The Basics of R

Day 2 – Charts & Analysis

AFS Alaska Workshop

An Introduction to the R Programming Language for Fishery Biologists

February 2022

Instructor: Justin Priest

https://github.com/justinpriest/R_Intro_AFS/

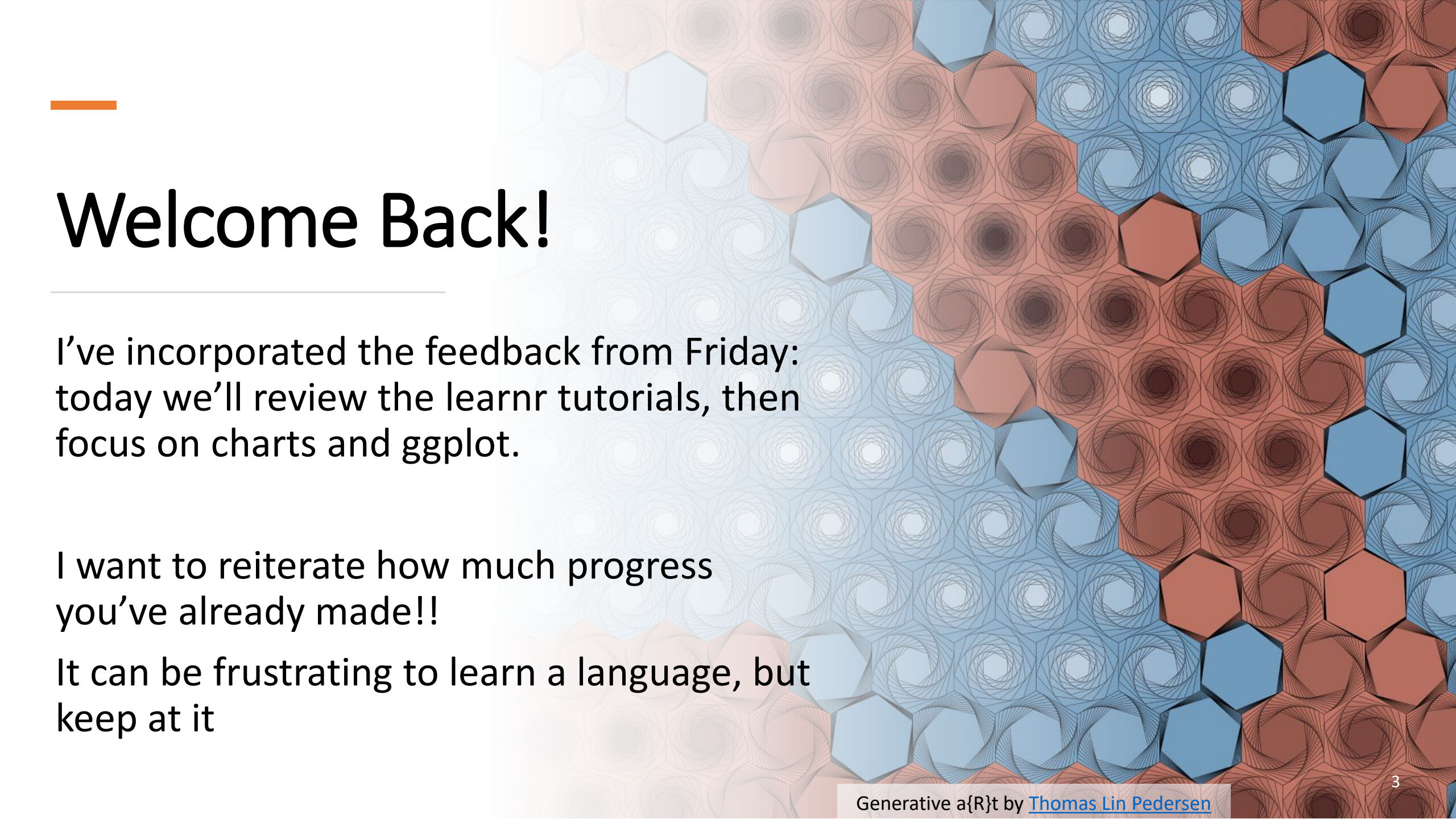
NOTE: this is the PDF version of the presentation. Much of this presentation relies on animations so you may not see everything!

Welcome Back!

I've incorporated the feedback from Friday: today we'll review the learnr tutorials, then focus on charts and ggplot.

I want to reiterate how much progress you've already made!!

It can be frustrating to learn a language, but keep at it



Generative a{R}t by Thomas Lin Pedersen

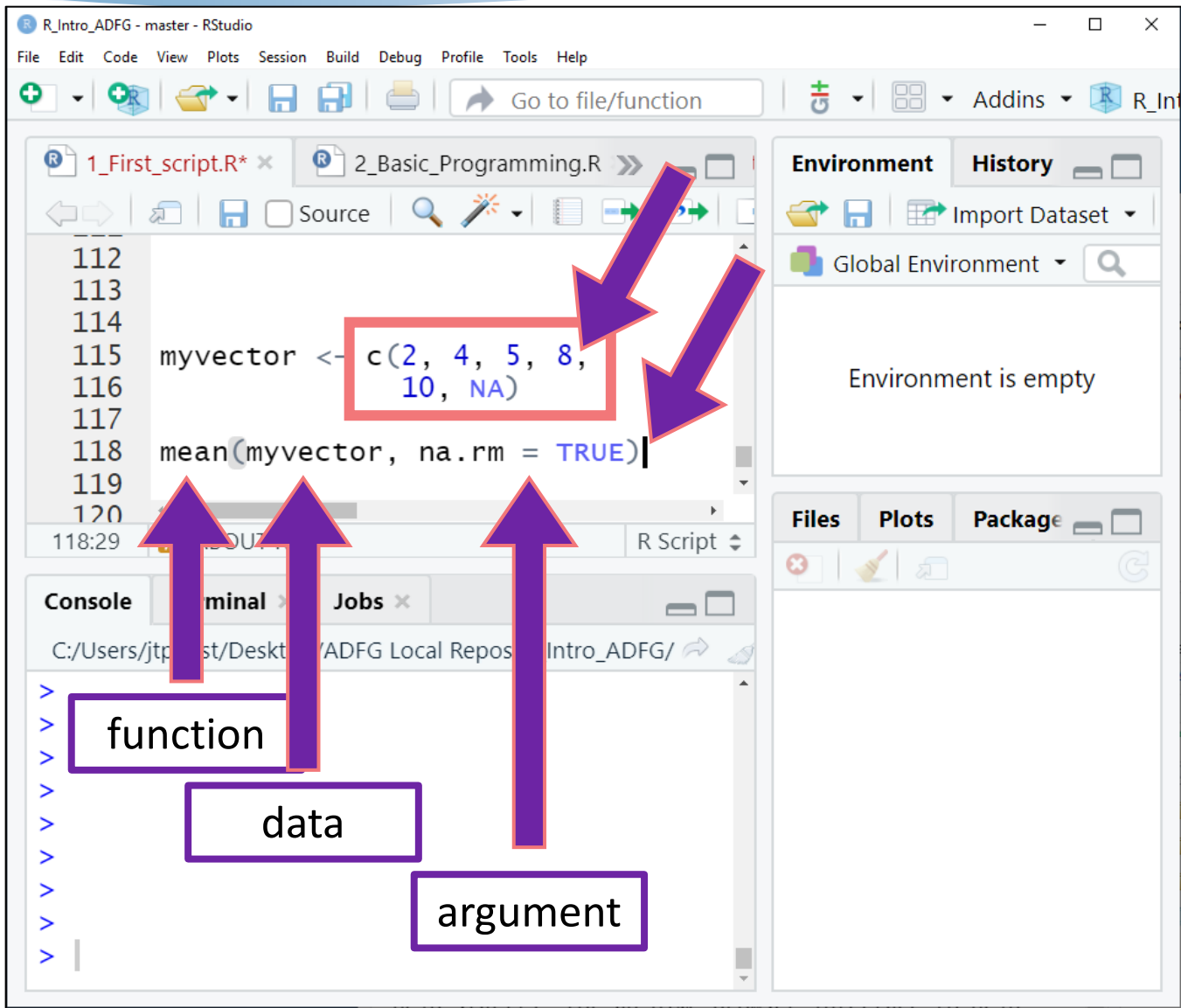
Today's Agenda

09:00–10:00	Review & learnr tutorials
10:00–12:00	Let's Make Charts
12:00–13:30	Lunch
13:30–14:30	Basic Analysis & Tidyverse (*presentation only*)
14:30–17:00	Project & Concluding Thoughts

A Brief Review

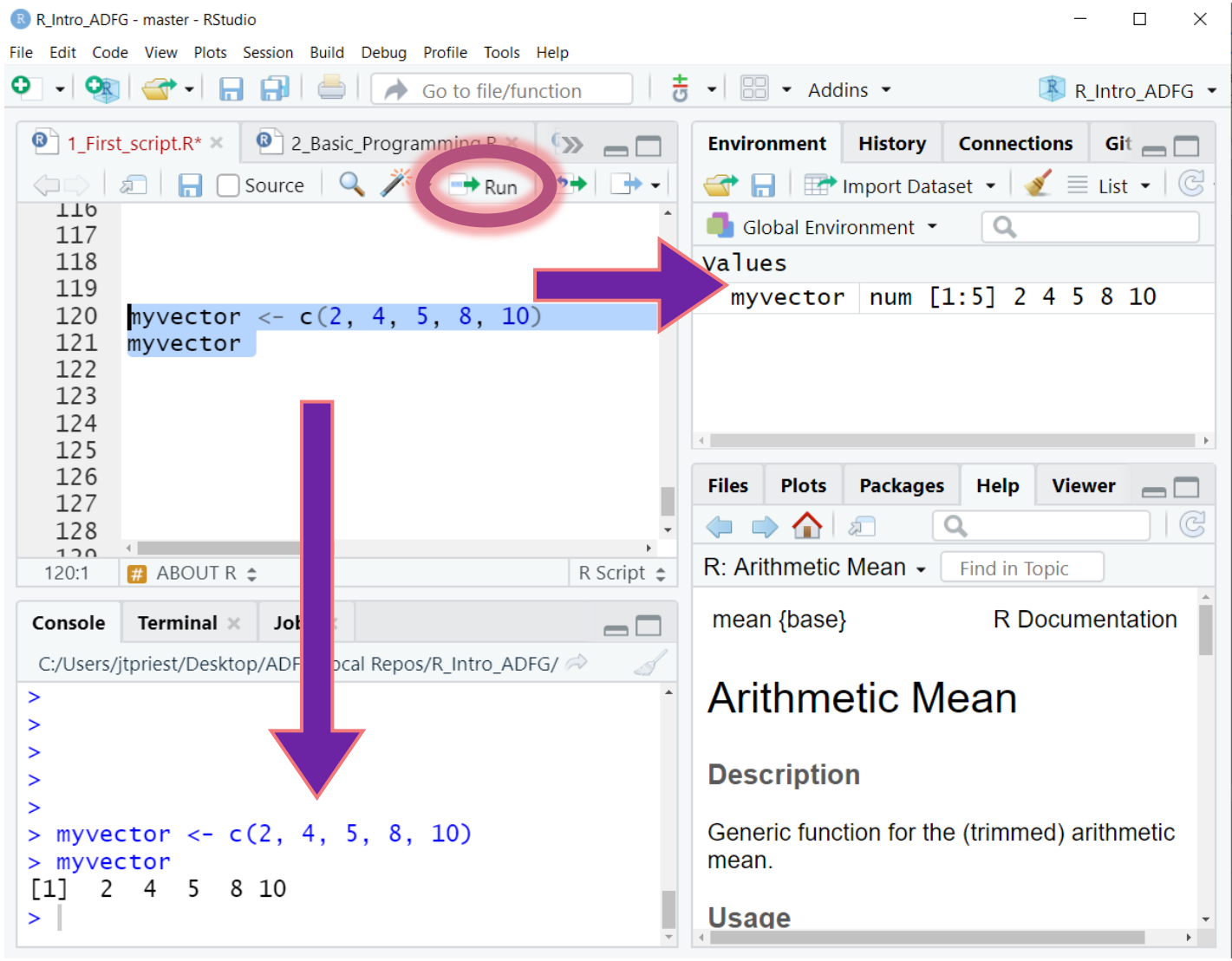
Review: General

- A function has arguments
  - mean(dfname, **na.rm = TRUE**)
- Use c() to concatenate items
- Code lines must end in a comma, pipe, or be completed



Review: General

- Use “<-” to save a dataframe
  - Otherwise code doesn’t save for access later (which is often fine!)
- Ctrl + Enter (or Run Button) evaluates the line of code (sending it to console)
  - Put cursor somewhere in that line, or highlight the chunk you want
- Objects in the environment disappear once we close R. We’ll need to re-run whole script to get them back. They don’t “live” anywhere permanently.



Review: Libraries

A library is a new group of functions

- The first time, use install.packages("packagename")
- Every time you open a script use library(packagename) to load a library
- If you see:

```
> library(idonthavethispkgyet)
Error in library(idonthavethispkgyet) :
  there is no package called 'idonthavethispkgyet'
```

Run once!

```
> install.packages("dplyr")
```

```
> library(dplyr)
```

Run every time you open R

Review: Data

Reading in Data

- Use `read_csv("subfolder/filename.csv")`
- Problems? Did you use correct:
  - Filename
  - Subfolder name
  - Are you in the correct directory?
- Use str() to check the types
- Use View() to inspect dataframe

Refer back to this when doing own project later today

Data Cleanup

- Keep certain rows: **`filter()`**
- Keep certain cols: **`select()`**
- Rename columns: **`rename()`**
- Add new column: **`mutate()`**

Pipe %>%

```
dataframename <- filter(dataframename, Year == 2018)
dataframename <- mutate(dataframename, surveymonth = month(surveydate))
dataframename <- select(dataframename, Year, surveymonth, totalcount)
```

```
dataframename <- dataframename %>% filter(Year == 2018) %>%
                    mutate(surveymonth = month(surveydate)) %>%
                    select(Year, surveymonth, totalcount)
```

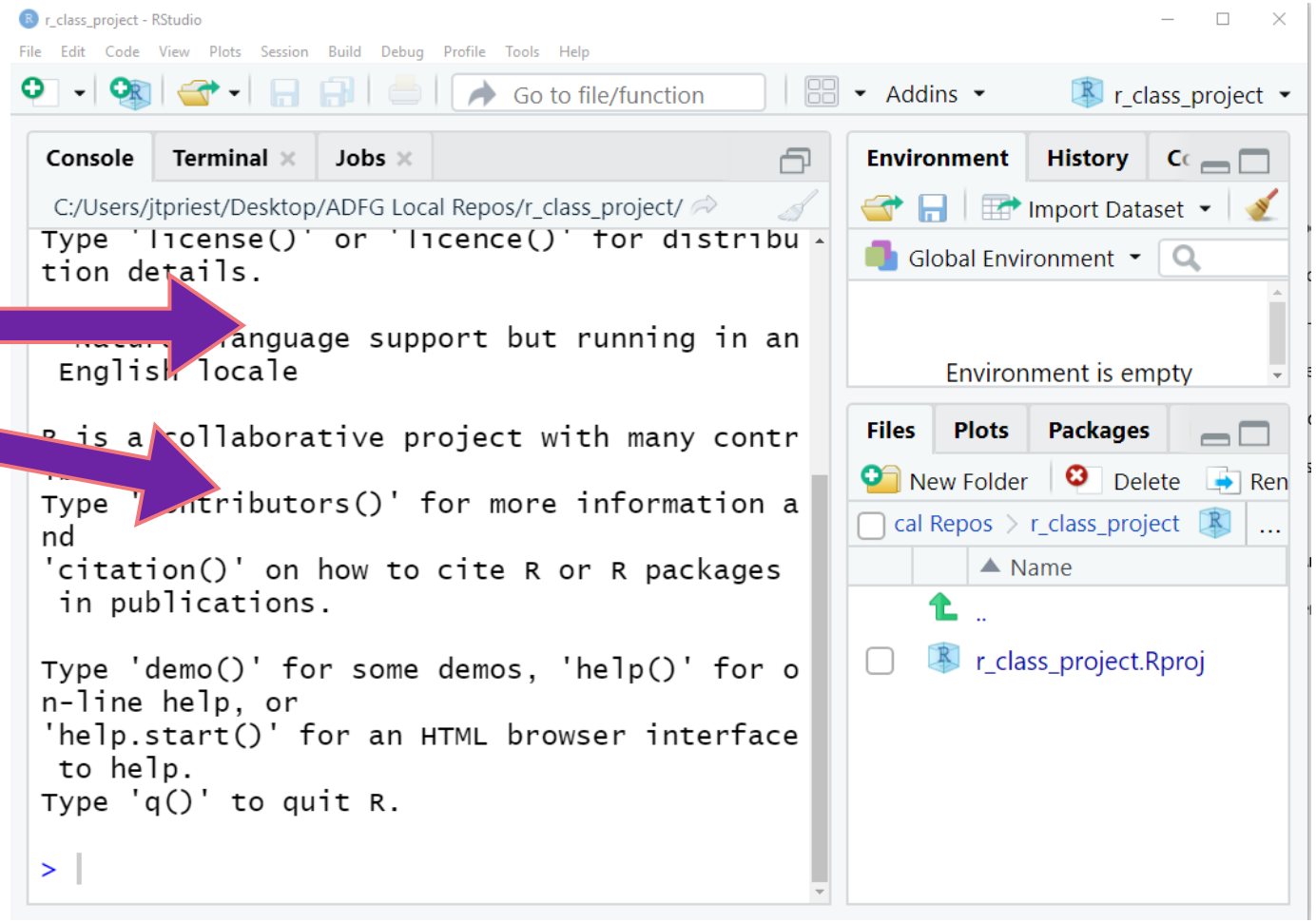
Creating a New Project

Let’s go over how to do everything from scratch!

Create a Project

- Open RStudio (any project or no project is fine)
- File -> New Project
  - Do you want to create a folder?
  - Do you have an existing folder?
- Choose New Project
- Give it a memorable name (no spaces) and choose what folder you want it saved in

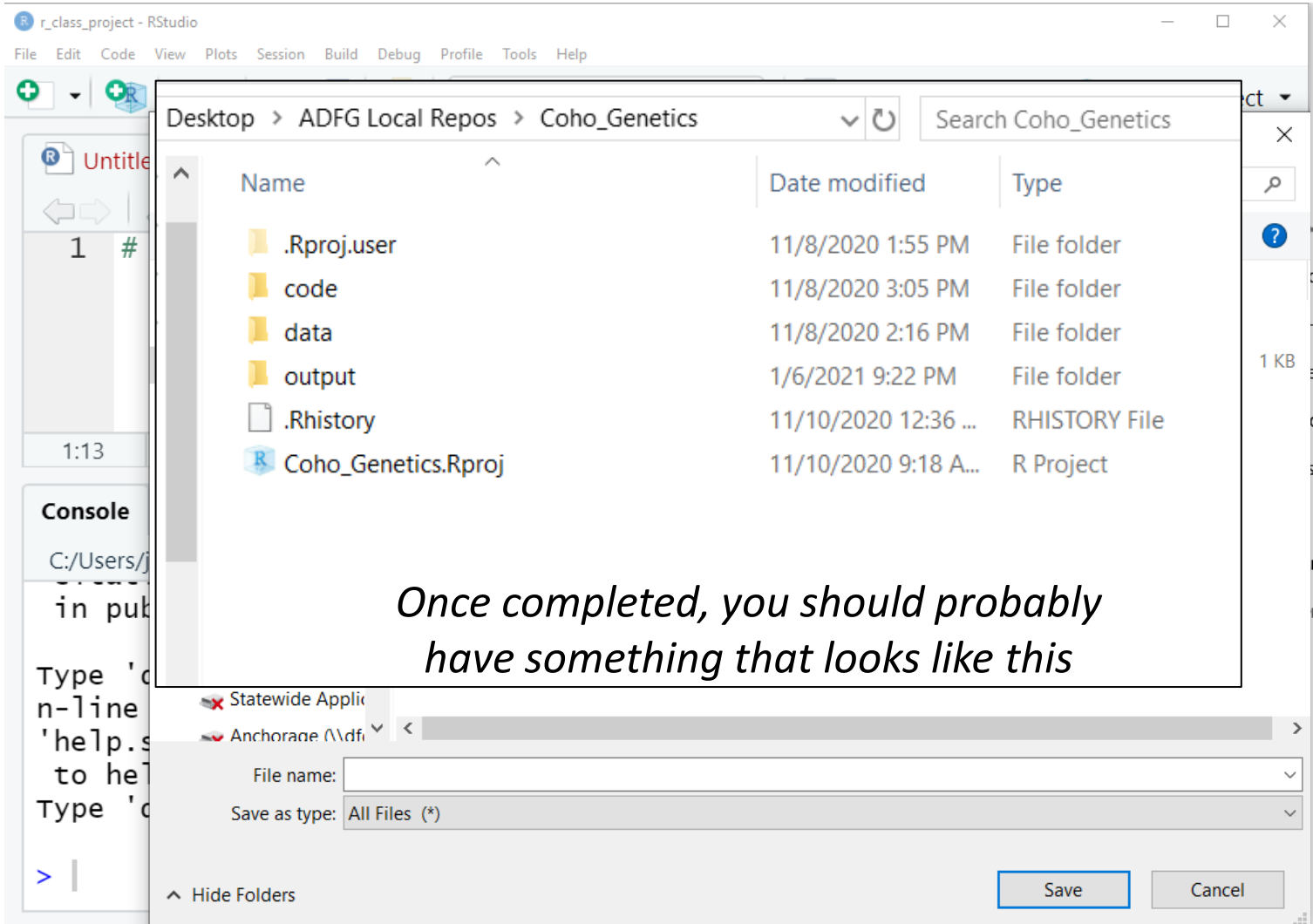
Success!



Create a Project, Part 2

- File --> New File --> New R Script (Ctrl+Shift+N)
- Write some code!
- Save file but make a new folder called “code” first!

- Open Windows Explorer folder. Create a folder named “data”, copy/paste some CSV files



5 – Let’s Make Charts!

Charts and plots and graphs, oh my!

Chart Types

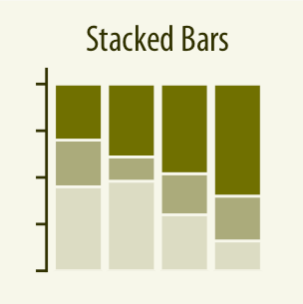
First, decide how to present your data

- Are you looking to show:
  - Trend over time
  - Composition
  - Compare / contrast
  - Distribution
  - Relationships

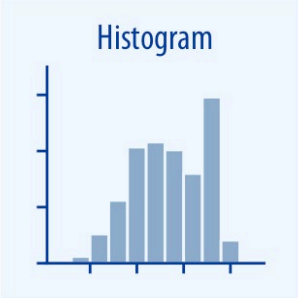
Often, I find it most useful to sketch out on paper what I want to see beforehand!

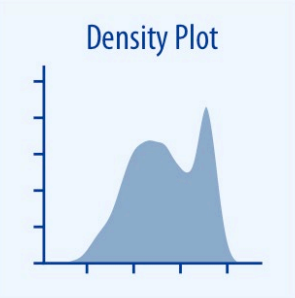
Chart Types

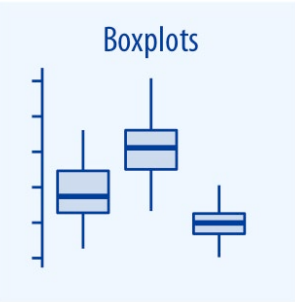
Proportion

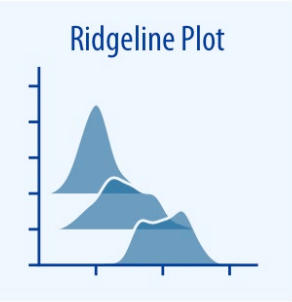


Distributions

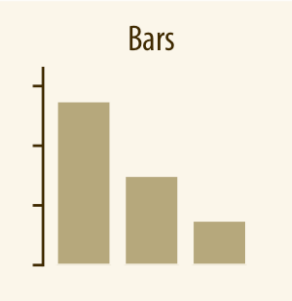


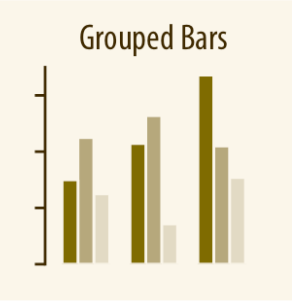


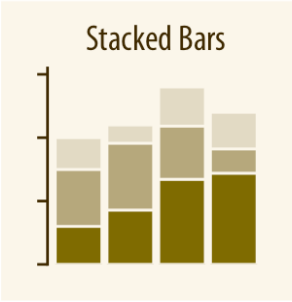




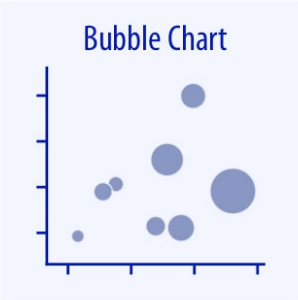
Amounts

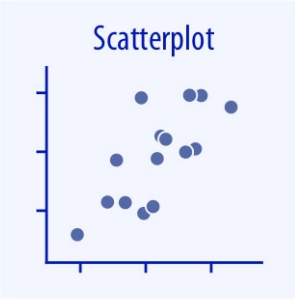


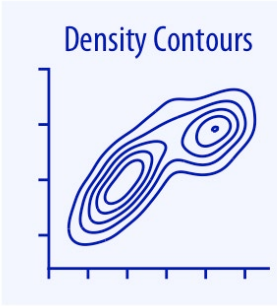


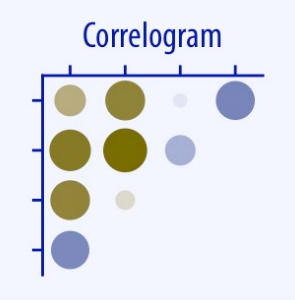


X-Y Relationships









Based on the book Fundamentals of Data Visualization

Chart Suggestions (incomplete list)

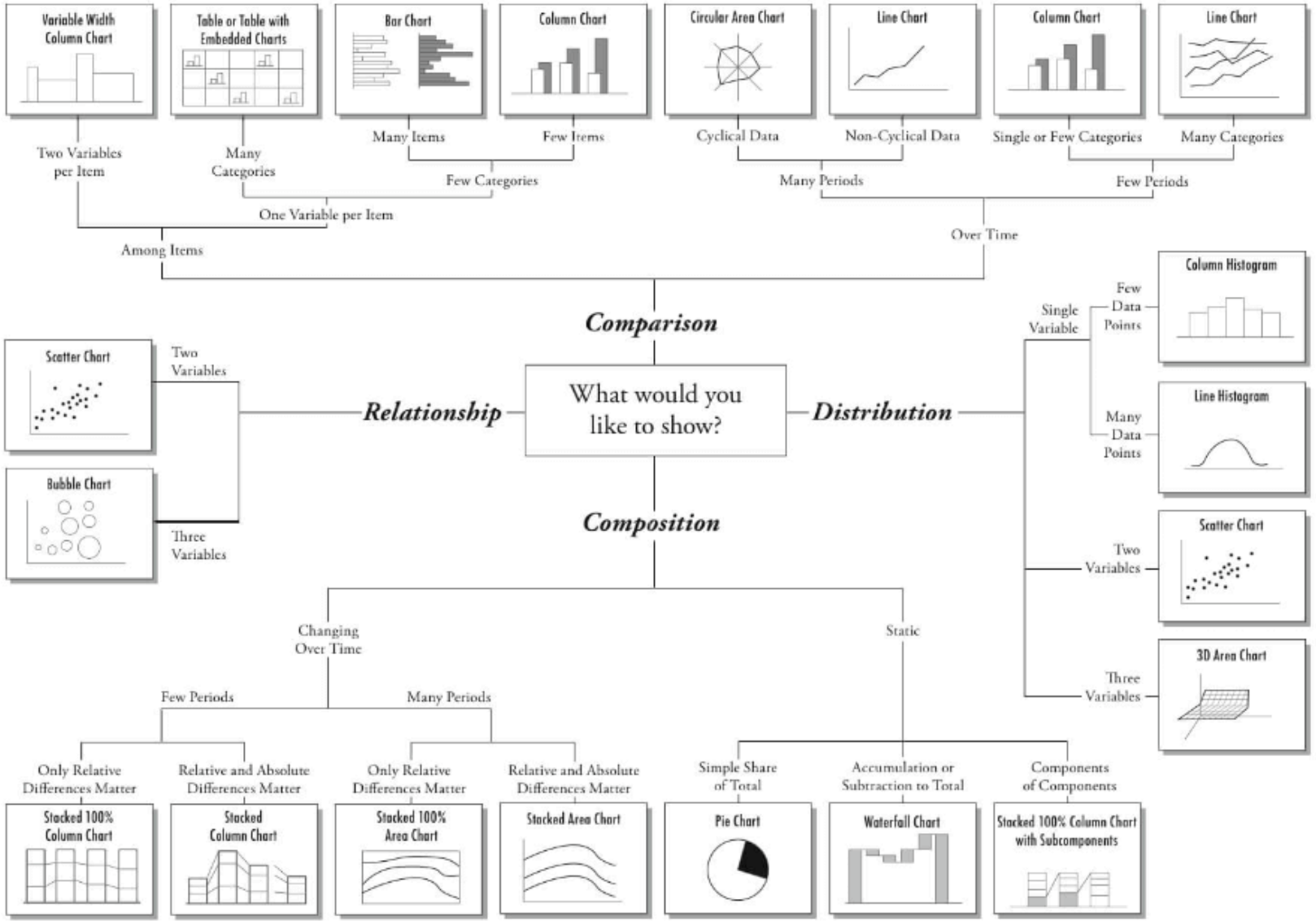
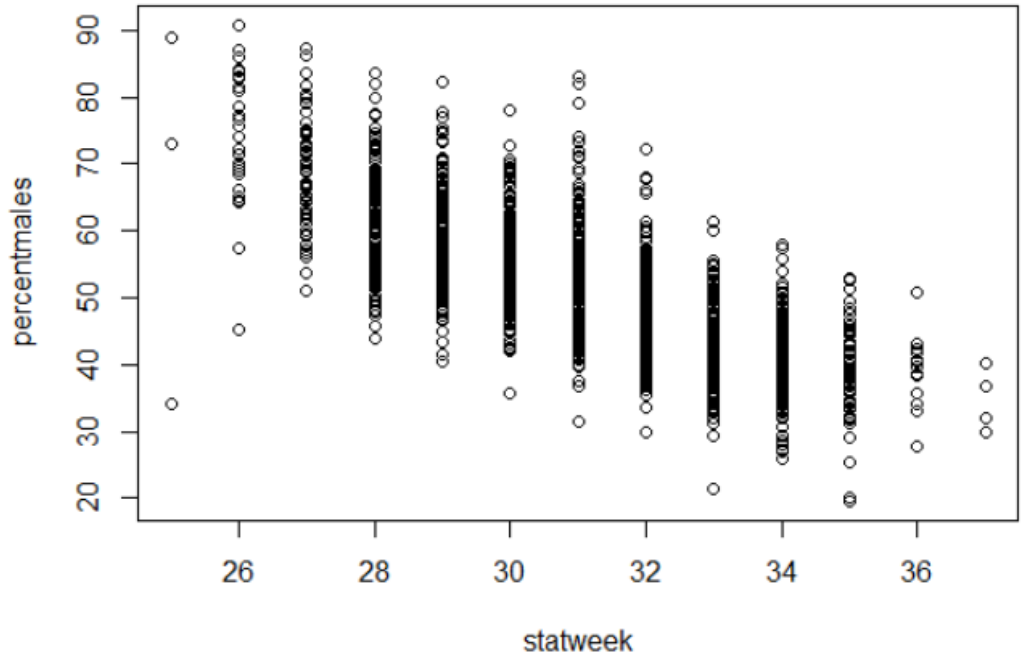


Chart Basics

- Use base function plot()
  - Syntax is `plot(yaxiscolumn ~ xaxiscolumn, data = dataname)`
  - OR alternately: `plot(x = dataname$xaxiscolumn, y = dataname$yaxiscolumn)`

- Using plot() is very quick and easy!



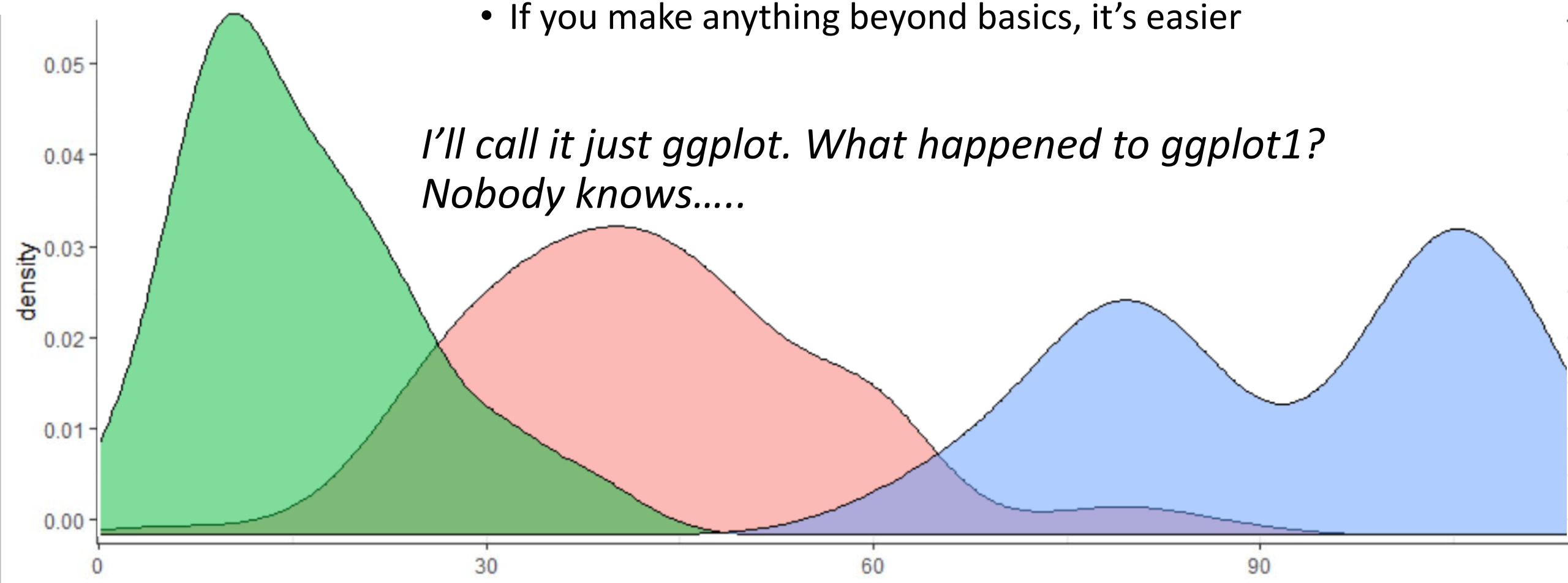

```
plot(percentmales ~ statweek,
   data = pinkratio)
```

ggplot2 – A Better Way

Why use ggplot2 over base plot?

- Faster to hit ground running
- If you make anything beyond basics, it's easier

I'll call it just ggplot. What happened to ggplot1? Nobody knows…..



ggplot2

- ggplot() function with data & aes()
  - aes() is the “aesthetics”. Set which cols are equal to x, y, color, fill, and group.
- Use a “+” to connect between lines
- Control aspects of the axes (and other parts) with “scale_”, e.g., scale_y_continuous, scale_x_discrete
- Add points & lines using “geom_”
  - e.g., geom_point(), geom_line(), etc
- Set visual elements using theme()

Establish the plot

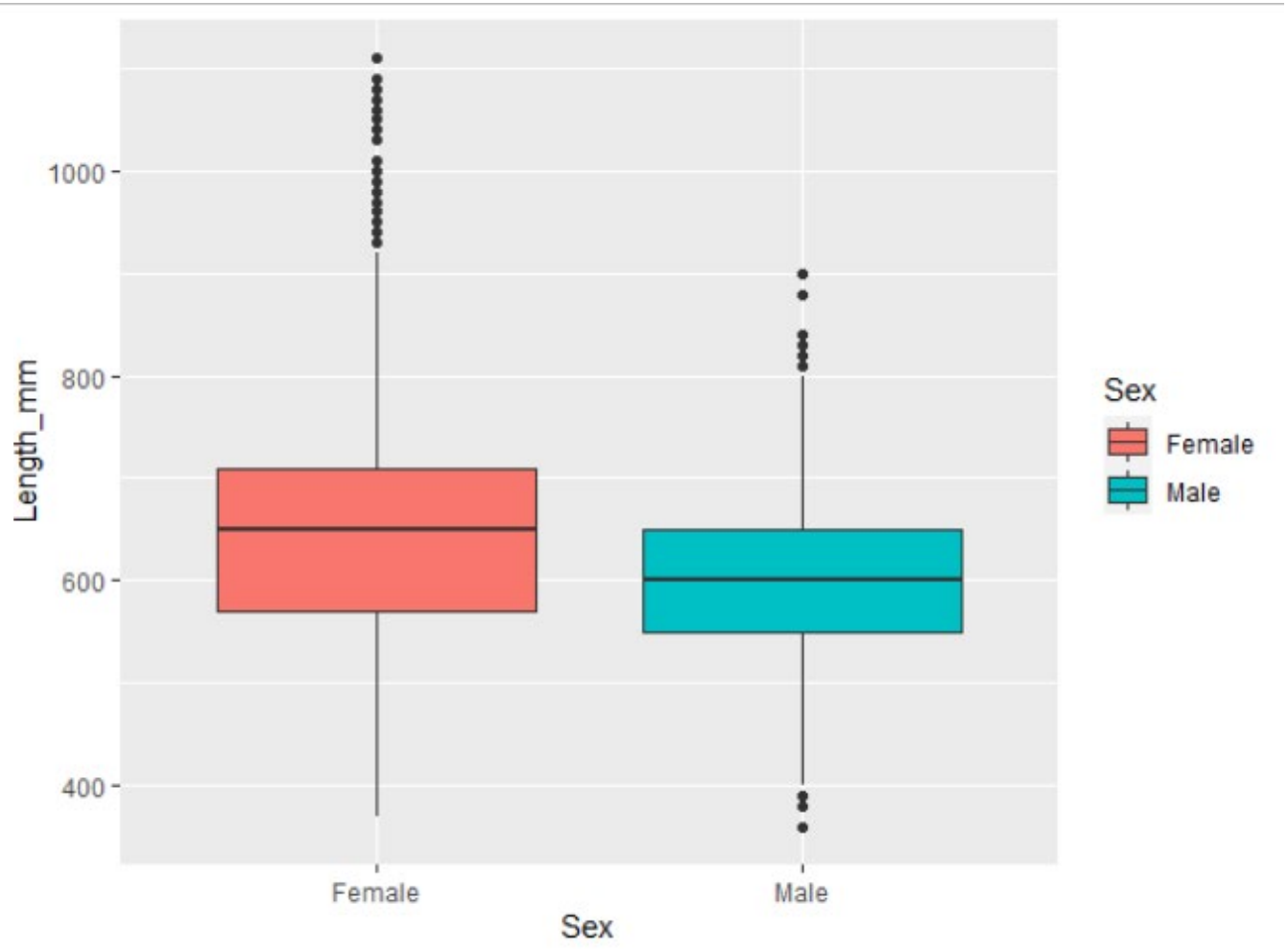
```
ggplot(data = mydataframe,
   aes(x = xcolumn, y = ycolumn,
   group = groupvar)) +
scale_x_continuous(breaks =
   c(1990, 2000, 2010, 2020)) +
geom_line()
```

Create lines!

Control x axis
(e.g., set axis breaks)

Anatomy of a ggplot

```
ggplot(data = sablefish,
       aes(x=Sex, y = Length_mm,
           fill = Sex)) +
  geom_boxplot()
```



Three required parts:

```
ggplot(data = xxx, aes()) +
geom_xxx()
```

ggplot parts: ggplot()

Two parts of ggplot(): data and aesthetics

- Establish data using data =

- Use aesthetics to "map" different variables to the x-axis, y-axis, fill, color, group (series), etc.
  - Done within the aes() function

Technically, specifying the data and the aes() could happen within the geom_xxx()

```
ggplot(data = mydataframe,
  aes(x = xcolumn, y = ycolumn,
  group = groupingvariable))
```

The aes() function can contain other parts:

shape = variable,

fill = variable,

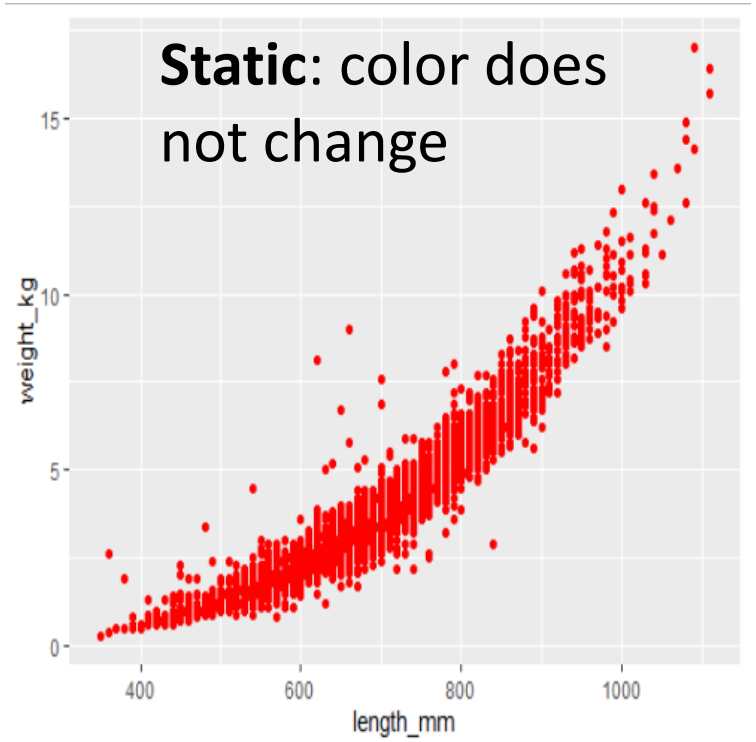
color = variable,

alpha = variable

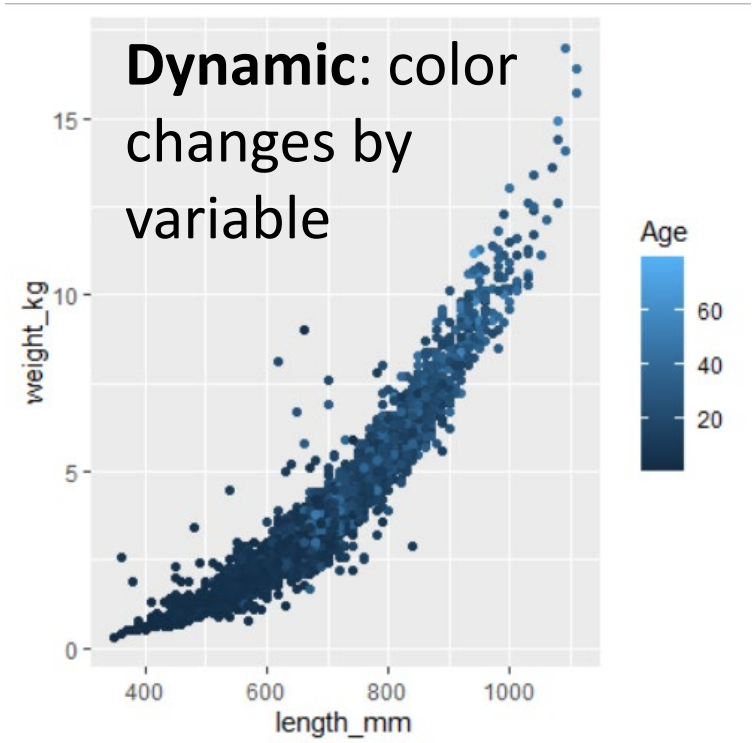
ggplot parts: aes()

It can be tricky to wrap your head around the aes function

- Make sure to keep the aes() inside the ggplot()
- Anything that is static goes **OUTSIDE** the aes(), in geom
- Anything dynamic (changes according to a variable) goes **INSIDE** the aes()



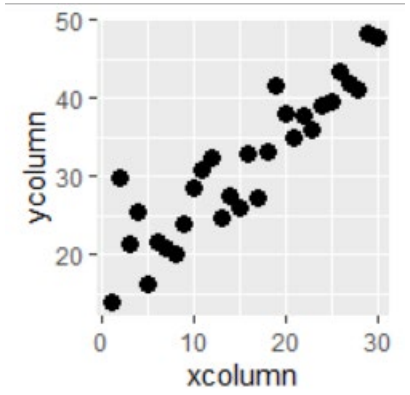

```
ggplot(sablefish, aes(x=length_mm,
   y = weight_kg)) +
  geom_point(color = "red")
```



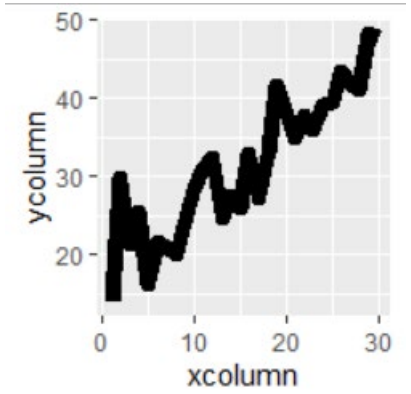

```
ggplot(sablefish, aes(x=length_mm,
   y = weight_kg, color = Age)) +
  geom_point()
```

ggplot parts: geoms

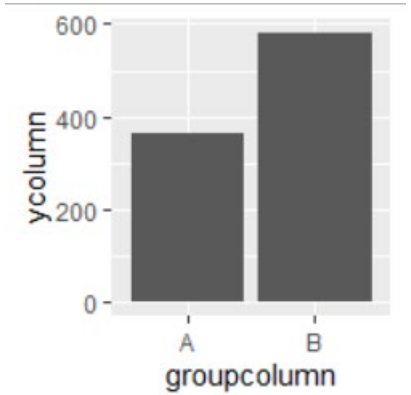
Tell ggplot what kind of plot you'd like:



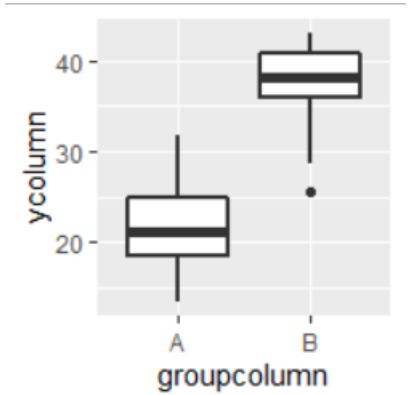
`geom_point()`



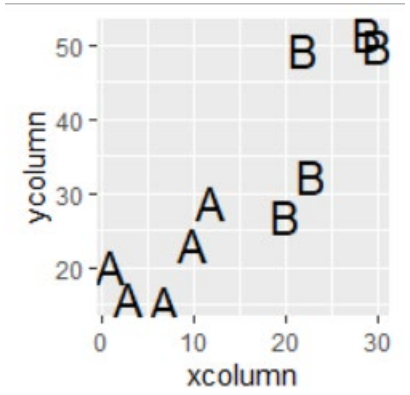
`geom_line()`



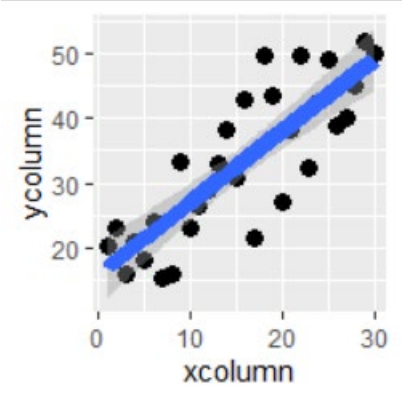
`geom_col()`



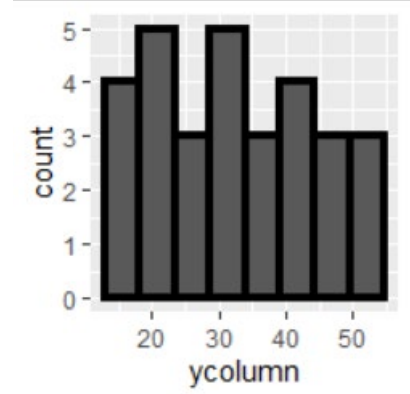
`geom_boxplot()`



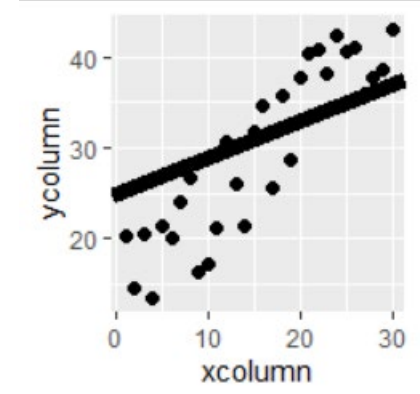
`geom_text()`



`geom_point() + geom_smooth()`



`geom_histogram()`



`geom_point() + geom_abline()`

Plot Type & Notes:

Basic scatterplot.

Basic line. Typical

Scatterplot but use text instead of symbols.

Add linear or LOESS

Basic bar (columns). "fill" is the bar color, color is

Add a boxplot, x is

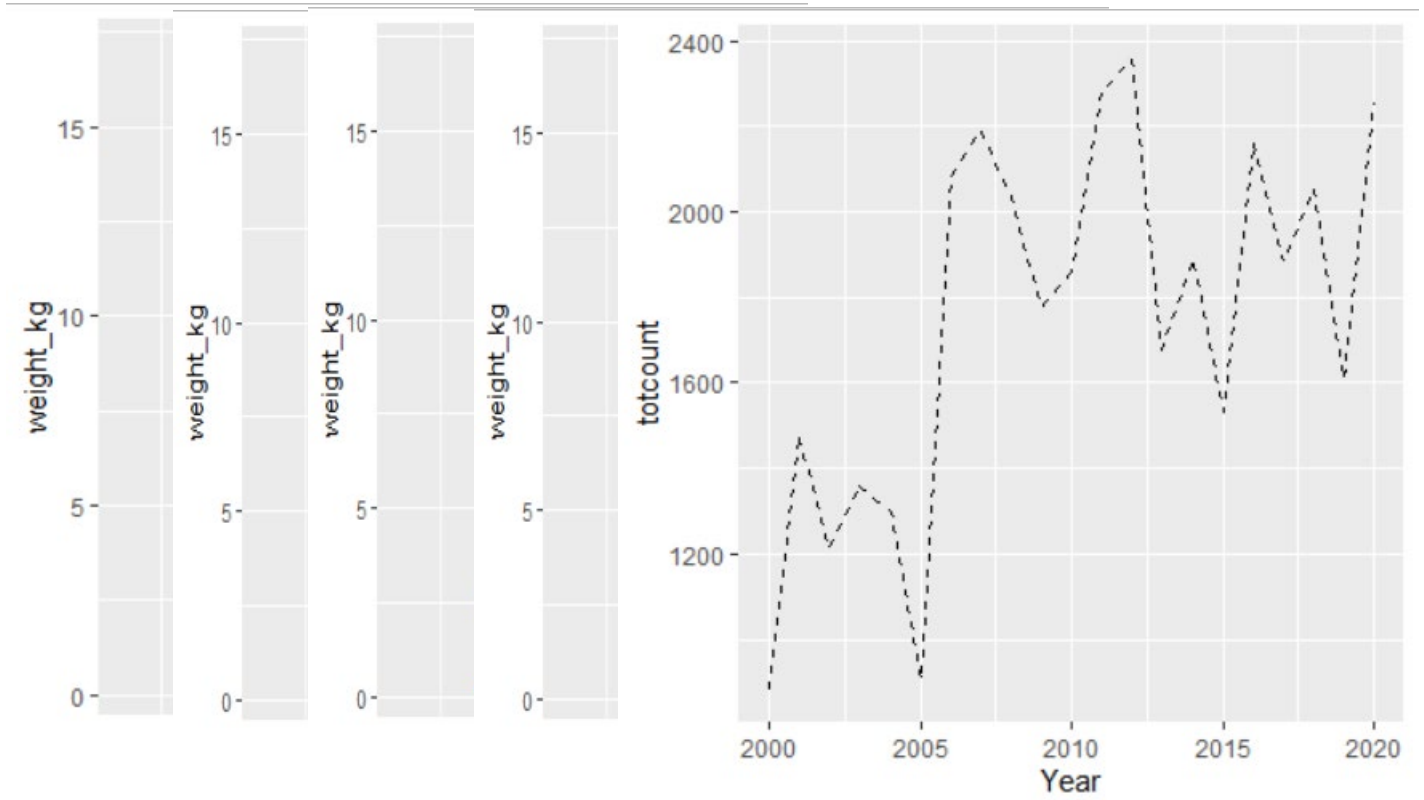
Histogram. No y-axis specified in aes()

Add straight line with arguments slope = 1, intercept = 0.

ggplot parts: geoms

To modify this plot, common arguments within a geom are:

- color
- shape (only for geom_point)
- alpha (transparency)
- linetype (only for geom_line)




```
ggplot ggp ggplot(s ggplot
  y =    y    weight    weig
geom_p gec geom_poi geom_p
```

```
ggplot(data = totalsablefish,
   aes(x=Year, y = totcount)) +
geom_line(linetype = 2)
```

ggplot parts

- So far, so good! These are the minimum parts required of a ggplot
  - ggplot(data = dataframename, aes(x=xcolumn, y=ycolumn)) + geom_xxx()

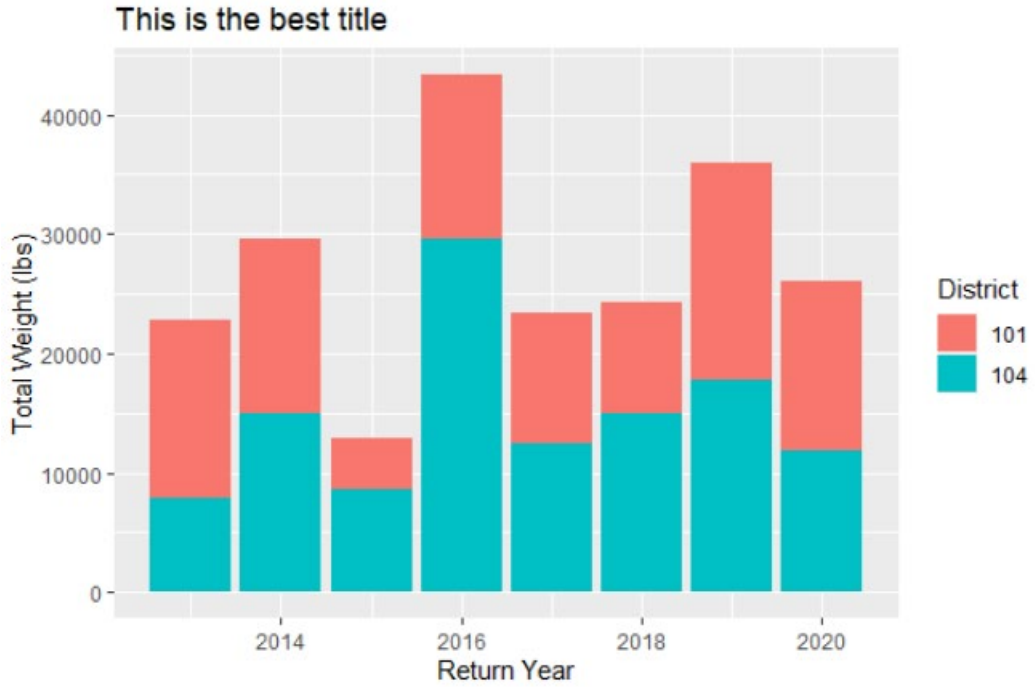
- What if we want to clean up the plot, specify colors, or change axis labels, etc.?

- Now we'll learn about adding more parts
  - Add lines together with a "+"

ggplot parts: scales

- The scale_ family are helper functions to control aspects such as specifying variable colors, variable fills, axis range / breaks, etc.
  - Far too many to review

- To change axis labels or title, use `labs()`



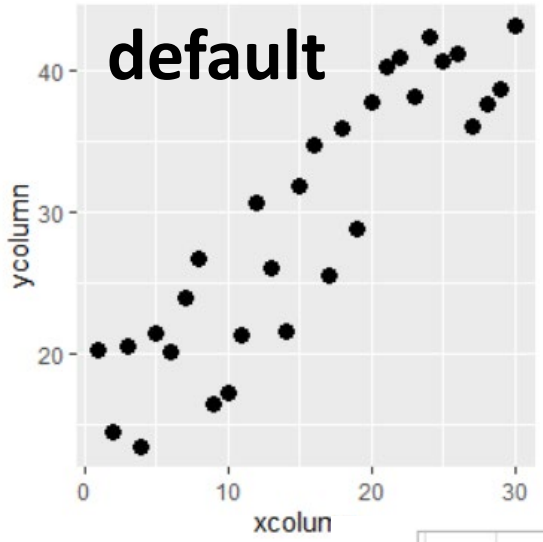

```
labs(x = "Return Year", y = "Total Weight
(lbs)", title = "This is the best title")
```

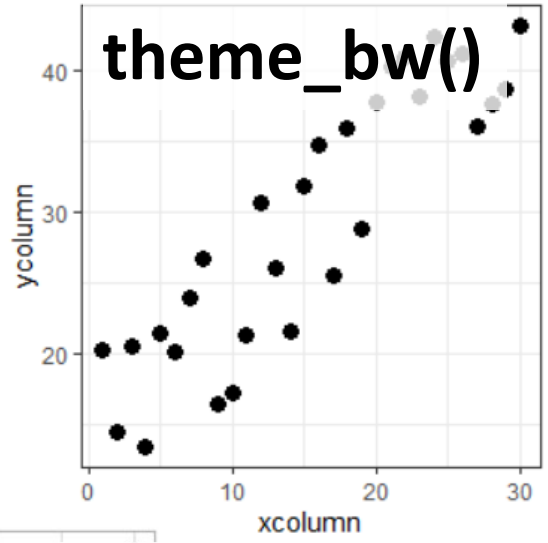
ggplot parts: theme

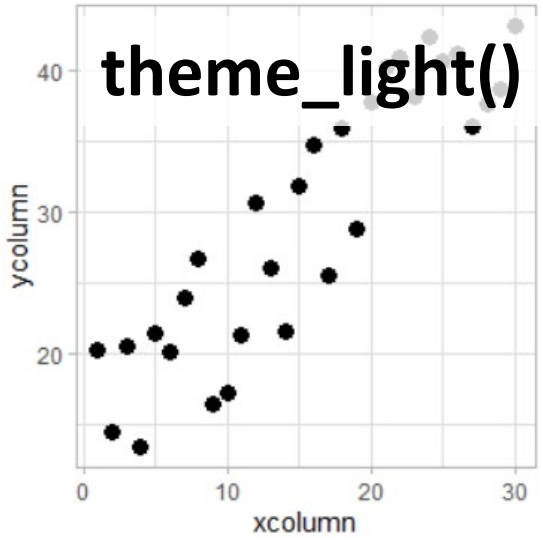
- To modify the appearance of text, axis lines, legend, etc., use theme()

- There are several built in themes
  - theme_bw()
  - theme_minimal()
  - theme_classic()
  - theme_light()

- I never remember how to do most theme modifications and google it







ggplot parts: theme

Some common theme() changes:

- Rotate x axis text

```
theme(axis.text.x = element_text(
angle = 45, vjust=1, hjust=1)
```

- Increase font size & change font

```
library(extrafont)
theme(base_size = 12, base_family
= “Arial”)
```

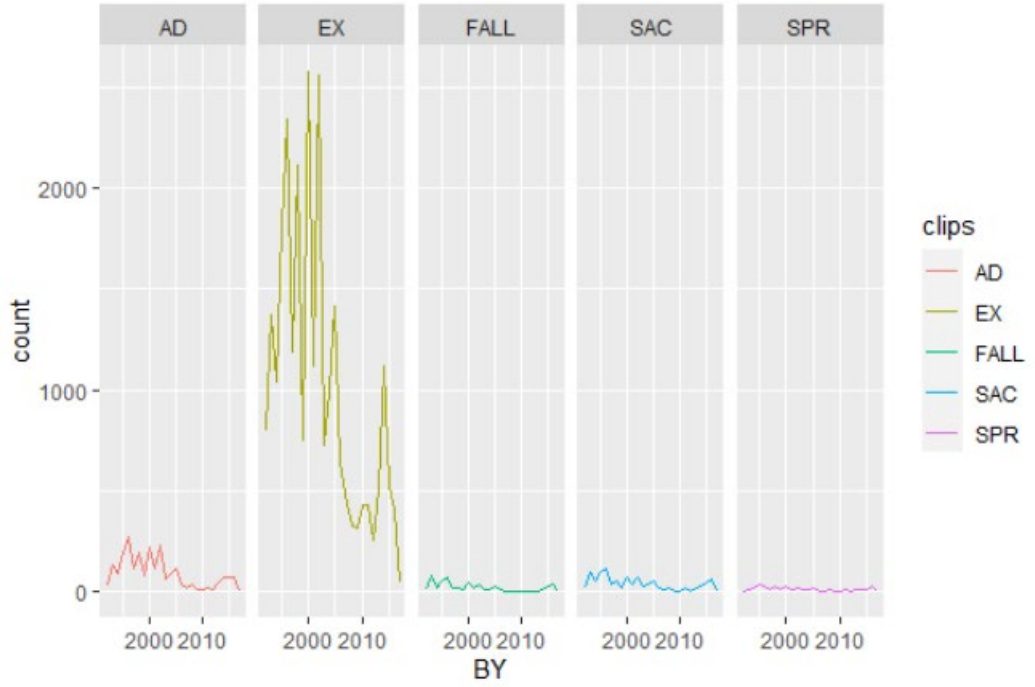
- Move legend position & remove legend title

```
theme(legend.position=c(0.5, 0.05),
legend.title = element_blank())
```

ggplot parts: Facets

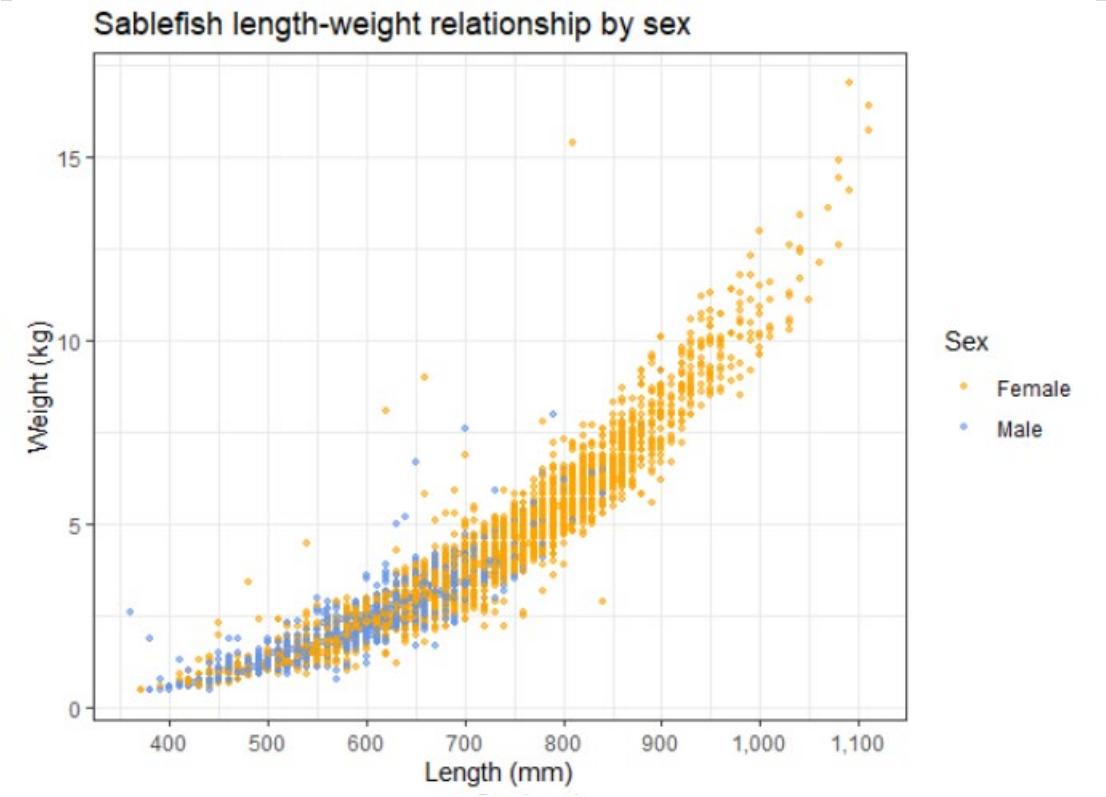
- One amazing feature of ggplot is the ability to create a different plot for specific subsets of a variable
- Use command `facet_wrap(~variable)` or `facet_grid(~variable)`

- You could even facet it based on two variables:
  - `facet_wrap(variable1 ~ variable2)`



Anatomy of a ggplot

```
> ggplot(data = sablefish,
         aes(x = Length_mm,
             y = Weight_kg,
             color = Sex)) +
  geom_point(size = 0.9, alpha = 0.6) +
  scale_color_manual(values =
       c("orange", "cornflowerblue")) +
  scale_x_continuous(breaks = c(400,
    500, 600, 700, 800, 900, 1000,
   1100, 1200), labels = scales::comma) +
 labs(x = "Length (mm)", y = "Weight(kg)",
 title = "Sablefish length-weight
     relationship by sex") +
 theme_bw()
```



Plotting Approach

- When starting out, it's tempting to paste a lot of code from a previous figure or something you found online.

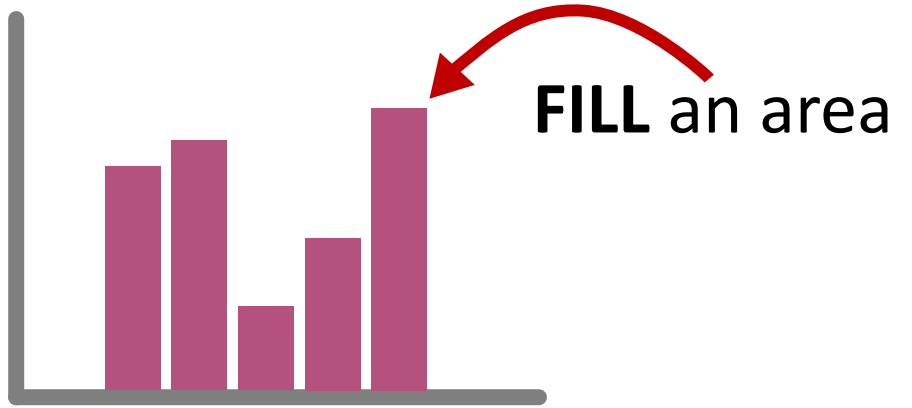
- If you can, try not to do this. A better approach is to build it step by step, line by line.
  - Add just the skeleton, e.g., `ggplot(aes()) + geom_line()`, and build out

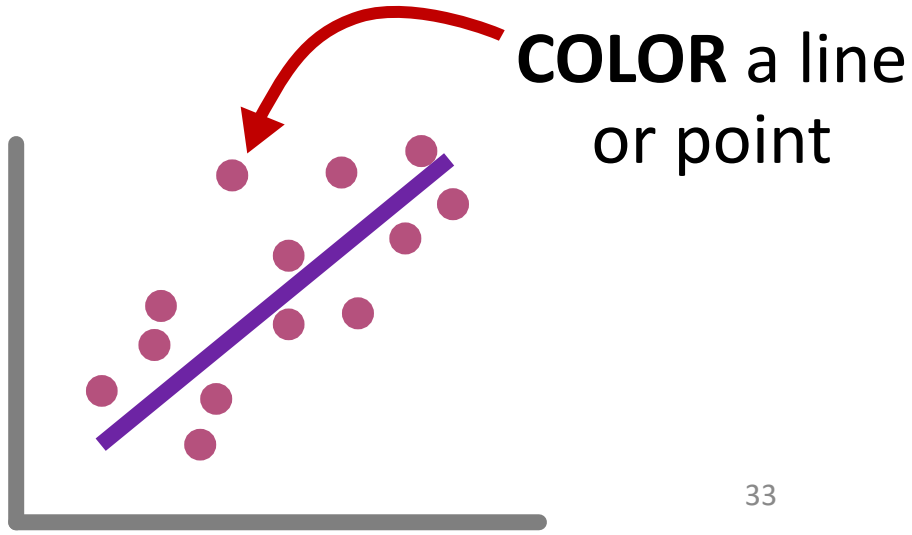
- This will immediately identify where errors exist

Fill vs Color

Remember to keep in mind the difference between "fill" and "color"

- "Fill" fills in an area (bar column)
- "Color" is the point or line
- If specifying manually, these take different commands depending on discrete vs continuous data





Fill vs color

	Color	Fill
Continuous	scale_color_gradientn(colors = c())	scale_fill_gradientn(colors = c())
Categorical	scale_color_manual(values = c("xxx"))	scale_fill_manual(values = c("xxxx"))

Example Code:

```
scale_color_gradientn(colors =
    terrain.colors(10))
```

```
scale_fill_gradientn(colors =
    terrain.colors(10))
```

```
scale_color_manual(values =
    c("orange", "cornflowerblue"))
```

```
scale_fill_manual(values =
    c("orange", "cornflowerblue"))
```

Saving!

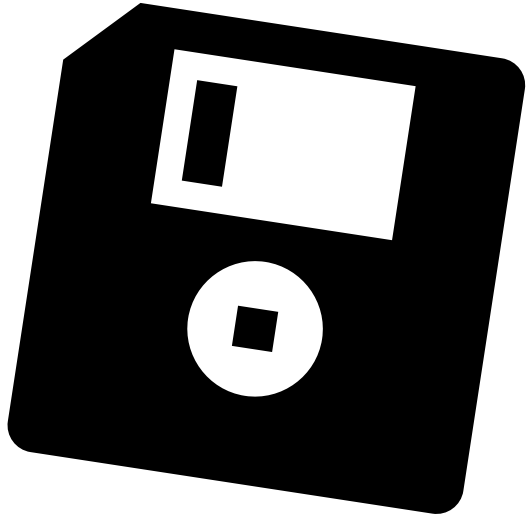
- If you want to save your ggplot, it's only one line of code:

```
ggsave(plot = plottosave,
  filename = "output/filesavename.png",
   dpi = 300,
  width = 6,
  height = 4,
  units = "in")
```

Object name

Dots per inch (≥300)

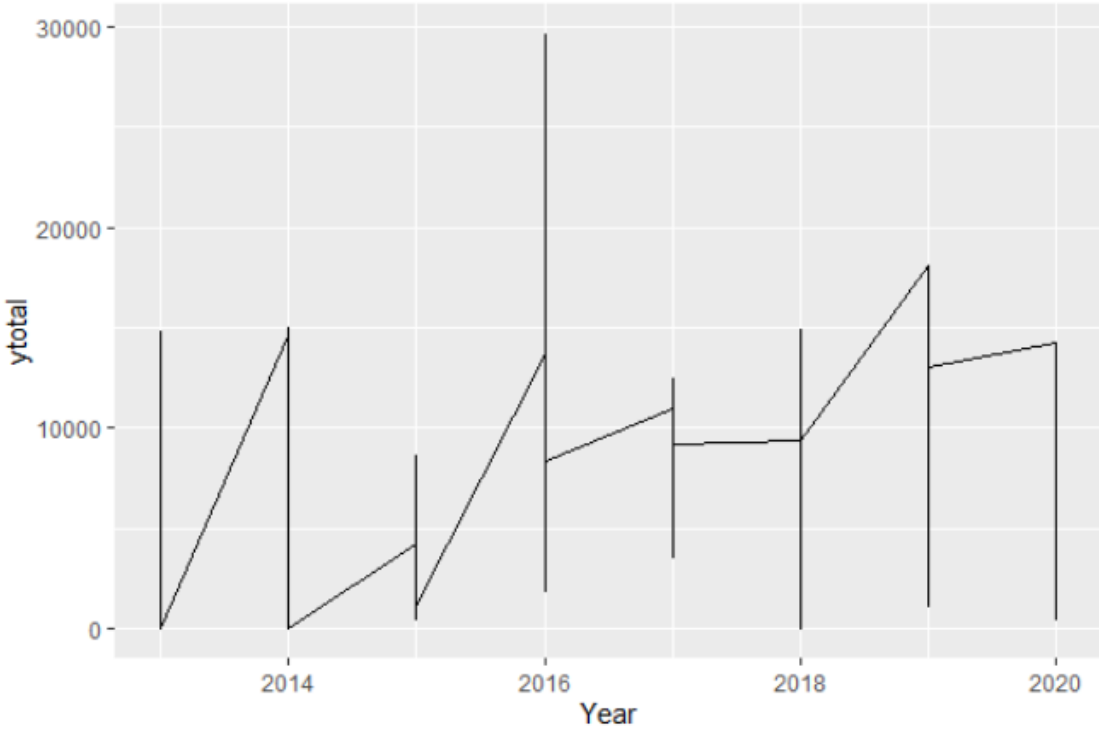
Set width, height and what units you're using



Common ggplot errors

The most common errors I see are:

- What goes inside vs outside aes()
  - Items inside aes() are variables (changeable), outside are static
- Jagged lines
  - You have multiple y points for each x point. Add a “group = ___” to plot a line for each group
- Color vs Fill
  - Easy mistake, remember the difference
- Overwriting a theme
  - ggplot will use the last theme settings you gave it




```
theme(line = element_blank()) +
theme_minimal()
```

Let's practice

- In RStudio, we'll run through the script. Please ask questions!
- Work through the really fun learnr tutorial!

- If you need another explanation later, Dr. Lendway's videos are great!
  - Intro to ggplot() – 8 min
  - Common ggplot mistakes – 3 min
  - ggplot demo – 32 min
  - These videos support material on her learning website

Lunch Break

See you all at 1:30 PM

6 – Basic Analysis

Because using 30-year-old Excel macros is so yesterday

Data Summary: `table()`

- Quickly summarizes dataframe
- table() makes a 1-way (1 variable) or 2-way summary table of all counts by variable
  - table(variable1, variable2)

```
> table(tannercrab$Year, tannercrab$Location)

     Barlow Cove Deadman Reach Excursion Inlet
2018         342           490             1155
2019         268           725              903
2020         307           672              668

     Gambier Bay Glacier Bay Holkham Bay
2018         260        1007        2049
2019         154         953        1938
2020           0           0           0

     Icy Strait Juneau Pybus Bay Seymour Canal
2018       1298   2477        163            569
2019       1046   2944        151            723
2020          0   3184          0              0

     St. James Bay Thomas Bay
2018           398        1071
2019           478        1749
2020           406           0
```

Data Summary: `summary()`

summary() is a quartile distribution of your continuous data

```
> summary(tannercrab$width)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
   10.0   110.0   128.0   126.9   144.0   197.0      76
>
>
> summary(tannercrab$chela)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
   1.00   19.00   25.00   25.28   32.00  117.00   23859
>
```

Summarize Data: `group_by()`

- `group_by() %>% summarize()`
  - Together, these are like a pivot table
- group_by() can be used to produce grouped calculations in mutate()
- Very fast and useful way to view data summaries

```
> dataframename %>%
    group_by(Year, statweek) %>%
    summarize(newcolname =
                mean(variable1))
```

Output:

For each year & statweek, what is the mean of variable1?

(New summary column is named "newcolname")

ANOVA

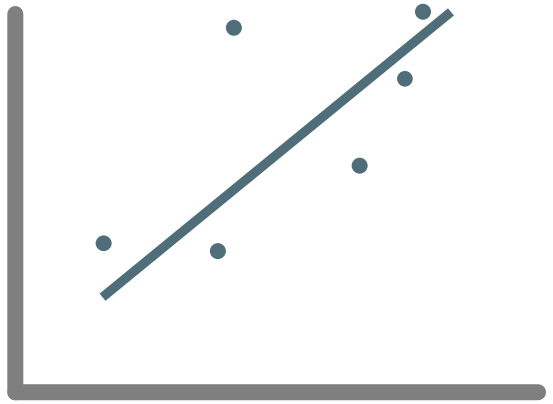
- ANOVAs test whether categorical variables have different continuous distributions. For example:
  - Is mean length significantly different between species?
  - Different mean weights between years?

- Use aov() and NOT anova()

- Remember that there are some items to be aware of for an ANOVA (type I, II, & III). More ANOVA info here
  - The Anova() function from package “car” allows specifying type!

Linear Modeling

- Modeling a linear trend in R is EASY!
  - lm(response ~ independentvar, data =df)
- Save this output or wrap in summary()
- Multiple variables are easy too:
  - lm(response ~ var1 + var2, data =df)



R notation:		**Formula notation:**
$lm(y \sim var1)$	=	$y = \beta_0 + \beta_1 var1$
$lm(y \sim var1 + var2)$		$y = \beta_0 + \beta_1 var1 + \beta_2 var2$

Reading lm() output

Formula

Residual diagnostics

```
> summary(lm(Chela ~ Width, data = tannercrab))

Call:
lm(formula = Chela ~ Width, data = tannercrab)

Residuals:
    Min      1Q  Median      3Q     Max
-35.581  -2.389   0.828   2.774  98.343

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -20.002011   0.416173  -48.06   <2e-16 ***
Width         0.351449   0.003189  110.20   <2e-16 ***

Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 4.481 on 4679 degrees of freedom
  (23867 observations deleted due to missingness)
Multiple R-squared:  0.7219,    Adjusted R-squared:  0.7218
F-statistic: 1.215e+04 on 1 and 4679 DF,  p-value: < 2.2e-16
```

Estimate = β_0 , β_1

Std. Error & t value

Pr(>|t|) = p-value

R^2

7 - Tidyverse

It’s like R, but different. And better. Definitely better.

Reference only – Just know this info is here and come back in the future!

What is the Tidyverse

- The Tidyverse is a collection of packages that extend the usefulness of R, use slightly different language syntax, and play well together
- Some of the most common Tidyverse packages are:

Loaded with library(tidyverse)

- **dplyr** – data manipulation (mutate, select, filter, summarise)
- **ggplot2** – produce beautiful charts
- **tidyr** – change data layout to one observation per row, and inverse
- **lubridate** – work with dates

And several others!

Why Use the Tidyverse?

- It's often simpler to read: filter(Year == 2010)
- Forces cleaner data which will save you time later
- Makes certain data types workable (e.g., dates)
- More powerful features
- Constantly updated (future of R?)

Downside

- Syntax is slightly different and you'll know two different ways to do same thing

dplyr

- Already learned about most of these functions
- Introduces the “pipe” operator %>%
- Add a new column using mutate(),
  choose specific rows based on column values using filter(),
  change whether a column is included using select(),
  change column names using rename()

***warning** you may find code online that uses the previous version of this package called “plyr”. If you find a function that uses plyr, I highly recommend not using it if you use “dplyr”. The two packages can (rarely) have major issues with each other.*

dplyr

- join functions add datasets together based on common columns
- Most often, I use left_join()
- `dfname1 %>% left_join(dfname2, by = c("Year" = "Yr"))`

Year	Var1	Catch
2010	A	10
2010	B	30
2011	A	5
2011	B	50
2012	A	15

dfname1

Yr	Rain
2010	0.5
2011	0.1
2012	1.2

dfname2

left_join()

Year	Var1	Catch	Rain
2010	A	10	0.5
2010	B	30	0.5
2011	A	5	0.1
2011	B	50	0.1
2012	A	15	1.2

ggplot2

- The most popular way to plot in R
- Uses a “grammar of graphics” which takes a bit to learn
- The basic structure is defining the source dataframe, the aesthetics (variables for x axis (& y axis), variables for color/shape, etc), and then a call to what type of plot you’re making (points, lines, histogram, etc)
- Strings together commands using “+” between the lines

lubridate

- Dates are probably the most difficult thing to deal with when starting out in R.
- lubridate provides a better interface; can easily convert between date formats, subtract time periods, calc day of year, etc.

- For “11/12/12”, what is the month? Are you sure?

- Excel is great at dealing with dates but it isn’t perfect either

lubridate examples

- Example 1: Convert “11/12/2020” to a date (12 Nov 2020)
- Answer 1:

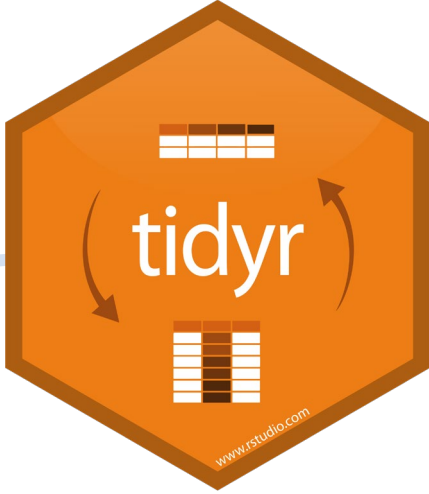
```
mutate(newdate = ymd(as.POSIXct(olddate, format =
"%m/%d/%Y", tz = "US/Alaska"))
```

This is the format of the existing data NOT the format you want it in

- Example 2: Convert “2020-12-31” to a date (31 Dec 2020)
- Answer 2:

```
mutate(newdate = ymd(as.POSIXct(olddate, format =
"%Y-%m-%d", tz = "US/Alaska"))
```

tidyr

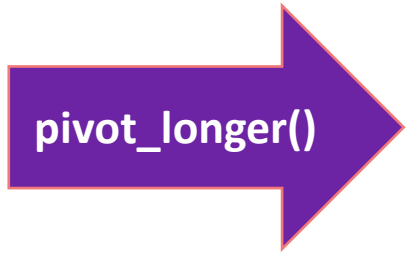


- tidyr is a package whose main functions help to turn “wide” data into “long” data and vice-versa
- pivot_wider() and pivot_longer()
  - These two easily turn data from wide to long (pivot_longer) or from long to wide (pivot_wider)
- When possible, keep data in long format to model / plot easier

tidyr example

```
dataframename %>%
pivot_longer(-leavethiscolumn,
             names_to = “newcolname”,
             values_to = “whatcellvaluesare”)
```

Year	StreamA	StreamB
2011	300	500
2012	150	450
2013	200	375
2014	120	450



Year	River	Count
2011	StreamA	300
2011	StreamB	500
2012	StreamA	150
2012	StreamB	450
2013	StreamA	200
2013	StreamB	375
2014	StreamA	120
2014	StreamB	450

```
dataframename %>%
pivot_longer(-Year, names_to = “River",
  values_to = "Count")
```

Concluding Thoughts

Some miscellaneous best practices and ways to have a smoother R experience

Workflow

When starting out, it's common to manipulate your data in Excel, save this file, then import/analyze in R

- *There is nothing wrong with this and if it works for you, great!*

It's MUCH better though to do all this in R

- Saves steps for cleanup that is performed each time

Best Practices

- R is tricky for tables but is excellent for figures
- Don’t use spaces in filenames or directories if you can
  - Name files descriptively
- Use easy to understand variable names; don’t overwrite
- Break up your data import / cleanup & analysis scripts
  - Use several scripts that “link” together rather than one long one
- Comment often and be VERY thorough, I promise that you’ll experience frustration at yourself for not being clearer
  - “Always comment your code as if the person who reviews your code will be a violent psychopath who knows where you live”

Surprise!

It’s going to be you…

Best Practices, cont.

- Update your R version and packages ~once/year
  - Unless you're about to publish and need absolutely nothing to change
- Follow people on GitHub / Twitter to read other people's code and learn better coding practices
- Read in data using .csv files (unless already in relational DB), never use file.choose(), never use the "Import Dataset" wizard in RStudio. Keep file paths relative and not static
- Use "Projects" in Rstudio to keep scripts organized

Not Covered

GitHub – Can sync your files to the internet which makes life MUCH simpler, especially if you are collaborating. Highly recommended

Rmarkdown – Create PDF, HTML, Word etc. files embedded with your R script outputs. Allows automatic report creations.

Shiny – This package allows for the creation of interactive apps

Spatial Analysis – Unique enough analyses to warrant its own discipline. If you can do it with ArcGIS, you can probably do it with R

That's all!

Great work!

Complete learnr tutorials and set up your project

Project

Use a provided dataset (or bring your own) to create a visualization

Project Goal

Create and save a script that does the following:

- Imports your dataset or one of the provided CSV files
- Loads needed libraries
- Filters to a specific subset of data (optional)
- Using ggplot, visualize some aspect of the data

Use the template, or create your own project / script

Remember! It's OK to Google how to do something. I do it for EVERY script

Project Steps

1. Choose a dataset
2. Open in Excel to look at data
3. Use provided template or create a new directory/project
4. Copy over all needed files
5. Create & save a blank script file
6. Import data into R
7. Perform needed data manipulation
8. View data, decide what to visualize
9. Plot in ggplot (and troubleshoot along way)

If you finish early, you can:

- Message & help others
- Make more plots
- Google other “touch ups”
- Do some simple analysis / summaries

If you feel like you’re not making progress that’s fine! Message me or a peer ☺

Example Projects

Some example projects that you could do are:

- Tanner crab chela width vs crab width
- Pink salmon harvest by district

- You are welcome (and expected) to copy code over that we've done together. Read error statements closely. Google your problems.
- If you want to pair up and work on the same dataset, it's great to talk it out and share code! However, write as much individually as you can: the more you write, the more the ideas are cemented

Work on Projects

Good luck! I'll try to help as many people as possible during this time

Start simple, work up to complicated.

Send me some of your plots by email / chat!

Project Steps

1. Choose a dataset
2. Open in Excel to look at data
3. Create a new directory/project & copy over all needed files (not required but helpful if you want to use this later)
4. Create & save a blank script file
5. Import data into R
6. Perform needed data manipulation
7. View data, decide what to visualize
8. Plot in ggplot (and troubleshoot along way)

Chart Presentation

Present plots

Plots

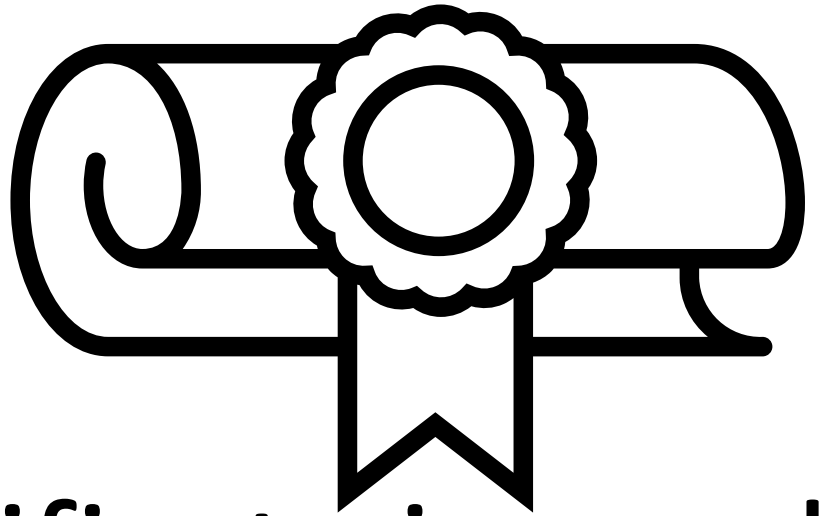
Parting Thoughts

- Like any language, becoming familiar with R requires a little practice over a long period of time. It's easy to get discouraged initially.

- Keep googling your problems: it's very unlikely that you're the first with this issue. I still google how to do something almost every other line of code.

- Programming is not the same as asking the right questions: **learn more statistics**! It is possible that you are attempting analysis that is best accomplished differently. Or worse, doing something you shouldn't.

Congrats!



This certificate is good for a free R help session, one-on-one!

Plus, you're *always* welcome to ask me for help; the worst I can say is I'm busy

Resources

Just Starting Out – Teach yourself!

★ • https://datacarpentry.org/R-ecology-lesson/00-before-we-start.html

- Learnr tutorial about NAs
- Learnr tutorial about filtering
- Learnr tutorial about summarizing

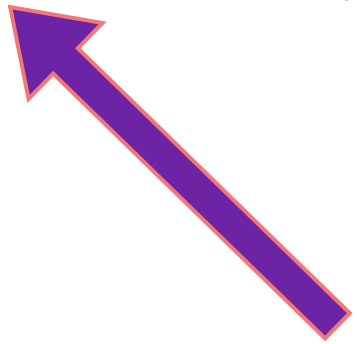
★ • https://r-bootcamp.netlify.app/

★ • https://rstudio.cloud/learn/primers

- https://ggplot-dplyr-intro.netlify.app/

Just Starting Out - Comprehensive

- In-depth Resource 1
- In-depth Resource 2



Similar layout to class!

Resources

Data Management

- [Data Organization in Spreadsheets]
- https://annebeaudreau.com/2018/02/04/data-management-tips/

Workflow

- https://whattheyforgot.org/project-oriented-workflow.html
- https://www.tidyverse.org/articles/2017/12/workflow-vs-script/

Resources

Style Guides

- http://r-pkgs.had.co.nz/style.html
- https://google.github.io/styleguide/Rguide.xml

Statistics

★ • http://www.shinystats.org/

★ • https://seeing-theory.brown.edu/regression-analysis/index.html

Cheatsheets

All cheatsheets

Getting started

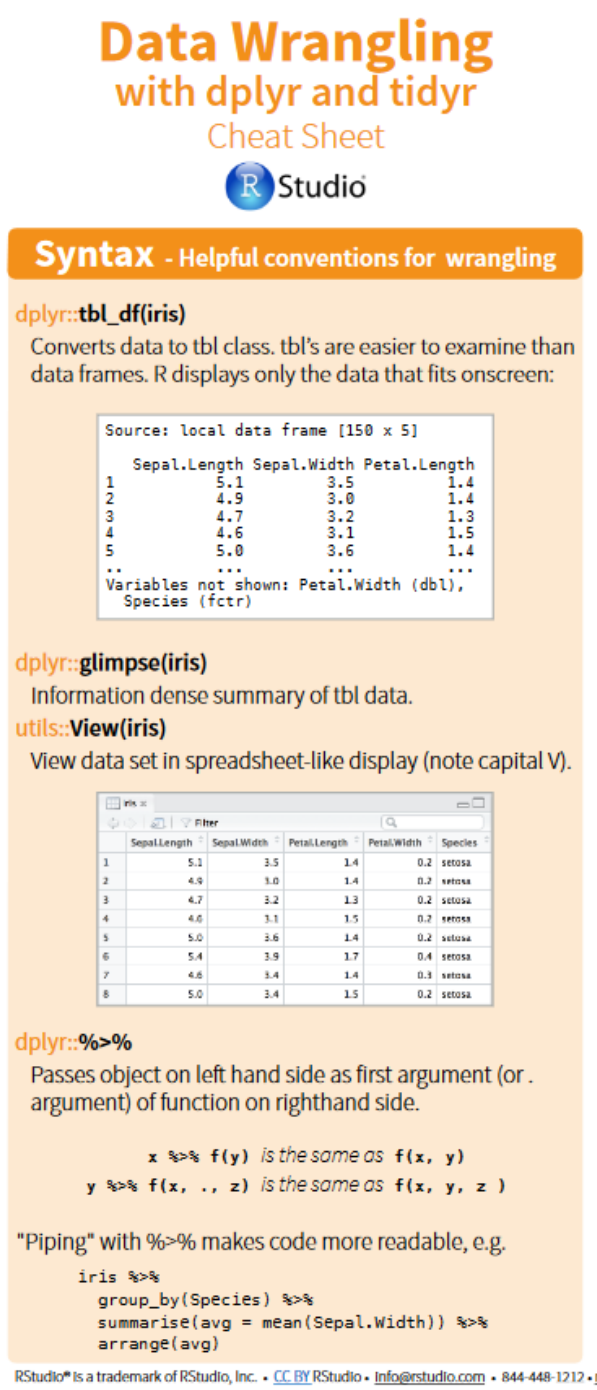
★ dplyr and tidyr

Data import

lubridate

★ ggplot2

RStudio



Data Wrangling
with dplyr and tidyr
Cheat Sheet

RStudio

Syntax - Helpful conventions for wrangling

dplyr::**tbl_df(iris)**

Converts data to tbl class. tbl's are easier to examine than data frames. R displays only the data that fits onscreen:

```
Source: local data frame [150 x 5]

   Sepal.Length Sepal.Width Petal.Length
1           5.1         3.5          1.4
2           4.9         3.0          1.4
3           4.7         3.2          1.3
4           4.6         3.1          1.5
5           5.0         3.6          1.4
..          ...         ...          ...
Variables not shown: Petal.Width (dbl),
  Species (fctr)
```

dplyr::**glimpse(iris)**

Information dense summary of tbl data.

utils::**View(iris)**

View data set in spreadsheet-like display (note capital V).

dplyr::**%>%**

Passes object on left hand side as first argument (or . argument) of function on righthand side.

`x %>% f(y)` *is the same as* `f(x, y)`

`y %>% f(x, ., z)` *is the same as* `f(x, y, z )`

"Piping" with %>% makes code more readable, e.g.

```
iris %>%
  group_by(Species) %>%
  summarise(avg = mean(Sepal.Width)) %>%
  arrange(avg)
```

RStudio® is a trademark of RStudio, Inc. • CC BY RStudio • Info@rstudio.com • 844-448-1212 • rstudio.com

Tidy Data - A foundation for wrangling

In a tidy data set:

Each **variable** is saved in its own **column**

&

Each **observation** is saved in its own row

Reshaping Data – Change the layout of a data set

tidyr::**gather(cases, "year", "n", 2:4)**

Gather columns into rows.

tidyr::**separate(storms, date, c("y", "m", "d"))**

Separate one column into several.

Subset Observations (Rows)

dplyr::**filter(iris, Sepal.Length > 7)**

Extract rows that meet logical criteria.

dplyr::**distinct(iris)**

Remove duplicate rows.

dplyr::**sample_frac(iris, 0.5, replace = TRUE)**

Randomly select fraction of rows.

dplyr::**sample_n(iris, 10, replace = TRUE)**

Randomly select n rows.

dplyr::**slice(iris, 10:15)**

Select rows by position.

dplyr::**top_n(storms, 2, date)**

Select and order top n entries (by group if grouped data).

Logic in R - ?Comparison, ?base::Logic

<	Less than	!=	Not equal to
>	Greater than	%in%	Group membership
==	Equal to	is.na	Is NA
<=	Less than or equal to	!is.na	Is not NA
>=	Greater than or equal to	&, |, !, xor, any, all	Boolean operators

devtools::install_github("rstudio/EDAWR") for data sets

Special Thanks

People who have taught & helped me or whose code I have shamelessly copied and learned from:

Franz Mueter (UAF)

- All UAF classmates

Ben Williams (NOAA/ADF&G)

Jordan Watson (NOAA)

Curry Cunningham (UAF/NOAA)

Jenny Bryan (UBC)

Hadley Wickham (RStudio)

Greg Wilson (RStudio)

Allison Horst (RStudio)

Rhea Ehresmann (ADF&G)