The Data Scientist's Toolbox (July 2015)

Part 1 of Johns Hopkins' Data Science Specialization

Jeff Leek, Ph.D., assistant professor of biostatistics
Bloomberg School of Public Health, Johns Hopkins University
simplystatistics.org | @jtleek

Week 1

1.1: Series motivation

Why do data science?

- Theodore Roosevelt quote -- the person who counts is the one who's actually out there in the arena trying to get things done, even when there are obstacles in the way -- not the person pointing out what that person is doing wrong.
- A lot of data science involves pushing through difficulties with large or messy data:
    - collecting it
    - cleaning it up
    - building new analysis techniques to explore new information about it
- Those steps can be complicated and can open you up to criticism - hence the quote.

The key challenge

- Dan Myer (math educator): "Ask yourselves, what problem have you solved, ever, that was worth solving, where you knew all of the given information in advance? Where you didn't have a surplus of information and have to filter it out, or you didn't have insufficient information and have to go find some."
- You either:
    - don't have enough data to answer your question and have to search it out, or
    - are overwhelmed with too much data and need to filter out the stuff you don't need to answer your question.
- *Answering questions with data is at the heart of data science. The question you're interested in comes first; the data should follow.*
    - Sometimes the data will answer *a* question, but not necessarily *your* question.

The instructors of this course

- All are on the biostatistics faculty of Johns Hopkins' Bloomberg School of Public Health, doing data-intensive statistics in biology and medicine.
    - Brian Caffo - statistics of analyzing brain imaging data
    - Jeff Leek - statistics of analyzing genomics data
    - Roger Peng - statistics of analyzing fine particulate matter
- These areas involve data that's not always clean or easy to handle, and complicated questions that must be broken down into parts to answer.

- Techniques we'll learn about can be used in a variety of areas, not just biology/medicine - which has seen a huge upsurge in the amount of data available.

Why should you learn about data science?

- There's so much more data out there.
    - Easier to collect
    - Cheaper to store
    - Free computing tools allow you to make sense of it all
- Big data is a new frontier
    - We have data in areas not previously available to us
        - For example, genome sequencing

Why statistical data science?

- Statistics = the science of learning from data
    - It's pretty rare that you'll get a data set where all the answers are really clear, with no uncertainty.
    - When there is uncertainty, that's where statistics plays a role.

Why are you lucky?

- This point in time is not unlike when Jeff Bezos discovered the Internet and created Amazon.
    - Internet usage was experiencing explosive growth
    - Bezos had chance to create something amazing, huge and wonderful
    - Opportunity to "jump on a rocket" and carry it into a larger endeavor
- Tools, competitions, websites developed around data-related projects that have high-profile results.
    - Heritage Health Prize - $3 million contest involving analysis of data to predict who'd be admitted to a hospital in the next year.
    - Exciting opportunities in projects that wouldn't have existed 5-10 years ago.

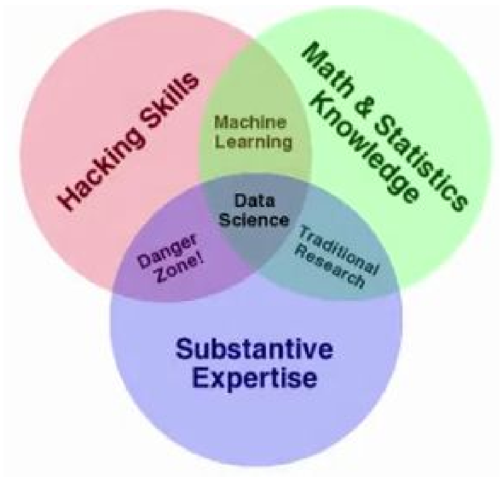
Why R?

- These courses will focus almost exclusively on the use of the R programming language.
    - It's increasingly become the dominant language used in data science.
        - Other languages can complement R, such as Python.
    - Has broad range of packages that can handle things from raw files to interactive reports, documents, shareable web apps.
        - Data access
        - Data cleaning
        - Analysis
        - Data reporting
    - It's free
    - It has one of the best development environments - RStudio
    - Has an amazing developer ecosystem

        - Lots of people developing R packages
        - Can reach out to them via mailing lists, email, Stack Overflow
    - Packages are easy to install, “play nicely together”

Who is a data scientist?
Some people you might not immediately think of (who might not even give themselves the title):

- Daryl Morey, GM of Houston Rockets - uses data to analyze players, transactions, trades
- Hilary Mason, Accel Partners - answers questions about mining the web, understanding how humans interact via social media
- Daphne Koller, CEO of Coursera - uses data gleaned from site to improve educational delivery, assessment at huge scale
- Nate Silver, FiveThirtyEight - used free public data to predict election results, with surprising accuracy



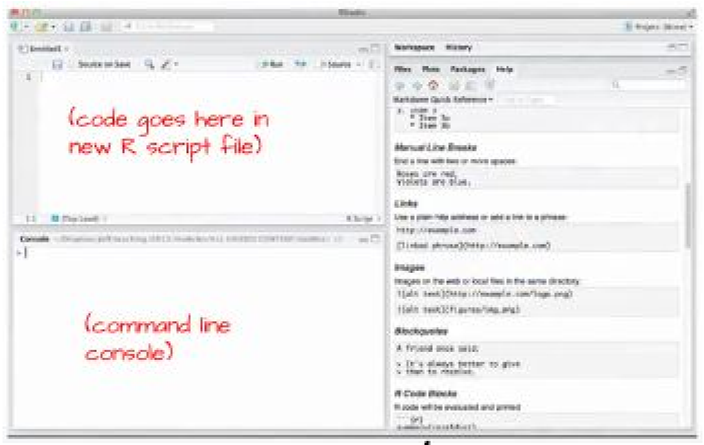
Our goal

- Data science lies at intersection of 3 different skill sets: hacking skills, math/statistics knowledge, substantive expertise.
- We’ll focus mainly on math/stats & hacking skills.
- Hacking skills involves 2 components:
    - computer programming
    - the ability to go out and answer questions for yourself

1.2: The data scientist’s toolbox

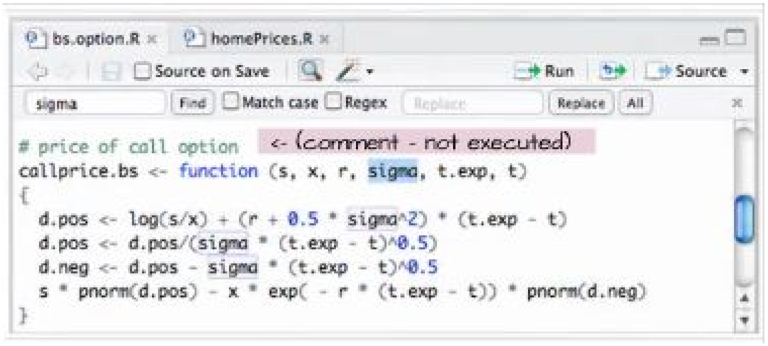
What do data scientists do?

- Define a question of interest
- Define the ideal data set to try to answer that question
- Determine what data you can access
- Obtain the data (for example, from a database, website)
- Clean the data (preparing it for analysis)
- Exploratory data analysis (for example, making plots or clusterings to identify patterns)
- Statistical prediction/modeling to build an intuition about what’s going to happen in the next sample you take
- Interpret results



RStudio

- Challenge results
- Synthesize/write up results
- Create reproducible code
- Distribute results to others (interactive graphics, write-ups, presentations, web apps

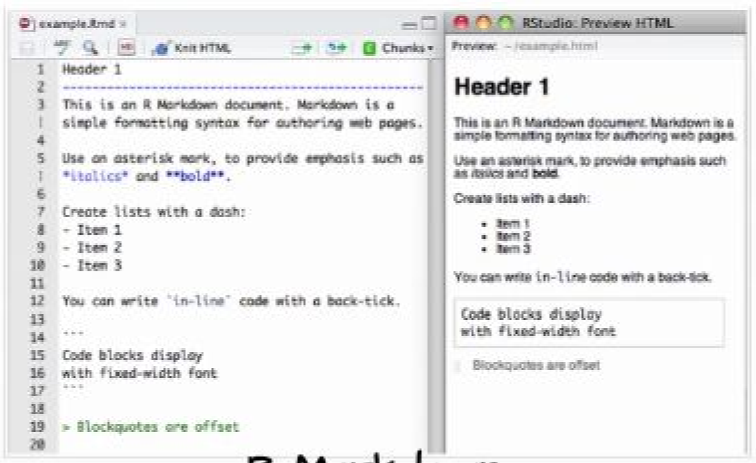


```
sigma
# price of call option
callprice.bs <- function (s, x, r, sigma, t.exp, t)
{
  d.pos <- log(s/x) + (r + 0.5 * sigma^2) * (t.exp - t)
  d.pos <- d.pos/(sigma * (t.exp - t)^0.5)
  d.neg <- d.pos - sigma * (t.exp - t)^0.5
  s * pnorm(d.pos) - x * exp( - r * (t.exp - t)) * pnorm(d.neg)
}
```

sample R script

What we'll be using

- R is the main workhorse of data science - we'll be installing it in Week 2
- We'll work on our coding in RStudio, an integrated development environment (IDE) for R.
  - Like R, this is free
  - We'll also be installing this in Week 2
- Primary file types used
  - R script
    - Text file ending in the extension .r
  - R markdown document
    - Structured-format text file ending in the extension .rmd
    - Primary vehicle to allow for reproducibility of your research



```
Header 1
--------------------------------------------------------
This is an R Markdown document. Markdown is a
simple formatting syntax for authoring web pages.

Use an asterisk mark, to provide emphasis such as
*italics* and **bold**.

Create lists with a dash:
- Item 1
- Item 2
- Item 3

You can write `in-line` code with a back-tick.

```
Code blocks display
with fixed-width font
```


> Blockquotes are offset
```

Header 1

This is an R Markdown document. Markdown is a simple formatting syntax for authoring web pages.

Use an asterisk mark, to provide emphasis such as *italics* and **bold**.

Create lists with a dash:

- Item 1
- Item 2
- Item 3

You can write `in-line` code with a back-tick.

```
Code blocks display
with fixed-width font
```

Blockquotes are offset

R Markdown

    - Can be "knitted" to HTML (creating a nicely formatted HTML file) with the click of a button
- We'll use Git and GitHub for sharing results
  - Class will include setting up a GitHub account and portfolio
  - Commands are run from the shell (command line interface)

1.3: Getting help

Asking questions

- In a MOOC, you can't exactly raise your hands and get a direct response from the instructor.
- Instead, you should post questions to the message board.
  - Others "upvote" your questions
  - Your instructor and peers respond (as often as possible)
- But often, the fastest answer is the one you find yourself.
  - Try to answer your own questions first

    - If the answer's in the help file or the top hit on Google, the answer will be "read the documentation" or [lmgtfy.com]
    - If you find an answer and see others in the forum with the same questions, post your solution!

A few important R functions

Useful when goal is to figure out how R is working for a particular function, not so much in trying to understand the underlying concepts

- Access help file `?rnorm`
    - *rnorm is the function*
- Search help files `help.search("rnorm")`
    - *search term does not need to be exact*
- Get arguments `args("rnorm")`
- See code `rnorm`
    - *function name typed with no brackets; reproduces entire code*
- [Here is a helpful R reference card].

How to ask a question about R

The tools we use are evolving over time, so it's important to keep the following in mind:

- What steps will reproduce the problem?
- What is the expected output?
- What do you see instead?
- What version of the product (e.g. R, packages, etc.) are you using?
- What operating system are you working on?

How to ask a data analysis question

You'll need to report a similar set of components:

- What's the question you're trying to answer?
- What steps/tools did you use to answer it?
- What did you expect to see?
- What do you see instead?
- What other solutions have you thought about?
    - Detailing the things you've already tried lets people helping you move on to something you haven't tried.

Be specific in question titles

The more specific you are, the faster you'll get an answer.

- Bad:
    - HELP! Can't fit linear model!
    - HELP! Don't understand PCA!
- Better:
    - R 2.15.0 lm() function produces seg fault with large data frame, Mac OS X 10.6.3

        - *contains version of R, specific fault produced, circumstances under which fault occurs, OS being used*
    - Applied principal component analysis to a matrix - what are U, D, and V^T?
- Even better:
    - R 2.15.0 lm() function on Mac OS X 10.6.3 -- seg fault on large data frame
        - *more succinct*
    - Using principal components to discover common variation in rows of a matrix, should I use U, D or V^T?

Etiquette for forums and help sites

- DO:
    - Describe the goal
    - Be explicit
    - Provide the minimum information (giving too much makes it more difficult to filter through to find the real problem)
    - Be courteous (this never hurts)
    - Follow up and post solutions (especially if you end up finding answer elsewhere)
    - Use forums rather than email to contact instructors
- DON'T:
    - Immediately assume you found a bug in a major program (such as R)
    - Grovel as a substitute for doing your work
    - Post homework questions on mailing lists or forums (people don't like doing your assignments for you, and it detracts from others' experience)
    - Email multiple mailing lists at once - or the wrong mailing list
    - Ask others to fix your code without explaining the problem
    - Ask about general data analysis questions on R forums
- Inspired in part by "How to Ask Questions the Smart Way," Eric Raymond and Rick Moen

1.4: Finding answers

Hacking skills?

- Much of the knowledge you'll need won't be found in standardized textbooks - it's scattered in a bunch of different places and must be found and synthesized.
- Key characteristics of hackers:
    - Willing to find answers on their own
    - Knowledgeable about where to find answers on their own
    - Not intimidated by new data types or packages
    - Not afraid to say they don't know the answer
    - In short: Polite, but relentless

Where to look for different types of questions

- R programming (see also: Roger Peng's video on getting help)
    - Search class forums' archive

    - Read the manual/help files
    - Search on the web
    - Ask a skilled friend
    - Post to the class forums
    - Post to the R mailing list or Stack Overflow
- Data analysis/statistics:
    - Search class forums' archive
    - Search on the web
    - Ask a skilled friend
    - Post to the class forums
    - Post to Cross Validated *(like Stack Overflow, but about statistics topics)*
- Other software: software-specific websites; GitHub

Googling data science questions isn't always easy …

- The forums are the best place to start for general questions
- Stack Overflow (use [r] tag), R mailing list for software questions, Cross Validated for more general questions
- Otherwise, try Googling "[data type] data analysis" or "[data type] R package"
- Data analysis is called different things based on the data type - identify the term you need. Some examples:
    - Biostatistics (medical data)
    - Data science (data from web analytics)
    - Machine learning (data in computer science/computer vision)
    - Natural language processing (data from texts)
    - Signal processing (data from electrical signals)
    - Business analytics (data on customers)
    - Econometrics (economic data)
    - Statistical process control (data on industrial processes).

1.5 - 1.12 are brief overviews of the other courses in this specialization.

1.13: Installing R on Windows

- Go to the Comprehensive R Archive Network - cran.r-project.org
- Select Windows, then base, then Download R [version number] for Windows
- Installer will walk you through the steps
    - default components setup should be fine
    - "customized startup" options are things like
        - MDI interface (one big window with subwindows) or SDI (separate windows)? (Peng prefers SDI)
        - Help files - plain or HTML?
        - You shouldn't be messing with the Internet Access setting.
        - shortcut in start menu

            - additional tasks

1.14: Installing RStudio

- You MUST have already installed R.
- Go to rstudio.com
- Click “Download RStudio” link *(note: no longer a green button as depicted in video)*
- Select the button for desktop version - site should auto-detect your operating system and recommend a version.

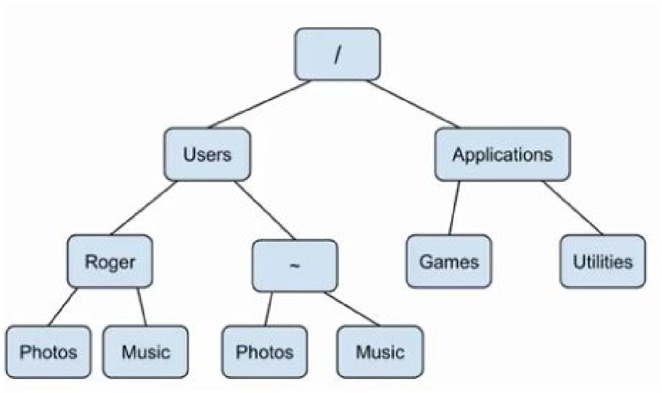
Week 2

2.1: Introduction to the command line interface

What is the command line interface?

- A means of working with files and folders by typing a command rather than pointing and clicking with a mouse.
- Nearly every computer comes with a CLI:
    - On Windows: Git Bash
    - On Mac/Linux: Terminal
- What can you do with it?
    - Navigate folders
    - Create and edit files, folders, programs
    - Run programs

Directory basics



- Directory = another name for folder
- On your computer, they are organized like a tree - directories can be inside other directories
    - The directory “up” from Specific Directory is the one containing Specific Directory.
        - Example: Debussy is inside my Music directory. So the Music directory is one directory “up” from Debussy.
- Some special directories worth knowing about:
    - Root (/): the directory at the top of the tree, containing all other directories. Represented with a slash.
    - Home (~): usually contains most of your personal files, pictures, music, etc. Represented by a tilde; usually its name is the name you use to log in to your computer.

CLI basics

- Opening the CLI:
    - On Windows: Open Start, search for and open Git Bash.
    - On Mac: Open Spotlight, search for and open Terminal
    -
- You'll see a prompt (something resembling the name of your computer), followed by your username, followed by a $. You always start in your home directory.
- The <u>working directory</u> is whatever directory you're currently working with.
- The <u>path</u> to your working directory is the set of directories you must follow to get from your current directory back to root.
    - The command **`pwd`** will display this (it stands for "print working directory").

CLI commands

- Commands follow this recipe: ***command flags arguments***
    - <u>command</u> = the command that does a specific task
    - <u>flags</u> = options we give to the command to trigger certain behaviors, preceded by -.
    - <u>arguments</u> = can be what the command will modify, or other options for the command
- Depending on the command, there may be zero or more flags or arguments.
    - `pwd` is an example of a command requiring no flags or arguments.
- Useful commands:
    - **`pwd`** displays the path to your working directory
    - **`clear`** clears out commands in current CLI window
    - **`ls`** lists unhidden files and folders in the current directory
        - **`ls -a`** lists hidden and unhidden files and folders
        - **`ls -al`** lists details for hidden and unhidden files and folders
        - Note that `-a` and `-l` are flags, combined into the flag `-al`
    - **`cd`** changes the directory.
        - takes as an argument the directory you want to visit.
        - `cd` with no argument takes you to the home directory.
        - **`cd ..`** lets you change directory to one level above your current directory.
    - **`mkdir`** makes a new directory - it's equivalent to "create new folder" in a GUI.
        - takes as an argument the name of the directory being created.
    - **`touch`** creates an empty file.
        - takes as an argument the name of the file being created.
    - **`cp`** creates a copy of a file.
        - the first argument is the file being copied; the second is path to where you want the copy to be placed.
        - can also be used to copy the contents of directories, but you must use the **`-r`** flag (stands for recursive).
            - **`cp -4 Documents More_docs`** copies the contents of `Documents` into `More_docs`.

    - `rm` removes (deletes) a file
        - takes as an argument the name of the file you want to remove
        - `rm -r` will remove entire directories and their contents
        - *This command cannot be undone, so be careful when using it.*
    - `mv` moves files between directories
        - takes as arguments the file being moved, the directory it's being moved to.
        - can also be used to rename file: `mv new_file renamed_file`
    - `echo` will print whatever arguments you provide
        - example: `echo Hello World!` → Hello World!
        - useful for printing out contents of particular variables that have been stored.
    - `date` will print today's date

2.2: Introduction to Git

What is version control?

- A system that records changes to a file or set of files over time so that you can recall specific versions later.
- One of data scientists' most commonly used tools
- Many of us constantly create something, save it, change it, save it again, etc. Version control manages this process reliably and efficiently.
- This is especially important when collaborating with others.

What is Git?

- A free, open source distributed version control system designed to handle projects quickly and efficiently, regardless of size.
- Created by the same folks who developed Linux.
- The most popular implementation of version control today
- Everything is stored in local repositories (repos) on your computer
- Operated from the command line
- More info at git-scm.com

How to install Git

- Download Git at git-scm.com/downloads
- Open file to begin installation
- Go with the default options unless you really know what you're doing.

Getting started with Git

- If you are a Windows user, open Git Bash - the command line environment for interacting with Git. It should be in the directory you installed Git to.
- You'll see a short welcome message, then the name of your computer and a dollar sign on the next line.
    - The dollar sign means it's your turn to type a command.

- You'll need to configure your username and email.
    - Each commit to a Git repository is tagged with the username of the person who made the commit.
    - To set these up, enter these commands in Git Bash, one at a time:
        - `$ git config --global user.name "Your Name Here"`
        - `$ git config --global user.email "your_email@example.com"`
        - To confirm your changes: `$ git config --list`
    - You only need to do this once, but you can use these same commands to make any changes later.
- Close Git Bash using the command: `$ exit`

2.3: Introduction to GitHub

What is GitHub?

- It's a web-based hosting service for software development projects that use Git.
- Lets you contribute to others' projects and have them contribute to yours.
- Allows users to "push" and "pull" their local repositories to and from remote repositories on the web.
- Provides users a homepage that displays their public repos
- Users' repos are backed up on the GitHub server in case something happens to local copies.
- Social aspect - users can follow one another and share projects.

Getting started

- Go to https://github.com and ener a username, email and password to sign up.
    - Use the same email address you used when setting up Git.
- Select the free plan, then "finish signup"
- Remember that having a GitHub account set up is required for this class.
- You'll be taken to a page with resources to learn more about Git and GitHub.
- Your username in upper righthand corner is a link to your profile - it displays all your activity on GitHub, lets you show others who you are and what you're working on, and will become a portfolio of your work.

2.4: Creating a GitHub repository

Recap

- You don't need GitHub to use Git
- Git is local on your computer; GitHub is remote hosting (on the web)
- GitHub lets you share repositories with others, access others' repositories, and store remote copies in case something happens to your local files.

Starting a repo from scratch

- Click the + sign next to your profile link in the upper right-hand corner, and select "New Repository." OR go directly to https://github.com/new

    - Create a name for your repo and type a brief description
    - Select “Public” (private repos require a paid or education account)
    - Check the box next to “Initialize this repository with a README”
        - This README file (coded in Markdown) shows up on repo’s main page.
    - Click “Create Repository”
- Create a local copy of the repo on your computer so you can make changes to it.
    - Open Git Bash
    - Create a local directory where you’ll store your copy of the repo:
        - **`$ mkdir ~/test-repo`**
    - Navigate to the new directory using **`$ cd ~/test-repo`**
    - Initialize a local Git repo: **`$ git init`**
    - Point your local repository at the remote repository you just created on GitHub:
        - **`$ git remote add origin`** **`https://github.com/yourUserNameHere/test-repo.git`** *(that is all on one line but the indent bumped it down)*

Starting a repo by “forking” another user’s

- “Forking” - making a copy of someone else’s repo - is an important aspect of open-source software development.
- Navigate to the desired repository on the GitHub website and click “Fork.”
- You now have a copy of the desired repository on your GitHub account.
- Now you need to “clone” the repo - make a local copy on your computer:
    - **`$ git clone https://github.com/yourUserNameHere/repoNameHere.git`**

Some other things

- If you make changes to your local copy, you’ll want to push those to GitHub at some point.
- You may also be interested in staying current with any changes made to the original repo from which you forked your copy.
- We’ll cover more basics in future lectures, but here are some resources:
    - https://help.github.com/articles/fork-a-repo
    - https://git-scm.com/book/en/Git-Basics-Getting-a-Git-Repository

2.5: Basic Git commands

A quick overview

- Locations
    - Workspace = directories where you’re working with files on your own computer
    - Index = tells Git the files it should

        be controlling under version control
    - Local repository = files version-controlled on your local computer
    - Remote repository = files version-controlled in the cloud - in our case, on GitHub.
- File is created in workspace
- File is added to index so Git knows to track that file and all its changes.
- File is committed - a version is put in the local repo so it can be stored and updated.
- After a few commits, file is pushed to the remote repo to update it on GitHub.

Adding

- If you're adding new files to a local repo under version control, you need to let Git know they need to be tracked.
    - `git add`   *adds all new files*
        - presumes you're in the directory where you added the new files
    - `git add -u`   *updates tracking for files that changed names or were deleted*
    - `git add -A`   *does both of the above*
- Do this BEFORE committing.

Committing

- Do a commit to save your changes as an intermediate version.
- The command: `git commit -m "message"`
    - *where "message" is a useful description of what you did*
- This only updates the local repo, not the remote one.

Pushing

- Do a push when you have saved local commits you'd like to update on the remote repo.
- The command: `git push`

Branches

- Say you're working on a project with a version used by many people - you may want to create a branch where you can work independently, rather than edit the version everyone's using and risk breaking their work.
- Command to create a branch: `git checkout -b branchname`
- To see what branch you're on: `git branch`
- To switch back to master branch: `git checkout master`

Pull requests

- If you fork someone's repo or have multiple branches, you'll both be working separately.
- If you want to merge your changes into the other branch/repo, you'll need to send a pull request.
- This is a unique feature of GitHub.
- On the repo page, select the branch, then click "Compare and Pull Request" button.
- This will issue a pull request to whoever owns the other branch/repo. They can decide whether to merge the pull request or not.

More resources:

- Git documentation - http://git-scm.com/doc
- GitHub help - https://help.github.com
- Google/Stack Overflow

2.6: Basic Markdown

- Markdown is a specific, simple way of formatting text files that can be recognized and used for a variety of tasks by things like GitHub, R and RStudio.
- We'll learn a bit more about it in later courses - for now we'll be covering the basics you'll need to create your first few directories in GitHub
- Markdown files end with the file extension .md.

Syntax

- Headings
  - `## This is a secondary heading` (sim. to <h2></h2>)
  - `### This is a tertiary heading` (sim. to <h3></h3>)
- Unordered lists - use asterisks to generate a bullet.

  -
    ```
    * first item in list
    * second item in list
    * third item in list
    ```

More Markdown resources

- http://daringfireball.net/projects/markdown
- RStudio has a quick guide (click the MD button)
- R Markdown guide (you won't need this until the Reproducible Research course): http://rstudio.com/ide/docs/authoring/using_markdown

2.7: Installing R packages

What are R packages?

- The version of R you download from CRAN is a base version that comes with basic functionality and implementation of the R language
- R packages extend R's basic functionality. The large collection of packages developed and published by the larger R community are one reason R is so useful.

Where do you get R packages?

- Primarily from CRAN, which as of lecture taping has about 5,200 packages on a variety of topics.
- The Bioconductor project has many packages available for biological and big data applications.
- The **`available.packages()`** function provides more information about the available packages on CRAN:

    - `a <-- available.packages()` *this will show you all of the thousands of packages*
    - You could instead use the head command to look at a certain number.
        - `head(rownames(a), 3)`
          `## shows the name of first few packages in alphabetical order`
- A list of some topics is available through the Task Views link, which groups together many R packages related to a given topic.

Installing an R package

- Use the **`install.packages()`** function in R, passing in the name of the package as the first argument.
    - `install.packages("slidify")` downloads the Slidify package from CRAN and installs it on your computer, along with any other packages required to run the Slidify package.
    - A single call to `install.packages()` can be used to install multiple packages at the same time - place the names of the packages in a character vector.
    - `install.packages(c("slidify", "ggplot2", "devtools"))`
- You can also install packages via RStudio.
    - Go to Tools menu, then down to Install Packages.
    - It will open a window that allows you to select the repository and package you want to install from.
- Installing packages from Bioconductor isn't *quite* as straightforward.
    - To get the basic installer and set of R packages *(installs multiple packages)*:
        - `source("http://bioconductor.org/biocLite.R")`
          `biocLite()`
    - Place the names of the R packages in a character vector.
        - `biocLite(c("GenomicFeatures", "AnnotationDbi"))`

Loading R packages

- Installing a package doesn't make it immediately available in R; you must load it.
- This is done using the **`library()`** function.
    - `library(ggplot2)` loads the ggplot2 package into R.
    - Note that before loading the named package, you'll need to have already loaded any packages needed as dependencies.
    - *Note that the package name is NOT in quotes!*
    - Some packages produce messages when they are loaded (but some don't)
- After loading a package, functions supported by that packages will be attached to the top of the **`search()`** list (after the workspace)

2.8: Installing Rtools

What is Rtools?

- A collection of tools needed to build R packages in Windows.

- Download it at http://cran.r-project.org/bin/windows/Rtools/.
- (You won't need this right away, but it will come up in the Developing Data Products course.)

Downloading and installing Rtools

- Select the .exe download link from the table that corresponds to your version of R.
    - Not sure what version you have? Open or restart R and it's the first thing that shows up in the console.
- If you have the most recent version of R, select the download at the top of the chart.
- It's best to go with the default selections at each step of installation, with two exceptions:
    - If you already have Cygwin on your machine, follow the instructions during installation (also here).
    - IMPORTANT: You should make sure the box is checked to have the installer edit your path.
- Once you're done installing Rtools, you should install the devtools R package if you haven't already.
    - Open RStudio
    - If you're not sure if you've installed devtools, enter `find.package("devtools")` in console to check
    - To install devtools: install.packages("devtools")
- Then, verify Rtools installation:
    - After devtools installation, load it: `library(devtools)`
    - Then type `find_rtools()`. If the install worked properly, this should return `TRUE` in the console.

Week 3

3.1: Types of data science questions

Listed in approximate order of difficulty for achieving the goal:

- **Descriptive**
    - *Goal: Describe a set of data.*
    - The first kind of data analysis performed; most commonly applied to census data.
    - Description and interpretation are different steps.
    - Descriptions usually cannot be generalized without additional statistical modeling - you're describing what you see in the data, but not saying what that might mean for the next person.
    - Examples:
        - Census.gov's 2010 census page - presents information about Americans but is not necessarily analyzing it to predict something about people elsewhere
        - Google Ngram Viewer - collects information about pairs or trios of words, and plots their association over time.
- **Exploratory**

- *Goal: Find relationships you didn't know about previously (but not necessarily confirm them)*
    - Good for discovering new connections; also useful for defining future studies.
    - Exploratory analyses are usually not the final say.
    - Exploratory analyses alone shouldn't be used for generalizing/predicting/
    - Correlation does not imply causation!
    - Examples:
        - Examining brain images and trying to ID which regions of the brain were lit up in response to a particular stimulus. Haven't confirmed what it means, but a new connection was discovered.
        - Johns Hopkins' Sloan Digital Sky Survey - terabytes (or more) of data collected from looking at the night sky at different times and places. Data is used for exploration, but not necessarily for confirming anything.
- **Inferential**
    - *Goal: Use a relatively small sample of data to generalize something about a larger population.*
    - Inference is commonly the goal of statistical models.
    - Involves estimating both the quantity you care about and your uncertainty about your estimate.
    - Depends heavily on both the population and the sampling scheme.
    - Example: Studying the effect of air pollution control on U.S. life expectancy, using data on a subset of 545 counties to infer what's generally happening between the two.
- **Predictive**
    - *Goal: Use the data on some objects to predict values for another object.*
    - If X predicts Y, it does not mean that X causes Y.
    - Accurate prediction depends heavily on measuring the right variables.
    - Although there are better and worse prediction models, more data and a simple model works really well.
    - Prediction is very hard, especially about the future.
    - Examples:
        - FiveThirtyEight - Nate Silver's election predictions, based on polling data, were very accurate in the 2008 and 2012 elections
        - Target figured out a teen girl was pregnant based on her purchase history, then sent her a targeted flyer … before her dad even knew. *Awk-ward.*
- **Causal**
    - *Goal: Find out what happens to one variable when you make another variable change*
    - Usually randomized studies are required to identify causation.
    - There are approaches to inferring causation in nonrandomized studies, but they are complicated and sensitive to assumptions.

    - Causal relationships are usually identified as average effects, but may not apply to every individual.
    - Causal models are usually the “gold standard” for data analysis.
    - Example: Study of fecal transplants of different bacteria to treat a particular type of recurrent infection. Randomized people to get the transplants, then determined that it was causally associated with better outcomes.
- **Mechanistic**
    - *Goal: Understand the exact changes in variables that lead to changes in other variables for individual objects.*
    - Incredibly hard to infer, except in simple situations.
    - Very rarely covered in data science, and rarely the goal of most analyses.
    - Usually modeled by a deterministic set of equations; most common applications are in physical or engineering science.
    - Generally the random component of the data is measurement error.
    - If the equations are known but the parameters are not, they may be inferred with data analysis.
    - Example: Trying to discover how differences in pavement design would directly lead to changes in that pavement’s functioning.

3.2: What is data?

- Wikipedia sez: values of qualitative or quantitative variables, belonging to a set of items.
    - <u>set of items</u> = the set of objects you’re interested in discovering something about; sometimes called the <u>population</u>.
    - <u>variables</u> = measurements or characteristics of an item.
        - <u>qualitative variables</u> = things that are not necessarily ordered or measured on a scale, like country of origin, sex, treatment
        - <u>quantitative variables</u> = things that are measured and ordered on a continuous scale, such as height, weight, blood pressure

What do data look like?

- Most data starts off in a very raw form - you want to parse specific sections, collect that data and make inferences.
    - FASTQ text file used to store nucleotide sequences
    - An API, like Twitter’s - access a URL and get back a structured form of data
    - Medical records - can be used to study ways to improve insurance or medical care. Text files may not be nicely formatted; you may want to do things like extract allergy names.
    - Video - used to create algorithm that could learn whether a video was of a cat or something else.
    - Audio - DarwinTunes project has people studying the evolution of music over time.
    - Open government websites such as data.gov - may be in a variety of formats.

        - Data very rarely shows up before a study looking like what you'd expect - a nicely formatted spreadsheet.

The data is the second most important thing (the most important is the question)

- Often, the data will limit or enable the questions.
- But having data can't save you if you don't have a question.

3.3: What about big data?

How much data is there?

- 1.8 zettabytes created/replicated in 2011, though only a tiny fraction of that could be applied to a specific question.

What about big data?

- It's a matter of perspective. Over time, as technology increases, big data will change.
    - Consider that one of the first hard drives IBM developed had less capacity than a cellphone today.
    - One way to solve the big data problem is to kind of wait until hardware catches up with the data size.
    - Most questions don't necessitate huge numbers of computers.
- Often linked to "the cloud" because some data sets are so big that they can't be analyzed on your local laptop computer.

Why big data now?

- People nowadays can collect much more data, and much more cheaply.
    - Milgram's "Small World" experiment spawned the idea of "six degrees of separation."
        - 296 people were sent letters, which asked them to send a letter from someone they knew, and so forth, until they went to a specific address.
        - 64 of these "chains" came back, with about 5.2 people between each starter and target.
    - Study looking at 30 billion conversations among 240 million people performed an experiment similar to Milgram's - found an average path length of 6.6.

Big or small, what's important is that you need the right data.

- With a powerful enough computer, you can analyze a large number of data sets important and relevant to you - without having to get into more complex techniques such as Hadoop (a very powerful technique useful for very massive data sets like those analyzed by Facebook and Google).
- The data may not contain the answer, no matter how big the data are.

3.4: Experimental design

Why should you care about experimental design?

- Study found patients' genome information to predict the type of chemotherapy they'd respond to best.
    - This would be a really cool result that would open the door to all sorts of personalized therapeutics.
    - BUT … both the study design and the statistical analysis were very flawed, as pointed out in another paper later.
    - And the study had been ongoing for a while, to the point that clinical trials using those predictions had begun. The people in the clinical trial filed a lawsuit against the original investigators.
- Moral of the story: A very exciting result can lead you astray if you're not very careful with experimental design and analysis.

Know and care about the analysis plan

- It's the first thing you want to be aware of.
- You do not want things like "(insert statistical method here)" in the abstract of your published paper (because the author didn't *know* what the method was).
- Being aware of this ensure you'll be aware of key issues that could trip you up along the way.

Have a plan for data and code sharing

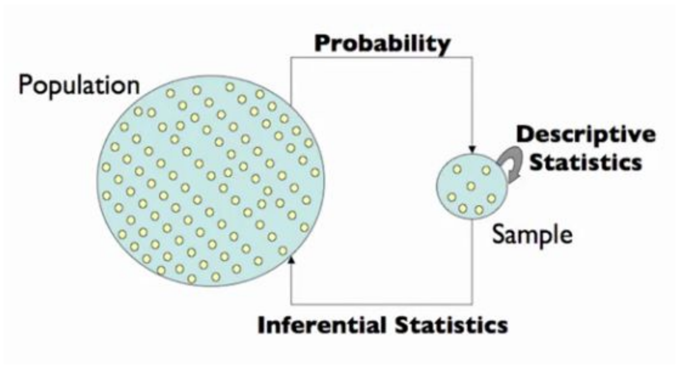
- Use GitHub to share your code and very small amounts of data.
- For larger amounts of data, you might use a site like figshare.com.
- If you have very large or unwieldy data that can't be shared on one of these sites, you might need another plan.
- Recommended resource - Leek's group guide to data sharing

Formulate your question in advance

- Data science is a scientific discipline; thus, it requires that you're answering a specific question.
- Example -- A study by Obama's re-election campaign used the question: *Does changing the text on your website improve donations?*
    - Two versions of the website were created, and visitors were randomly shown one version of another.
    - How much they donated was measured.
    - From that, it was determined which version made for more donations.

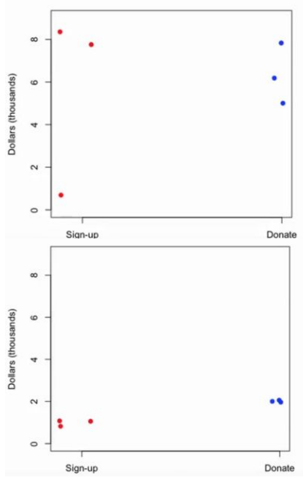
This brings us to a large component of data science - statistical inference.

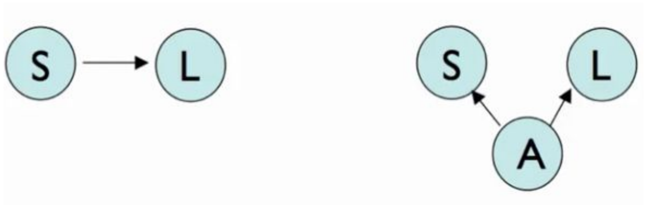
- It would be too expensive to show the website to the entire population (all potential donors in the U.S. who might donate through the Obama website)
- Instead, they might select a small subset with a probability argument to show them



one version or the other. This results in a <u>sample</u> - a much smaller number of people who'd observed the data and decided whether to donate or not.

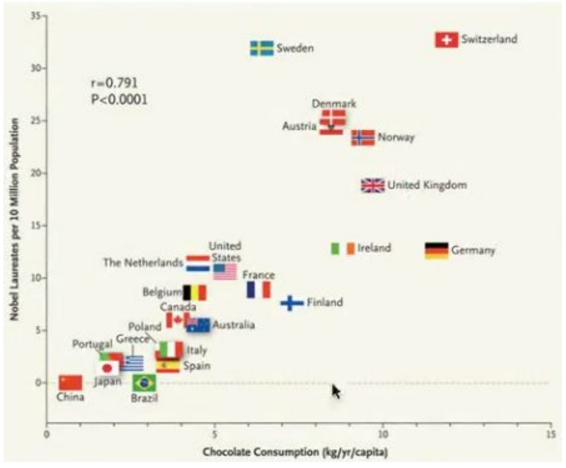
- They'd then calculate <u>descriptive statistics</u> (such as the total number of donations received over a certain number of visits, or average number of donations).
- Then, they'd use <u>inferential statistics</u> to try to decide if the statistics calculated from the small sample would play out the same way if applied to the full population.
    - Say the data for the two versions ("sign up" and "donate") showed the total number of dollars donated for every 1,000 visitors.
        - Scenario 1 - 3 different experiments might get you 3 different observations. The average may be more or less the same, or may be slightly different, but if you have highly variable answers for both versions of the site - it's hard to tell.
        - Scenario 2 - The "sign up" version got a very slightly smaller number of dollars donated - you can tell this because the variability is much smaller. However, the benefit of switching to the 'donate" version may be too small to be worthwhile.
            - The benefit is clearer if you get a result showing small variability in observations but a large difference in donations. That would suggest only the "donate" version of the site should be shown.





Confounding

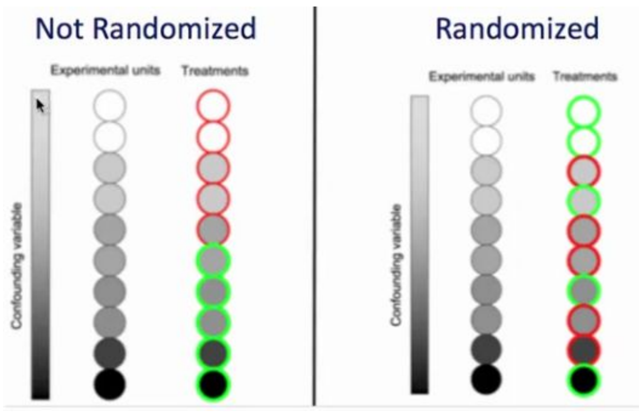
- Suppose you measured shoe size (S) and literacy (L) and were looking for correlations between the two.
    - You might observe a few, because people with small shoes tend to have less literacy.
    - But what you'd be missing is that <u>age</u> (A) is actually the variable causing this relationship. The very young lack literacy and have small feet, and both develop over time.
    - Age is a <u>confounding variable</u> for the relationship between shoe size and literacy.
    - If you only measure shoe size and literacy, you might be led astray if you observed a correlation between the two. This is what's called <u>confounding</u>.
- It's important to pay attention to other variables that might be causing a relationship. This plays out all the time in a variety of studies.
- *Correlation is <u>not</u> causation.*
    - Possibly serious *New England Journal of Medicine* article attempts to correlate



chocolate consumption (kilograms per capita) with Nobel laureates per 10 million population.
    - There are potentially many reasons for this - including that countries that have more money would eat more chocolate, and would probably also have better education systems (more Nobel winners). So the observed correlation may not be an actual relationship.
    - This study is an example of what's sometimes called <u>spurious correlation</u>.
    - Even if you see two things correlated with each other, you have to prove that the correlation is not the result of some other variables we didn't measure.

Some ways to deal with potential confounders: randomization and blocking

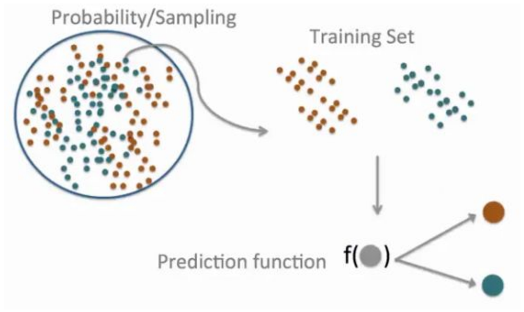
- If you can (and want to), fix a variable
    - Website always says "Obama 2012" on it
- If you don't fix a variable, stratify it.
    - If you're testing sign-up phrases and have two website colors, use both phrases equally on both colors.
- If you can't fix or stratify a variable, randomize it.
    - Randomly assign people to groups, by using a computer program or flipping a coin.
    - Why does this help?
        - Say we're performing a study with 10 experimental units. There's a confounding variable - lower values correspond to lighter colors, higher ones to darker colors, and they're in a range like this.



        - Splitting them up into low-confounder and high-confounder groups (left), and applying treatments accordingly, would make it impossible to tell whether any change was the result of differences in treatment or differences in confounding variable.
        - If we randomly assign treatments (as seen at right), each group will have both high-confounder and low-confounder units. That balance will help us better determine that the difference in treatment is what's being observed in the outcome.

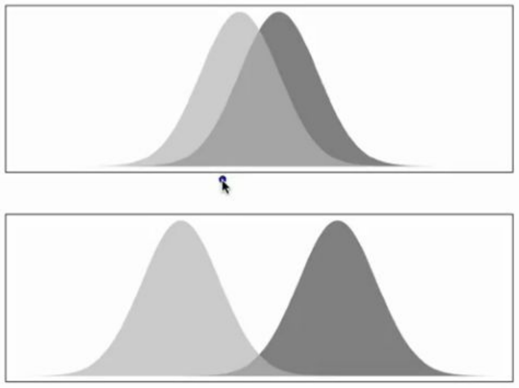
Prediction

- With a prediction study, you have slightly different issues than in an inferential study.



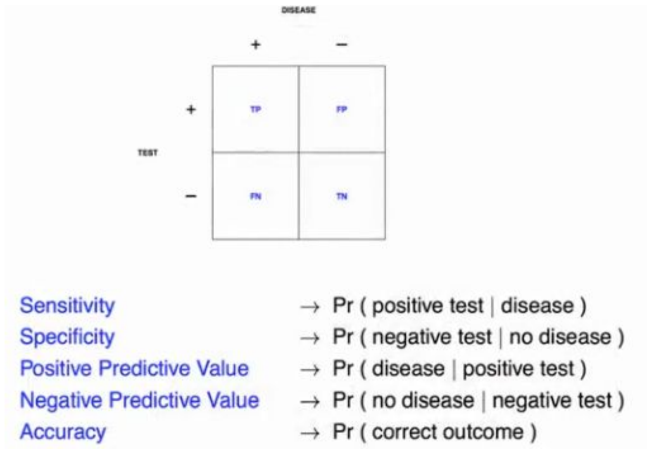
- You have a population for which you may not be able to collect <u>all</u> the data - but you want to predict something about them.
    - Say we have individuals come in and we measure something about their genome, to predict whether they'll respond to chemotherapy or not.
- We might collect a <u>training set</u> of observations from people who did respond to chemo, and a training set from people who didn't.
- Then, we want to build a predictive function that can be used to predict whether a new individual will respond to chemo or not.
- We're still dealing with probability and sampling and potential confounding variables, because we want this predictive function to be highly accurate.

Prediction vs. inference



- Prediction is slightly more challenging than inference.
    - In the top chart, you can see a difference in the P values (the means - or midpoints) of the two populations. But if I told you I observed a value that falls into both populations, it'd be very difficult to tell which one it falls under. It's relatively likely that it came from the light gray one, but also from the dark population.
    - For prediction (bottom chart), you actually need the distributions to be a bit more separated. These two distributions also have a different mean, but they're far enough apart (based on variability) that an answer is clearer.
- *It's important to pay attention to the relative size of effects when considering prediction vs. inference.*

Key quantities to be aware of in prediction



Sensitivity	→ Pr (positive test | disease)
Specificity	→ Pr (negative test | no disease)
Positive Predictive Value	→ Pr (disease | positive test)
Negative Predictive Value	→ Pr (no disease | negative test)
Accuracy	→ Pr (correct outcome)

- Positive and negative statuses
    - Example at right is from a medical testing scenario - +/- at top reflects those who have the disease; on the side, it reflects people whose tests come back positive.
        - <u>True positive</u> - test and disease positive
        - <u>False positive</u> - test positive, disease negative
        - <u>False negative</u> - test negative, disease positive
        - <u>True negative</u> - test and disease negative.
    - Quantities worth paying attention to:

        - Sensitivity - probability of a positive test, given that you have the disease
        - Specificity - probability of a negative test, given that you do not have the disease
        - Positive predictive value - probability you have disease, given a positive test *(this is what you're most likely to care about if you come into the clinic)*
        - Negative predictive value - probability you do not have disease, given a negative test
        - Accuracy - probability of a correct outcome

Beware of data dredging

- This would be if you found no link between two things you're testing, but continued to adjust your hypothesis and tried a bunch of different things until you got a result close enough to what you're looking for.

In summary

- Good experiments:
    - Have replication so you can measure variability
    - Measure variability and compare it to the signal being looked for
    - Generalize to the problem you care about
    - Are transparent
- Prediction is not inference
    - Both can be important
- Beware data dredging.