Weather Patterns X COVID-19

Final Project Documentation

Uhuru Kamau, Shuyang Lu, Yifan Zhao, Chenhao Zhao, Zhaofeng Liu

Contents

```r
# Load Required Packages
library(tidyverse)
library(kableExtra)
library(readr)
library(gridExtra)
library(knitr)
library(devtools)
library(usethis)
library(roxygen2)
library(testthat)
library(devtools)
library(rmarkdown)
library(PerformanceAnalytics)
library(lubridate)
library(reshape2)
library(timeSeries)
library(forecast)
library(tseries)
library(TSA)
```

```r
library(gsubfn)
library(proto)
# library(sqldf)
```

Project Workflow

Right Side of the Schema

```
knitr::include_graphics(path = "images/QBS181_1.png")
```

Figure 1: QBS181 Final Project: WorkFlow (Part A)

Left Side of the Schema

```r
knitr::include_graphics(path = "images/QBS181_2.png")
```

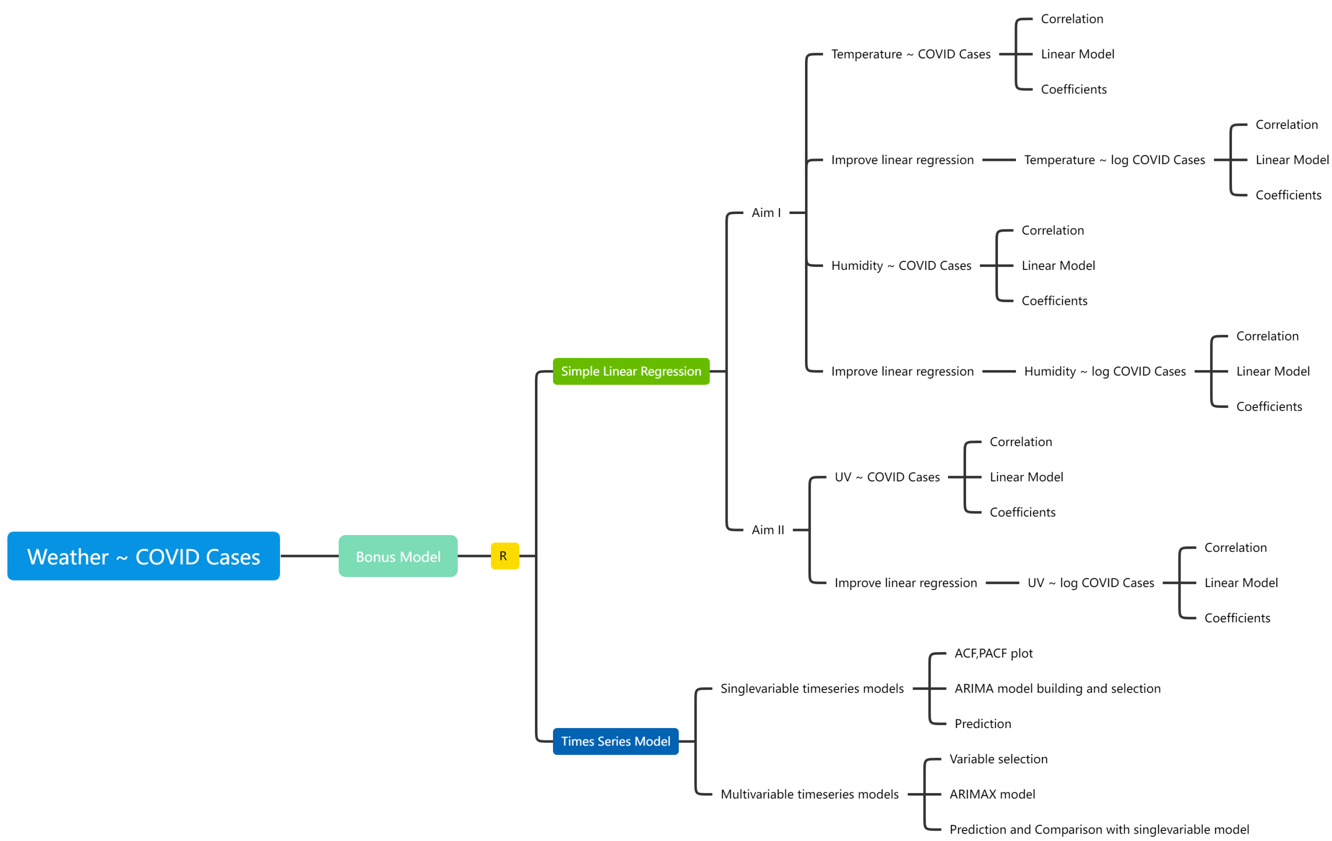


Figure 2: QBS181 Final Project: WorkFlow (Part B)

Data Acquisition

1. New York City COVID-19 Data Archive

- Source: NYC OpenData
- Acquisation Method
  - Download .csv file
- Purpose:
  - We will use this time series data to track changes in the incidence of COVID-19.

2. New York City Weather Data

- Source: Weather Underground - Weather Archive
- Acquisition Method
  - Webscraping/ API Tool
- Purpose:
  - Merge time series weather data with timeseries Covid-19 data and investigate potential associations

3. Daily UV Index Scores - New York City

- Source: Central New York's Live Weather Source
- Acquisition Method
  - UV index values are presented as tables (see figure)
  - Copy tables and paste into Microsoft Excel
  - Save as .csv file
- Purpose
  - Sunlight and Vitamin-D absorbtion
    * It is generally accepted that there is a positive association between exposure to sunlight and absorbtion of vitamin-D.
    * It is also generally accepted that there is a positive association between vitamin-D absorbtion and immune system capacity.
  - We will us UV-Index as a proxy for exposure to sunlight at the population level and test for associations between UV Index and the incidence of Covid-19.

Relational Schema

```
knitr::include_graphics(path = "images/Relational_Schema.png")
```

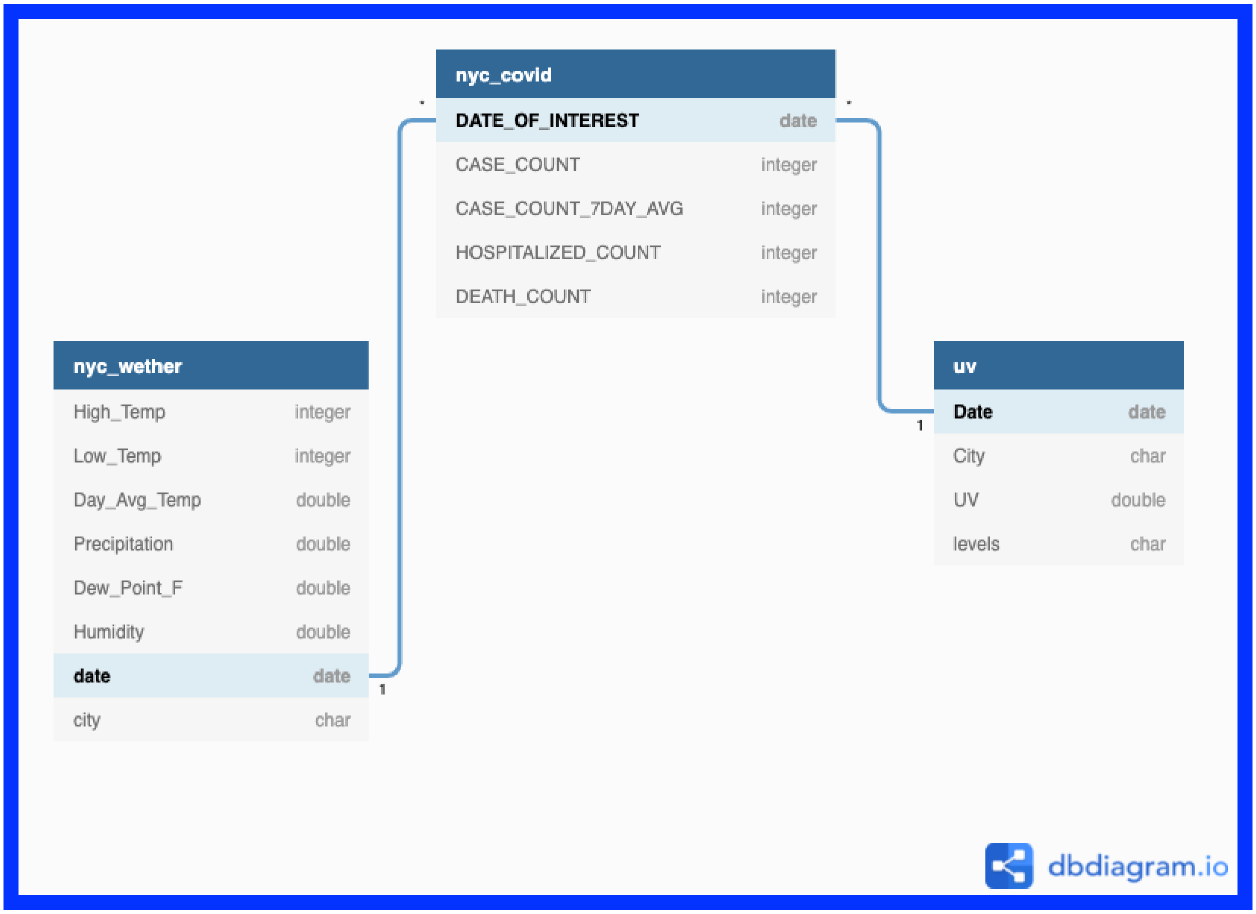


Figure 3: Highlighting the Keys to our Relational Database

Data Cleaning

1. New York City COVID-19 Data Archive

- Step 1: Read in the Covid-19 Date Frame

```
covid_df <- read.csv("data/Raw Data/nyc_covid19_data/NYC_Covid_Data_raw.csv")
```

2. New York City Weather Data

- Step 1: Read-in File from Raw Data File
  - The raw file has an issue with the column headers.
    - Several Headers include symbols that don't work with the interpretor
      - eg. $Low\_Temp(^oF)$, $High\_Temp(^oF)$
  - Solution: Update column names while reading in the file!

```
# vector with acceptable column names
headers <- c("High.Temp", "Low.Temp", "Avg.Temp", "Precip", "Dew.Point", "Humidity",
    "Date", "City")

# read in the raw data file
weather.raw <- read.csv(file = "data/Raw Data/nyc_weather_raw.csv", header = TRUE,
    col.names = headers)
```

- Step 2: Format all observations of the "Date" Variable
  - Variable is of class "character" by default

```
class(weather.raw$Date)
```

```
RESULT [1] "character"
```

- Reclassify the variable as a "Date"

```
weather.clean <- weather.raw %>%
    mutate(Date = as.Date(Date, "%m/%d/%Y"))
```

- Outcome:

```
class(weather.clean$Date)
```

```
RESULT [1] "Date"
```

- Step 3: Missing Data
  - Since we intend to do a time series, we need to identify any missing dates in the "date" column.
    - We will do this using a CUSTOM FUNCTION!

```r
# Custom function to find the missing date in the date column
find.missing.dates <- function(d) {
    date_range <- seq(min(d), max(d), by = 1)
    date_range[!date_range %in% d]
}
```

- Use the custom function to identify missing dates in our NYC Weather df.

```r
# Display the missing dates
date.missing = c()
date.missing <- find.missing.dates(weather.clean$Date)
print(date.missing)
```

```
RESULT [1] "2020-11-08"
```

- Step 4: Replace Missing Values
  - Method: Fill the missing data by averaging the former 6 days' data

```r
# Find the index of the day before '2020-11-08'
weather.clean$Date <- as.character(weather.clean$Date)
id.missing.date = which(weather.clean$Date == "2020-11-07") + 1
```

- Build a custom function to fill the missing data
  - Approach: use the average of the previous six days

```r
# Custom function to fill the missing data by averaging the former 6 days' data
fill.missing.values <- function(df, newrow.id) {
    newrow <- list()
    value <- c()
    first.row = newrow.id - 6
    last.row = newrow.id - 1
    col.num = ncol(df) - 2
    for (i in 1:col.num) {
        subs <- weather.clean[first.row:last.row, i]  # Create a new subset for each column
        value <- mean(subs)  # Calculate the mean
        newrow <- append(newrow, value)
    }
    return(newrow)
}
```

- Use the custom function to fill in the values of the missing row

```r
# Fill the missing values in the missing row
missing.row <- fill.missing.values(weather.clean, id.missing.date)
missing.row <- append(missing.row, "2020-11-08")
missing.row <- append(missing.row, "new york city")
```

- Build another custom function to insert the row into the df

```r
# Custom function to insert the new row
insertRow <- function(existingDF, newrow, r) {
    existingDF[seq(r + 1, nrow(existingDF) + 1), ] <- existingDF[seq(r, nrow(existingDF)),
        ]
    existingDF[r, ] <- newrow
    existingDF
}
```

- Insert the imputed value into the df!

```r
# Insert the missing row and store it into a new df
weather.clean <- insertRow(weather.clean, missing.row, id.missing.date)
```

- Step 4: Remove the "City" Variable
  - Every observation is is "new york city"
    - This variable is effectively just clutter.

```r
weather.clean = weather.raw %>%
    select(-City)
```

- Step 5: Display

```r
kable(x = weather.clean[1:5, ], digits = 2, align = "c")
```

High.Temp	Low.Temp	Avg.Temp	Precip	Dew.Point	Humidity	Date
44	26	35.46	0.00	13.67	41.83	3/1/2020
56	38	48.17	0.00	30.46	51.12	3/2/2020
58	48	52.41	0.01	44.59	75.47	3/3/2020
57	46	50.52	0.28	28.52	44.76	3/4/2020
52	40	44.75	0.00	25.38	48.50	3/5/2020

- Step 6: Write the Processed Data to a new .csv file

```r
write.csv(x = weather.clean, file = "data/Processed Data/nyc_weather.csv")
```

- Step 7: In Excel - Add a "Season" variable to the .csv generated in the prior section
  - Open File in Microsoft Excel
    - Summarise the process
  - Save the file as "nyc_clean_weather_add_season.csv"
  - Push to Github
- Step 8: Read in "nyc_clean_weather_add_season.csv" and reformat the "date" variable
  - Read in the file

```r
# vector with acceptable column names
headers <- c("High.Temp", "Low.Temp", "Avg.Temp", "Precip", "Dew.Point", "Humidity",
    "date", "City", "Month", "season")
add_season_nyc_weather <- read.csv(file = "data/nyc_clean_weather_add_season.csv",
    header = T, col.names = headers) %>%
    select(-City)
```

- Reformat the "date" variable

```r
# change date type from ymd to mdy
add_season_nyc_weather$date <- format(as.Date(add_season_nyc_weather$date, "%Y/%m/%d"),
    "%m/%d/%Y")
```

- Step 9: Display

```r
knitr::kable(x = add_season_nyc_weather[1:5, ], align = "c")
```

High.Temp	Low.Temp	Avg.Temp	Precip	Dew.Point	Humidity	date	Month	season
44	26	35.46	0.00	13.67	41.83333	03/01/2020	3	spring
56	38	48.17	0.00	30.46	51.12500	03/02/2020	3	spring
58	48	52.41	0.01	44.59	75.47059	03/03/2020	3	spring
57	46	50.52	0.28	28.52	44.76000	03/04/2020	3	spring
52	40	44.75	0.00	25.38	48.50000	03/05/2020	3	spring

3. Daily UV Index Scores - New York City

- Step 1: Read the 2020 csv file
  - Read the csv file from the website we collected of year 2020 ultraviolet rays as "uv" for each day in a year.

```r
# read the csv
uv_2020 <- read.csv("data/2020uv.csv")
```

- Step 2: Use the "reshape2" library to melt the data we collect to 3 columns which is day, month, uv

```r
# wide-format change
new_2020uv <- as.data.frame(melt(uv_2020, id.vars = c("Day")))

# change column names
colnames(new_2020uv)[2] <- "Month"
colnames(new_2020uv)[3] <- "UV"
```

- Step 3: Do the missing data correction, because Feburary only has 28 days, we need get rid of the "NA" in those certain dates.

```r
# delete February
test_2020 <- new_2020uv %>%
    dplyr::filter(!is.na(UV))

# add year column

test_2020 <- mutate(test_2020, Year = 2020)
```

- Step 4: # Use the function to convert month to number and then merge the day, month, year together to generate a new column date.

```
# function to change month to number
numMonth <- function(x) {
    months <- list(jan = 1, feb = 2, mar = 3, apr = 4, may = 5, jun = 6, jul = 7,
        aug = 8, sep = 9, oct = 10, nov = 11, dec = 12)
    x <- tolower(x)
    sapply(x, function(x) months[[x]])
}
# new column numeric month
test_2020$Month_change = numMonth(test_2020$Month)
# past day-month-year together
test_2020$date = paste(test_2020$Month_change, test_2020$Day, test_2020$Year, sep = "/")
```

- Step 5: Use the "lubridate" library to make the date to date that can be recognized by R

```
# make the 2020 date as order we want
test_2020$new_date = mdy(test_2020$date)
```

- Step 6: Import 2021 csv file

```
# import 2021 UV data
uv_2021 <- read.csv("data/2021uv.csv")
```

- Step 7: Use the reshape2 library to melt the data we collect to 3 columns which is day, month, uv

```
# change the first column name
colnames(uv_2021)[1] = "Day"
# wide-format change
new_2021uv <- as.data.frame(melt(uv_2021, id.vars = c("Day")))
# change column names
colnames(new_2021uv)[2] <- "Month"
colnames(new_2021uv)[3] <- "UV"
```

- Step 8: Do the missing data correction, because Februrary only has 28 days, we need get rid of the "NA" in those certain dates.

```
# delete February
test_2021 <- new_2021uv %>%
    dplyr::filter(!is.na(UV))

# add year new column
test_2021 <- mutate(test_2021, Year = 2021)
```

- Step 9: Use the function to convert month to number and then merge the day, month, year together to generate a new column date.

```r
# new column numeric month
test_2021$Month_change = numMonth(test_2021$Month)
# past day-month-year together
test_2021$date = paste(test_2021$Month_change, test_2021$Day, test_2021$Year, sep = "/")
```

```r
# make the date accessible
test_2021$new_date = mdy(test_2021$date)
```

```
RESULT Warning: 1 failed to parse.
```

```r
# fail reason for 11/31/2021: no UV data
```

- Step 10: Bind the data from 2020 and the data from 2021

```r
# r bind the two year together
uv_df = rbind(test_2020, test_2021)
```

- Step 11: Add the city name for each of the date and related UV

```r
# select the columns we need
uv_final = select(uv_df, new_date, UV)
# add a new column for city name
uv_final <- mutate(uv_final, City = "New York")
```

- Step 12: Filter the period of time period we needed for our project.

```r
# take out the period we want from 2020-03-01 to 2021-10-31
uv_final$new_date = as.Date(uv_final$new_date, format = "%Y-%m-%d")
uv_select = subset(uv_final, new_date > "2020-02-29" & new_date < "2021-11-01")
```

- Step 13: Preview the Data Frame.

```r
uv_final.display <- uv_final[1:5, ]
knitr::kable(uv_final.display)
```

new_date	UV	City
2020-01-01	0.7	New York
2020-01-02	1.3	New York
2020-01-03	0.9	New York
2020-01-04	0.7	New York
2020-01-05	0.8	New York

AIM 1

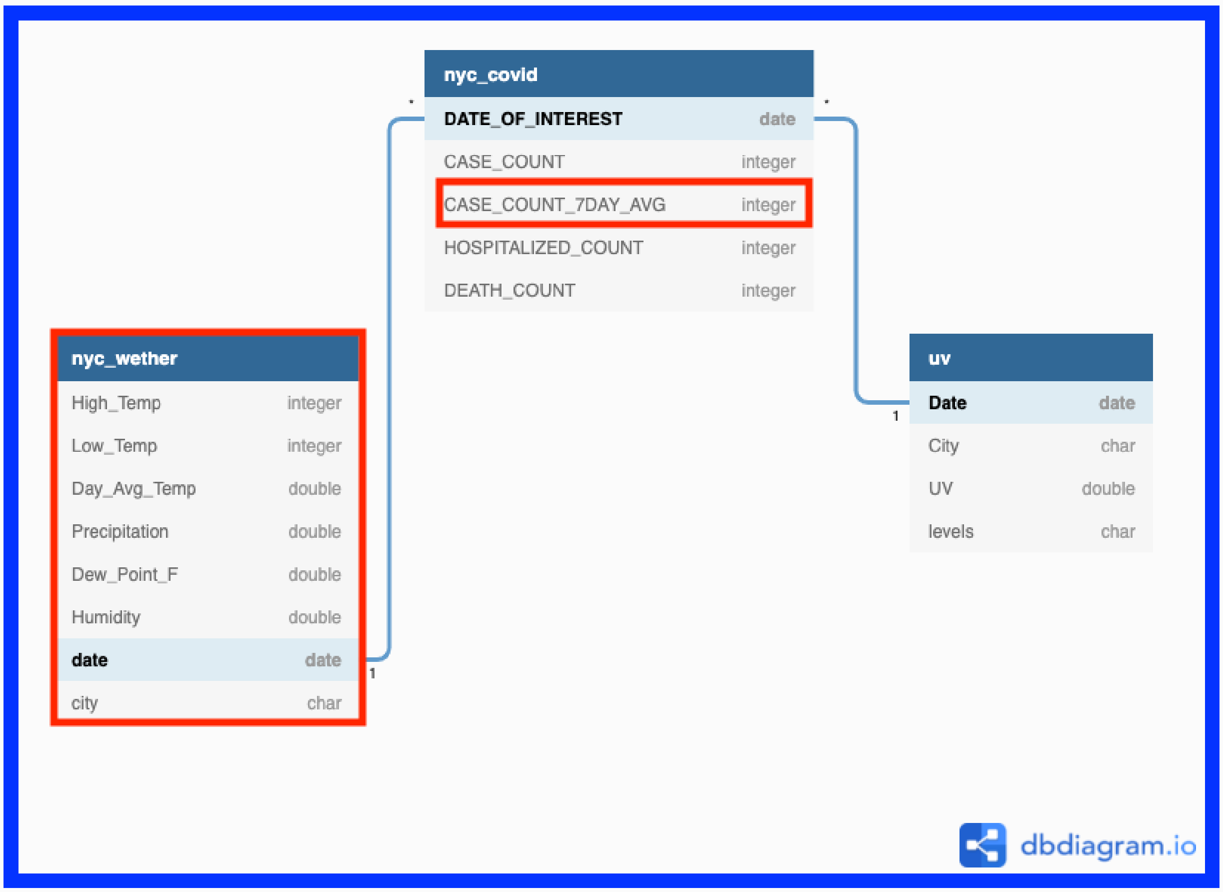
1. Is there a Difference in Number of Cases Observed in the Summer vs in the Winter?

- Step 1: Add the "CASE_COUNT_7DAY_AVG" variable from the covid df to the add_season_nyc_weather df

– Visual Aid

```
knitr::include_graphics(path = "images/weather_and_7day_avg.png")
```

\begin{figure}

{

}

\caption{CASE_COUNT_7DAY_AVG variable from the covid df to the add_season_nyc_weather df}
\end{figure}

- Code

```
add_season_test <- add_season_nyc_weather
rownames(covid_df) <- covid_df$DATE_OF_INTEREST
add_season_test$Case_Count_7Day_Avg <- covid_df[, "CASE_COUNT_7DAY_AVG"]
```

- Step 2: Create a df with ONLY the summer and winter observations
  - Code

```
sum_win <- add_season_test %>%
    dplyr::filter(season == "summer" | season == "winter")
```

- Step 3: Run a Wilcox Test to test the difference in COVID incidence (summer vs winter)
  - Code

```
wilcox.test(sum_win[which(sum_win$season == "summer"), ]$Case_Count_7Day_Avg, sum_win[which(sum_win$seas
    "winter"), ]$Case_Count_7Day_Avg)
```

```
RESULT
RESULT  Wilcoxon rank sum test with continuity correction
RESULT
RESULT data:  sum_win[which(sum_win$season == "summer"), ]$Case_Count_7Day_Avg and sum_win[which(sum_wi
RESULT W = 0, p-value < 2.2e-16
RESULT alternative hypothesis: true location shift is not equal to 0
```

- We can reject the null hypothesis that there is no difference in incidence.

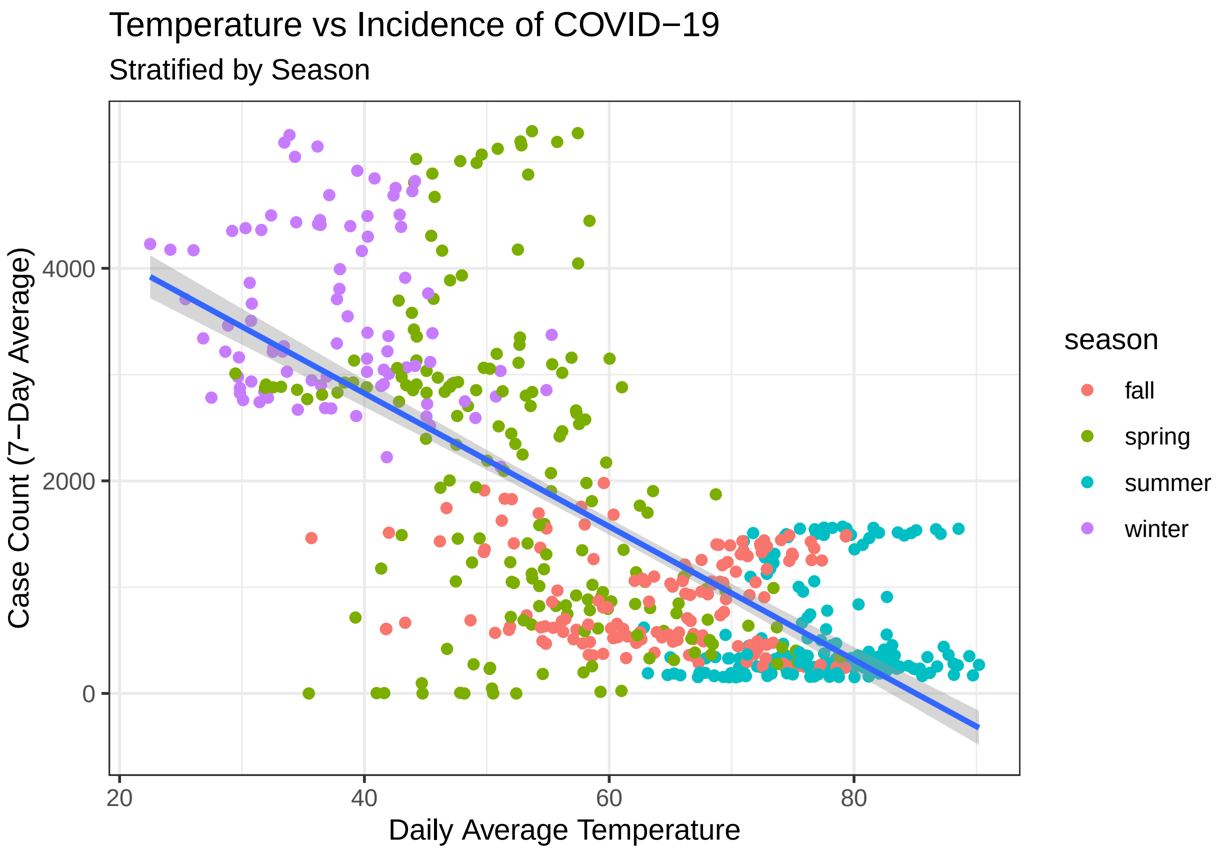
2. Is there an Association between temperature and Incidence?

- Step 1: Generate a Scatter Plot

```
ggplot(data = add_season_test) + geom_point(mapping = aes(x = Avg.Temp, y = Case_Count_7Day_Avg,
    color = season)) + labs(x = "Daily Average Temperature", y = "Case Count (7-Day Average)",
    title = "Temperature vs Incidence of COVID-19", subtitle = "Stratified by Season") +
    theme_bw() + geom_smooth(mapping = aes(x = Avg.Temp, y = Case_Count_7Day_Avg),
    method = "lm", inherits.aes = F)
```

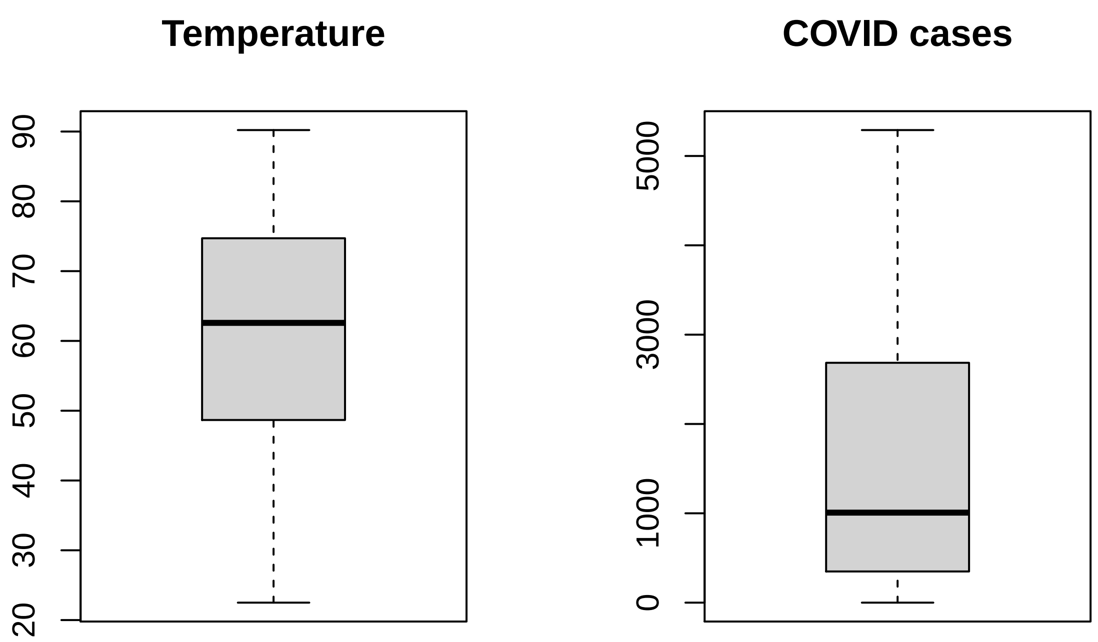
```
RESULT Warning: Ignoring unknown parameters: inherits.aes
```

```
RESULT `geom_smooth()` using formula 'y ~ x'
```



- Step 2: Generate a Boxplot

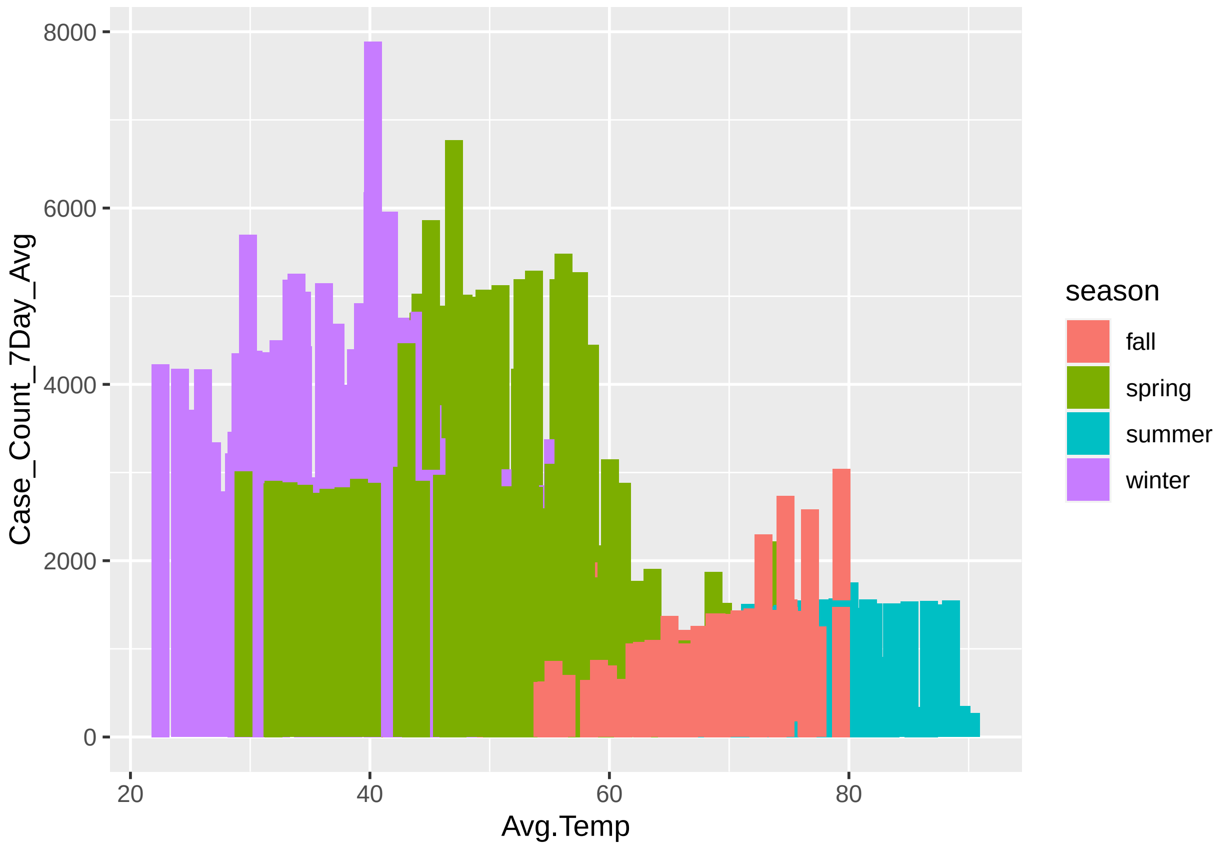
```
par(mfrow = c(1, 2))  # divide graph area in 2 columns
boxplot(add_season_test$Avg.Temp, main = "Temperature")
boxplot(add_season_test$Case_Count_7Day_Avg, main = "COVID cases")
```



- Step 3: Generate a Bar Graph

```
ggplot(data = add_season_test) + geom_col(mapping = aes(x = Avg.Temp, y = Case_Count_7Day_Avg,
    fill = season), width = 1.5)
```

```
RESULT Warning: position_stack requires non-overlapping x intervals
```



- Step 4: Compute the Correlation between Temperature and Case Count

```
cor(add_season_test$Avg.Temp, add_season_test$Case_Count_7Day_Avg)
```

```
RESULT [1] -0.7175508
```

- Step 5: Generate a Single Variable Linear Regression Model

  +Temperature ~ Cases

```
linearMod <- lm(add_season_test$Avg.Temp ~ add_season_test$Case_Count_7Day_Avg, data = add_season_test)
print(linearMod)
```

```
RESULT
RESULT Call:
RESULT lm(formula = add_season_test$Avg.Temp ~ add_season_test$Case_Count_7Day_Avg,
RESULT     data = add_season_test)
```

```
RESULT
RESULT Coefficients:
RESULT                       (Intercept)  add_season_test$Case_Count_7Day_Avg
RESULT                         73.416364                            -0.008214
```

- Step 6: View Linear Model Coefficients

```
summary(linearMod)$coefficients
```

```
RESULT                                        Estimate   Std. Error   t value
RESULT (Intercept)                         73.416363791 0.6608480427 111.09417
RESULT add_season_test$Case_Count_7Day_Avg -0.008214267 0.0003233623 -25.40268
RESULT                                         Pr(>|t|)
RESULT (Intercept)                         0.000000e+00
RESULT add_season_test$Case_Count_7Day_Avg 1.417949e-97
```

- Step 7: Improve Linear Model By using the Log incident case values

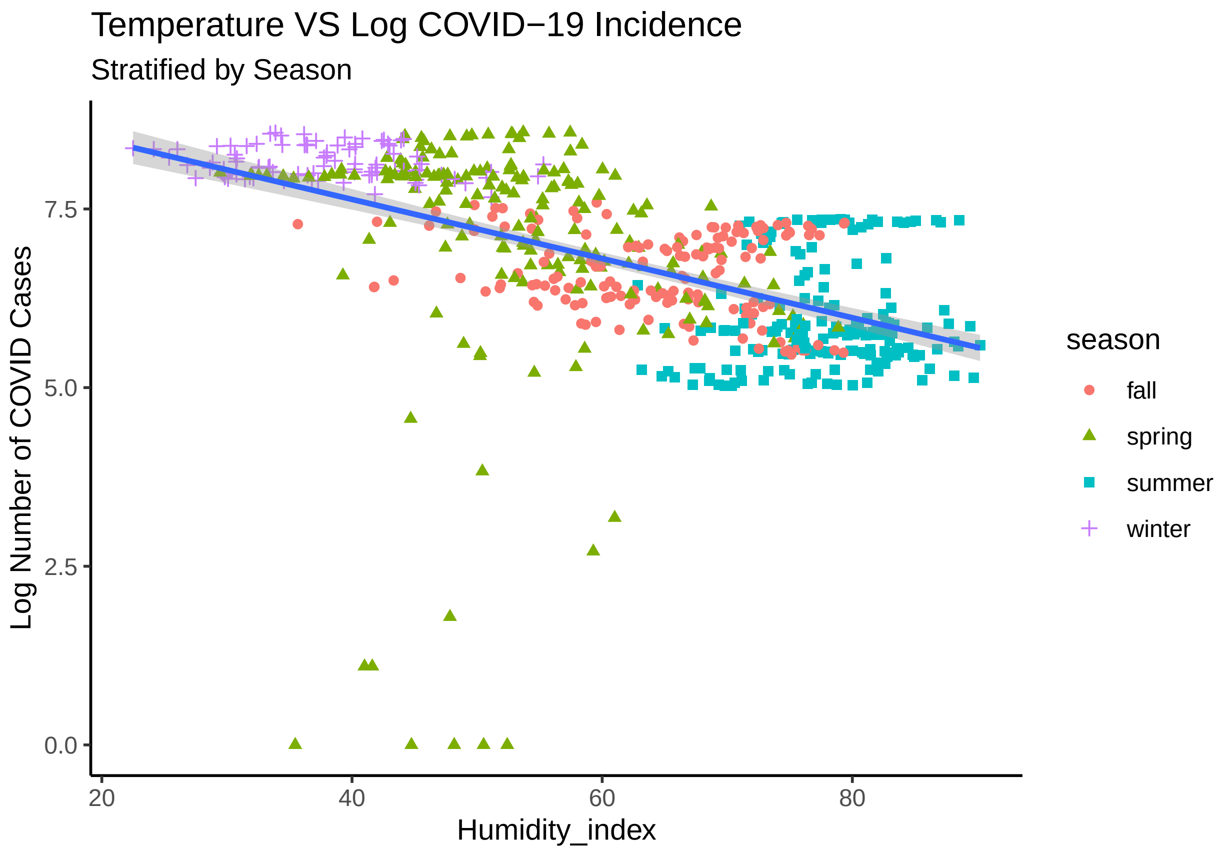
```
# log cases
add_season_test$log_cases <- log(add_season_test$Case_Count_7Day_Avg)
for (i in 1:nrow(add_season_test)) {
    if (add_season_test$Case_Count_7Day_Avg[i] == 0) {
        add_season_test$log_cases[i] = 0
    }
}
```

- Generate a Scatter Plot Using
  - Temperature vs Log cases

```
## test log cases
ggplot(data = add_season_test) + geom_point(mapping = aes(x = Avg.Temp, y = log_cases,
    color = season, shape = season)) + labs(x = "Humidity_index", y = "Log Number of COVID Cases",
    title = "Temperature VS Log COVID-19 Incidence", subtitle = "Stratified by Season") +
    theme_classic() + geom_smooth(mapping = aes(x = Avg.Temp, y = log_cases), method = "lm",
    inherits.aes = F)
```

```
RESULT Warning: Ignoring unknown parameters: inherits.aes
```

```
RESULT `geom_smooth()` using formula 'y ~ x'
```



- Step 7: Generate a Single Variable Linear Regression Model
  - Temperature Vs Log Cases

```
# linear model of humidity_index
linearMod <- lm(add_season_test$Avg.Temp ~ add_season_test$log_cases, data = add_season_test)  # build
print(linearMod)
```

```
RESULT
RESULT Call:
RESULT lm(formula = add_season_test$Avg.Temp ~ add_season_test$log_cases,
RESULT     data = add_season_test)
RESULT
RESULT Coefficients:
RESULT               (Intercept)  add_season_test$log_cases
RESULT                   104.263                     -6.388
```

- Step 8: View Coefficients for Linear Model (P-Value)
  - Temp vs New Log Cases

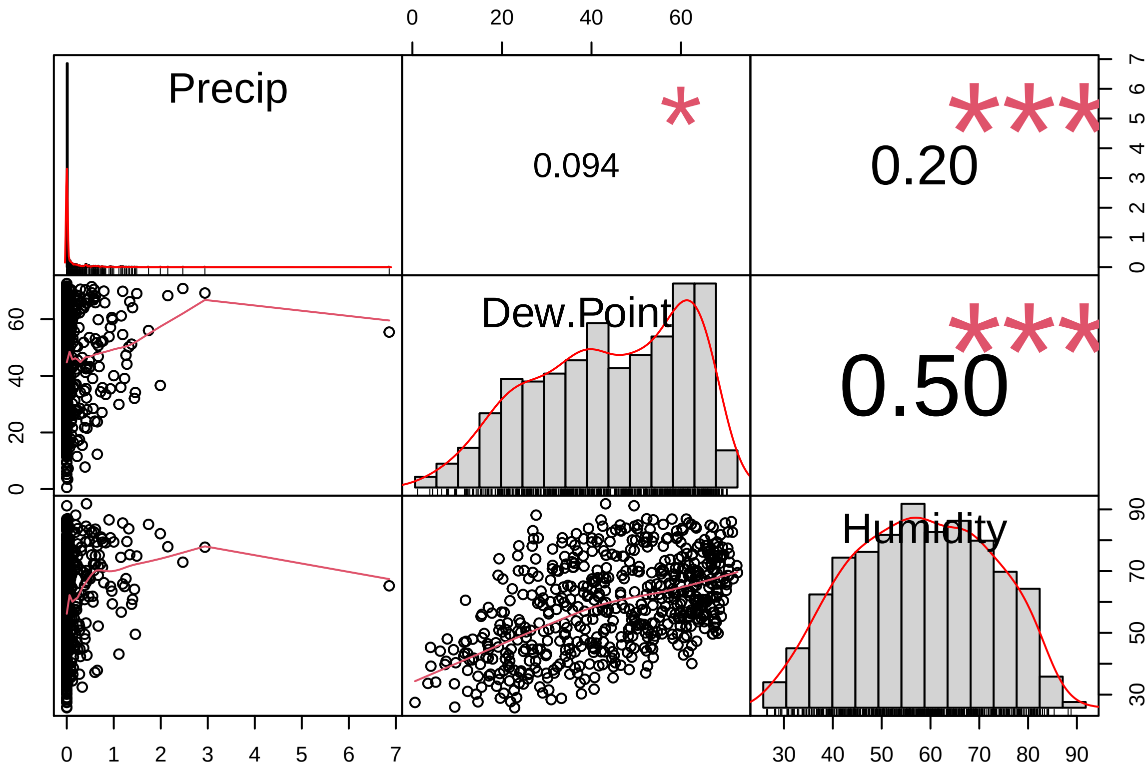
```
# coefficients test
summary(linearMod)$coefficients
```

```
RESULT                             Estimate Std. Error   t value      Pr(>|t|)
RESULT (Intercept)               104.263193  2.9725187  35.07571 3.314306e-148
RESULT add_season_test$log_cases  -6.387879  0.4319899 -14.78710  1.722615e-42
```

3. Is there an Association between Humidity Index and Cases?

- Step 1: Generate a Correlation Chart.

```
humidity_subset = add_season_test[, c(4, 5, 6)]
chart.Correlation(humidity_subset, histogram = TRUE, pch = 19)
```



- Step 2: Generate a "Humidity Index" variable
  - We use the following formula:
    * $Humidity\ Index = 0.01 \times (Precipitation + Dew\ Point + Humidity)$

```
# create humidity index column
add_season_test$humidity_index <- add_season_test$Precip + add_season_test$Dew.Point +
    add_season_test$Humidity * 0.01
# because humidity is a percentage, so we use humidity*0.01
```

- Step 3: Find missing observations of the humidity variable

```
# finding missing value
na_humidity <- c(which(is.na(add_season_test$humidity_index)))
na_humidity
```

```
RESULT [1]  65  67  91 110 265 270 325 489 590
```

- Step 4: Impute the missing values

- Imputation Formula:
    * Missing Value = Mean of the previous two observations

```
# fill the missing value-- 2 days mean
for (i in na_humidity) {
    add_season_test$Humidity[i] <- mean(add_season_test$Humidity[(i - 2):(i - 1)],
        2)
}
```

- Step 5: Fill in the imputed values at their corresponding positions

```
# refill the humidity index again after fill the missing value
add_season_test$humidity_index <- add_season_test$Precip + add_season_test$Dew.Point +
    add_season_test$Humidity * 0.01
```

- Step 6: Confirm that there are no more missing observations

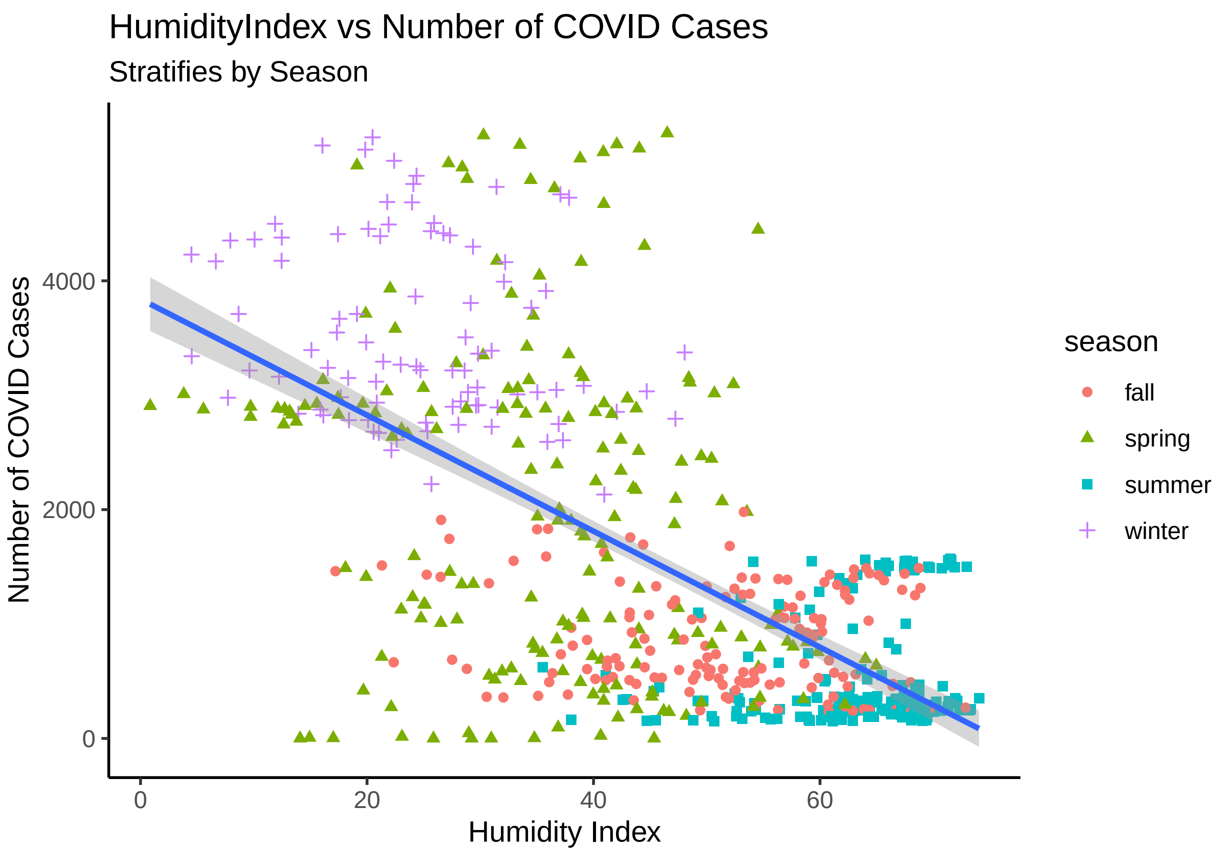
```
# check whether has missing value this time
count(add_season_test[is.na(add_season_test$humidity_index), ])
```

```
RESULT   n
RESULT 1 0
```

- Step 7: Generate a Scatter Plot
    - Humidity vs Cases

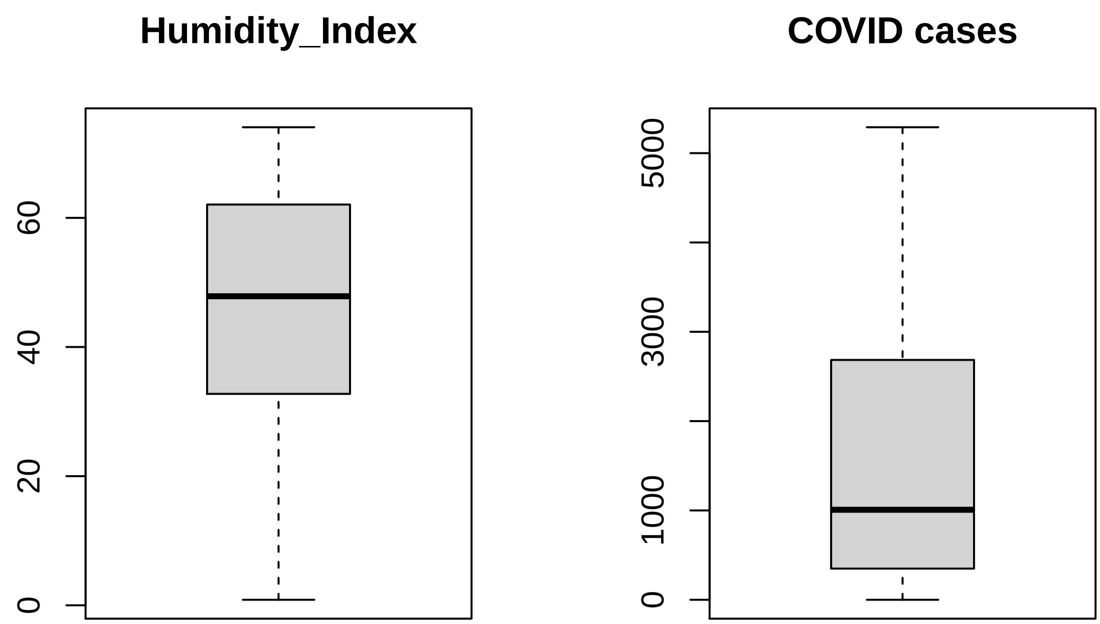
```
# scatter plot of humidity index

ggplot(data = add_season_test) + geom_point(mapping = aes(x = humidity_index, y = Case_Count_7Day_Avg,
    color = season, shape = season)) + labs(x = "Humidity Index", y = "Number of COVID Cases",
    title = "HumidityIndex vs Number of COVID Cases", subtitle = "Stratifies by Season") +
    theme_classic() + geom_smooth(mapping = aes(x = humidity_index, y = Case_Count_7Day_Avg),
    method = "lm", inherit.aes = F)
```

```
RESULT `geom_smooth()` using formula 'y ~ x'
```



- Step 8: Generate a Boxplot
  - Humidity vs Cases

```
# boxplot
par(mfrow = c(1, 2))  # divide graph area in 2 columns
boxplot(add_season_test$humidity_index, main = "Humidity_Index")
boxplot(add_season_test$Case_Count_7Day_Avg, main = "COVID cases")
```



- Step 9: Compute the Correlation Coefficient

```
# correlation
cor(add_season_test$humidity_index, add_season_test$Case_Count_7Day_Avg)
```

```
RESULT [1] -0.639719
```

- Step 10: Generate a Single Variable Linear Model
  - Humidity vs Cases

```
# linear model of humidity_index
linearMod <- lm(add_season_test$humidity_index ~ add_season_test$Case_Count_7Day_Avg,
    data = add_season_test)  # build linear regression model on full data
print(linearMod)
```

```
RESULT
RESULT Call:
RESULT lm(formula = add_season_test$humidity_index ~ add_season_test$Case_Count_7Day_Avg,
RESULT     data = add_season_test)
RESULT
RESULT Coefficients:
RESULT                         (Intercept)  add_season_test$Case_Count_7Day_Avg
RESULT                           58.216250                            -0.008072
```

- Step 11: Observe the P-value of the Linear Model

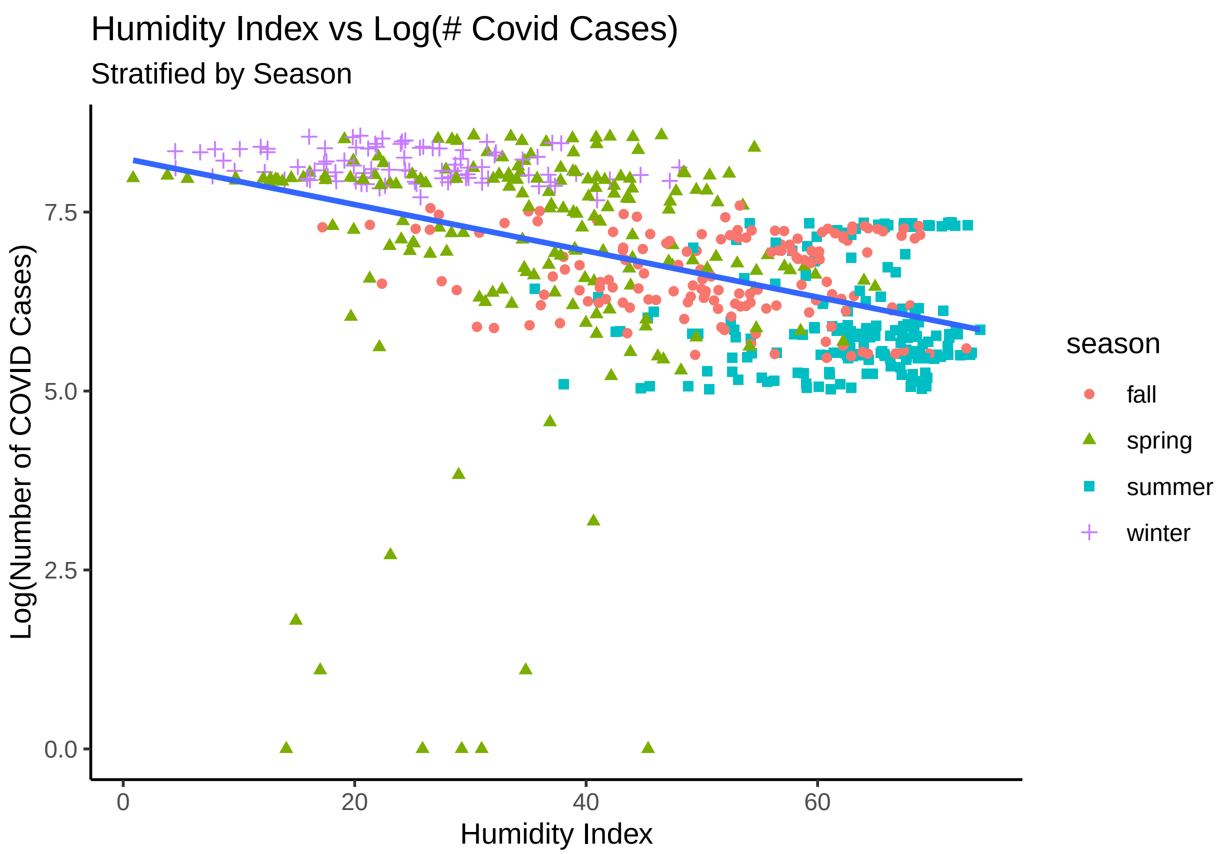
```
# coefficients test
summary(linearMod)$coefficients
```

```
RESULT                                          Estimate    Std. Error   t value
RESULT (Intercept)                           58.21624968 0.8037972340  72.42654
RESULT add_season_test$Case_Count_7Day_Avg -0.00807179 0.0003933093 -20.52275
RESULT                                            Pr(>|t|)
RESULT (Intercept)                           3.495849e-301
RESULT add_season_test$Case_Count_7Day_Avg  1.628114e-71
```

- Step 7: Improve Linear Model By using the Log incident case values
  - Step 7A: Scatter Plot
    - Humidity vs Log Cases

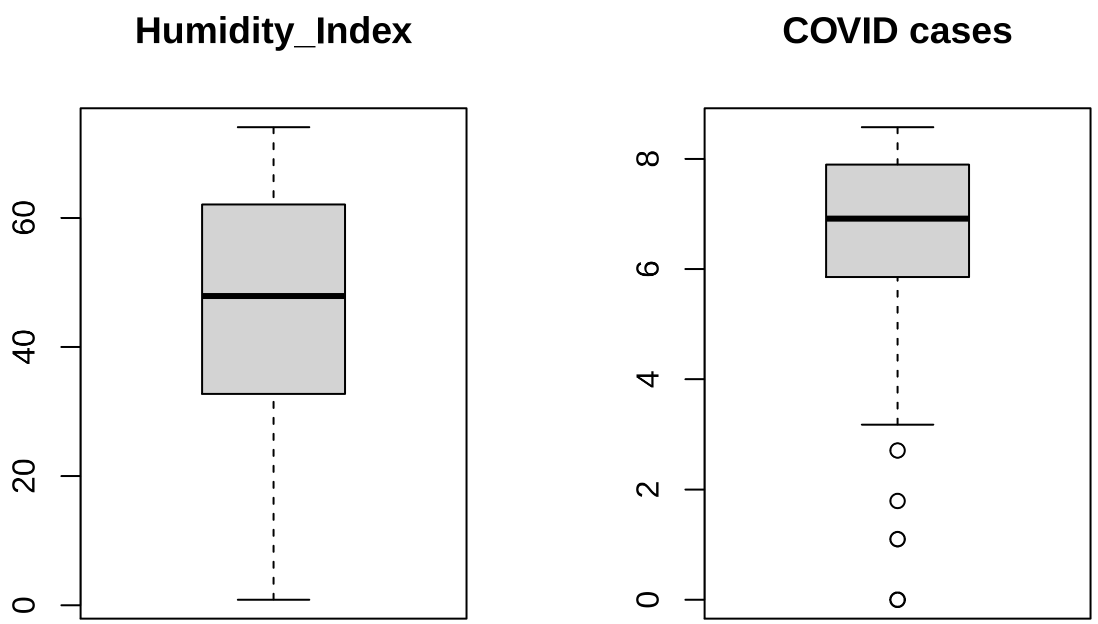
```
# scatter plot of humidity index

ggplot(data = add_season_test) + geom_point(mapping = aes(x = humidity_index, y = log_cases,
    color = season, shape = season)) + labs(x = "Humidity Index", y = "Log(Number of COVID Cases)",
    title = "Humidity Index vs Log(# Covid Cases)", subtitle = "Stratified by Season") +
    geom_smooth(mapping = aes(x = humidity_index, y = log_cases), method = "lm",
        se = F, fullrange = T) + theme_classic()
```

```
RESULT `geom_smooth()` using formula 'y ~ x'
```



- Step 7B: Generate a Boxplot
  - Humidity vs Log Cases

```r
# boxplot
par(mfrow = c(1, 2))  # divide graph area in 2 columns
boxplot(add_season_test$humidity_index, main = "Humidity_Index")
boxplot(add_season_test$log_cases, main = "COVID cases")
```



- Step 7C: Compute the Correlation Coefficient

```r
# correlation
cor(add_season_test$humidity_index, add_season_test$log_cases)
```

```
RESULT [1] -0.4426854
```

- Step 7D: Generate a Single Variable Linear Model
  - Humidity vs Log Cases

```r
# linear model of humidity_index
linearMod <- lm(add_season_test$humidity_index ~ add_season_test$log_cases, data = add_season_test)  #
print(linearMod)
```

```
RESULT
RESULT Call:
RESULT lm(formula = add_season_test$humidity_index ~ add_season_test$log_cases,
RESULT     data = add_season_test)
RESULT
RESULT Coefficients:
RESULT               (Intercept)  add_season_test$log_cases
RESULT                     87.06                      -6.06
```

- Step 7E: View the Coefficients of the Linear Model
  - P-value

```r
# coefficients test
summary(linearMod)$coefficients
```

```
RESULT                            Estimate Std. Error   t value     Pr(>|t|)
RESULT (Intercept)                87.06246  3.4255974  25.41526 1.213937e-97
RESULT add_season_test$log_cases -6.06033  0.4978349 -12.17337 1.157569e-30
```

AIM 2

1. Is There an Association between UV Index and COVID Incidence?

- Step 1: Read in UV data

```r
nyc_uv_level <- read.csv("data/uv_total_level.csv")
colnames(nyc_uv_level)[1] <- "Date"
nyc_uv_level$Date <- mdy(nyc_uv_level$Date)
nyc_uv_level <- nyc_uv_level[order(nyc_uv_level$Date), ]

# change date type from ymd to mdy
nyc_uv_level$Date <- format(as.Date(nyc_uv_level$Date, "%Y/%m/%d"), "%m/%d/%Y")

# rownames
rownames(nyc_uv_level) <- c(1:nrow(nyc_uv_level))
```

- Step 2: Join Data Frames using SQL in R (using "sqldf" package)

```r
total_df = sqldf("select * from add_season_test
      left join nyc_uv_level
      on add_season_test.date = nyc_uv_level.Date")
```

- Step 3: Delete the "City" variable
  - Every observation is "new york city"

```r
total_df <- select(total_df, -City)
```

- Step 4: Generate a Scatter Plot
  - UV vs Case Count

```r
ggplot(data = total_df) + geom_point(mapping = aes(x = UV, y = Case_Count_7Day_Avg,
    color = season, shape = season)) + labs(x = "UV", y = "Number of COVID Cases",
    title = "UV vs Number of COVID Cases") + theme_classic() + geom_smooth(mapping = aes(x = UV,
    y = Case_Count_7Day_Avg), method = "lm", inherit.aes = F)
```

- Step 5: Generate a Boxplot

```r
par(mfrow = c(1, 2))  # divide graph area in 2 columns
boxplot(total_df$UV, main = "UV")
boxplot(total_df$Case_Count_7Day_Avg, main = "COVID cases")
```

- Step 6: Compute Correlation Between UV and Cases

```r
cor(total_df$UV, total_df$Case_Count_7Day_Avg)
```

- Step 7: Single vVariable Linear Regression Model
  - UV vs Cases

```r
linearMod <- lm(total_df$UV ~ total_df$Case_Count_7Day_Avg, data = total_df)  # build linear regression
print(linearMod)
```

- Step 8: Check UV ~ Cases P-value

```r
summary(linearMod)$coefficients
```

- Step 9: Generate a Scatter Plot
  - UV ~ Cases to UV ~ log(cases)

```r
ggplot(total_df,aes(x=`UV`,y=`log_cases`,color=levels,shape=levels))+
  geom_point()+ #this controls the scatter plots
  labs(x="UV", y ="log_cases",title="UV ~ log cases Scatter Plot")+theme_classic()+geom_smooth(aes(x=`U
```

- Step 10: Generate a Boxplot
  - UV ~ Cases to UV ~ log(cases)

```r
par(mfrow = c(1, 2))  # divide graph area in 2 columns
boxplot(total_df$UV, main = "UV")
boxplot(total_df$log_cases, main = "log cases")
```

- Step 11: Compute the Correlation Between UV and log(cases)

```r
cor(total_df$UV, total_df$log_cases)
```

- Step 12: Generate a Single Variable Linear Regression Model
  - UV ~ log(cases)

```r
linearMod <- lm(total_df$UV ~ total_df$log_cases, data = total_df)  # build linear regression model on
print(linearMod)
```

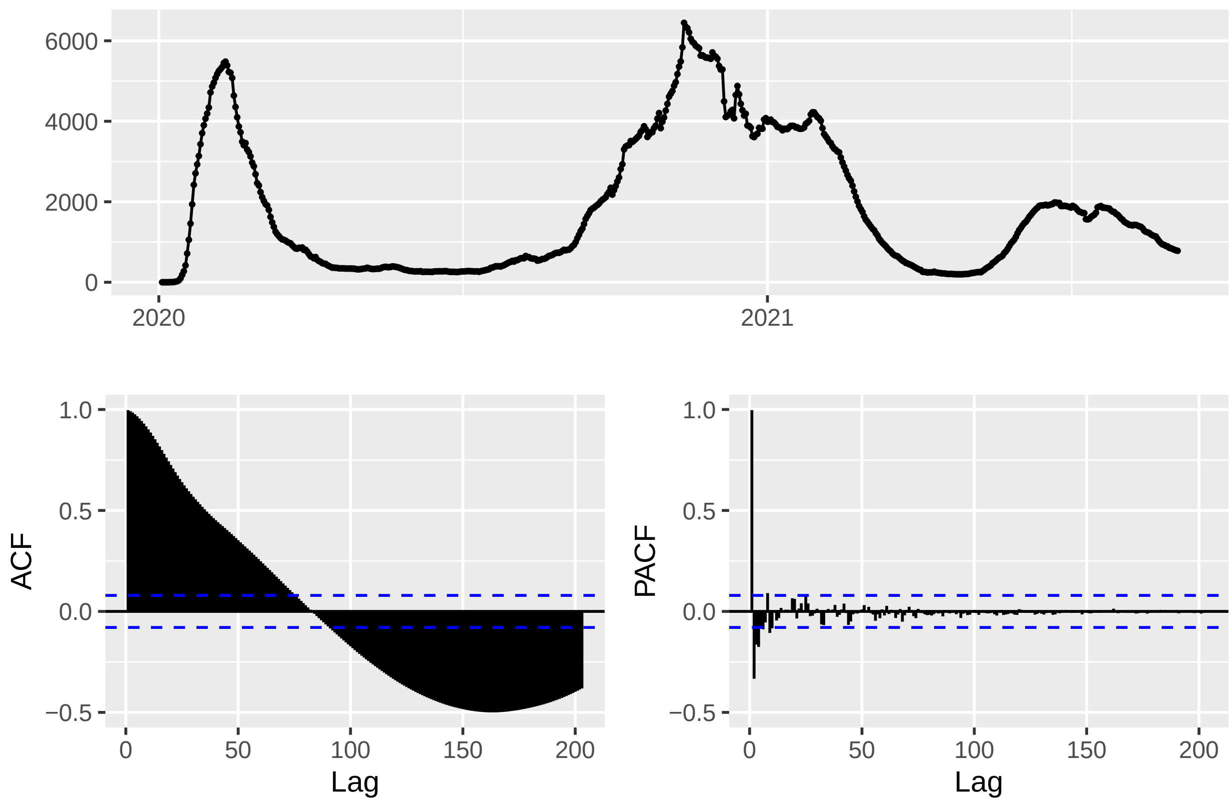
- Step 13: Check UV~ log cases P-value

```r
summary(linearMod)$coefficients
```

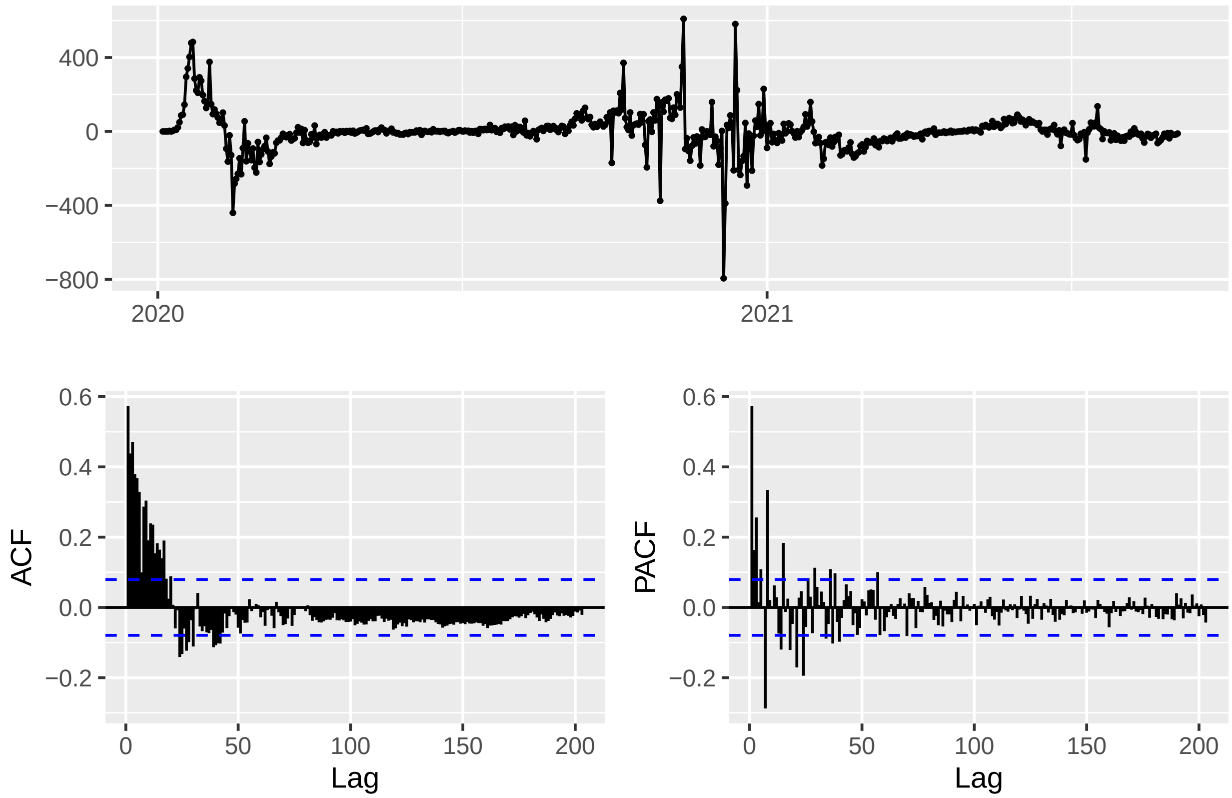
Predictive Model

```r
# Load Data
nyc <- readRDS("data/nycnew.rds")
covid <- read.csv("data/NYC_COVID_DATA/NYC_Covid_Data.csv")
uv <- read.csv("data/uv_total.csv")
rawdata <- ts(covid$all_case_count_7day_avg, start = c(2020, 3, 1), frequency = 365)
weather <- ts(nyc, start = c(2020, 3, 1), frequency = 365)
ultraviolet <- ts(uv$UV, start = c(2020, 3, 1), frequency = 365)
```

```r
# Basic timeseries visulization and prepare for the model
rawdata %>%
    ggtsdisplay()
```




```r
rawdata %>%
    diff() %>%
    ggtsdisplay()
```



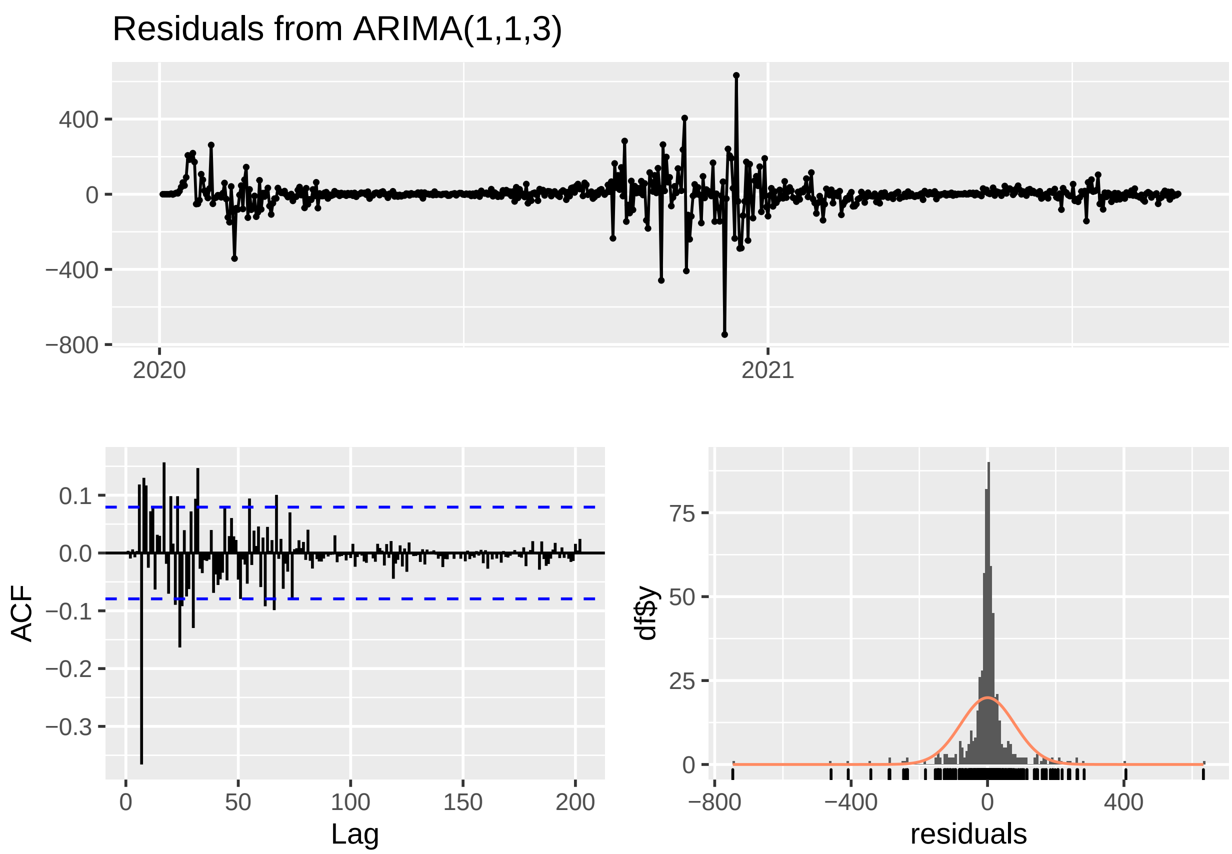

```r
# build best arima model and check the residuals
fit <- auto.arima(rawdata, biasadj = TRUE, parallel = TRUE, stepwise = FALSE)
```

```
RESULT Warning: The chosen seasonal unit root test encountered an error when testing for the first diffe
RESULT From stl(): series is not periodic or has less than two periods
RESULT 0 seasonal differences will be used. Consider using a different unit root test.
```

```r
summary(fit)
```

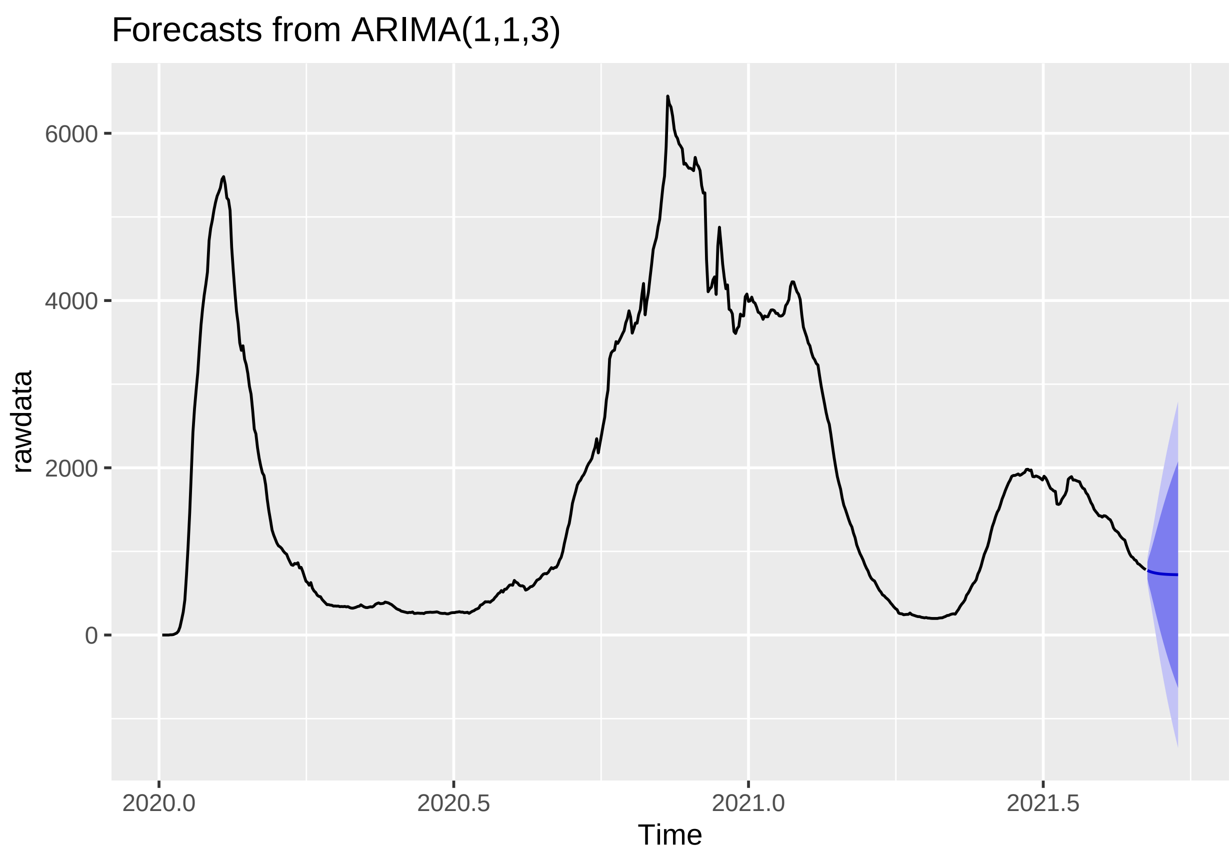
```
RESULT Series: rawdata
RESULT ARIMA(1,1,3)
RESULT
RESULT Coefficients:
RESULT          ar1      ma1      ma2     ma3
RESULT       0.8361  -0.4127  -0.1322  0.1599
RESULT s.e.  0.0491   0.0568   0.0496  0.0498
RESULT
RESULT sigma^2 estimated as 6262:  log likelihood=-3524.45
RESULT AIC=7058.9   AICc=7059   BIC=7080.96
RESULT
RESULT Training set error measures:
RESULT                     ME     RMSE    MAE       MPE     MAPE       MASE
RESULT Training set 0.3143574 78.80526 37.751 0.6747284 2.804311 0.03684213
RESULT                     ACF1
RESULT Training set 0.003730441
```

```
# fit <- arima(case_count, order = c(3,1,0))
checkresiduals(fit)
```




```
RESULT
RESULT  Ljung-Box test
RESULT
RESULT data:  Residuals from ARIMA(1,1,3)
RESULT Q* = 303.36, df = 118, p-value < 2.2e-16
RESULT
RESULT Model df: 4.   Total lags used: 122
```

```
# Make prediction
fit %>%
    forecast(h = 20) %>%
    autoplot()
```




```
temperature <- ts(nyc$Day.Average.Temp, start = c(2020, 3, 1), frequency = 365)
humidity <- ts(nyc$Humidity, start = c(2020, 3, 1), frequency = 365)

# temperature and humidity as outer variables for the models
fit2 <- auto.arima(rawdata, xreg = cbind(humidity), biasadj = TRUE, parallel = TRUE,
    stepwise = FALSE)
```

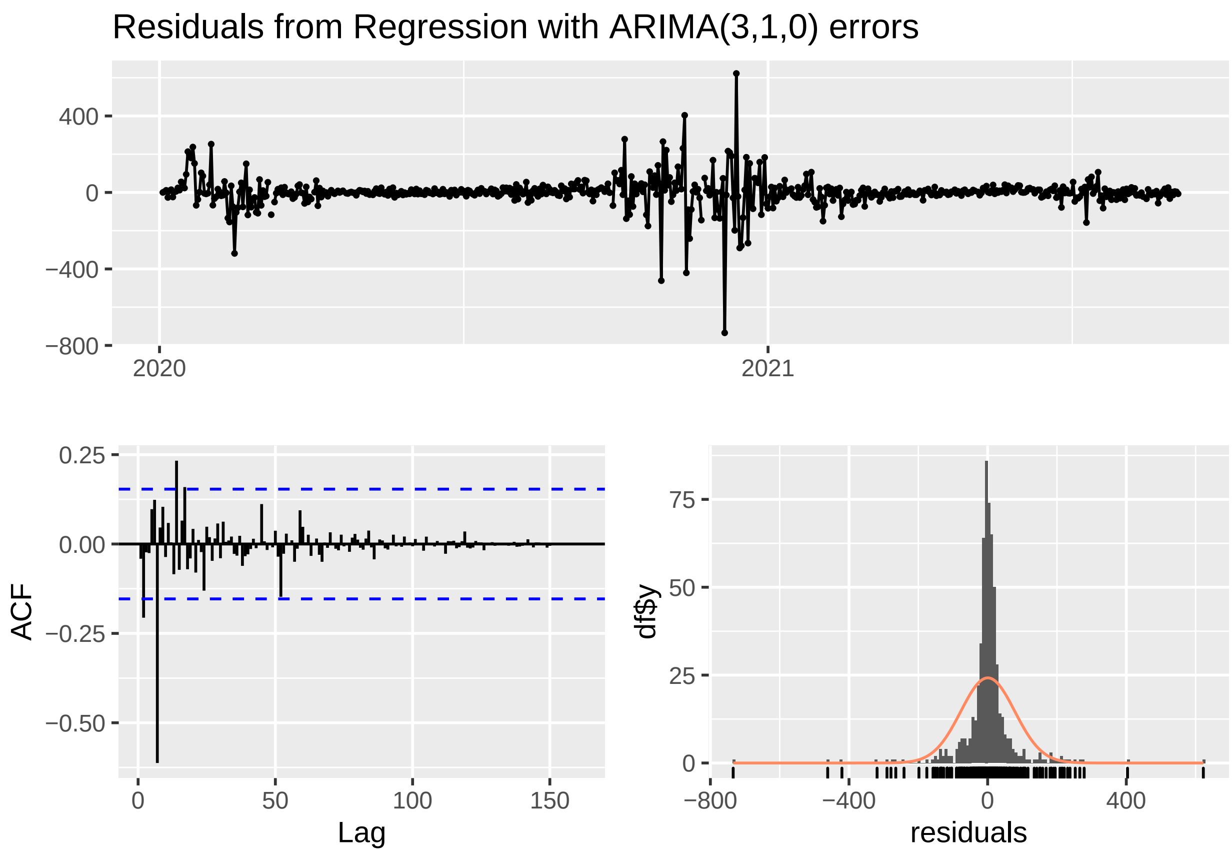
```
RESULT Warning: The chosen seasonal unit root test encountered an error when testing for the first diff
RESULT From stl(): series is not periodic or has less than two periods
RESULT 0 seasonal differences will be used. Consider using a different unit root test.
```

```
summary(fit2)
```

```
RESULT Series: rawdata
RESULT Regression with ARIMA(3,1,0) errors
RESULT
RESULT Coefficients:
RESULT          ar1     ar2     ar3     xreg
RESULT       0.4595  0.0177  0.2605  -0.6175
RESULT s.e.  0.0395  0.0438  0.0391   0.1781
RESULT
RESULT sigma^2 estimated as 6017:  log likelihood=-3470.45
RESULT AIC=6950.91   AICc=6951.01   BIC=6972.97
RESULT
RESULT Training set error measures:
RESULT                     ME     RMSE      MAE MPE MAPE       MASE        ACF1
```

```
RESULT Training set 0.4269349 77.82989 38.33502 NaN  Inf 0.03741209 0.006014637
```

```r
checkresiduals(fit2)
```




```
RESULT 
RESULT  Ljung-Box test
RESULT 
RESULT data:  Residuals from Regression with ARIMA(3,1,0) errors
RESULT Q* = 231.14, df = 118, p-value = 2.387e-09
RESULT 
RESULT Model df: 4.   Total lags used: 122
```

```r
# intro the ultraviolet as the variables
fit3 <- auto.arima(rawdata, xreg = cbind(ultraviolet, temperature, humidity), biasadj = TRUE,
    parallel = TRUE, stepwise = FALSE)
```

```
RESULT Warning: The chosen seasonal unit root test encountered an error when testing for the first diff
RESULT From stl(): series is not periodic or has less than two periods
RESULT 0 seasonal differences will be used. Consider using a different unit root test.
```

```r
summary(fit3)
```

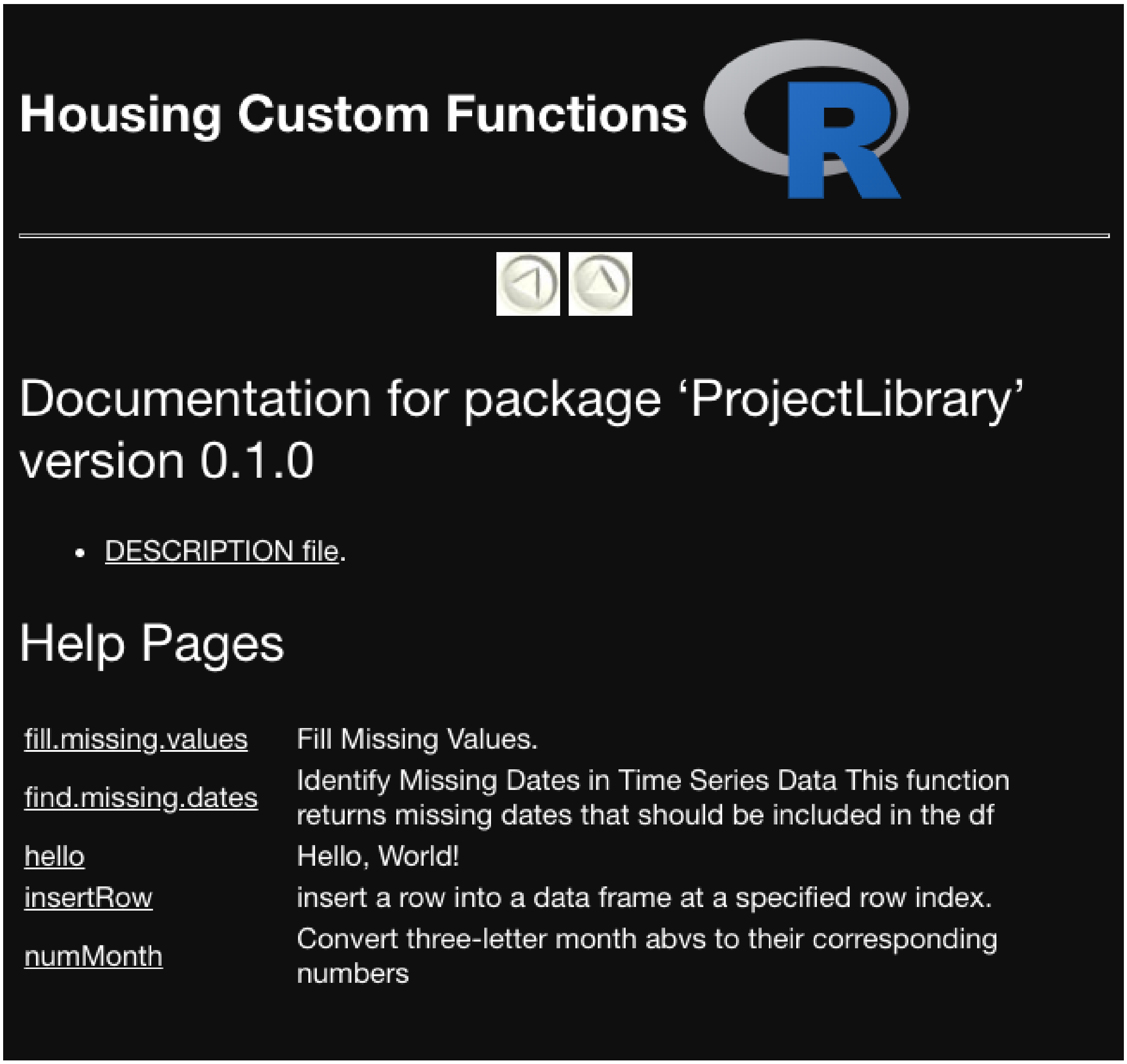
```
RESULT Series: rawdata
RESULT Regression with ARIMA(3,1,0) errors
```

```
RESULT
RESULT Coefficients:
RESULT          ar1     ar2     ar3  ultraviolet  temperature  humidity
RESULT       0.4586  0.0196  0.2601      -0.2157       0.4398   -0.6181
RESULT s.e.  0.0395  0.0439  0.0391       1.9445       0.5260    0.1891
RESULT
RESULT sigma^2 estimated as 6030:  log likelihood=-3470.1
RESULT AIC=6954.19   AICc=6954.38   BIC=6985.08
RESULT
RESULT Training set error measures:
RESULT                     ME     RMSE      MAE MPE MAPE       MASE        ACF1
RESULT Training set 0.4243668 77.78457 38.32381 NaN  Inf 0.03740115 0.006145892
```

Build Library

- Step 1: Create an R Package as a subdirectory of the project repository.
  - Location: file = "QBS181final/library/ProjectLibrary"
- Fill out the description
- Save custom functions in the "R" folder
  - Use "roxygen2" to generate out skeleton docstrings
  - Build the package
    - Generate the manuals

```r
knitr::include_graphics(path = "images/package_documentation.png")
```



Housing Custom Functions

Documentation for package ‘ProjectLibrary’ version 0.1.0

- DESCRIPTION file.

Help Pages

fill.missing.values	Fill Missing Values.
find.missing.dates	Identify Missing Dates in Time Series Data This function returns missing dates that should be included in the df
hello	Hello, World!
insertRow	insert a row into a data frame at a specified row index.
numMonth	Convert three-letter month abvs to their corresponding numbers

Figure 4: Custom Function Library: Package Documentation