How to Create Data Visualizations Using Best Practices

R Breakout Session

Stephen Salerno

Data for Public Good Symposium

February 26, 2021

Why do we visualize data?

Questions for You

- What do you use data visualizations for?
    - Data exploration/understanding
    - Figures for organizational reports
    - Figures for academic papers/grants
- Who is your typical audience when producing data visualizations?
    - Yourself
    - Members of your organization
    - Members of another organization
    - Members of the scientific community
- What is most important to you when thinking about data visualization?
    - Clarity
    - Reproducibility
    - Ease of creation

Why do we visualize data?

(and why exploratory data visualization is important)

The DatasauRus Dozen

Consider a dataset with (x,y) data coming from 12 groups:

Show 8 entries Search:

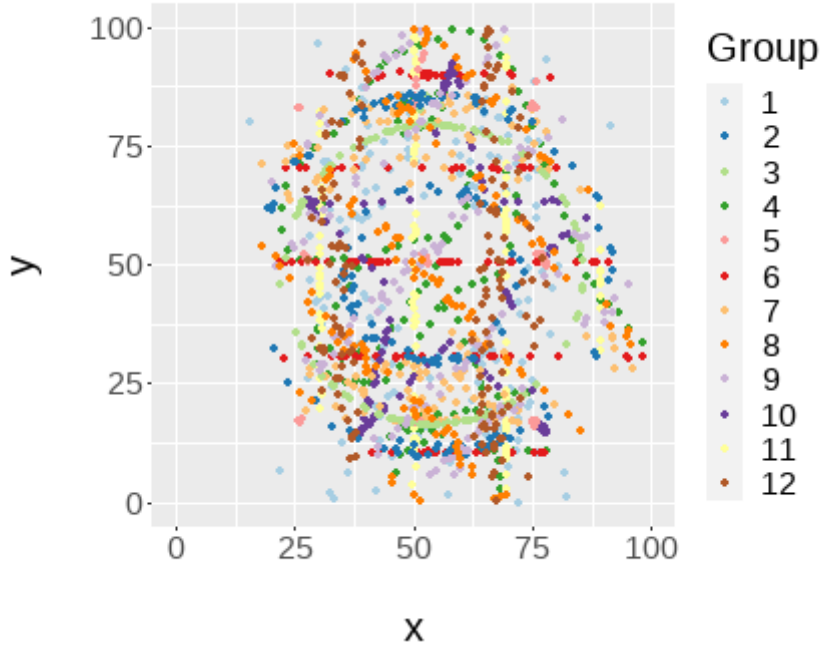
	Group	x	y
1	4	55.3846	97.1795
2	4	51.5385	96.0256
3	4	46.1538	94.4872
4	4	42.8205	91.4103
5	4	40.7692	88.3333
6	4	38.7179	84.8718
7	4	35.641	79.8718
8	4	33.0769	77.5641

Showing 1 to 8 of 1,704 entries

The DatasauRus Dozen

In aggregate, the data show no discernible pattern:

Visualizing the Data



Summary Statistics

Show 5 entries Search:

	Statistic	Value
1	Mean(x)	54.27
2	Mean(y)	47.83
3	SD(x)	16.71
4	SD(y)	26.85
5	Corr(x,y)	-0.07

Showing 1 to 5 of 5 entries

Previous 1 Next

The DatasauRus Dozen

Each group has summary statistics that are nearly identical:

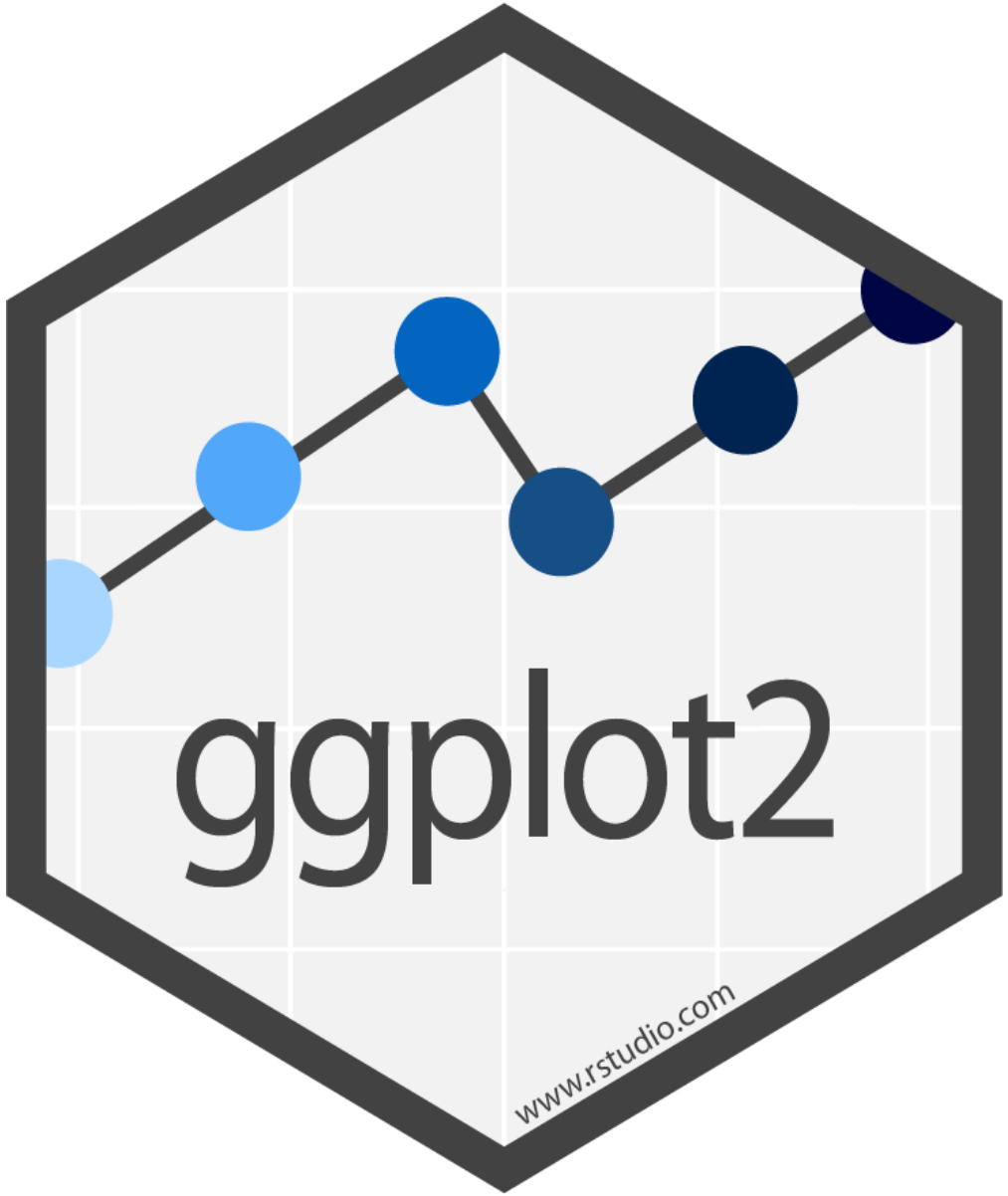
Show 8 entries Search:

	Group	Mean(x)	Mean(y)	SD(x)	SD(y)	Corr(x,y)
1	1	54.27	47.83	16.77	26.94	-0.06
2	2	54.27	47.83	16.77	26.94	-0.07
3	3	54.27	47.84	16.76	26.93	-0.07
4	4	54.26	47.83	16.77	26.94	-0.06
5	5	54.26	47.84	16.77	26.93	-0.06
6	6	54.26	47.83	16.77	26.94	-0.06
7	7	54.27	47.84	16.77	26.94	-0.07
8	8	54.27	47.84	16.77	26.94	-0.07

Showing 1 to 8 of 12 entries

The DatasauRus Dozen

Yet, plotting reveals that each group is quite distinct:


ggplot2
www.rstudio.com

ggplot2

- Though `R` has built-in functions for visualizing data, today's focus will be the `ggplot2` package
- **gg**plot2 is based on the **G**rammar of **G**raphics by Leland Wilkinson (see: ggplot2.tidyverse.org)
- Grammar states that all data visualizations consist of three parts:

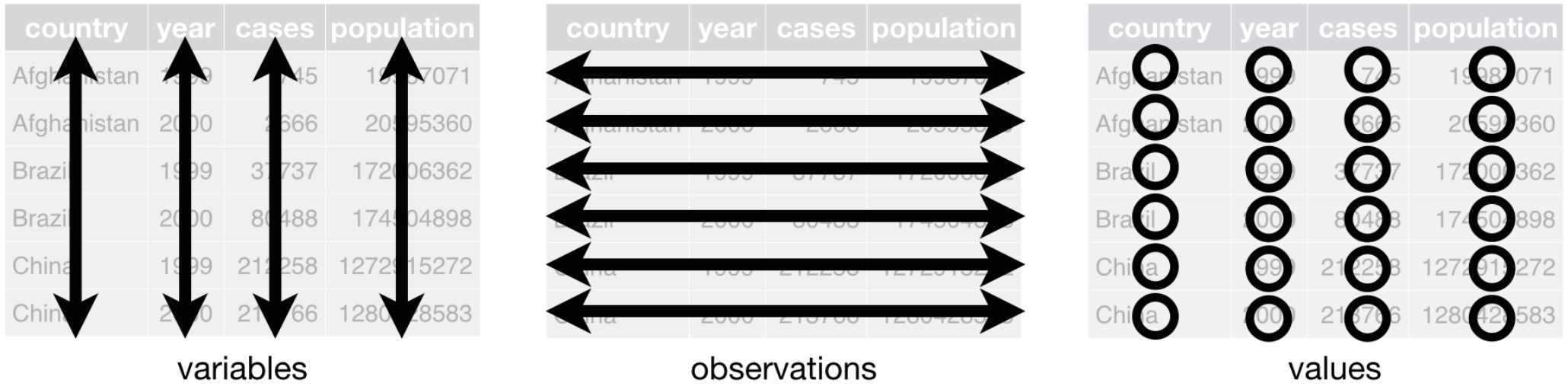
    1. Data
    2. Aesthetics
    3. Geometry

- `ggplot2` allows you to uniquely describe any plot as a combination of these three parts, via seven parameters:

```
ggplot(data = <DATA>) +
  <GEOMETRY_FUNCTION>(
     mapping = aes(<MAPPINGS>),
     stat = <STAT>,
     position = <POSITION>
  ) +
  <COORDINATE_FUNCTION>() +
  <FACET_FUNCTION>()
```

1. Data

- `ggplot2` works best with **tidy data**:
    1. Each variable in the data set has its own column
    2. Each observation has its own row
    3. Each value has its own cell
- It is (usually) best practice to have a single data source (see: cfss.uchicago.edu/notes/tidy-data/)



Untidy vs. Tidy Data

Untidy Data

Show 8 entries Search:

	name	quiz1	quiz2	test1
1	Billy		D	C
2	Suzy	F		
3	Lionel	B	C	B
4	Jenny	A	A	B

Showing 1 to 4 of 4 entries

Previous 1 Next

Tidy Data

Show 4 entries Search:

	Name	Assessment	Grade
1	Billy	Quiz 1	
2	Billy	Quiz 2	D
3	Billy	Test 1	C
4	Jenny	Quiz 1	A

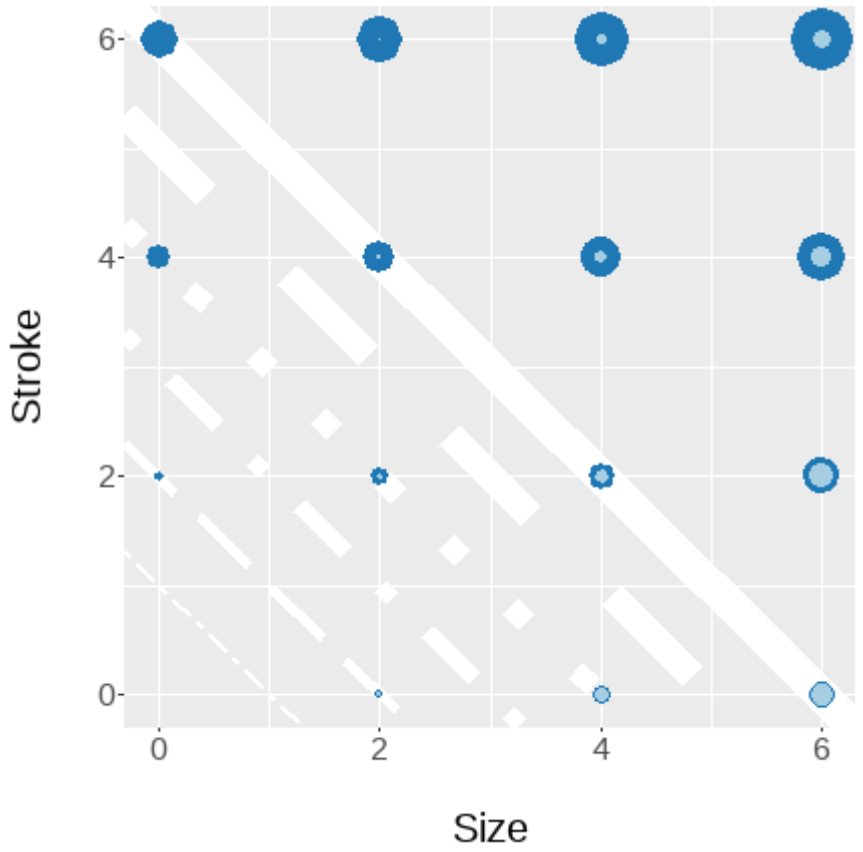
Showing 1 to 4 of 12 entries

Previous 1 2 3 Next

2. Aesthetics

- Aesthetics map the columns in our data to standard elements of a plot
- These define the 'visual cues' we use to distinguish aspects of our data:

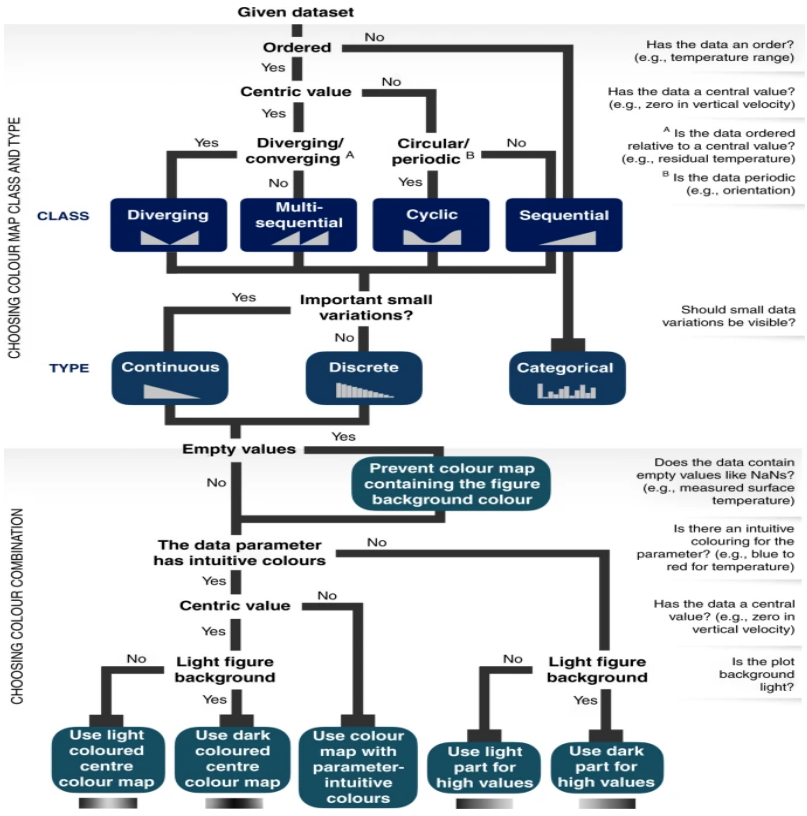
    1. Coordinate Positions (x- and y- axes)
    2. Color (i.e. 'outside' color)
    3. Fill (i.e. 'inside' color)
    4. Point Shapes
    5. Line Types
    6. Element Sizes



A Note on Color

Use color **intentionally** and **thoughtfully**:

- Meet basic accessibility guidelines for color
  - Color blindness
  - Intended medium (color vs. black and white)
- Do not reinforce gender or racial stereotypes
  - Blue for boys and pink for girls
  - Colors associated with skin tones
- Consider data class, type, and display
  - Continuous, Ordinal, Nominal
  - Increasing/Decreasing, Centric Valued



nature.com/articles/s41467-020-19160-7

Color Palettes

- `R` accommodates RGB, CMYK, or Hex color and has preset palettes to choose from (e.g. `ggsci` package):
  - Academic Journals (e.g. JAMA), Institutional (e.g. U Chicago), Just for Fun (e.g. Star Trek)
- Default colors for most data visualization software are difficult to see if color blind (`ggplot2` included)
- Thankfully, there are color blind friendly options (what I have been using thus far)

Viridis

```
library(scales)
# viridis also prints well in greyscale!
show_col(viridis_pal()(5), ncol = 5)
```

RColorBrewer

```
library(RColorBrewer)
# display.brewer.all(colorblindFriendly = T)
show_col(brewer.pal(5, "Set2"), ncol = 5)
```

3. Layers and Geometries

- A layer combines data, aesthetic mapping, a geom (geometric object), a stat (statistical transformation), and a position adjustment
- You will create layers using a `geom_XXXX()` function, overriding the default position and stat if needed
- Choice of geometry is specific to your data:
  - **Reference Lines:** geom_abline(), geom_hline(), geom_vline()
  - **Bar Charts:** geom_bar(), geom_col(), stat_count()
  - **Densities:** geom_density(), stat_density()
  - **Histograms:** geom_freqpoly(), geom_histogram(), stat_bin()
  - **... and many, many more!**
- See: ggplot2.tidyverse.org/reference/

Putting it All Together

Installing ggplot2

- Even though the package is sometimes just referred to as "ggplot," the package name is `ggplot2`
- `ggplot2` is a package included in the **tidyverse**: tidyverse.org
- To install the tidyverse packages (if you have not already), you will need to run:

```
install.packages("tidyverse") #-- Run this only once
```

- To load the tidyverse packages, run:

```
library(tidyverse) #-- Run this each session
```

- Be sure to load the tidyverse packages at the start of each session

What does ggplot() do?

- The `ggplot()` function returns a "gg/ggplot" object
- In the **Grammar of Graphics** workflow, ggplot objects are 'built up':

    1. Define a new plot (ggplot object) associated with your data
    2. Use (+) to add additional instructions (layers) to the object to build your plot
    3. Within layers, add aesthetics, geometries, statistical transformations, and/or position adjustments

```
ggplot(data = <DATA>) +              # 1. Define a new plot
  <GEOM_FUNCTION>(                   # 2. Add layers
                                     # 3. Within layers:
     mapping = aes(<MAPPINGS>),      #     Add aesthetics
     stat = <STAT>,                  #     Add statistical transformations
     position = <POSITION>           #     Add position adjustments
  )
```

- `ggplot()` can also be passed any default aesthetic mappings you would like all layers to use

```
ggplot(data = <DATA>, mapping = aes(<MAPPINGS>)) +
  <GEOM_FUNCTION>()
```

Mapping Values with aes()

- `aes()` is the function that defines a mapping from data to geometry aesthetics
- Signature: `aes(x, y, ...)`
  - The first two parameters default to x and y values
  - All other parameters should be named: `aes(x, y, <aesthetic> = column_name)`
- `aes()` expects parameters to be "symbols" or "expressions"
  - Do not use quotes or variables (see `aes_q` or `aes_string`)
  - May contain functions (e.g. `aes(x=a+b, y=sqrt(z))`)
  - Never use "`$`" inside an `aes()`

Mapping Values with aes()

- By default, layer `aes()` values are inherited from `ggplot()`
- Disable inheritance with `<GEOM_FUNCTION>(..., inherit.aes = FALSE)`
- You may also add, override, or remove `aes()` values:

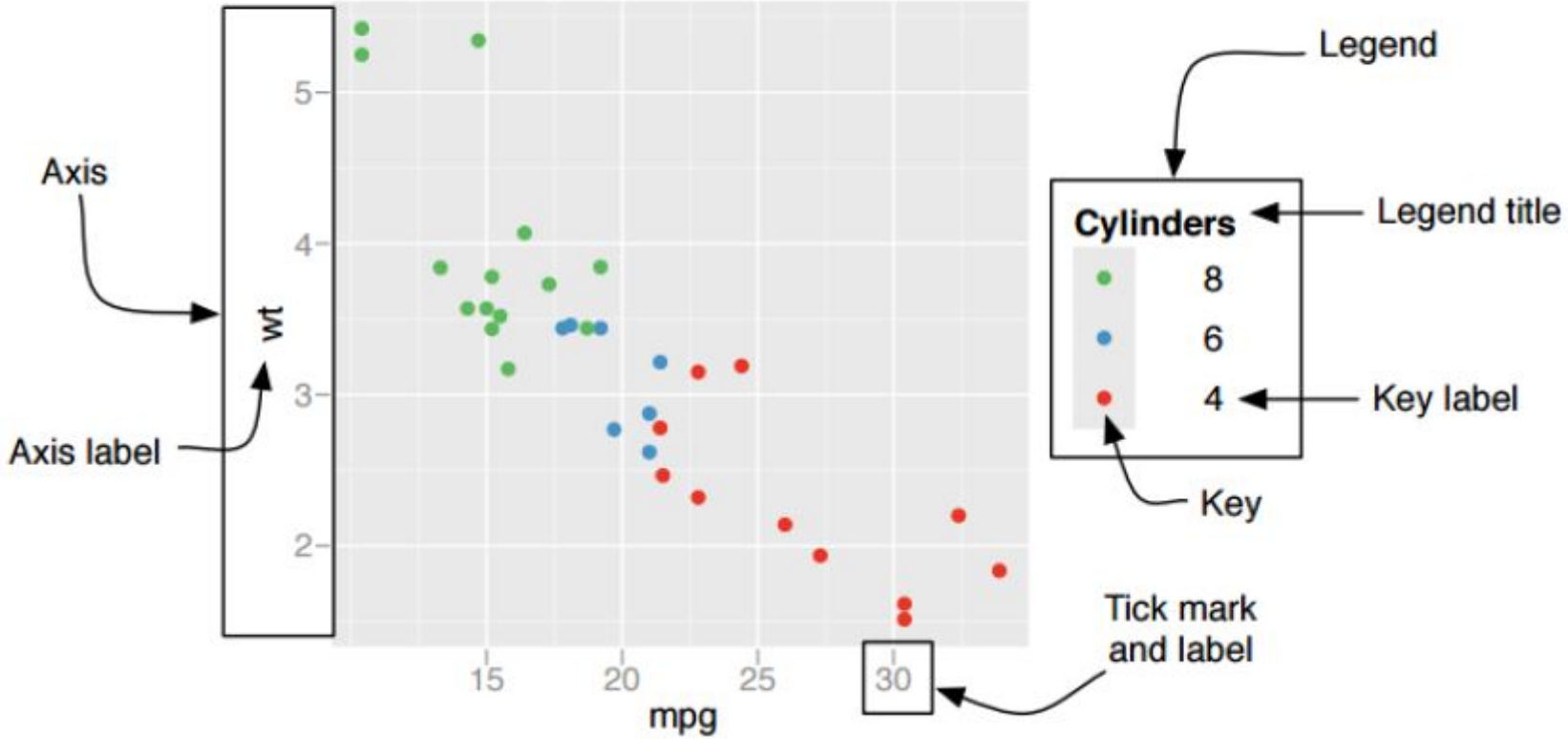
Show 3 entries Search:

	ggplot()	geom_XXXX()	Resulting Aesthetics	Operation
1	aes(x, y)	aes(color = Group)	aes(x, y, color = Group)	Add
2	aes(x, y)	aes(y = y2)	aes(x, y2)	Override
3	aes(x, y)	aes(y = NULL)	aes(x)	Remove

Showing 1 to 3 of 3 entries Previous 1 Next

There are many ways to get the same plot!

Plot parts


Axis
Axis label
wt
5
4
3
2
15
20
25
30
mpg
Legend
Cylinders
8
6
4
Legend title
Key label
Key
Tick mark
and label

Labeling

- You can label your axes, titles, and more with `labs()`:

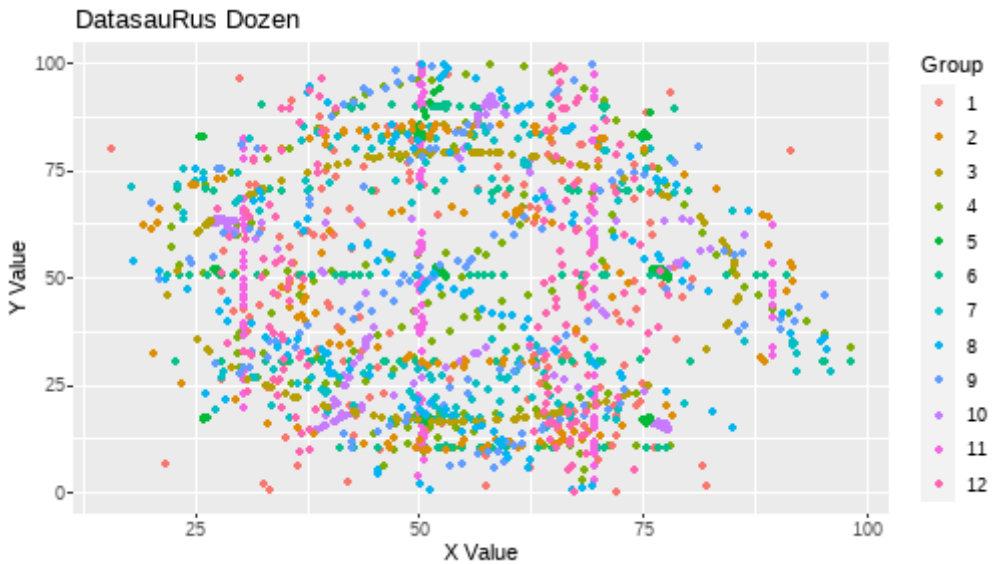
```
+ labs(x="X Name",
       y="Y Name",
       title="Plot Title"
       subtitle="Plot Subtitle")
```

- You can also include mathematical expressions (see `?plotmath`)

```
+ labs(title=expression(y==alpha+beta*x))
```

- Setting values to `""` shows no label, setting values to `NULL` removes space for label as well

```
ggplot(datasaurus_dozen2,
       aes(x, y, color=Group)) +
  geom_point() +
  labs(x="X Value",
       y="Y Value",
       title="DatasauRus Dozen")
```



Formatting Tick Labels

- Format 0-1 as percents: `+ scale_y_continuous(labels=scales::percent_format())`
- Format as dollar amounts: `+ scale_y_continuous(labels=scales::dollar_format("$"))`
- Format in thousands: `+ scale_fill_continuous(labels=scales::unit_format("k", 1e-3))`
- You can pass any function as the `labels=` parameter

Formatting Tick Marks

- You can set where your tick marks fall with `scale_x_continuous()`
    - `breaks=` vector of values where to draw major tick marks
    - `labels=` vector of values with what to draw at those tick marks
    - `minor_breaks=` vector of values where to draw minor (unlabeled) tick marks
    - `trans=` optional transformation to apply to axis
    - `expand=` how far to extend axis past observed data
    - `limits=` lower and upper bound for tick marks
- For date-time axes: `date_minor_breaks=` with units like "1 month" or "2 years"

Themes

- The overall "look" of a plot is set by the theme
- Just call one of the theme functions to see all the values you can customize
- The `ggthemes` package has many themes to try out
- You can also create your own theme objects

```
p <- ggplot(mpg, aes(cty, hwy, color=factor(cyl))) +
  geom_jitter() +
  geom_abline(color="grey50", size=2) +
  ggtitle("My Plot!")


p + theme_grey() # Default Theme
p + theme_bw()
p + theme_linedraw()
p + theme_light()
p + theme_dark()
p + theme_minimal()
```

Customizing a Theme

- You can customize parts of themes

```
p + theme(plot.title = element_text(color="red", margin=margin(t=20, b=20)))
p + theme(panel.background = element_rect(fill="linen"))
```

- Increase font size for presentation slides

```
p + theme_grey(base_size=18)
```

- Set default theme for session

```
theme_set(theme(...))
```

- The ggplot2 book or the "theme" help page on the ggplot2 website has more info

Examples!

SurvivoR Data

- `SurvivoR` is a collection of data sets in `R` detailing events across all 40 seasons and 596 episodes of the U.S. television show, Survivor
- Data include castaway information, vote history, immunity and reward challenge winners, and jury votes
- gradientdescending.com/survivor-data-from-the-tv-series-in-r/
- github.com/doehm/survivoR

```
devtools::install_github("doehm/survivoR") # Data are already tidy!
```

Season Summaries

```
library(survivoR)
season_summary
```

Show 3 entries Search:

	season_name	season	location	country	tribe_setup	full_name	winner
1	Survivor: Borneo	1	Pulau Tiga, Sabah, Malaysia	Malaysia	Two tribes of eight new players	Richard Hatch	Richard
2	Survivor: The Australian Outback	2	Herbert River at Goshen Station, Queensland, Australia	Australia	Two tribes of eight new players	Tina Wesson	Tina
3	Survivor: Africa	3	Shaba National Reserve, Kenya	Kenya	Two tribes of eight new players	Ethan Zohn	Ethan

Showing 1 to 3 of 40 entries Previous 1 2 3 4 5 … 14 Next

Viewership Over Time

```
p1_dat <- season_summary %>%
  select(season, viewers_premier, viewers_finale, viewers_reunion, viewers_mean) %>%
  pivot_longer(cols = -season, names_to = "episode", values_to = "viewers") %>%
  mutate( episode = str_replace(episode, "viewers_", ""))

DT::datatable(p1_dat, fillContainer = F, options = list(pageLength = 3))
```

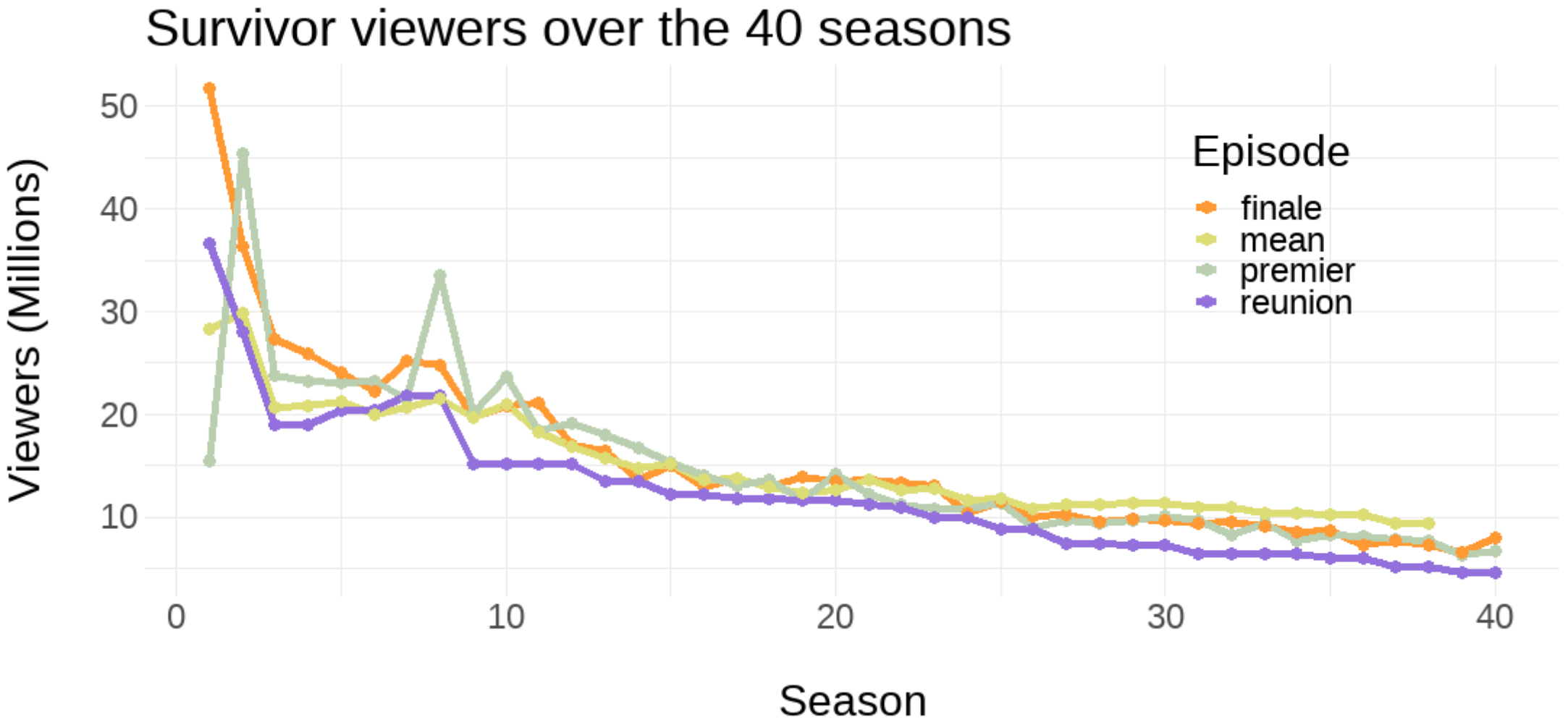
Show 3 entries Search:

	season	episode	viewers
1	1	premier	15.51
2	1	finale	51.69
3	1	reunion	36.7

Showing 1 to 3 of 160 entries Previous 1 2 3 4 5 ... 54 Next

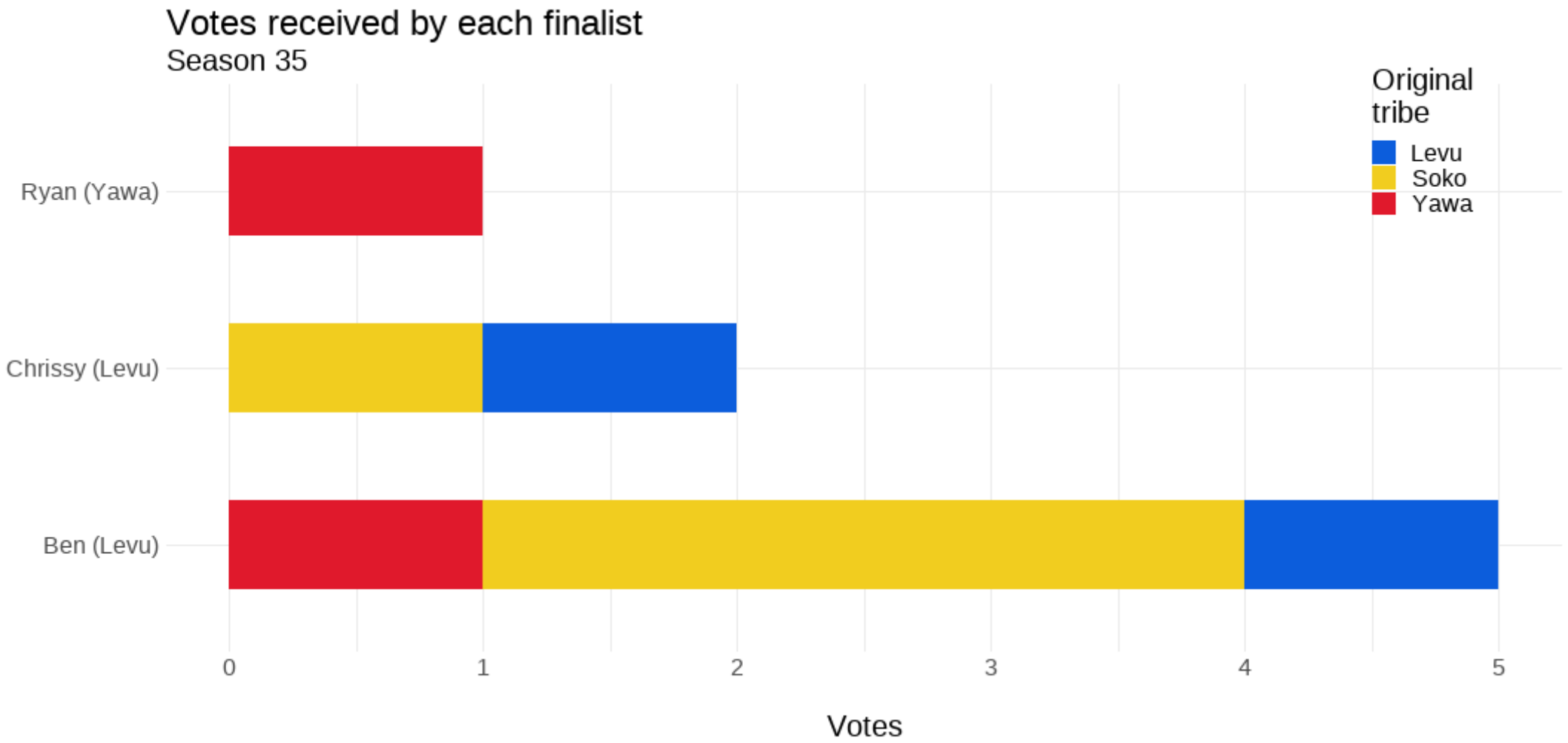
Viewership Over Time

```
ggplot(p1_dat, aes(x = season, y = viewers, colour = episode)) +
  geom_line(size = 2) + # Note 'size' is not in aes()
  geom_point(size = 4) + # geom_point after geom_line overlays points on lines
  theme_minimal() + # Many theme options, this keeps minor axes obly
  scale_colour_survivor(16) + # Custom palette with colors from the show
  labs(
    title = "Survivor viewers over the 40 seasons",
    x = "\nSeason", # '\n' is shorthand for 'newline'
    y = "Viewers (Millions)\n", # '\n' adds space between axis title and ticks
    colour = "Episode"
  ) +
  theme(
    text = element_text(size = 30), # Changes the size for all text elements
    legend.position = c(0.8, 0.7)   # Puts the legend in the top right corner
  )
```


Survivor viewers over the 40 seasons
Viewers (Millions)
50
40
30
20
10
0
10
20
30
40
Season
Episode
finale
mean
premier
reunion

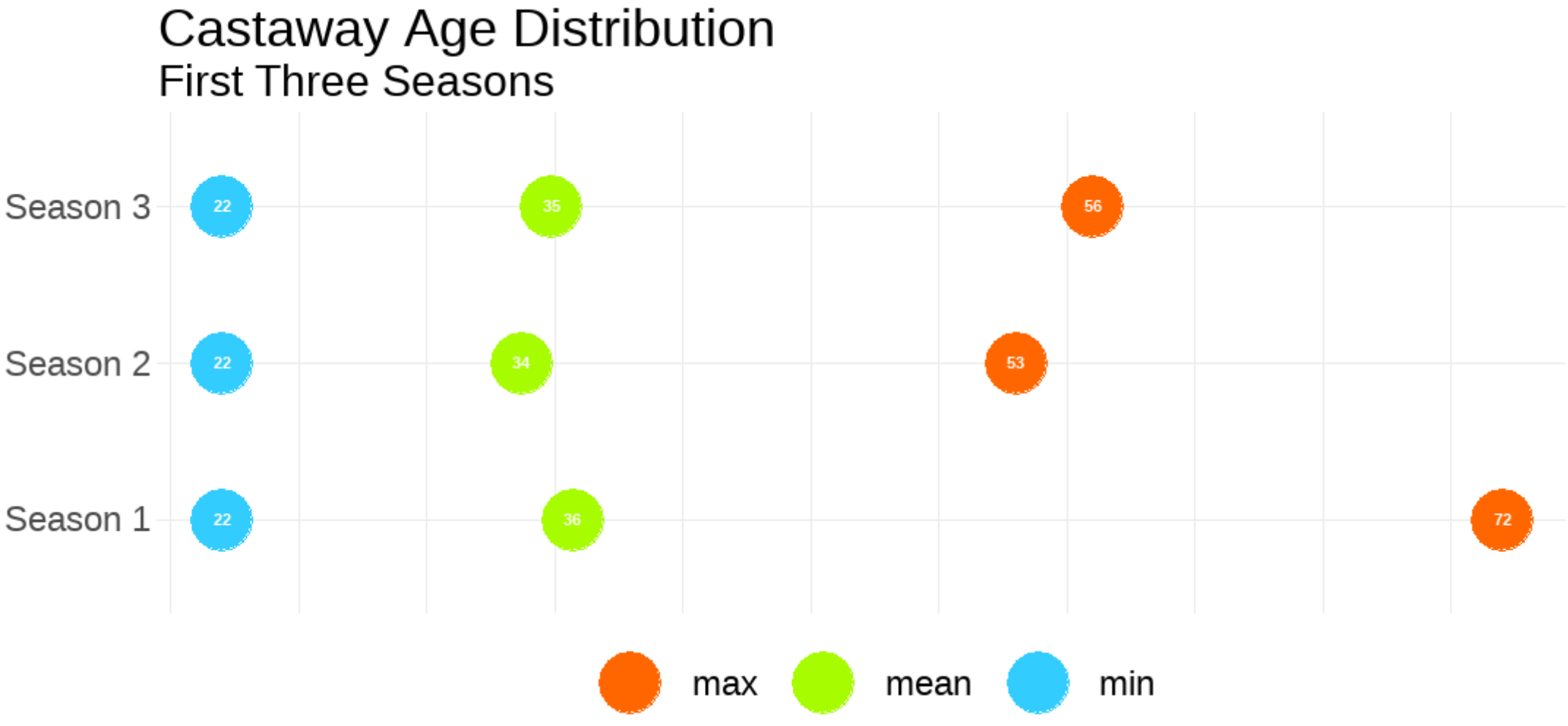
Votes Received by Each Finalist

```
library(glue)

labels <- castaways %>%
  filter(season == 35, str_detect(result, "Sole|unner")) %>%
  mutate(label = glue("{castaway} ({original_tribe})")) %>%
  select(label, castaway)

jury_votes %>%
  filter(season == 35) %>%
  left_join(castaways %>% filter(season == 35) %>% select(castaway, original_tribe), by = "castaway'
  group_by(finalist, original_tribe) %>%
  summarise(votes = sum(vote)) %>%
  left_join(labels, by = c("finalist" = "castaway")) %>%
  {ggplot(., aes(x = label, y = votes, fill = original_tribe)) +
      geom_bar(stat = "identity", width = 0.5) +
      scale_fill_survivor(35, tribe = .$original_tribe) +
      coord_flip() +
      theme_minimal() +
      labs(x = "", y = "\nVotes", fill = "Original\ntribe",
           title = "Votes received by each finalist", subtitle = "Season 35") +
      theme(legend.position = c(0.9, 0.9),
            text = element_text(size = 20))}
```


Votes received by each finalist
Season 35
Original tribe
Levu
Soko
Yawa
Ryan (Yawa)
Chrissy (Levu)
Ben (Levu)
0
1
2
3
4
5
Votes

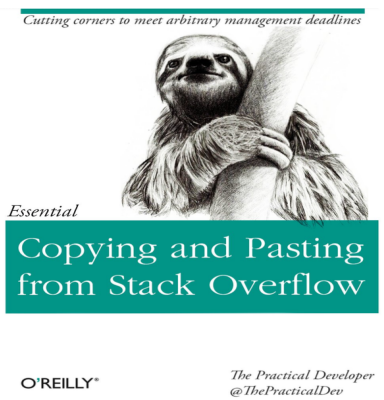
Age Distribution by Season

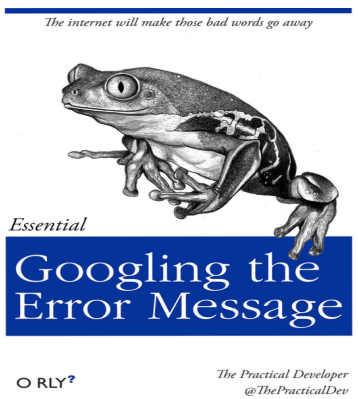
```
cols <- RColorBrewer::brewer.pal(3, "Blues")

castaways %>%
  filter(season %in% 1:3) %>%
  group_by(season) %>%
  summarise(min_age  = min( age, na.rm = T),
            mean_age = mean(age, na.rm = T),
            max_age  = max( age, na.rm = T)) %>%
  pivot_longer(min_age:max_age) %>%
  mutate(name = name %>% str_replace("_age", "")) %>%
  ggplot(aes(x = value, y = paste("Season", season), color = name)) +
    theme_minimal() +
    geom_point(size = 20) +
    geom_text(aes(label = round(value)), colour = "white", fontface = "bold") +
    scale_colour_survivor(2) +
    labs(title = "Castaway Age Distribution",
         subtitle = "First Three Seasons",
         y = NULL,
         colour = NULL) +
    theme(text = element_text(size = 30),
          axis.text.x = element_blank(),
          axis.title.x = element_blank(),
```

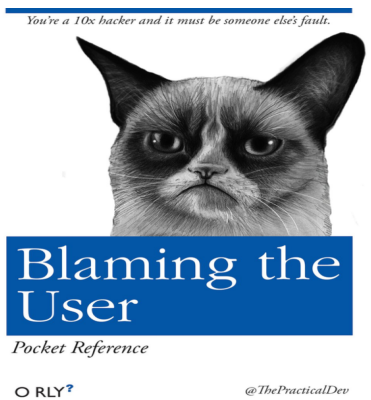

Castaway Age Distribution
First Three Seasons
Season 3
Season 2
Season 1
22
35
56
22
34
53
22
36
72
max
mean
min

Getting Help

- Use `R`'s built-in help documentation (e.g. > ?ggplot)
- Go to the `ggplot2` website: ggplot2.tidyverse.org/reference/
- Read the creator's (Hadley Wickham) book: *ggplot2: Elegant Graphics for Data Analysis*, second edition
- Google (my preferred method)







arstechnica.com/gaming/2017/10/tim-oreilly-on-why-the-future-probably-wont-be-all-that-be-terrible/

References

These Slides

- Adapted from "ggplot2," a presentation by Matthew Flickinger, Ph.D. on Sept 12, 2018
- Slide were produced in `R` using the `xaringan` package: github.com/yihui/xaringan

The DatasauRus Dozen

- https://cran.r-project.org/web/packages/datasauRus/vignettes/Datasaurus.html
- https://www.autodesk.com/research/publications/same-stats-different-graphs

SurvivoR Data

- gradientdescending.com/survivor-data-from-the-tv-series-in-r/
- github.com/doehm/survivoR

Data Visualizations

- stephanieevergreen.com

Questions?

salernos@umich.edu

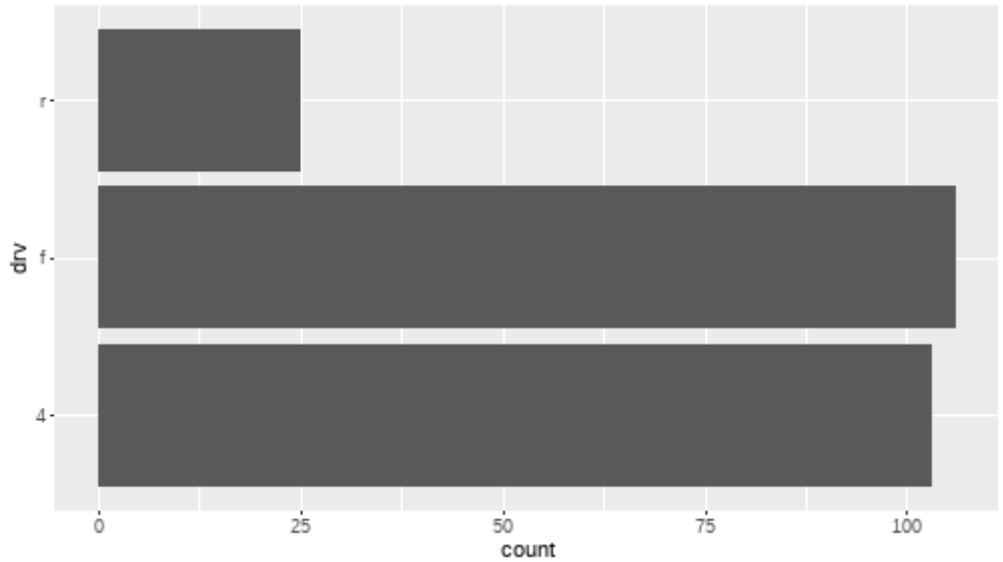
Potpourri

(time permitting)

Coordinate Transformations

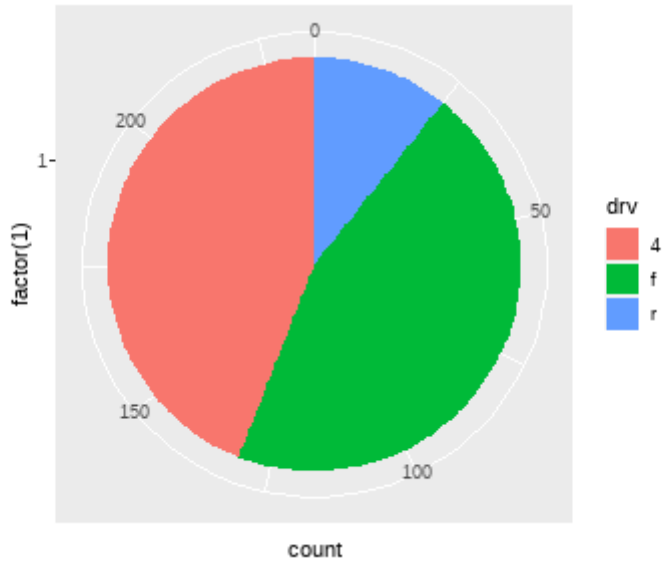
`coord_flip()`: Swap x & y axes

```
ggplot(mpg, aes(x=drv)) +
  geom_bar() +
  coord_flip()
```



`coord_polar()`: Make "pie" charts

```
ggplot(mpg) +
  geom_bar(aes(factor(1),fill=drv),width=1) +
  coord_polar(theta="y")
```

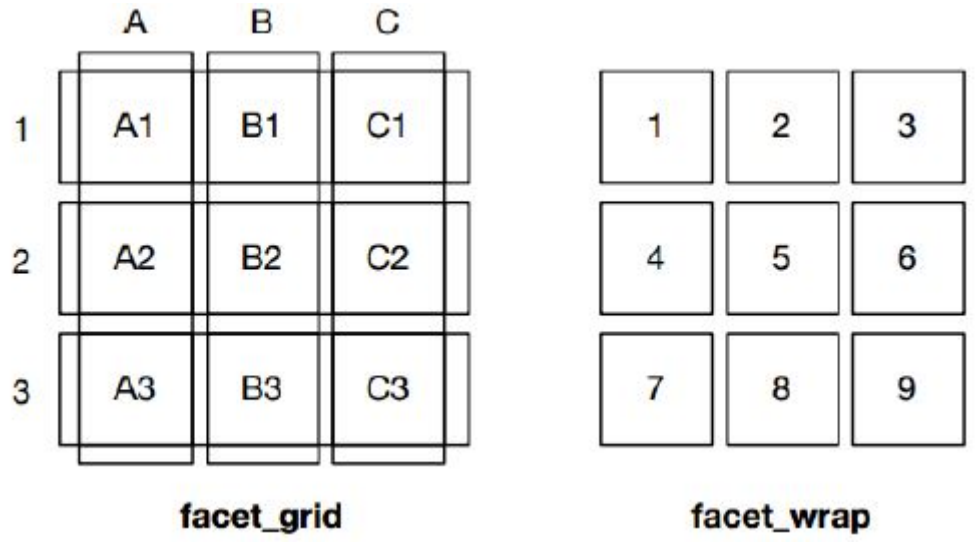


Coordinate Transformations

- `coord_cartesian()`: limit the plotting window
  - xlim = range of x values c(lower, upper)
  - ylim = range of y values c(lower, upper)
  - Differs from changing limits on scales which will subset data
- `coord_fixed ()`: fix the distance ratio for x and y

Faceting

- Divide plot into subgroups and draw layers for each subset of data based on faceting variables
- Two primary options:
  - `facet_grid()`: Variables for rows/columns
  - `facet_wrap()`: No panel structure
- Each facet gets all layers
- Each layer's data split on same variables



facet_wrap()

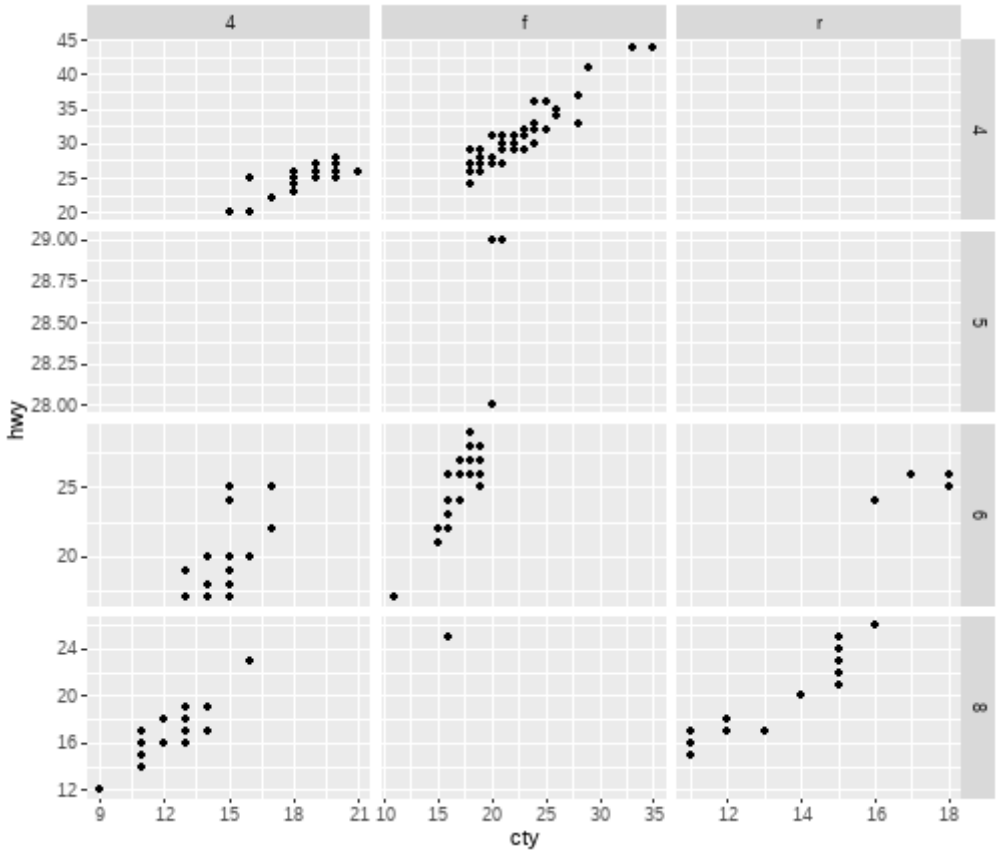
- Requires a right hand formula: `~ var1 + var2`
- Each combination of columns gets it own panel
- `scales=` options:
  - "fixed": scales same in all
  - "free_x": x range can vary
  - "free_y": y range can vary
  - "free" domain and range can change for each panel
- `ncol` = number of columns

```
library(gapminder)
ggplot(gapminder, aes(x=lifeExp)) +
  geom_density(fill="grey40") +
  facet_wrap(~continent)
```

facet_grid()

- Requires two arguments:
  - rows = vars(var1)
  - cols = vars(var2)
- Shared axis across panels
- Results in "rectangular" output

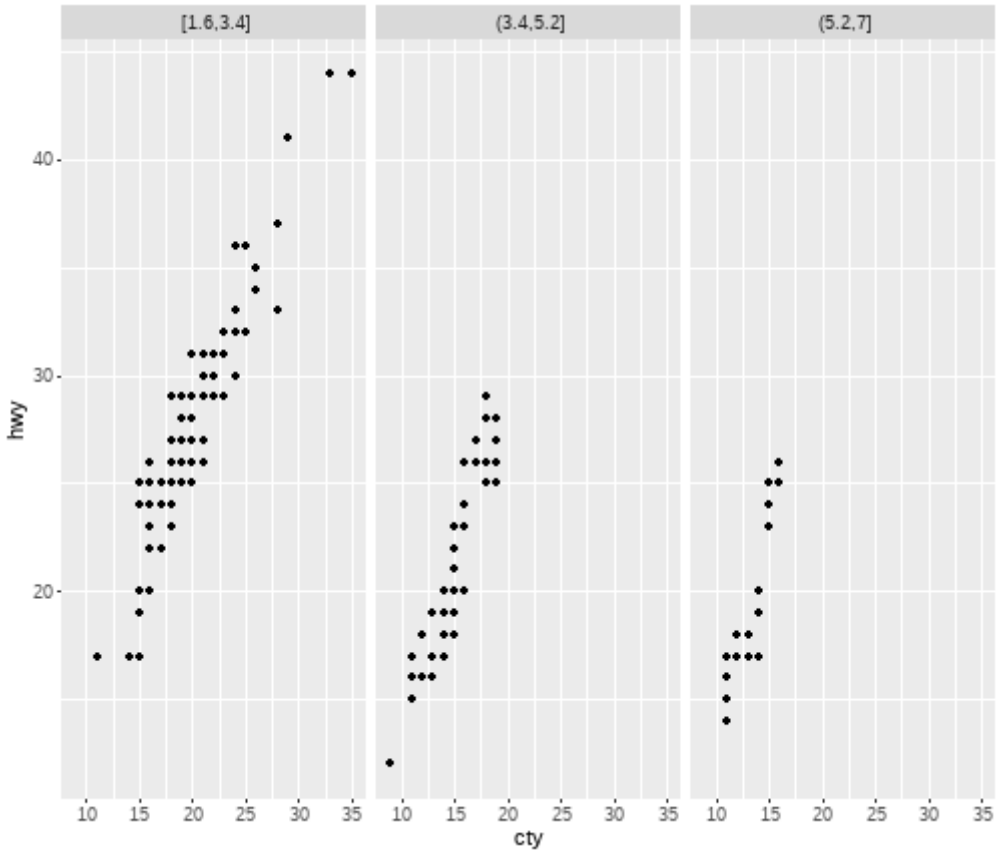
```
ggplot(mpg, aes(cty, hwy)) +
  geom_point() +
  facet_grid(rows=vars(cyl),
             cols=vars(drv),
             scales="free"
  )
```



Faceting with Continuous Variables

- Faceting works best with categorical variables
- To use with categorical variables, we cut the values into groups
- Useful helper functions:
  - **Bins of same length**: `cut_interval()`
  - **Bins of a certain width**: `cut_width()`
  - **Bins with roughly equal counts**: `cut_number()`
- For example:

```
mpg %>%
  mutate(d = cut_interval(displ, 3)) %>%
  ggplot(aes(cty, hwy)) +
  geom_point() +
  facet_wrap (~d)
```



Legend

- Only mapped `aes()` values appear in the legend
- You can disable the legend for a layer with `show.legend=FALSE`
- You can also control where the legend goes:
  - "Outside"
    - `+ theme(legend.postion="right")`
    - `+ theme(legend.postion="bottom")`
  - "Inside"
    - `+ theme(legend.postion=c(0, 1))`
    - (0,0) is lower left, (1,1) is top right, can be any number in between
  - No Legend: `+ theme(legend.postion="none")`

Legend Guides

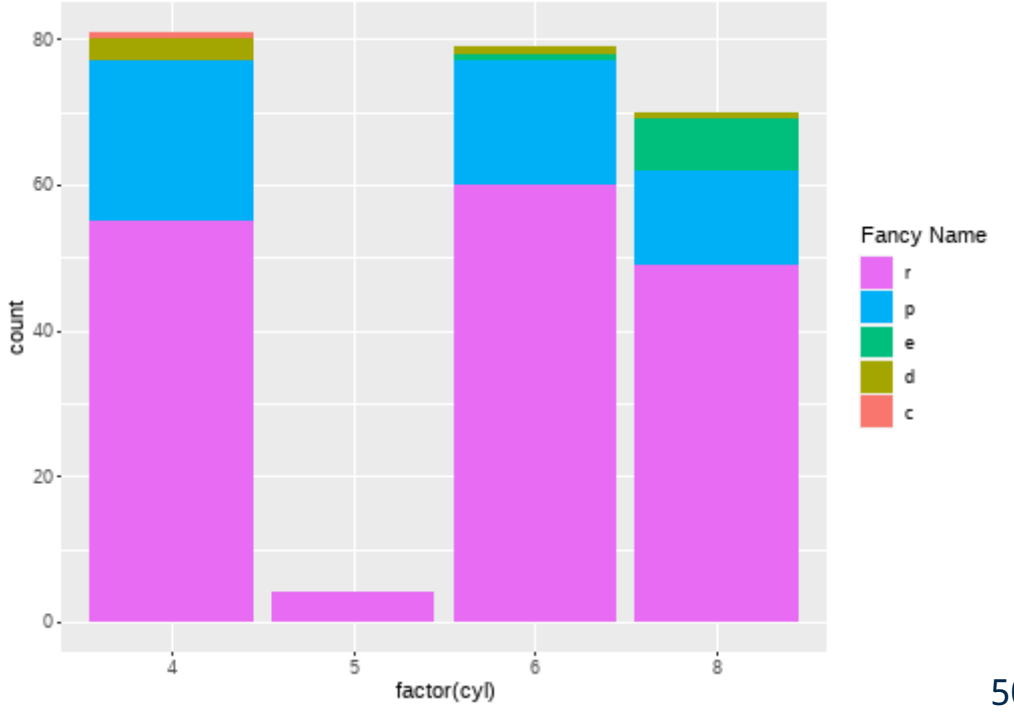
- The `guides()` function can control how keys are drawn in the legend
- Layout key values:

```
# Create more than 1 column
+ guides(fill = guide_legend(ncol=2))

# Change row direction
+ guides(fill = guide_legend(ncol=2, byrow=T))

# Reverse order:
+ guides(fill = guide_legend(reverse=T))
```

- Remove guide: `+ guides(color="none")`
- Set title:

```
+ guides(color=guide_legend("Name"))
+ labs(color="Name")
```

```
ggplot(mpg, aes(factor(cyl), fill=fl)) +
  geom_bar() +
  guides(fill=guide_legend(reverse=T)) +
  labs(fill="Fancy Name")
```



Scales

- Scales describe how raw data values should be converted to aesthetic values
- Default scales are determined by the class of the variables in your data
- Each aesthetic (e.g. color, fill, size, shape) can have at most one scale
- Scales can have guides:
    - Axes for positions
    - Legends for everything else
- Color is mapped differently depending if year is a numeric or character vector
- Default color scales
    - **Numeric:** `scale_color_continuous()`
    - **Factor:** `scale_color_discrete()`

Scaling Based on Data Type

```
p <- gapminder %>%
  filter(country=="United States") %>%
  ggplot(aes(gdpPercap, lifeExp))
```

- Compare the following output based on whether `year` is continuous or a factor

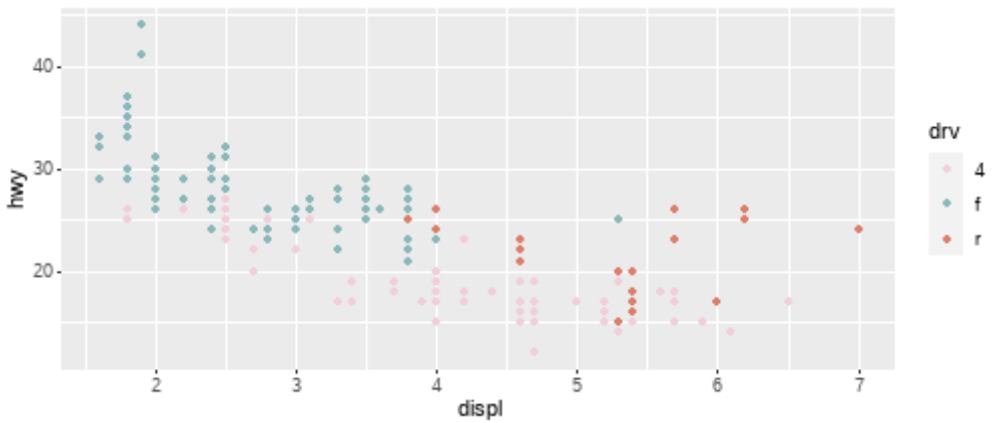
```
p + geom_point(aes(color=year))
```

```
p + geom_point(aes(color=factor(year)))
```

Manually Setting Discrete Colors

- You can customize the default color scales
- Or you can create your own manual scale
  - http://colorbrewer2.org/
  - `RColorBrewer::display.brewer.all()`
- Get RGB/HEX values from anywhere
  - http://colormind.io/
  - http://color.adobe.com
- `scale_color_manual()` expects:
  - Vector of colors
  - Named vector of colors
  - Vector of colors for levels named in breaks

```
p <- ggplot(mpg, aes(displ, hwy)) +
  geom_point(aes(color=drv))

cust_cols <- c("4"="#F2CED8",
               "f"="#88B8B8",
               "r"="#DE7E68")

p + scale_color_manual(values=cust_cols)
```



Manually Setting Continuous Colors

- Continuous values are plotted with gradients:

```
scale_color_gradient()  # 2 Color
scale_color_gradient2() # 3 Color
scale_color_gradientn() # N Color
```

- Examples:

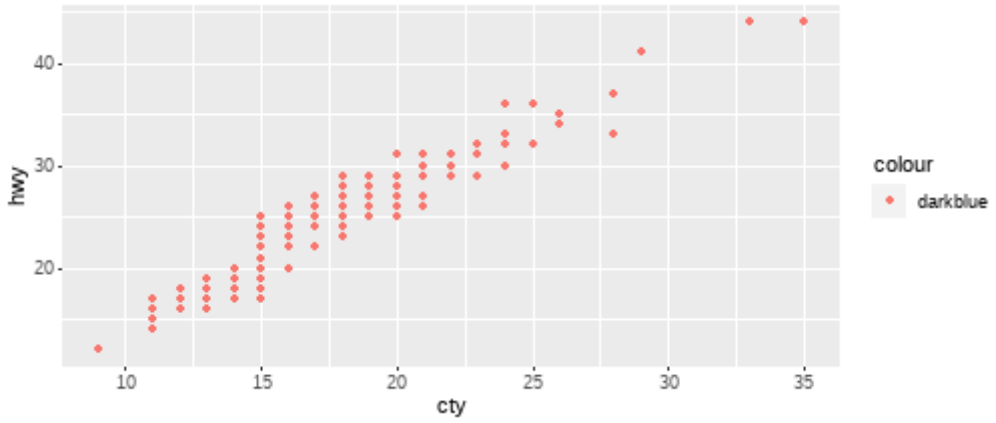
```
p <- ggplot (mpg, aes(cty, hwy)) +
  geom_point(aes(color=displ))

p + scale_color_gradient(low="white",
                         high="orchid")

p + scale_color_gradient2(low="white",
                          high="orchid",
                          mid="tomato")

p + scale_color_gradientn(colors=c("blue",
                                   "wheat",
                                   "green"))
```

Setting v.s. Mapping

- Notice the difference that `color=` makes inside v.s. outside the `aes()` function
- Only things inside an `aes()` get a legend (only the mappings)
- If you have a column that has color values, use `scale_color_identity()` to prevent remapping

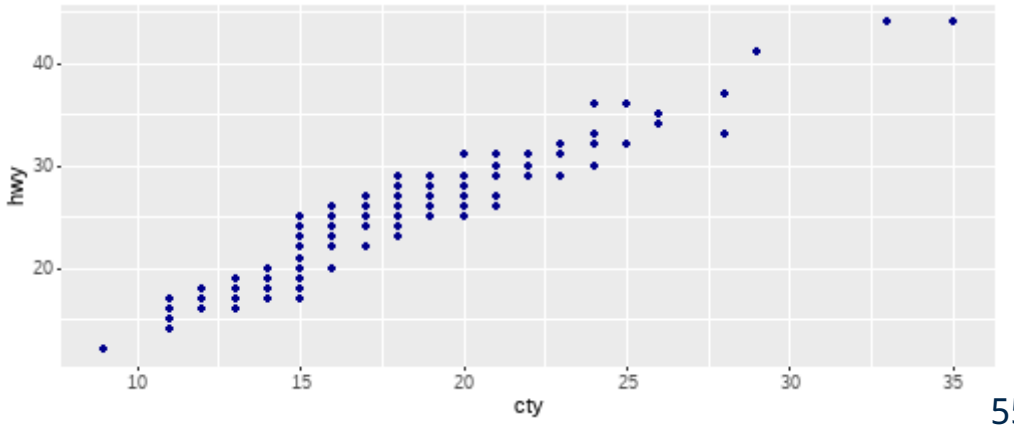
Odd Behavior

```
ggplot(mpg, aes(cty, hwy)) +
  geom_point(aes(color = "darkblue"))
```



Correct Behavior

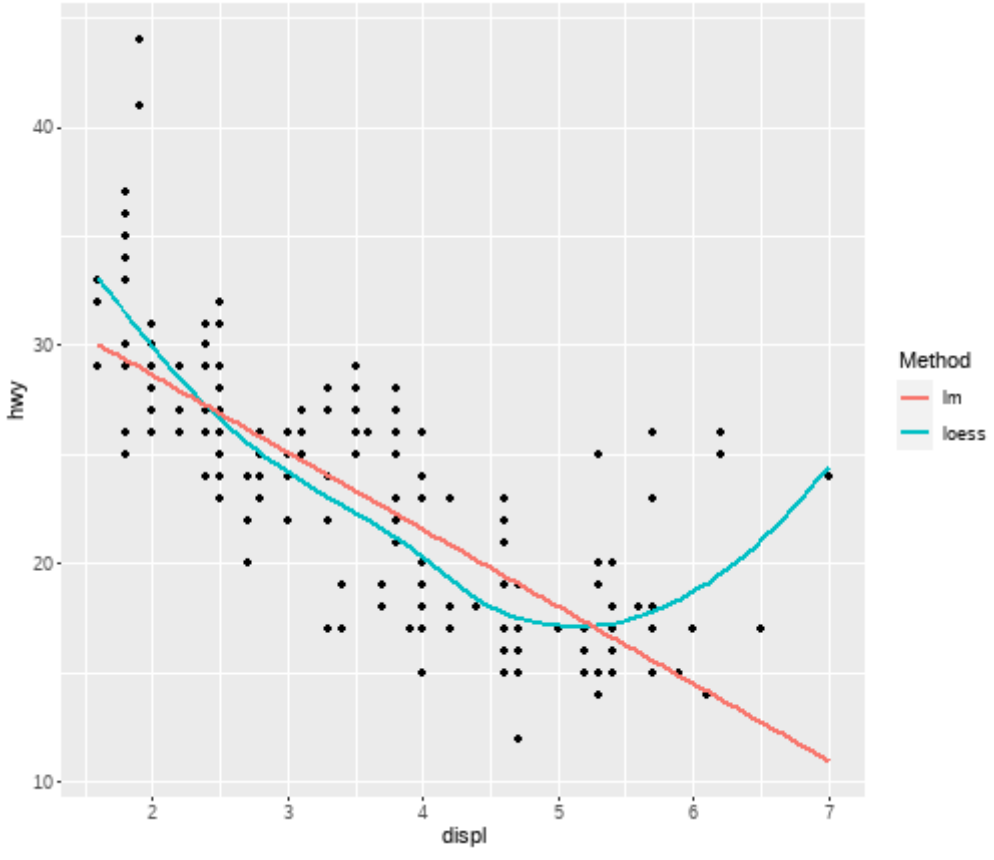
```
ggplot(mpg, aes(cty, hwy)) +
  geom_point(color="darkblue")
```



Literal String Mappings

- Specifying a literal mapping can be useful if using multiple layers
- Here we add two layers with different smoothers
- Specify `color=` in the `aes()` for a nice legend

```
ggplot(mpg, aes(displ, hwy)) +
  geom_point() +
  geom_smooth(aes(color = "loess"),
    method = "loess", se = FALSE) +
  geom_smooth(aes(color = "lm"),
    method = "lm", se = FALSE) +
  labs(color = "Method")
```



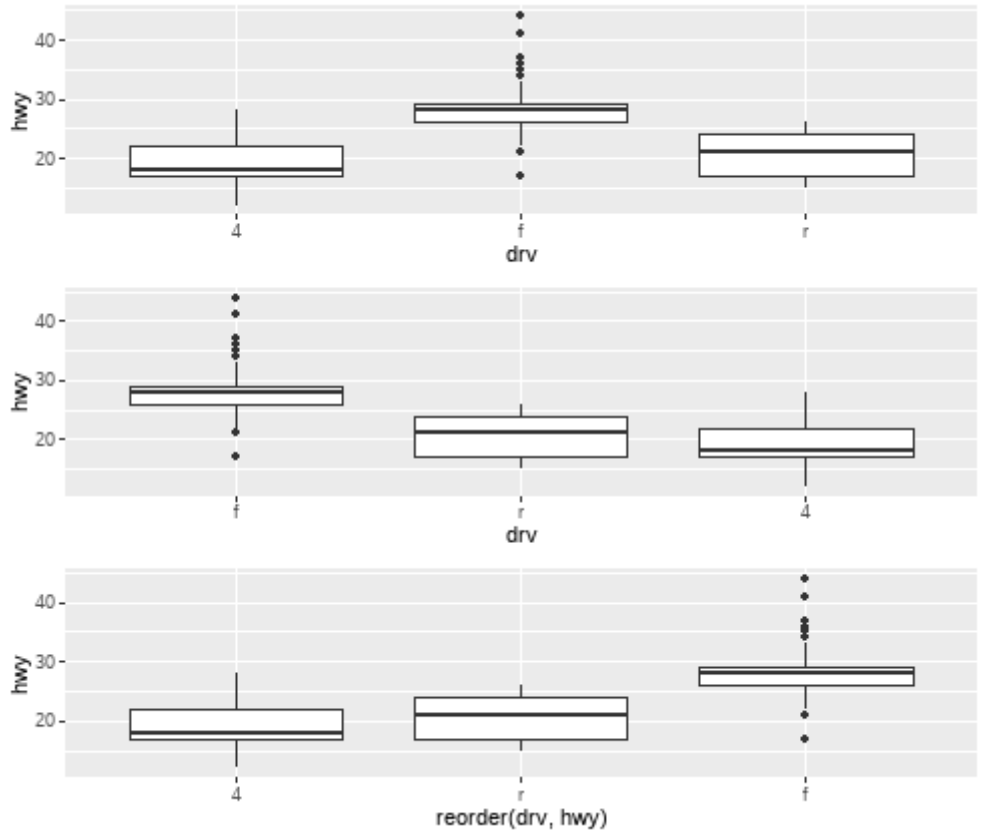
Axes are Scales!

- You can change transformations of x/y axes via scales
  - `scale_x_log10()`
  - `scale_x_sqrt()`
  - `scale_x_reverse()`
- Can also take finer control over tick marks
  - Numeric Values: `scale_x_continuous()`
  - Date/Time Values: `scale_x_datetime()`
- Control display of factor levels
  - `scale_x_discrete()`
  - Choose new labels for factor levels

Discrete Axis Plotting Order

- Discrete axes are drawn in the order of the levels() of the corresponding factor
- You can change that order by changing the axis scale
- Or you can re order the factor itself (see `?reorder`)

```
p <- ggplot(mpg, aes(y=hwy))

# Default
p + geom_boxplot(aes(drv))

# Use Scale
p + geom_boxplot(aes(drv)) +
  scale_x_discrete(limits=c("f","r","4"))

# Use Data
p + geom_boxplot(aes(reorder(drv, hwy)))
```

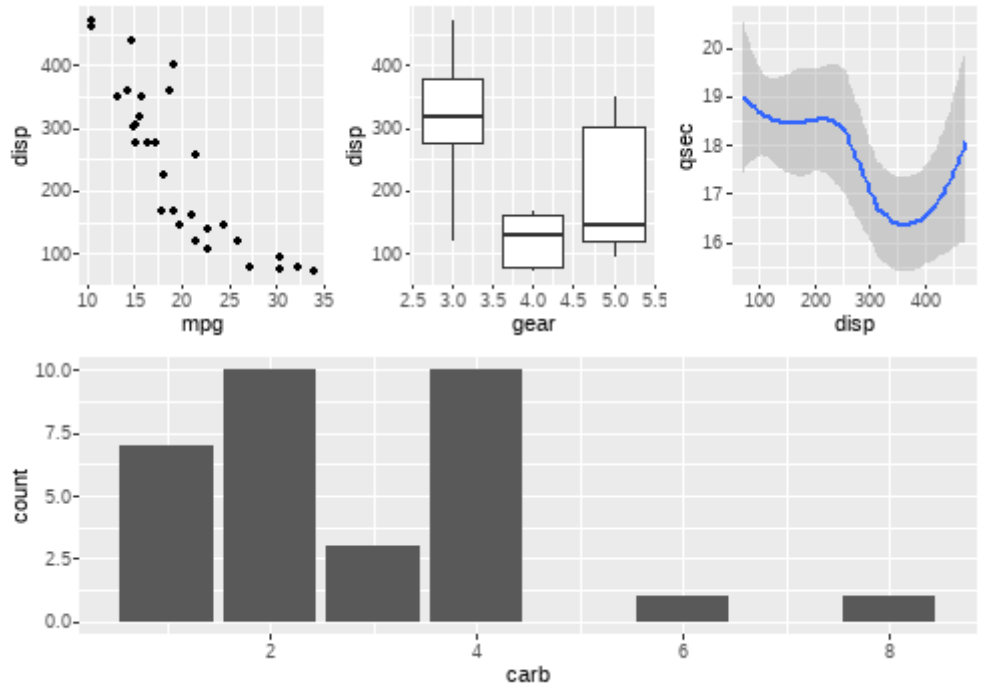


Maps

Arranging Multiple Plots

- The `patchwork` package makes it incredibly simple to combine separate ggplots into the same graphic
- Uses arithmetic operators (e.g. +, |, /) and order of operations to combine ggplot objects

```
library(patchwork)

p1 <- ggplot(mtcars) +
  geom_point(aes(mpg, disp))

p2 <- ggplot(mtcars) +
  geom_boxplot(aes(gear, disp, group = gear))

p3 <- ggplot(mtcars) +
  geom_smooth(aes(disp, qsec))

p4 <- ggplot(mtcars) +
  geom_bar(aes(carb))

# Put them all together!

(p1 | p2 | p3) / p4
```



Vector v.s. Raster

- Two main image format categories
- Vector images:
  - Remembers the shapes drawn
  - Infinitely zoom-able
  - pdf, svg eps,
- Raster/bitmap images:
  - Remembers just the pixels of the image
  - Number of pixels depends on the dots per inch (DPI) of your image
- Typically vector is better, but if you have lots of points, raster may be easier to work with

ggsave()

- Once you've made your masterpiece, use `ggsave()` to save it
- It will create a file in your current working directory (`getwd()`/`setwd()`)
- Saves last plot printed
- Looks at file name to determine type:

```
ggsave("plot.pdf");  ggsave("plot.eps");
ggsave("plot.png");  ggsave("plot.jpg");
ggsave("plot.tiff"); ggsave("plot.svg");
```

- Can also pass plot object to save any plot:

```
ggsave("plot.png", ggplot(mpg, aes(cty, hwy)) + geom_point())
```

Thank You!