Analysis 3: Statistical Programming with Traffic Data

Anthony Hu

06/12/2021

Instructions

Overview: For each question, show your R code that you used to answer each question in the provided chunks. When a written response is required, be sure to answer the entire question in complete sentences outside the code chunks. When figures are required, be sure to follow all requirements to receive full credit. Point values are assigned for every part of this analysis.

Helpful: Make sure you knit the document as you go through the assignment. Check all your results in the created PDF file.

Submission: Submit via an electronic document on Sakai. Must be submitted as an HTML or a PDF file generated in RStudio.

Introduction

Sensors are used by traffic management systems across the world to monitor the flow of vehicular traffic in urban road systems. Two metrics, occupancy and volume, indicate the quality of traffic flow. Occupancy is a variable that is between 0 an 100 and measures the percentage of time a sensor is blocked by a vehicle. Volume is a variable which represents the number of vehicles that pass a sensor in a period of time. The datasets "April_Occupancy_3.csv" and "April_Volume_3.csv" record 3-minute occupancy and volume measures for Monday through Friday in the month of April for 7 sensor locations in Athens, Greece.

The first goal of this analysis is to help you build a foundation of R programming skills. The second goal of this analysis is for you to analyze the traffic data in Greece.

Below is a preview of the 2 datasets. The tibble named `DATA1` contains traffic occupancies and the tibble named `DATA2` contains traffic volumes. The first 7 columns represent the 7 sensor locations. Each row represents the traffic occupancy or traffic volume measured on a 3 minute interval. The rows are ordered according to the variable `TIME` which indicates the sequence in which these measures are observed. The variable `DAY` indicates the day in April that the measure was recorded.

```r
DATA1=as_tibble(read.csv("April_Occupancy_3.csv"))
DATA2=as_tibble(read.csv("April_Volume_3.csv"))
head(DATA1)
```

```
## # A tibble: 6 x 9
##    L101  L102  L103  L104  L106  L107  L108   DAY  TIME
##   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <int> <int>
## 1  1.58 21.3   12.8  56.1 13.6   7.87 12.8      3     1
## 2  3.34 10.4   10.4  26.2 13.0   4.92  6.50     3     2
## 3  2.76 11.0   10.4  28.2 12.2   6.1   8.66     3     3
## 4  3.15 23.8   17.3  35.2 17.7   6.3  13.6      3     4
## 5  3.15 20.5   12.0  43.1 16.9   6.69  8.46     3     5
## 6  1.18  6.50  10.0  26.2  8.46  4.53  3.15     3     6
```

```r
head(DATA2)
```

```
## # A tibble: 6 x 9
##    L101  L102  L103  L104  L106  L107  L108   DAY  TIME
##   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <int> <int>
## 1    12    75    76    76    88    73    66     3     1
## 2    23    61    68    62    89    52    56     3     2
## 3    19    60    67    66    94    56    50     3     3
## 4    20    65    80    73   107    59    65     3     4
## 5    25    86    79    71   107    60    61     3     5
## 6    10    39    38    40    53    41    21     3     6
```

```r
unique(DATA1$DAY)
```

```
##  [1]  3  4  5  6  7 10 11 12 13 14 17 18 19 20 21 24 25 26 27 28
```

Follow the steps to accomplish specific tasks, and do not modify code on lines with the comment `#DO NOT CHANGE`. Add code in R code chunks wherever you see `COMPLETE`.

Assignment

Part 1: Getting Day Names

We want to start by creating a factor variable called `DAYNAME` that contains the abbreviated day names "M", "T", "W","Th","F". Currently, the unique values in `DAY` are 3, 4, 5, 6, 7, 10, 11, 12, 13, 14, 17, 18, 19, 20, 21, 24, 25, 26, 27, 28. Think about which of these days correspond to their appropriate abbreviated names. The first element in `DAY`, which is *3*, corresponds to "M".

```r
length(unique(DATA1$DAY))
```

```
## [1] 20
```

```r
unique(DATA1$DAY)
```

```
##  [1]  3  4  5  6  7 10 11 12 13 14 17 18 19 20 21 24 25 26 27 28
```

Q1 *(4 Points)*

Although this is our ultimate goal, let's start to think about how we are going to do this by practicing this on a random vector of numbers in `unique(DATA1$DAY)`. Using the sample function, I initiated a random vector containing the values 3, 4, 5, 6, 7, 10, 11, 12, 13, 14, 17, 18, 19, 20, 21, 24, 25, 26, 27, 28. Furthermore, I printed the first 20 observations from this random vector called `x`.

```r
set.seed(216) #DO NOT CHANGE
x=sample(DATA1$DAY,1000) #DO NOT CHANGE
print(x[1:20]) #DO NOT CHANGE
```

```
##  [1] 18 21 10 24 10  6  3 12 18 24 11 12 17 24 20 28 21  4 10 18
```

In the block of code below, use a **for** loop and sequence of **if-else** statements to create a new character vector called `y` which contains the abbreviated day names corresponding to the simulated days in April found in `x`. If you do not follow instructions and use a **for** loop along with **if-else** statements, you will lose points. Think about the code that is given and modify it so it works. The final statement will print the first 20 observations in `y` which should contain abbreviated day names of Monday through Friday.

Code and Output:

```
y = rep(NA,length(x)) #DO NOT CHANGE

for (i in seq_along(x)){
  y[i] =  if(x[i] %in% c(3,10,17,24)){
              "M"
          }else if(x[i] %in% c(4,11,18,25)){
              "T"
          }else if(x[i] %in% c(5,12,19,26)){
              "W"
          }else if(x[i] %in% c(6,13,20,27)){
              "Th"
          }else{
              "F"
          }
}
print(y[1:20]) #DO NOT CHANGE
```

```
##  [1] "T"  "F"  "M"  "M"  "M"  "Th" "M"  "W"  "T"  "M"  "T"  "W"  "M"  "M"
## [15] "Th" "F"  "F"  "T"  "M"  "T"
```

Q2 *(4 Points)*

The previous code was just practice. Now, we must utilize this process for our two tibbles `DATA1` and `DATA2`. In the code chunk below, we start with a vector called `x` which is based off the variable `DAY` from `DATA1`. I want you to apply the previous code to create a vector called `y` with the abbreviated day names. Then, I want you to create a factor vector called `y.factor` based on `y` with ordered levels according to the abbreviated day names.

Code and Output:

```
x = DATA1$DAY #DO NOT CHANGE
?vector
unique(DATA1$DAY)
```

```
##  [1]  3  4  5  6  7 10 11 12 13 14 17 18 19 20 21 24 25 26 27 28
```

```
y <- vector("character",nrow(DATA1))
for (i in seq_along(x)){
  y[i] =  if(x[i] %in% c(3,10,17,24)){
              "M"
          }else if(x[i] %in% c(4,11,18,25)){
              "T"
          }else if(x[i] %in% c(5,12,19,26)){
              "W"
          }else if(x[i] %in% c(6,13,20,27)){
```

```r
              "Th"
          }else{
              "F"
          }
}

y.factor = factor(y,levels = c("M","T","W", "Th", "F"))
print(y.factor[1:20]) #DO NOT CHANGE
```

```
##  [1] M M M M M M M M M M M M M M M M M M M M
## Levels: M T W Th F
```

Q3 *(4 Points)*

Now that you understand the process, we want to create a function called `DAYNAME.func` that takes a single argument called `data`. This function should extract the appropriate numeric vector of days in April and output a factor vector of categorical day names. The content of this function should be very similar to the previous code. The final four lines of the code chunk will create the new variable `DAYNAME` in both datasets and then print the first 6 rows using `head()`.

Code and Output:

```r
DAYNAME.func = function(data){

  x = data$DAY #DO NOT CHANGE

  for (i in seq_along(x)){
    y[i] =  if(x[i] %in% c(3,10,17,24)){
                "M"
            }else if(x[i] %in% c(4,11,18,25)){
                "T"
            }else if(x[i] %in% c(5,12,19,26)){
              "W"
            }else if(x[i] %in% c(6,13,20,27)){
              "Th"
            }else{
              "F"
            }
  }

  y.factor = factor(y,levels = c("M","T","W", "Th", "F"))

  return(y.factor)
}


DATA1$DAYNAME = DAYNAME.func(DATA1) #DO NOT CHANGE
DATA2$DAYNAME = DAYNAME.func(DATA2) #DO NOT CHANGE
head(DATA1) #DO NOT CHANGE
```

```
## # A tibble: 6 x 10
##    L101  L102  L103  L104  L106  L107  L108   DAY  TIME DAYNAME
##   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <int> <int> <fct>
## 1  1.58 21.3   12.8  56.1 13.6   7.87 12.8      3     1 M
## 2  3.34 10.4   10.4  26.2 13.0   4.92  6.50     3     2 M
## 3  2.76 11.0   10.4  28.2 12.2   6.1   8.66     3     3 M
## 4  3.15 23.8   17.3  35.2 17.7   6.3  13.6      3     4 M
## 5  3.15 20.5   12.0  43.1 16.9   6.69  8.46     3     5 M
## 6  1.18  6.50  10.0  26.2  8.46  4.53  3.15     3     6 M
```

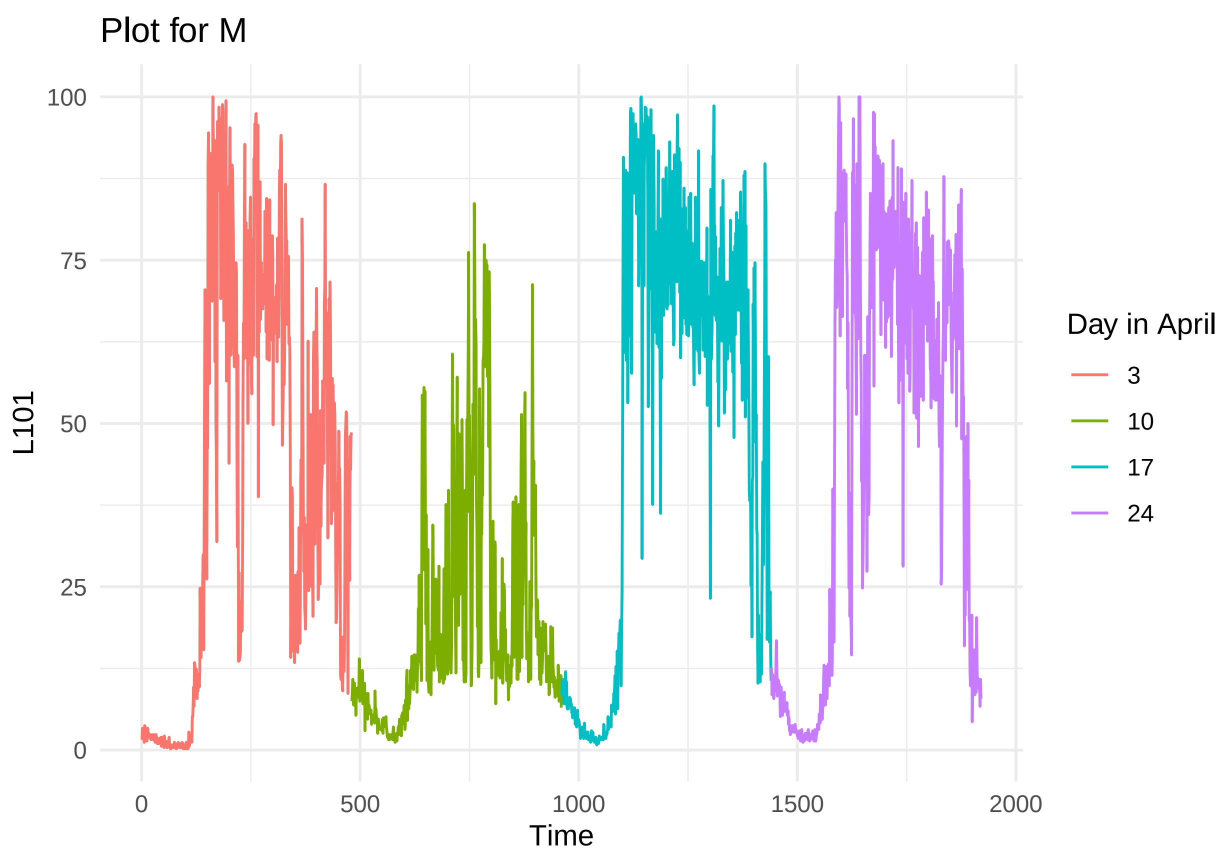
```
head(DATA2) #DO NOT CHANGE
```

```
## # A tibble: 6 x 10
##    L101  L102  L103  L104  L106  L107  L108   DAY  TIME DAYNAME
##   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <int> <int> <fct>
## 1    12    75    76    76    88    73    66     3     1 M
## 2    23    61    68    62    89    52    56     3     2 M
## 3    19    60    67    66    94    56    50     3     3 M
## 4    20    65    80    73   107    59    65     3     4 M
## 5    25    86    79    71   107    60    61     3     5 M
## 6    10    39    38    40    53    41    21     3     6 M
```

Q4 *(6 Points)*

Assuming you followed all directions, the code below displays a plot that shows the change in traffic occupancy for all four Mondays at the location `L101`. Different colors are used to identify the different Mondays in the month of April. Change `eval=F` TO `eval=T` once your previous code works to view the plot.

```
#DO NOT CHANGE ANY OF THIS CODE
SUB.DATA = DATA1 %>% select("L101",DAY,DAYNAME) %>% filter(DAYNAME=="M")
plot=ggplot(data=SUB.DATA) +
      geom_line(aes(x=1:dim(SUB.DATA)[1],y=get("L101"),color=as.factor(DAY))) +
      theme_minimal() +
      xlab("Time") + guides(color=guide_legend(title="Day in April"))+
      ylab("L101") + ggtitle(paste("Plot for ","M",sep=""))
plot
```



I want you to create a function called `DAYPLOT.func` which takes three arguments `data`, `sensor`, and `dayabbrev`. This function should allow a user to specify the tibble (`DATA1` or `DATA2`), the sensor location ("L101","L102","L103",etc.), and the abbreviated weekday name ("M","T","W",etc.). Your function `DAYPLOT.func()` should create the image above if you use arguments `data=DATA1`, `sensor="L101"`, and `dayabbrev="M"`.

Code *(4 Points)*:

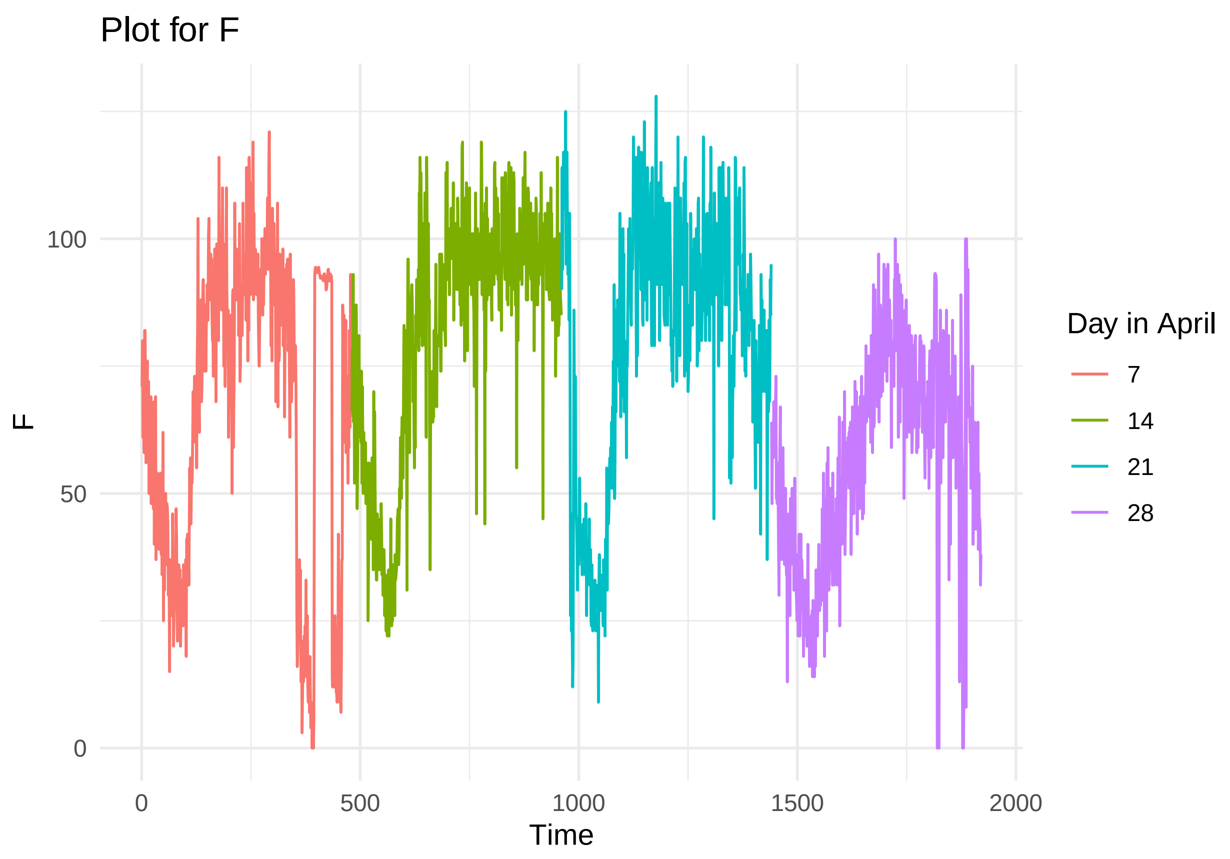
```r
DAYPLOT.func=function(data,sensor,dayabbrev){

      SUB.DATA = data %>% select(sensor,DAY,DAYNAME) %>% filter(DAYNAME==dayabbrev)
      plot=ggplot(data=SUB.DATA) +
      geom_line(aes(x=1:dim(SUB.DATA)[1],y=get(sensor),color=as.factor(DAY))) +
      theme_minimal() +
      xlab("Time") + guides(color=guide_legend(title="Day in April"))+
      ylab(dayabbrev) + ggtitle(paste("Plot for ",dayabbrev,sep=""))
plot

  return(plot)
}
```

Now, I want you to use the function you created to output a figure for a sensor location and weekday of your choice showing the change in traffic volume over time. To do this figure for traffic volume, be sure to use `DATA2`. Choose any sensor location except "L101" and any day except "M".

Code and Output *(2 Points)*:

```
DAYPLOT.func(DATA2,"L102", "F")
```

```
## Note: Using an external vector in selections is ambiguous.
## i Use `all_of(sensor)` instead of `sensor` to silence this message.
## i See <https://tidyselect.r-lib.org/reference/faq-external-vector.html>.
## This message is displayed once per session.
```



Part 2: Time Plots of Aggregated Occupancy and Volume Across Locations

Suppose we would want to aggregate information across multiple sensor locations at each time point. This would allow us to assess the overall traffic quality across the entire urban network. Our goal here is to capture the change in traffic occupancy and volume across all 7 locations. To begin, we start by creating data frames `OCC` and `VOL` that only contain the first 7 columns in their respective datasets. These data frames only contain the measurements without the days of the month and time indices. Aggregating across locations involves applying functions to the rows of `VOL` and `OCC`. Since the tibbles `VOL` and `OCC` are essentially matrices, we can use the `apply()` function for this purpose rather than looping through all the rows. Functions such as `apply()`, `lapply()`, and `sapply()` are far more efficient when functions are required to be repeated across rows, columns, or elements in a list.

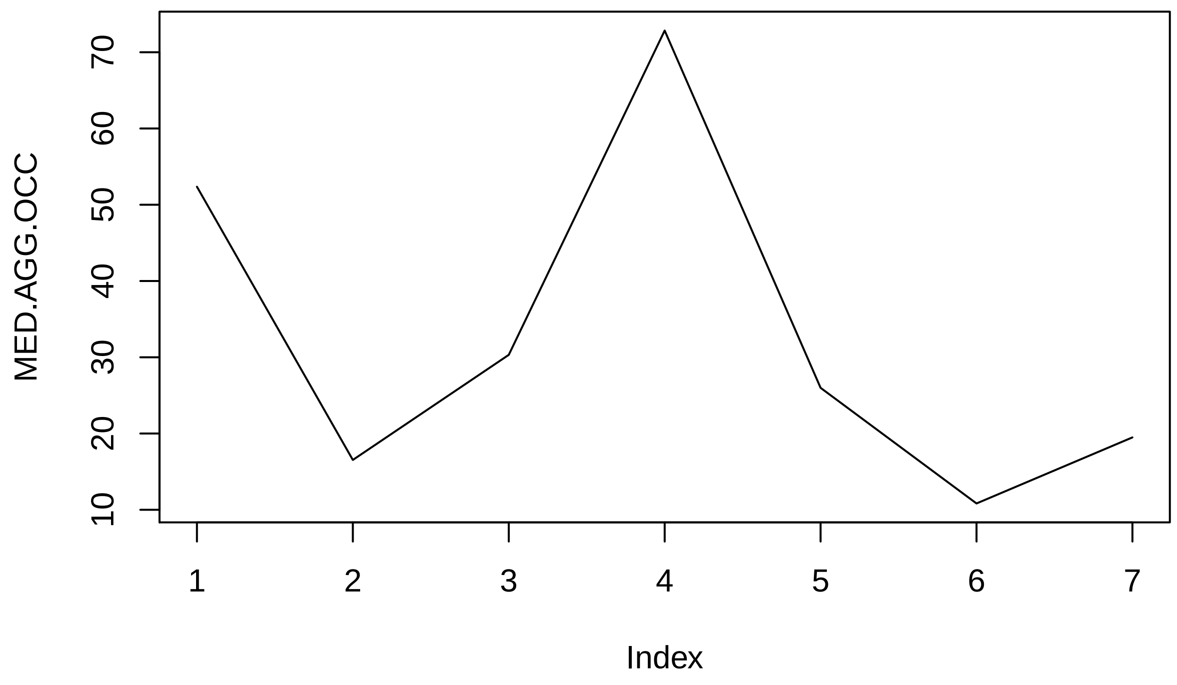
```
OCC=DATA1[,1:7] #DO NOT CHANGE
VOL=DATA2[,1:7] #DO NOT CHANGE
```

Q1 ***(3 Points)***

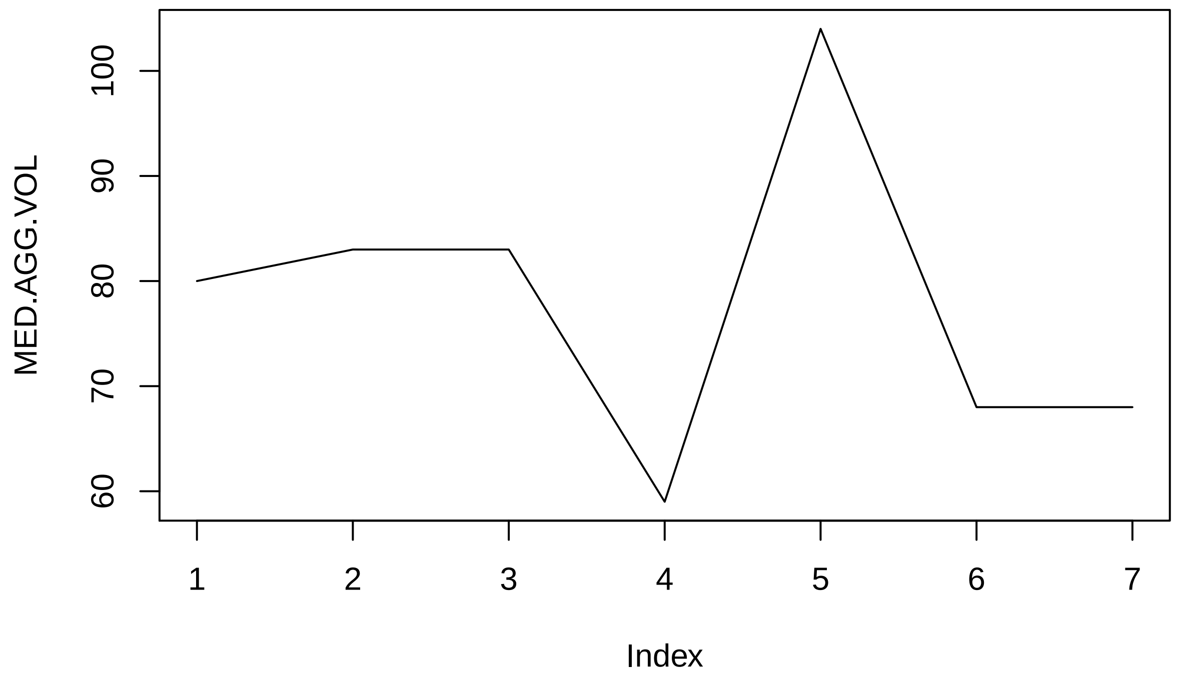
Use `apply()` to obtain the median across different locations of each day in both `OCC` and `VOL`. The output from `apply()` is a vector. First, use `apply()` to create a vector called `MED.AGG.OCC` that contains the medians of `OCC`. Then, use `apply()` to create a vector called `MED.AGG.VOL` that contains the medians of `VOL`. The built-in R function `median()` should be helpful here. If you have used `apply()` correctly, the figures will display the change in median occupancy and median volume over time.

Code and Output:

```
?apply
MED.AGG.OCC = apply(OCC, 2, median)
MED.AGG.VOL = apply(VOL, 2, median)

plot(MED.AGG.OCC,type="l") #DO NOT CHANGE
```



```
plot(MED.AGG.VOL,type="l") #DO NOT CHANGE
```



Q2 ***(4 Points)***

Run the following code line-by-line and observe what is happening.

```
x=c(35,27,28,40)     #DO NOT CHANGE

range=max(x)-min(x)  #DO NOT CHANGE
print(range)         #DO NOT CHANGE
```

```
## [1] 13
```

Create a function called `RANGE.func` that inputs a vector and outputs the range of the observations in that vector. Then create a vector called `try` containing any five numbers of your choice and show that the function works. Use `print()` to display the vector that you created along with the output from using `RANGE.func()`.

Code and Output:

```
#Function
RANGE.func=function(x){
  range = max(x) - min(x)
}

#Output
try = c(1, 2, 3, 4, sample.int(100000, 1))
try
```

```
## [1]     1     2     3     4 71002
```

```
print(RANGE.func(try)) #DO NOT CHANGE
```

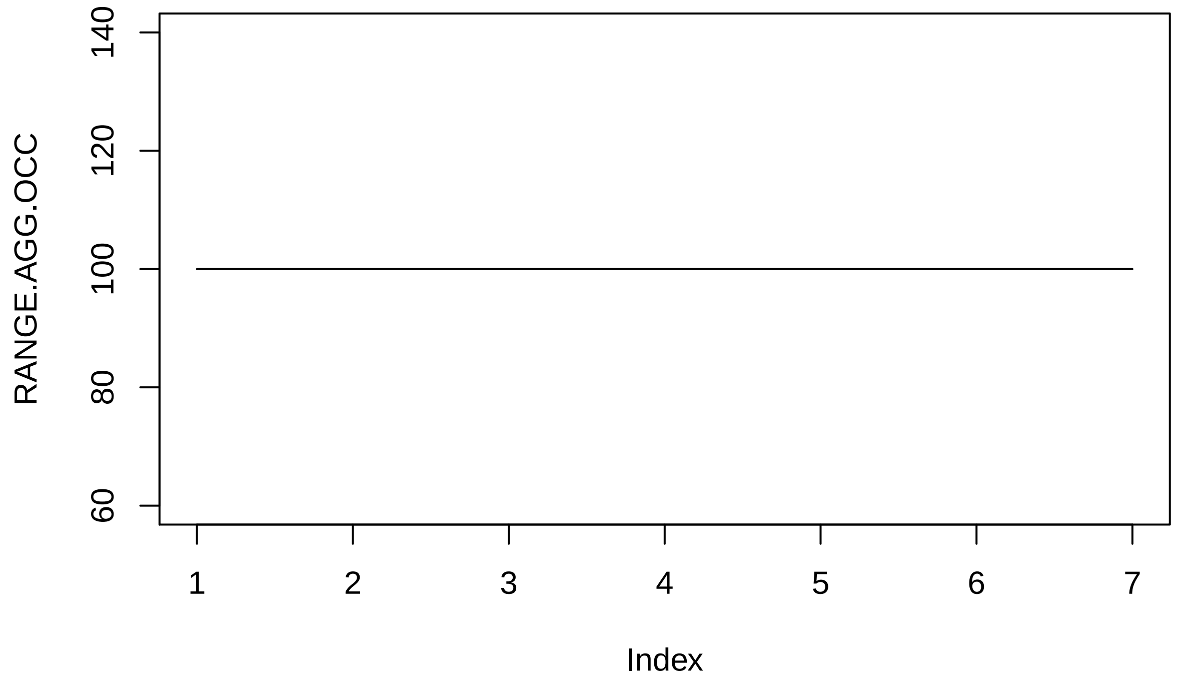
```
## [1] 71001
```

Q3 ***(3 Points)***

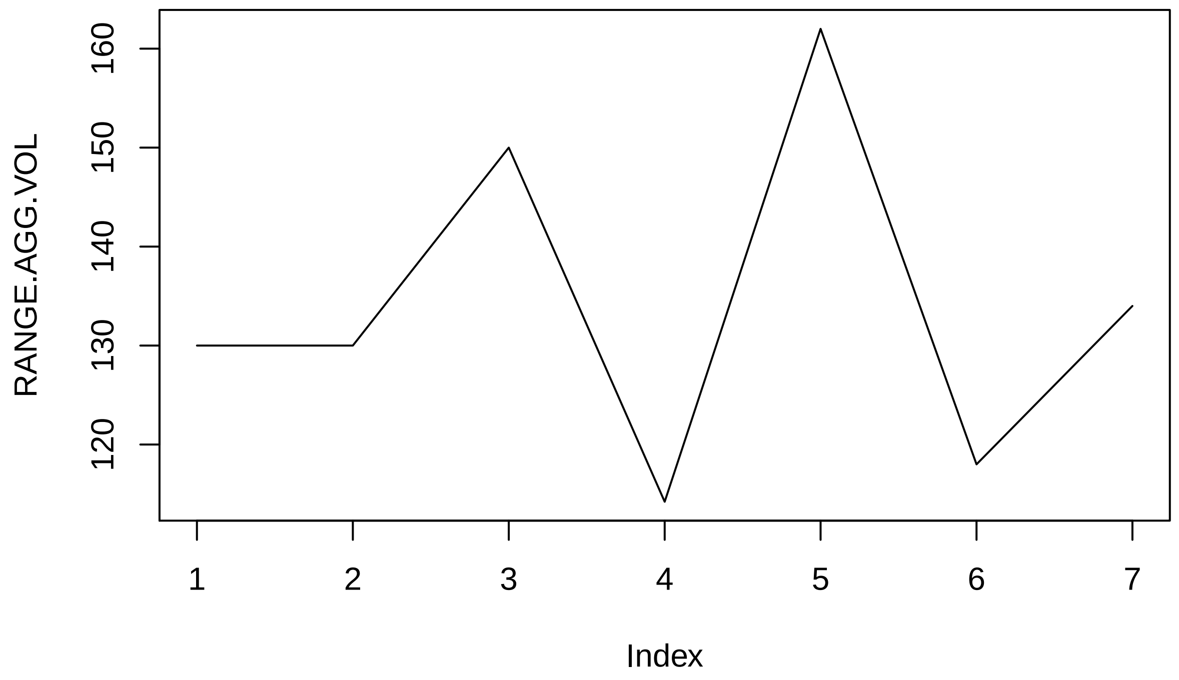
Use `apply()` with your function `RANGE.func()` to obtain the range of occupancy and volume across different locations of each day. The process here is similar to when we obtained the medians in Q1 except now we are not using a built-in R function. First, use `apply()` to create a vector called `RANGE.AGG.OCC` that contains the ranges of occupancy from `OCC`. Then, use `apply()` to create a vector called `RANGE.AGG.VOL` that contains the ranges of volumes from `VOL`.

Code and Output:

```
RANGE.AGG.OCC = apply(OCC, 2, RANGE.func)
RANGE.AGG.VOL = apply(VOL, 2, RANGE.func)

plot(RANGE.AGG.OCC,type="l") #DO NOT CHANGE
```



```
plot(RANGE.AGG.VOL,type="l") #DO NOT CHANGE
```


RANGE.AGG.VOL
Index
120
130
140
150
160
1
2
3
4
5
6
7