**Week 2 – Introducing R through Tidyverse and Git Name \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
DATA 110 Date \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
Saidi**

* Introducing Tidyverse
* Ggplot2
* Tibble
* Github

# Introducing Tidyverse

[**Click here**](http://rpubs.com/rsaidi/518417) for my Rpubs file for these notes.

# **Week 2 Intro to Tidyverse**

**Rachel Saidi**

**August 19, 2019**

# Load Data from Three Different Sources

In the following notes, you will load data directly from a URL, directly from pre-build datasets in R, and finally from a file you save in your own folder.

## Load Data from a URL

You can load data from a folder or you can load data directly from a URL. The next example loads the dataset, “Allscores”, directly from the URL where it resides.

# install.packages("tidyverse")  
getwd()

## [1] "C:/Users/rsaidi/Dropbox/Rachel/MontColl/DATA110/Notes"

library(tidyverse)

## -- Attaching packages -------------------------------------- tidyverse 1.2.1 --

## v ggplot2 3.1.0 v purrr 0.3.2  
## v tibble 2.1.3 v dplyr 0.8.3  
## v tidyr 0.8.3 v stringr 1.4.0  
## v readr 1.3.1 v forcats 0.4.0

## Warning: package 'tibble' was built under R version 3.5.3

## Warning: package 'tidyr' was built under R version 3.5.3

## Warning: package 'purrr' was built under R version 3.5.3

## Warning: package 'dplyr' was built under R version 3.5.3

## -- Conflicts ----------------------------------------- tidyverse\_conflicts() --  
## x dplyr::filter() masks stats::filter()  
## x dplyr::lag() masks stats::lag()

allscores <- readr::read\_csv("https://goo.gl/MJyzNs")

## Parsed with column specification:  
## cols(  
## group = col\_double(),  
## pre = col\_double(),  
## post = col\_double(),  
## diff = col\_double()  
## )

dim(allscores)

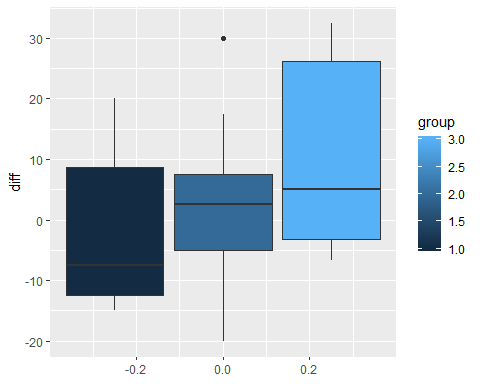
## [1] 22 4

Notice R interprets the variable “group” as continuous values (col\_double). We will fix this later. The command “dim” provides the dimensions of the data, which are 22 observations (rows) by 4 variables (columns).

## Use Side-by-Side Boxplots

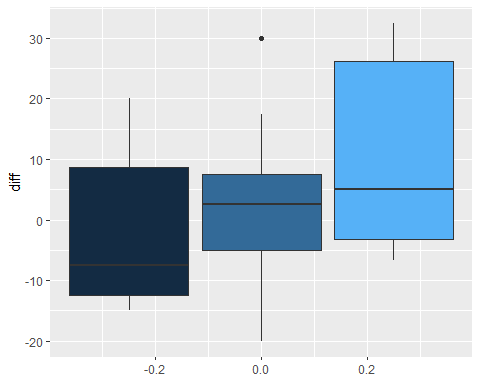
Here is some easy code to create 3 groups of boxplots with some easy-to-access data, filled by group. Since the groups are discrete, you can get rid of the shading.

boxpl <- allscores %>% ggplot() + geom\_boxplot(aes(y=diff, group=group,  
 fill=group))  
boxpl



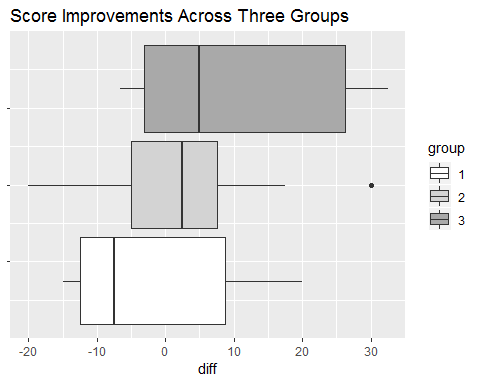
Notice that the legend give a continuous range of values for the scores, even though the scores are only 1, 2, or 3. The code guides(fill = FALSE) will get rid of the legend.

boxpl2 <- boxpl + guides(fill=FALSE)  
boxpl2



Ensure that the groups are considered as factors, rather than numbers. Then manually fill with the 3 colors: white, light gray, and dark gray. Make the boxplots orient horizontally.

allscores %>%  
 mutate(group=factor(group, levels=c("1","2","3"), ordered=TRUE)) %>%  
 ggplot() + geom\_boxplot(aes(y=diff, group=group, fill=group)) +  
 scale\_fill\_manual(values=c("white","lightgray","darkgray")) +  
 theme(axis.text.y=element\_blank()) +  
 ggtitle("Score Improvements Across Three Groups") +  
 coord\_flip()



## Load Built in Data from R

Some data frames are built in to R, such as mpg. Load the data, then use str and head to look at the data.

{r mpg} loads the data. Alternatively, you can use the command: load(“mpg”)

You will look at the data using the command “str” (gives the structure of the data), “head” (lists the first 6 rows of observations in the dataset), and “describe” from the “psych” package (gives quite detailed summary statistics on the continuous variables).

# install.packages("tidyverse")  
# install.packages("psych")  
library(tidyverse)  
library(psych)

##   
## Attaching package: 'psych'

## The following objects are masked from 'package:ggplot2':  
##   
## %+%, alpha

str(mpg)

## Classes 'tbl\_df', 'tbl' and 'data.frame': 234 obs. of 11 variables:  
## $ manufacturer: chr "audi" "audi" "audi" "audi" ...  
## $ model : chr "a4" "a4" "a4" "a4" ...  
## $ displ : num 1.8 1.8 2 2 2.8 2.8 3.1 1.8 1.8 2 ...  
## $ year : int 1999 1999 2008 2008 1999 1999 2008 1999 1999 2008 ...  
## $ cyl : int 4 4 4 4 6 6 6 4 4 4 ...  
## $ trans : chr "auto(l5)" "manual(m5)" "manual(m6)" "auto(av)" ...  
## $ drv : chr "f" "f" "f" "f" ...  
## $ cty : int 18 21 20 21 16 18 18 18 16 20 ...  
## $ hwy : int 29 29 31 30 26 26 27 26 25 28 ...  
## $ fl : chr "p" "p" "p" "p" ...  
## $ class : chr "compact" "compact" "compact" "compact" ...

head(mpg)

## # A tibble: 6 x 11  
## manufacturer model displ year cyl trans drv cty hwy fl class  
## <chr> <chr> <dbl> <int> <int> <chr> <chr> <int> <int> <chr> <chr>  
## 1 audi a4 1.8 1999 4 auto(~ f 18 29 p comp~  
## 2 audi a4 1.8 1999 4 manua~ f 21 29 p comp~  
## 3 audi a4 2 2008 4 manua~ f 20 31 p comp~  
## 4 audi a4 2 2008 4 auto(~ f 21 30 p comp~  
## 5 audi a4 2.8 1999 6 auto(~ f 16 26 p comp~  
## 6 audi a4 2.8 1999 6 manua~ f 18 26 p comp~

describe(mpg)

## vars n mean sd median trimmed mad min max range  
## manufacturer\* 1 234 NaN NA NA NaN NA Inf -Inf -Inf  
## model\* 2 234 NaN NA NA NaN NA Inf -Inf -Inf  
## displ 3 234 3.47 1.29 3.3 3.39 1.33 1.6 7 5.4  
## year 4 234 2003.50 4.51 2003.5 2003.50 6.67 1999.0 2008 9.0  
## cyl 5 234 5.89 1.61 6.0 5.86 2.97 4.0 8 4.0  
## trans\* 6 234 NaN NA NA NaN NA Inf -Inf -Inf  
## drv\* 7 234 4.00 0.00 4.0 4.00 0.00 4.0 4 0.0  
## cty 8 234 16.86 4.26 17.0 16.61 4.45 9.0 35 26.0  
## hwy 9 234 23.44 5.95 24.0 23.23 7.41 12.0 44 32.0  
## fl\* 10 234 NaN NA NA NaN NA Inf -Inf -Inf  
## class\* 11 234 NaN NA NA NaN NA Inf -Inf -Inf  
## skew kurtosis se  
## manufacturer\* NA NA NA  
## model\* NA NA NA  
## displ 0.44 -0.91 0.08  
## year 0.00 -2.01 0.29  
## cyl 0.11 -1.46 0.11  
## trans\* NA NA NA  
## drv\* NaN NaN 0.00  
## cty 0.79 1.43 0.28  
## hwy 0.36 0.14 0.39  
## fl\* NA NA NA  
## class\* NA NA NA

It is essential to recognize that variables may be: int (integer), num (numeric), or double vs char (character) and factor (for categories)

Typically, chr or factor are used for discrete variables and int, dbl, or num for continuous variables.

## Introducing ggplot2 and the grammar of graphics

The “gg” in ggplot2 stands for “grammar of graphics,” an approach to drawing charts devised by the statistician Leland Wilkinson. Rather than thinking in terms of finished charts like a scatter plot or a column chart, it starts by defining the coordinate system (usually the X and Y axes of a cartesian system), maps data onto those coordinates, and then adds layers such as points, bars and so on. This is the logic behind ggplot2 code.

Some key things to understand about ggplot2:

* ggplot This is the master function that creates a ggplot2 chart.
* aes This function, named for “aesthetic mapping,” is used whenever data values are mapped onto a chart. So it is used when you define which variables are plotted onto the X and Y axes, and also if you want to change the size or color of parts of the chart according to values for a variable.
* geom All of the functions that add layers to a chart start with geom, followed by an underscore, for example geom\_point() or geom\_bar(). The code in the brackets for any geom layer styles the items in that layer, and can include aes mappings of values from data.
* theme This function modifies the appearance of elements of a plot, used, for example, to set size and font face for text, the position of a legend, and so on.
* scale Functions that begin with scale, followed by an underscore, are used to modify the way an aes mapping of data appears on a chart. They can change the axis range, for example, or specify a color palette to be used to encode values in the data.
* + is used each time you add a layer, a scale, a theme, or elements like axis labels and a title After a + you can continue on the same line of code or move the next line. I usually write a new line after each +, which makes the code easier to follow.

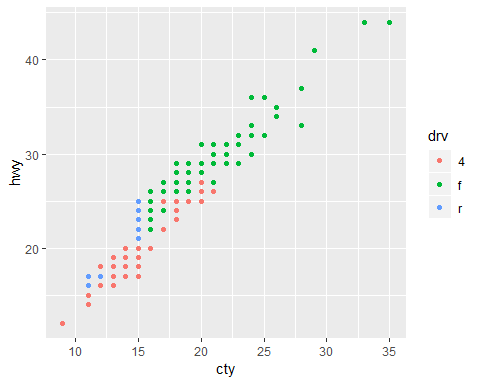
## Now make a scatterplot using ggplot2

Make a scatterplot of city vs highway miles per gallon, but sort/color points by either 4-wheel, front-wheel, or rear-wheel drive

Here is how we will code:

1. name the plot: “plot1” <-
2. call back the name of the dataset “mpg” and “pipe it” (more on that later) to create the frame for your plot
3. call “ggplot” to make a set of axes, with the aesthetics (aes) for city and highway mpg, but color points by the factors for drv
4. add geom\_point to see the points
5. call plot1 to see the entire plot

plot1 <- mpg %>% ggplot(aes(cty, hwy, color = drv))+   
 geom\_point()  
plot1

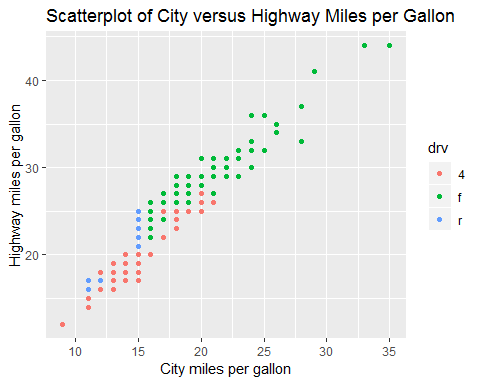


Notice that the blue points for rear-wheel drive are only at the lower left side of the plot (i.e., not great mpg). Red points for 4-wheel drive have a wider spread of points, but they are also mainly at the lower left corner of the plot. The green points for front-wheel drive are mostly at the upper right, for the higher mpg.

## Add a title and labels

Although there are already axes labels, we can do better. We should also add a title

plot1 <- mpg %>% ggplot(aes(cty, hwy, color = drv))+   
 geom\_point()+  
 xlab("City miles per gallon") +  
 ylab("Highway miles per gallon") +  
 ggtitle("Scatterplot of City versus Highway Miles per Gallon")  
plot1



## Clean data headings and variable names

Very soon, you will find data from other sources. The data will require some cleaning. Here are some important points to check: 1. Be sure the format is .csv 2. Be sure there are no spaces between variable names (headers). 3. Set all variable names to lowercase so you do not have to keep track of capitalizing.

Download this dataset, Household\_debt, from <http://bit.ly/2P3084E> and save it in your dataset folder. Change your working directory to load the dataset from YOUR folder. Then run this code.

setwd("C:/Users/rsaidi/Dropbox/Rachel/MontColl/Datasets/Datasets")  
household <- read\_csv("household\_debt.csv")

## Parsed with column specification:  
## cols(  
## Period = col\_character(),  
## Mortgage = col\_double(),  
## `HE Revolving` = col\_double(),  
## `Auto Loan` = col\_double(),  
## `Credit Card` = col\_double(),  
## `Student Loan` = col\_double(),  
## Other = col\_double(),  
## Total = col\_double()  
## )

### Make all headings (column names) lowercase. Then remove all spaces between words in headings and replace them with underscores. Then look at it with “head”.

names(household) <- tolower(names(household))  
names(household) <- gsub(" ","\_",names(household))  
head(household)

## # A tibble: 6 x 8  
## period mortgage he\_revolving auto\_loan credit\_card student\_loan other  
## <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  
## 1 03:Q1 4.94 0.24 0.64 0.69 0.24 0.48  
## 2 03:Q2 5.08 0.26 0.62 0.69 0.24 0.49  
## 3 03:Q3 5.18 0.27 0.68 0.69 0.25 0.48  
## 4 03:Q4 5.66 0.3 0.7 0.7 0.25 0.45  
## 5 04:Q1 5.84 0.33 0.72 0.7 0.26 0.45  
## 6 04:Q2 5.97 0.37 0.74 0.7 0.26 0.42  
## # ... with 1 more variable: total <dbl>

**Tibble Data Format in R: Best and Modern Way to Work with Your Data**

<https://cran.r-project.org/web/packages/tibble/vignettes/tibble.html>

The **tibble** R package provides easy to use functions for creating tibbles, which is a modern rethinking of data frames.

Compared to the traditional [**data.frame**()](http://www.sthda.com/wiki/(easy-r-programming-basics#data-frames)), the modern **data\_frame**():

* never converts string as factor
* never changes the names of variables
* never create row names

Tibbles are a modern take on data frames. They keep the features that have stood the test of time, and drop the features that used to be convenient but are now frustrating (i.e. converting character vectors to factors).

**Advantages of tibbles compared to data frames**

1. Tibbles have nice printing method that show only the first 10 rows and all the columns that fit on the screen. This is useful when you work with large data sets.
2. When printed, the data type of each column is specified (see below):
   * : for double
   * : for factor
   * : for character
   * : for logical

read\_csv()), which are faster than R base functions and import data into R as a **tbl\_df** (pronounced as “tibble diff”).

**tbl\_df** object is a data frame providing a nicer printing method, useful when working with large data sets.

* It never adjusts the names of variables:

**names**(**data.frame**(`crazy name` = 1))

#> [1] "crazy.name"

**names**(**tibble**(`crazy name` = 1))

#> [1] "crazy name"

* It evaluates its arguments lazily and sequentially:

**tibble**(x = 1:5, y = x ^ 2)

#> # A tibble: 5 x 2

#> **x** **y**

#> *<int>* *<dbl>*

#> 1 1 1

#> 2 2 4

#> 3 3 9

#> 4 4 16

#> 5 5 25

* It never uses row.names(). The whole point of tidy data is to store variables in a consistent way. So it never stores a variable as special attribute.

## Tibbles vs data frames

There are three key differences between tibbles and data frames: printing, subsetting, and recycling rules.

### Printing

When you print a tibble, it only shows the first ten rows and all the columns that fit on one screen. It also prints an abbreviated description of the column type, and uses font styles and color for highlighting:

**tibble**(x = -5:1000)

#> # A tibble: 1,006 x 1

#> x

#> *<int>*

#> 1 -5

#> 2 -4

#> 3 -3

#> 4 -2

#> 5 -1

#> 6 0

#> 7 1

#> 8 2

#> 9 3

#> 10 4

#> # … with 996 more rows

You can control the default appearance with options:

* options(tibble.print\_max = n, tibble.print\_min = m): if there are more than n rows, print only the first mrows. Use options(tibble.print\_max = Inf) to always show all rows.
* options(tibble.width = Inf) will always print all columns, regardless of the width of the screen.

## **Subsetting**

Tibbles are quite strict about subsetting. [ always returns another tibble. Contrast this with a data frame: sometimes [ returns a data frame and sometimes it just returns a vector:

df1 <- **data.frame**(x = 1:3, y = 3:1)

**class**(df1[, 1:2])

#> [1] "data.frame"

**class**(df1[, 1])

#> [1] "integer"

df2 <- **tibble**(x = 1:3, y = 3:1)

**class**(df2[, 1:2])

#> [1] "tbl\_df" "tbl" "data.frame"

**class**(df2[, 1])

#> [1] "tbl\_df" "tbl" "data.frame"

To extract a single column use [[ or $:

**class**(df2[[1]])

#> [1] "integer"

**class**(df2$x)

#> [1] "integer"

Tibbles are also stricter with $. Tibbles never do partial matching, and will throw a warning and return NULL if the column does not exist:

df <- **data.frame**(abc = 1)

df$a

#> [1] 1

df2 <- **tibble**(abc = 1)

df2$a

#> Warning: Unknown or uninitialised column: 'a'.

#> NULL

As of version 1.4.1, tibbles no longer ignore the drop argument:

**data.frame**(a = 1:3)[, "a", drop = TRUE]

#> [1] 1 2 3

**tibble**(a = 1:3)[, "a", drop = TRUE]

#> [1] 1 2 3

### **tibble subclass**

readr now returns results with a spec\_tbl\_df subclass. This differs from a regular tibble only in that the spec attribute (which holds the column specification) is lost as soon as the object is subset and a normal tbl\_df object is returned.

Historically tbl\_df’s lost their attributes once they were subset. However recent versions of tibble retain the attributes when subsetting, so the spec\_tbl\_df subclass is needed to retain the previous behavior.

This should only break compatibility if you are explicitly checking the class of the returned object. A way to get backwards compatible behavior is to call subset with no arguments on your object, e.g. x[].

data <- read\_csv(file)

class(data)

*#> [1] "spec\_tbl\_df" "tbl\_df" "tbl" "data.frame"*

class(data[])

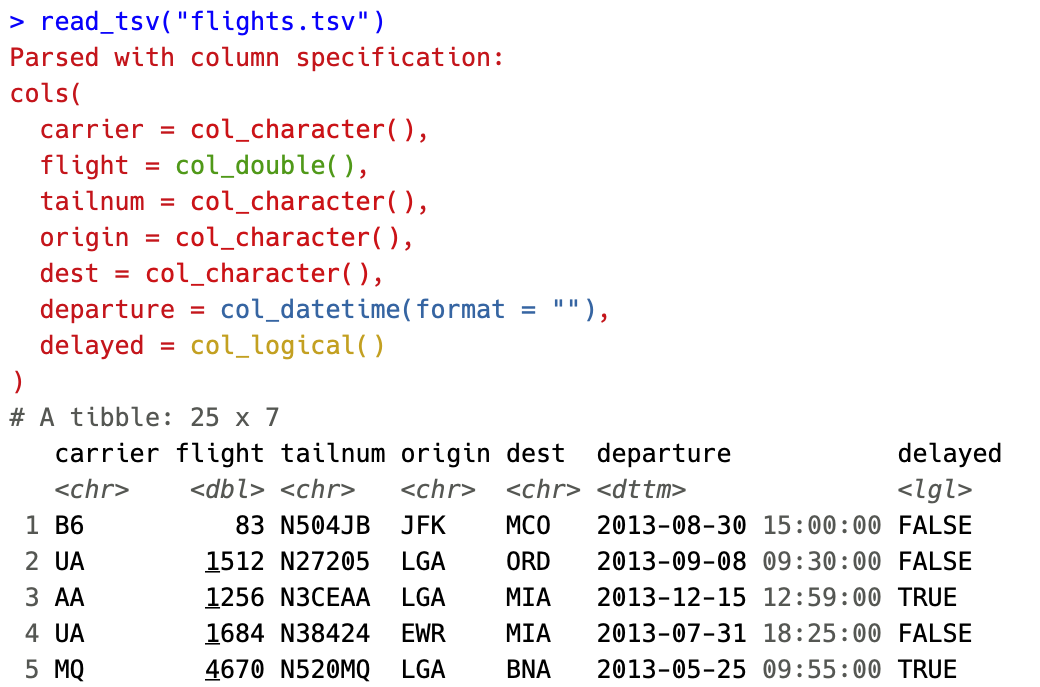
*#> [1] "tbl\_df" "tbl" "data.frame"*

## **New features**

### **Colored specifications**

The most user visible change is coloration of the column specifications. The column types are now colored based on 4 broad classes

* **Red** - Character data (characters, factors)
* **Green** - Numeric data (doubles, integers)
* **Yellow** - Logical data (logicals)
* **Blue** - Temporal data (dates, times, datetimes)



By coloring the specification, we hope to make it easier to spot when a column differs from the rest or when guessing leads to import with an unexpected type.

# The coloring can be disabled by setting options(crayon.enabled = FALSE).

Using GitHub

(From Peter Aldhous) In this week’s class we will learn the basics of version control, so that you can work in a clean folder with a single set of files, but can save snapshots of versions of your work at each point and return to them if necessary.

Version control was invented for programmers working on complex coding projects. But it is good practice for any project — even if all you are managing are versions of a simple website, or a series of spreadsheets.

This tutorial borrows from the [Workflow and GitHub](https://newmedia.report/classes/coding/2016/workflow-and-github/) lesson in Jeremy Rue’s [Advanced Coding Interactives](https://newmedia.report/classes/coding/2016/) class and Coursera’s [Data Science Toolbox](https://www.coursera.org/learn/data-scientists-tools/home/welcome) — see the further reading links below.

Introducing Git, GitHub and GitHub Desktop

The version control software we will use is called [**Git**](https://git-scm.com/). It is installed automatically when you install and configure [**GitHub Desktop**](https://desktop.github.com/). GitHub Desktop allows you to manage version control for local versions of projects on your own computer, and sync them remotely with [**GitHub**](https://github.com/). GitHub is a social network, based on Git, that allows developers to view and share one another’s code, and collaborate on projects.

Even if you are working on a project alone, it is worth regularly synching to GitHub. Not only does this provides a backup copy of the entire project history in the event of a problem with your local version, but GitHub also allows you to host websites. This means you can go straight from a project you are developing to a published website. If you don’t already have a personal portfolio website, you can host one for free on GitHub.

The files we will use today

Download the files for this session from [here](file:///C:\Users\rsaidi\Dropbox\Rachel\MontColl\DATA110\Notes\week4), unzip the folder and place it on your desktop. It contains the following folders and files:

index.html index2.html Two simple webpages, which we will edit and publish on GitHub.  
css fonts js Folders with files to run the Bootstrap web framework.

Some terminology

* repository or repo Think of this as a folder for a project. A repository contains all of the project files, and stores each file’s revision history. Repositories on GitHub can have multiple collaborators and can be either public or private.
* clone Copy a repository from GitHub to your local computer.
* master This is the main version of your repository, created automatically when you make a new repository.
* branch A version of your repository separate from the master branch. As you switch back and forth between branches, the files on your computer are automatically modified to reflect those changes. Branches are used commonly when multiple collaborators are working on different aspects of a project.
* pull request Proposed changes to a repository submitted by a collaborator who has been working on a branch.
* merge Taking the changes from one branch and applying them to another. This is often done after a pull request.
* push or sync Submitting your latest commits to the remote repository, on GitHub and syncing any changes from there back to your computer.
* gh-pages A special branch that is published on the web. This is how you host websites on GitHub. Even if a repository is private, its published version will be visible to anyone who has the url.
* fork Split off a separate version of a repository. You can fork anyone’s code on GitHub to make your own version of their repo.

[Here](https://help.github.com/articles/github-glossary/) is a more extended GitHub glossary.

[Here](https://github.com/rjsaidi) is a link to my GitHub Page. (<https://github.com/rjsaidi>)

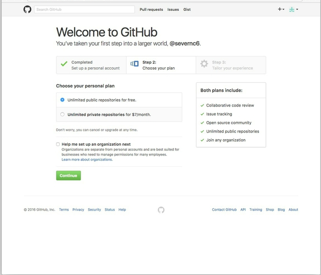
[Here](https://www.youtube.com/watch?v=l40x1EshOBE) is a great link to GitHub introductory training.

Create and secure your GitHub account

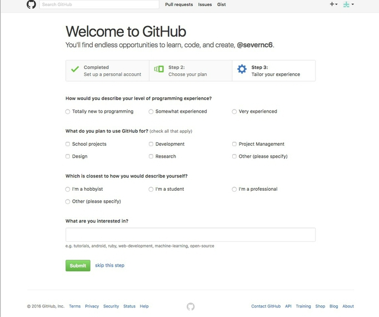
Navigate to [GitHub](https://github.com/) and sign up:



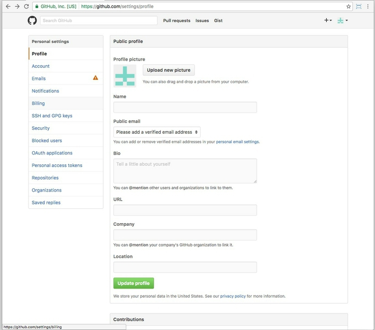
Choose your plan. If you want to be able to create private repositories, which cannot be viewed by others on the web, you will need to upgrade to a paid account. But for now select a free account and click Continue:



At the next screen, click the skip this step link:



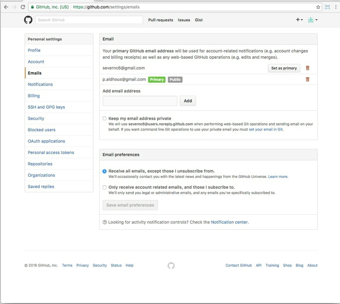
Now view your profile by clicking on the icon at top right and selecting **Your profile**. This is your page on GitHub. Click Edit profile to see the following:

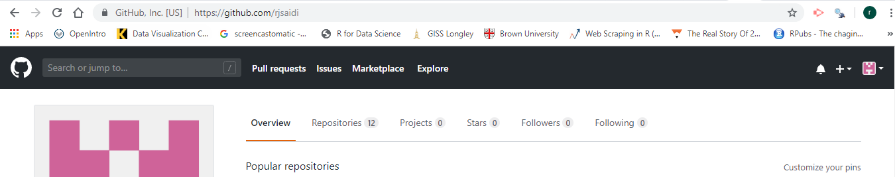


Here you can add your personal details, and a profile picture. For now just add the name you want to display on GitHub. Fill in the rest in your own time after class.

You should have been sent a confirmation email to the address you used to sign up. Click on the verification link to verify this address on GitHub.

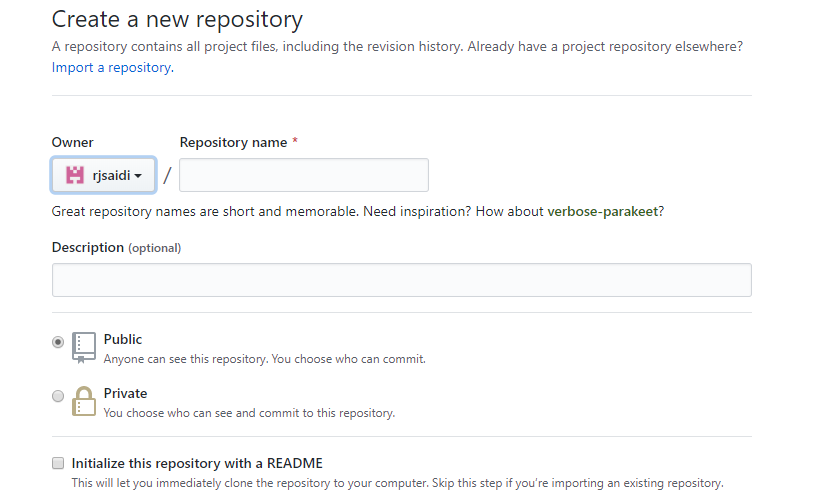
Back on the GitHub website, click on the **Emails** link in the panel at left. If you wish, you can add another email to use on GitHub, which will need to be verified as well. If you don’t wish to display your email on GitHub check the **Keep my email address private** box.





## Creating a Github Repository

1. Start a repo from scratch
2. Click “Create a New Repo” (click the plus sign at the top right) or
3. Github.com/new
4. “Fork” another’s repo
5. Request to “Pull” another’s repo to make edits
6. “Push” a repo



When you create a new repo, create a **Googleable**  Repository name and description.

Make it “Public”

Check the box for “Initialize this repo w/README”

Further reading

[Workflow and Github](https://newmedia.report/classes/coding/2016/workflow-and-github/)  
  
Lesson from Jeremy Rue’s [Advanced Coding Interactives](https://newmedia.report/classes/coding/2016/) class.

[Getting Started with GitHub Desktop](https://help.github.com/desktop/guides/getting-started/)

[Getting Started with GitHub Pages](https://guides.github.com/features/pages/)  
This explains how you can creates web pages automatically from GitHub. However, I recommend authoring them locally, as we covered in class.

[Git Reference Manual](https://git-scm.com/doc)

[Basic Git Bash Commands](https://confluence.atlassian.com/bitbucketserver/basic-git-commands-776639767.html)

Week 2 Homework Assignment

1. (Ungraded) Reread these notes and try copying, pasting, and running the code provided to create a scatterplot in gglpot2.
2. (Worth 10 points) Follow the notes (and videos) on Week 2 Notes to learn about Github. Set up your own Github account. Send me the url for your GitHub account.
3. (Worth 10 points) Follow the [Week 2 Homework Tutorial](http://rpubs.com/rsaidi/518422)  (<http://rpubs.com/rsaidi/518422>). In your own new Markdown file, copy the code to create the **four plots**. (Plot 1, Plot 2, Plot 3, Plot 4). Knit the markdown and publish it in Rpubs, then post the Rpubs link in the Assignment Dropbox. Optional: feel free to make any changes in the plots to make them slightly different in some way from my tutorial code.

Submit the two assignments via the course Week 2 Assignment Dropbox by **11:59 pm next Monday, Sept. 16th.**We will present/discuss your submissions during the next class.