

# Changing the plot type

- ▶ In ggplot2 we use geom functions to determine the type of plot we create
- ▶ In ggvis we use layer functions
- ▶ Not all geoms are currently available as layers

# Layers

red  red red red  red red red  red red red  red red red  red red   red red

## Function Description

$\text{layer}_p$ oints Adds data as points  $\text{layer}_h$ istograms Adds data as a histogram

```
> tubeData $ \%>% $  
+ ggvis(x = ~Line, y = ~Excess) $ \%>% $  
+ layer_boxplots()
```

# Exercise



red



red

red



red

red



red

red



red



red

- ▶ Update the plot of mpg against wt to include a smooth line of the data
- ▶ Add a confidence interval to the smooth line and colour in red
- ▶ Add a regression line and colour it blue
- ▶ Create a boxplot of mpg split by cylinder (hint: the cylinder variable will need to be a factor)

Layers So far, you've seen two layer functions:

`layer_points()` and `layer_histograms()`. There are many other layers, and the

Simple, which include primitives like points, lines and rectangles.

Compound, which combine data transformations with one or more simple layers.

All layer functions use the plural, not the singular. Think the verb, not the noun: I'm going to layer some points onto my plot.

Simple layers There are five simple layers:

Points,

`layer_points()`, with properties `x`, `y`, `shape`, `stroke`, `fill`, `strokeOpacity`, `fill`

`mtcars` 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5 wt 10 12 14 16 18 20 22

24 26 28 30 32 34 mpg Paths and polygons, `layer_paths()`.

`df <- data.frame(x = 1:10, y = runif(10))` `df` 1 2 3 4 5 6 7 8 9 10 x

0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 y If you supply a fill, you'll get a polygon

`t <- seq(0, 2 * pi, length = 100)` `df <- data.frame(x = sin(t), y = cos(t))` `df` -1.0 -0.8 -0.6 -0.4 -0.2 0.0 0.2 0.4 0.6

0.8 1.0 x -1.0 -0.8 -0.6 -0.4 -0.2 0.0 0.2 0.4 0.6 0.8 1.0 y Filled areas,

`layer_ribbons()`. Use properties `xy` and `dy` to control the extent of the area.

```
df_j- data.frame(x = 1:10, y = runif(10)) df 1 2 3 4 5 6 7 8 9 10 x
0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 y df 1 2 3 4 5 6 7 8
```

```
9 10 x -0.1 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 y + 0.1
```

`layer_rects()`. The location and size of the rectangle is controlled by the `x`, `x2`,

```
set.seed(1014) df_j- data.frame(x1 = runif(5), x2 = runif(5), y1 =
runif(5), y2 = runif(5)) df 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 x1
```

```
0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0 y1
```

`layer_text()`. The text layer has many new options to control the appearance of `text(the label)`, `dx` and `dy` (margin in pixels between text and anchor point),

```
df_j- data.frame(x = 3:1, y = c(1, 3, 2), label = c("a", "b", "c"))
df 1.0 1.2 1.4 1.6 1.8 2.0 2.2 2.4 2.6 2.8 3.0 x 1.0 1.2 1.4 1.6 1.8
```

```
2.0 2.2 2.4 2.6 2.8 3.0 y a b c df 1.0 1.2 1.4 1.6 1.8 2.0 2.2 2.4 2.6
2.8 3.0 x 1.0 1.2 1.4 1.6 1.8 2.0 2.2 2.4 2.6 2.8 3.0 y a b c df 1.0
```

```
1.2 1.4 1.6 1.8 2.0 2.2 2.4 2.6 2.8 3.0 x 1.0 1.2 1.4 1.6 1.8 2.0 2.2
2.4 2.6 2.8 3.0 y a b c
```

Compound layers The four most common compound layers are:

`layer_lines()` which automatically orders by the `x` variable :

```
t = seq(0, 2 * pi, length = 20) df = data.frame(x = sin(t), y = cos(t))
df
```

x	y
-1.0	-0.8
-0.8	-0.6
-0.6	-0.4
-0.4	-0.2
-0.2	0.0
0.0	0.2
0.2	0.4
0.4	0.6
0.6	0.8
0.8	1.0
1.0	-0.8
-0.8	-0.6
-0.6	-0.4
-0.4	-0.2
-0.2	0.0
0.0	0.2
0.2	0.4
0.4	0.6
0.6	0.8
0.8	1.0
1.0	-0.8

*layer\_lines()* is equivalent to *arrange()* + *layer\_paths()* :

```
df
```

x	y
-1.0	-0.8
-0.8	-0.6
-0.6	-0.4
-0.4	-0.2
-0.2	0.0
0.0	0.2
0.2	0.4
0.4	0.6
0.6	0.8
0.8	1.0
1.0	-0.8
-0.8	-0.6
-0.6	-0.4
-0.4	-0.2
-0.2	0.0
0.0	0.2
0.2	0.4
0.4	0.6
0.6	0.8
0.8	1.0

*layer\_histograms()* and *layer\_freqpolys()* which allows you to explore the distribution of

```
mtcars %>% summarise(width = 1) %>% range / 24
```

mpg	count
0.0	0.0
0.5	0.5
1.0	1.0
1.5	1.5
2.0	2.0
2.5	2.5
3.0	3.0
3.5	3.5
4.0	4.0
4.5	4.5
5.0	5.0

```
mtcars %>% summarise(width = 1) %>% range / 24
```

mpg	count
0.0	0.0
0.5	0.5
1.0	1.0
1.5	1.5
2.0	2.0
2.5	2.5
3.0	3.0
3.5	3.5
4.0	4.0
4.5	4.5
5.0	5.0

32 You can control the degree of wiggleness with the span

red  red    red    red  red    red    red    red  red    red    red    red  red    red    red    red  red    red  red  red  red    red

*Can't output dynamic/interactive ggvis plots in a knitr document.* >

*interactive*)versionoftheplot.1.52.02.53.03.54.04.55.05.5wt1214161820



## Interactivity

As well as mapping visual properties to variables or setting them to specific values, you can also connect them to interactive controls.

# MAKING PLOTS INTERACTIVE

## Basic interactivity

The most basic interactivity we can add is "hover over" changes

We can change properties by using `property.hover` arguments

```
fill.hover := "red" ; tubeData % % + ggvis( Excess) % % +  
layer_histograms(fill.hover = "red")
```

## Interactive Input

We can also set properties to be the output of an interactive control `opacity := input_slider(0, 1, label =`

`"Opacity")` We use the setting `:= "for this input`

We can optionally set labels next to the control



red



red

red



red

red



red

red



red



red

## Interactive Input Functions Function Description

`input_slider` *Slidertoselectvaluesorrangesofvalues* `input_checkbox` *Asing*

```

i tubeData % % + ggvis(x = Month, y = Excess, + opacity :=
input_slider(0, 1, +value = 0.7, label = "Opacity"), +size :=
input_numeric(30, label = "Pointsize"), +fill :=
input_select(c("red", "orange", "blue"), +label =
"Pointcolour")) % % +

```

## Tooltips

`add_tooltip` allows us to include other behaviour when we hover

$$\text{layer}_{points}()$$



red



red

red



red

red



red

red



red



red

```
j tubeData % % + ggvis(x = Month, y = Excess) % % +  
layer_points() % % + add_tooltip(function(data) data$Excess)
```

# Exercise



red



red

red



red

red



red

red



red



red

- ▶ Update the previous plot of mpg against wt so points change colour when they hover over
- ▶ Add a tooltip that shows the value of mpg when the point is hovered over
- ▶ Add a slider for the span of the smooth line so that values can be set between 0 and 1



The following example allows you to control the size and opacity of points with two sliders:

```
mtcars %>%  
  ggvis(~wt, ~mpg,  
    size := input_slider(10, 100),  
    opacity := input_slider(0, 1)  
  ) %>%  
  layer_points()
```

You can also connect interactive components to other plot parameters like the width and centers of histogram bins:

```
mtcars %>%
  ggvis(~wt) %>%
  layer_histograms(width = input_slider(0, 2,
    step = 0.10,
    label = "width"),
    center = input_slider(0, 2,
    step = 0.05,
    label = "center"))
```

Behind the scenes, interactive plots are built with shiny, and you can currently only have one running at a time in a given R session. To finish with a plot, press the stop button in Rstudio, or close the browser window and then press Escape or Ctrl + C in R.

As well as

`input_slider()`, *ggvis* provides `input_checkbox()`, `input_checkboxgroup()`, `input`

You can also use keyboard controls with

`left_right()` and `up_down()`. Press the left and right arrow to control the size of

```
keys_s <- left_right(10, 1000, step = 50)
mtcars %>% ggvis(~wt, ~mpg, size := keys_s, opacity := 0.
#> Warning: Can't output dynamic/interactive ggvis plots
#> Generating a static (non-dynamic, non-interactive) ver
```

1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5 wt 10 12 14 16 18 20 22 24 26  
28 30 32 34 mpg

# Interactivity : Tooltips

You can also add on more complex types of interaction like tooltips:

```
mtcars %>% ggvis(~wt, ~mpg) %>%  
  layer_points() %>%  
  add_tooltip(function(df) df$wt)
```

You'll learn more about complex interaction in the interactivity vignette.



## More details

There are other optional components that you can include:

- ▶ scales, to control the mapping between data and visual properties.  
These are described in the properties and scales vignette.
- ▶ legends and axes to control the appearance of the guides produced by the scales.  
See the axes and legends vignette for more details.



# Installing ggvis

- ▶ Current Version
- ▶ Dependencies

## Using ggvis - a word of warning!

```
> require(ggvis)
Loading required package: ggvis
The ggvis API is currently rapidly evolving. We strongly
recommend that you do not rely on this for production,
but feel free to explore. If you encounter a clear bug,
please file a minimal reproducible example at
https://github.com/rstudio/ggvis/issues. For questions
and other discussion, please use
https://groups.google.com/group/ggvis.
```

Figure: