QTM 151

Week 3 – plotly

Umberto Mignozzetti
Feb 12

Recap

- Last week we learned `qplot`. It is a great function simple dataviz.
- Do you have any questions about `qplot`?
- We also discussed about GitHub. Did you get an account?
- I also learned that some of you have difficulty to download data from GitHub. I posted a video to help with this.
- The quiz will be posted today, at 4:00PM
- Our GitHub page is: https://github.com/umbertomig/qtm151

Today's Agenda

- Plotly
- Creating `plotly` from ggplot graphs
- Using the `plotly` interactive device

plotly

plotly

- `plotly` is an R package for creating interactive web-based graphs.
- It is based on `plotly.js`, a javascript to plot interactive graphs in R and Python.
- They are great for presentations and for displaying data for a non-stats crowd.

plotly

- There are two functions that you can use to create a plotly:
    - The `plot_ly()` function, that transforms data into a `plotly` object.
    - The `ggplotly()` function, that transforms a ggplot object into a `plotly` object.
- Regardless of how a `plotly` object is created, printing it results in an interactive web-based visualization enabled by default.

Getting Started: loading packages

```r
# Loading tidyverse
library(tidyverse)

# Loading plotly
library(plotly)
```

```
##
## Attaching package: 'plotly'

## The following object is masked from 'package:ggplot2':
##
##     last_plot

## The following object is masked from 'package:stats':
##
##     filter

## The following object is masked from 'package:graphics':
```

Loading datasets

```r
# Loading tips dataset
tips ← read.csv('https://raw.githubusercontent.com/umbertomig/qtm
head(tips, 2)
```

```
##   obs totbill  tip sex smoker day  time size
## 1   1   16.99 1.01   F     No Sun Night    2
## 2   2   10.34 1.66   M     No Sun Night    3
```

```r
# Loading PErisk dataset
PErisk ← read.csv('https://raw.githubusercontent.com/umbertomig/c
head(PErisk, 2)
```

```
##     country courts      barb2 prsexp2 prscorr2    gdpw2
## 1 Argentina      0 -0.7207754       1         3  9.69017
## 2 Australia      1 -6.9077550       5         4 10.30484
```

```r
# Load gapminder
```

Loading datasets

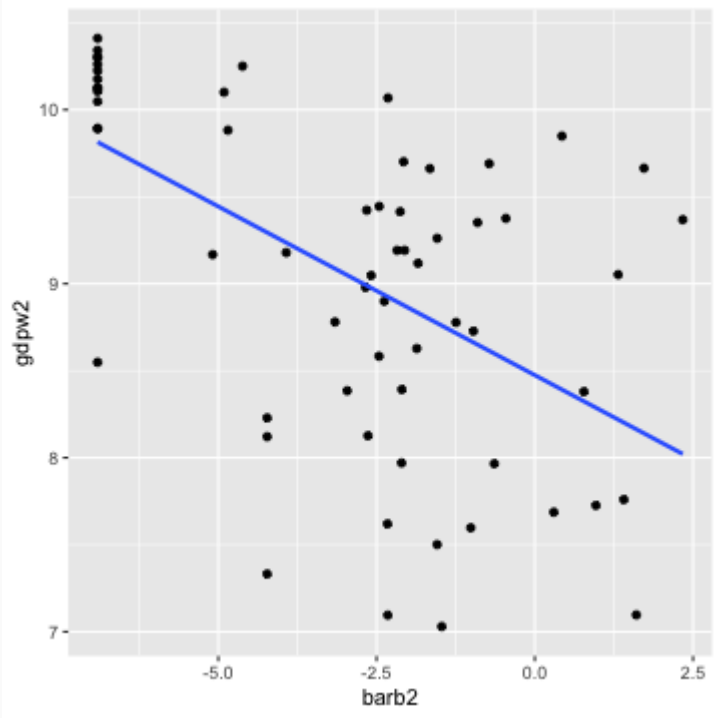
```
# Load gapminder
library(gapminder)
head(gapminder, 2)
```

```
## # A tibble: 2 x 6
##   country     continent  year lifeExp     pop gdpPercap
##   <fct>       <fct>     <int>   <dbl>   <int>     <dbl>
## 1 Afghanistan Asia       1952    28.8 8425333      779.
## 2 Afghanistan Asia       1957    30.3 9240934      821.
```

ggplot

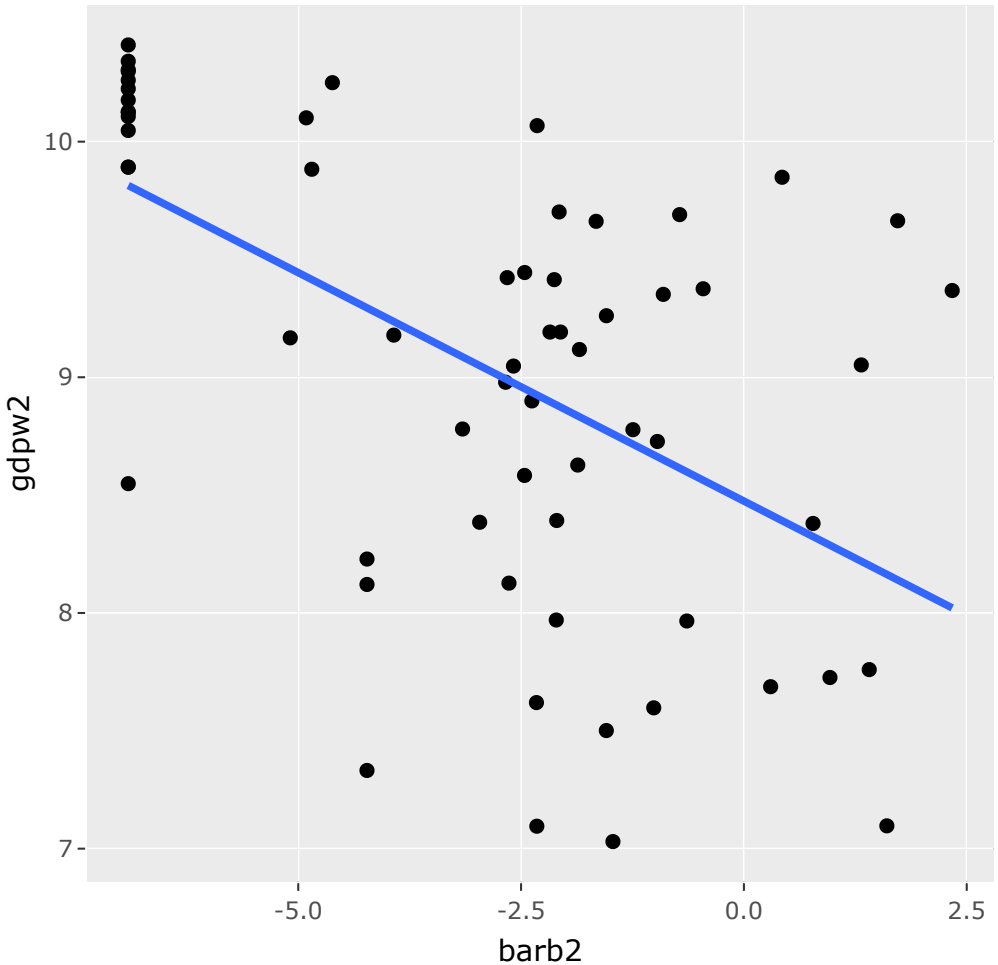
```
p ← ggplot(PErisk, aes(x=barb2, y=gdpw2)) +
  geom_point() +
  geom_smooth(method="lm", se=F)

p
```

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



plotly

```
ggplotly(p)
```



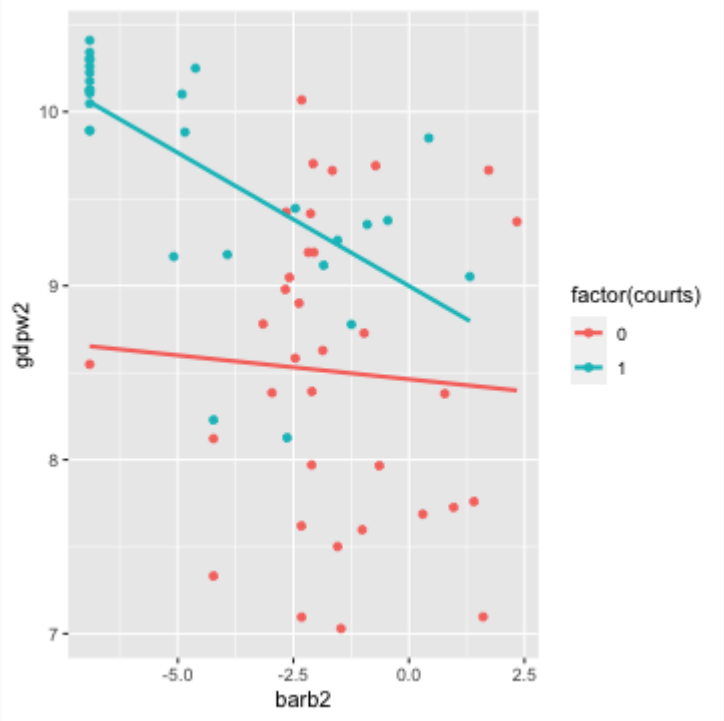
ggplotly

- **Your turn:** Create a `ggplotly` with `totbill` and `tip`.
- We can do even better: create a plotly with colors by factor.

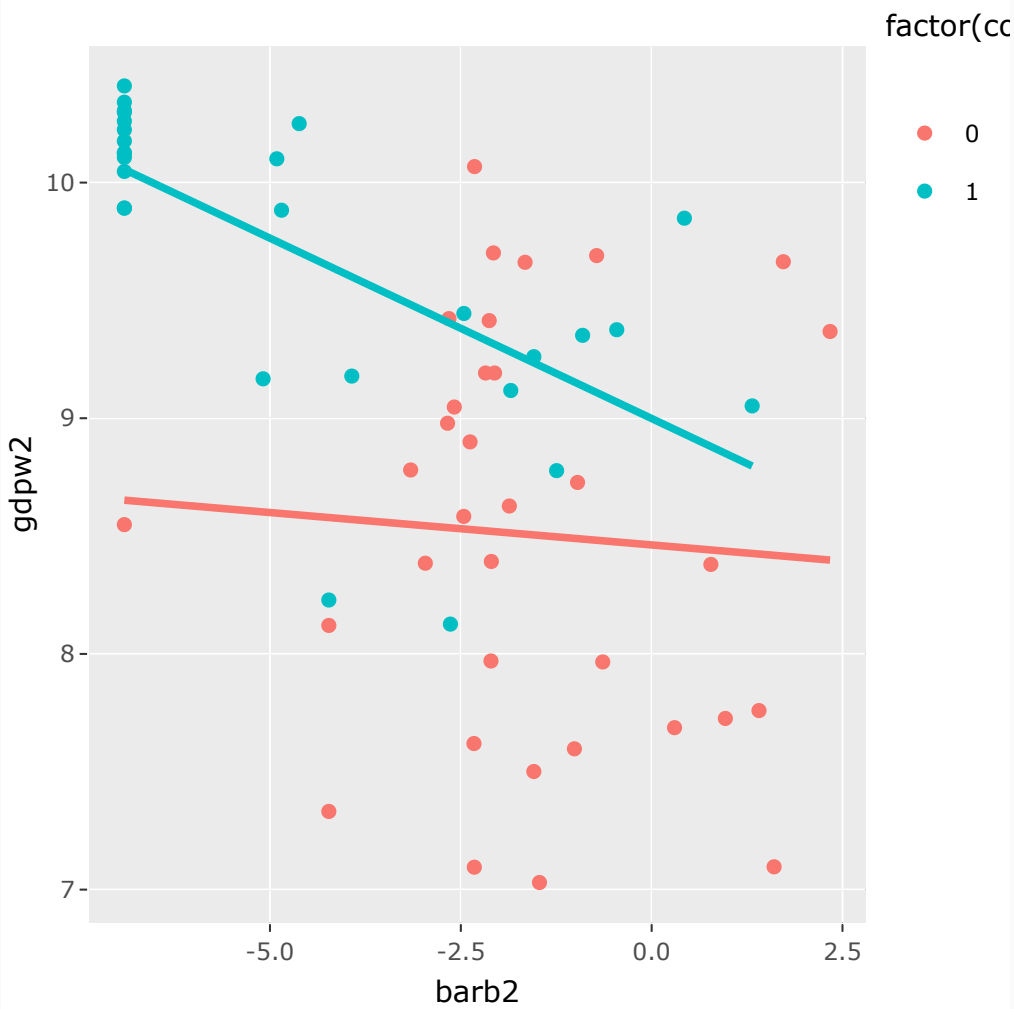
ggplot

```
p ← ggplot(PErisk, aes(x=barb2, y=gdpw2)) +
  geom_point(aes(col = factor(courts))) +
  geom_smooth(aes(col = factor(courts)), method="lm", se=F)
p
```

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



plotly

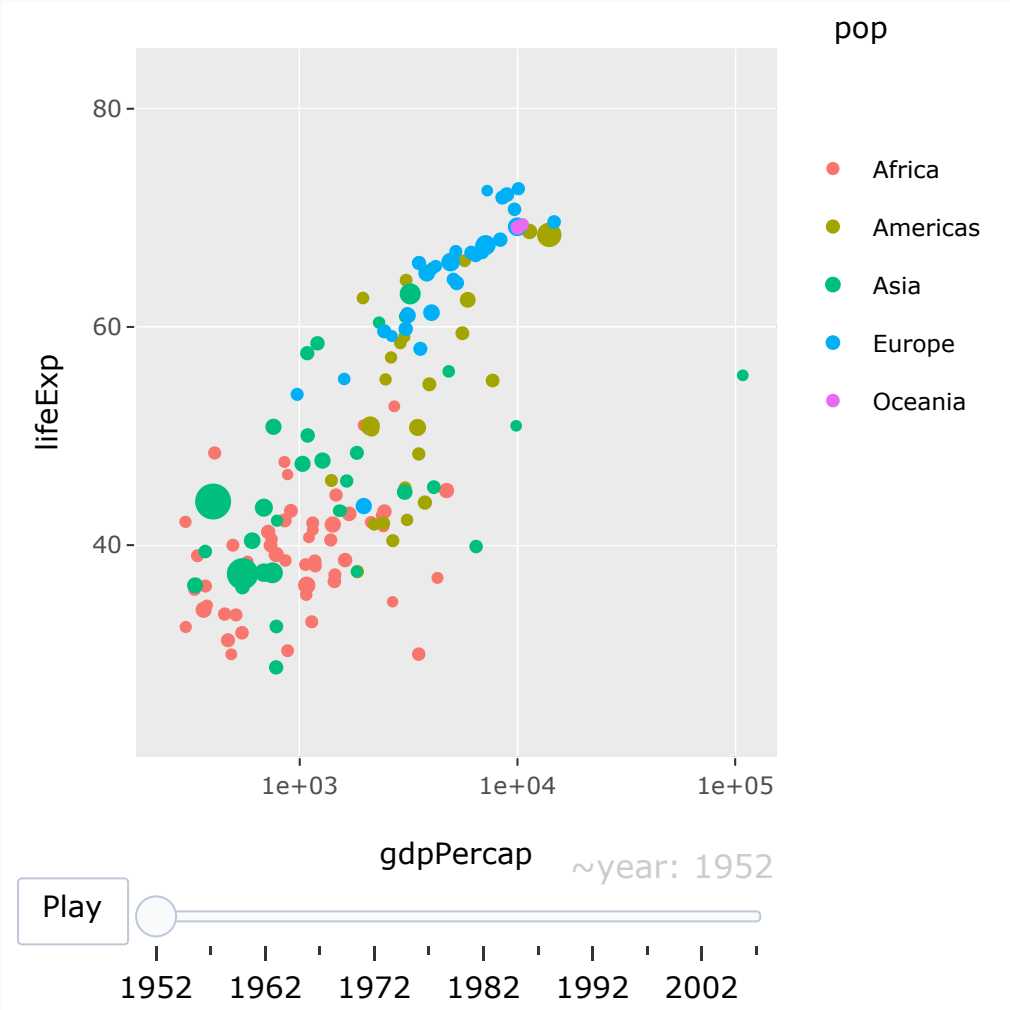


Intro to Animations in ggplot2

- Animations can be created by using the *frame* argument aesthetic in ggplotly().
- By default, animations populate a play button and slider component for controlling the state of the animation (to pause an animation, click on a relevant location on the slider bar).
- One example

plotly

```
## Warning: Ignoring unknown aesthetics: frame, ids
```



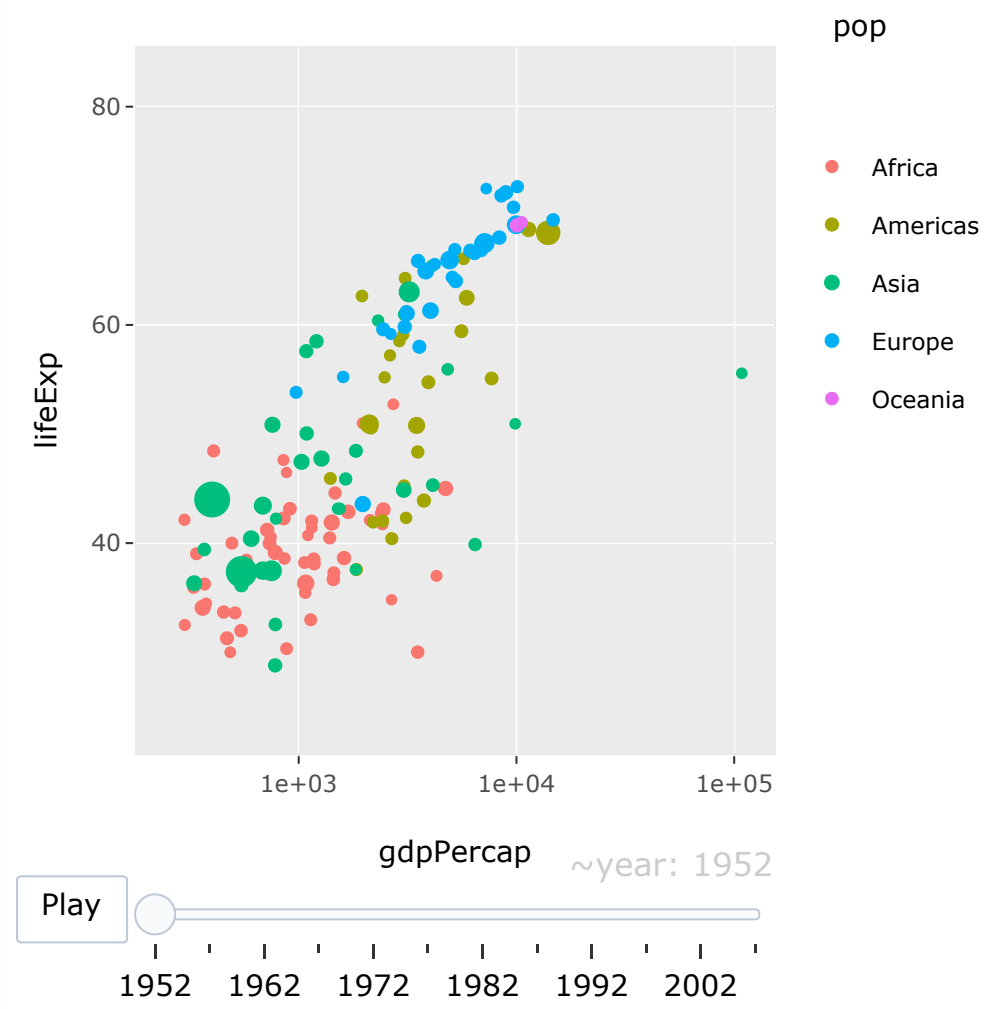
ggplotly

- **Your turn:** Create a plotly with `totbill` and `tip`, varying by day and time.

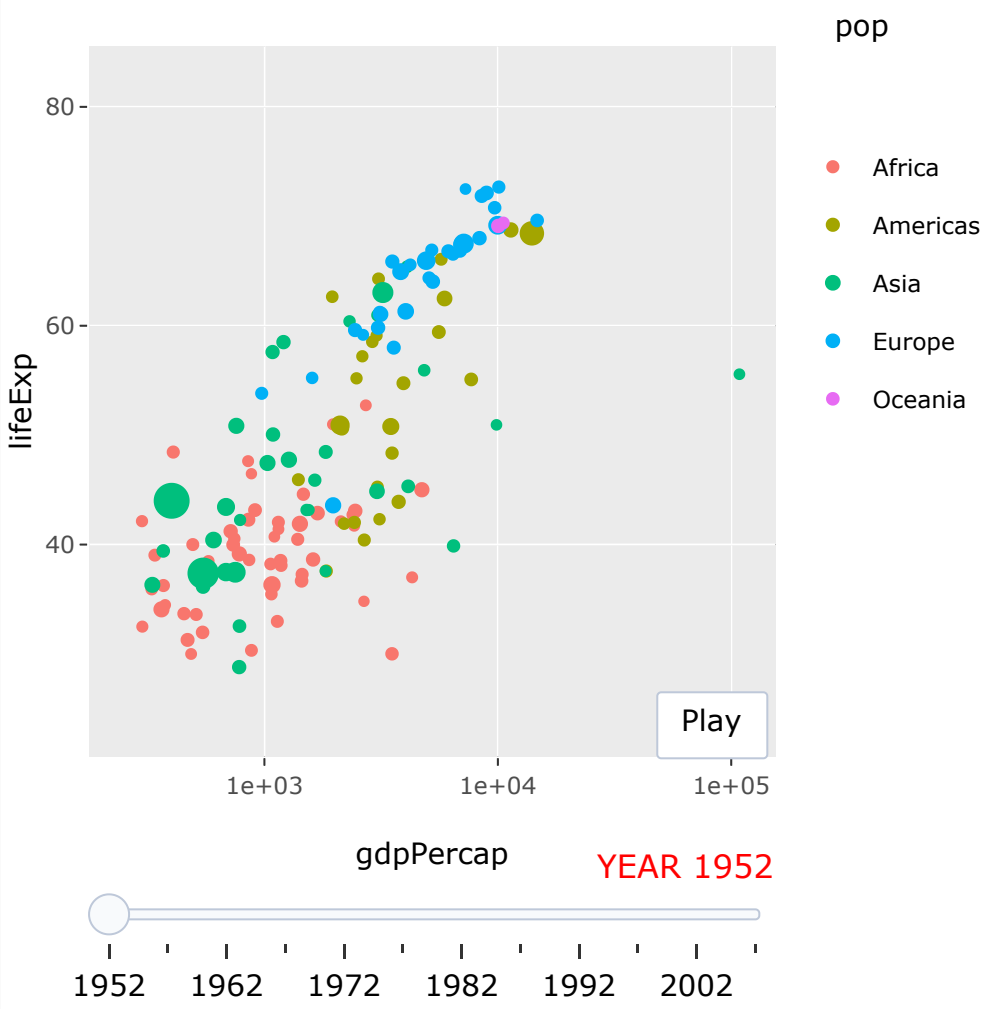
- And you can keep adding animation options:

  - transition: the duration of the smooth transition between frame (in milliseconds)
  - easing: the type of transition easing
  - mode: describes how a new animate call interacts with currently-running animations.

ggplotly: more animation options



ggplotly: more animation options



ggplotly

- **Your turn:** Make some animation options in your previous plot: `totbill` x `tip`, varying by day and time.

plot_ly

- The *plot_ly()* function provides a direct interface to plotly.js, so anything in the figure reference can be specified via *plot_ly()*.

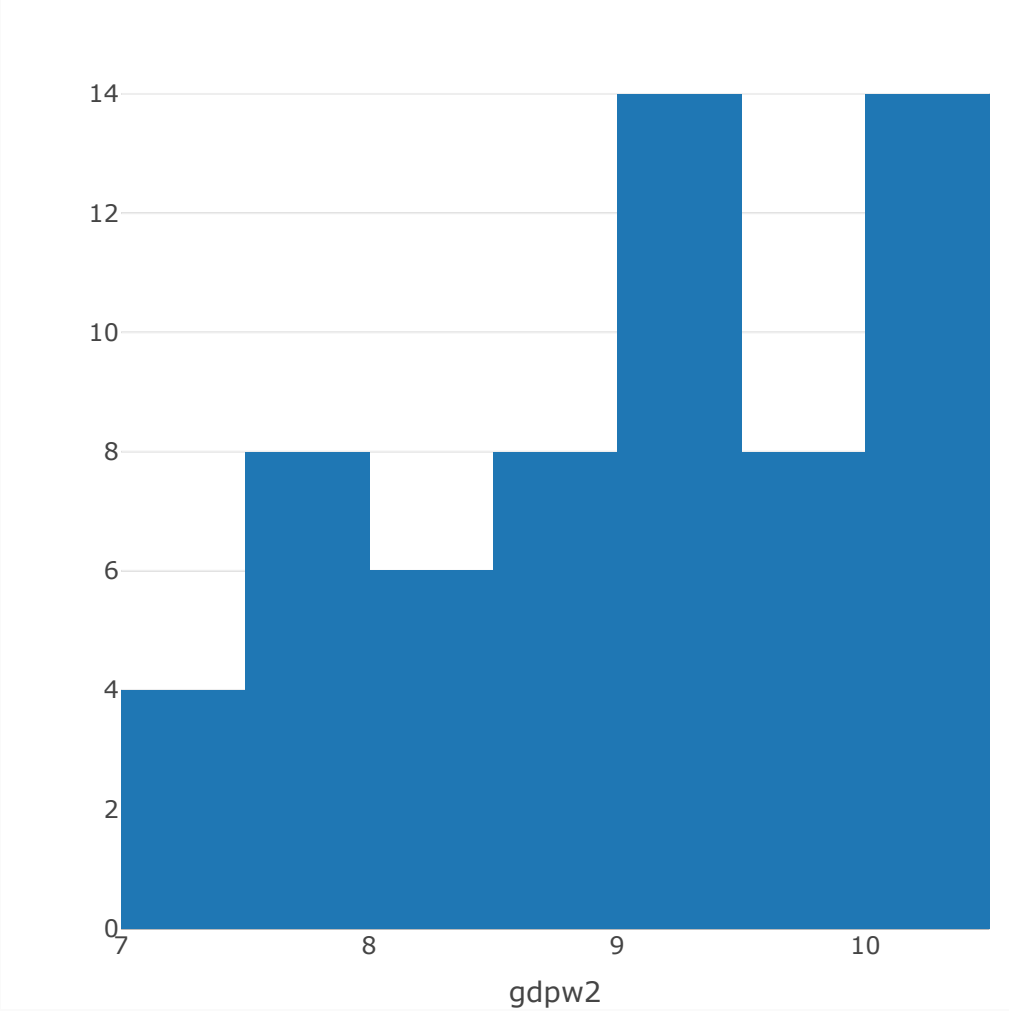
- A plotly visualization is composed of one (or more) trace(s), and every trace has a *type* (the default trace type is "scatter") can be used to draw a large amount of geometries along with the *add_XX()* functions.

- The *plot_ly()* function has a number of arguments that make it easier to scale data values to visual aesthetics (e.g., *color/colors, symbol/symbols, linetype/linetypes, size/sizes*).

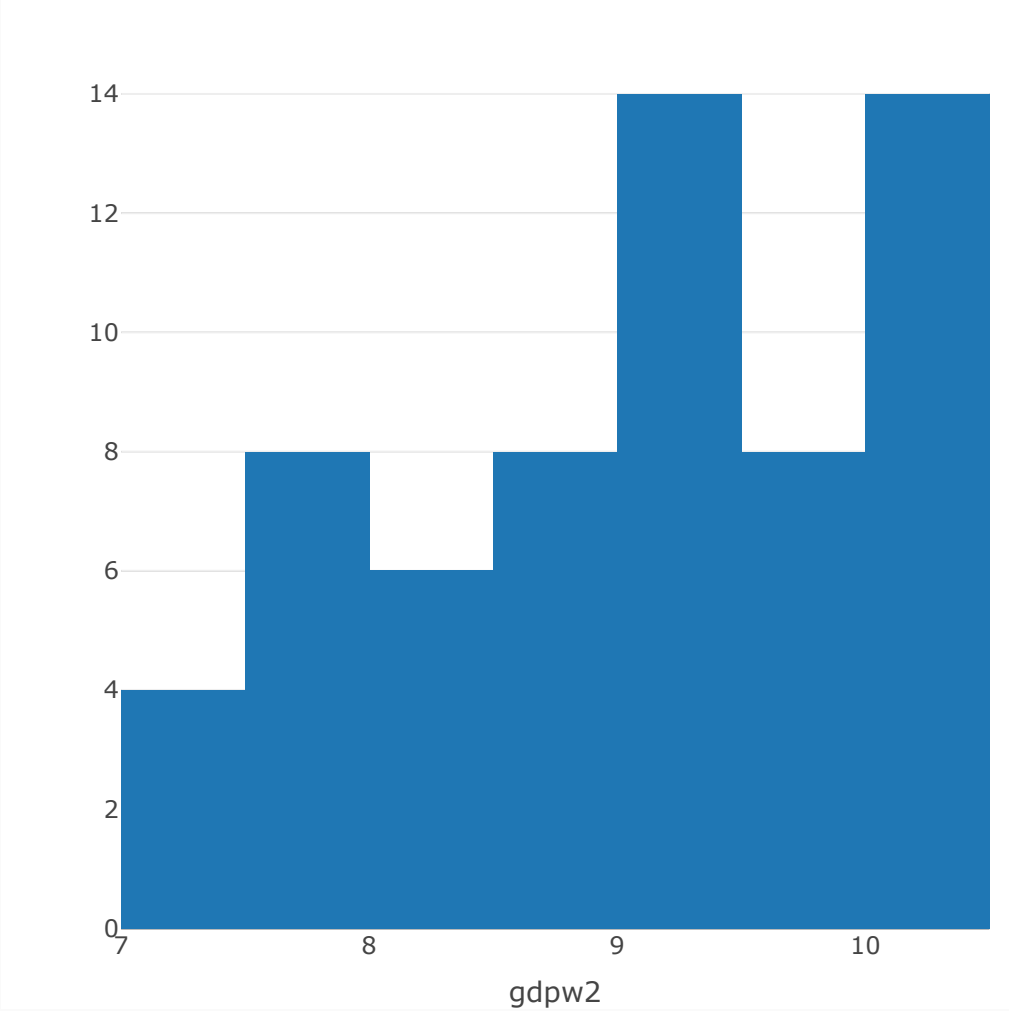
plot_ly

```
p←plot_ly(tips, x=~tip, type="histogram")

p1←plot_ly(tips, x=~tip) %>% add_histogram(name="tip")
p2←plot_ly(tips, x=~totbill) %>% add_histogram(name="totalbill")

subplot(p1,p2) %>% hide_legend()
```

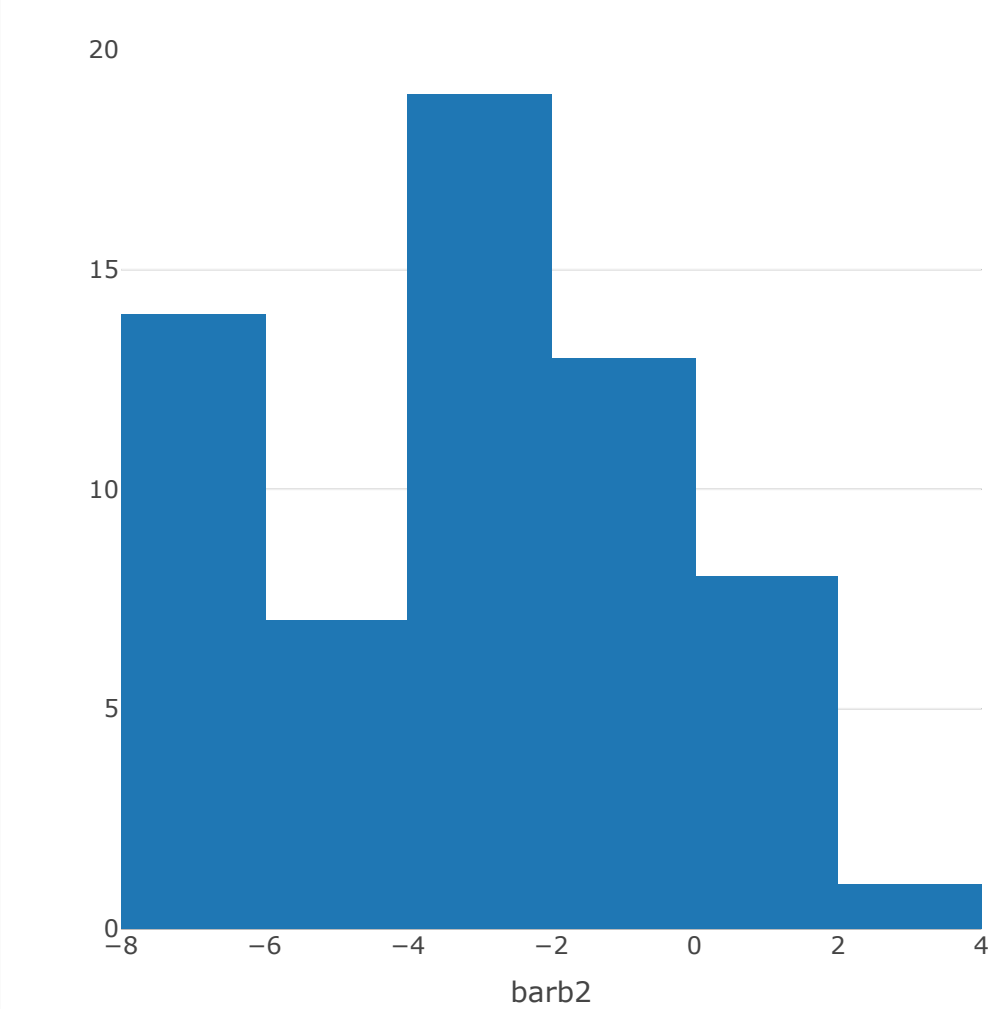
plot_ly



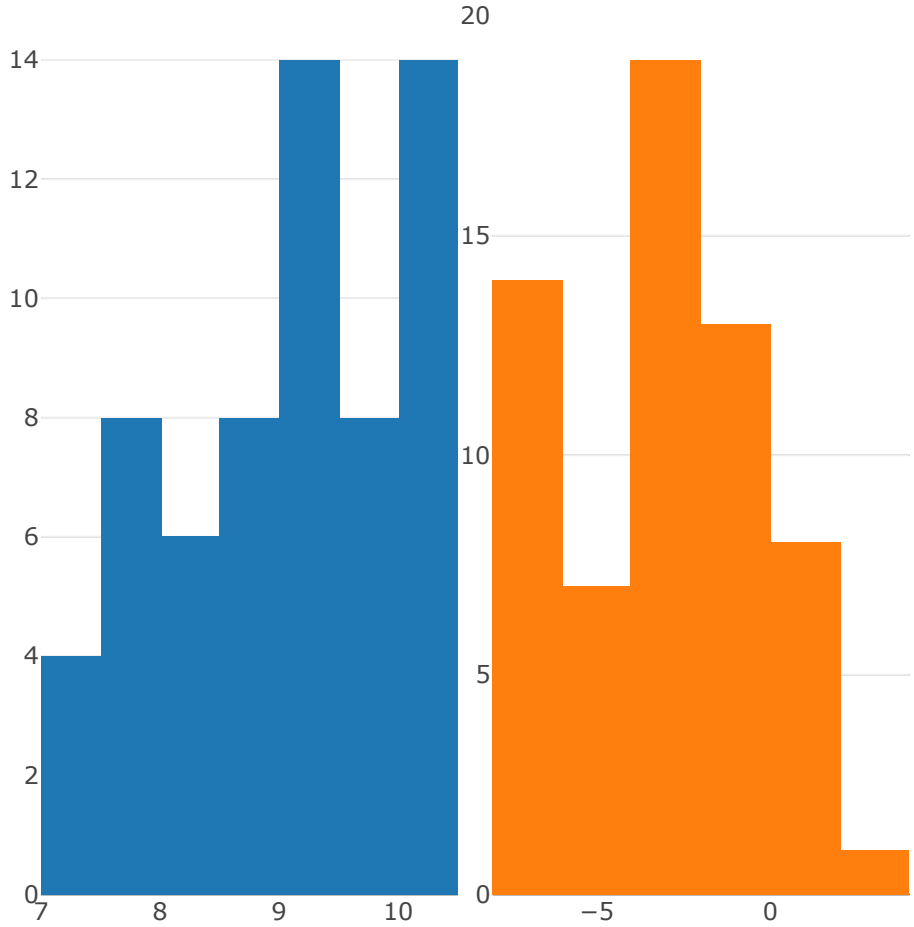
plot_ly



plot_ly



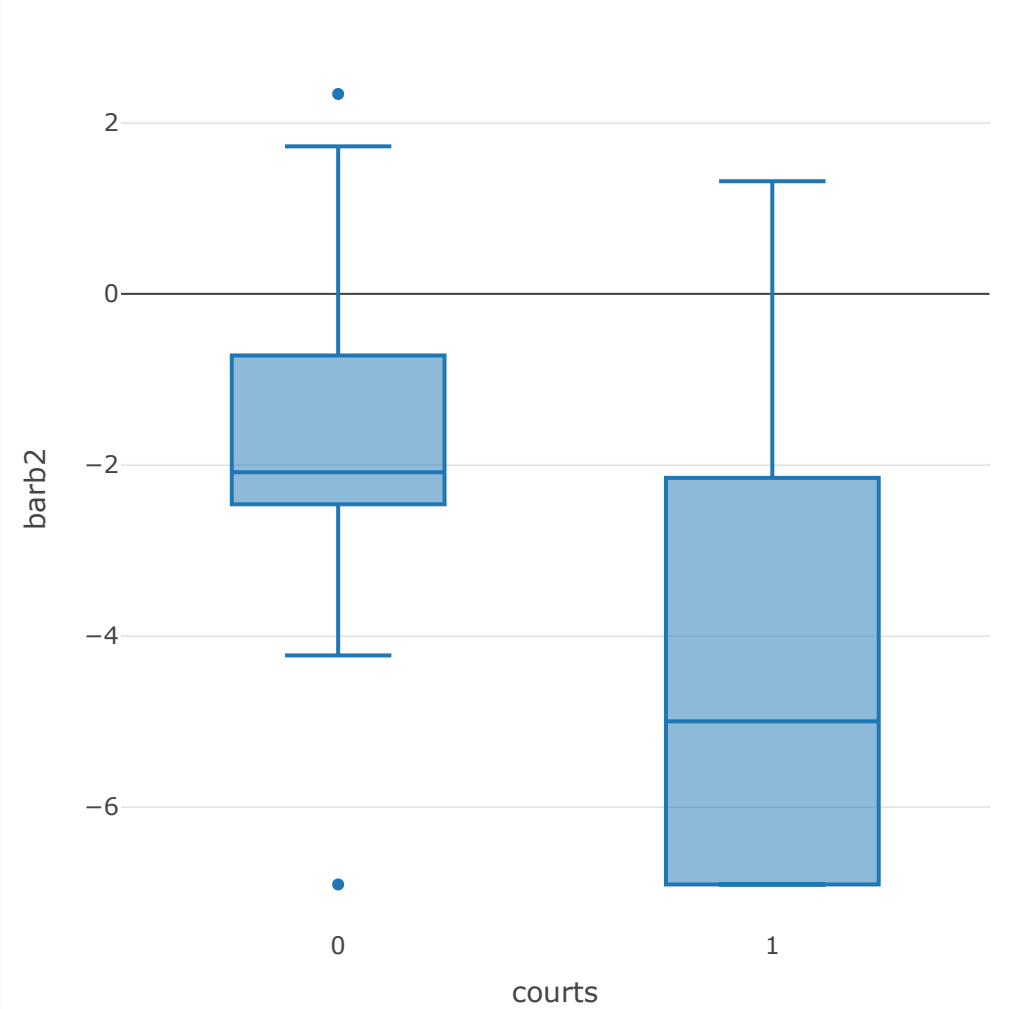
plot_ly



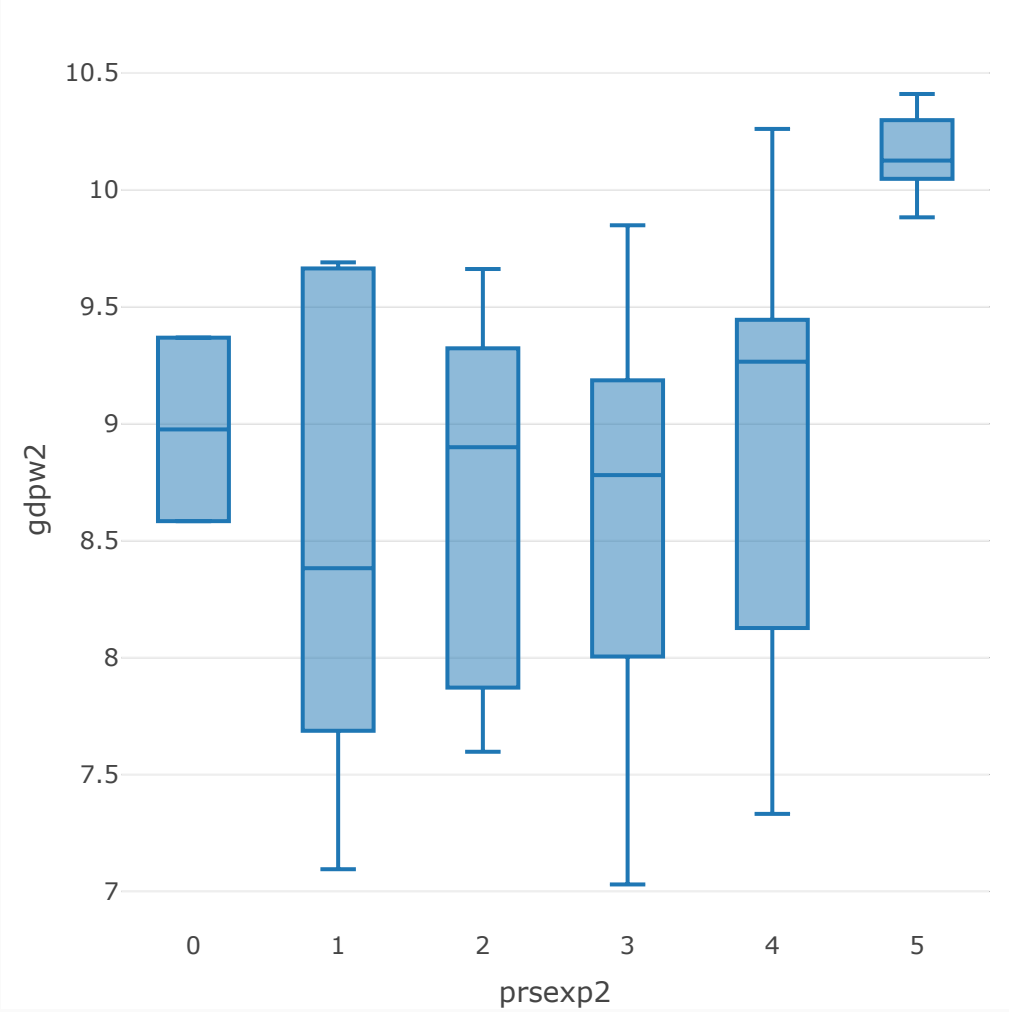
plot_ly

- **Your Turn**: do the same plots for `tip` and `totbill`.
- We can also do boxplots.

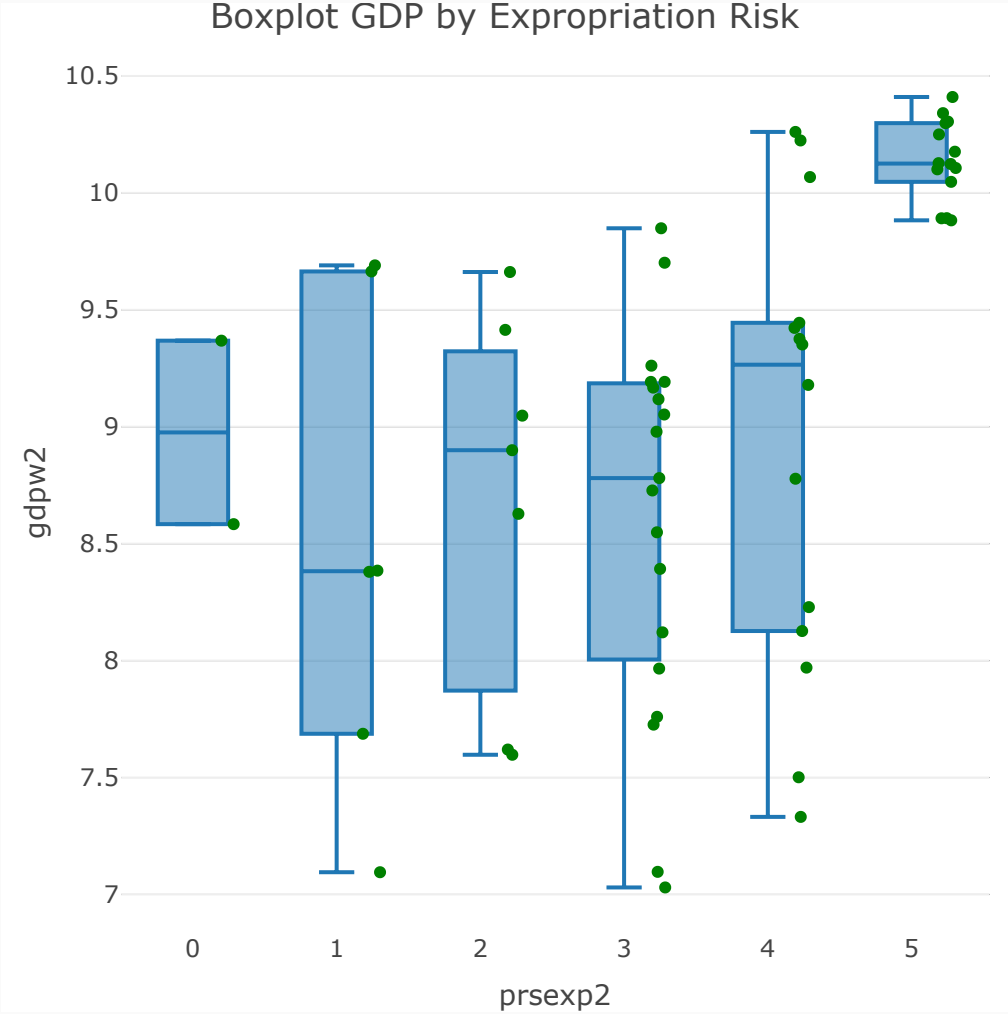
plot_ly: boxplot



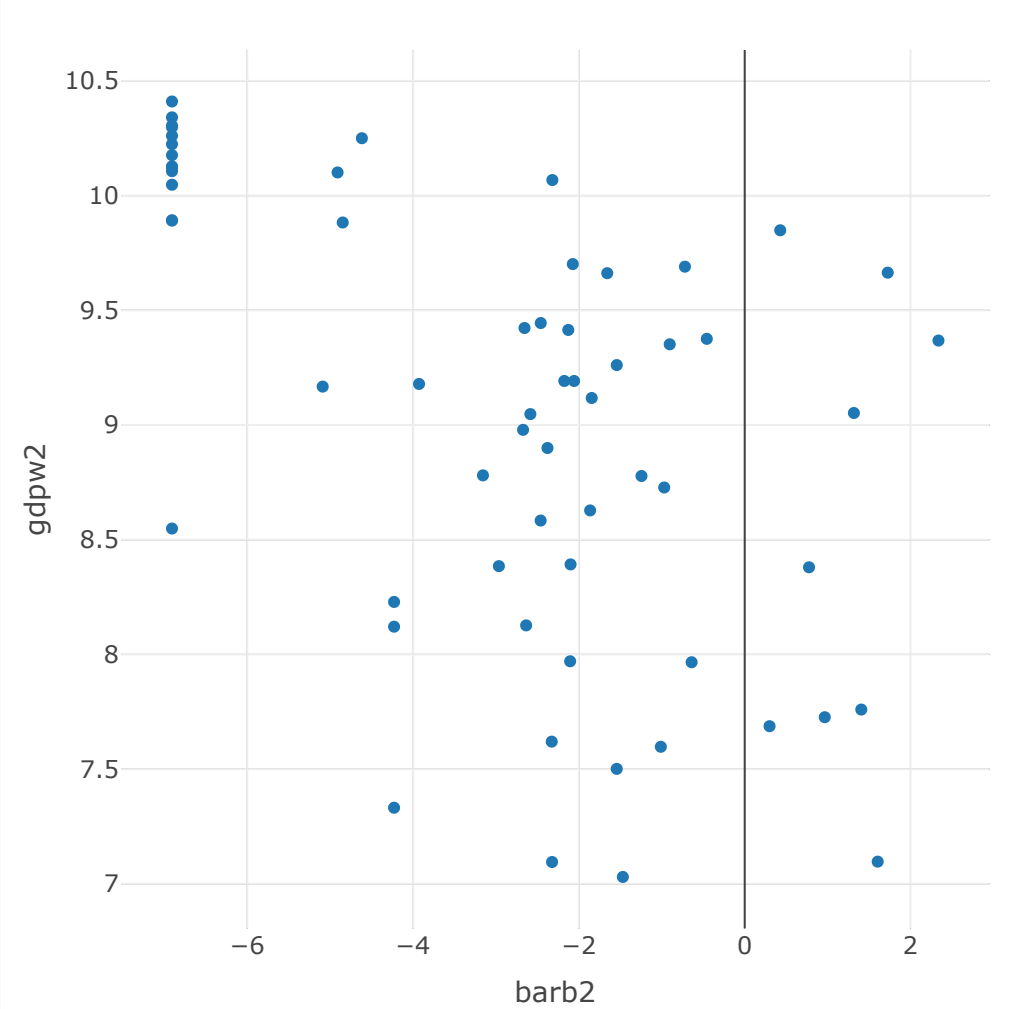
plot_ly: boxplot



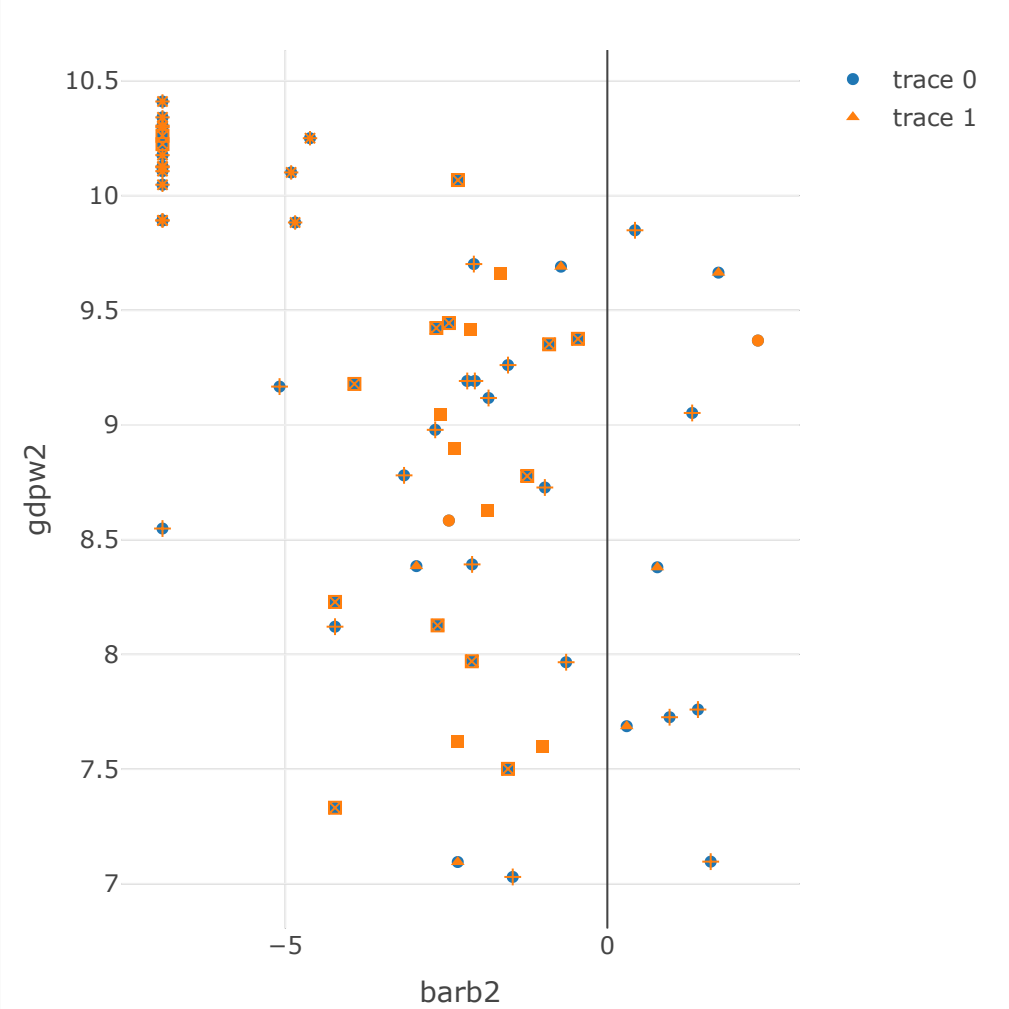
plot_ly: boxplot



plot_ly: scatterplots



plot_ly: scatterplots



Questions?

Have a great weekend!