For a capital city, it’s fairly small, but it’s increasingly urbanizing (we just got lightrail transit). Segregated bicycle lanes and paths are becoming more common too and many of these paths have trackers on them that count how many bicycles cross a particular street or path each day. What’s great is that this [data is shared publicly](https://open.ottawa.ca/datasets/bicycle-trip-counters) by the city.

library(tidyverse)

library(leaflet)

library(leafpop)

We’ll start by reading in the data from the GitHub repositiory. There’s a lot of missing data, so much that R gets confused about the data structure of some of the columns. We need to add another argument to *read\_csv* telling it the type of data in each column. The *col\_types* argument takes a letter for each column, with *?* meaning that we let R decide what the data is and *n* meaning ‘numeric’.

bikes <- read\_csv("<https://raw.githubusercontent.com/whipson/Ottawa_Bicycles/master/bikes_app.csv>", col\_types = c("?nnnnnnnnnnnnnn"))

bikes

## # A tibble: 3,560 x 15

## date alexandra\_bridge eastern\_canal ottawa\_river

##

## 1 2010-01-01 00:00:00 0 0 0

## 2 2010-01-02 00:00:00 0 0 0

## 3 2010-01-03 00:00:00 0 0 0

## 4 2010-01-04 00:00:00 0 0 0

## 5 2010-01-05 00:00:00 0 0 0

## 6 2010-01-06 00:00:00 0 0 0

## 7 2010-01-07 00:00:00 0 0 0

## 8 2010-01-08 00:00:00 0 0 0

## 9 2010-01-09 00:00:00 0 0 0

## 10 2010-01-10 00:00:00 0 0 0

## # ... with 3,550 more rows, and 11 more variables: western\_canal ,

## # laurier\_bay , laurier\_lyon , laurier\_metcalfe ,

## # somerset\_bridge , otrain\_young , otrain\_gladstone ,

## # otrain\_bayview , portage\_bridge , adawe\_crossing\_a ,

## # adawe\_crossing\_b

Each row is a day and the columns are bicycle counters spread across the city. Let’s start by creating the graphs we want in the Shiny app. It’s easier to do this outside of the Shiny framework first. We’ll start by plotting total bicycle counts over time.

bikes\_total <- bikes %>%

pivot\_longer(names\_to = "counter", values\_to = "count", -date) %>%

group\_by(date) %>%

mutate(daily\_total = sum(count, na.rm = TRUE))

bikes\_total

## # A tibble: 49,840 x 4

## # Groups: date [3,560]

## date counter count daily\_total

##

## 1 2010-01-01 00:00:00 alexandra\_bridge 0 0

## 2 2010-01-01 00:00:00 eastern\_canal 0 0

## 3 2010-01-01 00:00:00 ottawa\_river 0 0

## 4 2010-01-01 00:00:00 western\_canal NA 0

## 5 2010-01-01 00:00:00 laurier\_bay NA 0

## 6 2010-01-01 00:00:00 laurier\_lyon NA 0

## 7 2010-01-01 00:00:00 laurier\_metcalfe NA 0

## 8 2010-01-01 00:00:00 somerset\_bridge NA 0

## 9 2010-01-01 00:00:00 otrain\_young NA 0

## 10 2010-01-01 00:00:00 otrain\_gladstone NA 0

## # ... with 49,830 more rows

And now to plot it over time:

bikes\_total %>%

ggplot(aes(x = date, y = daily\_total)) +

geom\_line(size = .5, alpha = .80, color = "#36648B") +

scale\_x\_datetime(date\_breaks = "2 years", date\_labels = "%Y") +

labs(x = NULL,

y = "Count",

title = "Total Bicycle Crossings in Ottawa",

subtitle = "Jan 2010 - Sep 2019") +

theme\_minimal(base\_size = 16) +

theme(plot.title = element\_text(hjust = .5),

axis.text.x = element\_text(size = 16))

There’s clear seasonality, with bicycle crossings peaking in the summer months and troughing in the winter. There also appears to be a trend, increasing from 2010 to 2017, then leveling out. Does this mean that bicycling is leveling off in Ottawa? We may want to look at specific counters to get a better sense of this.

bikes %>%

pivot\_longer(names\_to = "counter", values\_to = "count", -date) %>%

ggplot(aes(x = date, y = count)) +

geom\_line(size = .5, alpha = .80, color = "#36648B") +

labs(x = NULL,

y = "Count",

title = "Bicycle Crossings in Ottawa by Location",

subtitle = "Jan 2010 - Sep 2019") +

facet\_wrap(~counter) +

theme\_minimal(base\_size = 16) +

theme(plot.title = element\_text(hjust = .5),

axis.text.x = element\_blank())

## Warning: Removed 2191 rows containing missing values (geom\_path).

This graph tells us that we have to be a bit careful about interpreting the total count because some counters are introduced later or go out of commission. The drop in total counts for 2018 could be due to the Western Canal counter going offline that year. What about average counts over time?

bikes %>%

pivot\_longer(names\_to = "counter", values\_to = "count", -date) %>%

group\_by(date) %>%

mutate(daily\_average = mean(count, na.rm = TRUE)) %>%

ggplot(aes(x = date, y = daily\_average)) +

geom\_line(size = .5, alpha = .80, color = "#36648B") +

scale\_x\_datetime(date\_breaks = "2 years", date\_labels = "%Y") +

labs(x = NULL,

y = "Count",

title = "Average Bicycle Crossings in Ottawa",

subtitle = "Jan 2010 - Sep 2019") +

theme\_minimal(base\_size = 16) +

theme(plot.title = element\_text(hjust = .5),

axis.text.x = element\_text(size = 16))

There may be an upward trend, but it’s less clear compared to the total count. We again have to be careful because earlier years have fewer counters online so the average is based on less data. However, knowing both the total and the average counts gives us a pretty clear picture of how cycling is changing over time in Ottawa.

**Maps with Leaflet**

Now we’ll add the functionality of an interactive map – one that shows where the counters are located geographically and allows the user to select specific counters. Earlier we loaded up the *leaflet* and *leafpop* packages. These will help us construct our map of Ottawa.

We’ll also need the latitude and longitude coordinates of the counters. Using information from the [Open Data Ottawa](https://open.ottawa.ca/datasets/bicycle-trip-counters), I found the location of each counter and obtained its latitude and longitude using Google Maps. I also added a bit of descriptive information for each counter. We can put all of this in a dataframe as follows:

coords <- data.frame(counter = names(bikes[,2:15]),

name = c("Alexandra Bridge", "Eastern Canal Pathway", "Ottawa River Pathway", "Western Canal Pathway",

"Laurier at Bay", "Laurier at Lyon", "Laurier at Metcalfe", "Somerset Bridge", "OTrain at Young",

"OTrain at Gladstone", "OTrain at Bayview", "Portage Bridge", "Adawe Crossing A", "Adawe Crossing B"),

lat = c(45.430366, 45.420924, 45.411959, 45.406280,

45.415893, 45.417036, 45.419790, 45.420512,

45.402859, 45.404599, 45.408636, 45.421980,

45.426282, 45.426575),

long = c(-75.704761, -75.685060, -75.723424, -75.681814,

-75.705328, -75.702613, -75.697623, -75.684625,

-75.712760, -75.714812, -75.723644, -75.713324,

-75.670234, -75.669765),

desc = c("Ottawa approach to the NCC Alexandra Bridge Bikeway. This counter was not operational for most of 2010

due to bridge construction. This is one of the more consistent counters, until the internal battery

failed in August 2019.",

"NCC Eastern Canal Pathway approximately 100m north of the Corktown Bridge.",

"NCC Ottawa River Pathway approximately 100m east of the Prince of Wales Bridge. Canada Day in 2011

boasts the highest single day count of any counter.",

"NCC Western Canal Pathway approximately 200m north of “The Ritz”. Out of operation for much of 2018.

MEC Bikefest on May 17, 2015 accounts for the large spike that day.",

"Laurier Segregated Bike lane just west of Bay. Minimal data available due to inactivity after 2014.",

"Laurier Segregated Bike lane just east of Lyon. No longer in operation since 2016.",

"Laurier Segregated Bike lane just west of Metcalfe. Construction in late 2012 accounts for unusual dip

in counts.",

"Somerset bridge over O-Train west-bound direction only. Inexplicably large spike in 2012 followed by a

typical seasonal pattern. Inactive since late 2018.",

"O-Train Pathway just north of Young Street. Minimal data available due to inactivity after 2016. See

O-Train at Gladstone counter for a better estimate.",

"O-Train Pathway just north of Gladstone Avenue. In operation since mid-2013. Shows unusual spike in

November of 2017.",

"O-Train Pathway just north of Bayview Station. In operation since mid-2013. Trending upward.",

"Portage Bridge connecting Gatineau to Ottawa. Installed in late 2013, this counter registered

relatively high traffic but seems to have experienced outages during Winter months. Inactive since early

2016.",

"Adàwe Crossing Bridge bike lane. This counter is one of a pair on this pedestrian bridge. Installed in

2016, it seems to have experienced an outage during the Winter of its inaugural year.",

"The second of two counters on the Adàwe Crossing Bridge. This counter may pick up more pedestrian than

bike traffic, as suggested by the trend over time."))

Now we just pipe the coordinate data into leaflet.

leaflet(data = coords) %>%

addTiles() %>%

addMarkers(~long, ~lat)

Leaflet automatically generates a map of size to fit all the markers. There are a few modifications to make though. One is to have it so that when the user hovers the mouse over a marker a label pops up with the name of that counter. Another is to make the map more aesthetically pleasing. Finally, we may want to add some bounds so that the user can’t scroll too far away from the markers.

leaflet(data = coords) %>%

addTiles() %>%

addMarkers(~long, ~lat, label = ~name) %>%

setMaxBounds(-75.65, 45.38, -75.75, 45.46) %>%

addProviderTiles(providers$CartoDB.Positron)

Great. So we now have the two components of the app: the time plots and the map. Time to bring in Shiny and put it all together. Now, if you have never used Shiny before, this probably isn’t the easiest example to start with.

**Creating the Shiny App**

There are two parts to every Shiny app: the *UI* or User Interface and the *Server*. The UI is like the look and feel of the app, it’s where we tell Shiny what kinds of inputs and outputs we want, how we want to organize the panels, and so on. In contrast, the Server is the engine of the app. We’ll start by constructing the UI. It’s important to note that it’s easier to build a Shiny app in a new R script. So we’re basically going to start over in a new script, which means we’ll reload the packages and the data as if we were starting new:

**Create a new R script**

We’ll start with the packages and data. We haven’t done anything with the UI or Server yet. We usually want to keep the data outside the UI. We’ll also transform our data as we did earlier to generate the total and average time plots.

library(tidyverse)

library(leaflet)

library(leafpop)

library(shiny)

library(shinythemes)

library(shinyWidgets)

bikes <- read\_csv("<https://raw.githubusercontent.com/whipson/Ottawa_Bicycles/master/bikes_app.csv>", col\_types = c("?nnnnnnnnnnnnnn"))

#For ease, I've put the coordinates in a separate file, but you could just as easily rerun the 'coords' object above

coords <- read\_csv("<https://raw.githubusercontent.com/whipson/Ottawa_Bicycles/master/coords.csv>")

bikes\_plot <- bikes %>%

pivot\_longer(names\_to = "counter", values\_to = "count", -date) %>%

left\_join(coords, by = "counter")

bikes\_total <- bikes\_plot %>%

group\_by(date) %>%

summarize(count = sum(count, na.rm = TRUE))

bikes\_mean <- bikes\_plot %>%

group\_by(date) %>%

summarize(count = mean(count, na.rm = TRUE))

Now, still in the same R script, we can build the UI. It’s going to look a bit strange with parentheses all over the place. It’s just customary Shiny scripting to use hanging parentheses.

**Specifying the UI**

ui <- fluidPage(theme = shinytheme("flatly"),

sidebarLayout( #Layout

sidebarPanel(id = "Sidebar", #Side panel

h2("Ottawa Bicycle Counters", align = "center", tags$style("#Sidebar{font-family: Verdana;}")),

fluidRow( # Row 1 of side panel

htmlOutput("caption"), # Caption output, provides descriptive text

tags$style("#caption{font-size: 16px; height: 200px; font-family: Verdana;}")

),

fluidRow( # Row 2 of side panel

htmlOutput("stats"), # Statistics output, provides descriptive statistics

tags$style("#stats{font-size: 16px; height: 125px; font-family: Verdana;}")

),

fluidRow( # Row 3 of side panel

switchInput("average", # User input, allows the user to turn a switch to display the average

"Display Average",

value = FALSE)

),

fluidRow( # Row 4 of side panel

htmlOutput("caption2"), # More caption output

tags$style("#caption2{font-size: 12px; height: 80px; font-family: Verdana;}")

),

fluidRow( # Row 5 of side panel

downloadButton("download", "Download Data") # A button so that users can download the data

)

),

mainPanel(id = "Main", # Main panel (this is where the plots and map go)

fluidRow( # Row 1 of main panel

leafletOutput("map", height = 400) # Here's the output for the map

),

fluidRow( # Row 2 of main panel

plotOutput("timeplot", height = 300) # Here's the output for the time plots

)

)

)

)

There’s the code for the UI. Starting from the top, we use the *FluidPage* function and here I’m using the theme *flatly*. Then I say that I want to use a *sidebarLayout*. From here, I split the code into a *sidebarPanel* and a *mainPanel*. I further split things into *fluidRows* which just helps to organize the layout. All of the #s are notes, of course, and will not actually be run.

The big thing to notice is that there are inputs and outputs. The only input is a *switchInput* which lets the user choose whether to display totals or averages. Everything else is an output. Each of these gets a name, for example, I’m calling the leafletOutput *map*. These names are important, as they will correspond with what we provide in the server part.

**Specifying the Server**

server <- function(input, output) {

output$map <- renderLeaflet({ # Map output

leaflet(data = coords) %>%

addTiles() %>%

addMarkers(~long, ~lat, label = ~name) %>%

setMaxBounds(-75.65, 45.38, -75.75, 45.46) %>%

addProviderTiles(providers$CartoDB.Positron)

})

output$caption2 <- renderUI({ # Lower caption output

str1 <- paste("Created by ", a("Will Hipson.", href = "<https://willhipson.netlify.com/>"))

str2 <- paste("Data courtesy of ", a("Open Data Ottawa.", href = "<https://open.ottawa.ca/datasets/bicycle-trip-counters>"))

str3 <- "2010-01-01 - 2019-09-30"

str4 <- "Updated on 2019-10-24"

HTML(paste(str1, str2, str3, str4, sep = '  
'))

})

observeEvent(input$map\_marker\_click, { # If the user clicks a marker, this line is run.

output$timeplot <- renderPlot({

if(input$average == TRUE) { # if average is selected we get average overlayed

ggplot() +

geom\_line(data = bikes\_plot[bikes\_plot$lat == input$map\_marker\_click$lat, ],

aes(x = date, y = count), size = .5, alpha = .70, color = "#36648B") +

geom\_line(data = bikes\_mean, aes(x = date, y = count), alpha = .50, color = "#9F79EE") +

scale\_x\_datetime(date\_breaks = "2 years", date\_labels = "%Y") +

scale\_y\_continuous(limits = c(0, 6000)) +

labs(x = NULL,

y = "Count",

title = paste(bikes\_plot[bikes\_plot$lat == input$map\_marker\_click$lat,]$name)) +

theme\_minimal(base\_size = 16) +

theme(plot.title = element\_text(hjust = .5),

axis.text.x = element\_text(size = 16),

text = element\_text(family = "Verdana"))

} else { # if average is not selected, then it's just the total

ggplot() +

geom\_line(data = bikes\_plot[bikes\_plot$lat == input$map\_marker\_click$lat, ],

aes(x = date, y = count), size = .5, alpha = .70, color = "#36648B") +

scale\_x\_datetime(date\_breaks = "2 years", date\_labels = "%Y") +

scale\_y\_continuous(limits = c(0, 6000)) +

labs(x = NULL,

y = "Count",

title = paste(bikes\_plot[bikes\_plot$lat == input$map\_marker\_click$lat,]$name)) +

theme\_minimal(base\_size = 16) +

theme(plot.title = element\_text(hjust = .5),

axis.text.x = element\_text(size = 16),

text = element\_text(family = "Verdana"))

}

})

output$caption <- renderUI({ # counter specific description

str1 <- coords[coords$lat == input$map\_marker\_click$lat, ]$desc

HTML(str1)

})

output$stats <- renderUI({ # counter specific statistics

str1 <- "**Statistics**"

str2 <- paste("Total count: ", format(round(sum(bikes\_plot[bikes\_plot$lat == input$map\_marker\_click$lat,]$count, na.rm = TRUE)), big.mark = ","))

str3 <- paste("Average count: ", format(round(mean(bikes\_plot[bikes\_plot$lat == input$map\_marker\_click$lat,]$count, na.rm = TRUE), 1), big.mark = ","))

str4 <- paste("Busiest day: ", bikes\_plot[which.max(bikes\_plot[bikes\_plot$lat == input$map\_marker\_click$lat,]$count),]$date)

HTML(paste(str1, str2, str3, str4, sep = '  
'))

})

})

observeEvent(input$map\_click, ignoreNULL = FALSE, { # If the user clicks on the map it goes back to the cumulative data

output$timeplot <- renderPlot({

if(input$average == TRUE) { # if the average is selected, it displays average

ggplot(data = bikes\_mean, aes(x = date, y = count)) +

geom\_line(size = .5, alpha = .70, color = "#36648B") +

scale\_x\_datetime(date\_breaks = "2 years", date\_labels = "%Y") +

labs(x = NULL,

y = "Count") +

theme\_minimal(base\_size = 16) +

theme(plot.title = element\_text(hjust = .5),

axis.text.x = element\_text(size = 16),

text = element\_text(family = "Verdana"))

} else { # if average is not selected it is the total

ggplot(data = bikes\_total, aes(x = date, y = count)) +

geom\_line(size = .5, alpha = .70, color = "#36648B") +

scale\_x\_datetime(date\_breaks = "2 years", date\_labels = "%Y") +

labs(x = NULL,

y = "Count") +

theme\_minimal(base\_size = 16) +

theme(plot.title = element\_text(hjust = .5),

axis.text.x = element\_text(size = 16),

text = element\_text(family = "Verdana"))

}

})

output$caption <- renderUI({ # the default caption

str1 <- "Presenting data from bicycle counters across Ottawa. There are 14 counters spread across the city. The graph below displays how daily counts change over time. Click on a map marker to select a specific counter."

HTML(str1)

})

output$stats <- renderUI({ # Statistics output

str1 <- "**Statistics**"

str2 <- paste("Total count: ", format(round(sum(bikes\_total$count, na.rm = TRUE)), big.mark = ","))

str3 <- paste("Average count: ", format(round(mean(bikes\_total$count, na.rm = TRUE), 1), big.mark = ","))

str4 <- paste("Busiest day: ", bikes\_total[which.max(bikes\_total$count),]$date)

HTML(paste(str1, str2, str3, str4, sep = '  
'))

})

})

output$download <- downloadHandler( # download button. Will turn 'bikes' object into a csv file.

filename = function() {

paste("ottawa\_bikes", ".csv", sep = "")

},

content = function(file) {

write.csv(bikes, file)

}

)

}

The code for the server is much busier and it can be overwhelming. Essentially we’re just saying what we want to do with the inputs and outputs. We generate a little code chunk for each output. Look at the first one for *map*. This is where we generate the map. We say we want to *renderLeaflet* and then we just copy the code that we made earlier into this block.

Where things get a bit more complicated is when we want our output to change based on user input. If the user selects the switch that converts the data to averages, for example. I used if and else statements to modulate the output based on whether ‘average’ is selected. What happens, is when the user clicks on the switch, the value of input$average changes to TRUE. Using if and else functions, I just say what I want to happen when ‘average’ is TRUE and what happens if it’s FALSE.

Finally, we want the user to be able to click on specific markers and have the output change to that specific marker. We use the *observeEvent* function and specify the input, ‘map\_marker\_click’. We also want the user to be able to click off the marker to go back to the default output. Again, we use *observeEvent* but now with ‘click\_map’.

Once we have all the other outputs in place for the downloads and the captions, we put it all together using the *shinyApp* function.

shinyApp(ui, server)

And there it is, a user-friendly app for exploring bicycling data in Ottawa. Future avenues include building in some time-series forecasting. It would be cool to show the user how the trend is expected to change over time.