Data Visualisation

Chapter 8: Adding Interactivity

Dr James Baglin

How to use these slides

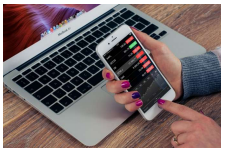
Viewing slides…

- Press ‘f’ enable fullscreen mode
- Press ‘o’ or ‘Esc’ to enable overview mode
- Pressing ‘Esc’ exits all of these modes.
- Hold down ‘alt’ and click on any element to zoom in. ‘Alt’ + click anywhere to zoom back out.
- Use the Search box (top right) to search keywords in presentation

Printing slides…

- Click here to open a printable version of these slides.
- Right click and print from browser or save as PDF (e.g. Chrome)

Getting Interactive



Categories of Interactivity

Introducing Plotly

- **Plotly** - https://plot.ly/
- An online, open source data analysis and visualisation tool
- Takes data visualisations produced using the best scientific languages (R, Matlab, Python, Perl, Julia etc.) and converts them into D3.js interactive, web-based plots.

Introducing Plotly Cont.

- **Data Driven Documents (D3.js)** is a powerful, open source javascript library that uses HTML, SVG and CSS to create highly accessible web-based interactive data visualisations.
- We don't need to learn D3.js because we know R.
- We will use Plotly to translate.
- However, if you go further with data visualisation, you should learn it. I recommend this book:

Interactive Data Visualizations for the Web
by Scott Murray

- Sign-up for a Plotly account - https://plot.ly/

Bicycle Dataset

- The `Bicycle` dataset, downloaded from the data.vic website, contains data recording cycling traffic volume recorded at 21 counter sites within Melbourne between 2005 to 2012. The dataset contains over 57,000 rows of data.

Specifically, the `Bicycle` contains the following variables:

- **Unique_ID**: Self-evident
- **NB_TRAFFIC_SURVEY**: Survey Number
- **NB_LOCATION_TRAFFIC_SURVEY**: Location survey Number
- **Sort Des**: Short Description of the location

Bicycle Dataset Cont.

- **DS_LOCATION**: Location Description
- **DT_ANALYSIS_SUMMARY**: Date
- **NB_YEAR**: Year data collected
- **NB_MONTH**: Month data collected
- **NB_WEEKDAY_NONHOL_QTR**: Holiday period indication
- **CT_VOLUME_AMPEAK**: Max hour in morning peak
- **CT_VOLUME_PMPEAK**: Max hour in evening peak
- **CT_VOLUME_4HOUR_OFFPEAK**: 4 hour off peak volume (12:00 to 4:00 PM)
- **CT_VOLUME_12HOUR**: 12 hour volume (7:00 AM to 7:00 PM)
- **CT_VOLUME_24HOUR**: 24 hour volume

Bicycle Dataset Cont. 2

- **DS_HOLIDAY**: Holiday description
- **NB_SEASONALITY_PERIOD**: Seasonality period indication (1 to 27)
- **NB_TYPE_PERIOD**: Seasonality period type indication (1 to 3)
- **Primary**: Primary site indication (True / False)
- **weekend**: Weekend indication (True / False)
- **Quarter**: Number quarter (1 to 4)
- **Season**: Weather season
- **Cyclying**: Season Cycling season
- **day**: Day of the week

Basic Interactivity

- Let's create a basic times series plot looking at yearly bicycle traffic volume across the Melbourne survey sites.
- We start with some data wrangling…

```
Bicycle <- read.csv("../data/Bicycle.csv")

Bicycle$Season <- Bicycle$Season %>%
  factor(levels = c("Summer", "Autumn", "Winter",
  "Spring"),
         ordered = TRUE)

library(lubridate)

Bicycle$DT_ANALYSIS_SUMMARY <-
  Bicycle$DT_ANALYSIS_SUMMARY %>% dmy()

Bicycle_yearly_season <- Bicycle %>% group_by(Season,
  NB_YEAR) %>%
  summarise(volume = mean(CT_VOLUME_12HOUR, na.rm =
  TRUE))
```

Basic Interactivity Cont.

- Here is a preview of the summarised `data.frame`.

Show 5 entries Search:

	Season		NB_YE
1	Summer	2006	718.157556270097
2	Summer	2007	707.697690217391
3	Summer	2008	750.643852978454
4	Summer	2009	793.476333133613
5	Summer	2010	774.616995810892

Showing 1 to 5 of 28 entries

Previous 1 2 3 4 5 6 Next

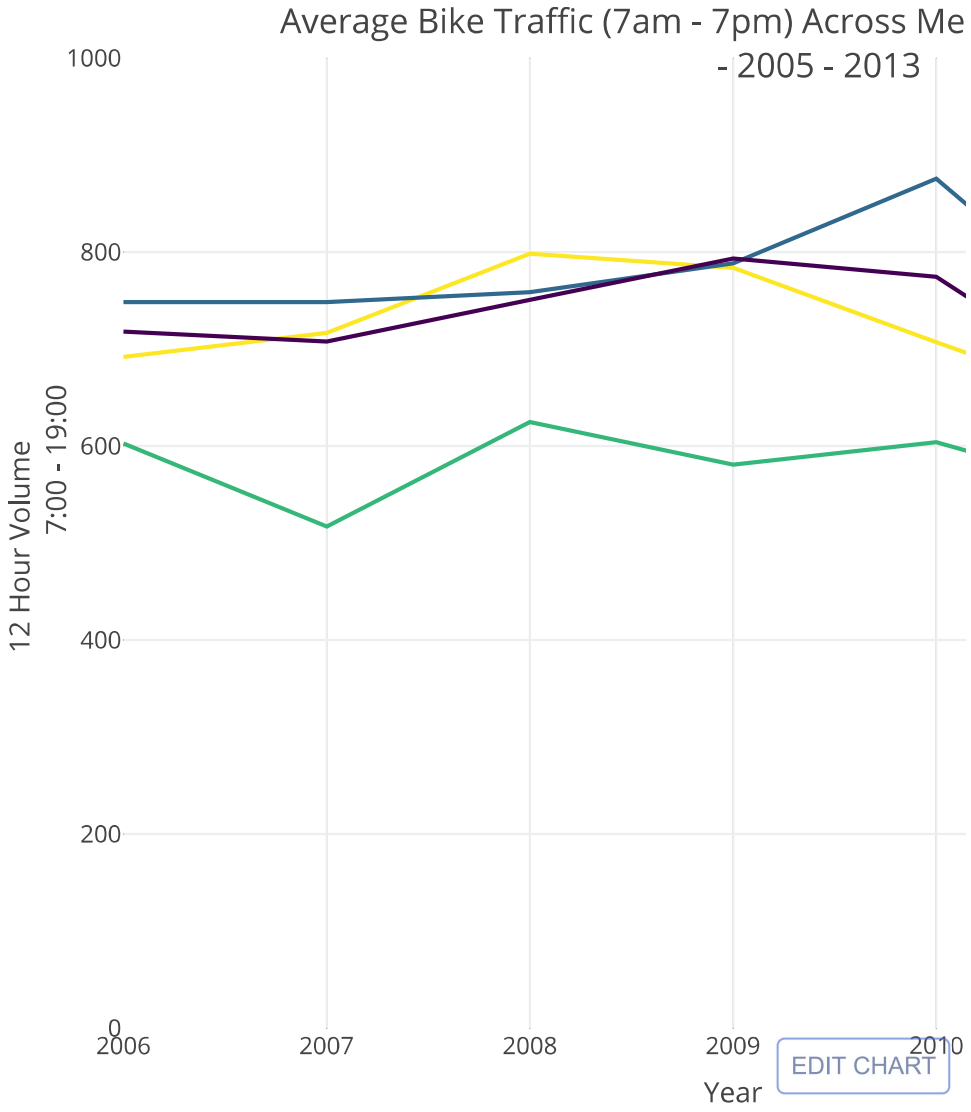
Basic Interactivity Cont. 2

- Now for a simple `plot_ly` visualisation:

```
library(plotly)

p1 <- plot_ly(Bicycle_yearly_season) %>%

  add_lines(x = ~NB_YEAR, y = ~volume, group =
  ~Season,
            color = ~Season) %>%

  layout( title = "Average Bike Traffic (7am - 7pm)
          Across Melbourne by Season - 2005 - 2013",
          yaxis = list(zeroline = FALSE, title = "12
  Hour Volume
                       7:00 - 19:00", range = c(0,
  1000)),
          xaxis = list(zeroline = FALSE, title =
  "Year"))
```

Basic Interactivity Cont. 3

- and the result...



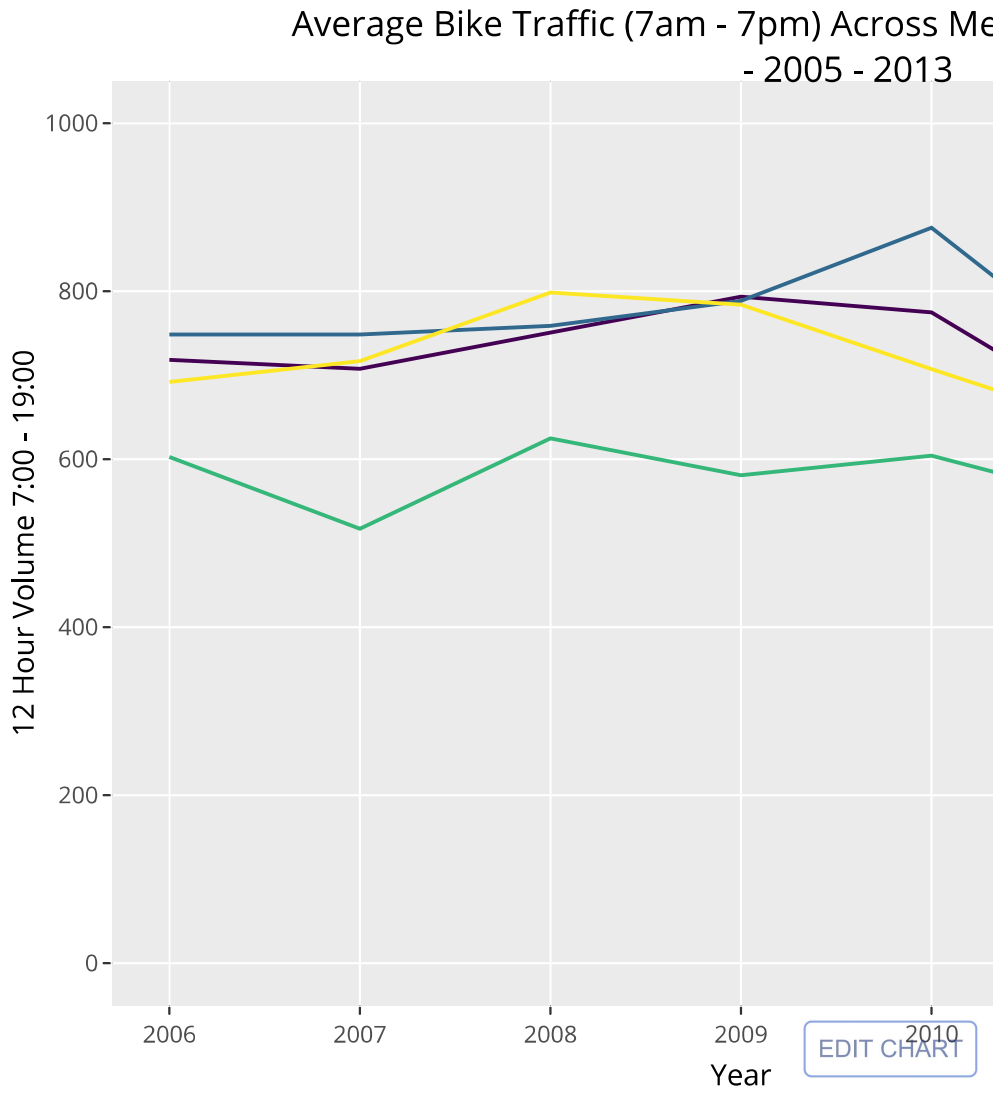
ggplotly

- You can also easily convert ggplots to plot_ly using the ggplotly() function.

```
Bicycle_yearly_season <- Bicycle_yearly_season %>%
  ungroup()

p2 <- ggplot(data = Bicycle_yearly_season,
             aes(x = NB_YEAR, y = volume, colour =
  Season,
                 group = Season)) + geom_line() +
  labs(x = "Year",
       y = "12 Hour Volume 7:00 - 19:00",
       title = "Average Bike Traffic (7am - 7pm)
  Across Melbourne by Season
         - 2005 - 2013") +
  scale_y_continuous(limits = c(0,1000))

p2 <- ggplotly(p2)
```

`ggplotly` Cont.

- and the result...



Animation

- Let's create an animation showing how the weekly bike traffic volume changes across time. First, a bit of data wrangling:

```
library(lubridate)

Bicycle <- Bicycle %>%
  mutate(week =
  as.numeric(paste0(year(Bicycle$DT_ANALYSIS_SUMMARY),
        formatC(week(Bicycle$DT_ANALYSIS_SUMMARY),
  width = 2,
                format = "d", flag = "0"))))

Bicycle_filt <- Bicycle %>%
  filter(year(DT_ANALYSIS_SUMMARY) == 2012)

Bicycle_sum <- Bicycle_filt %>% group_by(week,day)
  %>%
   summarise(volume = mean(CT_VOLUME_12HOUR, na.rm =
  TRUE))
```

Animation Cont.

- Here a preview of the data:

Show 5 entries Search:

	week	day	
1	2012.01	Fri	697.307692307692
2	2012.01	Mon	212.756756756757
3	2012.01	Sat	413.25641025641
4	2012.01	Sun	196.710526315789
5	2012.01	Thu	760.487179487179

Showing 1 to 5 of 366 entries

Previous 1 2 3 4 5 … 74 Next

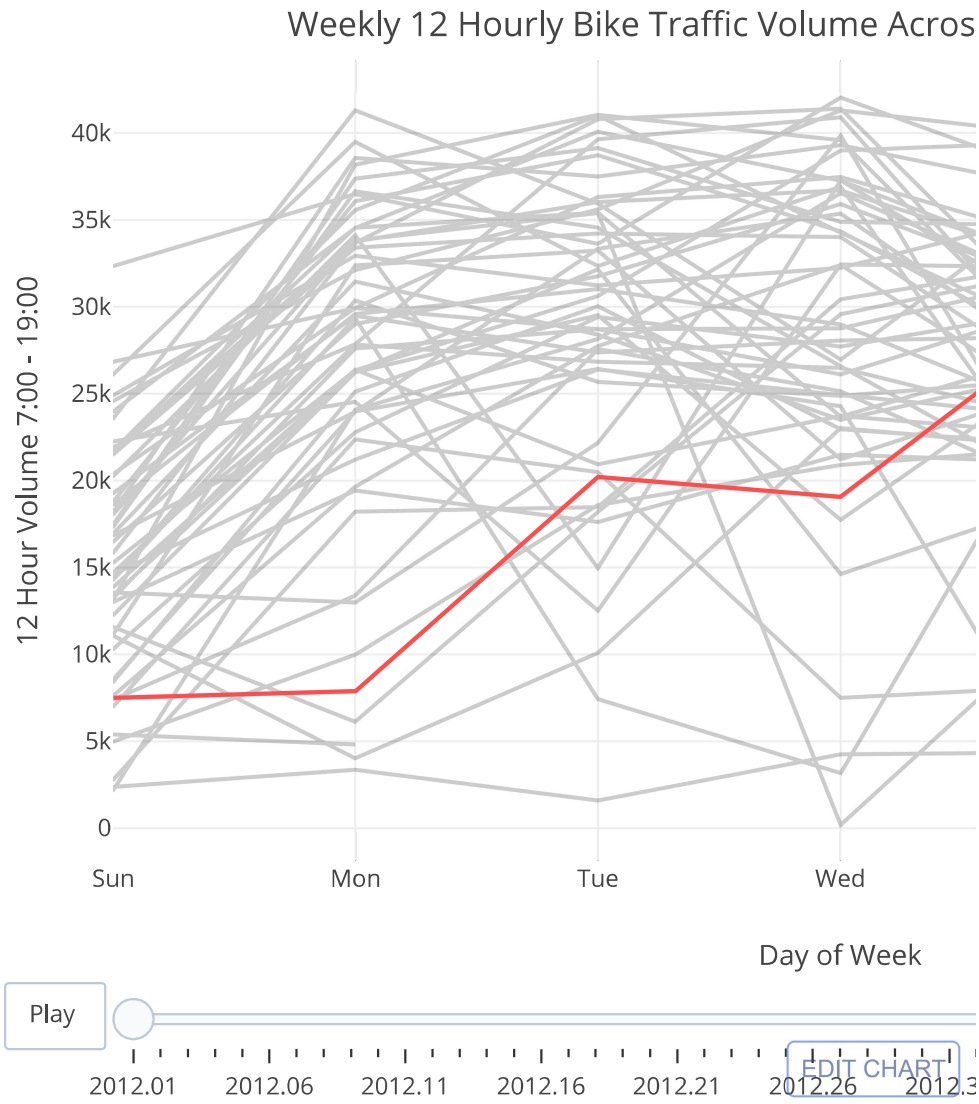
Animation Cont. 2

Now for the plot…

```
p3 <- plot_ly(Bicycle_sum) %>%

  add_lines(x = ~day, y = ~volume, group = ~week,
  opacity = .2,
            showlegend = FALSE) %>%

  add_lines(x = ~day, y = ~volume, frame = ~week,
  showlegend = FALSE,
            line = list(color = "#FF4C4C")) %>%

  layout( title = "Average Bike Traffic (7am - 7pm)
          Across Melbourne by Week",
          yaxis = list(zeroline = FALSE, title = "12
  Hourly
                       Volume 7:00 - 19:00", range =
  c(0,1200)),
```

Animation Cont. 3

- and the result...



Ghosting (Experimental!)

- Can we use the idea of cummulative animations to create ghosting in our Plotly animations?
- Ghosting is a great way to track the change and path of features across the frames.
- We create a series of the lagged datasets for ghosting…
- We will use a four week ghost…

Ghosting (Experimental!) Cont.

```
lag_1_week <- Bicycle_sum
lag_1_week$week <- lag_1_week$week + .01
lag_1_week<-rbind(Bicycle_sum[1:7,],lag_1_week)

lag_2_week <- Bicycle_sum
lag_2_week$week <- lag_2_week$week + .02
lag_2_week<-rbind(Bicycle_sum[1:14,],lag_2_week)

lag_3_week <- Bicycle_sum
lag_3_week$week <- lag_3_week$week + .03
lag_3_week<-rbind(Bicycle_sum[1:21,],lag_3_week)

lag_4_week <- Bicycle_sum
lag_4_week$week <- lag_4_week$week + .04
lag_4_week<-rbind(Bicycle_sum[1:28,],lag_4_week)
```

Ghosting (Experimental!) - Cont. 2

- Now for the visualisation.

```
p4 <- plot_ly() %>%

  add_lines(x = ~day, y = ~volume, opacity = .1,
            data = Bicycle_sum,
            showlegend = FALSE,
            line = list(color = "#808080 ")) %>%

  add_lines(x = ~day, y = ~volume, frame = ~week,
            data = lag_4_week,
            showlegend = FALSE,
            line = list(color = "#fee5d9")) %>%

  add_lines(x = ~day, y = ~volume, frame = ~week,
            data = lag_3_week,
            showlegend = FALSE,
            line = list(color = "#fcae91")) %>%
```

Ghosting (Experimental!) - Cont. 3

- Does it work….

Custom Controls - Range Selectors

- We can also add a useful range selector….
- First, let's visualise the data as a single time series.

```
Bicycle_time_series <- Bicycle %>%
  group_by(DT_ANALYSIS_SUMMARY) %>%
   summarise(volume12 = sum(CT_VOLUME_12HOUR, na.rm =
  TRUE),
             volume24 = sum(CT_VOLUME_24HOUR, na.rm =
  TRUE))
```

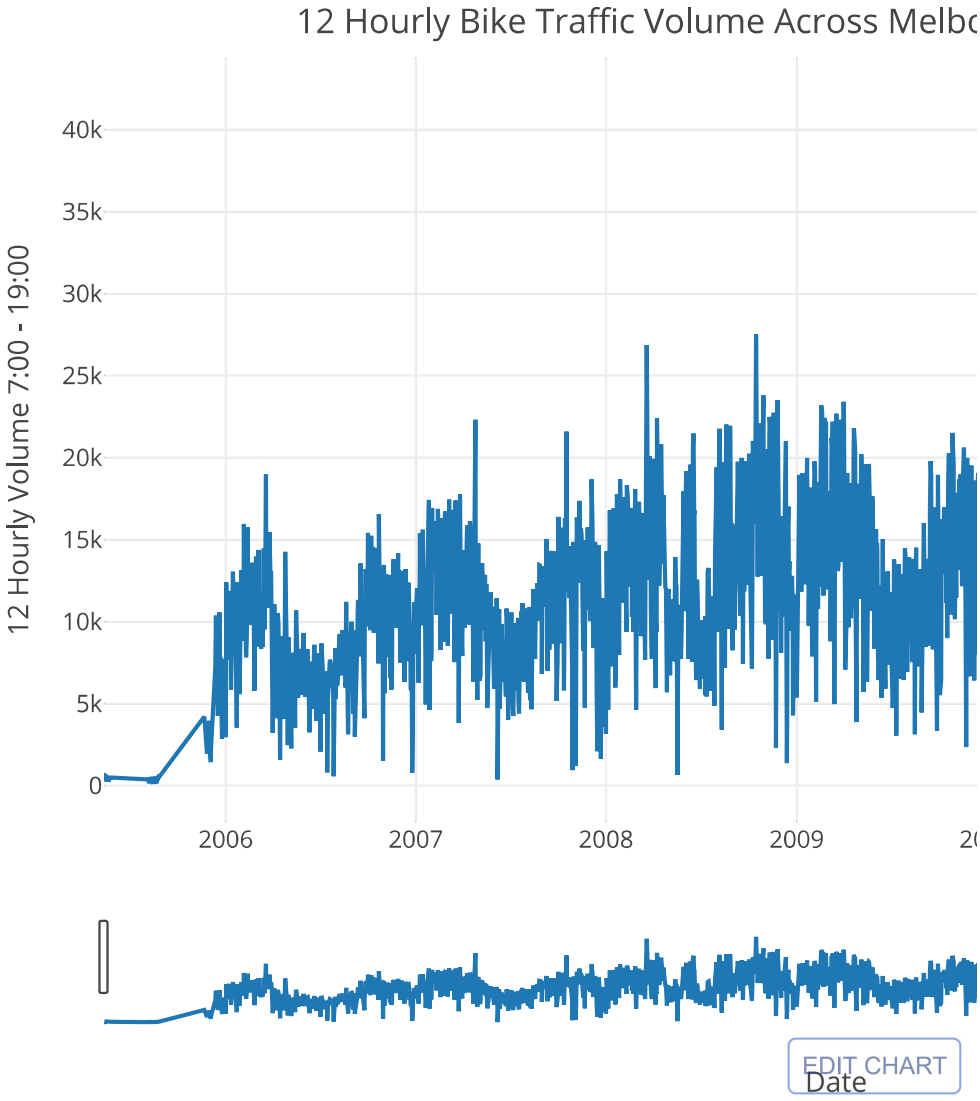
Range Selectors Cont.

- Now we add a range selector using the `rangeslider` option.

```
p5 <- plot_ly(Bicycle_time_series) %>%

  add_lines(x = ~DT_ANALYSIS_SUMMARY, y = ~volume12,
            showlegend = FALSE) %>%

  layout( title = "12 Hourly Bike Traffic Volume
  Across Melbourne
          - 2005 - 2013",
          yaxis = list(zeroline = FALSE, title = "12
  Hourly Volume
                       7:00 - 19:00"),
          xaxis = list(zeroline = FALSE, title =
  "Date",
                       rangeslider = list(type =
  "date")))
```

Range Selectors Cont. 2

- and the result...



Custom Controls - Buttons

- We can add buttons as well. For example, what if we want to look at 24 hourly volume or 12 hourly volume?
- First we define a basic menu:

```
updatemenus <- list(
  list(
    active = 0, x = -.125, type= 'buttons',
    buttons = list(
      list(
        label = "12 Hourly",
        method = "update",
        args = list(list(visible = c(TRUE,
  "legendonly")))),
      list(
        label = "24 Hourly",
        method = "update",
        args = list(list(visible = c("legendonly",
  TRUE))))
    )
  )
)
```

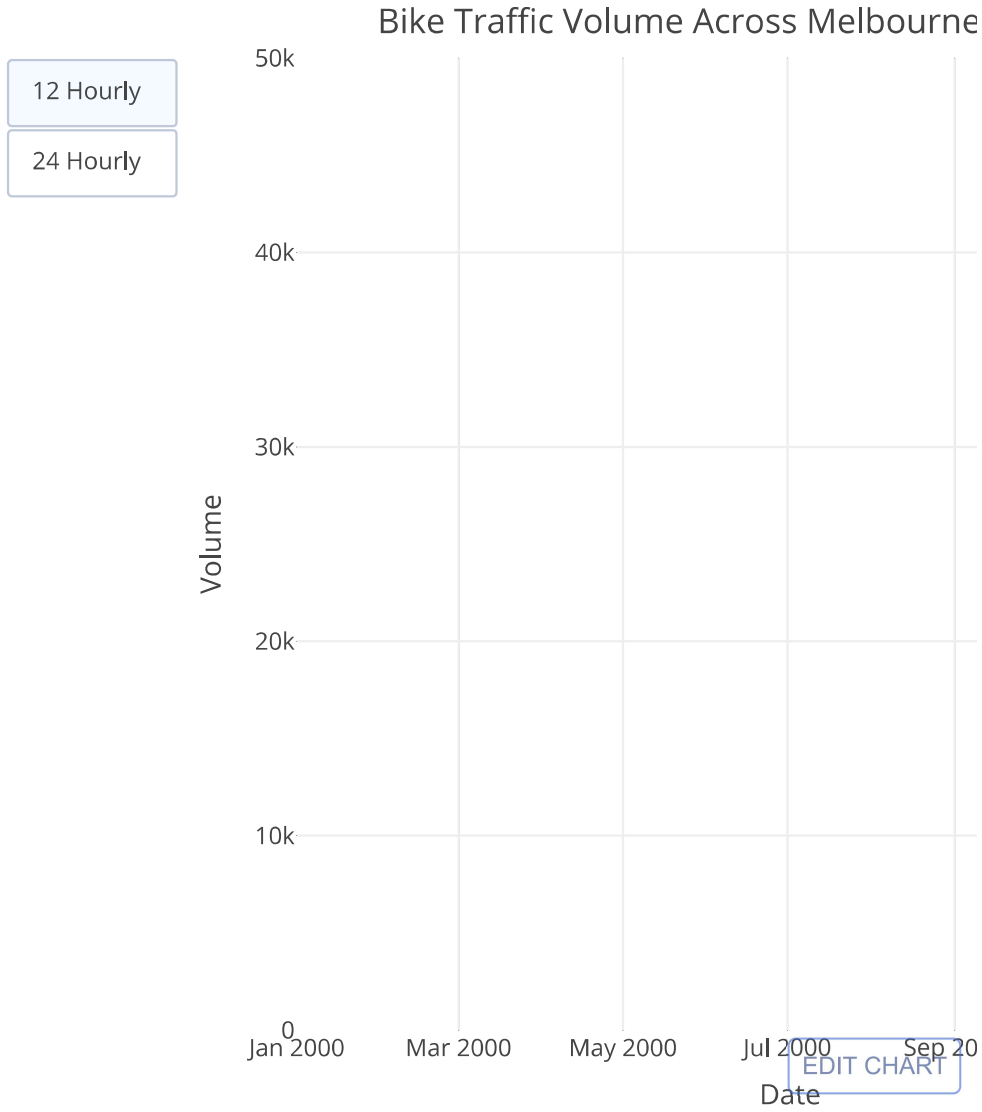
Buttons Cont.

- Now we add updatemenus to a plot:

```
p6 <- plot_ly(Bicycle_time_series) %>%

  add_lines(x = ~DT_ANALYSIS_SUMMARY, y = ~volume12,
            visible = "legendonly", name = "7:00 -
  19:00") %>%

  add_lines(x = ~DT_ANALYSIS_SUMMARY, y = ~volume24,
            visible = "legendonly", name = "00:00 -
  24:00") %>%

  layout( title = "Bike Traffic Volume Across
  Melbourne - 2005 - 2013",
          yaxis = list(zeroline = FALSE, title =
  "Volume",
                       range = c(0, 50000)),
          xaxis = list(zeroline = FALSE, title =
```

Buttons Cont. 2

- and the result...



Concluding Caution

- Adding interactive features to data visualisations is lots of fun, but always keep the following in mind:
  - The interactivity should always enhance and never detract.
  - Even modest interactivity usually requires extra time to design, code and deploy.
  - Interactivity adds many technological requirements such a web access and servers.

References

Buja, A., J. A. McDonald, J. Michalak, and W. Stuetzle. 1991. “Interactive data visualization using focusing and linking.” In *VIS ’91 Proceedings of the 2nd Conference on Visualization ’91*, edited by G. M. Nielson and L. Rosenblum, 156–63. San Diego, California. https://dl.acm.org/citation.cfm?id=949633.

Kirk, A. 2012. *Data visualization: a successful design process*. Birmingham, UK: Packt Publishing Ltd.

Murray, S. 2013. *Interactive data visualization for the web*. Sebastopol, CA: O’Reilly Media, Inc.