STAT 215A Fall 2019
Week 2

Tiffany Tang

9/6/19

Announcements

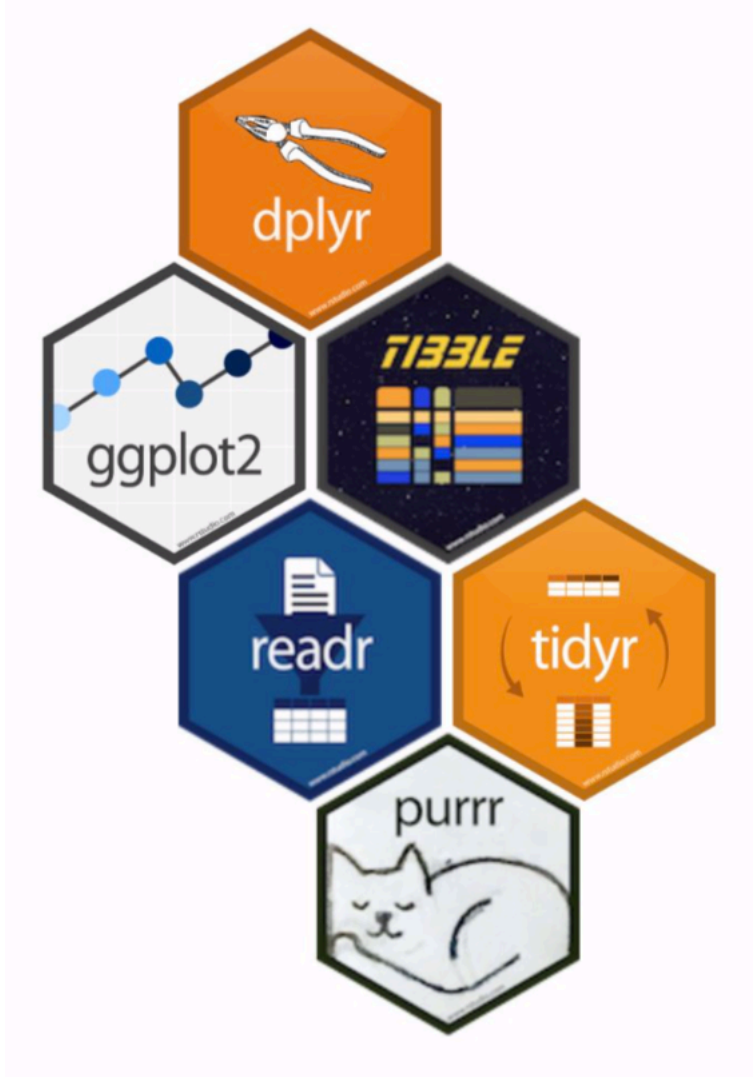
- Lab 0 example solutions available on my STAT-215A-Fall-2019 GitHub
- Make sure you have access to Piazza and Bcourses
  - If you don't, send me an email: tiffany.tang@berkeley.edu
- Lab 1 will be released on Bcourses and my STAT-215A-Fall-2019 GitHub repo at the end of the lab section.
  - **Due Thursday, September 19 at 11:59pm**

GitHub repositories I have access to…

Aybolek Amanmyradova
Brandon Mannion
Brooke Staveland
Cameron Adams
Chao Zhang
Chris Rowe
Corrine Elliott
Dandan Ru
Dimitrios Vlachogiannis
Ella Hiesmayr
Emily Barnes
Jiaxi Liu
Jue Wang
Kanaad Deodhar
Katherine Kempfert
Liang Zhang
Mehdi Ouaki
Mike Janson
Namita Trikannad
Partow Imani
Phil Ryjanovsky
Qianhua Luo
Reid Whitaker
Robbie Netzorg
Sam Stein
Sergey Kaniskin
Shubei Wang
Sichen Li
Spencer Wilson
Teng Li
Yang Li
Yanlei Fengtest
Yanting Pan
Yihuan Song
Yiyi He
Yuxin Zhang
Zhenni Ye
Ziyang Zhou

Plan for Today

- Tidyverse
- Establishing a workflow
- Introduce Lab 1


dplyr
ggplot2
tibble
readr
tidyr
purrr


tidyverse
www.rstudio.com

R Tidyverse

Tidy Data

1. Each variable forms a column
2. Each observation forms a row
3. Each type of observational unit forms a table

For more on tidy data: http://vita.had.co.nz/papers/tidy-data.pdf

For an introduction to R and tidyverse: https://r4ds.had.co.nz/

For more advanced topics in R: https://adv-r.hadley.nz/

If you ever run into questions or need help with R:

Google and StackOverflow are your friend

Or type in the R console: `? function_name (e.g., ? read.table)`

Tidyverse is great, but not perfect:

https://github.com/matloff/TidyverseSkeptic


dplyr
ggplot2
TIBBLE
readr
tidyr
purrr

dplyr: Piping Operator (%>%)

- Pronounce “%>%” as “then”
- Piping is a way of chaining together functions so that you don’t have to keep redefining variables.
- Piping always puts the object being piped into the first argument of the function

Piping

```
> iris %>%
+   group_by(Species) %>%
+   summarise(n = n())
# A tibble: 3 x 2
  Species        n
  <fct>      <int>
1 setosa        50
2 versicolor    50
3 virginica     50
```

No Piping

```
> iris_by_species <- group_by(iris, Species)
> summarise(iris_by_species, n = n())
# A tibble: 3 x 2
  Species        n
  <fct>      <int>
1 setosa        50
2 versicolor    50
3 virginica     50

> summarise(group_by(iris, Species), n = n())
```

dplyr: “Grammar of Data Manipulation”

- Contains functions for editing variables in a data frame or tibble
  - **filter():** find rows where the specified condition is true
  - **select():** keep or remove certain variables/features
  - **rename():** for renaming columns
  - **mutate():** modifying/creating new columns
  - **group_by():** change the scope of each function from operating of the entire dataset to operating on groups
  - **summarise():** reduces multiple values down to a single summary
  - **arrange():** order rows in data frame by specific column(s)
  - Also see filter_*(), select_*(), rename_*(), mutate_*(), where * = if, at, all
- The first argument of each function is a data frame (or tibble)
- The result is always a new data frame (or tibble)

dplyr::filter()

- Finds rows where the specified condition is true
- Ex. Extract observations corresponding to the species “versicolor”

Piping + Filter

```
> iris %>%
+   filter(Species == "versicolor") %>%
+   head()
  Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
1          7.0         3.2          4.7         1.4 versicolor
2          6.4         3.2          4.5         1.5 versicolor
3          6.9         3.1          4.9         1.5 versicolor
4          5.5         2.3          4.0         1.3 versicolor
5          6.5         2.8          4.6         1.5 versicolor
6          5.7         2.8          4.5         1.3 versicolor
```

Base R

```
> head(iris[iris$Species == "versicolor", ])
   Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
51          7.0         3.2          4.7         1.4 versicolor
52          6.4         3.2          4.5         1.5 versicolor
53          6.9         3.1          4.9         1.5 versicolor
54          5.5         2.3          4.0         1.3 versicolor
55          6.5         2.8          4.6         1.5 versicolor
56          5.7         2.8          4.5         1.3 versicolor
```

- See also filter_all(), filter_if(), and filter_at()

dplyr::select() and dplyr::rename()

- select(): keep certain variables, or remove certain variables
- rename(): rename columns
- Ex. Get Sepal.Length and Sepal.Width columns and rename to Length and Width, respectively

Piping + select + rename

```
> iris %>%
+   select(Sepal.Length, Sepal.Width) %>%
+   rename(Length = Sepal.Length,
+          Width = Sepal.Width) %>%
+   head()
  Length Width
1    5.1   3.5
2    4.9   3.0
3    4.7   3.2
4    4.6   3.1
5    5.0   3.6
6    5.4   3.9
```

Base R

```
> iris_sepal <- iris[, c("Sepal.Length", "Sepal.Width")]
> colnames(iris_sepal) <- c("Length", "Width")
> head(iris_sepal)
  Length Width
1    5.1   3.5
2    4.9   3.0
3    4.7   3.2
4    4.6   3.1
5    5.0   3.6
6    5.4   3.9
```

- See also select_all(), select_if(), select_at(), rename_all(), rename_if(), rename_at()

dplyr::select_if()

- select_if(): keep or remove certain variables if a condition holds
- Ex. Get numeric columns only

```
> iris %>%
+   select_if(is.numeric) %>%
+   head()
  Sepal.Length Sepal.Width Petal.Length Petal.Width
1          5.1         3.5          1.4         0.2
2          4.9         3.0          1.4         0.2
3          4.7         3.2          1.3         0.2
4          4.6         3.1          1.5         0.2
5          5.0         3.6          1.4         0.2
6          5.4         3.9          1.7         0.4
```

dplyr::mutate()

- Create new variables consisting of functions of existing variables.
- Ex. Create a new variable which is the sum of the Sepal Length and Sepal Width

Piping + mutate

```
> iris %>%
+   mutate(Sepal.Sum = Sepal.Width + Sepal.Length) %>%
+   head()
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species Sepal.Sum
1          5.1         3.5          1.4         0.2  setosa       8.6
2          4.9         3.0          1.4         0.2  setosa       7.9
3          4.7         3.2          1.3         0.2  setosa       7.9
4          4.6         3.1          1.5         0.2  setosa       7.7
5          5.0         3.6          1.4         0.2  setosa       8.6
6          5.4         3.9          1.7         0.4  setosa       9.3
```

Base R

```
> iris$Sepal.Sum <- iris$Sepal.Width + iris$Sepal.Length
> head(iris)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species Sepal.Sum
1          5.1         3.5          1.4         0.2  setosa       8.6
2          4.9         3.0          1.4         0.2  setosa       7.9
3          4.7         3.2          1.3         0.2  setosa       7.9
4          4.6         3.1          1.5         0.2  setosa       7.7
5          5.0         3.6          1.4         0.2  setosa       8.6
6          5.4         3.9          1.7         0.4  setosa       9.3
```

- See also mutate_all(), mutate_if(), and mutate_at()

dplyr::mutate_at()

- Mutate at existing variables via some function
- Ex. Multiply each of Sepal.Length and Sepal.Width by 2

```
> iris %>%
+   mutate_at(vars(contains("Sepal")), list(~ 2 * .)) %>%
+   head()
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species Sepal.Sum
1         10.2         7.0          1.4         0.2  setosa      17.2
2          9.8         6.0          1.4         0.2  setosa      15.8
3          9.4         6.4          1.3         0.2  setosa      15.8
4          9.2         6.2          1.5         0.2  setosa      15.4
5         10.0         7.2          1.4         0.2  setosa      17.2
6         10.8         7.8          1.7         0.4  setosa      18.6
```

dplyr::group_by() and dplyr::summarise()

- group_by(): changes the scope of each function from operating on the entire dataset to operating on it group-by-group.
- summarise(): reduces multiple values down to a single summary.
- Ex. Compute the mean and median Sepal.Length for each Species

```
> iris %>%
+   group_by(Species) %>%
+   summarise(Sepal.Length.mean = mean(Sepal.Length),
+             Sepal.Length.median = median(Sepal.Length))
# A tibble: 3 x 3
  Species    Sepal.Length.mean Sepal.Length.median
  <fct>                  <dbl>                <dbl>
1 setosa                  5.01                  5
2 versicolor              5.94                  5.9
3 virginica               6.59                  6.5
```

dplyr::arrange()

- Order rows by an expression involving its variables.
- Ex. Order rows first by decreasing Petal Length and then by increasing Sepal Width (if there are ties among the Petal Length)

```
> iris %>%
+   arrange(desc(Petal.Length), Sepal.Width) %>%
+   head()
  Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
1          7.7         2.6          6.9         2.3 virginica
2          7.7         2.8          6.7         2.0 virginica
3          7.7         3.8          6.7         2.2 virginica
4          7.6         3.0          6.6         2.1 virginica
5          7.9         3.8          6.4         2.0 virginica
6          7.3         2.9          6.3         1.8 virginica
```


dplyr
ggplot2
TIBBLE
readr
tidyr
purrr

tidyr: “Grammar of Tidy Data”

- Used to be called “reshape2”
- Contains functions for changing the shape of the data from long-form to wide-form.
- Main functions
  - **spread():** to go from long to wide format
  - **gather():** to go from wide to long format

tidyr::spread() vs tidyr::gather()

spread: convert long format to wide format

```
> iris_wide <- iris_long %>%
+   spread(key = "Variable", value = "Value")
> head(iris_wide)
   id    Species Petal.Length Petal.Width Sepal.Length Sepal.Sum Sepal.Width
1   1     setosa          1.4         0.2          5.1       8.6         3.5
2  10     setosa          1.5         0.1          4.9       8.0         3.1
3 100 versicolor          4.1         1.3          5.7       8.5         2.8
4 101  virginica          6.0         2.5          6.3       9.6         3.3
5 102  virginica          5.1         1.9          5.8       8.5         2.7
6 103  virginica          5.9         2.1          7.1      10.1         3.0
```

gather: convert wide format to long format

```
> iris_long <- iris %>%
+   rownames_to_column("id") %>%
+   gather(key = "Variable", value = "Value", -Species, -id)
> head(iris_long)
  id Species     Variable Value
1  1  setosa Sepal.Length   5.1
2  2  setosa Sepal.Length   4.9
3  3  setosa Sepal.Length   4.7
4  4  setosa Sepal.Length   4.6
5  5  setosa Sepal.Length   5.0
6  6  setosa Sepal.Length   5.4
```


dplyr
ggplot2
tibble
readr
tidyr
purrr

ggplot2: “Grammar of Graphics”

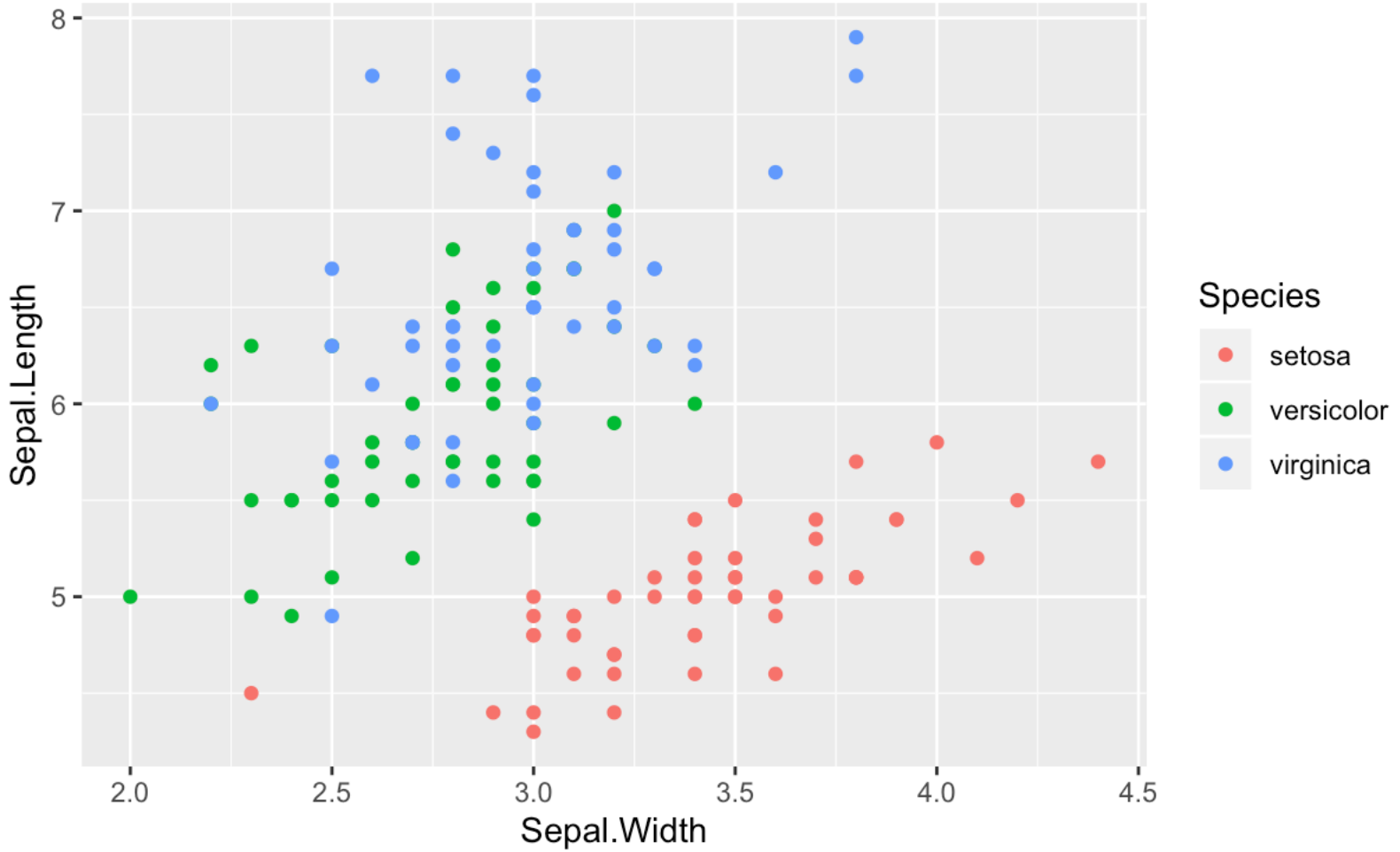
- Always begin with **ggplot(data.frame)**
- Then add layers to the base plot by using + (similar to piping %>%)
  - Examples of layers you may want to add:
    - **geom_point()**
    - **geom_histogram()**
    - **geom_text()**
    - **theme()**
    - **scale_x_continuous()**
    - **labs()**

```
> ggplot(iris) +
+   geom_point(aes(x = Sepal.Width, y = Sepal.Length, color = Species))
```

- You can do many many things with ggplot, and I can’t even begin to list the different layers, color themes, formatting options that you can specify in ggplot. **Google will be your best friend here!**

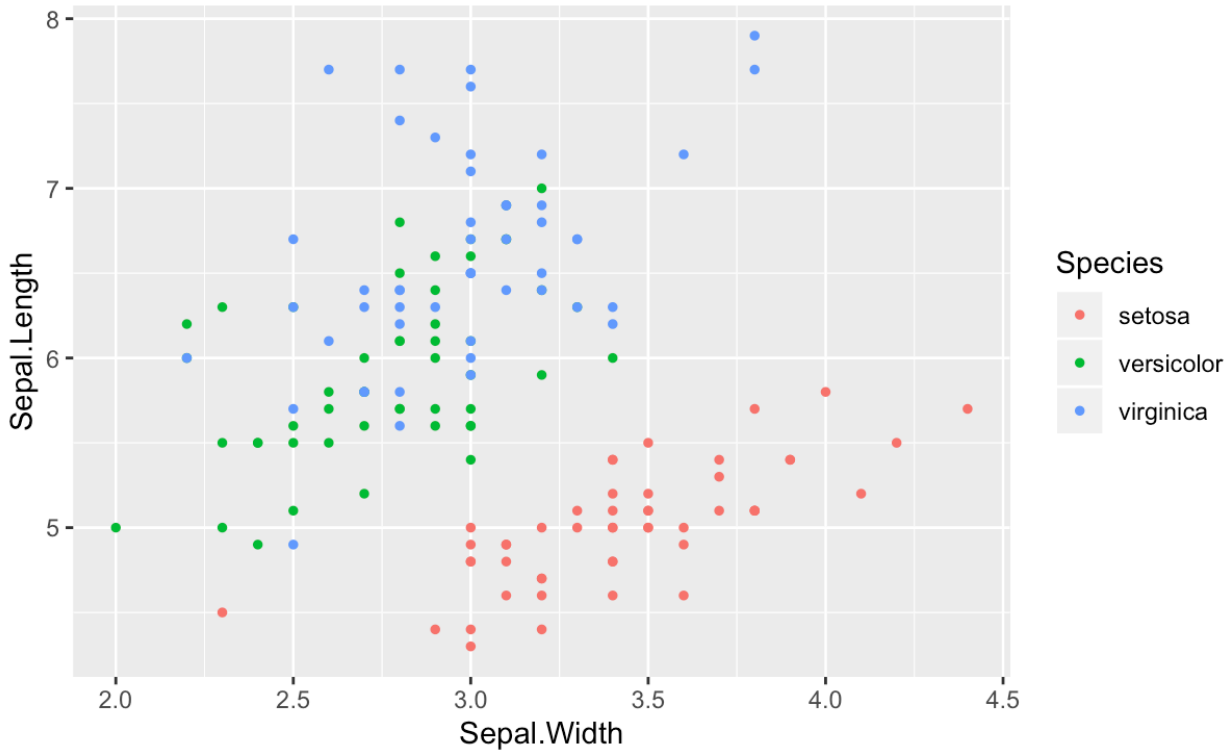
ggplot2: “Grammar of Graphics”

```
> ggplot(iris) +
+   geom_point(aes(x = Sepal.Width, y = Sepal.Length, color = Species))
```

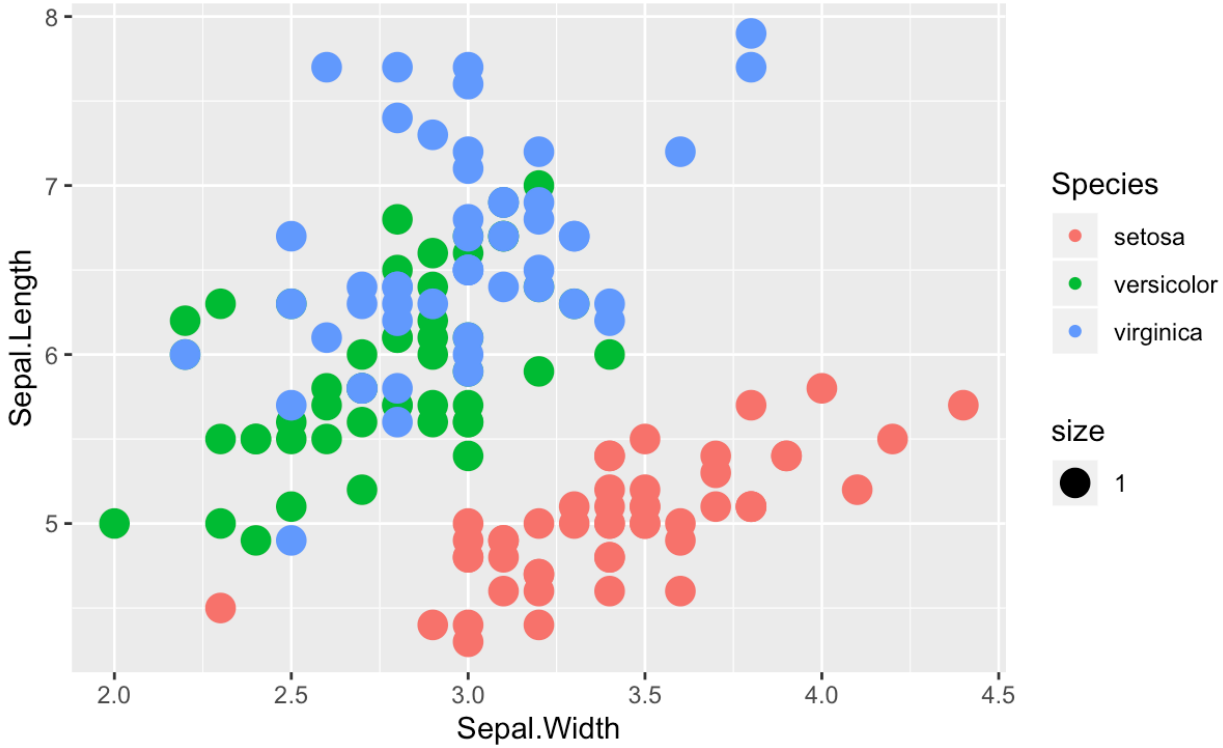


ggplot2: “Grammar of Graphics”

```
> ggplot(iris) +
+   geom_point(aes(x = Sepal.Width, y = Sepal.Length,
+                  color = Species), size = 1)
```

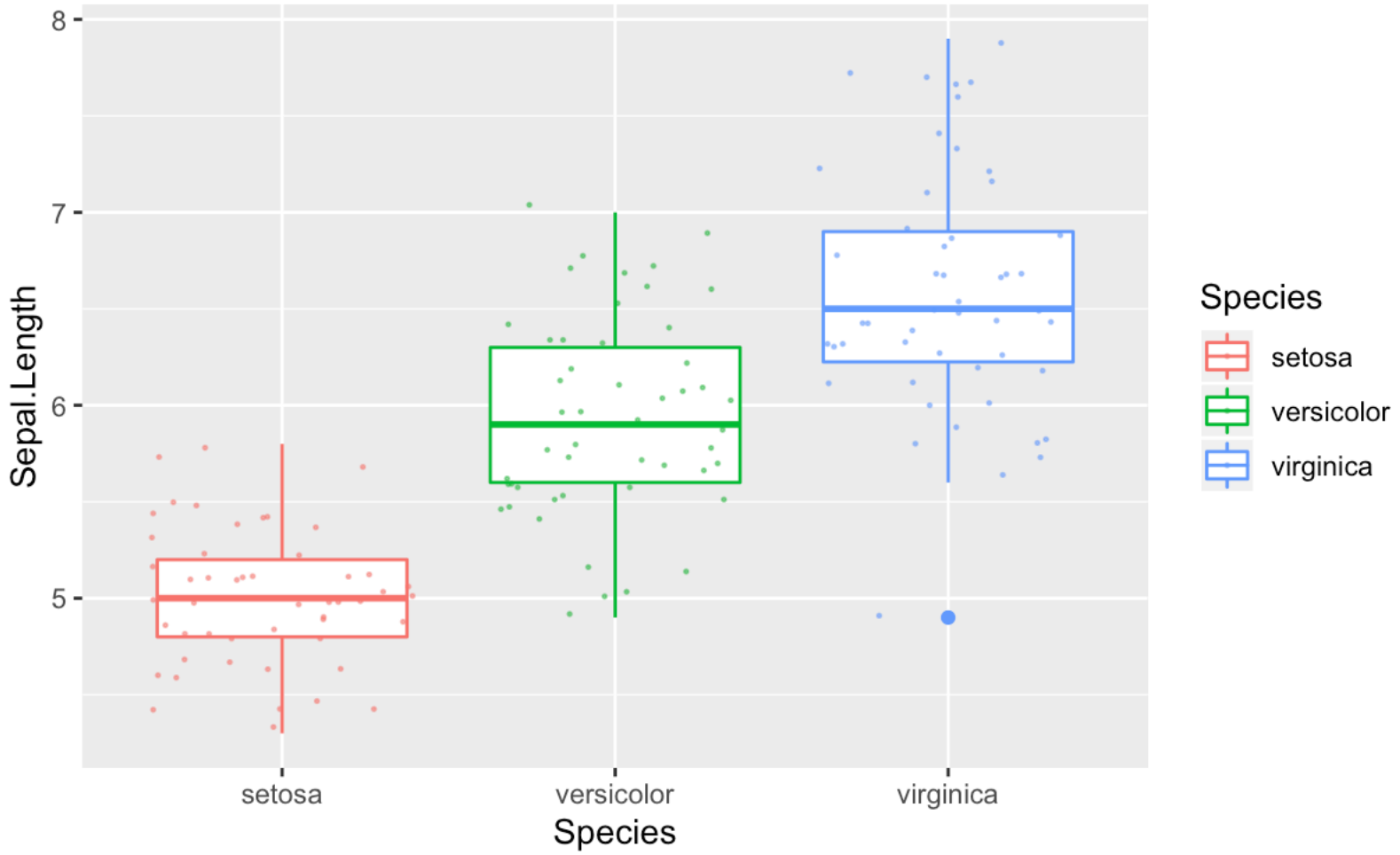



```
> ggplot(iris) +
+   geom_point(aes(x = Sepal.Width, y = Sepal.Length,
+                  color = Species, size = 1))
```



ggplot2: “Grammar of Graphics”

```
> ggplot(iris) +
+   aes(x = Species, y = Sepal.Length, color = Species) +
+   geom_boxplot() +
+   geom_jitter(size = .25, alpha = .5)
```

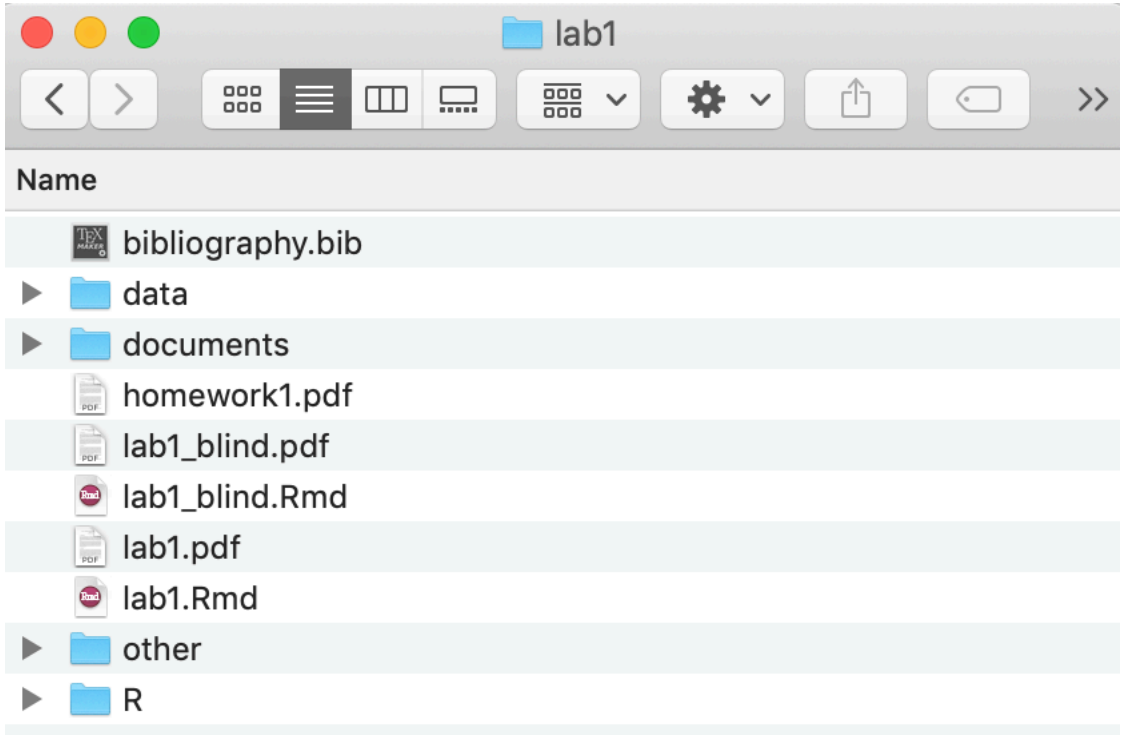


Let’s get some practice

Source: http://www.manageartworks.com/features/artwork-approval-workflows/

Project File Structure

- **data/:** store raw and processed data
- **documents/:** store relevant papers, instructions, meeting notes, etc.
- **R/:** store R code, utility functions, scripts
- **other/:** miscellaneous



Project File Structure

R/

- **load.R** – file containing function(s) for reading in the data

```
> loadData(path_to_data)
```

- **clean.R** – file containing function(x) for cleaning the loaded data

```
> cleanData(loaded_data)
```

data/

- Contains datasets
- Not uploaded to GitHub (can automate this using .gitignore)

Project File Structure

lab1.Rmd – your final report combining code (not printed in the output) and text/narrative

- Should be written like a paper; focus on communicating well

lab1.pdf – pdf output from lab1.Rmd

lab1_blind.Rmd – same as lab1.Rmd but without name

lab1_blind.pdf – pdf output from lab1_blind.Rmd

explore.Rmd (optional) – a separate .Rmd file that contains your exploratory code and figures

- A useful place for exploring the data and saving avenues of exploration that you don't necessarily want to include in your final report

bibliography.bib (optional) – a .bib file for easy citations within the lab reports

homework1.pdf – can be submitted electronically or in person at Friday lab section

Workflow: General Tips

Make code readable

- Be kind to both your peer reviewer and your future self

Keep your code modular – write functions

- Separate your functions from your analysis file (lab1.Rmd) and store them in R/
- In doing so, you create a bank of useful functions that you can load into any analysis script for your project (or future projects)
  - To load in a single file:

```
> source("./R/filename.R")
```

  - To load in all files in the R/ directory:

```
> library(R.utils)

> sourceDirectory("./R/", modifiedOnly = F, recursive = F)
```

- Group together related functions in the same .R script

  (e.g. put all data cleaning functions in clean.R)

Workflow: General Tips

Documentation

- Write lots of comments in your code and ask yourself: why are you writing this particular piece of code?
- Document functions (think about the R help pages)
  - Always add comments section immediately below the function definition line
  - What does this function do?
  - Describe the inputs and outputs

```
CalculateSampleCovariance <- function(x, y, verbose = TRUE) {
  # Computes the sample covariance between two vectors.
  # Args:
  #   x: One of two vectors whose sample covariance is to be calculated.
  #   y: The other vector. x and y must have the same length, greater than one,
  #      with no missing values.
  #   verbose: If TRUE, prints sample covariance; if not, not. Default is TRUE.
  # Returns:
  #   The sample covariance between x and y.
...
}
```

Workflow: General Tips

Test your code

- Write tests to make sure your functions are doing the right thing
- Write these tests as you go

Don’t Repeat Yourself (DRY)

- If you find yourself copying and pasting similar lines of code, write a reusable function instead

Establish consistencies – follow Google R Style Guide

Workflow: Code Style

Follow Google's R Style Guide when writing code

(See https://google.github.io/styleguide/Rguide.xml and part I Analyses of https://style.tidyverse.org/syntax.html#object-names)

Variable names

- All lowercase
- Separate words by "." or "_" (be consistent with the one you choose)

  Good: `avg.tmp, avg_tmp`

  Bad: `AvgTmp`

Workflow: Code Style

Function names

- Camel-case
- Make function names verbs

  Good: `CalculateAvgClicks, calculateAvgClicks`

  Bad: `calculate_avg_clicks, calculuate.avg.clicks`

Line Length: maximum length of 80 characters

- Go to: Preferences → code → display → check show margin and set margin column = 80

Indentation: When indenting your code, use two spaces (rather than tabs)

Workflow: Code Style

Spacing

- Place spaces around all binary operators (=, +, –, <–, etc.)
- Always put a space after a comma, never before, just like in regular English

Good:
```
df.prior <- df[df$days.from.opt < 0, "campaign.id"]

x[, 1]
```

Bad:
```
calculate_avg_clicks, calculuate.avg.clicks

x[,1], x[ , 1]
```

Line Length: maximum length of 80 characters

- Go to: Preferences → code → display → check show margin and set margin column = 80

Indentation: When indenting your code, use two spaces (rather than tabs)

Workflow: Code Style

Assignment

- Use `<-` instead of `=`

Curly Braces

- An opening curly brace should never go on its own line; a closing curly brace should always go on its own line
- Always begin the body of a block on a new line

Good:

```
if (x > 0) {
  print(x)
}
```

Bad:

```
if (x > 0) print(x)
```

Workflow: Code Style

Most importantly, BE CONSISTENT

Let's get some hands-on practice

- About Gapminder: http://www.gapminder.org/about-gapminder/
- Resources for this tutorial:
  - ggplot: http://swcarpentry.github.io/r-novice-gapminder/08-plot-ggplot2/
  - dplyr: http://swcarpentry.github.io/r-novice-gapminder/13-dplyr/
- See **lab_gapminder.Rmd** in the week2 folder on my GitHub

Lab 1
Redwood Trees

Due: Thurs,
Sep 19 @ 11:59pm

Lab 1 Goals

Data cleaning

Exploratory Data Analysis and Visualization

Lab 1 Redwood Introduction

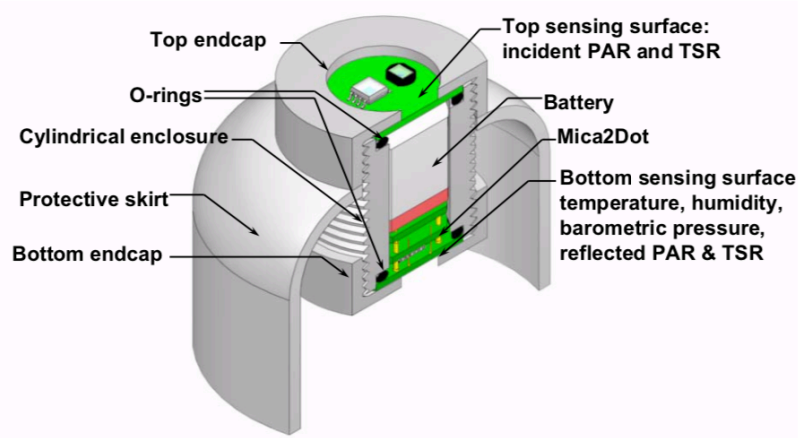


Figure 2: Sensor node and packaging

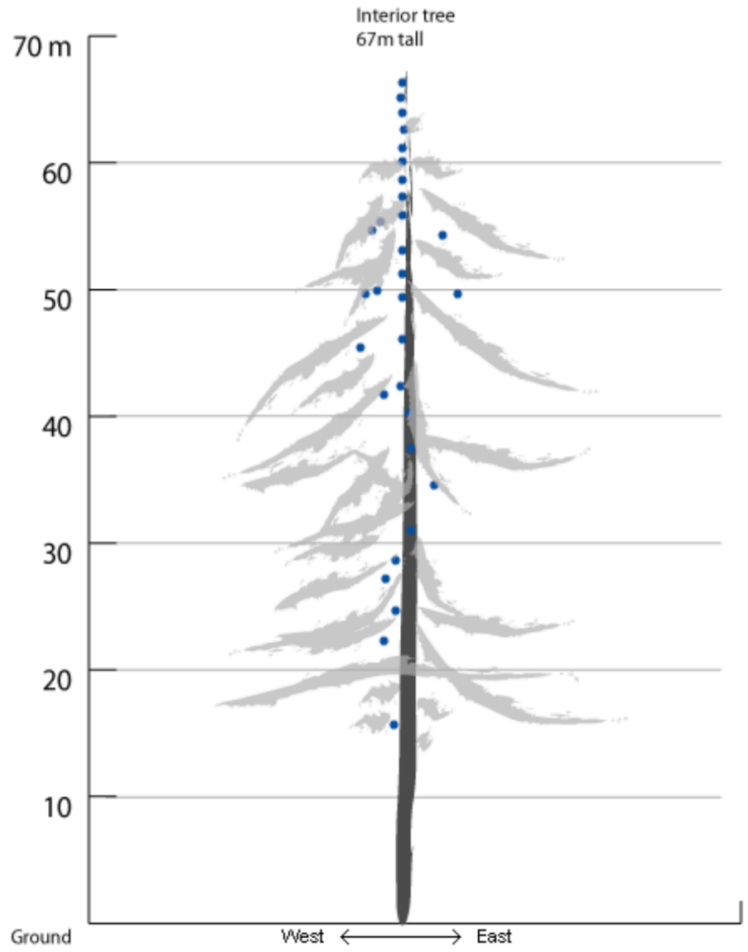


Figure 1: The placement of nodes within the tree

Lab 1 Introduction

- Read the paper carefully **sensys05-TollePolastreEtAl-redwoods.pdf** in the lab1 folder on Bcourses
- The lab1 folder on Bcourses will also contain a template to follow when putting together your lab as well as loading and cleaning functions that you may use/fill in
- The `exploration.Rmd` file was put together by Rebecca, a previous GSI, to get you started looking at the data, but you cannot use these plots as part of your lab report
- Do **not** push this `exploration.Rmd` file (or your own explore files) or the data folder to your stat-215-a repo
  - Can easily do this with `.gitignore` file

Collaboration Policy

- You are allowed to discuss **ideas** with others, but you must submit your own report
- Do not share code or copy/paste any part of the writeup
- If you do discuss ideas with others, be sure to acknowledge these students in your report

Lab 1 Rubric

Redwood Tree Lab (~60 points)

- Readability and grammar
- Readability of code (+ comments)
  - Follow Google's R Style Guide (a slight modification of the Tidyverse Style Guide)
- Reproducibility of report
  - I should be able to pull your `lab1/` folder from GitHub, manually add the `data/` folder, open `lab1.Rmd`, click knit, and get the same .pdf file as you
- Data cleaning (description and validity)
  - Describe *any* problems/inconsistencies you see with the data, how you cleaned the data, and why you cleaned the data in that way
- Three findings (creativeness, interestingness, and quality of figure)
  - Fix titles, axis and legend titles, choose appropriate color schemes, adjust size of figure
- Graphical critique
- Figures that are not for the findings (relevance and quality)
- Overall quality and level of detail of report
  - Attempts to incorporate domain information (from the paper) and place your analysis in the domain context

Homework – Some Basic Statistics (8 points)

Start Early!!!!!!!!