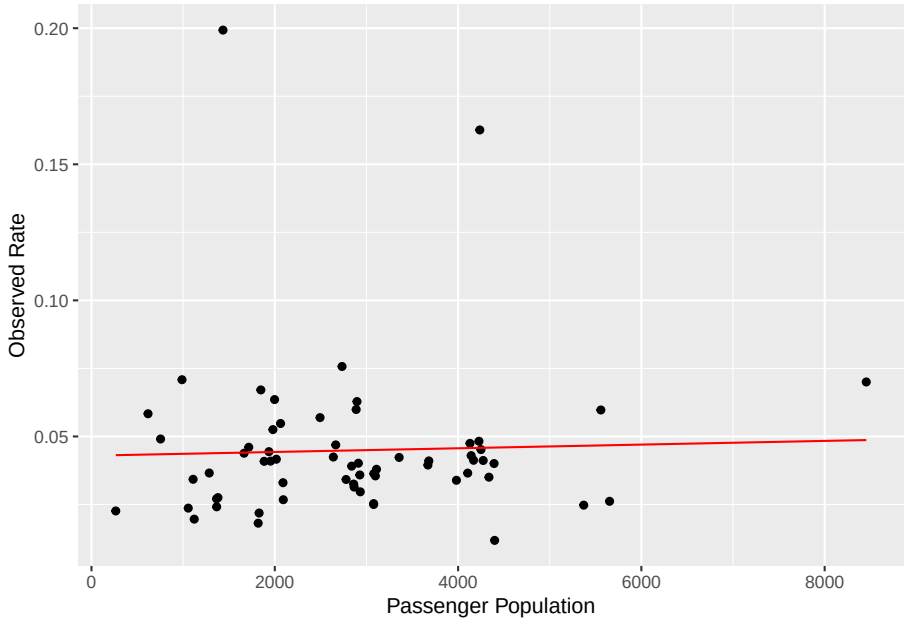
We'll create a data frame for our model's predictions of rate, declaring an empty column for that prediction and placing in our reported rate as well as our voyage duration as additional columns.

```
df.predm1 <- data.frame(yhat_rate = NA,
                        actual_rate = cruisedata.df$infection,
                        size = cruisedata.df$size)

df.predm1$yhat_rate <- predict(m1,newdata=df.predm1)

head(df.predm1)
```

```
##    yhat_rate actual_rate size
## 1 0.04577220  0.04125690 4169
## 2 0.04588604  0.03504727 4337
## 3 0.04493127  0.03586066 2928
## 4 0.04492043  0.04017857 2912
## 5 0.04407544  0.04384384 1665
## 6 0.04419945  0.06709957 1848
```

We'll use `ggplot` to visualize the model on a line plot, with the X-axis being our voyage duration, the Y-axis being our observed rates, and the rate predictions of the model shown as a red line.

```
ggplot(data = df.predm1, aes(x=size, y=actual_rate)) +
  labs(x="Passenger Population", y="Observed Rate") +
  geom_point() +
  geom_line(aes(y=yhat_rate), color='red')
```



Binomial Regression

The `glm` function in the `stats` R package is a simple yet effective tool for this next model.

Using the following framework for our model:

$$Data\ Model : [z = y] Process\ Model : y_{ij} \sim Binom(n_i, \phi_i) Parameter\ Model : [\phi_i \sim logit(\phi_i) = \beta_0 + \beta_1 * X + \eta_s + \eta_t \phi_s \quad (2)$$

We'll use our intercept of size but as a category this time, then add in our voyage duration, latitude, and longitude. We set the `family="quasibinomial"` since it creates the same effect as `family="binomial"`, but avoids a repetitive error code.

```
m2 <- glm(infection ~ size.cat + total_days + cruisedata.df$Start_lat + cruisedata.df$

summary(m2)
```

```
##
## Call:
## glm(formula = infection ~ size.cat + total_days + cruisedata.df$Start_lat +
##     cruisedata.df$Start_long, family = "quasibinomial", data = cruisedata.df)
##
## Coefficients:
##                            Estimate Std. Error t value Pr(>|t|)
## (Intercept)              -3.354517   0.448056  -7.487 6.01e-10 ***
## size.cat2                -0.066468   0.384998  -0.173   0.8636
## size.cat3                -0.138738   0.374774  -0.370   0.7127
## size.cat4                -0.148188   0.368990  -0.402   0.6895
## size.cat5                 0.070229   0.361981   0.194   0.8469
## total_days                0.032019   0.012450   2.572   0.0129 *
## cruisedata.df$Start_lat   0.004797   0.003678   1.304   0.1976
## cruisedata.df$Start_long  0.002612   0.001564   1.670   0.1006
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for quasibinomial family taken to be 0.01363758)
##
##     Null deviance: 0.84225  on 62  degrees of freedom
## Residual deviance: 0.64622  on 55  degrees of freedom
## AIC: NA
##
## Number of Fisher Scoring iterations: 6
```

Once again, a data frame for our prediction is built, with predicted rate, size categories, observed rate of infection, voyage duration, latitude, and longitude. Then we fill the empty prediction column with expected rates of infection.

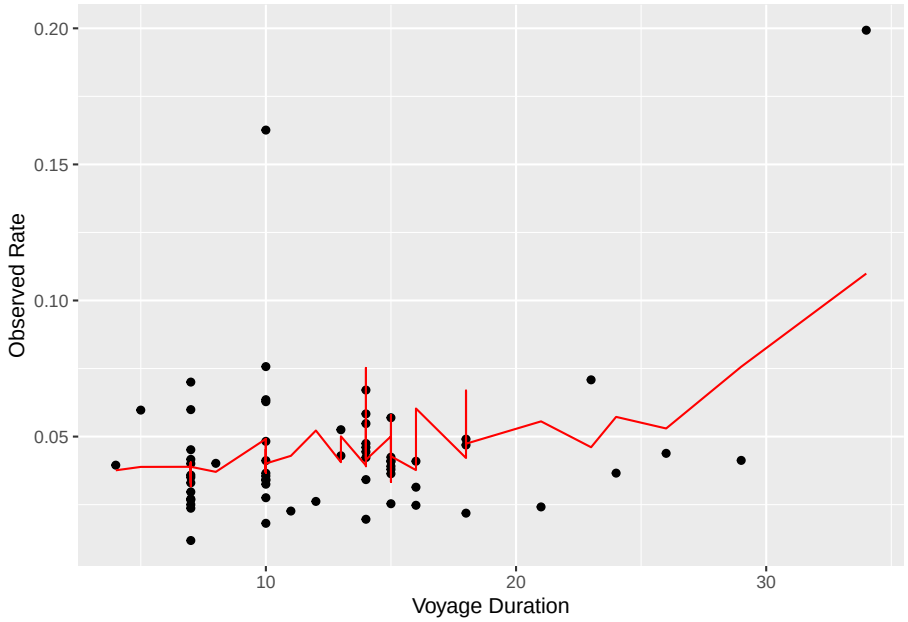
```
df.predm2 <- data.frame(yhat_rate = NA,
                        size.cat = cruisedata.df$size.cat,
                        actual_rate = cruisedata.df$infection,
                        total_days = cruisedata.df$total_days,
                        lat = cruisedata.df$Start_lat,
                        long = cruisedata.df$Start_long)

df.predm2$yhat_rate <- predict(m2,newdata=df.predm2)

df.predm2$yhat_rate <- 1 / (1 + exp(-df.predm2$yhat_rate))
```

```
head(df.predm2)
```

```
##    yhat_rate size.cat actual_rate total_days      lat      long
## 1 0.07566357        5  0.04125690    29 days  33.7405 -118.2786
## 2 0.03889552        5  0.03504727     7 days  33.7405 -118.2786
## 3 0.03590919        4  0.03586066     7 days  39.2876  -76.6108
## 4 0.03703431        4  0.04017857     8 days  39.2876  -76.6108
## 5 0.05298255        3  0.04384384    26 days -22.8955  -43.1822
## 6 0.03939088        3  0.06709957    14 days  32.7157 -117.1611
```

Plotting this the way we did before does not achieve much due to how many variables we're working with. The plot below details that concept, in my opinion, very well.

```
ggplot(data = df.predm2, aes(x=as.numeric(total_days), y=actual_rate)) +
  labs(x="Voyage Duration", y="Observed Rate") +
  geom_point() +
  geom_line(aes(y=yhat_rate), color='red')
```



Program Validation

Our data is organized, our models are fit. I find it appropriate to look at the data that's being returned by the code that's been written, to ensure everything is rational to the intent. This portion is a practice I have become fond of in my personal experience, but it is not necessary since it doesn't follow any statistical methodology

We'll observe the predictions our models have made.

```
##    yhat_rate actual_rate size
## 1 0.04577220  0.04125690 4169
## 2 0.04588604  0.03504727 4337
## 3 0.04493127  0.03586066 2928
## 4 0.04492043  0.04017857 2912
## 5 0.04407544  0.04384384 1665
## 6 0.04419945  0.06709957 1848
```

```
##    yhat_rate        actual_rate          size
##  Min.   :0.04313   Min.   :0.01182   Min.   : 265
##  1st Qu.:0.04419   1st Qu.:0.03195   1st Qu.:1838
##  Median :0.04487   Median :0.04018   Median :2839
##  Mean   :0.04486   Mean   :0.04486   Mean   :2824
##  3rd Qu.:0.04554   3rd Qu.:0.04785   3rd Qu.:3832
##  Max.   :0.04868   Max.   :0.19930   Max.   :8454
```

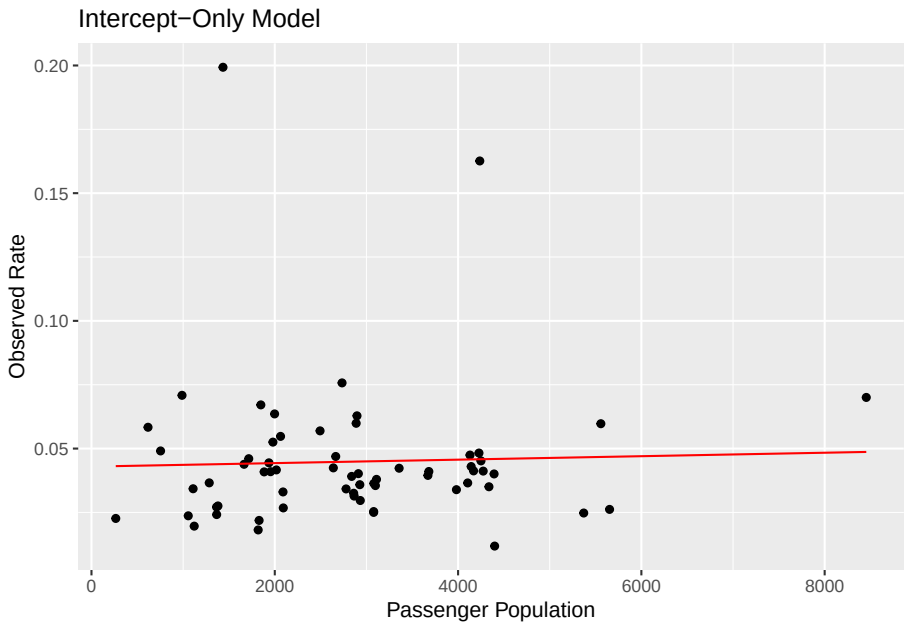
```
##    yhat_rate size.cat actual_rate total_days      lat      long
## 1 0.07566357        5  0.04125690    29 days  33.7405 -118.2786
## 2 0.03889552        5  0.03504727     7 days  33.7405 -118.2786
## 3 0.03590919        4  0.03586066     7 days  39.2876  -76.6108
## 4 0.03703431        4  0.04017857     8 days  39.2876  -76.6108
## 5 0.05298255        3  0.04384384    26 days -22.8955  -43.1822
## 6 0.03939088        3  0.06709957    14 days  32.7157 -117.1611
```

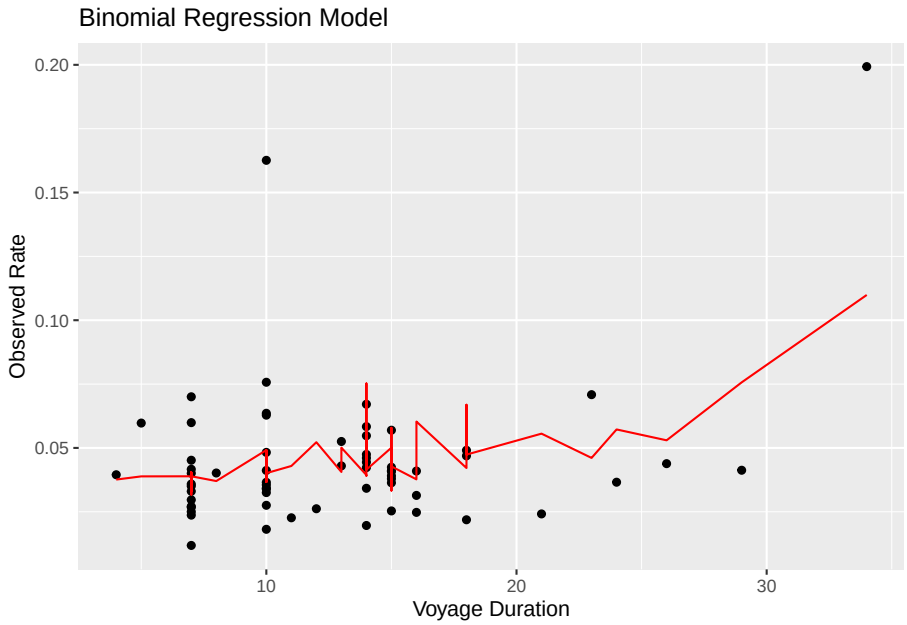
```
##    yhat_rate       size.cat  actual_rate       total_days
##  Min.   :0.03173   1: 3     Min.   :0.01182   Length:63
##  1st Qu.:0.03732   2: 9     1st Qu.:0.03195   Class :difftime
##  Median :0.04148   3:15     Median :0.04018   Mode  :numeric
##  Mean   :0.04486   4:18     Mean   :0.04486
##  3rd Qu.:0.04827   5:18     3rd Qu.:0.04785
##  Max.   :0.10990            Max.   :0.19930
##       lat              long
##  Min.   :-33.05   Min.   :-149.44
##  1st Qu.: 25.94   1st Qu.:-117.16
##  Median : 32.72   Median : -80.19
```

```
##  Mean   : 29.22   Mean   : -82.01
##  3rd Qu.: 40.69   3rd Qu.: -75.36
##  Max.   : 60.10   Max.   : 139.69
```

For each model, the results of our predicted rates are rational to the models themselves. Low variance on model 1 is to be expected, model 2 looks appropriate given the syntax it's constructed from.

Visualizing it in our plots may not be inherently useful to the analysis of our data, but I've learned that it's exceptionally good at parsing out any errors that might have occured in our code blocks.





The plots don't return any errors, and our lines of fit coincide with what the data frames have been telling us. Interpreting them isn't the current goal, so whether or not the second model is "over-fit" isn't relevant at the moment.

Troubleshooting

The majority of errors you'll experience during this process with come from two sources:

Out of order set-up

As stated previously, if you don't build your code in the order it has been presented so far, you will encounter continuous errors. In the event you make this error, clear the environment. Rebuild your variables and data frames. Double check you have followed the correct path laid out by this tutorial.

Incorrect specification of file pathing

There are 5 files that have been provided through dropbox URLs. If you:

1) Have not downloaded them.

2) Have not placed them into a folder your environment can path to.

3) Have not deleted the file paths in the code provided so far and re-specified the appropriate paths for your system.

You will encounter errors that won't resolve until you complete those three steps.

Remember the golden rules of troubleshooting all technology:

> Power cycle then repeat the fault
>
> When in doubt, check for updates
>
> If all else fails, Google the error message

Fully power cycling a device or program is not only definitively and objectively a mystical solution to 99% of all technological errors the instant the IT support tech shows up, it's also essential in the process of "repeat the fault". If you can figure out how to make the error repeatedly, you can work to locate the source.

Updates fix programs and hardware while breaking other things in the process, further updates fix them and break different aspects than the previous one. Failing to maintain updates keeps things broken forever.

Google is free. People become Chess Grand masters with free online tools they access from Google.

Part IV

Model Evaluation

Model Checking

Checking the predictive accuracy of our model is the next step in this process. There are numerous methods for performing model checks and their efficacy varies based on the data/model/function being used.

For this project and tutorial we'll be using 4 separate methods:

- Regression Diagnostics
- Graphical Diagnostics
- Within-Sample Validation
- Forecasting

Regression Diagnostics

Graphical Diagnostics

Within-Sample Validation

Forecasting