**Study Guide Unit 2 Name \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
DATA 110 Date \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
Saidi**

**Handling Data**

* [What is a tibble](#_Tibble_Data_Format)
* [Github basics](#_Using_GitHub)
* [What is data](#what-is-data?)
* [Where to get data](#_Where_to_get)
* [R Basics](#_R_Basics_(an) for handling data
* [Web Scraping](#_R_for_Web)
* [Homework Week 2](#_Homework_Week_2)

# **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"*

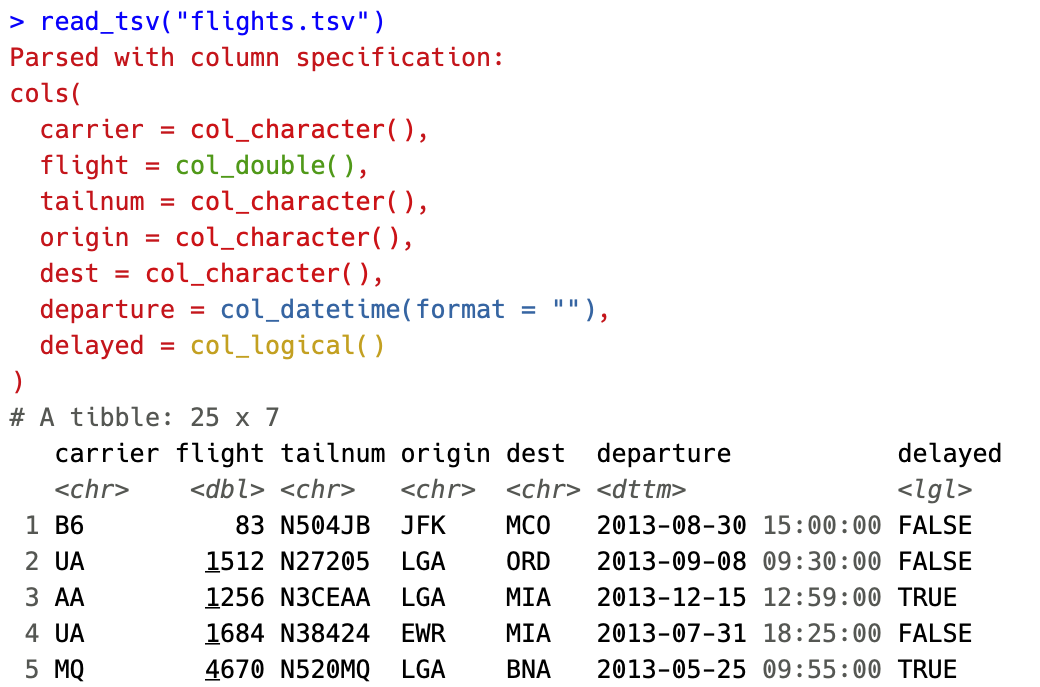
class(data[])

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

# **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.

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>)

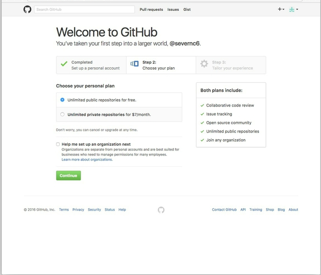
[Here](https://www.youtube.com/watch?v=l40x1EshOBE) is a great link to GitHub introductory training.

Create and secure your GitHub account

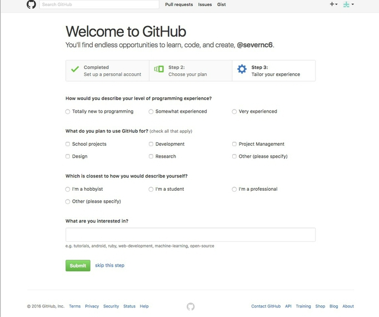
Navigate to [GitHub](https://github.com/) and sign up:



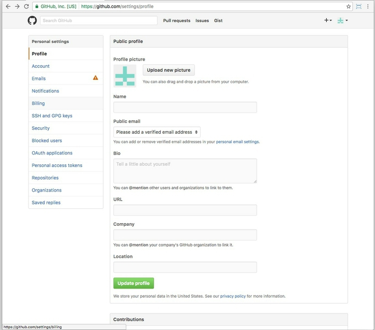
Choose your plan. If you want to be able to create private repositories, which cannot be viewed by others on the web, you will need to upgrade to a paid account. But for now select a free account and click Continue:



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



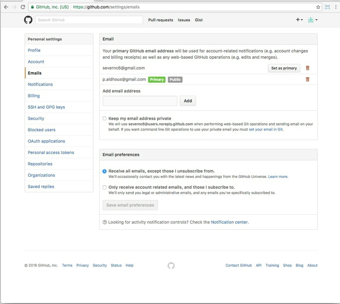
Now view your profile by clicking on the icon at top right and selecting **Your profile**. This is your page on GitHub. Click Edit profile to see the following:

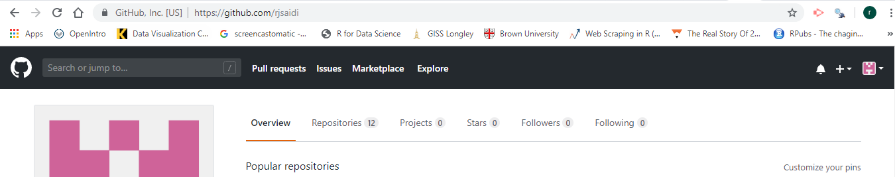


Here you can add your personal details, and a profile picture. For now just add the name you want to display on GitHub. Fill in the rest in your own time after class.

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.

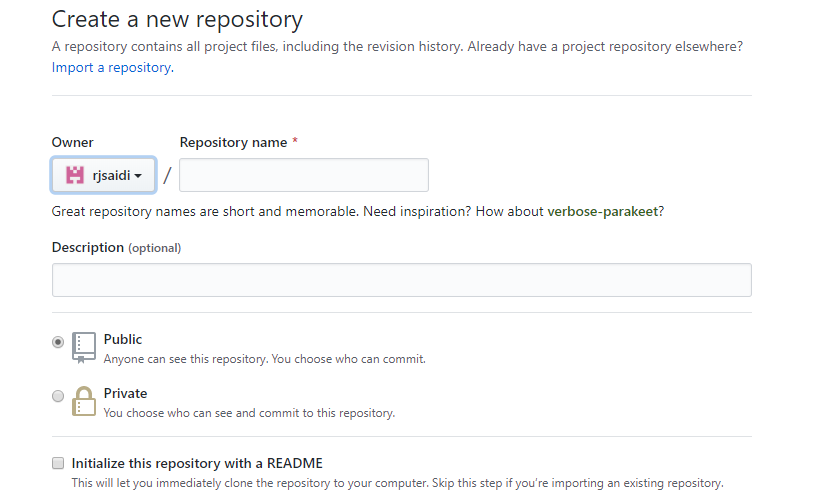
Back on the GitHub website, click on the **Emails** link in the panel at left. If you wish, you can add another email to use on GitHub, which will need to be verified as well. If you don’t wish to display your email on GitHub check the **Keep my email address private** box.





## 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 Github 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 mentioned in these notes.

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

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

# **What is data?**

This is a class in data visualization. Before we leap into making charts and maps, we will consider the nature of data, and some basic principles that will help you to “interview” datasets to find and tell stories. We will review some fundamental statistical concept as well.

Data visualization and statistics provide a view of the world that we cannot otherwise obtain. They give us a framework to make sense of daunting and otherwise meaningless masses of information. The “lies” that data and graphics can tell arise when people misuse statistics and visualization methods, not when they are used correctly.

The best data journalists understand that statistics and graphics go hand-in-hand. Just as numbers can be made to lie, graphics may misinform if the designer is ignorant of or abuses basic statistical principles. You do not have to be an expert statistician to make effective charts and maps, but understanding some basic principles will help you to tell a convincing and compelling story — enlightening rather than misleading your audience.

I hope you will get hooked on the power of a statistical way of thinking. As data artist [Martin Wattenberg](http://paldhous.github.io/www.bewitched.com/) of Google has said: [“Visualization is a gateway drug to statistics.”](http://www.newscientist.com/blogs/culturelab/2011/02/data-artists-visualisation-as-a-gateway-drug.html) Source: Peter Aldhous

## **Where to get data?** Here are some potential sources:

1. <https://ourworldindata.org/>
2. Open Data Network Through Socrata:  <https://dev.socrata.com/data/>
3. <https://www.kaggle.com/datasets>
4. <https://www.cdc.gov/>
5. <https://www.data.gov/>
6. [http://data.un.org](http://data.un.org/)
7. [World Bank Database](https://data.worldbank.org/indicator)  (https://data.worldbank.org/indicator)
8. <https://www.ipums.org/>
9. <https://knoema.com/atlas>
10. Open Data from Montgomery County: <https://data.montgomerycountymd.gov/>
11. https://www.icpsr.umich.edu/icpsrweb/content/about/thematic-collections.html
12. [R Datasets](http://vincentarelbundock.github.io/Rdatasets/datasets.html)   (You can browse those sets with this guide:  [R Data Guide](https://bb-montgomerycollege.blackboard.com/bbcswebdav/pid-4295118-dt-content-rid-25785804_1/xid-25785804_1))
13. WHO data: <http://www.who.int/>
14. Baltimore City has made its government data open: <https://data.baltimorecity.gov/>

And many more sites….. Some sites are better than others for accessing data on geography, sports, government and politics.

## Set your working directory

Now we can set the working directory to this folder by selecting from the top menu Session>Set Working Directory>To Source File Location. (Doing so means we can load the files in this directory without having to refer to the full path for their location, and anything we save will be written to this folder.)

Notice how this code appears in the console:

setwd("~/Desktop/week2notes")

### Save your data

The panel at top right has two tabs, the first showing the Environment, or all of the “objects” loaded into memory for this R session. We can save this as well, so we don’t have to load and process data again if we return to return to a project later.

(The second tab shows the History of the operations you have performed in RStudio.)

### Comment your code

Anything that appears on a line after # will be treated as a comment, and will be ignored when the code is run. Get into the habit of commenting your code: Don’t trust yourself to remember what it does!

## Some R code basics

* <- is known as an “assignment operator.” It means: “Make the object named to the left equal to the output of the code to the right.”
* & means AND, in Boolean logic
* | means OR, in Boolean logic.
* ! means NOT, in Boolean logic.
* When referring to values entered as text, or to dates, put them in quote marks, like this: "United States", or "2016-07-26". Numbers are not quoted.
* When entering two or more values as a list, combine them using the function c, with the values separated by commas, for example: c("2016-07-26","2016-08-04")
* As in a spreadsheet, you can specify a range of values with a colon, for example: c(1:10) creates a list of integers (whole numbers) from one to ten.
* Some common operators:
  + + - add, subtract.
  + \* / multiply, divide.
  + > < greater than, less than.
  + >= <= greater than or equal to, less than or equal to.
  + != not equal to.
* Equals signs can be a little confusing, but see how they are used in the code we use today:
  + == test whether an object is equal to a value. This is often used when filtering data, as we will see.
  + = make an object equal to a value; works like <-, but used within the brackets of a function.

**Important:** Object and variable names in R should not contain spaces.

## Install and load R packages

Much of the power of R comes from the thousands of “packages” written by its community of open source contributors. These are optimized for specific statistical, graphical or data-processing tasks. To see what packages are available in the basic distribution of R, select the Packages tab in the panel at bottom right. To find packages for particular tasks, try searching Google using appropriate keywords and the phrase “R package.”

In this class, we will work with two incredibly useful packages developed by [Hadley Wickham](http://hadley.nz/), chief scientist at RStudio:

* [**readr**](https://cran.r-project.org/web/packages/readr/readr.pdf) For reading and writes CSV and other text files.
* [**dplyr**](https://cran.r-project.org/web/packages/dplyr/dplyr.pdf) For processing and manipulating data.

These and several other useful packages have been combined into a super-package called [**tidyverse**](http://insight.livestories.com/).

To install a package, click on the Install icon in the Packages tab, type its name into the dialog box, and make sure that Install dependencies is checked, as some packages will only run correctly if other packages are also installed. Click Install and all of the required packages should install:

Notice that the following code appears in the console:

install.packages("tidyverse")

So you can also install packages with code in this format, without using the point-and-click interface.

Each time you start R, it’s a good idea to click on Update in the Packages panel to update all your installed packages to the latest versions.

Installing a package makes it available to you, but to use it in any R session you need to load it. You can do this by checking its box in the Packages tab. However, we will enter the following code into our script, then highlight these lines of code and run them:

# load package to read, write and manipulate data

library(tidyverse)

At this point, and at regular intervals, save your script, by clicking the save/disk icon in the script panel, or using the ⌘-S keyboard shortcut.

# **R Basics (an introduction to R from Peter Aldhous)**

### The data we will use today ([found in this google drive](https://drive.google.com/drive/u/0/folders/1AMRfddeMwKRaNidOV87JP1iCVn_z-Uv_))

Download the data for this session and save it to your datasets folder. It contains the following files, used in reporting [this story](https://www.newscientist.com/article/dn18806-revealed-pfizers-payments-to-censured-doctors/), which revealed that some of the doctors paid as “experts” by the drug company Pfizer had troubling disciplinary records:

* pfizer.csv Payments made by Pfizer to doctors across the United States in the second half on 2009. Contains the following variables:
  + org\_indiv Full name of the doctor, or their organization.
  + first\_plus Doctor’s first and middle names.
  + first\_name last\_name. First and last names.
  + city state City and state.
  + category of payment Type of payment, which include Expert-led Forums, in which doctors lecture their peers on using Pfizer’s drugs, and `Professional Advising.
  + cash Value of payments made in cash.
  + other Value of payments made in-kind, for example puschase of meals.
  + total value of payment, whether cash or in-kind.
* fda.csv Data on warning letters sent to doctors by the U.S. Food and Drug Administration, because of problems in the way in which they ran clinical trials testing experimental treatments. Contains the following variables:
  + name\_last name\_first name\_middle Doctor’s last, first, and middle names.
  + issued Date letter was sent.
  + office Office within the FDA that sent the letter.

## Load and view data

#### Load data

You can load data into the current R session by selecting Import Dataset>From Text File... in the Environment tab.

However, we will use the read\_csv function. Copy the following code into your script and Run:

# load data of pfizer payments to doctors and warning letters sent by food and drug administration

pfizer <- read\_csv("pfizer.csv")

fda <- read\_csv("fda.csv")

Notice that the Environment now contains two objects, of the type tbl\_df, a variety of the standard R object for holding tables of data, known as a **data frame**:

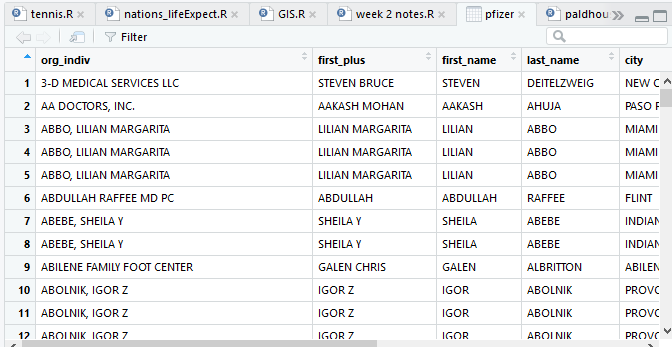
The Value for each data frame details the number of columns, and the number of rows, or observations, in the data.

You can remove any object from your environment by checking it in the Grid view and clicking the broom icon.

#### Examine the data

We can View data at any time by clicking on its table icon in the Environment tab in the Grid view.

Here, for example, is the pfizer view:



The str function will tell you more about the columns in your data, including their data type. Copy this code into your script and Run:

# view structure of data

str(pfizer)

This should give the following output in the R Console:

Classes ‘tbl\_df’, ‘tbl’ and 'data.frame': 10087 obs. of 10 variables:

$ org\_indiv : chr "3-D MEDICAL SERVICES LLC" "AA DOCTORS, INC." "ABBO, LILIAN MARGARITA" "ABBO, LILIAN MARGARITA" ...

$ first\_plus: chr "STEVEN BRUCE" "AAKASH MOHAN" "LILIAN MARGARITA" "LILIAN MARGARITA" ...

$ first\_name: chr "STEVEN" "AAKASH" "LILIAN" "LILIAN" ...

$ last\_name : chr "DEITELZWEIG" "AHUJA" "ABBO" "ABBO" ...

$ city : chr "NEW ORLEANS" "PASO ROBLES" "MIAMI" "MIAMI" ...

$ state : chr "LA" "CA" "FL" "FL" ...

$ category : chr "Professional Advising" "Expert-Led Forums" "Business Related Travel" "Meals" ...

$ cash : int 2625 1000 0 0 1800 750 0 825 3000 0 ...

$ other : int 0 0 448 119 0 0 47 0 0 396 ...

$ total : int 2625 1000 448 119 1800 750 47 825 3000 396 ...

chr means “character,” or a string of text (which can be treated as a categorical variable); int means an integer, or whole number.

Also examine the structure of the fda data frame using the following code:

str(fda)

This should be the console output:

Classes ‘tbl\_df’, ‘tbl’ and 'data.frame': 272 obs. of 5 variables:

$ name\_last : chr "ADELGLASS" "ADKINSON" "ALLEN" "AMSTERDAM" ...

$ name\_first : chr "JEFFREY" "N." "MARK" "DANIEL" ...

$ name\_middle: chr "M." "FRANKLIN" "S." NA ...

$ issued : Date, format: "1999-05-25" ...

$ office : chr "Center for Drug Evaluation and Research" "Center for Biologics Evaluation and Research" "Center for Devices and Radiological Health" "Center for Biologics Evaluation and Research" ...

Notice that issued has been recognized as a Date variable. Other common data types include num, for numbers that may contain decimals and POSIXct for full date and time.

If you run into any trouble importing data with **readr**, you may need to specify the data types for some columns — in particular for date and time. [This link](https://r4ds.had.co.nz/data-import.html) from Hadley Wickham’s Chapter 11 on Data Importing explains how to set data types for individual variables when importing data with **readr**.

To specify an individual column use the name of the data frame and the column name, separated by $. Type this into your script and run:

# print values for total in pfizer data

pfizer$total

The output will be the first 10,000 values for that column.

If you need to change the data type for any column, use the following functions:

* as.character converts to a text string.
* as.numeric converts to a number.
* as.factor converts to a categorical variable.
* as.integer converts to an integer
* as.Date converts to a date
* as.POSIXct convets to a full date and time.

(Conversions to full dates and times can get complicated, because of timezones.

Now add the following code to your script to convert total in the pfizer data to a numeric variable (which would allow it to hold decimal values, if we had any).

# convert total to numeric variable

pfizer$total <- as.numeric(pfizer$total)

str(pfizer)

Notice that the data type for total has now changed:

Classes ‘tbl\_df’, ‘tbl’ and 'data.frame': 10087 obs. of 10 variables:

$ org\_indiv : chr "3-D MEDICAL SERVICES LLC" "AA DOCTORS, INC." "ABBO, LILIAN MARGARITA" "ABBO, LILIAN MARGARITA" ...

$ first\_plus: chr "STEVEN BRUCE" "AAKASH MOHAN" "LILIAN MARGARITA" "LILIAN MARGARITA" ...

$ first\_name: chr "STEVEN" "AAKASH" "LILIAN" "LILIAN" ...

$ last\_name : chr "DEITELZWEIG" "AHUJA" "ABBO" "ABBO" ...

$ city : chr "NEW ORLEANS" "PASO ROBLES" "MIAMI" "MIAMI" ...

$ state : chr "LA" "CA" "FL" "FL" ...

$ category : chr "Professional Advising" "Expert-Led Forums" "Business Related Travel" "Meals" ...

$ cash : int 2625 1000 0 0 1800 750 0 825 3000 0 ...

$ other : int 0 0 448 119 0 0 47 0 0 396 ...

$ total : num 2625 1000 448 119 1800 ...

The summary function will run a quick statistical summary of a data frame, calculating mean, median and quartile values for continuous variables:

# summary of pfizer data

summary(pfizer)

Here is the last part of the console output:

total

Min. : 0

1st Qu.: 191

Median : 750

Mean : 3507

3rd Qu.: 2000

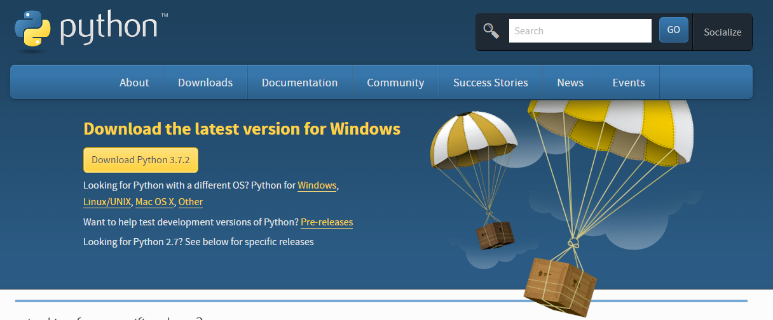
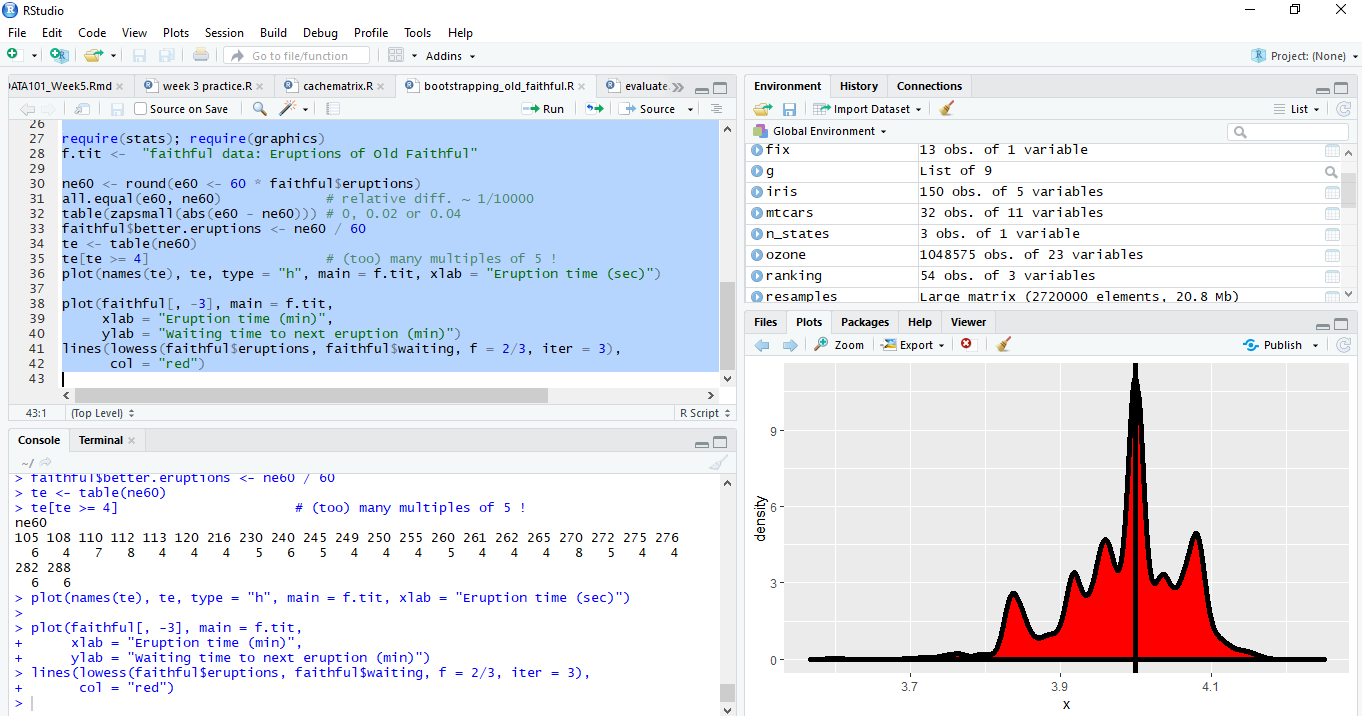
Max. :1185466

This is where we will end with the introduction to R. In the next section, you will be using software to gather data from a website and explore that data in R.

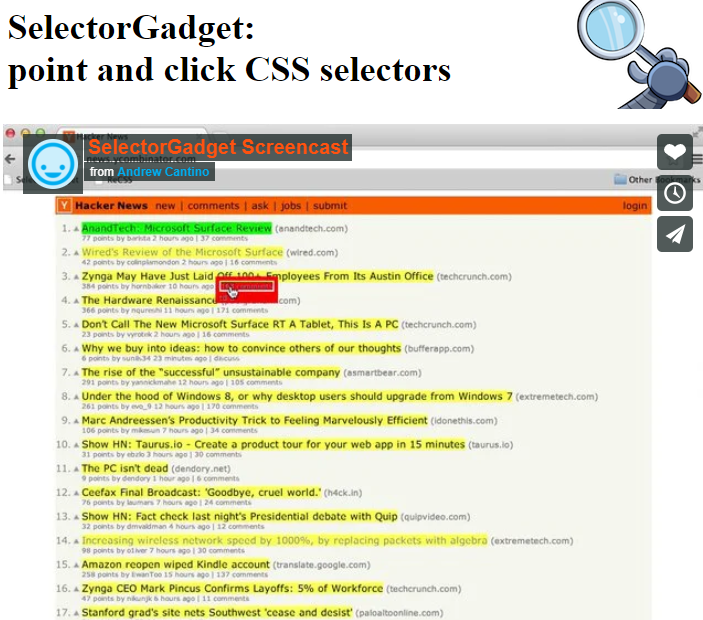
# **R for Web Scraping**

Unless you are using Python, you can skip pages 30-37 in your text, as they take you through Web Scraping with Python. Instead, we will use Web Scraping with R with a Chrome extension called SelectorGadget. See below.

Web Scraping can be extremely useful (and fun) to extract information from a web page that uses html and css coding to extract that page’s information and turn it in to a dataset.

1. Click on this link to follow how to scrape the web for html information [Beginner’s Guide on Web Scraping in R (using rvest) with hands-on example](https://www.analyticsvidhya.com/blog/2017/03/beginners-guide-on-web-scraping-in-r-using-rvest-with-hands-on-knowledge/) You will install “rvest” on R and download the Chrome extension from SelectorGadget



Watch this video on how to use SelectorGadget.

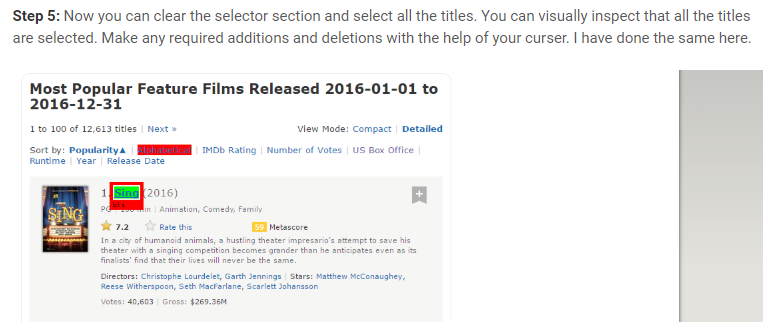
Be sure that once you have loaded the Chrome extension, you see this symbol on the top right of your Chrome page

By the way, HTML, HyperText Markup Language, gives content structure and meaning by defining that content as, for example, headings, paragraphs, or images. CSS, or Cascading Style Sheets, is a presentation language created to style the appearance of content—using, for example, fonts or colors.

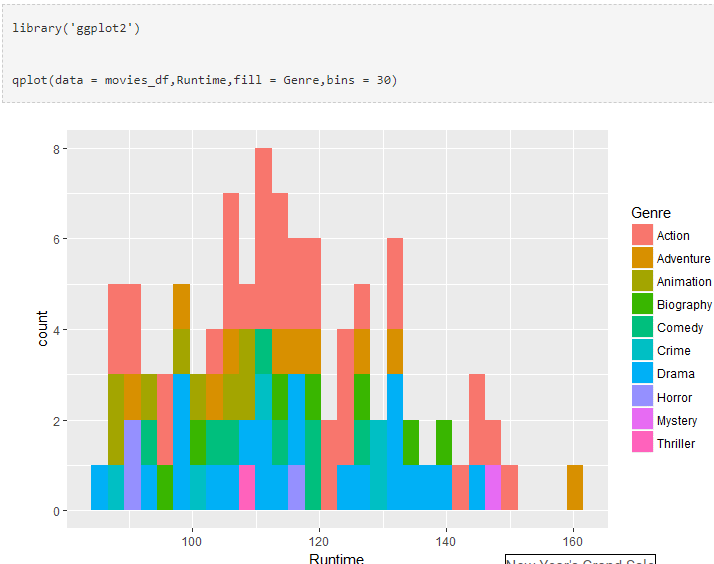
The two languages—HTML and CSS—are independent of one another. As a rule, HTML will always represent content, and CSS will always represent the appearance of that content.

1. Follow step 4 for Web Scraping the movie rankings

(Believe me – this is fun, AND it works!!!!)

1. Follow steps 5 and 6 for web scraping for movie titles.

Have fun following steps 7-11 for Description, Runtime, Genre, Rating, Metascore, Votes, Gross\_Earning\_in\_Mil , Director and Actor data. Try to complete all sections so that you can create the data frame to then analyze the data and create data visualizations.



# Homework Week 2

1. (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 – **due at 11:59 pm on Wednesday, June 17th.** Optional: feel free to make any changes in the plots to make them slightly different in some way from my tutorial code.
2. (Ungraded) Explore the links at the beginning of the notes for potential sources of data.
3. (Ungraded) Reread these notes and try copying, pasting, and running the code provided to create a scatterplot in gglpot2.
4. (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.
5. (Worth up to 10 points) Web Scraping Lab Assignment  
   Click on  [Beginner’s Guide on Web Scraping in R (using rvest) with hands-on example](https://www.analyticsvidhya.com/blog/2017/03/beginners-guide-on-web-scraping-in-r-using-rvest-with-hands-on-knowledge/)

(<https://www.analyticsvidhya.com/blog/2017/03/beginners-guide-on-web-scraping-in-r-using-rvest-with-hands-on-knowledge/>)

1. Follow all steps on this site, copying and pasting the code into your own markdown file.
2. Answer the 3 questions based on those 3 graphs.
3. Once you have tested your code and it works, knit and publish your work to Rpubs.

This code will correct the problem with gross\_data:

#Using CSS selectors to scrap the gross revenue section  
gross\_data\_html <- html\_nodes(webpage,'.ghost~ .text-muted+ span')  
#Converting the gross revenue data to text  
gross\_data <- html\_text(gross\_data\_html)  
#Let's have a look at the gross data  
head(gross\_data)  
#Data-Preprocessing: removing '$' and 'M' signs  
gross\_data<-gsub("[^0-9]\*","",gross\_data)  
head(gross\_data)  
#Let's check the length of gross data  
length(gross\_data)  
#Filling missing entries with NA  
for (i in c(29,33,40,41,43,72,74,75,76,100)){     
a<-gross\_data[1:(i-1)]     
b<-gross\_data[i:length(gross\_data)]      
gross\_data<-append(a,list("NA"))      
gross\_data<-append(gross\_data,b)

#Data-Preprocessing: converting gross to numerical  
unlist(gross\_data)  
gross\_data <- gross\_data[-c(101,102)]  
gross\_data<-as.numeric(gross\_data)  
#Let's have another look at the length of gross data  
length(gross\_data)  
summary(gross\_data)

Be sure to get this lab done all on the same day. Otherwise your code will fail when you return to it at a later date.

1. Follow all steps on this site, copying and pasting the code into your Markdown file.
2. Once you have tested your code and it works, aswer the 3 questions based on those 3 graphs
3. Publish your rmd file.

Submit items 4, 5, and 6 assignments via the course Week 2 Assignment Dropbox by **11:59 pm, June 21st.**We will present/discuss your submissions during the next class.