

Plotly & ggplotly in R

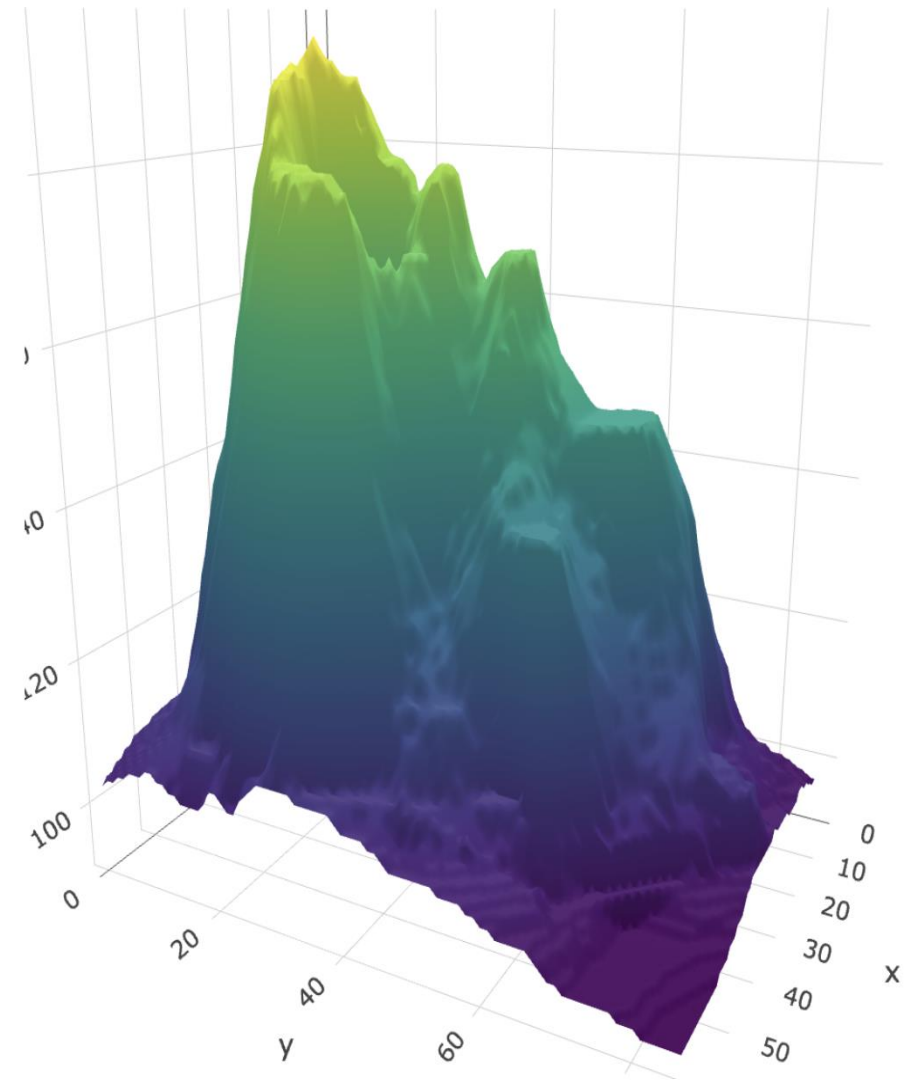
Classroom demonstration: interactive charts, functions, and practice tasks

R + plotly

ggplotly bridge

Hands-on demo

Jagarwal Rajani –Maria Kanterina
Statistics, Visualization and More Using "R"



Introduction

We will start with a small example. Then add one concept at a time: mapping, trace, layout, conversion, and combinations.

Why choose Plotly?

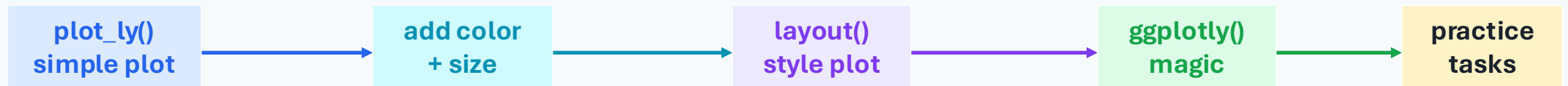
- **Digital & Online Format** - Native HTML rendering.
- **Interactivity** - Hover for info, zoom in on data points.
- **Beyond ggplot2** - Advanced web customization.
- **High Resolution** - Export as SVG or PNG.
- **Collaboration** - Same ecosystem in R as Python/JS .
- **Unusual Charts** - Native support for 3D and Maps.

Student goal

- **Developing interactive graph with plotly**
- **Understand various functions and their arguments**

Dataset

- mtcars
- Iris
- economic



Before the demo: install and load packages

```
install.packages("plotly")  
install.packages("ggplot2")
```

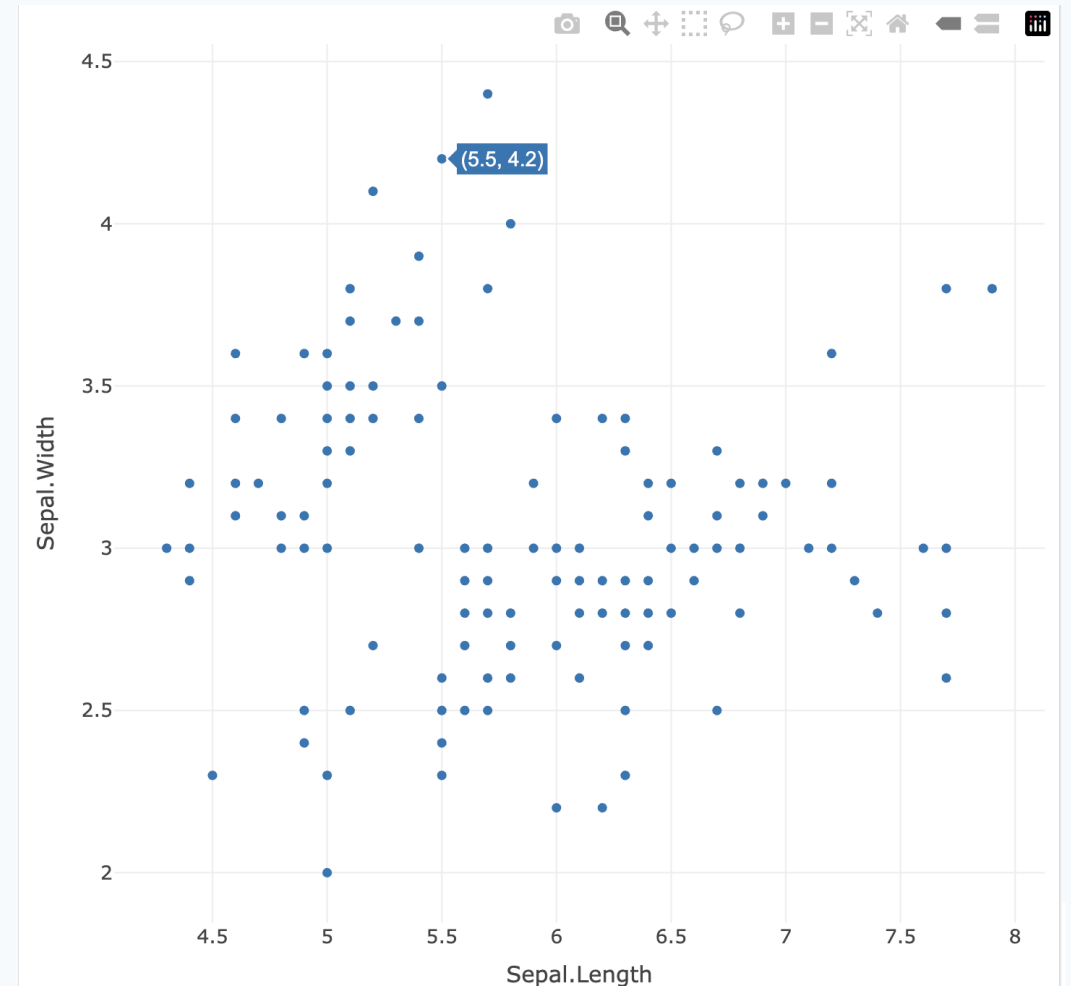
```
library(plotly)  
library(ggplot2)
```

Comment

- plotly gives us `plot_ly()`, `ggplotly()`, `subplot()`, `config()`, and many `add_*` functions.
- `ggplotly()` belongs to the plotly package. `ggplot2` creates static plots that `ggplotly` can convert.
- Demo on Rstudio

Dataset

- **iris** with columns (Sepal.Length Sepal.Width Petal.Length Petal.Width Species)



```
plot_ly(iris, x = ~Sepal.Length, y = ~Sepal.Width)
```

Core Plotly functions in R

A compact function map

Function	Purpose	Comments
plot_ly()	Entry point to generate the plots	Create the canvas
add_trace() add_lines() add_bars() add_histogram() add_boxplot() Add_heatmap()	Add any type of plot layer like type, mode Add line chart Add bar chart (needs y) Add histogram Add boxplot Add heatmap	Add another visual element
add_markers()	Add points for a scatter plot	Shortcut for scatter markers
layout()	Style plot	Title, axes, legend, hover mode
ggplotly()	Convert ggplot	Static ggplot → interactive chart
subplot()	Combine plots	Multiple charts in one view
config()	Control UI	Toolbar, export, scroll zoom
add_surface()	3D plots	

plot_ly(): Generate an interactive plot

Use it when you want to build a Plotly graph from scratch

Basic Syntax

```
plot_ly(data = <dataset>, x = ~<x_variable>, y = ~<y_variable>,  
        type = <plot_type>, mode = <mode>)
```

Key Idea: Formula Notation (~)

`x = ~Sepal.Length`

data

- The dataset you are using

x and y

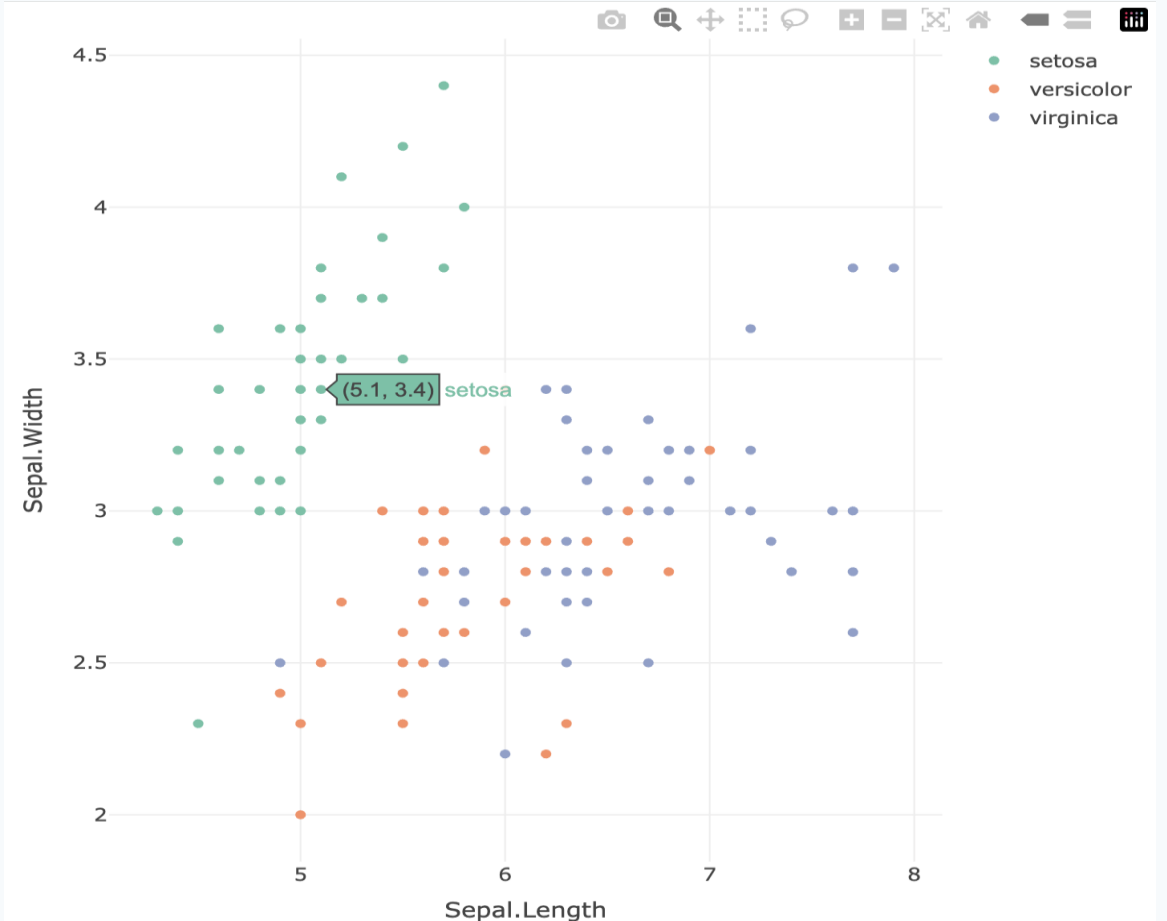
- Variables mapped to axes

type

- "scatter"
- "bar"
- "histogram"
- "box"

mode (only for scatter type) - Controls how data is displayed

- "markers" → points
- "lines" → line plot
- "lines+markers" → both



Demo on RStudio

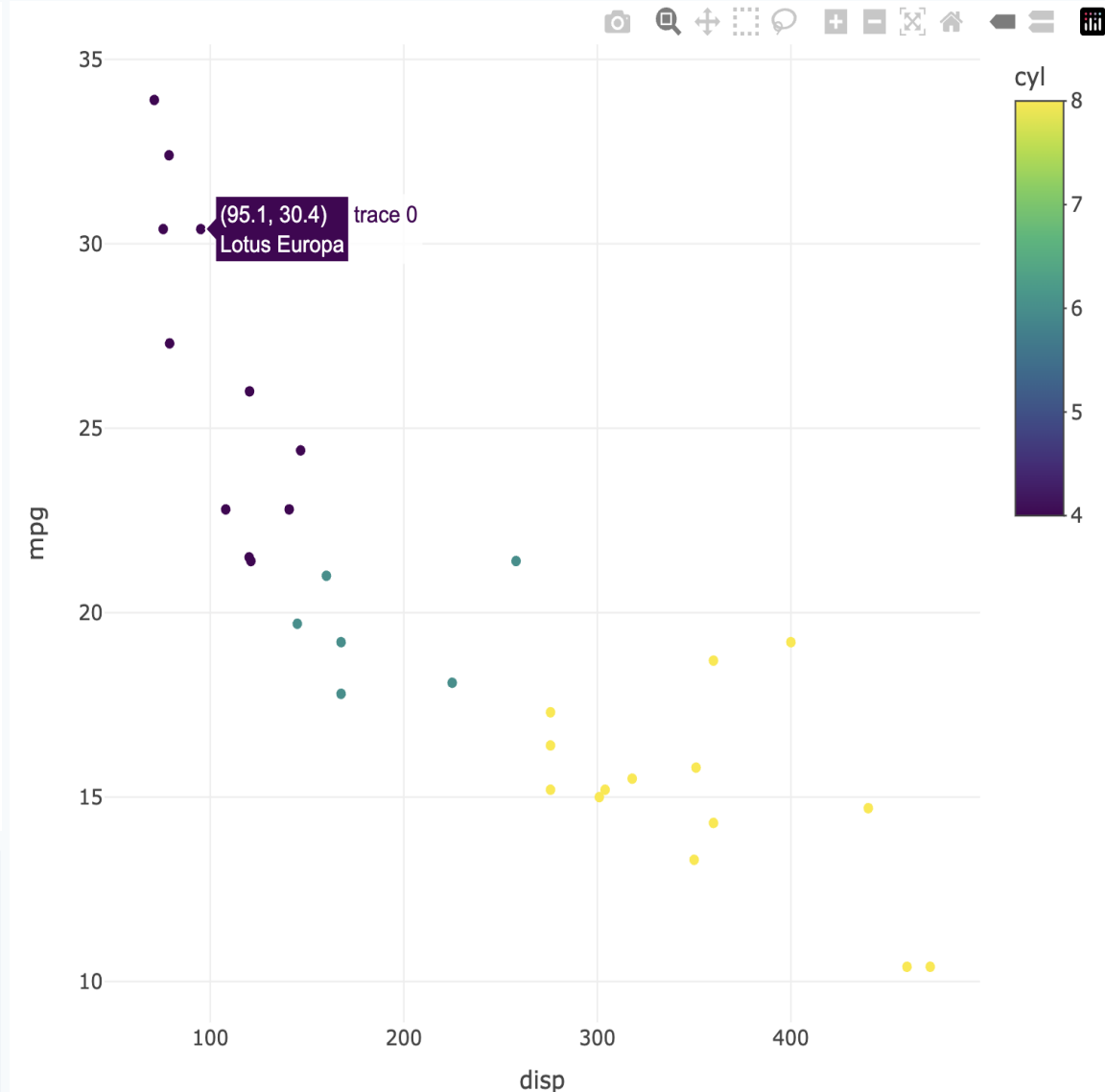
- Complete Scatter Plot
- Adding Color (Grouping)
- Size & Hover Text

plot_ly(): Exercise 1

```
# Exercise 1
library(tidyverse)
mtcars <- mtcars %>% as_tibble(rownames="cars")
head(mtcars)

plot_ly(mtcars,
        x=~<...>,
        y=~<...>,
        color=~cyl,
        type="scatter",
        text=~<...>,
        mode = "markers")
```

Create a scatter plot using mtcars (disp vs mpg) , when hover on the dots shows the name of the car.



add_trace() : Add layers

Plotly is modular: each visual layer is a trace

`add_trace()` is a **general-purpose function** used to add *any type of plot layer* to a Plotly graph.

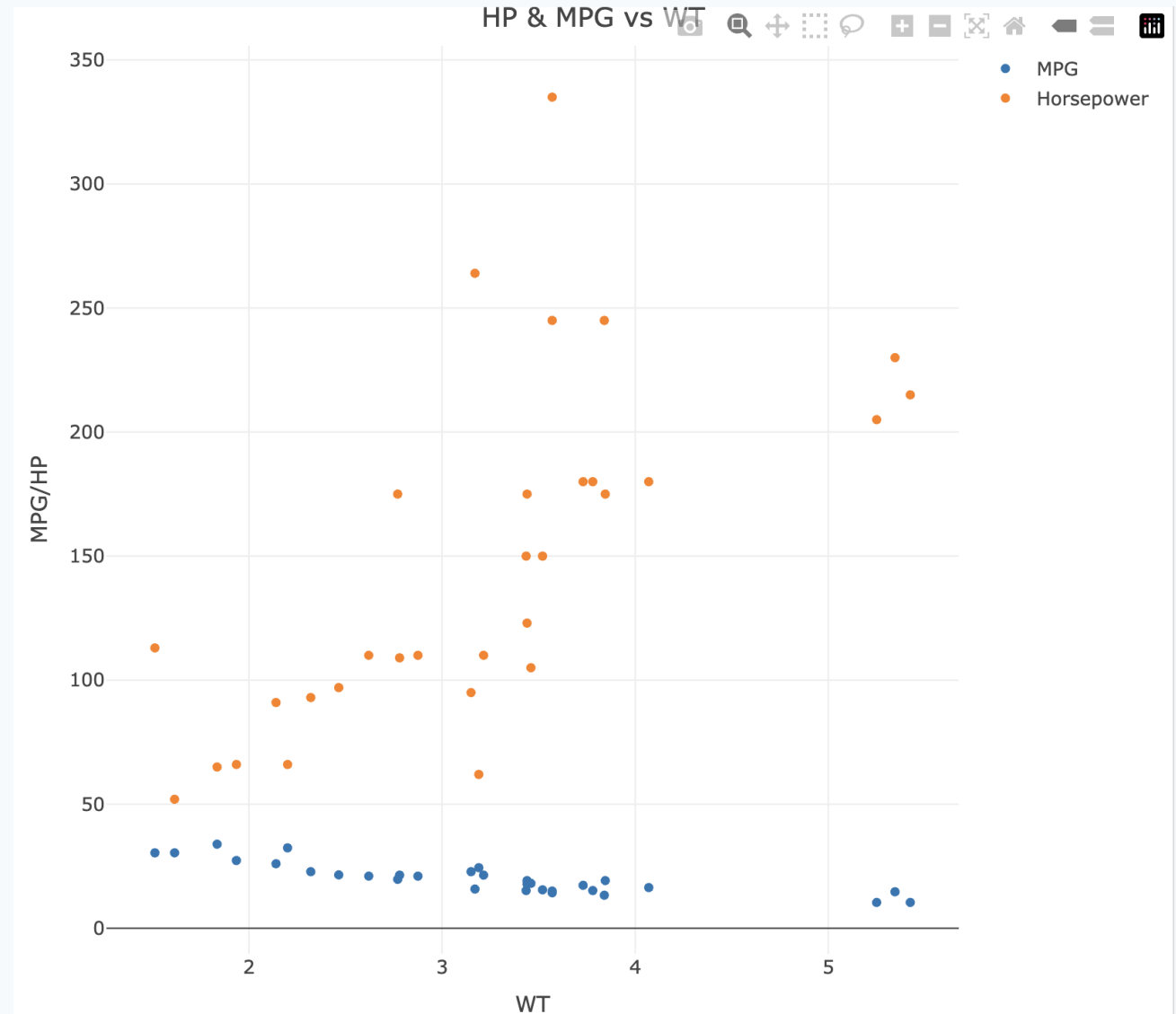
All specialized functions like:

- `add_markers()`
- `add_lines()`
- `add_bars()`

are actually **wrappers** around `add_trace()`

Basic Syntax

```
plot_ly(data = , x = ~x, y = ~y) %>%  
  add_trace(type = <...>, mode = <...>) %>%  
  add_trace(type = <...>, mode = <...> )
```



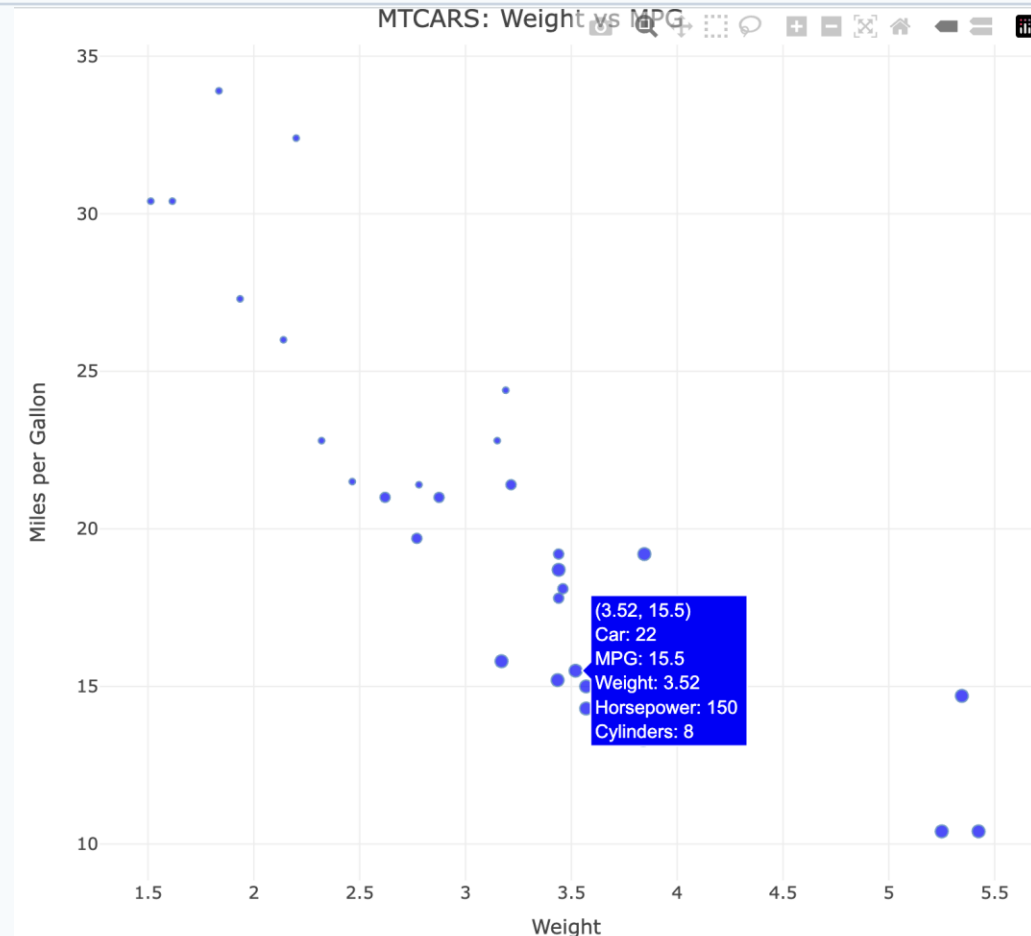
Specialized add_* functions (Part 1)

Add_markers and add-lines

add_markers() → scatter plot

Important Parameters

- color = ~var → grouping
- size = ~var → point size
- text → hover info
- marker = list(...) → styling



add_lines() → line chart

Important Parameters

- line = list(color, width)
- name → legend label

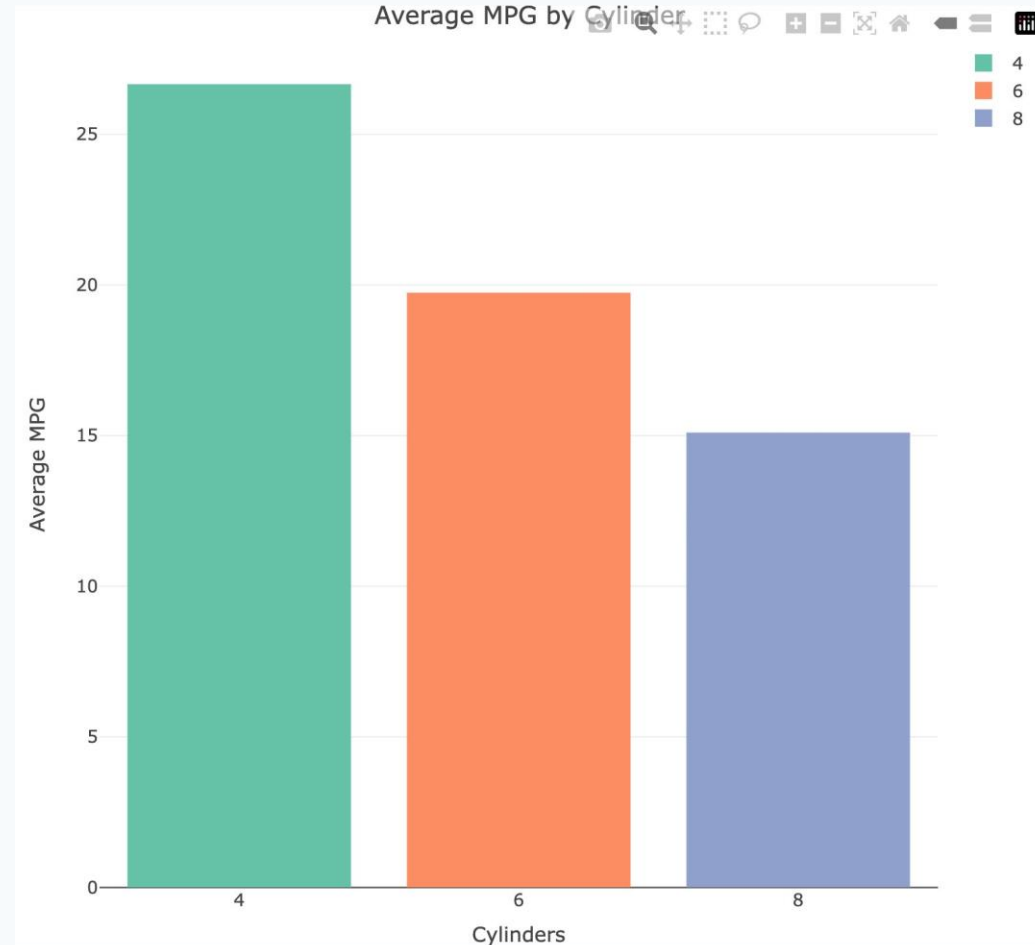
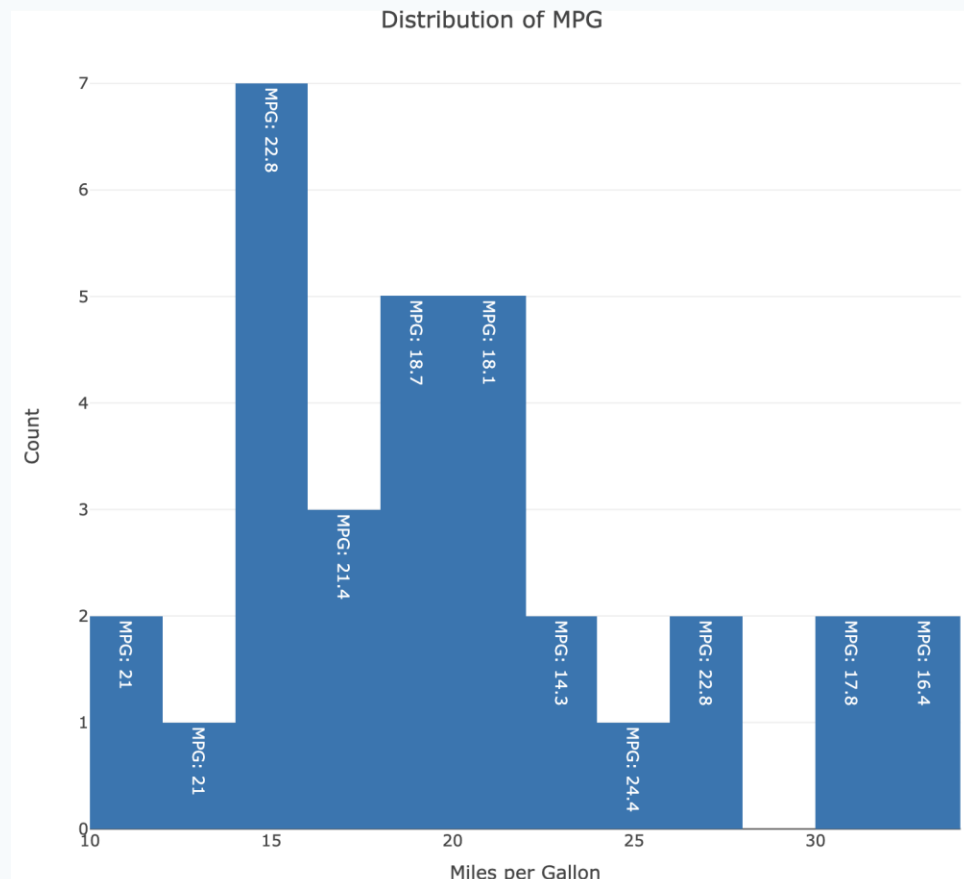
Specialized add_* functions (Part 2)

Add_bars and add_histogram

`add_bars()` → bar chart

Important Parameters

- `color` → grouped bars
- `marker` → styling
- `name` → legend



`add_histogram()` → histogram

Important Parameters

- `nbinsx` → number of bins
- `marker` → color

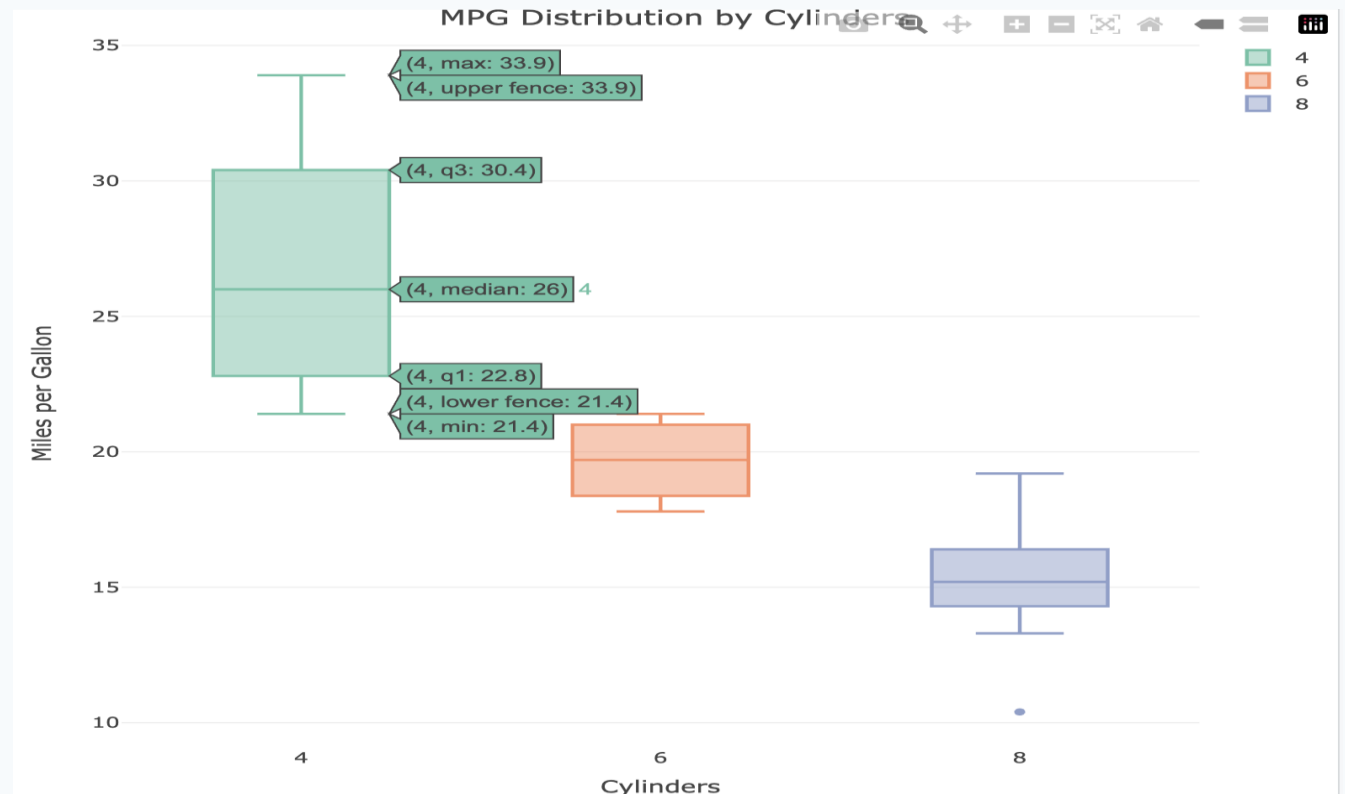
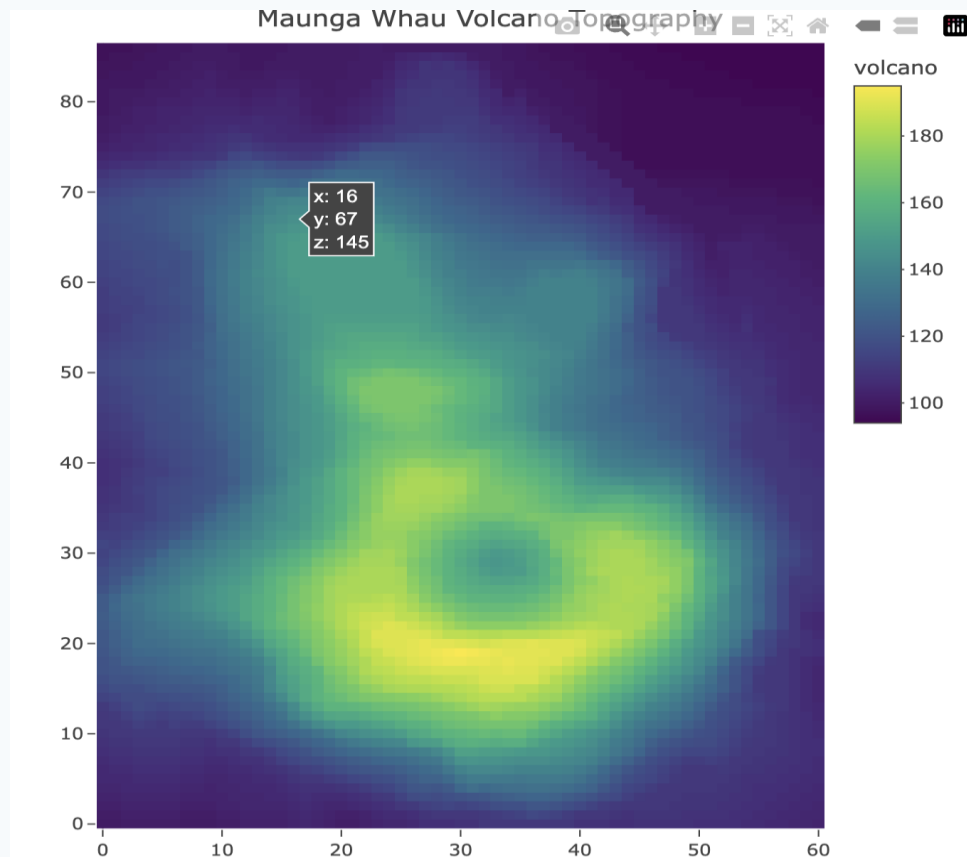
Specialized add_* functions (Part 3)

Add_boxplot

add_boxplot() → boxplot

Important Parameters

- color → grouping
- boxpoints → show individual points



add_heatmap() → boxplot

Important Parameters

- Z → mandatory
- Colorscale → "Viridis", "Hot", "Greys", "Electric", or "Portland"

layout(): style the plot

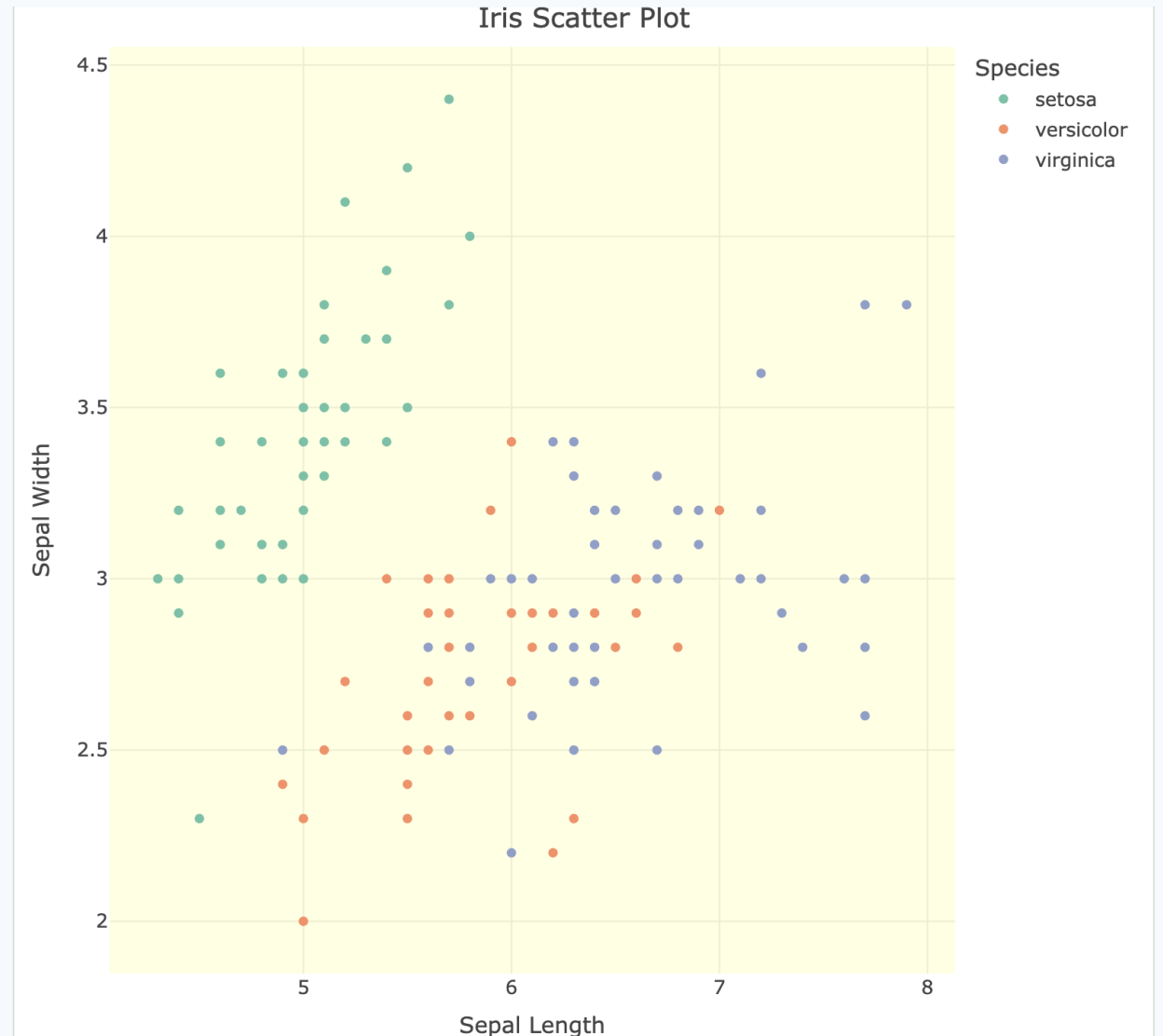
Use layout() for title, axes, legend, hover behavior, and margins

Use layout() when you need...

- clear title
- understandable axis labels
- better legend position
- hover behavior
- dashboard-ready formatting

Key Components

- Title – adds main title
- Legend
- Axis -
 - range → control axis limits
 - showgrid → grid lines
 - zeroline → zero axis
 - rangeslider -> changes the range
- Colors & Background
- Annotations (Text on plot)
- Margins

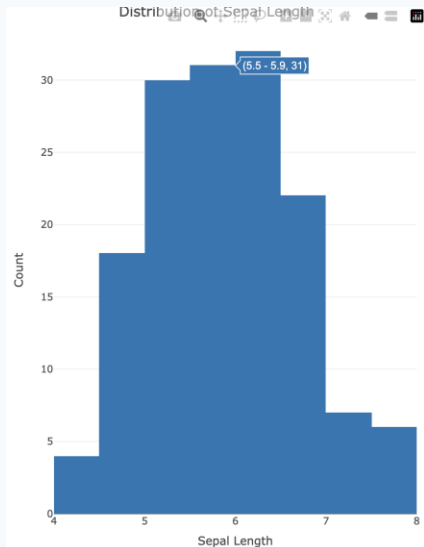


histogram() & boxplot(): Exercise 2

1. Create a histogram of Sepal.Length with bin 15
2. Create a boxplot of Sepal.Length grouped by Species

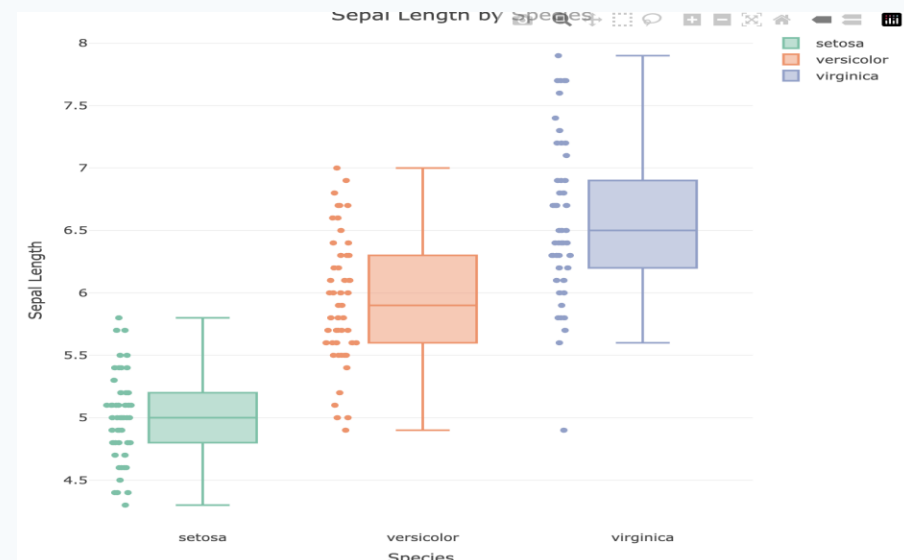
```
plot_ly(iris, x = ~Species, y = ~<...>) %>%  
  add_<...>(  
    color = ~<...>,  
    boxpoints = "all"  
  ) %>%  
  layout(  
    title = "Sepal Length by Species",  
    xaxis = list(title = "Species"),  
    yaxis = list(title = "Sepal Length")  
  )  
|
```

add_histogram() & add_boxplot(): Exercise 2 - Answers



```
plot_ly(iris, x = ~Sepal.Length) %>%  
  add_histogram(nbinsx = 15) %>%  
  layout(  
    title = "Distribution of Sepal Length",  
    xaxis = list(title = "Sepal Length"),  
    yaxis = list(title = "Count")  
  )
```

```
plot_ly(iris, x = ~Species, y = ~Sepal.Length) %>%  
  add_boxplot(  
    color = ~Species,  
    boxpoints = "all"  
  ) %>%  
  layout(  
    title = "Sepal Length by Species",  
    xaxis = list(title = "Species"),  
    yaxis = list(title = "Sepal Length")  
  )
```



ggplotly(): static ggplot → interactive

Best when students already know ggplot2

```
# Install packages
install.packages("plotly")
install.packages("ggplot2")

# Load libraries
library(plotly)
library(ggplot2)

# Create static ggplot
p <- ggplot(mtcars,
  aes(x = wt,
      y = mpg,
      color = factor(cyl),
      text = paste("HP:", hp,
                   "<br>Gear:", gear))) +
  geom_point(size = 3) +
  labs(
    title = "Static ggplot → Interactive Plotly",
    x = "Weight",
    y = "Miles per gallon",
    color = "Cylinders"
  )

# Convert to interactive plot
ggplotly(p, tooltip = "text")
```

ggplot2
static



ggplotly()

Plotly
interactive

Comment

“ggplotly() is a bridge: we keep the ggplot grammar but get hover, zoom, and interactivity.”

Limitation

Very complex ggplot themes, annotations, or facets may not translate perfectly.

Excercise 3 — ggplotly ()

Build a ggplot scatter plot and convert it with ggplotly().

subplot(): combine multiple Plotly charts

Useful for dashboard-style comparisons

```
# Shared data
d <- SharedData$new(mtcars)

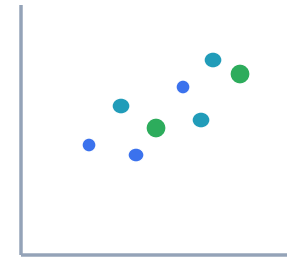
# Scatter
p1 <- plot_ly(
  d,
  x = ~wt,
  y = ~mpg,
  type = "scatter",
  mode = "markers"
)

# Histogram
p2 <- plot_ly(
  d,
  x = ~mpg,
  type = "histogram"
)

# Combine + link
subplot(p1, p2) %>%
  highlight(on = "plotly_selected")
```

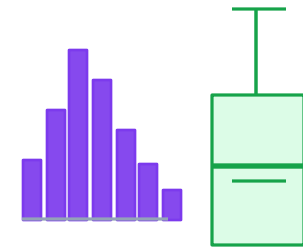
subplot(): one canvas, multiple views

Output idea: interactive scatter



hover • zoom • select

Distribution views



Use case

Put a relationship plot and a distribution plot side by side. This helps students see both pattern and spread.

Exercise 4 — subplot ()

Combine one scatter plot and one histogram using subplot().

config()

Advanced but important for dashboards

```
# Control the Plotly toolbar and behavior
config(
  p,
  displayModeBar = TRUE,
  scrollZoom = TRUE,
  toImageButtonOptions = list(format = "png")
)
```

config()

Controls user interface options, such as toolbar visibility, export buttons, and scroll zoom:

- toolbar visible
- zoom with mouse wheel
- save as PNG

Comment

Plotly is not only a chart library, but an interaction layer.

add_surface()

3D surface plots

```
# Create grid
x <- seq(-2, 2, length.out = 50)
y <- seq(-2, 2, length.out = 50)

# Create Z values (matrix!)
z <- outer(x, y, function(x, y) x^2 + y^2)

# Plot surface
plot_ly(
  x = x,
  y = y,
  z = z
) %>%
add_surface()
```

add_surface() is used to create 3D surface plots in Plotly.

- rotate the surface
- zoom in/out
- explore from different angles

Comment

Surface plots allow us to visualize how a variable changes depending on two other variables.

x, y → coordinates

z → height values

outer() → creates a matrix for the surface

Key takeaways

End with the mental model

“Use Plotly when you want full control over interactivity, and ggplotly when you want a quick upgrade from static to interactive.”



Plotly can generate interactive graphs directly.

`ggplotly()` converts static `ggplot2` plots into interactive Plotly objects.

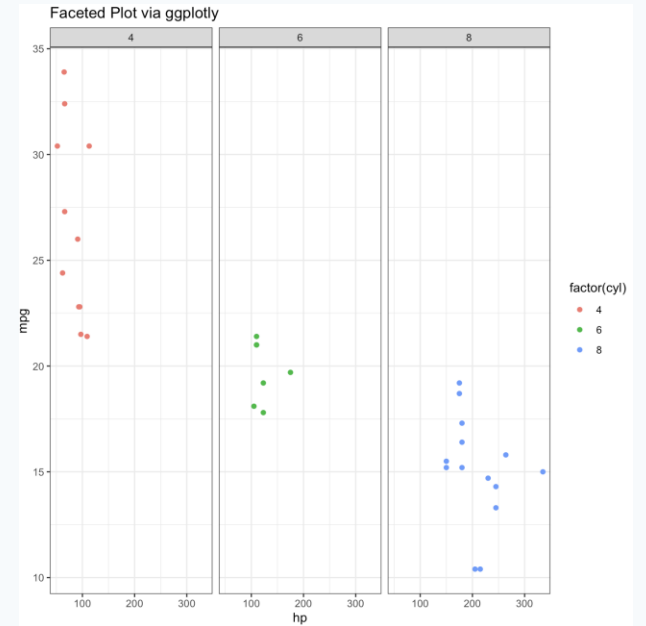
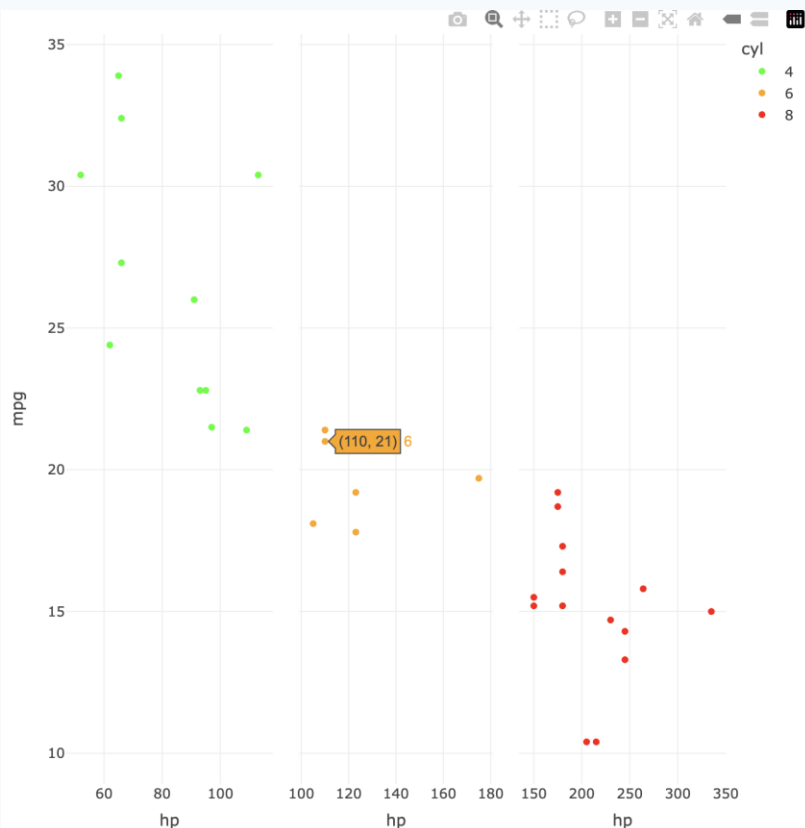
`subplot()` combines multiple charts into one teaching or dashboard view.

`config()` control interaction and interface behavior.

Facet_wrap mimic in plotly

```
# Create the ggplot object
p <- ggplot(mtcars, aes(x = hp, y = mpg, color = factor(cyl))) +
  geom_point() +
  facet_wrap(~cyl) +
  labs(title = "Faceted Plot via ggplotly") +
  theme_bw()
```

p



```
# Method 2 : Without ggplotly
plots <- mtcars %>%
  group_split(cyl) %>%
  lapply(function(df) {
    plot_ly(df, x = ~hp, y = ~mpg, type = "scatter",
            color = ~as.factor(cyl),
            colors = c("4" = "green", "6" = "orange", "8" = "red"),
            mode = "markers") %>%
    layout(legend = list(
      title = list(text = "cyl"),
      orientation = "v"
    ))
  })
```

```
# 2. Use subplot to arrange them
subplot(plots, nrow = 1, shareX = TRUE, shareY = TRUE)
```