Study Guide Unit 2
DATA 110
Saidi

- What is a tibble
- What is piping
- Github basics
- Data Visualization principles (TED Talk – Lies, Damned Lies, and Statistics)
- Using color effectively and ColorBrewer
- What is data
- Where to get data
- 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**(), the modern **data_frame**():

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

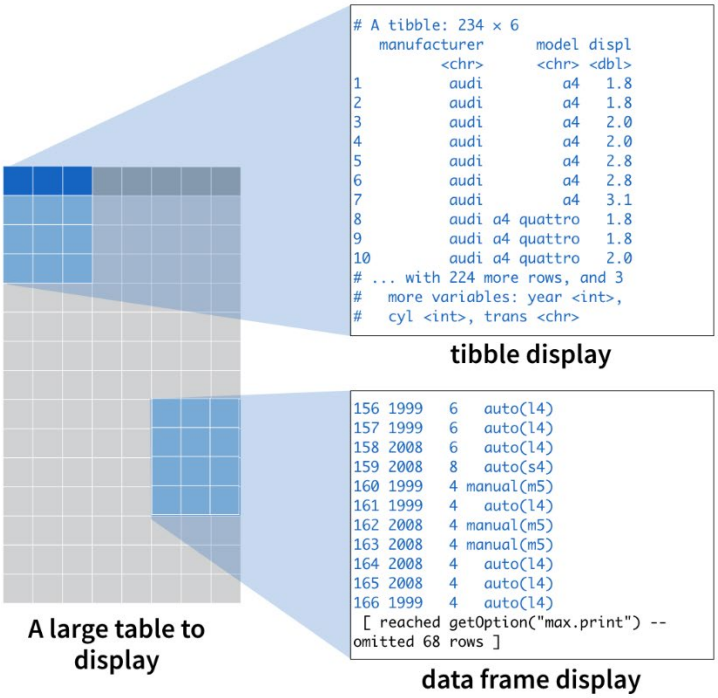
Tibbles are a modern take on data frames. They keep the features that have stood the test of time, and drop the features that used to be convenient but are now frustrating (i.e. converting character vectors to factors).

Advantages of tibbles compared to data frames

1. A tibble is a special type of table. R displays tibbles in a refined way whenever you have the **tibble** package loaded: R will print only the first ten rows of a tibble as well as all of the columns that fit into your console window. R also adds useful summary information about the tibble, such as the data types of each column and the size of the data set.

   Whenever you do not have the tibble packages loaded, R will display the tibble as if it were a data frame. In fact, tibbles *are* data frames, an enhanced type of data frame.

   You can think of the difference between the data frame display and the tibble display like this:



2. **as_tibble()**

You can transform a data frame to a tibble with the as_tibble() function in the tibble package, e.g. as_tibble(cars).

3. When printed, the data type of each column is specified (see below):
   a. : for double
   b. : for factor
   c. : for character
   d. : 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.

Color 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, it might be easier to spot when a column differs from the rest or when guessing leads to import with an unexpected type.

```
> read_tsv("flights.tsv")
Parsed with column specification:
cols(
  carrier = col_character(),
  flight = col_double(),
  tailnum = col_character(),
  origin = col_character(),
  dest = col_character(),
  departure = col_datetime(format = ""),
  delayed = col_logical()
)
```

What is Piping Anyway?

The Pipe Operator in R: Introduction (From Data Camp)

To understand what the pipe operator in R is and what you can do with it, it's necessary to learn the history behind it. Where does this weird combination of symbols come from? And why was it made like this?

Now, you can look at the history from three perspectives: from a mathematical point of view, from a holistic point of view of programming languages, and from the point of view of the R language itself. You'll cover all three in what follows!

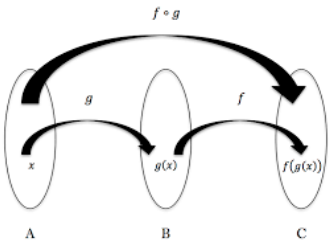
History of the Pipe Operator in R

Mathematical History

If you have two functions, $f: B \rightarrow C$ and $g: A \rightarrow B$ you can chain these functions together by taking the output of one function and inserting it into the next. In short, "chaining" means that you pass an intermediate result onto the next function.

For example, you can say, $f(g(x))$: $g(x)$ serves as an input for $f()$, while x serves as input to $g()$. This is often called a composite function, where f is ***composed*** of g.

If you would want to note this down, you will use the notation $f \circ g$, which reads as "f follows g", or “f of g”. Alternatively, you can visually represent this as:



Pipes in R

The history of this operator in R starts in 2012 when Hadley Wickham started the `dplyr` package on GitHub, which is based off of F# (pronounced F Sharp, as in Visual F# Programming Language, which is an open source, cross platform compiler, which can generate JavaScript and graphics processing unit (GPU) code. The question started:

How can you implement F#'s forward pipe operator in R? The operator makes it possible to easily chain a sequence of calculations. For example, when you have an input data and want to call functions `foo` and `bar` in sequence, you can write `data |> foo |> bar`?

```
pi |> sin   # start the value pi, and then put i tinto the function sin
[1] 1.224606e-16
pi |> sin |> cos
[1] 1
cos(sin(pi))
[1] 1
```

What is a Pipe?

In R, the pipe operator is, as you have already seen, `%>%` or `|>`. You can think of this operator as being similar to the `+` in a `ggplot2` statement. It takes the output of one statement and makes it the input of the next statement. When describing it, you can think of it as a "THEN".

This is one of the most powerful things about the Tidyverse. In fact, having a standardized chain of processing actions is called "a pipeline". Making pipelines for a data format is great, because you can apply that pipeline to incoming data that has the same formatting and have it output in a `ggplot2` friendly format, for example.

Why Use It?

R is a functional language, which means that your code often contains a lot of parenthesis, `(` and `)`. When you have complex code, this often will mean that you will have to nest those parentheses together. This makes your R code hard to read and understand. Here's where `|>` comes in to the rescue!

Take a look at the following example, which is a typical example of nested code:

```
# Initialize `x`
x <- c(0.109, 0.359, 0.63, 0.996, 0.515, 0.142, 0.017, 0.829, 0.907)
#Compute the logarithm of `x`, return suitably logged and iterated differences,
compute the exponential function and round the result
round(exp(diff(log(x))), 1)
```

With the help of `|>`, you can rewrite the above code as follows:

```
# Import `magrittr`
library(magrittr)
# Perform the same computations on `x` as above
x |>log() |>
  diff() |>
  exp() |>
  round(1)
```

How to Use Pipes in R

Now you know how the `%>%` or `|>`operator originated, what it actually is and why you should use it. It is time for you to discover how you can actually use it to your advantage. You will see that there are quite some ways in which you can use it!

Basic Piping

Before you go into the more advanced usages of the operator, it's good to first take a look at the most basic examples that use the operator. In essence, you'll see that there are 3 rules that you can follow when you're first starting out:

`f(x)` can be rewritten as `x |> f`

In short, this means that functions that take one argument, `function(argument)`, can be rewritten as follows: `argument |> function()`.

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 lesson in Jeremy Rue's Advanced Coding Interactives class and Coursera's Data Science Toolbox — see the further reading links below.

Introducing Git, GitHub and GitHub Desktop

The version control software we will use is called **Git**. It is installed automatically when you install and configure **GitHub Desktop**. GitHub Desktop allows you to manage version control for local versions of projects on your own computer, and sync them remotely with **GitHub**. 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.

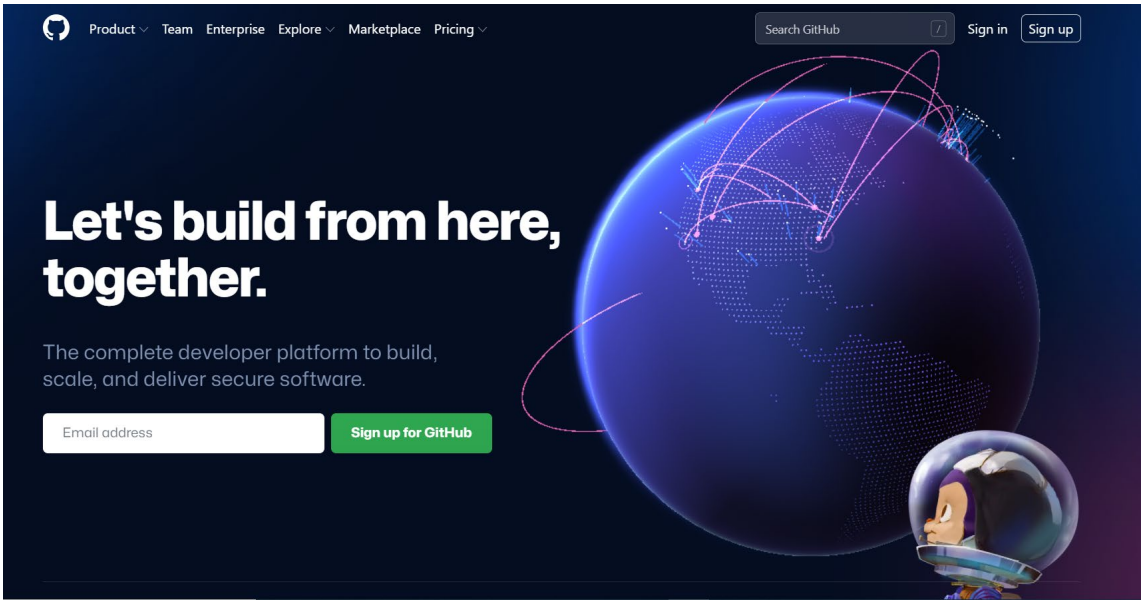
Here is a more extended GitHub glossary.

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

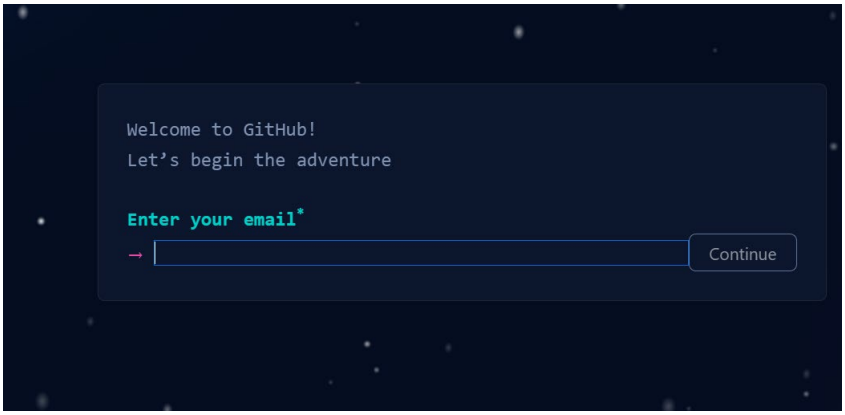
Here is a great link to GitHub introductory training.

Create and secure your GitHub account

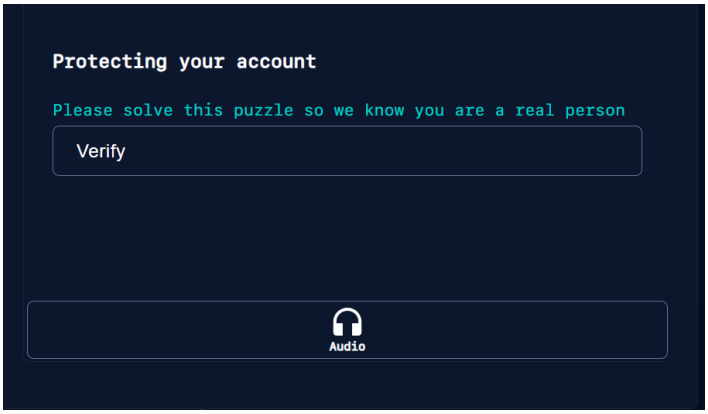
Navigate to GitHub and sign up:



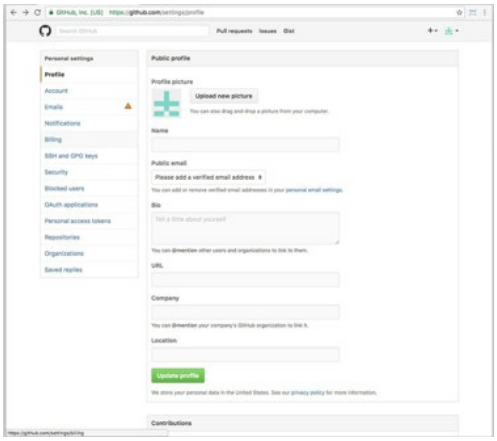
Choose your plan. If you want to be able to create private repositories, which cannot be viewed by others on the web, you will need to upgrade to a paid account. But for now select a free account and click Continue:



At the next screen, you will have to "solve a puzzle"



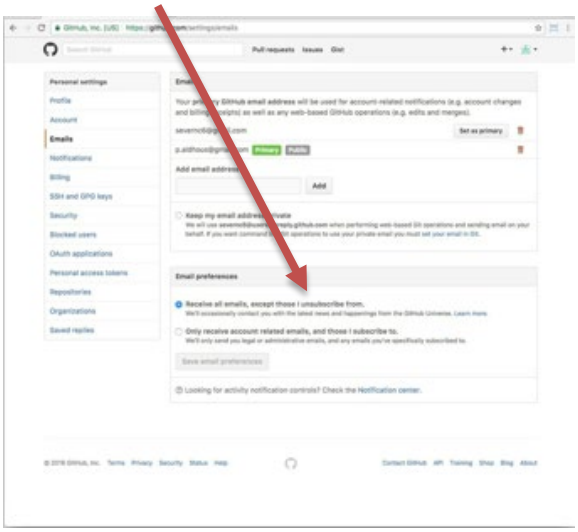
Now view your profile by clicking on the icon at top right and selecting **Your profile**. This is your page on GitHub. Click Edit profile to see the following:



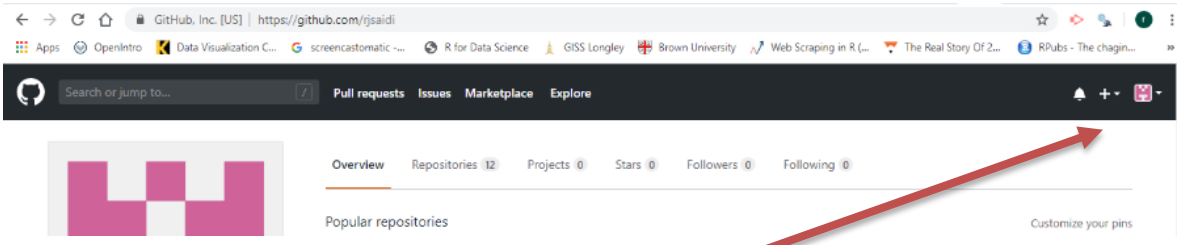
Here you can add your personal details, and a profile picture. For now just add the name you want to display on GitHub. Fill in the rest in your own time after class.

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.

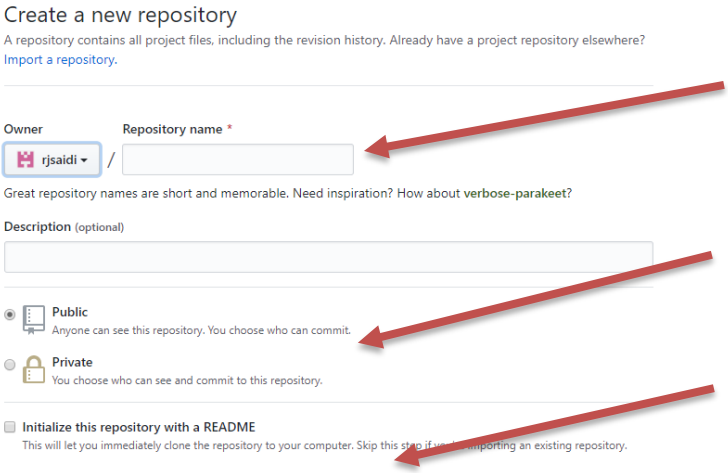


Creating a Github Repository



1. Start a repo from scratch
   a. Click "Create a New Repo" (click the plus sign at the top right) or
   b. Github.com/new

2. "Fork" another's repo
3. Request to "Pull" another's repo to make edits
4. "Push" a repo



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

Make it "Public"

Check the box for "Initialize this repo w/README"

Further Github reading

Workflow and Github

Lesson from Jeremy Rue's Advanced Coding Interactives class.

Getting Started with GitHub Desktop

Getting Started with GitHub 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

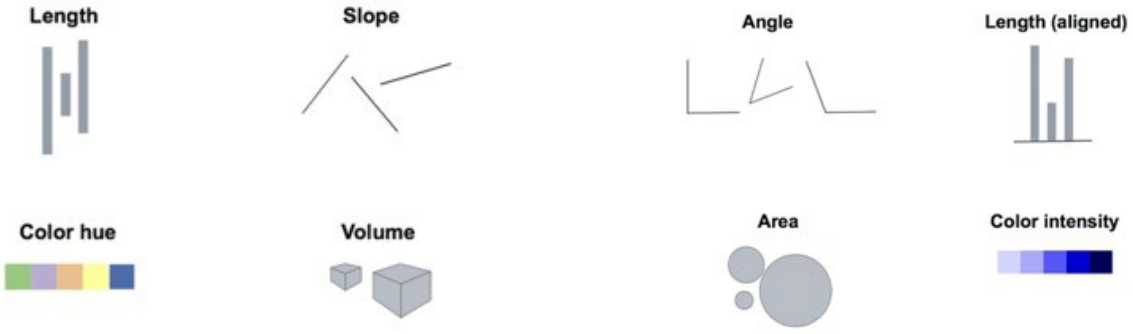
Basic Git Bash Commands

Visualization Tools/Patterns and Forms of Communication

Watch this TED TALK: Lies, Damned Lies, and Statistics (5:51)

Data visualization: basic principles (from Peter Aldhous)

Whenever we visualize, we are encoding data using visual cues, or "mapping" data onto variation in size, shape or color, and so on. There are various ways of doing this, as this primer illustrates:



Visual cues are graphical elements that draw the eye to what you want your audience to focus upon. They are the fundamental building blocks of data graphics, and the choice of which visual cues to use to represent which quantities is the central question for the data graphic composer. Yau identifies nine distinct visual cues, for which we also list whether that cue is used to encode a numerical or categorical quantity:

Position (numerical) where in relation to other things?

Length (numerical) how big (in one dimension)?

Angle (numerical) how wide? parallel to something else?

Direction (numerical) at what slope? In a time series, going up or down?

Shape (categorical) belonging to which group?

Area (numerical) how big (in two dimensions)?

Volume (numerical) how big (in three dimensions)?

Shade (either) to what extent? how severely?

Color (either) to what extent? how severely? Beware of red/green color blindness

These cues are not created equally, however. In the mid-1980s, statisticians William Cleveland and Robert McGill ran some experiments with human volunteers, measuring how accurately they were able to perceive the quantitative information encoded by different cues. This is what they found:

Visual cues are graphical elements that draw the eye to what you want your audience to focus upon. They are the fundamental building blocks of data graphics, and the choice of which visual cues to use to represent which quantities is the central question for the data graphic composer.

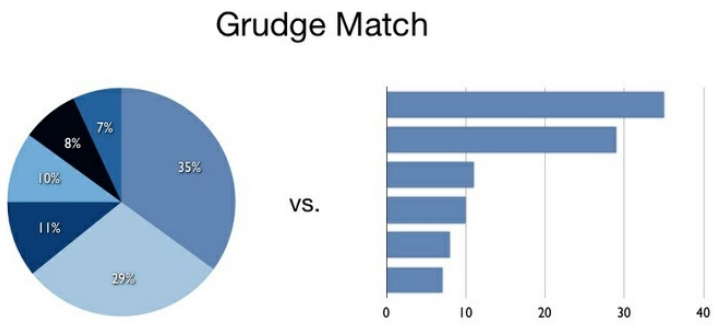
This perceptual hierarchy of visual cues is important. When making comparisons with continuous variables, aim to use cues near the top of the scale wherever possible.

But this doesn't mean that everything becomes a bar chart.

Length on an aligned scale may be the best option to allow people to compare numbers accurately, but that doesn't mean the other possibilities are always to be avoided in visualization. Indeed, color hue is a good way of encoding categorical data. The human brain is particularly good at recognizing patterns and differences. This means that variations in color, shape and orientation, while poor for accurately encoding the precise value of continuous variables, can be good choices for representing categorical data.

You can also combine different visual cues in the same graphic to encode different variables. But always think about the main messages you are trying to impart, and where you can use visual cues near the top of the visual hierarchy to communicate that message most effectively.

If you have spent any time reading blogs on data visualization, you will know the disdain in which pie charts are often held. It should be clear which of these two charts is easiest to read:



(Source: VizThinker)

Bar charts are almost always better than pie charts. People are not very good at assessing angles, and in recent years, data viz folks have really started to push back against the use of pie charts.

Small multiples and layers

One of the fundamental challenges of creating data graphics is condensing multivariate information into a two-dimensional image. While three-dimensional images are occasionally useful, they are often more confusing than anything else. Instead, here are three common ways of incorporating more variables into a two-dimensional data graphic:

Small multiples Also known as *facets*, a single data graphic can be composed of several small multiples of the same basic plot, with one (discrete) variable changing in each of the small sub-images.

Layers It is sometimes appropriate to draw a new layer on top of an existing data graphic. This new layer can provide context or comparison, but there is a limit to how many layers humans can reliably parse.

Animation If time is the additional variable, then an animation can sometimes effectively convey changes in that variable. Of course, this doesn't work on the printed page, and makes it impossible for the user to see all the data at once.

Color (either) to what extent? how severely? Beware of red/green color blindness

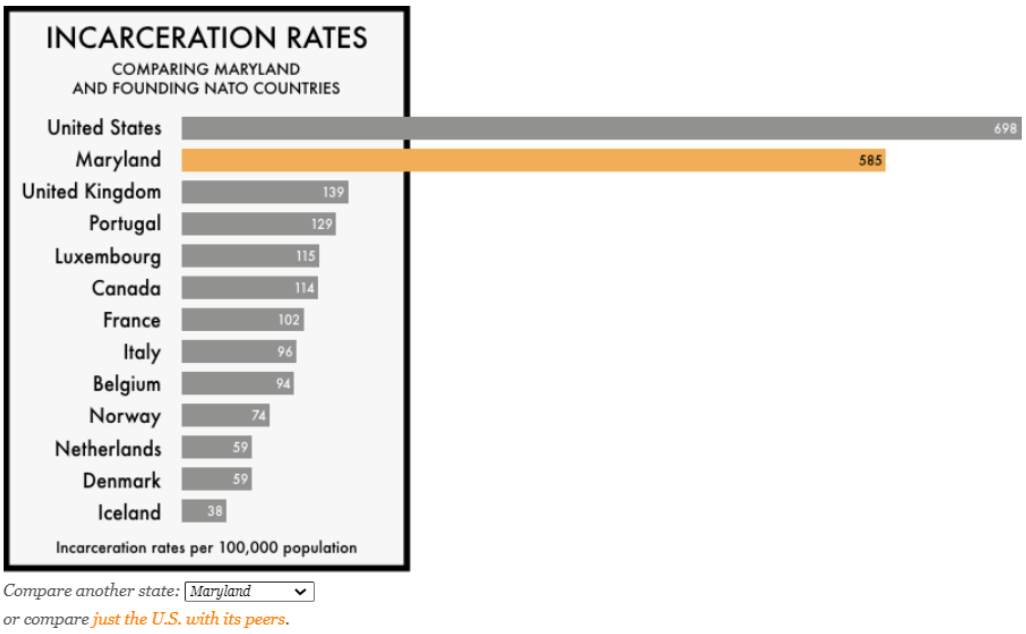
Which chart type should I use?

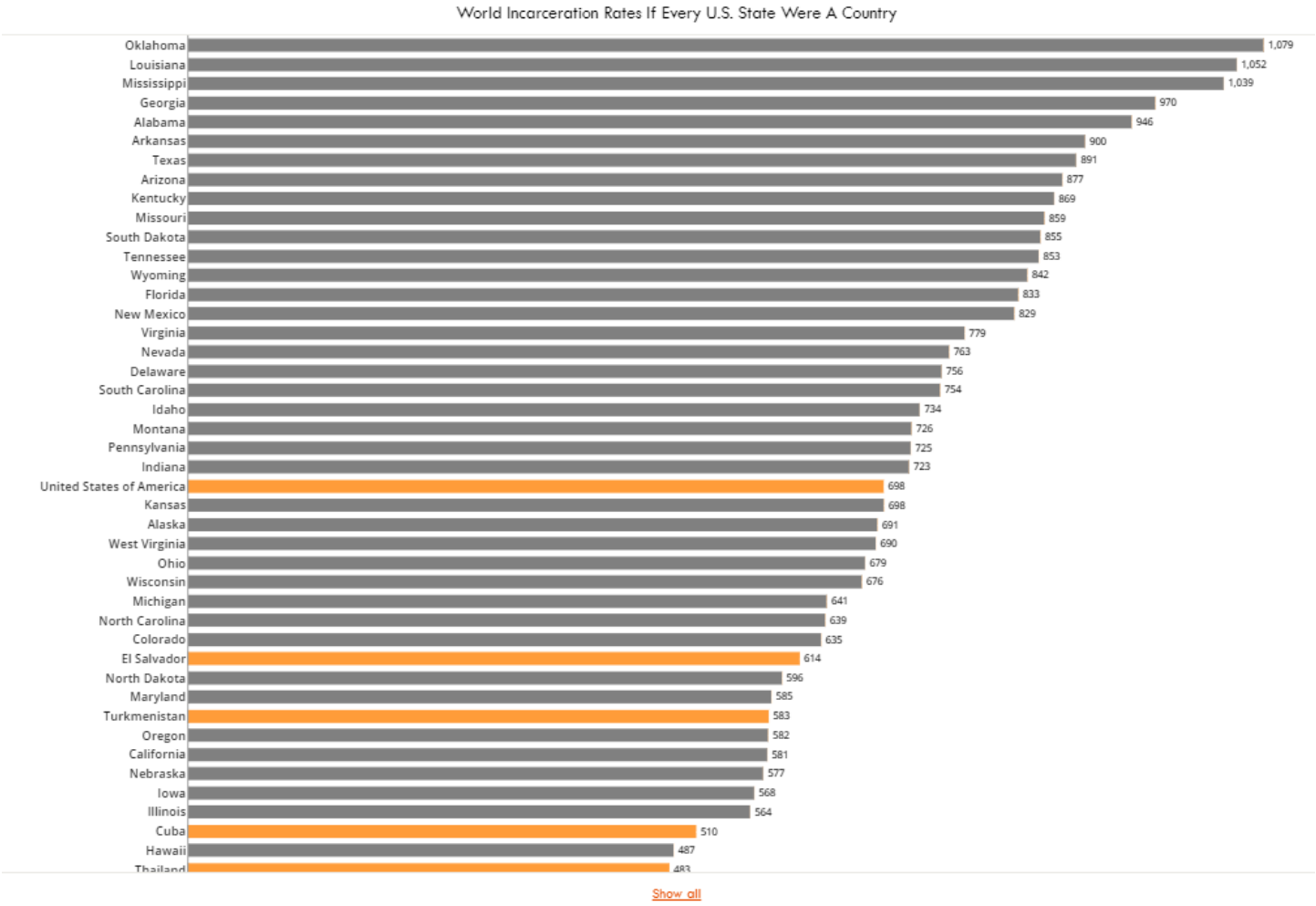
This is a frequently asked question, and the best answer is: Experiment with different charts, to see which works best **to liberate the story in your data.**

Simple comparisons: bars and columns

Below is a powerful bar and column comparison (use the link to see the entire visualization):

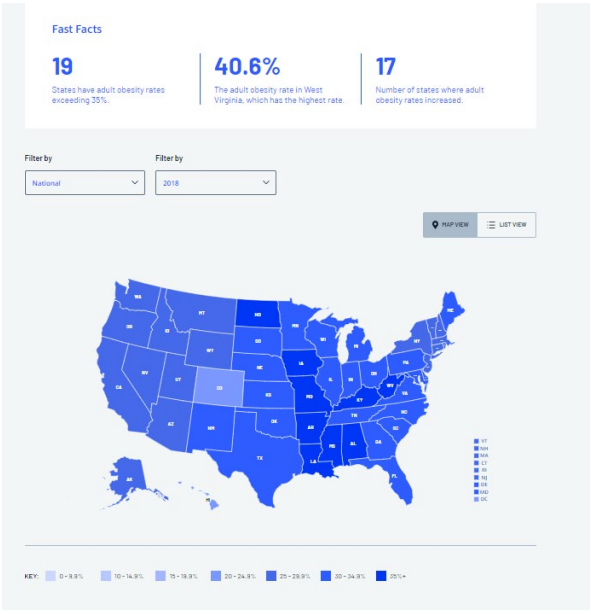
[World Incarceration Rates If Every U.S. State & Territory Were A Country]

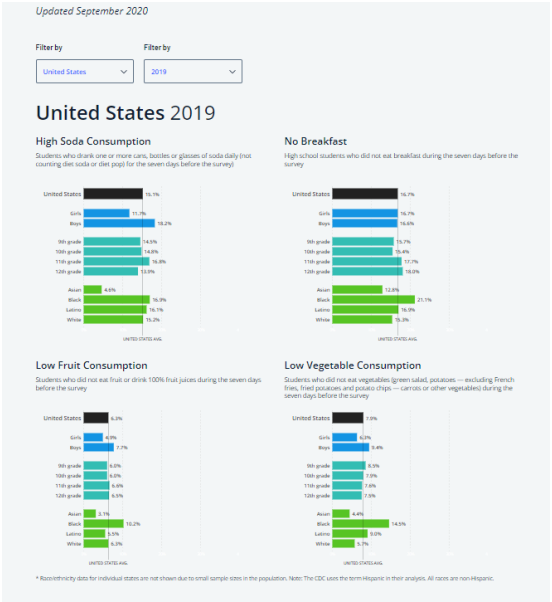




Comparisons over Time

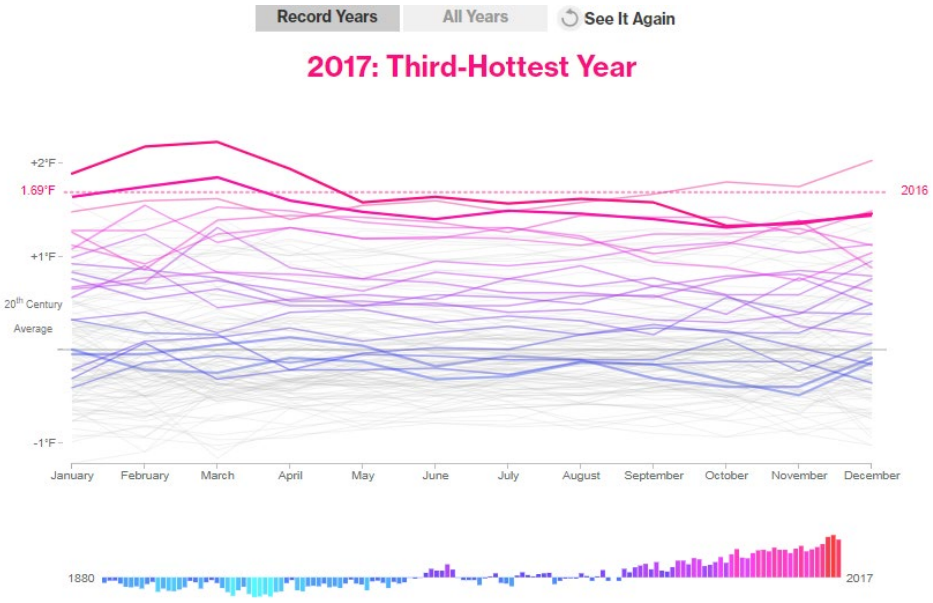
Watch obesity rates in the US change over time since the 1990's to 2017. Click Obesity Rates (https://stateofobesity.org/adult-obesity/).





Check this one out (pretty dismal, though)

[Earth's Relentless Warming Sets New Brutal Record in 2017]



If you prefer circles to spaghetti plots, try this link (even more alarming) [137 years of #climate anomalies visualized. Wait for it...] (Data Source NASA GISS)

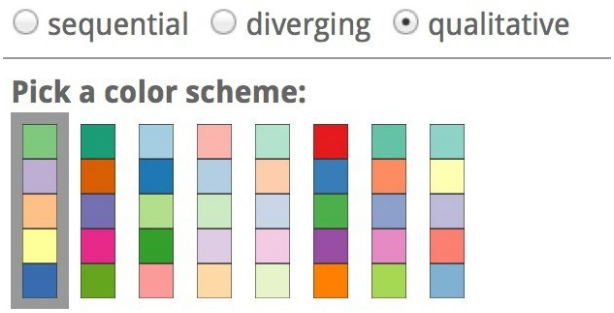
Use color effectively

Color falls low on the perceptual hierarchy of visual cues, but as we have seen above, it is often deployed to highlight particular elements of a chart, and sometimes to encode data values. Poor choice of color schemes is a problem that bedevils many news graphics, so it is worth taking some time to consider how to use color to maximum effect.

It helps to think about colors in terms of the color wheel, which places colors that "harmonize" well together side by side, and arranges those that have strong visual contrast — blue and orange, for instance — at opposite sides of the circle:

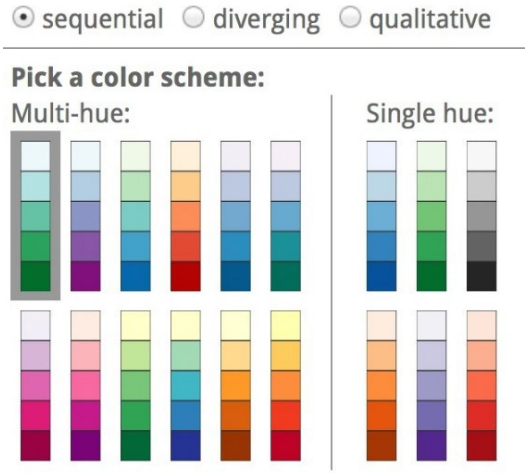
(Source: Wikimedia Commons)

When encoding data with color, take care to fit the color scheme to your data, and the story you are aiming to tell. Color may be used to encode the values of categorical data. Here you want to use "qualitative" color schemes, where the aim is to pick colors that will be maximally distinctive, as widely spread around the color wheel as possible:



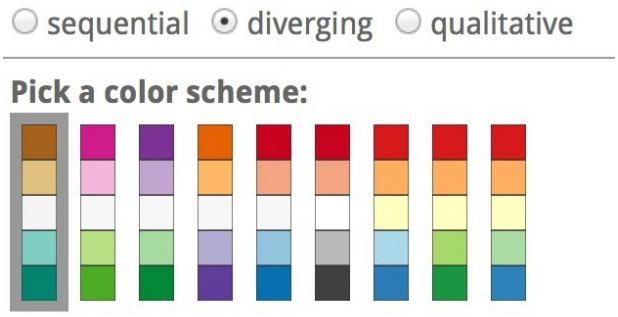
(Source: ColorBrewer)

When using color to encode continuous data, it usually makes sense to use increasing intensity, or saturation of color to indicate larger values. These are called “sequential” color schemes:



(Source: ColorBrewer)

In some circumstances, you may have data that has positive and negative values, or which highlights deviation from a central value. Here, you should use a “diverging” color scheme, which will usually have two colors reasonably well separated on the color wheel as its end points, and cycle through a neutral color in the middle:



(Source: ColorBrewer)

Choosing color schemes is a complex science and art for every graphic you make. Many visualization tools include suggested color palettes, and you can use the website from which the examples above were taken called ColorBrewer. Originally designed for maps, but useful for charts in general, these color schemes have been

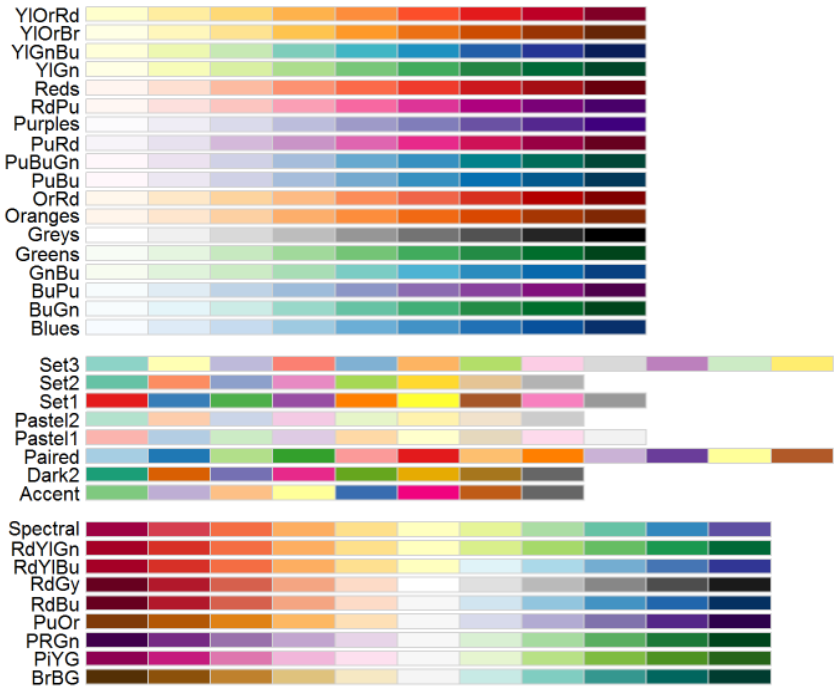
rigorously tested to be maximally informative. For the "Nations" dataset homework assignment, you explored the two palettes "set1" and "set2".

In ColorBrewer, notice that the colors it suggests can be displayed according to their values on three color "models": HEX, RGB and CMYK. Here is a brief explanation of these and other common color models.

- **RGB** Three values, describing a color in terms of combinations of red, green, and blue light, with each scale ranging from 0 to 255; sometimes extended to RGB(A), where A is alpha, which encodes transparency. Example: rgb(169, 104, 54).
- **HEX** A six-figure "hexadecimal" encoding of RGB values, with each scale ranging from hex 00 (equivalent to 0) to hex ff (equivalent to 255); HEX values will be familiar if you have any experience with web design, as they are commonly used to denote color in HTML and CSS. Example: #a96836
- **CMYK** Four values, describing a color in combinations of cyan, magenta, yellow and black, relevant to the combination of print inks. Example: cmyk(0, 0.385, 0.68, 0.337)
- **HSL** Three values, describing a color in terms of hue, saturation and lightness (running from black, through the color in question, to white). Hue is the position on a blended version of the color wheel in degrees around the circle ranging from 0 to 360, where 0 is red. Saturation and lightness are given as percentages. Example: hsl(26.1, 51.6%, 43.7%)
- **HSV/B** Similar to HSL, except that brightness (sometimes called value) replaces lightness, running from black to the color in question. hsv(26.1, 68.07%, 66.25%)

Check out this documentation on ColorBrewer: https://r-graph-gallery.com/38-rcolorbrewers-palettes

```
RColorBrewer::display.brewer.all()
```

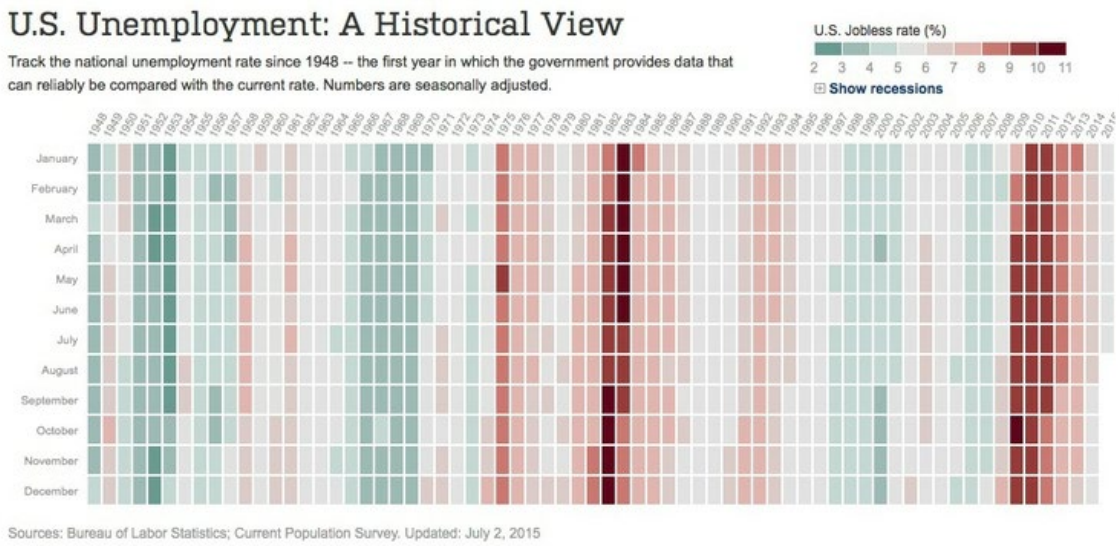


Colorizer is one of several web apps for picking colors and converting values from one model to another.

Custom color schemes can also work well, but experiment to see how different colors influence your story. The following graphic from *The Wall Street Journal*, for instance, uses an unusual pseudo-diverging scheme to encode data — the US unemployment rate — that would typically be represented using a sequential color scheme. It has the effect of strongly highlighting periods where the jobless rate rises to around 10%, which likely was the designer's aim.

Package: viridis
https://cran.r-project.org/web/packages/viridis/vignettes/intro-to-viridis.html

An Example of Use of a Diverging Color Scheme



What are 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 of Google has said: "Visualization is a gateway drug to statistics." Source: Peter Aldhous

Where to get data?

Here are some potential sources:

1. Open Case Studies (from the folks at JHU)
2. (new one for me) https://gender-pay-gap.service.gov.uk/viewing/download
3. https://www.cdc.gov/
4. https://www.data.gov/
5. https://datahub.io/
6. https://ourworldindata.org/
7. https://openpolicing.stanford.edu/
8. https://data.unicef.org/adp/downloads/ UNICEF's data portal on Adolescent health
9. https://datasetsearch.research.google.com/
10. Open Data Network Through Socrata: https://dev.socrata.com/data/
11. https://www.kaggle.com/datasets
12. http://data.un.org
13. World Bank Database (https://data.worldbank.org/indicator)
14. Census Bureau Data: https://www.census.gov/data.html
15. https://www.ipums.org/
16. Open Data from Montgomery County: https://data.montgomerycountymd.gov/
17. NYPD Data: https://www1.nyc.gov/site/nypd/stats/reports-analysis/stopfrisk.page
18. https://www.icpsr.umich.edu/icpsrweb/content/about/thematic-collections.html
19. WHO data: http://www.who.int/
20. https://data.mendeley.com/
21. 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, health, climate, government or politics.

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, chief scientist at RStudio:

- **readr** for reading and writes CSV and other text files.
- **dplyr** for processing and manipulating data.
- **ggplot** for creating plots and graphs

These and several other useful packages have been combined into a super-package called **tidyverse**.

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.

Homework Week 2

1. **(Ungraded)** Explore the links at the end of the notes for potential sources of data.

2. **(Ungraded)** Reread these notes and try copying, pasting, and running the code provided this week and last week. Remember – you are responsible for all code presented in these notes.

3. **(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 in the assignment dropbox.

4. **(Worth 10 points)** Follow the Airquality Homework Tutorial (https://rpubs.com/rsaidi/935731). In your own new Quarto document, copy the code to create **all four plots plus your own fifth plot** (Plot 1, Plot 2, Plot 3, Plot 4, and Plot 5).

   Knit the quarto file 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 items 3 and 4 via the course Week 2 Assignment Dropboxes by **11:59 pm, on Tuesday, ____.** We will present/discuss your submissions during the next class.