

# STAT 215A Fall 2020

## Week 1

---

James Duncan

# I stand on the shoulders of giants!



Thanks to Becca, Zoe, and Tiffany and other past GSIs for their hard work.

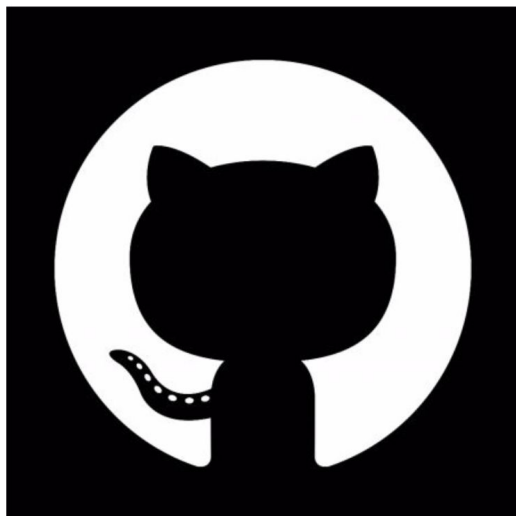
# Slides on bCourses and GitHub

- <https://github.com/jpdunc23/stat-215a-fall-2020>

# Breakout, ~5 minutes

- Different breakout groups for lab than lecture
- Introduce yourself to your neighbors
  - Name, major, year
  - What did you do this past summer?

## Git & Github



- Git basics
- Set up GitHub repos

## R & the “tidyverse”

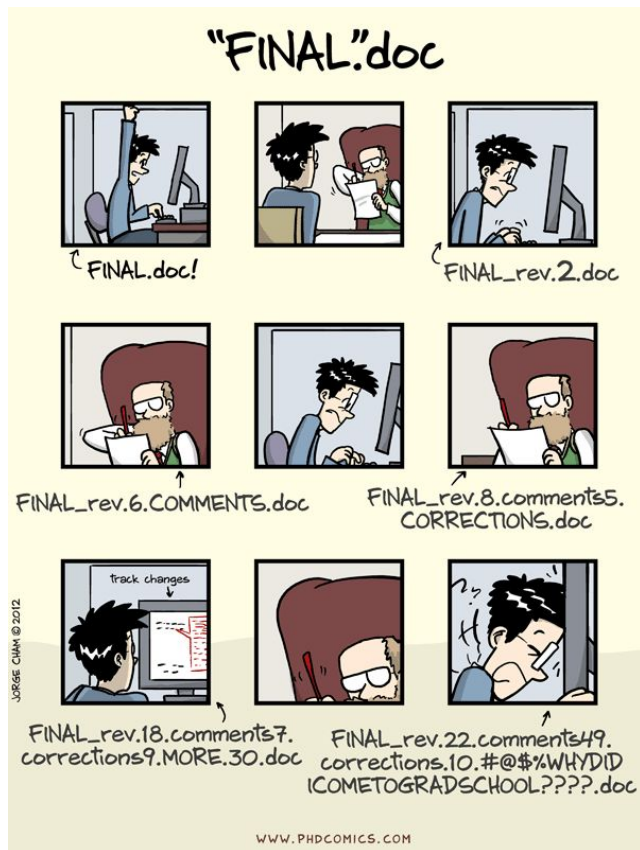


- R Markdown and knitr
- Tidyverse intro

# So, what is git?

- A version control system
- Stores data as a series of snapshots
- If files have not changed, it will simply access the file from a previous commit instead of saving it again
- Allows access to all the committed steps along the way

# The problem



# The solution



# Git vs. GitHub

## Local Git Repository

- You have a local version of the folder on your computer
- History stored in .git file
- Only you can see the changes made in the local version



## Remote GitHub Repository

- On the GitHub website lives a remote version of the folder
- Everyone can see these changes (if repository is public)





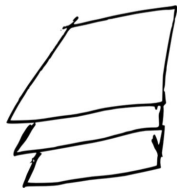
# Why is this important?

- Imagine working on a project with several collaborators...
- Using Git/GitHub allows everyone to have their own local version of the project while still maintaining a “master” version of the project, hosted remotely on GitHub
- You can make changes freely without people seeing what you are doing
  - You can thoroughly test your changes before adding to the master copy
- Version control!!
  - Especially great if your changes create bugs because you can backtrack/revert

# Typical Pipeline

## (2) make local changes

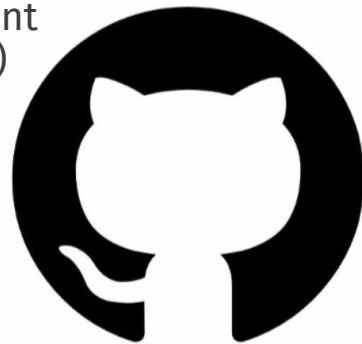
(e.g. create file called filename.txt)



(1) **git pull**  
(to retrieve the most recent version from the server)



**Remote Repository**



## (3) **git add filename.txt**

## (4) **git commit -m "add description"**

(add and commit when you have made some changes and want to be able to save your current checkpoint as a snapshot)

## (5) **git push**

(make changes available to everyone with access to the repo)

**Warning:** Good idea to ``git pull`` before you ``git push`` and then you may have to mitigate any merge conflicts.

Time to set up your GitHub repos...

---

# Before class, hopefully you:

1. Installed Git on your system  
(<https://git-scm.com/book/en/v2/Getting-Started-Installing-Git>)
2. Signed up for GitHub (<https://github.com/>)
3. ~~Went to <https://education.github.com/students> and signed up for the student pack to get unlimited private repositories via GitHub Pro.~~

As of January 2019, GitHub allows unlimited private repos for individuals (but you can still get lots of benefits from the GitHub Student Developer Pack: <https://education.github.com/pack>).

# Setting up your GitHub repo

4. Locally on your machine, open a terminal and change directories to where you want to keep class materials
5. Clone my stat-215-a repository: **git clone https://github.com/jpdunc23/stat-215-a**  
This will create a copy of my repository on your own computer

```
james@james-HP-Spectre-x360 ~ ➤ git clone https://github.com/jpdunc23/stat-215-a
```

6. On the GitHub website, log in and create a **private** remote repository called **stat-215-a**. Add me (**jpdunc23**) as a collaborator for this repository (check out settings on the repo website).

# Setting up your GitHub repo

7. Back in the terminal, change directories to your local stat-215-a folder:  
**cd ./stat-215-a**
8. Set the origin of your local repo to be the remote repo that you just made: **git remote set-url origin https://github.com/USERNAME/stat-215-a.git** (change USERNAME to your username). This tells git which remote repository to push your changes to when you use git push.

```
james@james-HP-Spectre-x360 ~ ➤ cd ./stat-215-a
james@james-HP-Spectre-x360 ~/stat-215-a ➤ git remote set-url origin https://github.com/jpdunc23/stat-215-a.git
```

9. Edit *info.txt* to reflect your own information.

# Setting up your GitHub repo

10. Now we need to push the changes you made to your remote repo:

a. In terminal, type **git status** and you should see that *info.txt* has been modified:

```
james@james-HP-Spectre-x360 ~/stat-215-a master ● git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   info.txt
```

b. Add (**git add info.txt**) and commit (**git commit -m "Updated info.txt with my own information"**) your edited *info.txt* file.

```
james@james-HP-Spectre-x360 ~/stat-215-a master ● git add info.txt
james@james-HP-Spectre-x360 ~/stat-215-a master + git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   info.txt

james@james-HP-Spectre-x360 ~/stat-215-a master + git commit -m"Updated info.txt with my own information"
[master 01b9350] Updated info.txt with my own information
1 file changed, 1 insertion(+)
```

c. Finally, use **git push** to send your changes to the remote repo on GitHub

5 min break

---





# R Tidyverse



# Tidy Data is at the core of the Tidyverse

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/>

**Nice resource:** <http://www.rebeccabarter.com/tutorials/>

**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

**Tidyverse is great, but not perfect:** <https://github.com/matloff/TidyverseSkeptic>



# dplyr: “Grammar of data manipulation”

- Contains functions for manipulating variables in a data frame or tibble:
  - **filter()**: find rows where a 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 on the entire dataset to operating on subgroups.
  - **summarize()**: reduces multiple values down to a single summary.
  - **arrange()**: order rows in a data frame using specific column(s).
- The first argument of each function is a data frame (or tibble)
- The result is always a new data frame (or tibble).

# 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
```

# 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(),<sup>23</sup>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 |



# 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

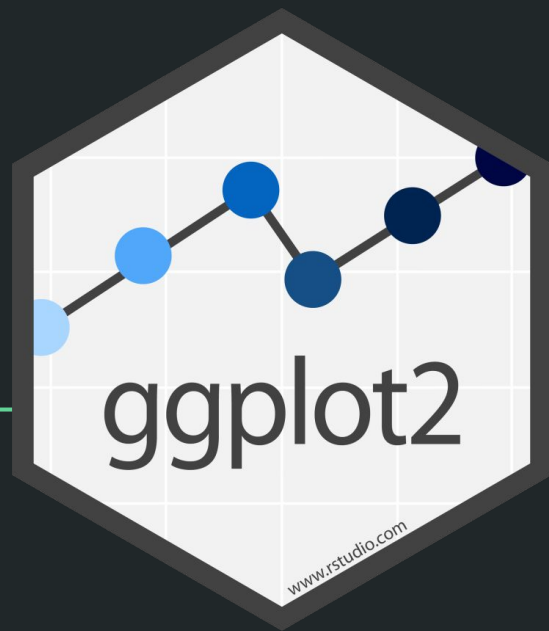
```
> iris_wide <- iris_long %>%  
+   spread(key = "Variable", value = "Value")  
> head(iris_wide)
```

|   | 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   |





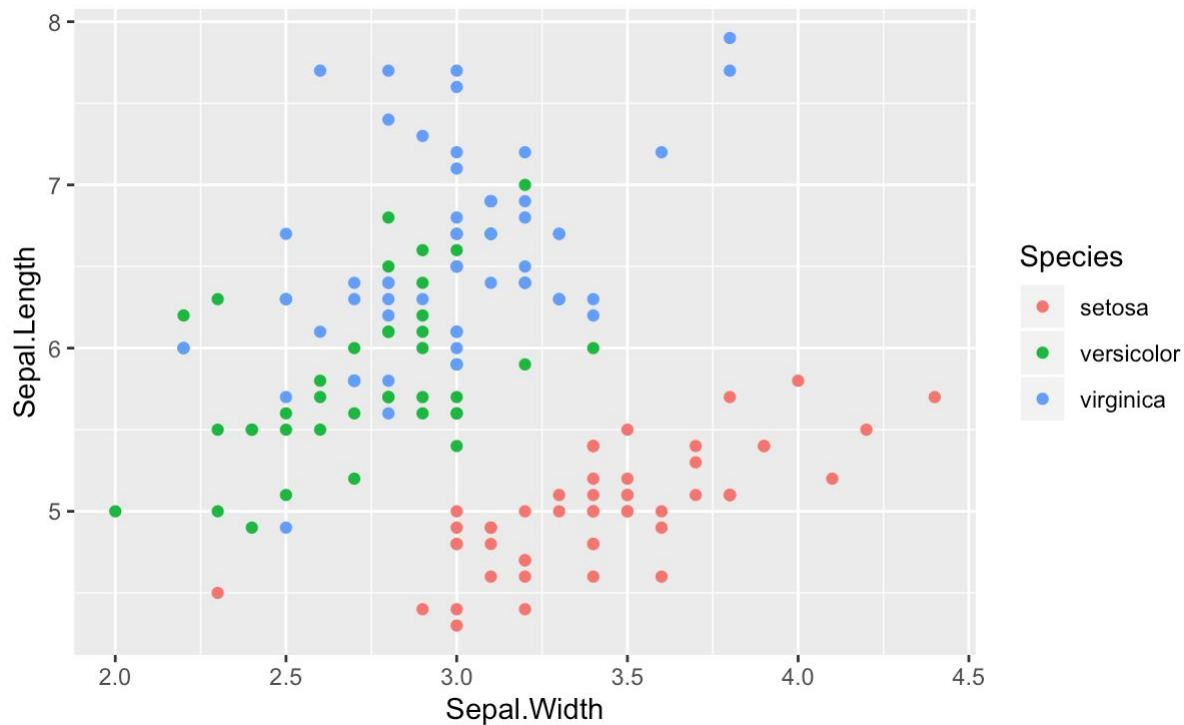
# ggplot2: “Grammar of Graphics”

- 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))
```

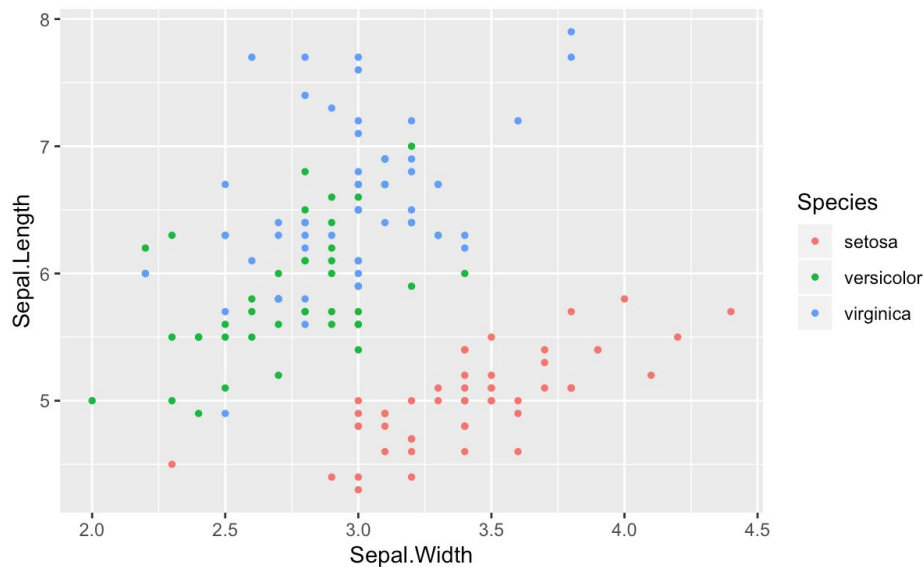
- You can do many many things with ggplot, and I can't even begin to list the different layers, color themes, formatting options that you can specify in ggplot. **Google will be your best friend here!**

# ggplot2: “Grammar of Graphics”

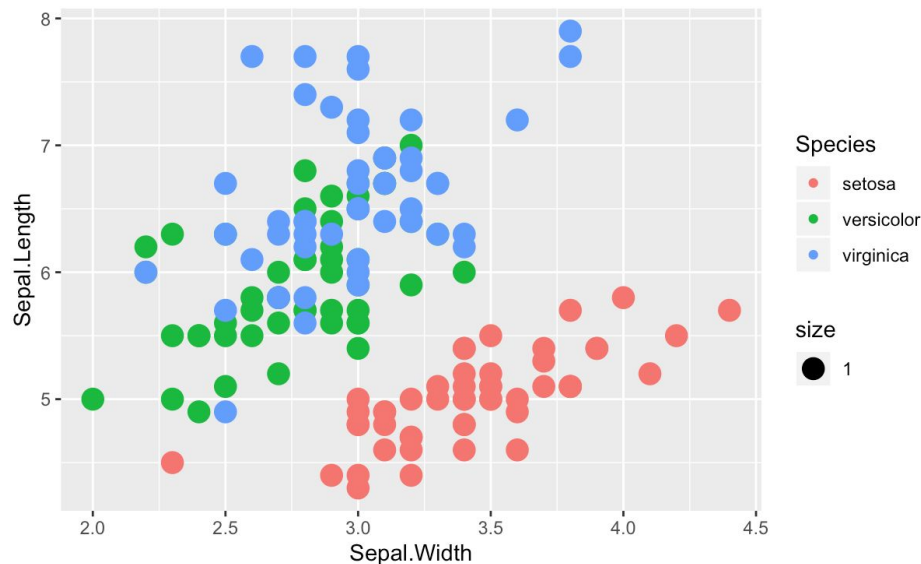


# ggplot2: “Grammar of Graphics”

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

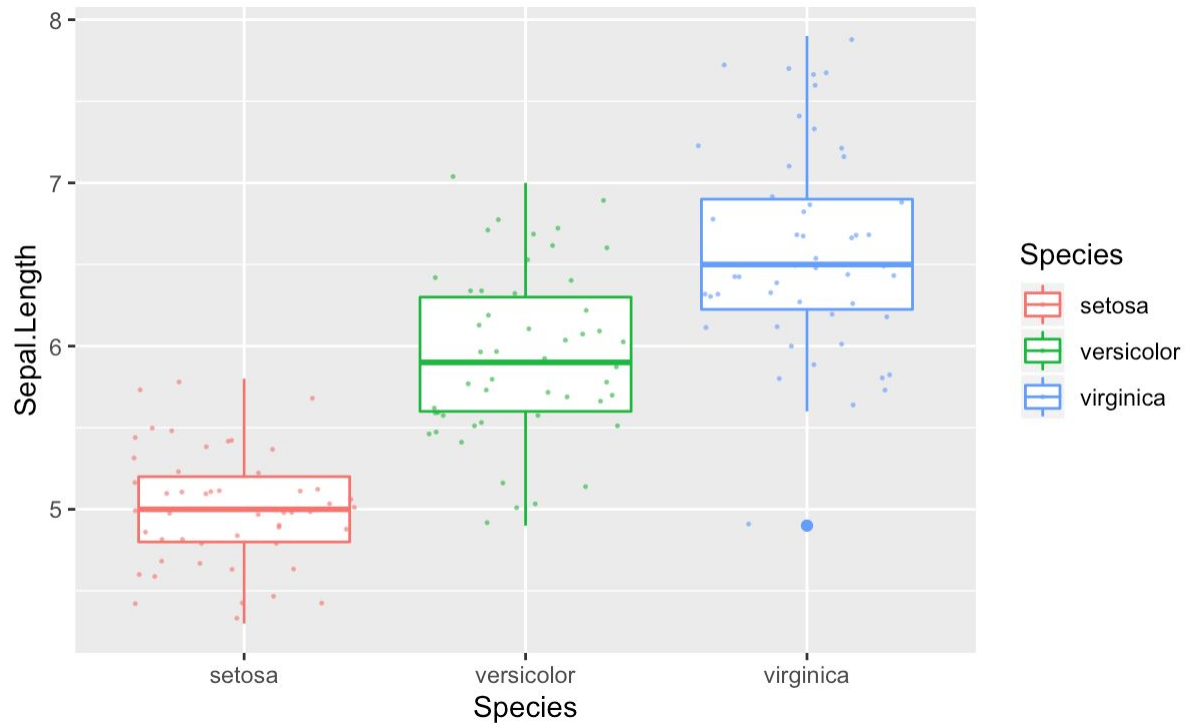


```
> ggplot(iris) +  
+   geom_point(aes(x = Sepal.Width, y = Sepal.Length,  
+                 color = Species, size = 1))
```



# 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...

---

# For next week

- Lab 0 is due next Friday (Sep 4)
- Lab 0 is not for a grade, but at a minimum, please submit an empty project/lab so that we can ensure your GitHub is working accordingly
- Tentative agenda for next week:
  - Workflow + R Markdown
  - Tips and tricks to make life easier in R/R Markdown
  - Lab 1 Assigned