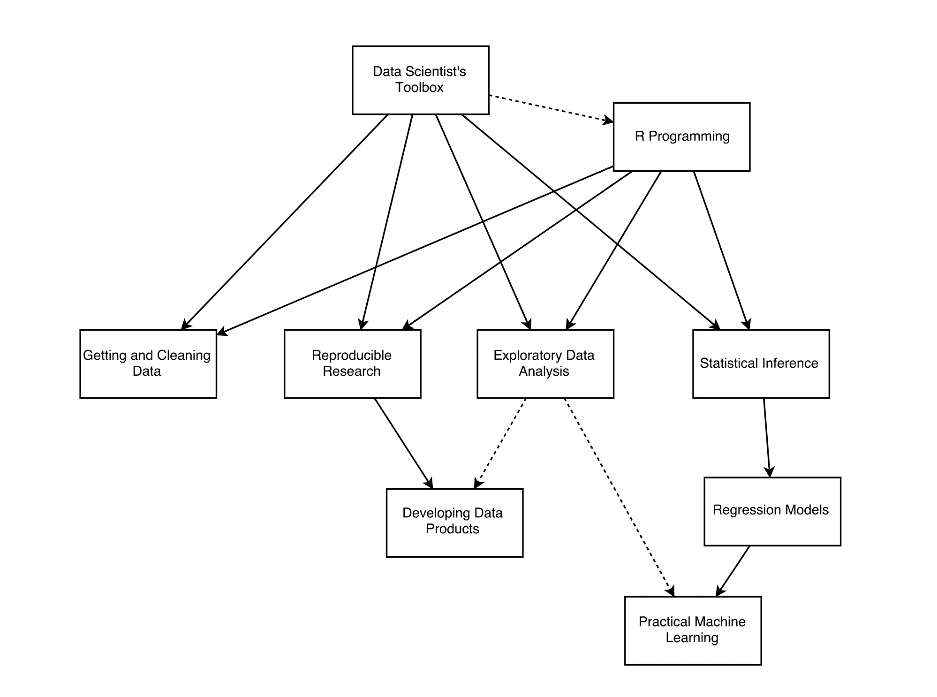
DATA SPECIALIZATION

**Link**: <https://www.coursera.org>

**Overview**: This is the first course in the Data Science Specialization, and can serve as your first step on your path to working in the field. In this course, you will get an introduction to the main tools and ideas in the data scientist's toolbox. The course gives an overview of the data, questions, and tools that data analysts and data scientists work with. There are two components to this course. The first is a conceptual introduction to the ideas behind turning data into actionable knowledge. The second is a practical introduction to the tools that will be used in the program like version control, markdown, git, Github, R, and Rstudio.

**Objective**: Ask the right questions, manipulate data sets, and create visualizations to communicate results.

This Specialization covers the concepts and tools you'll need throughout the entire data science pipeline, from asking the right kinds of questions to making inferences and publishing results. In the final Capstone Project, you’ll apply the skills learned by building a data product using real-world data. At completion, students will have a portfolio demonstrating their mastery of the material.

**Who:**

Jeff Leek – professor JHU School of Public Health. Statistics of genetic data.

* <http://biostat.jhsph.edu/~jleek/>, <http://simplystatistics.org/>, <https://github.com/jtleek>

Roger Peng

* <http://www.biostat.jhsph.edu/~rpeng/>, http://simplystatistics.org/

Brian Caffo – statistics to analyze brain image data.

* [www.bcaffo.com](http://www.bcaffo.com), <https://githup.com/bcaffo>

**Plagiarism**

Johns Hopkins University defines plagiarism as "...taking for one's own use the words, ideas, concepts or data of another without proper attribution. Plagiarism includes both direct use or paraphrasing of the words, thoughts, or concepts of another without proper attribution." We take plagiarism very seriously, as does Johns Hopkins University.

We recognize that many students may not have a clear understanding of what plagiarism is or why it is wrong. Please see the JHU referencing guide for more information on plagiarism.

It is critically important that you give people/sources credit when you use their words or ideas. If you do not give proper credit -- particularly when quoting directly from a source -- you violate the trust of your fellow students.

The Coursera Honor code includes an explicit statement about plagiarism:

I will register for only one account. My answers to homework, quizzes and exams will be my own work (except for assignments that explicitly permit collaboration). I will not make solutions to homework, quizzes or exams available to anyone else. This includes both solutions written by me, as well as any official solutions provided by the course staff. I will not engage in any other activities that will dishonestly improve my results or dishonestly improve/hurt the results of others.

**Reporting plagiarism on course projects**

One of the criteria in the project rubric focuses on plagiarism. Keep in mind that some components of the projects will be very similar across terms and so answers that appear similar may be honest coincidences. However, we would appreciate if you do a basic check for obvious plagiarism and report it during your peer assessment phase. It is currently very difficult to prove or disprove a charge of plagiarism in the MOOC peer assessment setting. We are not in a position to evaluate whether or not a submission actually constitutes plagiarism, and we will not be able to entertain appeals or to alter any grades that have been assigned through the peer evaluation system. But if you take the time to report suspected plagiarism, this will help us to understand the extent of the problem and work with Coursera to address critical issues with the current system.

# The Data Scientist’s Toolbox

**Course Description:**

In this course you will get an introduction to the main tools and ideas in the data scientists toolbox. The course gives an overview of the data, questions, and tools that data analysts and data scientists work with. There are two components to this course. The first is a conceptual introduction to the ideas behind turning data into actionable knowledge. The second is a practical introduction to the tools that will be used in the program like version control, markdown, git, Github, R, and Rstudio.

This course focuses primarily on getting you set up with the appropriate tools and accounts you will need for the rest of the specialization and on giving you a solid grounding in the key conceptual ideas. If you feel like the material is basic, that is ok, you will see much more in depth treatment of each topic in the subsequent courses in the track.

Course Content:

* Track motivation
* Getting help
* Introduction to basic tools
* R
* Rstudio
* Git
* Github
* Types of data questions
* Steps in a data analysis
* Putting the science in data science

Weekly quizzes

* There are three weekly quizzes.
* You must earn a grade of at least 80% to pass a quiz
* You may attempt each quiz up to 3 times in 8 hours.
* The score from your most successful attempt will count toward your final grade.

The Course Project

In the Course Project, you will demonstrate that you've set up all of the necessary accounts for the tools we'll be using.

You are required to evaluate and grade at least four of your classmates' projects. For this course, the project can be evaluated with a series of yes/no answers to determine whether people completed the required installations.

Part of the course project includes submitting a screenshot to demonstrate you have installed the relevant software. Be sure not to take a screenshot with other applications open that may reveal personal information or anything else you don't want others to see.

Grading policy

You must score at least 80% on all assignments (Quizzes & Project) to pass the course.

Your final grade will be calculated as follows:

* Quiz 1 = 20%
* Quiz 2 = 20%
* Quiz 3 = 20%
* Course project = 40%

Differences of opinion

Keep in mind that currently data analysis is as much art as it is science - so we may have a difference of opinion - and that is ok! Please refrain from angry, sarcastic, or abusive comments on the message boards. Our goal is to create a supportive community that helps the learning of all students, from the most advanced to those who are just seeing this material for the first time.

## Week 1: Welcome

* TR: it’s not the critic who points out the problems, but the doer, no effort without error.
* Challenge is to
* When have you ever solved a problem with all the information in advance, a surplus of information, and insufficient time and capacity to find it.
* Answer questions with data.
* The key is the ***science*** not the data.
* Data deluge: cheap to collect, cheap to store, free tools to do stuff with it.
* Big Data: not well defined. Mostly now have data that we didn’t have traditionally (GPS points from all cars, genetic information, online activity, etc.)
* Statistics – science of learning from data.
* Science – there is no certainty.
* Expected explosive growth in Data…great opportunity.
* Why JH likes R: very common language for data science.
  + Free
  + Comprehensive data packages (data access, cleaning, analysis, reporting)
  + Development environment
* Who is a data scientist:
  + Daryl Morey: Huston Rockets GM (data scientist)
  + Hilary Mason: Excel Partners – data mining and social media
  + Daphne Koller: CEO Coursera – improve educational delivery and assessment
  + Nate Silver: used free public data to predict elections.
* Skill sets:
  + Hacking Skills, Substantive Experience, Math and Statistics Knowledge
  + Self-Learning: the answers aren’t captured in traditional sources

**The Data Scientists Toolbox**

* Define question
* Define ideal data set
* Determine what data you can access
* Obtain the data
* Clean the data
* Explortory data analysis
* Statistical prediction/modeling
* Interepret results
* Challenge results
* Synthesize/write up results
* Create reproducible code: R markdown documents help this (.rmd)

**Tools (for this course)**

* R – widely used and supported by developer community
* RStudio: IDE for R

**Getting Help**

* MOOC: massive open online class – 100k people
* Post questions to the message boards
* Try to answer questions yourself
* R functions: ?rnorm help.search(“rnorm”) args(“rnorm”)
* Just type the function: rnorm gives the code
* R functions: <http://cran.r-project.org/doc/contrib/Short-refcard.pdf>
* How to ask an R question:
  + What steps will reproduce the problem
  + What was your expected output
  + What did you see instead
  + What version of product
  + What operating system
* How to ask a data question
  + What question you trying to answer
  + What steps/tools did you use
  + What did you expect
  + What did you see instead
  + What other solutions did you think about
* Be specific in your message title
* Use forums rather than email, be polite, be explicit, provide info, repost answer when found.
* Use R forums for R, use Coursera forums for coursera

**Finding Answers**

* Critical skills for data science
* Hackers:
  + Willing to find answers on own
  + Knowledgeable about where to find answers
  + Unintimidated by new data types or packages
  + Unafraid to say they don’t know the answer
  + ***Polite but relentless***
* R Programming:
  + <http://bit.ly/Ufaadn>
  + Help manual
  + Search web
  + Post to class forums
* Data questions
  + Search archives
  + Stackoverflow tag [r]
  + CrossValidated for general questions
  + Google: [data type] data analysis or [data type] R package
  + Include Type of analysis

**Overview lectures:**

* R
* Data Overview (sql, merge, raw processed, processed data)
* Exploratory Analysis (analytic graphics, R plotting tools, quicker analysis tools, K-means)
* Reproducible Research (create code and documents that transparently reproduces research so others can verify it)
* Steps in Data Analysis: (11 step version)
* Statistical Inference:
* Regression Models – widely used tool for any analysis
* Practical Machine Learning – use R with machine learning.
* Building Data Products (building R packages, interactive charts: rCharts, interactive apps: Slidify, Shiny)
* Install R on Windows.
* Install R Studio on windows.

## Week 2: Installing the Toolbox

***Tips from Coursera Users***

* ***Time***: Preparation, lectures, constant attention
* ***Difficulty***: Hard courses
* ***Technology*** intimidation
* Take notes
* Re-watch the videos
* Peer review and forums – communicate with peers
* TA assistance

**Command Line Interface**

* Windows: Git Bash
* Max/Linux: Terminal
* Uses of CLI: navigate, create, edit files and folders; run programs
* Directory organization like a tree
* / - root directory, contains all other directories
* ~ - home directory (your user directory)
* You will see the prompt: ~ rich $
* “working directory” – current directory you are working in
* “path” – directory path; navigate from one directory to another
* $ pwd prints working directory
* CLI commands [command, flags, arguments]
  + Command is the command to do the task
  + Options to the command to trigger behaviors (starts with a “-“)
  + Arguments are passed to be modified or other options
  + There can be zero or more flags and args
* $ clear clear all commands and gives prompt
* $ ls –a list all hidden and unhidden files
* $ ls –al lists details for all hidden and unhidden files
* $ cd change directory to the one you want to visit
* $ mkdir make a directory (with argument name of directory)
* $ touch creates an empty file (with arg name of file)
* $ cp copy file FROM to TO args
* $ cp –r copy recursive contents from one directory to another directory
* $ rm remove files
* $ rm –r recursively remove all files in a directory ***CAUTION: there is no undelete.***
* $ mv move file from one directory to another
* $ mv move file to same directory with new name (like renaming)
* $ echo prints arguments provided
* $ date prints the date

**Intro to Git**

* Version control (or revision control): means of managing process of create, change, save
  + Critical when collaborating
  + Able to save intermediate versions (able to go back and reconstruct)
* Git (free and open source distributed version control system)
  + Developed by linux devs
  + Everything is stored in local repositories
  + Operated from command line
* Install Git
  + Git Bash is the terminal environment
  + $ git config –global user.name “Your Name Here”
  + $ git config – global user.email “[your@email.com](mailto:your@email.com)”
  + $ git config –list review the list of configuration changes
  + $ exit
* Git via Git: If you already have Git installed, you can get the latest development version via Git itself:

git clone https://github.com/git/git

* Checkout windows style, commit Unix style line endings
  + Git will convert LF to CRLF when checking out text files. When committing text files, CRLF will be converted to LF. For cross-platform projects, this is recommended option.

**Intro to GitHub**

* Allows collaboration using Git
* Web based hosting for software development
* Push/pull local repositories to remote repositories on web
* Backup on GitHub servers
* Social aspect (public projects)
* USE SAME EMAIL AS THE GIT SETUP
* Tutorials for Git and GitHub
* Edit profile – be able to take credit for it.

**Create a GitHub repository**

* GitHub allows: share with others, access other repos, store remote copies of local repos (backup)
* Two methods to start:
  + Start a new repo from scratch on GitHub
    - Click on add
    - Create a name and a description
    - Make public or private
    - Start with a README file
  + Fork another repo
    - Find the repo on GitHub and then hit the Fork button; creates a copy of the repo
    - Make a local copy with clone command
    - $ git clone https://github.com/userName/reponame.git
* Create a Local Copy
  + Open Get Bash
  + $ mkdir ~/new\_repo\_dir make dir
  + $ cd /new\_repo\_dir navigate to the new dir
  + $ git init inside the directory init Git functions
  + $ git remote add origin https://github.com/UserName/test-repo.git link to a repo on GitHub

**Basic Git Commands**

* add :: file to the index

$ git add . add all new files in current dir

$ git add –u updates tracking for files that were name changed or deleted

$ git add –A does both above

* commit :: file from index to the local repo

$ git commit –m “message” useful description message; commit to the local repo

* push :: file from local repo to the remote repo

$ git push push all changes from local repo to the remote repo

* Branch – copy of the repo to work independently in a different direction.

$ get checkout –b branchname create a branch

$ git branch see what branch you are on

$ git branch master switch to the master branch

* Pull requests – feature of GitHub; allows you to merge changes into other branch or repo

**Basic Markdown**

* Markdown – formatted text file that can be used by other

Create a file: readme.md (.md) is a markdown file

Start the header:

## This is a secondary heading

### This is a tertiary heading

Create unordered list: \* first item, \*

**Installing R Packages**

* Cran = comprehensive R archive network (CRAN)
* Bioconductor Project (used for many biological apps)
* R packages: extend basic functions of R
* In R:

a <- available.packages()

Head(rownames(a), 3) ## show names of first few packages

* Install R package: install.packages(“solidify”)
* Dependencies are included in the install
* Install multiple R packages: install.packages(c(“solidify”, “rtools”))
* In RStudio:
  + Tools – Install Packages – choose package
* From Bioconductor:

Source(<http://bioconductor.org/biocLite.R>)

biocLite() all the packages

biocLite(c(“GenomicFeatures”,”AnnotationDbi”) install packages in the character array

* Must load the packages to use them – dependencies are included

Library(ggplot2) Note: do not put package in quotes

Search() give all the functions part of the package

**Installing Rtools**

* Build packages in Windows
* Make sure you check to install the path box.
* Install DevTools R package:
  + Open RStudio; find.package(“devtools”) to see if it is there; install.packages(“devtools”)

Library(devtools)

Find\_rtools() TRUE it has been installed.

## Week 3: Conceptual Issues

The Week 3 lectures focus on conceptual issues behind study design and turning data into knowledge. If you have trouble or want to explore issues in more depth, please seek out answers on the forums. They are a great resource! If you happen to be a superstar who already gets it, please take the time to help your classmates by answering their questions as well. This is one of the best ways to practice using and explaining your skills to others. These are two of the key characteristics of excellent data scientists.

**Types of Data Science Questions** (in order of difficulty)

* Descriptive
  + Describe only, no decisions
  + Commonly applied to census data
  + Description and interpretation steps
  + Descriptions cannot be generalized without additional modeling
* Exploratory
  + Discover new connections (not confirm)
  + Define future studies
  + ***NOT THE FINAL SAY – correlation does not imply causation***
  + Do not use alone for final say (decisions)
  + Example: Sloan Digital Sky Survey (<http://www.sdss.org/>)
* Inferential
  + Use relatively small sample of data and say something about the population (extrapolation)
  + Common goal of statistics models
  + ***ESTIMATE BOTH QUANTITY OF INTEREST AND THE UNCERTAINTY***
  + Depends heavily on the method used
  + Example: air pollution control and life expectancy study for all counties
* Predictive
  + Use data to predict other values
  + ***If X predicts Y does not mean that X causes Y***
  + Accuracy depends heavily on using right variables
  + More data and simple models tend to work really well
  + Prediction is very hard, especially about the future.
  + Example: 538 blog (pres. Election)
  + Example: Target figured out a teen was pregnant and sent materials before dad knew
* Causal
  + Find out what happens to one variable when you make changes to another variable.
  + Usually randomized studies to identify causation
  + Inferring causation from non-randomized studies are difficult
  + Causal relationships identified as average effects, but may not apply to every individual
  + Gold Standard for analysis
  + Example: New England Journal of Medicine
* Mechanistic
  + Understand exact change in variables that lead to exact changes in other variables
  + Incredibly hard to infer
  + Usually modeled by deterministic set of equations (physical/engineering science)
  + Generally the random component of the data is measurement error
  + If equations are known, but parameters not then can be inferred with data analysis

**What is Data?**

* **Data** are values of quantitative or qualitative values belonging to a set of items.
* **Variables** are a measurement or characteristic of an item.
* **Qualitative:** do not necessarily order or measured.
* **Quantative**: have measure and order
* Data can be:
  + raw values (structured)
  + API that returns blocks of structured data
  + Text files, more or less structured in text files or others
  + Video/photo/audio
* Data not usually neat and ready to process
* Question is the start, most important. The data comes second… it cannot save you without a good question.

**What About Big Data?**

* Worlds data is doubling every 2 years, but only a small fraction is available for analysis.
* Cloud is synomous due to size of data sets
* Hardware is improving and helps solve certain problems (HD sizes, CPU speeds,…)
* Travers and Milgram Sociometry (1969) – “The Small World Problem”
  + About 5.2 links in chain between people during the experiment (6 degrees of separation)
* Leskovec and Horvitz WWW ‘08
  + Found 6.6 links between users on messanger service
* You need the right data – no matter how big the data are.
  + Hadoop Hatred – not all data problems are “BIG”

**Experimental Design**

* Poor analysis leads to poor results – and high risks (cost, litigation, etc.)
* How to avoid:
  + Care about the analysis plan.
  + Plan for data and code sharing. (github, figshare)
    - <http://github.com/jtleek/datasharing> - guide for data sharing
  + Formulate question in advance
  + Address confounding (shoe size and literacy) – pay attention to other variables
* Deal with confounding
  + Randomization and Blocking (fix, fix and stratify, or randomize the variable)
* Prediction
  + Probability/sampling gives a training set --- determine a predictive function
  + Helps if the means are far apart – variability is distinct
  + Key quantities:
    - ***Sensitivity*** – Probability ( positive test | disease )
    - ***Specificity*** – Probability ( negative test | no disease )
    - ***Positive Predictive Value*** – Probability (disease | positive test )
    - ***Negative Predictive Value*** – Pr ( no disease | negative test )
    - ***Accuracy*** – Pr (correct outcome )
* Beware of Data Dredging
  + Keep going and keep digging until you get a “good” result.
  + Avoid with starting with good and valuable question.
* Good experiments have:
  + Have replication
  + Measure variability
  + Generalize to the problem you care about
  + Are transparent
* Prediction is not Inference
  + Both can be important
* Beware of:
  + Un-reviewed code or data
  + Confounding
  + Data dredging

## Week 4: Course Project Submissions and Evaluation

**Due NLT 28 Feb:**

The Course Project for this course is pretty straightforward. Now that you’ve learned about the basic tools found in the data scientist’s toolbox, we want to make sure that you’ve assembled those tools and are ready to begin using them. This easy project is your chance to demonstrate that you’ve done the basic software setup (R, RStudio, and Github) that will get you through the rest of the Data Science Specialization.

***Unit 1 Project submitted on 11 Feb 2016.***

***Grade:***

# R Programming

**Overview**: In this course you will learn how to program in R and how to use R for effective data analysis. You will learn how to install and configure software necessary for a statistical programming environment and describe generic programming language concepts as they are implemented in a high-level statistical language. The course covers practical issues in statistical computing which includes programming in R, reading data into R, accessing R packages, writing R functions, debugging, profiling R code, and organizing and commenting R code. Topics in statistical data analysis will provide working examples.

[Roger D. Peng](https://eventing.coursera.org/api/redirectStrict/sh9yI_2s80e7O66XnBXHIROq66DpXWH5xW-ov7MXnvb_asGwtusQDDGFmCmo7PmYQefmsBYkiGzG0QCdW6b5QQ.rQO5BW6nq6DhFYM2AHhN1Q.138ciG5RbTRESxdNGhBjFmA6G8quqaXFoOrsol5MbL1ItAT3FKEqnjyG2esxctmIDWHi9iEtyw4918TyrmqvQ-e-ikX4mdIyvYFeLzywBqgO0aUH8JNXxXuz8xJJYbHhbnMPw4SZqAddta-pVBa0wA2n_vHCPXKZbM2svizRtClxKJyxSJOSOXwNHT8Sy6jXqdGpTn6KThyhmRr4E_2S-hmPk1X_R0r_9rdjNF27V5t1GkBY2GFQ9Bxz_Ozi1uFN8npeA_Pby3hPjRRIdkYjpCh6WciqWFFc1iIIXP06USWH3a0xY74kYr_jn_W7zuO5tIzJUdrtyqqRBgDFgdozLQZjuLkpQ8jVxee_rrqxiSbQhAX5UcSQWdHaAJuxn4qb)

## Course Description

In this course you will learn how to program in R and how to use R for effective data analysis. You will learn how to install and configure software necessary for a statistical programming environment, discuss generic programming language concepts as they are implemented in a high-level statistical language. The course covers practical issues in statistical computing which includes programming in R, reading data into R, accessing R packages, writing R functions, debugging, and organizing and commenting R code. Topics in statistical data analysis and optimization will provide working examples.

Course Content

* Week 1: Overview of R, R data types and objects, reading and writing data
* Week 2: Control structures, functions, scoping rules, dates and times
* Week 3: Loop functions, debugging tools
* Week 4: Simulation, code profiling

Background lectures

Background lectures about the content of the course with respect to other quantitative courses, course logistics, and the R programming language are provided as reference material. It is not necessary to watch the videos to complete the course, however you may find them useful.

Course Textbook

The book [*R Programming for Data Science*](https://eventing.coursera.org/api/redirectStrict/JMQUBuB8hrmmdlqqAoFHkX4ujrNwuSpftf_qMjfr6b2Y7FIap3TPRqwSZnrff3H1qpt9TNsMumh_4laZ0DNMRg.lzIL9TMlQNDqOqP0a3Ds3w.IK8naJ673rYud2fDqVDhsEdW6-nKpPM-7ZiF7LmUfu71ORdHGCLInEKbKdYS57v2wRcPfTVpD9mArg0X-kQIRi4X2lbDuwLbKBqe7emBCioFMyWvFiRauqkhD8CeS_rEtYnXEJDaBvwh-bwVdnxtawir1avuaYCfxRX8d_vmZ6uwh2CXo0RHatiEIH7Ctu-i3swo6dGGeg2otM0pne0eVcvFiH6DyGF2FjEO7FpmYbxcIUQOcvSN9WpUxhJkQ40UgfRgSM-7Ro8jav3KPTBQ7GpUu5ZfhZpoVOoas3qQ_jI) covers all of the lecture material in this course.

Assessments

Quizzes

* There are four weekly quizzes.
* You must earn a grade of at least 80% to pass a quiz.
* You may attempt each quiz up to 3 times in 8 hours.
* The score from your most successful attempt will count toward your final grade.

Programming Assignments

* There are **three required** programming assignments.
* You must earn a grade of at least 80% to pass a programming assignment
* Programming Assignments 1 and 3 are graded via unit tests using a submission script that compares the output of your functions to the correct output.
* Programming Assignment 2 is submitted differently and graded via a peer review.

swirl Programming Assignment (practice)

* In this course, you have the option to use the [swirl R package](https://eventing.coursera.org/api/redirectStrict/OUy57uBsmUZqUCiqVi-PBQ0ndcUxly5qZpRxszUa9z3xjxHBcDHmGcLobzSeCasHN0pnblIWFtM4kHXJRcqr3Q.Ij1JfjlAYUr4w-OcAh7NBQ.vOBE9d_lbayzozGvinwimvjTk3flQ332Yu9IUPSFxAaGp98X5yr5xVGB78Qvf_pDhAV2cO6WO9lBZW4Gq5TsB8ApBSPsbjyxc9SxhZmpU0W5tJ0SLJ4Kk-GBL_vzDbWY5bSuVoKlmom7B88gE_qzp1PIKd5DzEnyAovV-enpu-6uj63DUl3zl9fiEah1awGdmAQrfFWOQ-dV3NyBAkAV4YcMXEc65fPIEsqHWybeYXEGh9w9Ws7vtV5G69Ow5tuz1Mmf6qNhrGXS94tDEn39bA) to practice some of the concepts we cover in lectures.
* While these lessons will give you valuable practice and you are encouraged to complete as many as possible, please note that they are **completely optional** and you can get full marks in the class without completing them.

Grading Policy

You must score at least 80% on all required assignments (Quizzes & Programming Assignments) to pass the course. Your final grade will be calculated as follows:

* Week 1 Quiz - 20%
* Week 2 Quiz - 10%
* Week 3 Quiz - 5%
* Week 4 Quiz - 10%
* Programming Assignment 1 (Air Pollution) - 20%
* Programming Assignment 2 (Lexical Scoping) - 10%
* Programming Assignment 3 (Hospital Quality) - 25%
* swirl Programming Assignment (practice) - 0%

## Background

* Welcome to R Programming
* About the Instructor
* Pre-Course Survey
* Syllabus
* Course Textbook
* Course Supplement: The Art of Data Science
* Data Science Podcast: Not So Standard Deviations

### Installing R on a Mac / Installing R on Windows

* Comprehensive R Archive Network - authoritative base site for R
* cran.rstudio.com

### Installing R Studio (Mac) / Installing R on Windows

1. Must already have R installed.
2. <http://www.rstudio.com>

### Writing Code / Setting Your Working Directory (Windows)

getwd() get the working directory

dir() list files in this directory

ls() list

source(“name”) source a bit of R code from file in the working directory

* Set default working directory:

C:\Users\rich\Desktop\NEXT\course - coursera data specialization\r\_projects

* Write code using the text editor in R (file – new script)

## Week 1: Getting Started and R Nuts and Bolts

### Introduction

This week is all about getting started with R and learning some of the basic details of the language. If you haven't already installed R, you should go to the [R web site](https://eventing.coursera.org/api/redirectStrict/6laHBfqn0fVSptNW5hDijoP9SAWI07xFZL05Cjd7IGKK1WEmTXT6hlFqmgTF_9MxF046qFsoSia71f_owEvWFw.9g71FBByheDpsmhvo8bV_A.lyxJCKqLT7E1dahc3OouUT1a8FQKBQFdd_sfA7d6WbyMXHOZd1b6fhLBVto-IWOgPfD34h1L_PIZTxdqWubv-9hScKiHkLEqow5dom7g33BTudGk8ZAPVOLNxxW58N_wLfrA7kyjnXFEZnA3lsjxoJYhcqoT9Byf-YGsmBF6N8CkD4W3wNixPIetK-VIDuTyHG6m4ly5YZAo6-RZnVNjNwNkiJK4awqRBgluGYPY4nn3JkNTILrGKotxdEUqmr0Huj8qrvXpvZ1-vK_iN89J9A) and download R for your platform (Windows, Mac, or Unix/Linux). Also, if you want, you can download [RStudio](https://eventing.coursera.org/api/redirectStrict/lk41lGyA5zcpR8RK8uygGdDFSrj3dpaUXJu_rGpSsIai8-BcxDq_v_UWf6tkYMtDFmWWs8yN8wHEVBQULFg5CA.tVdkFQY9GhYOcBg6x8iGvw.GTO395k9kAoNsPXmfWxAoIVQ6nAEn3fZr3H1U5ucwMup4O1ijGVsdQQt4DWCELJ8olkkV9ghcC8j2we-LUDygHR_I81xAjLvGLWkze_O3Hm4bQOb3MsP6IvHLgWjTzllDPrs0OgXgvhUhZQ2_dye4cNYYzWUl6w-T8DGLWyxG0CWcs1rrsrclPgklYU7KUfM3ejVU6f-BJU3M0qTT4IvXZ0QWPe0dp5p-cSES0K8tsK0TwS29mfuIZvO3Y4u8kBvkeYVfNMGBBnXgmq82Ft3ZA), which is a free interactive development environment designed for R that is very useful and we use quite a bit in the Data Science Specialization. I've made some videos to help you along with the installation process:

* Installing R on Windows
* Installing R on a Mac
* Installing R on RStudio (on a Mac)

Before you start using R, one key concept is the **working directory**. This is the directory/folder on your computer where you will store project files, data, and code. It's important that you tell R where the working directory is that you will be using so that it knows where to find the appropriate file (the working directory can be any directory on your computer). These videos tell you how to set your working directory:

* Setting your working directory (Windows)
* Setting your working directory (Mac)

**Learning Objectives**

By the end of week 1 you should be able to:

* Install the R and RStudio software packages
* Download and install the swirl package for R
* Describe the history of the S and R programming lectures
* Describe the differences between atomic data types
* Execute basic arithmetic operations
* Subset R objects using the "[", "[[", and "$" operators and logical vectors
* Describe the explicit coercion feature of R
* Remove missing (NA) values from a vector

**Assessments**

* Quiz 1 - 80% or better required to pass
* There is **no official graded programming assignment for this week**. However, we have developed a series of practice exercises to get you started with R. These exercises are implemented using the swirl package for R. **The swirl programming assignment is NOT required**.

### Overview and History of R

* R is a dialectic of S
  + S was developed at Bell Labs as an internal stats analysis environment
  + 1988 rewritten in C
  + Statistical Models in S – “the white book” by Chambers and Hastings
  + Programming in Data – “the green book”
  + Since 2008 TIBCO continues to develop S-PLUS
  + Goal: interactive environment that user is not consciously programming and allow clearer and more sophisticated tools to be built later. Easy to go from user to developer.
* R developed starting in 1991 (released 1993)
  + Used GNU GPL in 1995
  + 1997 R Core Group controls the source code (some associates of S-PLUS
  + Runs on any standard computing platform/OS
  + Frequent releases and bug development
  + Lean and package oriented
  + Good graphics capabilities and control
* Free Software (<http://www.fsf.org> )
  + Freedom 0: run the program, for any purpose
  + Freedom 1: study how program works and adapt to your needs, access to the source code required.
  + Freedom 2: redistribute copies so you can help your neighbor
  + Freedom 3: improve the program, release improvements to the public, must make source code available.
* Drawbacks to R
  + Based on 40 year old technology
  + Not much support for dynamic or 3D graphics
  + Contributions based on user demand and contributions (you need to build functions)
  + Objects must be stored in physical memory (some advances)
  + Not ideal for all situations
* Design of R
  + Base R system download from CRAN
  + Everything else
  + Packages in base system: list
  + Recommended packages: list
  + About 4000 packages on CRAN – quality standards
  + Bioconductor project (<http://bioconductor.org>)
  + Personal packages on all sorts of websites
* R Resources
  + An Introduction to R
  + Writing R Extensions - use if writing packages
  + R Data Inport/Export
  + R Installation and Administration (build R from source)
  + R internals (not for faint of heart)
* R Books
  + <http://www.r-project.org/doc/bib/R-books.html>

### Getting Help

* Search the class forums (archives)
* Search the web
* Read the manual
* Read a FAQ
* Use inspection or experimentation
* Ask a skilled friend
* Read the source code
* Google the error statements

***When you go to email use this tricks:***

* Ensure you note you read the manual (or searched)
* Describe the goal not just the steps (find alternatives)
* What steps will reproduce the problem
* What is expected output – what do you see instead
* What version of product you are using – what OS
* Put the problem directly in the subject header
* Follow up your question with the solution (for the archive)

### R Console Input and Evaluation

* At the R prompt we type expressions
* <- symbol for assignment operator
* print(x)
* msg <- “hello”
* ## comments
* *Auto printing:* default print the value of object listed on line
* The [1] shows which element of the vector
* X <- 1:20 assigns a series of values to x as a vector with 20 int values

### Data Types - R Objects and Attributes

* Everything is an object
* 5 atomic classes:
  + ***Character***
  + ***Numeric*** (real numbers)
  + ***Integer***
  + ***Complex***
  + ***Logical*** (True/False)
* ***Vector*** – most basic object
  + Only same class objects in the vector
  + ***List*** represented as a vector but may have mixed class objects
  + Vector () creates a new vector - pass class and number in vector.
* All numbers are real unless explicitly cast
  + 1L gives an int value
  + Inf represents infinity where 1/0 = Inf 1/Inf = 0
  + NaN represents undefined value such as 0/0 or a missing value.
* R object attributes
  + Names, dimnames
  + Dimensions (matrices and arrays)
  + Class
  + Length
  + Other user defined attribute/metadata
  + attributes() function to access the attributes to set

### Data Types – Vectors and Lists

* c() creates vectors and objects, used to concatenate
* 1+0i and 2+4i complex numbers
* X <- vector(“numeric”, length = 10) set empty numeric vector size 10
* Mixing objects won’t throw an error due to implied coercion:
  + The coersion rule goes logical -> integer -> numeric -> complex -> character.
* Casting types: as.numeric(x) as.logical(x) as.character(x)
* class(x) gives the class type of a vector
* when casting doesn’t work, then NA assigned
* Lists: special type of vector that can contain different classes.

### Data Types – Matrices

* Matrices are vectors with a dimension attribute (nrow, ncol)
* dim(m) returns the rows and cols
* matrices are created column wise, start upper left and then move down then right.
* Create a matrix from a vector by adding the dimension attribute
* Create a matrix by col or row binding
  + cbind(x)
  + rbind(x)

### Data Types – Factors

* special type of data type used to store categorical data.
* Can be ordered or unordered.
* Factors are treated special by functions lm() and glm()
* Good to use since “self-subscribing” - not trying to set a code to interpret values or dip switch
* x <- factor(c(“yes”, “yes”, “no”, “yes”, “no”)) levels are yes/no
* table(x) gives the tallies for each level
* R uses baseline level due to alphabetical order: so use levels I the factor creation
* factor(c(“yes”, “yes”, “no”, “yes”, “no”), levels = c(“yes”, “no”))

### Data Types - Missing Values

* Denoted by NA or NaN for undefined mathematical operations
* is.na() used to test for NA - will return true for NaN
* is.nan() used to test for NaN - will return false for NA
* they return a logical vector to describe where the missing values are

### Data Types - Data Frames

* used to store tabular data
* special type of list where every element has the same length
* each element can be a column with different type
* matrices – must store same type of object in each column
* row.names attribute to describe data
* read.table() or read.csv() to create data frame
* convert to a matrix (if all same data) data.matrix()
* nrow(x) and ncol(x)

### Data Types - Names Attribute

* R objects can have names – good for readable code and self-describing objects
* names(x) <- c(“foo”, “bar”, “norf”)
* x <- list(a=1, b=2, c=3) where a,b,c are the names
* dimnames(m) <- list(c(“a”,”b”,”c”), c(“c”,”d”)) where row names a,b,c and col names c,d

### Data Types – Summary

* atomic classes: numeric, logical, character, integer, complex
* vectors, lists
* factors
* missing values
* data frames
* names

### Reading Tabular Data

* principle functions
  + read.table and read.csv read text files of tabular data
  + readlines read lines of text
  + source read R code
  + dget read in R code that is deparsed
  + load read binary objects
  + serialized single R objects in binary fom
* analaglous functions to write data
  + write.table
  + write.lines
  + dump
  + dput
  + save
  + serialized
* read.table most common
  + automatically skip lines, figure out how many rows, type of variables
  + read.table default separator is a space
  + read.csv is identical except the default separator is a comma; header is always true

### Reading Large Tables

* larger data sets use read.table
* calculate the size of available RAM
* set comment.char = “ “ if there are no commented lines in file
* read read.table help page
* setting colClasses argument is important (tell it ahead of time, won’t have to calculate)
* Find windows tool to tell number of lines in data
* Setting nrows argument to keep track of memory usage
* Know the System
  + How much memory (RAM)
  + What other apps in use
  + Are other users logged in
  + What OS
  + Is the OS 32- or 64-bit
* Calculate Memory Reqs
  + NROWS x NCOLS x (8 bytes/numeric) / 2^20 bytes/MB
  + 2^10 MB per GB
  + ***Overhead is about twice as much memory to read in the data frame***

### Textual Data Formats

* dumping and dputing are useful for text formats that is edit-able
* preserves the metadata versus csv or table file
* Textual formats are useful in use with version control programs
* Sometimes easier to fix corruption problems (you can see what happened)
* Textual formats adhere to the “Unix philosophy” – store all kinds of data in text
* Not very space efficient – probably need to be compressed

### Connections: Interfaces to the Outside World

* Data are read in using connection interfaces
* File – opens a connection file
* gzfile – open a connection to compressed gzip
* bzfile – open connection to file compressed with bzip2
* url – open connection to webpage
* readlines(con) can read lines from the connection….

### Subsetting – Basics

* extract subsets of objects in R
* [ always returns object of same class as original
* [[ extract elements of list or data frame – returns object may not be a list or data frame
* $ extract elements of list or data frame by name
* Use a numeric index (1,2,3….)
* Use a logical index (condition to test to get those elements
* Use a logical test for every element (TRUE, FALSE, etc) for each element.

### Subsetting – Lists

* Double bracket or single bracket
* Single – return same class as the original
* Double – sequence only
* Subset an element with its NAME, don’t have to remember where it is – use the NAME to get the element
* To extract multiple elements from the list – must use single bracket operator
* Double bracket – can return a computed index (variable)
* Single bracket or $ must use a literal symbol (no computed)

### Subsetting – Matrices

* Row then col index values
* Subset a single element of a matrix = you get back a vector with length one
  + Turn off the default with drop = FALSE - where drop means drop the dimension of object

### Subsetting - Partial Matching

* Partial matching of names is allowed with [[ and $
  + Set exact = FALSE, which allows partial matching

### Subsetting - Removing Missing Values

* Common to remove the NA values
* For vector, matrix, or dataframe
* Create a logical vector that shows where the missing data are located
  + bad <- is.na(x)
  + x [ !bad ] returns the values that are not missing (not NA’s)
* For vectors, use the complete.cases(x,y)
  + good <- complete.cases( x, y )
  + x[good]
  + y[good]
* complete.cases() also used on matrices

### Vectorized Operations

* vectorized operations are good for use on command line without use of too much looping
* operations are vectorized on the vector values (x) and don’t have to loop through.
* Vectorized Matrices
  + X \* y becomes an element multiplication instead of Matrix math.
  + X %\*% y is the matrix multiplication

## Practical R Exercises in swirl

**Introduction to swirl**

* Statistics With R Learning
* Optional for the course.
* Install swirl: install.packages(“swirl”)
* Check version: packageVersion(“swirl”)
* Load swirl: library(swirl)
* Install R Programming course in swirl: install\_from\_swirl(“R Programming”)
* Start swirl: swirl()

| You can exit swirl and return to the R prompt (>) at any

| time by pressing the Esc key. If you are already at the

| prompt, type bye() to exit and save your progress. When you

| exit properly, you'll see a short message letting you know

| you've done so.

| When you are at the R prompt (>):

| -- Typing skip() allows you to skip the current question.

| -- Typing play() lets you experiment with R on your own;

| swirl will ignore what you do...

| -- UNTIL you type nxt() which will regain swirl's attention.

| -- Typing bye() causes swirl to exit. Your progress will be

| saved.

| -- Typing main() returns you to swirl's main menu.

| -- Typing info() displays these options again.

| Hi! I see that you have some variables saved in your workspace. To keep things running smoothly, I recommend you clean up before starting swirl.

| Type ls() to see a list of the variables in your workspace. Then, type

| rm(list=ls()) to clear your workspace.

| Type swirl() when you are ready to begin.

### Assignment: swirl Lesson 1: Basic Building Blocks

* R is a programming language (not a calculator) which enables automating complicated and repetitious tasks
* Assignment to a variable
* Data structure is any object that contains data; Vector is a collection of numbers

| If at any point you'd like more information on a particular

| topic related to R, you can type help.start() at the prompt,

| which will open a menu of resources (either within RStudio

| or your default web browser, depending on your setup).

| Alternatively, a simple web search often yields the answer

| you're looking for.

The easiest way to create a vector is with the c() function,

| which stands for 'concatenate' or 'combine'. To create a

| vector containing the numbers 1.1, 9, and 3.14, type c(1.1,

| 9, 3.14)

| Other common arithmetic operators are `+`, `-`, `/`, and `^`

| (where x^2 means 'x squared'). To take the square root, use

| the sqrt() function and to take the absolute value, use the

| abs() function.

| When given two vectors of the same length, R simply performs

| the specified arithmetic operation (`+`, `-`, `\*`, etc.)

| element-by-element. If the vectors are of different lengths,

| R 'recycles' the shorter vector until it is the same length

| as the longer vector.

| When we did z \* 2 + 100 in our earlier example, z was a

| vector of length 3, but technically 2 and 100 are each

| vectors of length 1.

| Behind the scenes, R is 'recycling' the 2 to make a vector

| of 2s and the 100 to make a vector of 100s. In other words,

| when you ask R to compute z \* 2 + 100, what it really

| computes is this: z \* c(2, 2, 2) + c(100, 100, 100).

| To see another example of how this vector 'recycling' works,

| try adding c(1, 2, 3, 4) and c(0, 10). Don't worry about

| saving the result in a new variable.

### Assignment: swirl Lesson 2: Workspace and Files

* R has common API (common set of commands) for workspace and file management.
* It is often helpful to save the settings that you had before you began an analysis and then go back to them at the end.

| Determine which directory your R session is using as its

| current working directory using getwd().

| List all the objects in your local workspace using ls().

| Some R commands are the same as their equivalents commands

| on Linux or on a Mac. Both Linux and Mac operating systems

| are based on an operating system called Unix. It's always a

| good idea to learn more about Unix!

| List all the files in your working directory using

| list.files() or dir().

| As we go through this lesson, you should be examining the

| help page for each new function. Check out the help page for

| list.files with the command ?list.files.

| Using the args() function on a function name is also a handy

| way to see what arguments a function can take.

| Use dir.create() to create a directory in the current

| working directory called "testdir".

| Set your working directory to "testdir" with the setwd()

| command.

| In general, you will want your working directory to be

| someplace sensible, perhaps created for the specific project

| that you are working on. In fact, organizing your work in R

| packages using RStudio is an excellent option. Check out

| RStudio at http://www.rstudio.com/

| Create a file in your working directory called "mytest.R"

| using the file.create() function.

| Check to see if "mytest.R" exists in the working directory

| using the file.exists() function.

| Access information about the file "mytest.R" by using

| file.info().

| Change the name of the file "mytest.R" to "mytest2.R" by

| using file.rename().

You might now try to delete

| mytest.R using file.remove('mytest.R'), but that won't work

| since mytest.R no longer exists. You have already renamed

| it.

| Make a copy of "mytest2.R" called "mytest3.R" using

| file.copy().

| You now have two files in the current directory. That may

| not seem very interesting. But what if you were working with

| dozens, or millions, of individual files? In that case,

| being able to programatically act on many files would be

| absolutely necessary. Don't forget that you can,

| temporarily, leave the lesson by typing play() and then

| return by typing nxt().

| Provide the relative path to the file "mytest3.R" by using

| file.path().

| Create a directory in the current working directory called

| "testdir2" and a subdirectory for it called "testdir3", all

| in one command by using dir.create() and file.path().

> dir.create(file.path("testdir2", "testdir3"), recursive = TRUE)

| To delete a directory you need to use the recursive = TRUE

| argument with the function unlink(). If you don't use

| recursive = TRUE, R is concerned that you're unaware that

| you're deleting a directory and all of its contents. R

| reasons that, if you don't specify that recursive equals

| TRUE, you don't know that something is in the directory

| you're trying to delete. R tries to prevent you from making

| a mistake.

| Delete the "testdir2" directory that you created by using

| unlink().

> unlink("testdir2", recursive = TRUE)

| Why is this command named "unlink" rather than something

| more sensible like "dir.delete" or "dir.remove"? Mainly,

| history. unlink is the traditional Unix command for removing

| directories.

| Go back to your original working directory using setwd().

| (Recall that we created the variable old.dir with the full

| path for the orginal working directory at the start of these

| questions.)

| Take nothing but results. Leave nothing but assumptions.

| That sounds like 'Take nothing but pictures. Leave nothing

| but footprints.' But it makes no sense! Surely our readers

| can come up with a better motto . . .

### Assignment: swirl Lesson 3: Sequences of Numbers

* Sequence of numbers useful in many R tasks

| The simplest way to create a sequence of numbers in R is by

| using the `:` operator. Type 1:20 to see how it works.

| Remember that if you have questions about a particular R

| function, you can access its documentation with a question

| mark followed by the function name: ?function\_name\_here.

| However, in the case of an operator like the colon used

| above, you must enclose the symbol in backticks like this:

| ?`:`. (NOTE: The backtick (`) key is generally located in

| the top left corner of a keyboard, above the Tab key. If you

| don't have a backtick key, you can use regular quotes.)

| This gives us the same output as 1:20. However, let's say

| that instead we want a vector of numbers ranging from 0 to

| 10, incremented by 0.5. seq(0, 10, by=0.5) does just that.

| Try it out.

| Or maybe we don't care what the increment is and we just

| want a sequence of 30 numbers between 5 and 10. seq(5, 10,

| length=30) does the trick. Give it a shot now and store the

| result in a new variable called my\_seq.

> my\_seq <- seq(5, 10, length=30)

| To confirm that my\_seq has length 30, we can use the

| length() function. Try it now.

| There are several ways we could do this. One possibility is

| to combine the `:` operator and the length() function like

| this: 1:length(my\_seq). Give that a try.

> 1:length(my\_seq)

[1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19

[20] 20 21 22 23 24 25 26 27 28 29 30

| Another option is to use seq(along.with = my\_seq). Give that

| a try.

> seq(along.with = my\_seq)

[1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19

[20] 20 21 22 23 24 25 26 27 28 29 30

R has a

| separate built-in function for this purpose called

| seq\_along(). Type seq\_along(my\_seq) to see it in action.

> seq\_along(my\_seq)

[1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19

[20] 20 21 22 23 24 25 26 27 28 29 30

rep(), which stands for 'replicate'. Let's look at a few

| uses.

| If we're interested in creating a vector that contains 40

| zeros, we can use rep(0, times = 40). Try it out.

| If instead we want our vector to contain 10 repetitions of

| the vector (0, 1, 2), we can do rep(c(0, 1, 2), times = 10).

| Go ahead.

let's say that rather than repeating the vector (0,

| 1, 2) over and over again, we want our vector to contain 10

| zeros, then 10 ones, then 10 twos. We can do this with the

| `each` argument. Try rep(c(0, 1, 2), each = 10).

### Assignment: swirl Lesson 4: Vectors

* The simplest and most common data structure in R is the vector.
* Vectors come in two different flavors: atomic vectors and lists. An atomic vector contains exactly one data type, whereas a list may contain multiple data types
* Types of atomic vectors include numeric, logical, character, integer, and complex.
* Logical vectors can contain the values TRUE, FALSE, and NA (for 'not available'). These values are generated as the result of logical 'conditions'. Let's experiment with some simple conditions.

| The `<` and `>=` symbols in these examples are called

| 'logical operators'. Other logical operators include `>`,

| `<=`, `==` for exact equality, and `!=` for inequality.

| If we have two logical expressions, A and B, we can ask

| whether at least one is TRUE with A | B (logical 'or' a.k.a.

| 'union') or whether they are both TRUE with A & B (logical

| 'and' a.k.a. 'intersection'). Lastly, !A is the negation of

| A and is TRUE when A is FALSE and vice versa.

| Right now, my\_char is a character vector of length 3. Let's

| say we want to join the elements of my\_char together into

| one continuous character string (i.e. a character vector of

| length 1). We can do this using the paste() function.

Try paste(LETTERS, 1:4, sep = "-"), where

| LETTERS is a predefined variable in R containing a character

| vector of all 26 letters in the English alphabet.

### Assignment: swirl Lesson 5: Missing Values

* Missing values play an important role in statistics and data analysis. Often, missing values must not be ignored, but rather they should be carefully studied to see if there's an underlying pattern or cause for their missingness.
* The key takeaway is to be cautious when using logical expressions anytime NAs might creep in, since a single NA value can derail the entire thing.
* NA stands for Not Available
* NaN stands for Not a Number
* In R, Inf stands for infinity.

| In R, NA is used to represent any value that is 'not

| available' or 'missing' (in the statistical sense). In this

| lesson, we'll explore missing values further.

| Any operation involving NA generally yields NA as the

| result. To illustrate, let's create a vector c(44, NA, 5,

| NA) and assign it to a variable x.

| To make things a little more interesting, lets create a

| vector containing 1000 draws from a standard normal

| distribution with y <- rnorm(1000).

| Finally, let's select 100 elements at random from these 2000

| values (combining y and z) such that we don't know how many

| NAs we'll wind up with or what positions they'll occupy in

| our final vector -- my\_data <- sample(c(y, z), 100).

| The trick is to recognize that underneath the surface, R

| represents TRUE as the number 1 and FALSE as the number 0.

| Therefore, if we take the sum of a bunch of TRUEs and

| FALSEs, we get the total number of TRUEs.

Call the sum() function on my\_na

| to count the total number of TRUEs in my\_na, and thus the

| total number of NAs in my\_data.

| Now that we've got NAs down pat, let's look at a second type

| of missing value -- NaN, which stands for 'not a number'. To

| generate NaN, try dividing (using a forward slash) 0 by 0

| now.

### Assignment: swirl Lesson 6: Subsetting Vectors

* extract elements from a vector based on some conditions that we specify.

| The way you tell R that you want to select some particular

| elements (i.e. a 'subset') from a vector is by placing an

| 'index vector' in square brackets immediately following the

| name of the vector.

| For a simple example, try x[1:10] to view the first ten

| elements of x.

| Index vectors come in four different flavors -- logical

| vectors, vectors of positive integers, vectors of negative

| integers, and vectors of character strings -- each of which

| we'll cover in this lesson.

| Type y <- x[!is.na(x)] to capture all non-missing values

| from x.

| Since NA is not a value, but rather a placeholder for an

| unknown quantity, the expression NA > 0 evaluates to NA.

| Hence we get a bunch of NAs mixed in with our positive

| numbers when we do this.

| Combining our knowledge of logical operators with our new

| knowledge of subsetting, we could do this -- x[!is.na(x) & x

| > 0]. Try it out.

> x[!is.na(x) & x > 0]

[1] 1.5091107 1.4494596 1.5068571 0.3333023 0.2402206 0.7126111

[7] 0.9733860

| In this case, we request only values of x that are both

| non-missing AND greater than zero.

| Many programming languages use what's called 'zero-based

| indexing', which means that the first element of a vector is

| considered element 0. R uses 'one-based indexing', which

| (you guessed it!) means the first element of a vector is

| considered element 1.

| Can you figure out how we'd subset the 3rd, 5th, and 7th

| elements of x? Hint -- Use the c() function to specify the

| element numbers as a numeric vector.

> x[c(3,5,7)]

[1] 1.509111 -1.301244 NA

| Again, nothing useful, but R doesn't prevent us from asking

| for it. This should be a cautionary tale. You should always

| make sure that what you are asking for is within the bounds

| of the vector you're working with.

| Luckily, R accepts negative integer indexes. Whereas x[c(2,

| 10)] gives us ONLY the 2nd and 10th elements of x, x[c(-2,

| -10)] gives us all elements of x EXCEPT for the 2nd and 10

| elements. Try x[c(-2, -10)] now to see this.

| A shorthand way of specifying multiple negative numbers is

| to put the negative sign out in front of the vector of

| positive numbers. Type x[-c(2, 10)] to get the exact same

| result.

| Create a numeric vector with three named elements using vect

| <- c(foo = 11, bar = 2, norf = NA).

> vect <- c(foo = 11, bar = 2, norf = NA)

| We can also get the names of vect by passing vect as an

| argument to the names() function. Give that a try.

| Then, we can add the `names` attribute to vect2 after the

| fact with names(vect2) <- c("foo", "bar", "norf"). Go ahead.

> names(vect2) <- c("foo", "bar", "norf")

| Now, let's check that vect and vect2 are the same by passing

| them as arguments to the identical() function.

### Assignment: swirl Lesson 7: Matrices and Data Frames

* In this lesson, we'll cover matrices and data frames. Both represent 'rectangular' data types, meaning that they are used to store tabular data, with rows and columns.
* matrices can only contain a single class of data, while data frames can consist of many different classes of data.
* the first number is the number of rows and the second is the number of columns. Therefore, we just gave my\_vector 4 rows and 5 columns.

| The dim() function tells us the 'dimensions' of an object.

| What happens if we do dim(my\_vector)? Give it a try.

> dim(my\_vector)

NULL

| All that practice is paying off!

|======= | 14%

| Clearly, that's not very helpful! Since my\_vector is a

| vector, it doesn't have a `dim` attribute (so it's just

| NULL), but we can find its length using the length()

| function. Try that now.

> length(my\_vector)

[1] 20

| Ah! That's what we wanted. But, what happens if we give

| my\_vector a `dim` attribute? Let's give it a try. Type

| dim(my\_vector) <- c(4, 5).

| Another way to see this is by calling the attributes()

| function on my\_vector. Try it now.

> attributes(my\_vector)

$dim

[1] 4 5

> dim(my\_vector) <- c(4,5)

* the first number is the number of rows and the second is the number of columns. Therefore, we just gave my\_vector 4 rows and 5 columns.

Now it's a matrix. View the contents of my\_vector

| now to see what it looks like.

> my\_vector

[,1] [,2] [,3] [,4] [,5]

[1,] 1 5 9 13 17

[2,] 2 6 10 14 18

[3,] 3 7 11 15 19

[4,] 4 8 12 16 20

| Now, let's confirm it's actually a matrix by using the

| class() function. Type class(my\_vector) to see what I mean.

> class(my\_vector)

[1] "matrix"

create a vector containing one element for

| each column. Create a character vector called cnames that

| contains the following values (in order) -- "patient",

| "age", "weight", "bp", "rating", "test".

> cnames <- c("patient", "age", "weight", "bp", "rating", "test")

| Now, use the colnames() function to set the `colnames`

| attribute for our data frame. This is similar to the way we

| used the dim() function earlier in this lesson.

> ?colnames

> colnames(my\_data) <- cnames

## JHSPH Student Handbook on Referencing

<http://www.jhsph.edu/academics/degree-programs/master-of-public-health/current-students/JHSPH-StudentReferencing_handbook.pdf>

* Plagiarism is a violation of academic integrity.
* proper attribution: quotation marks, single spacing and indentation with reference; or reference if not reproduced.
* ***Accurately and appropriately citing your sources is your best defense against any allegations of plagiarism.***
* Referencing acknowledges contributions of others to work and maintains integrity.
  + Properly attribute words and ideas to their ***owners***
  + Enhances the credibility of your arguments by drawing from credible sources.
  + Provide readers with a background into your area of interest.
  + Advance the field of inquiry by providing cross referencing and exploratory avenues for others.
* When to cite sources:
  + When you quote a source
  + When you paraphrase a source
  + When you summarize information from a source
  + When you use facts or data in your work
    - Statistics
    - Descriptions of specific methods or events
    - Technical definitions (standards)
    - Data results from experiments
    - Opinions, arguments, or reasoning of experts
  + If in doubt ***CITE IT***!
* General Rules for Citiation
  + Adhere to a bibliography style or a reference list style
  + Identify source information
  + Consistency
  + Electronic sources

**REFERENCE LIST: CHICAGO STYLE**

In text parenthetical reference: Developing cultural competency is important for lawyers and expert witnesses involved in capital defense cases (Perlin and McClain 2009, 257).

Corresponding reference list entry: Perlin, Michael L. and Valerie McClain. 2009. “WHERE SOULS ARE FORGOTTEN: Cultural Competencies, Forensic Evaluations, and International Human Rights.” Psychology, Public Policy, and Law 15: 257-277.

**BIBLIOGRAPHY: CHICAGO STYLE**

In text footnote: Developing cultural competency is important for lawyers and expert witnesses involved in capital defense cases.1

Footnote: 1 Michael L. Perlin and Valerie McClain. “WHERE SOULS ARE FORGOTTEN: Cultural Competencies, Forensic Evaluations, and International Human Rights.” Psychology, Public Policy, and Law 15 (2009): 257.

Corresponding bibliography entry: Perlin, Michael L. and Valerie McClain. “WHERE SOULS ARE FORGOTTEN: Cultural Competencies, Forensic Evaluations, and International Human Rights.” Psychology, Public Policy, and Law 15 (2009): 257-277.

* Two citation styles common to students:
  + American Psychological Association (APA): <http://www.apastyle.org/>
  + Chicago Manual of Style: <http://www.chicagomanualofstyle.org>
* Tools for Referencing:
  + Endnote
  + Reference Manager
  + RefWorks
  + Microsoft Word “References” section
    - <https://support.office.com/en-us/article/Create-a-bibliography-3403c027-96c8-40d3-a386-bfd5c413ddbb>
* Assistance with work
  + Ensure group or individual work: brainstorming ideas, develop argument, writing drafts, and final work product
  + This can include editing and grammatical errors

## Week 2: Programming with R

### Control Structures – Introduction

* Allow you to control flow of execution of the program, depending on runtime conditions.
* Mostly not used at command line interactive session
* Mostly used in R functions as a structured format

### Control Structures - If-else

* If-else: test a logical condition
* If() else if () else()
* Else is not necessary for every if statement

### Control Structures - For loops

* Loops take an iterator variable and assign successive values form vector
* Examples:

For ( I in 1:4 ) { print(x[i]) }

For ( I in seq\_along(x) ) { print(x[i] }

For ( letter in x) { print(letter) }

For (I in 1:4) print(x [1]) \*\*\* not always good style for one line

* Matricies uses a double nexted for loops
* Generally don’t want to go beyond 2-3 nested for loops
  + Can use functions to help with readability of the code.

### Control Structures - While loops

* Tests for a logical condition, then executes the loop body, then tests for the logical condition again.
* MUST ENSURE THE CONDITION THAT IS FALSE WILL OCCUR
  + For loop has a defined limit of execution.
* && conditions are always checked from left to right

### Control Structures - Repeat, Next, Break

* ***Repeat***: initiate an infinite loop and then call break to stop the loop.
* May use in an optimization routine when checking for a convergence toward a tolerance in an objective function.
* No guarantee that loop will stop
  + May be better to set a hard limit to loop in a for loop and report back the results (converged y/n)
* ***Next***: used to skip an iteration of a loop
* ***Return***: can be used to exit a loop

### Your First R Function

* Write function in a text file and save it outside the command line
* Eventually add function to an R package
* Specify default values
* exFunction.R

### Functions (part 1)

* created using ***function()*** directive
* R objects of class “function”
* Function are “first class objects” = can be treated like any other object in R
  + Can be passed as argument to other functions
  + Can be nested, return value of a function is last expression in the function body to be evaluated
* ***Named arguments*** which potentially have ***default values***
  + *Formal arguments* included in function definition
  + ***formals***() function returns the list of all formal arguments
  + Arguments can be ***missing*** or have *a* ***default value***
* Function arguments matched positionally or by name
  + Named arguments don’t have to be in order
  + Recommend not messing around the order of arguments – readability and consistency
* ***args()*** returns the argument list for a function.
* Argument matching
  + Named arguments are useful on command line since not easy to remember
  + Named arguments good for plotting so useful to not have to remember order
* Can be partially matched
  + Check for exact match for a named argument
  + Check for a partial match
  + Check for positional match

### Functions (part 2)

* Defining a Function
  + f <- function( a, b = 1, c =2, d = NULL) { }
* Lazy Evaluation
  + Arguments to a function are evaluated only as they are needed.
  + No error if arguments are not needed and not given (positional matching)
* “…” augment indicates a variable number of arguments that are usually passed to other functions
  + Used to ***extend*** another function, don’t want to copy all the entire argument list.
  + Used for ***generic functions***, extra arguments passed to methods
  + Needed when number arguments not known in advance
    - ***Paste()*** function concatenates a number of objects (unknown number)
  + Arguments after the “…” argument must be explicitly named (no partial matching)

### Scoping Rules - Symbol Binding

* How does R know which value to assign which symbol
  + Lm define a function or the stats package
  + ***BINDING***
* R retrieve symbol value from the environment
  + ***Global environment***
  + ***Namespace*** of packages on the search list
  + ***Search()*** returns the search list packages
    - .GlobalEnv
    - Autoloads
    - packages
* Global environment is always first on list; base package is always last
* Cannot assume package order (which can confound the namespace)
* When user loads package with ***library***, the namespace is inserted into the second position.
* Scoping Rules – how a value is bound to a variable in a function
  + Make R different from S
  + R uses lexical scoping (static scoping) instead of dynamic scoping
  + Good for stats calculations
  + Related how search list is used for binding
* ***Free variables*** are not ***formal arguments*** (function definition)l and are not ***local variables*** (assigned in function body)
* Lexical scoping
  + Values of free variables are searched for in the environment in which function defined
  + ***Environment***: a collection of symbol, value pairs.
  + Every environment has a parent environment (and may have multiple children)
  + The only environment without a parent is the empty environment
  + A function + an environment = a closure or function closure.
* Searching for a free variable
  + Defined function environment
  + Parent environment
  + Sequence of subsequent parents until we hit top level (usually global environment)
    - Namespace for a package
  + Then goes down package down to the base package and throws error if reach the empty environment
* Why does this matter?
  + Values are usually found in the user’s workspace (global environment)
  + R can define functions inside functions (not something that can be done in C)
    - In this case the environment is inside another function.
      * Example a constructor function.
* Example:

Make.power <- function(n) {

Pow <- function(x) {

X ^ n

}

Pow

}

> Cube <- make.power(3)

> Square <- make.power(2)

> Cube(3)

[1] 27

> square(3)

[1] 9

* What is in a functions environment?

> ls(environment(cube))

[1] “n” “pow”

> get (“n”, environment(cube))

[1] 3

> ls(environment(square))

[1] “n” “pow”

> get(“n”, environment(square))

[1] 2

* Dynamic scoping
  + Variable looked up in the environment from which the function was ***called*** (not defined)
* Other lexical scoping languages:
  + Scheme
  + Perl
  + Python
  + Common Lisp (theory: all languages converge to LISP)
* Consequences of Lexical Scoping
  + For R, All objects must be stored in memory
    - Difficult with large data sets
  + All functions must carry a pointer to the defining environment which could be anywhere
  + S-PLUS, free variables looked up in the global workspace (all the same, less complex)

### Scoping Rules - R Scoping RulesN

* Application to Optimization routines: argument is a vector of parameters
  + optim
  + nlm
  + optimize
* objective function needs to be minimized or maximized
  + function might depend on other things besides parameters
* constructor function
  + most functions in R attempt to minimize functions
  + a negative function will maximize
  + constructor returns the function as the return object
* Lexical scoping summary
  + Objective functions can be built which contain the necessary data for the function
    - Not so long argument lists (good for exploratory and interactive work)
  + Simple and clean code
  + Lexical Scope and Statistical Computing JCGS (2000)

### Scoping Rules - Optimization Example (OPTIONAL)

Examples:

1. Plot the negLogLik example (scoping optimization slides)

### Coding Standards

1. Write code in text editor and save text file (typically ASCII text; others for other than English language)
   1. Least common denominator and easy to read
2. Indentation shows control flow
3. Limit the width of the code (80 columns)
   1. You can change indentation on preferences for RStudio (at least 4, maybe 8 space indents)
   2. Using space indent with 80 column margin keeps code slimmer and helps with readability when constructing code.
4. Limit length of individual functions
   1. Limit to one basic activity
      1. Read data, process data, display data
   2. Helps with readability
   3. Helps with traceback, profiler or debugger; when bug occurs it can narrow down the problem

### Dates and Times

* R uses data classes for dates and times
  + Dates use the **Date** class
  + Times use **POSIXct** or **POSIClt** class
  + Dates stored internally as number of days since 1970-01-01
  + Times stored internally as number of seconds since 1970-01-01

X <- as.Date(“1970-01-01”)

Unclass(x)

[1] 0

Unclass(as.Date(“1970-01-02”)

[1] 2

* POSIXct: very large integer vector to store in a data frame
* POSIClt: list that stores other useful data such as day of week, day of year, month, day of month
* Times can be coerced from character string using as.POSIXlt or as.POSIXct functions
* Strptime function: able to identify the datestring type fields to convert to TIME objects

EXAMPLE POSIXlt and POSIXct coerced data type. Slide 6/10

EXAMPLE strptime

* Use mathematical operations on dates and times ( +, -, ==, <=, >=, etc.)
* Keeps track of leap years, leap seconds, daylight savings, and time zones.
* Many plotting functions recognize Date Time classes.

### Practical R Exercises in swirl Part 2

#### Assignment: swirl Lesson 1: Logic

* TRUE, FALSE, ==, !=, <, >, <=, >=, !, &, &&
* &: both sides true then true else false; evaluates over entire vector using recycling
* &&: only first member of vector

| Let's look at how the AND operator works. There are two AND

| operators in R, `&` and `&&`. Both operators work similarly,

| if the right and left operands of AND are both TRUE the

| entire expression is TRUE, otherwise it is FALSE. For

| example, TRUE & TRUE evaluates to TRUE. Try typing FALSE &

| FALSE to how it is evaluated.

| You can use the `&` operator to evaluate AND across a

| vector. The `&&` version of AND only evaluates the first

| member of a vector. Let's test both for practice. Type the

| expression TRUE & c(TRUE, FALSE, FALSE).

* | : OR operator; evaluates over entire vector using recycling
* || : OR operator only first member of vector

| The OR operator follows a similar set of rules. The `|`

| version of OR evaluates OR across an entire vector, while

| the `||` version of OR only evaluates the first member of a

| vector.

* Order of operations: AND, OR

| The function isTRUE() takes one argument. If that argument

| evaluates to TRUE, the function will return TRUE. Otherwise,

| the function will return FALSE. Try using this function by

| typing: isTRUE(6 > 4)

| The function identical() will return TRUE if the two R

| objects passed to it as arguments are identical. Try out the

| identical() function by typing: identical('twins', 'twins')

| You should also be aware of the xor() function, which takes

| two arguments. The xor() function stands for exclusive OR.

| If one argument evaluates to TRUE and one argument evaluates

| to FALSE, then this function will return TRUE, otherwise it

| will return FALSE. Try out the xor() function by typing:

| xor(5 == 6, !FALSE)

| create a vector of integers called ints. Create this vector by

| typing: ints <- sample(10)

| The vector `ints` is a random sampling of integers from 1 to

| 10 without replacement. Let's say we wanted to ask some

| logical questions about contents of ints. If we type ints >

| 5, we will get a logical vector corresponding to whether

| each element of ints is greater than 5. Try typing: ints > 5

| We can use the resulting logical vector to ask other

| questions about ints. The which() function takes a logical

| vector as an argument and returns the indices of the vector

| that are TRUE. For example which(c(TRUE, FALSE, TRUE)) would

| return the vector c(1, 3).

| Use the any() function to see if any of the elements of ints

| are less than zero.

| Use the all() function to see if all of the elements of ints

| are greater than zero.

#### Assignment: swirl Lesson 2: Functions

| Functions are one of the fundamental building blocks of the

| R language. They are small pieces of reusable code that can

| be treated like any other R object.

| The Sys.Date() function returns a string representing today's date.

| By writing functions, you can gain serious insight into how R works. As John Chambers, the creator of R once said:

| To understand computations in R, two slogans are helpful:

1. Everything that exists is an object.

2. Everything that happens is a function call.

| If you want to see the source code for any function, just type the function name

| without any arguments or parentheses. Let's try this out with the function you just

| created.

# You can pass functions as arguments to other functions just like you can pass

# data to functions.

| You may be surprised to learn that you can pass a function as an argument without

| first defining the passed function. Functions that are not named are appropriately

| known as anonymous functions.

# Let's explore how to "unpack" arguments from an ellipses when you use the

# ellipses as an argument in a function. Below I have an example function that

# is supposed to add two explicitly named arguments called alpha and beta.

#

# add\_alpha\_and\_beta <- function(...){

# # First we must capture the ellipsis inside of a list

# # and then assign the list to a variable. Let's name this

# # variable `args`.

#

# args <- list(...)

#

| You're familiar with adding, subtracting, multiplying, and dividing numbers in R. To do this you use

| the +, -, \*, and / symbols. These symbols are called binary operators because they take two inputs,

| an input from the left and an input from the right.

| In R you can define your own binary operators. In the next script I'll show you how.

# User-defined binary operators have the following syntax:

# %[whatever]%

# where [whatever] represents any valid variable name.

# "%mult\_add\_one%" <- function(left, right){ # Notice the quotation marks!

# left \* right + 1

# }

#

# I could then use this binary operator like `4 %mult\_add\_one% 5` which would

# evaluate to 21.

#### Assignment: swirl Lesson 3: Dates and Times

| Dates are represented by the 'Date' class and times are represented by the 'POSIXct' and 'POSIXlt' classes. Internally, dates are stored as the number of days since 1970-01-01 and times are stored as either the number of seconds since 1970-01-01 (for 'POSIXct') or a list of seconds, minutes, hours, etc. (for 'POSIXlt').

| Use the class() function to confirm d1 is a Date object.

| We can use the unclass() function to see what d1 looks like internally. Try it out.

| Now, let's take a look at how R stores times. You can access the current date and time using the Sys.time() function with no arguments. Do this and store the result in a variable called t1.

| By default, Sys.time() returns an object of class POSIXct, but we can coerce the result to POSIXlt with as.POSIXlt(Sys.time()). Give it a try and store the result in t2.

| t2, like all POSIXlt objects, is just a list of values that make up the date and time. Use str(unclass(t2)) to have a more compact view.

> str(unclass(t2))

List of 11

$ sec : num 12.4

$ min : int 31

$ hour : int 16

$ mday : int 21

$ mon : int 1

$ year : int 116

$ wday : int 0

$ yday : int 51

$ isdst : int 0

$ zone : chr "EST"

$ gmtoff: int -18000

- attr(\*, "tzone")= chr [1:3] "" "EST" "EDT"

| Now that we have explored all three types of date and time objects, let's look at

a few functions that extract useful information from any of these objects -- weekdays(), months(), and quarters().

| strptime() converts character vectors to POSIXlt. In that sense, it is similar to as.POSIXlt(), except that the input doesn't have to be in a particular format (YYYY-MM-DD).

| Finally, there are a number of operations that you can perform on dates and times, including arithmetic operations (+ and -) and comparisons (<, ==, etc.)

| Use difftime(Sys.time(), t1, units = 'days') to find the amount of time in DAYS that has passed since you created t1.

| In this lesson, you learned how to work with dates and times in R. While it is important to understand the basics, if you find yourself working with dates and times often, you may want to check out the lubridate package by Hadley Wickham.

### Quiz: Week 2

9/10

The **parent frame** of a function evaluation is the environment in which the function was called.

### Programming Assignment 1: Air Pollution

**Introduction**

For this first programming assignment you will write three functions that are meant to interact with dataset that accompanies this assignment. The dataset is contained in a zip file **specdata.zip** that you can download from the Coursera web site.

**Although this is a programming assignment, you will be assessed using a separate quiz**.

Data

The zip file containing the data can be downloaded here:

* [specdata.zip](https://eventing.coursera.org/api/redirectStrict/Pj9y83ays6uv9WvC59mjRVOSFkdxVnBRYbHjFdYFt8dV_ykeSWo7kPrkeKolTTTlZLpRSyn5kWfq-oC6FX3_IA.di7PLBZsRFAPF6Fbv-jJlg.plVIZaXHgNlUTgqAuMOzYIAdNc1uNR4T5avHy0Ty5eruVvc1JOPa9JZDoES3wq74U17EvxF8LxRws9oo6h7xExH3wWtnbbxRKBJe1PmzRuBmXVYEDWdCgTlF9Y5nfir-uUo9WJOZTAVLM8mZtjC68NGd7pd4RiNS-qi5iDU9Fn4u1nGZzK_TROr3oO_goR7XlVClPliQeqNFtN-Jzszp6aXmmqR4NHoX2IIPND6Z4aR8IT4opfcBcUK43tmsW0lmmbMOvN3Y0uGJEf3GMF_yz4NNnwciBo5UO2UFgaWuUT5dsl_ozP3Lq8kt_lLtQlv04sdn6ye7ppHFRb5_ceqI2_upqPdwzXBMACgOC4y-00tLHrQv0QbfYxv8RF0XYHZb) [2.4MB]

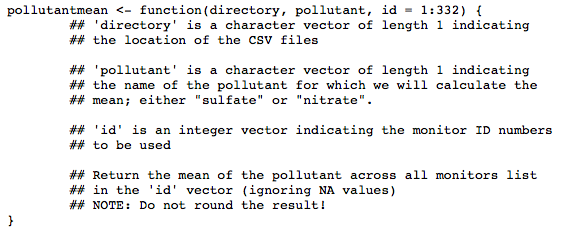
The zip file contains 332 comma-separated-value (CSV) files containing pollution monitoring data for fine particulate matter (PM) air pollution at 332 locations in the United States. Each file contains data from a single monitor and the ID number for each monitor is contained in the file name. For example, data for monitor 200 is contained in the file "200.csv". Each file contains three variables:

* Date: the date of the observation in YYYY-MM-DD format (year-month-day)
* sulfate: the level of sulfate PM in the air on that date (measured in micrograms per cubic meter)
* nitrate: the level of nitrate PM in the air on that date (measured in micrograms per cubic meter)

For this programming assignment you will need to unzip this file and create the directory 'specdata'. Once you have unzipped the zip file, do not make any modifications to the files in the 'specdata' directory. In each file you'll notice that there are many days where either sulfate or nitrate (or both) are missing (coded as NA). This is common with air pollution monitoring data in the United States.

Part 1

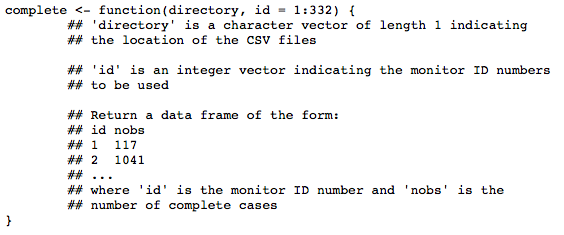
Write a function named 'pollutantmean' that calculates the mean of a pollutant (sulfate or nitrate) across a specified list of monitors. The function 'pollutantmean' takes three arguments: 'directory', 'pollutant', and 'id'. Given a vector monitor ID numbers, 'pollutantmean' reads that monitors' particulate matter data from the directory specified in the 'directory' argument and returns the mean of the pollutant across all of the monitors, ignoring any missing values coded as NA. A prototype of the function is as follows



You can see some [example output from this function](https://eventing.coursera.org/api/redirectStrict/8MfLgoOLIpDOCagqbJ4Pt00ApA2cJ61n5o-ngREEJOBXb7XcKp8Su0NiDDZVtWSM3PynbM6zmQNPmOi-osbA5w.LgVdWNSIyn6CyavU8a3VBA.3mLhipSvbNhLxO7_RKSzC3u9XKprfhx__YwcOy1ANTC0Jnhxwqa070F7WePD1XIW1-Y1fBnsl612NAowN5ZtIHVrIIl8X7du9WCk6cYvy24KwNTY1DWt9UqOzCYZ8-sedjh4ivzjfYEoqdqDlWWPuO4AQoF3IHffZ_Vi8cgsAfBUwjlfAwbR0Dh-nUP_-JSj81Ej2sd48Kc9qV-3cHBNl1nrJVX6PMqhq2TxECN5A4mMkAl4roQF_ft2rtTRRa5Opp4XcqpYZRfIxjKz8rhgsnf5Rr7qxTSylJQ7r9gcoDyPpa-ATzgieE9gFjxhxa5Qz8Ow7KTlDMJ-H1bxdxabGI-oQfO8oqx6ULefqyv7Q5gIiFIArSxNPDIWORw2of24thWWSc9iNEFGXkKkZEIMqhVrEewyZrswFp2OZKncn90). The function that you write should be able to match this output. Please save your code to a file named pollutantmean.R.

Part 2

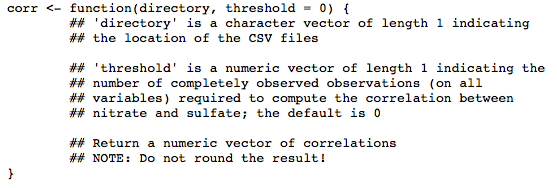
Write a function that reads a directory full of files and reports the number of completely observed cases in each data file. The function should return a data frame where the first column is the name of the file and the second column is the number of complete cases. A prototype of this function follows



You can see some [example output from this function](https://eventing.coursera.org/api/redirectStrict/WQT5PtHTsv0O-9cFiMlS1bH2rdgy-7KTocsV038EHrucJeCiyLZFbIHh2wW55SxlPcOiwlsl-FAQOTTGfjY5-A.p-c0dFXkjbDf7OoNEg1gWA.n3EmBJlbEuNha_XTlkMsqecNt6ASB6Tkz3vu6IHtRtecKXIq8e2ApMWK2bpHJbglrBV7cl_SiQxmOJ_QlMMsRJD1mBgCRBlxWmhdo881arSYmpEGFNikcVQ0NUKJPM2z421munpriMKbs0kQ-gmi8W4kWMeH-Xi6JOjspcOA-NvMKE-iSG6XY0RnL1gTv8M-1GA5GAAD9J1T1cMy4fQo8EZfgoCJQhNMIiYIDIJQBhxNV0oDubtCYYj0_-hWnr86Tsg14KMk7mPMy0_3-0tS7bMSZgejJHD9i6W3XWje1EHL1qzYeeoew0lRdFlzGo8RP7UB8miqeq6VoF-DtnqAoWuMHkeFT8uwZFEqtnOnb7GCZrjJL4zXgQtuvrmo0U2C3P3seOAmsFx-n7IKSVhS_A). The function that you write should be able to match this output. Please save your code to a file named complete.R. To run the submit script for this part, make sure your working directory has the file complete.R in it.

Part 3

Write a function that takes a directory of data files and a threshold for complete cases and calculates the correlation between sulfate and nitrate for monitor locations where the number of completely observed cases (on all variables) is greater than the threshold. The function should return a vector of correlations for the monitors that meet the threshold requirement. If no monitors meet the threshold requirement, then the function should return a numeric vector of length 0. A prototype of this function follows



For this function you will need to use the 'cor' function in R which calculates the correlation between two vectors. Please read the help page for this function via '?cor' and make sure that you know how to use it.

You can see some [example output from this function](https://eventing.coursera.org/api/redirectStrict/dxoBl2u882Ge9lJazEmz6Y11ztg6twy81nLB8B4xR8P5XpzmiSyzQPMczWb7AY0G0aCVHF4SoyqArRe4cyLjvA.3k4pd5wctrDG7BUgf46dkQ.Fp3Xon3pRwQAYPYCAAW2mNp_Jgx6c_dtIghns3-OZ-mHmXon9mU-PymF2LQxl-usA5d6SkFeUaUieLYHgP5VKoitNZ0x8c_B8w7jsNxsJnH8QN3HSG7Rb9w3u_CsfbaRmv612jMR6nwkYtB-b_LpTfYOEkgcMZZf42qmt3FGSXHrsmSsVtlznmLBx8AGbuFeKf7E1nuZHezsCQbLZw2dz-4wA-SK-bzQqvzIPPwtiKzlhi2ca4MRV886vSezwyyrTafOZRv3YAcYEko7dXgGrQUCLzY8cAU9_lWw4lGQxnZf_U3-4SQtcmY7XhtPiPp5wBj8jW_BiEQsIIf1Caf6AFP1Cfr2linMmgsIzN-UYug8hUVj2guQiXrXakkXVja1). The function that you write should be able to match this output. Please save your code to a file named corr.R. To run the submit script for this part, make sure your working directory has the file corr.R in it.

Grading

This assignment will be graded using a quiz.

### Quiz: Programming Assignment 1: Quiz

10/10

## Week 3: Loop Functions and Debugging

***Overview***

This week is what I call "loop functions" in R, which are functions that allow you to execute loop-like behavior in a compact form. These functions typically have the word "apply" in them and are particularly convenient when you need to execute a loop on the command line when using R interactively. These functions are some of the more interesting functions of the R language. This week we also cover the debugger that comes with R and how to interpret its output to help you find problems in your programs and functions. The lecture videos for this week are:

* Loop functions: lapply, apply, tapply, split, mapply
* Debugger, Parts 1-3

***Learning Objectives***

By the end of this week you should be able to:

* Define an anonymous function and describe its use in loop functions [see lapply]
* Describe how to start the R debugger for an arbitrary R function
* Describe what the traceback() function does and what is the function call stack

### Loop Functions – lapply

* Loop function executes a loop over an object or over a set of objects in a compact way
* For-While loops work but not so easy on command line
* Loop functions in R usually have “apply” in them
  + ***lapply***: loop over a list and evaluate a function on each element
  + ***sapply***: same as lapply but try to simplify the result
  + ***apply***: apply a function over the margins of an array
  + ***tapply***: apply a function over subsets of vector
  + ***mapply***: multivariate version of lapply
* ***split*** – useful when used with apply; split object into pieces
* ***lapply***()
  + coerce the object into a list (vectors and object)
  + list is returned from the function
  + can set arguments to the function used by name
* anonymous function – function defined within lapply() call that goes away once lapply executed.
* ***sapply***() – will try to simplify the list that is returned (vector, matrix, list)

### Loop Functions – apply

* apply a function to rows or columns of a matrix or an array
* NOT TRUE currently that it is more efficient.
  + Now is less typing.
* Array – a vector with dimensions attached to it.
  + apply( X is an array, MARGIN, FUN, …..)
  + MARGIN is sort of how you want to collapse the array (1 collapses the columns, 2 collapses rows)
* Optimized use of means and sums on an array – use these instead.
  + rowSums()
  + rowMeans()
  + colSums()
  + colMeans()

### Loop Functions – mapply

* multivariate version of apply, sapply – applies a function in parallel over a set of arguments.
  + Takes multiple lists and works in parallel
  + mapply( FUN, …. , MoreArgs = NULL, Simplfiy = TRUE, USE.NAMES = TRUE )

list ( rep( 1, 4 ), rep ( 2, 3 ), rep (3, 2 ) , rep (4, 1 ) )

mapply ( rep, 1:4, 4:1 )

* vectorize a function that does not accept vector arguments

### Loop Functions – tapply

* apply over segments of vector

tapply ( X, INDEX, FUN = NULL, ….., simplify = TRUE )

* + X is vector
  + INDEX is factor or list of factors f <- gl( 3, 10 ) gives vector of levels 1, 2, 3
  + FUN is function to apply
  + …. Contains the arguments to pass to FUN

**range()** gives the min and max values in the list.s

### Loop Functions – split

* Split takes a vector or other objects and splits into groups determined by a factor or list of factors
* Not a loop function, but can be used with loop functions
* Split( x, f, drop = false, …. )
  + X is vector, list or data frame
  + F is a factor (or coerced to one) or a list of factors
  + Drop – indicates if empty factor levels should be dropped
* Common to use split followe by lapply
  + May be able to use tapply In same way
  + Split works with much more complicated data sets

Example:

$ sapply ( s, function(x) colMeans(x[, c(“Ozone”, “Solar.R”, “Wind”)]))

Example: interactions creates empty factors – time to use drop = TRUE to return list with only observations

Interaction(f1, f2) - combine levels between two factors

List(f1, f2) #Automatically calls the interaction function

### Debugging Tools - Diagnosing the Problem

* Indications there is a problem
  + ***Message***: message but function continues (message function)
  + ***Warning***: something wrong but not necessarily fatal; function executes (warning function)
  + ***Error***: fatal problem occurred; execution stops (stop function)
  + ***Condition***: generic concept for indicating something unexpected occurs (you can create custom)
* ***Invisible(x)*** prevents auto printing to console.
  + Returns the object, but won’t print the messages to the console.
  + Use with if(is.na(x)) to wrap the function body and can
* What to look for:
  + What was input? How did you call the function?
  + What was expected?
  + What did you get?
  + How does get differ from expected?
  + Were your expectations incorrect?
  + Can you reproduce the problem (exactly)?
    - Networking functions can be very hard to reproduce…
    - Interactive/internet data may change over time…

### Debugging Tools - Basic Tools

* Traceback(): prints out the function call stack after an error occurs; does nothing if no error.
* Debug(): flags function with debug mode; able to step through execution of a function one line at a time.
  + Always debug at top of function
* Browser(): suspend the execution of a function wherever it is called puts function in debug mode
  + Sets stop point somewhere else in the code
* Trace(): insert debug code into a function without having to change the code itself
* Recover(): related to traceback; allows changes to default traceback behavior; error will stop execution at where the error occurred, print out the function call stack and then can look around in the browser.
* Blunt force: add print() cat() statements to view code execution.

\*\* **NOTE**: debugging may encourage sloppy code writing – think first, then write, then test, test, test.

### Debugging Tools - Using the Tools

* Must use traceback() right away after the error.
* Browser environment is the function environment ONLY (what is in the function)
  + N + Enter gives you the next line of code run.

Options(error=recovery)

* + Gives the recovery mode which will stop if there is an error and give the function call stack.

### Practical R Exercises in swirl - Part 3

### Assignment: swirl Lesson 1: lapply and sapply

* Flags is dataset; head(flags) will get you the first six rows
* Dim(flags) dimensions
* Viewinfo() gets an info text file about the data set. Where does this come from?
* Class(flags) “data.frame”
* Data.frame is a list of vectors
  + Lapply takes a list and applies a function to each element of the list.
* As.list(flags) gives the list of vectors.
* Sapply() automate simplifying

| When given a vector, the unique() function returns a vector with all duplicate elements removed. In other words, unique() returns a vector of only the 'unique' elements. To see how it works, try unique(c(3, 4, 5, 5, 5, 6, 6)).

> unique(c(3,4,5,6,5,6,6))

[1] 3 4 5 6

| In the last lesson, you learned about the two most fundamental members of R's \*apply family of functions: lapply() and sapply(). Both take a list as input, apply a function to each element of the list, then combine and return the result. lapply() always returns a list, whereas sapply() attempts to simplify the result.

| In this lesson, you'll learn how to use vapply() and tapply(), each of which serves a very specific purpose within the Split-Apply-Combine methodology.

| The Flags dataset from the UCI Machine Learning Repository contains details of various nations and their flags. More information may be found here: http://archive.ics.uci.edu/ml/datasets/Flags

### Assignment: swirl Lesson 2: vapply and tapply

| I've stored the data in a variable called flags. If it's been a while since you

| completed the 'lapply and sapply' lesson, you may want to reacquaint yourself with

| the data by using functions like dim(), head(), str(), and summary() when you

| return to the prompt (>). You can also type viewinfo() at the prompt to bring up

| some documentation for the dataset. Let's get started!

| As you saw in the last lesson, the unique() function returns a vector of the

| unique values contained in the object passed to it. Therefore, sapply(flags,

| unique) returns a list containing one vector of unique values for each column of

| the flags dataset. Try it again now.

| Whereas sapply() tries to 'guess' the correct format of the result, vapply()

| allows you to specify it explicitly. If the result doesn't match the format you

| specify, vapply() will throw an error, causing the operation to stop. This can

| prevent significant problems in your code that might be caused by getting

| unexpected return values from sapply().

| You might think of vapply() as being 'safer' than sapply(), since it requires you

| to specify the format of the output in advance, instead of just allowing R to

| 'guess' what you wanted. In addition, vapply() may perform faster than sapply()

| for large datasets. However, when doing data analysis interactively (at the

| prompt), sapply() saves you some typing and will often be good enough.

| As a data analyst, you'll often wish to split your data up into groups based on

| the value of some variable, then apply a function to the members of each group.

| The next function we'll look at, tapply(), does exactly that.

| The 'landmass' variable in our dataset takes on integer values between 1 and 6,

| each of which represents a different part of the world. Use table(flags$landmass)

| to see how many flags/countries fall into each group.

> table(flags$landmass)

1 2 3 4 5 6

31 17 35 52 39 20

| The 'animate' variable in our dataset takes the value 1 if a country's flag

| contains an animate image (e.g. an eagle, a tree, a human hand) and 0 otherwise.

| Use table(flags$animate) to see how many flags contain an animate image.

> table(flags$animate)

0 1

155 39

| If you take the arithmetic mean of a bunch of 0s and 1s, you get the proportion of

| 1s. Use tapply(flags$animate, flags$landmass, mean) to apply the mean function to

| the 'animate' variable separately for each of the six landmass groups, thus giving

| us the proportion of flags containing an animate image WITHIN each landmass group.

> tapply(flags$animate, flags$landmass, mean)

1 2 3 4 5 6

0.4193548 0.1764706 0.1142857 0.1346154 0.1538462 0.3000000

| Similarly, we can look at a summary of population values (in round millions) for

| countries with and without the color red on their flag with

| tapply(flags$population, flags$red, summary).

> tapply(flags$population, flags$red, summary)

$`0`

Min. 1st Qu. Median Mean 3rd Qu. Max.

0.00 0.00 3.00 27.63 9.00 684.00

$`1`

Min. 1st Qu. Median Mean 3rd Qu. Max.

0.0 0.0 4.0 22.1 15.0 1008.0

### Quiz

5/5 on second try.

Data sets that come with R:

> library(datasets)

> data(iris)

> ?iris #This is Edgar Anderson's Iris Data

> tapply(mtcars$mpg, mtcars$cyl, mean)

4 6 8

26.66364 19.74286 15.10000

> sapply(split(mtcars$mpg, mtcars$cyl),mean)

4 6 8

26.66364 19.74286 15.10000

> with(mtcars, tapply(mpg,cyl,mean))

4 6 8

26.66364 19.74286 15.10000

### Practice Assignment

<https://github.com/derekfranks/practice_assignment/blob/master/practice_assignment.rmd>

To begin, download this file and unzip it into your R working directory.  
<http://s3.amazonaws.com/practice_assignment/diet_data.zip>

You can do this in R with the following code:

dataset\_url <- "http://s3.amazonaws.com/practice\_assignment/diet\_data.zip"

download.file(dataset\_url, "diet\_data.zip")

unzip("diet\_data.zip", exdir = "diet\_data")

Patient.Name Age Weight Day

1 Andy 30 140 1

2 Andy 30 140 2

3 Andy 30 140 3

4 Andy 30 139 4

5 Andy 30 138 5

6 Andy 30 138 6

To create a list of variables based on a dependent variable:

> andy[1,"Weight"]

[1] 140

> andy[30,"Weight"]

[1] 135

> andy[which(andy$Day == 30), "Weight"]

[1] 135

> andy[which(andy[,"Day"] == 30), "Weight"]

[1] 135

> subset(andy$Weight, andy$Day==30)

[1] 135

> files <- list.files("diet\_data")

> files

[1] "Andy.csv" "David.csv" "John.csv" "Mike.csv" "Steve.csv"

> files\_full <- list.files("diet\_data", full.names = TRUE)

> files\_full

[1] "diet\_data/Andy.csv" "diet\_data/David.csv" "diet\_data/John.csv" "diet\_data/Mike.csv"

[5] "diet\_data/Steve.csv"

So what if we wanted to create one big data frame with everybody's data in it? We'd do that with rbind and a loop. Let's start with rbind:

andy\_david <- rbind(andy, read.csv(files\_full[2]))

This line of code took our existing data frame, Andy, and added the rows from David.csv to the end of it. We can check this with:

head(andy\_david)

tail(andy\_david)

This is because you can't rbind something into a file that doesn't exist yet. So let's create an empty data frame called 'dat' before running the loop.

dat <- data.frame()

for (i in 1:5) {

dat <- rbind(dat, read.csv(files\_full[i]))

}

str(dat)

We need to get rid of those NA's for the purposes of calculating the median. There are several approaches. For instance, we could subset the data using complete.cases() or is.na(). But if you look at ?median, you'll see there is an argument calledna.rm that will strip the NA values out for us.

median(dat$Weight, na.rm=TRUE)

Are you calculating the mean value for each file and then taking the mean of those means? That's not the correct approach. You need to combine all of the relevant data into a single data frame or vector and take the mean of *that*.

a <- c(1:5)

b <- c(1:10)

c <- c(10:15)

mean(c(a,b,c))

[1] 6.904762

mean(c(mean(a),mean(b),mean(c)))

[1] 7

You want the first approach, not the second.

Recall from the lectures that $ makes R look for a literal name match. That's not what you want. You want to subset by the value of pollutant (either "sulfate" or "nitrate"), not by "pollutant" since you don't have a column by that name. So, you need to use brackets instead of $.

That could be either something like [[pollutant]] or [, pollutant]

It works, but generally speaking, you don't want to build data frames or vectors by copying and re-copying them inside of a loop. If you've got a lot of data it can become very, very slow. However, this tutorial is meant to provide an introduction to these concepts, and you can use this approach successfully for programming assignments 1 and 3.

If you're interested in learning the better approach, check out Hadley Wickam's excellent material on functionals within R:<http://adv-r.had.co.nz/Functionals.html>. But if you're new to both programming and R, I would skip it for now as it will just confuse you. Come back and revisit it (and the rest of this section) once you are able to write working functions using the approach above.

A higher-order function is a function that takes a function as an input or returns a function as output. We’ve already seen one type of higher order function: closures, functions returned by another function. The complement to a closure is a **functional**, a function that takes a function as an input and returns a vector as output.

A common use of functionals is as an alternative to for loops. For loops have a bad rap in R. They have a reputation for being slow (although that reputation is only partly true, see [modification in place](http://adv-r.had.co.nz/memory.html#modification) for more details)

Improve the speed of your code: <http://adv-r.had.co.nz/Profiling.html#profiling>

### Assignment: Programming Assignment 2: Lexical Scoping

* Instructions:
  + R function that:
    - Cache time consuming computations ( like mean of a long vector or matrix inversion)
    - Function that sets the vector/matrix and includes setter and getter functions (and mean)
    - Using a cachemean() function for that vector/matrix type, the mean is checked in the other data structure before calculating (and saving some processing time)
  + Write the following functions:
    - ***makeCacheMatrix***: This function creates a special "matrix" object that can cache its inverse.
    - ***cacheSolve***: This function computes the inverse of the special "matrix" returned by makeCacheMatrix above. If the inverse has already been calculated (and the matrix has not changed), then the cachesolve should retrieve the inverse from the cache.
  + For this assignment, assume that the matrix supplied is always invertible.
  + Computing the inverse of a square matrix can be done with the solve function in R. For example, if X is a square invertible matrix, then solve(X) returns its inverse.

Repo for Project2: <https://github.com/rbmorrison/ProgrammingAssignment2.git>

SHA-1: 4db301ffcebd846435e9407226b7d89660a81963

### Review Classmates: Programming Assignment 2: Lexical Scoping

5 of 4 assignments reviewed.

## Week 4: Simulation and Profiling

This week covers how to simulate data in R, which serves as the basis for doing simulation studies. We also cover the profiler in R which lets you collect detailed information on how your R functions are running and to identify bottlenecks that can be addressed. The profiler is a key tool in helping you optimize your programs. Finally, we cover the str function, which I personally believe is the most useful function in R.

**Learning Objectives**

By the end of this week you should be able to:

* Call the str function on an arbitrary R object
* Describe the difference between the "by.self" and "by.total" output produced by the R profiler
* Simulate a random normal variable with an arbitrary mean and standard deviation
* Simulate data from a normal linear model

### The str Function

* Most important function in all of R **str()**
* Compactly display the internal structure of an R object (alternative to summary() )
* Good for displaying lists (nested lists)
* **Split()** – function to view the data split along some variable value.

### Simulation - Generating Random Numbers

* Important for statistics and other disciplines
* Functions for probability distributions in R:
  + **rnorm():** generate normal random variates with given mean and standard deviation.
  + **dnorm():** evaluate the normal probability density with given mean/SD at a point or vector of points
  + **pnorm()**: evaluate the cumulative distribution function for a normal distribution
  + **rpois()**: generate random Poisson variates with given rate
* for variations of every probability distributions:
  + d for density
  + r for random number generation
  + p for cumulative distribution (cdf)
    - probability that sample is <= to a value given
  + q for quantile function (inverse distribution function)
* Norm, Gamma, Pois
* Summary() get summary stats from a vector
* Set.seed() - ensures reproducibility
  + Pseudorandom numbers on a computer
  + Set the seed with a value that ensures repeatability of the series of generated numbers

### Simulation - Simulating a Linear Model

* Rbinom() binomial random variables (choosing sexes)
* Create a linear model and evaluate with variates in the independent variable (predictor) and the error variable

### Simulation - Random Sampling

* Sample function (able to draw randomly from a given set of values)
  + Sample( set, and number to be drawn)
  + Sample(set, replace = TRUE ) ## set the replacement = true

### R Profiler (part 1)

* Good to help write code that runs more efficiently
* Why is my code so slow?
  + Profiling = systematic way to determine where code is processing
  + Piece of code may be repeatedly run
* Optimizing code should not be done first( Make it Run, Make it Clear, Make it Right, Make it Fast as needed)
* ***Premature optimization is root of all evil***
* Measure and collect data, do not guess.
* **System.time()**
  + Evaluates an expression and returns the time to evalutate
  + Returns **proc\_time** object
  + **User time**: time charged to the CPU(s) for this expression
  + **Elapsed time**: wall clock time
  + Usually these are close to each other
  + May not be close if CPU is doing a lot of time waiting (multi core or parallel processing)
* Base R does not use multiple cores
  + Links to libraries that do multi-threaded BLAS libraries (Basic Linear Algebra )
* Using this method assumes you know where to look.

### R Profiler (part 2)

* R must be compiled with profiler support (usually the case)
* **Rprof()** starts the R profiler
* **summaryRprof()** summarizes the Rprof() output
* DO NOT USED both sys.time() and Rprof()
* Rprof() keeps track of the function call stack.
  + Default sampling is 0.02 seconds
* Two methods for normalizing data with summaryRprof()
  + By.total - divides by total time
  + By.self - divides by total after removing the other functions higher in the stack.
    - Most of work can be done in lower level helper functions.
* C and Fortran code is not profiled

### Quiz: Week 4 Quiz

Due 07 March 2016

10/10

### Practical R Exercises in swirl Part 4

#### Assignment: swirl Lesson 1: Looking at Data

* First look at data set
* Check class of data: class(data)

| Data.frame: It's very common for data to be stored in a data frame. It is the default Class for data read into R using functions like read.csv() and read.table(), which you'll learn about in another lesson.

* Check dimensions of data: dim(data) or nrow(plants) ncol(plants)
* Check size in memory: object.size(data)
* Get any column (variable) names: names(data)
* Get a summary of data: summary(data)

| summary() provides different output for each variable, depending on its class. For numeric data such as Precip\_Min, summary() displays the minimum, 1st quartile, median, mean, 3rd quartile, and maximum. These values help us understand how the data are distributed.

| For categorical variables (called 'factor' variables in R), summary() displays the | number of times each value (or 'level') occurs in the data.

| You can see that R truncated the summary for Active\_Growth\_Period by including a catch-all category called 'Other'. Since it is a categorical/factor variable, we can see how many times each value actually occurs in the data with

| table(plants$Active\_Growth\_Period).

#### Assignment: swrl Lesson 2: Simulation

* Sample()

| Let's simulate rolling four six-sided dice: sample(1:6, 4, replace = TRUE).

(The probability of rolling

| the exact same result is (1/6)^4 = 0.00077, which is pretty small!)

> sample(1:6, 4, replace=TRUE)

[1] 3 6 3 3

| Now, suppose we want to simulate 100 flips of an unfair two-sided coin. This

| particular coin has a 0.3 probability of landing 'tails' and a 0.7 probability of

| landing 'heads'.

| Let the value 0 represent tails and the value 1 represent heads. Use sample() to

| draw a sample of size 100 from the vector c(0,1), with replacement. Since the coin

| is unfair, we must attach specific probabilities to the values 0 (tails) and 1

| (heads) with a fourth argument, prob = c(0.3, 0.7). Assign the result to a new

| variable called flips.

> flips <- sample(c(0,1), 100, replace=TRUE, prob = c(0.3, 0.7))

| Each probability distribution in R has an r\*\*\* function (for "random"), a d\*\*\*

| function (for "density"), a p\*\*\* (for "probability"), and q\*\*\* (for "quantile").

| We are most interested in the r\*\*\* functions in this lesson, but I encourage you

| to explore the others on your own.

| my\_pois <- replicate(100, rpois(5, 10)) will repeat the operation 100 times and

| store the result.

| And let's take a look at the distribution of our column means by plotting a

| histogram with hist(cm).

> hist(cm)

#### Assignment: swirl Lesson 3: Base Graphics

| We do not cover the more advanced portions of graphics in R in this lesson. These

| include lattice, ggplot2 and ggvis.

* **Plot()** : scatter plot of data
* **Boxplot()**
* **Hist()**

### Programming Assignment 3 INSTRUCTIONS: Hospital Quality

### Quiz: Programming Assignment 3

Due 07 March 2016

> d <- transform(df, rank= ave(df$Value, df$State, FUN=function(x) rank(x, ties.method="min")))

> d

State City Value rank

1 State1 city1 1 1

2 State1 city2 2 2

3 State1 city3 3 3

4 State1 city4 4 4

5 State1 city5 4 4

6 State1 city6 6 6

7 State1 city7 7 7

8 State1 city8 8 8

9 State1 city9 9 9

10 State1 city10 10 10

11 State2 city1 11 1

12 State2 city2 12 2

13 State2 city3 13 3

14 State2 city4 14 4

15 State2 city5 15 5

16 State2 city6 16 6

17 State2 city7 17 7

18 State2 city8 18 8

19 State2 city9 19 9

20 State2 city10 20 10

TO REDO

* Csv data input – column types (see the recommendations on discussion board)
* Reuse code (best, worst, rank)
* Use more lapply functions to make the code
* Rename the column headers – shorter names for the function to use
  + Use column numbers to make switch within brackets “heart attack” = 17
* What are different ranking and filtering methods in R
  + Get a sorted list of rows by state
  + Get a rank of total list of outcomess
  + Get a rank of each State outcomes

# Getting and Cleaning Data

Starts: 14 March

## Welcome to Week 1

Welcome to Week 1 of Getting and Cleaning Data! This course focuses on preparing you for collecting and cleaning data for downstream analysis and sharing.

One of the major components of a data scientist's job is to collect and clean data. Whether at a small organization or a major enterprise, the first step in using data is getting, cleaning and understanding the data. In this course, we focus on R packages and a few outside tools that can be used to collect data from a variety of sources, from Excel files to databases like MySQL. We will also cover a variety of formats including JSON, XML, and flat files (.csv, .txt).

The emphasis of this course is on creating tidy data sets that can be used in downstream analyses. Once you have mastered the material in this course you will be ready to learn about the techniques for exploring, analyzing, and summarizing data offered through our courses track or other Statistics, Data Science, or Machine Learning MOOCs.

One important note is that as part of this class you will be required to use the [Github account](https://eventing.coursera.org/api/redirectStrict/5Zm9J4Qye228sIzkxX9ylOJROMOmywiI6H-m0PZfIUl0VCikg5xIkmycgPQYQ9QXn2Fgph1FkCfhAq3znCEc3g.BKXgkFI7d2xSQt-_Snxbcw.flbMkr3o46SJrhHonWuyjaRBUOjAoml7nvO-OPQTDJ31eYRaX4VkgxTU4u9j1R6Mfj2kzYvuzMg0wrQXPGHPVPxlS1umwHmdElcAKmN00KI2SZ9ZdtFH-pTj6SfwZ9prEbK3aVbQVAY7KhSia8_l7EIgcvd9Wt31ZuV5VHSJPUfx27kwawnWevUQ9cidUz63i2c35nVf-jZ85oboAIG9HUdWMp6BdGH_9VcFOK9ZvYq9pWRSprA44LzYgAUuTJNwF5QWyoctt0vohIvbWOy3mA) you set up in The Data Scientist's Toolbox. Github is a tool for collaborative code sharing and editing. During this course and other courses in the track you will be submitting links to files you publicly place in your Github account as part of peer evaluation. If you are concerned about preserving your anonymity you should set up an anonymous Github account and be careful not to include any information you do not want made available to peer evaluators.

The recommended background for this course are the courses [The Data Scientist's Toolbox](https://eventing.coursera.org/api/redirectStrict/bTpxCSAK6ljjUlCHnSFUmgWkQvkHPMtJObRJR-oRkfNXpc--Hn6HgrTI8wJtLAR-Rp372PTqd-Ut3asiy7sLyg.EyzNfrbkNzATMBNgz_4dDg.7xfDVxkoOP5NmleG-TiuBXAfVVvAd2881erljtO1qvSNnvgZ2dSSfwJF8VYHpjEWQtRoWbAQNxwhODEO9nJx3HbQRAItzp5U08N5vLP35GNjaZaRyCVFqh1rYrSiS1PNhfB_Sssb_M4OHx0hv5M_iQKNfXCmP3ymNE0F9vUnlleZHCybz1qfbvkyjKrxDjvoyK-yAAzrc9UzfmUWhaE4jCNQozogsBV7U5SW9I1bKGn7eqJM6LRahVLrW9u_kQvrTWfzRyDek9a3uM66oeC1DFyLggFMvS0AHC5gLroiLUgU5_q_fmk1zIz50NPYm4SPvcPDmVSiDpT4tyKsHdoq0A) and [R programming](https://eventing.coursera.org/api/redirectStrict/18eoy2PhS1bh5JqApja_C2MZOnfnH_cZZSGz0kkREWG-Kz99T-ucNXD-HZJ4wFLuNsO1ml1fv74v0QpBy41qPw.UDcCmsTsqr28l3ANGTJz3A.NcjmJqUHf9ff-0IsL9bOKtVRZXUBNMOuWohSP88bfOiXT4a5JzEjNaEjDRcM1AfFp8Jimbf_3c8EPTGZWGIlxMDm4aTYM_2JLAiRuK-SwLJ88gDAEQV7AHd0Ppaux25MmY3NPnE4BTu4gh98ilG06fBwKzMN7blj3B18ETpQWQDcIyhb8nAVQDT3HycG9d3wpAYwvORpgmM3vkU3IsFk5ASgQW8YecUkbE_dXW_LQYEVzdOxdsGhGBKaBXp9loCD5_tIuykBHB_VKrF2WwX6mCWGI-QRpcfdyhMXqm4MY2m2wB91ICr4KyKH0RfAnWyV). It will be challenging to take this class concurrently with those classes. You may have to read ahead in previous classes to get the relevant background for this class. For a complete set of course dependencies in the Data Science Specialization please see the [course dependency chart](https://eventing.coursera.org/api/redirectStrict/aWWw2nXYB-D9NZxESfBjUM53q0r6tJWisd6kGPweIULjh6OKbWXdCONLzt-MNgzBK9hhnKL0TV41neMYExiqxw.P6mvXQmpcXS9W5J0cjgDAQ.ioo6IhUFh9re-puAzNMwl3MOB4vHKTm9jV20kxPPdgC8RX6M76owiIZNAUew7n1mvvj5qsomOh5SYSizOu5vLqY-AxBFTr0agsmWf9k76nvKeFdEBA4gL6m0rYMcbk22luqi-wGQjs8RHM_SUQmYJKgWMdwsP-yqWivw_xgniYIDS8DVz4XTVdcgRX6Q7trYWhMQG9HkHgycye3edBXnajiaUql868R6c0AQdoPk-ZvVLQkV4pPOAA9L46el3Gz7Kt-Pm40A_n2XXHnkTHi5GaZCe0RKQoLPzdkr9tnrEEjkQY-H2w3tBTdJzRGEbtdNzkF3YYHYHSg4haZeHbszp0BjqEtgPMuPxSvXiqql8RF_YRQxqzdJIg0VozMKRi1c9sbfB6ypv607LPVzba6913RlH0NBYjhu9H1dmVMONzM).

Please see the course syllabus for information about the Quizzes, the Course Project, and grading. Don't forget to say hi on the discussion forums. The community developed around these courses is one of the best places to learn and the best things about taking a MOOC!

Jeff Leek and the Data Science Track Team

## Syllabus

**Course Description**

Before you can work with data you have to get some. This course will cover the basic ways that data can be obtained. The course will cover obtaining data from the web, from APIs, and from colleagues in various formats including raw text files, binary files, and databases. It will also cover the basics of data cleaning and how to make data tidy. Tidy data dramatically speed downstream data analysis tasks. The course will also cover the components of a complete data set including raw data, processing instructions, codebooks, and processed data. The course will cover the basics needed for collecting, cleaning, and sharing data.

**Content**

Data collection

* Raw files (.csv,.xlsx)
* Databases (mySQL)
* APIs

Data formats

* Flat files (.csv,.txt)
* XML
* JSON

Making data tidy

Distributing data

Scripting for data cleaning

**Quizzes**

* There are four weekly quizzes.
* You must earn a grade of at least 80% to pass a quiz
* You may attempt each quiz up to 3 times in 8 hours.
* The score from your most successful attempt will count toward your final grade.

**Course Project**

The purpose of this project is to demonstrate your ability to collect, work with, and clean a data set. The goal is to prepare tidy data that can be used for later analysis. You will be graded by your peers on a series of yes/no questions related to the project. You will be required to submit:

1. a tidy data set as described below
2. a link to a Github repository with your script for performing the analysis
3. a code book that describes the variables, the data, and any transformations or work that you performed to clean up the data called CodeBook.md.

Finally, you are required to review four of your classmates' projects. You must earn a grade of at least 80% to pass the Course Project.

**swirl Programming Assignment (Optional)**

In this course, you have the option to use the [swirl R package](https://eventing.coursera.org/api/redirectStrict/jdWYEbpR0kjjB-zK0K12-aT0CbyFYQXVJpom7XD2OH89t8XTQKlSUPPz-0AiNVnj_OGIbSQorfVt6LoOhg8Uew.WZPYqxEzoSfQ3v0CV_X5Tg.9y66RKNPF-O-yxfzCsyH_oykMa88oeA1QvN7cZ2hD-lHIktEdfOgSz40T0PoTUMVlOCDe41Mai6FwfZbmZI-6Ze_CIXWU4C-YC3t8UCjPtqEXAvqaBaHAPfB3-9Ck_hYIc1-82yqEf-wIUqspjIoUH2j7I8IAaxVCP7pYIkpUOhi6KEPouL4IQQxCstnfCxXC85ycM1fxf1jpx6NtCaodEOppPSk06xiLBUrgWFs9GubaspFezlKAuG_PZGcYBtuBJea8V5nqrpvE5N6Wxtq3g) to practice some of the concepts we cover in lectures. You can find the instructions for how to install and use swirl in the Programming Assignments section of the course under *Week 1*.

**Grading Policy**

You must score at least 80% on all graded assignments (Quizzes & Project) to pass the course.

Your final grade will be calculated as follows:

* Quiz 1 = 15%
* Quiz 2 = 15%
* Quiz 3 = 15%
* Quiz 4 = 15%
* Course project = 40%
* swirl Programming Assignment (optional) - 0%

**Differences of opinion**

Keep in mind that currently data analysis is as much art as it is science - so we may have a difference of opinion - and that is ok! Please refrain from angry, sarcastic, or abusive comments on the message boards. Our goal is to create a supportive community that helps the learning of all students, from the most advanced to those who are just seeing this material for the first time.

**Plagiarism**

Johns Hopkins University defines plagiarism as "...taking for one's own use the words, ideas, concepts or data of another without proper attribution. Plagiarism includes both direct use or paraphrasing of the words, thoughts, or concepts of another without proper attribution." We take plagiarism very seriously, as does Johns Hopkins University.

We recognize that many students may not have a clear understanding of what plagiarism is or why it is wrong. Please see the JHU referencing guide for more information on plagiarism.

It is critically important that you give people/sources credit when you use their words or ideas. If you do not give proper credit -- particularly when quoting directly from a source -- you violate the trust of your fellow students.

The Coursera Honor code includes an explicit statement about plagiarism:

I will register for only one account. My answers to homework, quizzes and exams will be my own work (except for assignments that explicitly permit collaboration). I will not make solutions to homework, quizzes or exams available to anyone else. This includes both solutions written by me, as well as any official solutions provided by the course staff. I will not engage in any other activities that will dishonestly improve my results or dishonestly improve/hurt the results of others.

Reporting plagiarism on course projects

One of the criteria in the project rubric focuses on plagiarism. Keep in mind that some components of the projects will be very similar across terms and so answers that appear similar may be honest coincidences. However, we would appreciate if you do a basic check for obvious plagiarism and report it during your peer assessment phase.

It is currently very difficult to prove or disprove a charge of plagiarism in the MOOC peer assessment setting. We are not in a position to evaluate whether or not a submission actually constitutes plagiarism, and we will not be able to entertain appeals or to alter any grades that have been assigned through the peer evaluation system.

But if you take the time to report suspected plagiarism, this will help us to understand the extent of the problem and work with Coursera to address critical issues with the current system.

## Pre-Course Survey

Done

## Obtaining Data Motivation

* too often assumed that data is tidy when going through process descriptions
  + Ideally one observation in a row, good valid data
  + Real data looks much more like raw text files of RNA data strings
* JSON is very neat and organized, but hard to use in downstream data tools
* Data is everywhere and in different formats
* Pipeline
  + Raw Data
  + Processing Script
  + Tidy data
  + Data analysis
  + Data communication

## Raw and Processed Data

* Data are values of qualitative or quantitative variables belonging to a set of items
* Raw data
  + Original source of data
  + Often hard to analyze
  + Data includes the processing
  + Raw data may only need a single processing
* CRTICIAL TO KEEP A RECORD FOR ALL PROCESSING
* Processed data
  + Data ready for analysis
  + Processing can include merging, sub-setting, transforming
  + ALL STEPS MUST BE RECORDED
* Example: sequencing DNA used to be 10 years and $1B – now 1 week and $10K
  + Results in a fastQ file
* Beware of what the initial processing steps are and how they affect the data
  + Be careful that measurement to data is accounted for properly

## Components of Tidy Data

* Tidy data is the target, useful for down stream processing
* Going from Raw data to Tidy data:
  + Have raw data
  + Have tidy data set
  + Code book describing each variable and values in tidy data set.
  + An explicit and exact method to go from raw to processed data
    - Example is an R script.
* Rad Data
  + Strange binary file
  + Unformatted Excel file
  + Complicated JSON from scraping Twitter API
  + Hard numbers entered while viewing microscope
  + Do not remove
  + Do not modify
  + No softeware included
* Tidy Data
  + dl
  + Save in one file per table
* Codebook
  + Information about variable (and units)
  + Information about the summary choices made
  + Information about study design you used.
  + A common format is word/txt file
  + Should be a section Study Design that describes how the data collected
  + Cod Book section describes variable and units.
* Instruction List
  + Ideally R script
  + Input for the script is the raw data
  + Output for script is tidy data
  + No parameters for the script (no modification needed)
  + In some cases you need to give all the steps – to splain.

## Downloading Files

* More frequently how to get the data
* Include how data were downloaded is important for the script
* Working Directory:
  + getwd() and setwd()
  + relative or absolute directories
  + In windows: setwd(“C:\\Users\\Andrew\\Downloads”)
* Check for and create directory
  + file.exists(“directory name”)
  + dir.create(“directory Name”)
  + list.files(“directoryName”)

if (!file.exists(“data”)) {

dir.create(“data”)

}

* Get data from internet
  + download.file(“url”, “destfile”, method)
    - useful for .csv, xslx, pdf
  + date() to give the date downloaded
* Methods with download.file()
  + if url starts with http you can use download.file()
  + if url starts with https on windows you should be good
  + if urs starts with https on mac you will need to use method=”curl”

To get Excel files (since they are binary files) set mode=”wb”

> download.file(fileURL, destfile = "natgas.xlsx", mode="wb" )

## Reading Local Files

* Flat files are text files, comma delimited text files, tab delimited text files

if (!file.exists(“data”) ) {

dir.create(“data”)

}

fileUrl <- <https://baltimore.data.gov/some_data>

download.file(fileUrl, destfile = “cameras.csv”, method = “curl”)

dateDownloaded <- date()

* load flat files with read.table()t
  + main function
  + flexible and robust but requires more parameters
  + read data into RAM – big data can cause problems
  + important parameters
  + Tab delimited files are default
    - set sep ( sep = “,”)
    - header = TRUE
  + quotes: quotes = “” means no quotes
  + na.strings = character that represents missing value
  + nrows = number of rows to read of the file
  + skip = skip number of rows

## Reading Excel Files

* Scientists snobbery to excel files
* Google sheets is more popular due to the ability to collaborate online
* package xlsx is useful for this

library(xlsx)

colIndex <- 2:3

rowIndex <-1:4

cameraData <- read.xlsx(“./data/cameras.xlsx”, sheetIndex=1, colIndex=colIndex, rowIndex=rowIndex)

* write.xlsx() write to Excel file
* XLConnect may be better for more frequent excel file uses with R

***.csv and .txt files are more easy to distribute and share***

## Reading XML

* Extensible Markup Language (XML)
* frequently used for structured data on internet pages
* Extracting XML is basis for web-scraping
* Tags – correspond to general labels
* XML is a text file with tags and content
  + Tree structure (DAG)
* Use the XML package in R
  + useInternal loads the file internally to be able to access the nodes in memory.

library(XML)

fileURL < <http://www.w3schools.com/xml/simple.xml>

doc <- xmlTreeParse(fileUrl, useInternal=TRUE)

rootNode <- xmlRoot(doc)

xmlName(rootNode)

xmlName(rootNode)

* access the nodes like a list in R

rootNode[[1]] ## get the first full node in the file

rootNode[[1]][[1]] ## get the first node, first element in file

xmlSApply(rootNode, xmlValue) ## recursively return all tagged text in the file (messy)

* specific elements using XPath language (difficult to learn all of it)
  + /node Top level node
  + //node Node at any level
  + node[@attr-name] Node with an attribute
  + node[@attr-name=”bob”] Node with attribute “bob”
  + good lecture notes here:

<http://www.stat.berkeley.edu/~statcur/Workshop2/Presentations/XML.pdf>

* Example: get items on menu and prices

return the values of the name tag values in a list

xpathSApply( rootNode, “//name”, xmlValue )

return values of price tags in a list

xpathSapply( rootNode, “//price”, xmlValue )

* Ravens example

fileUrl <- <http://espn.go.com/nfl/team/_/name/bal/baltimore-ravens>

doc <- htmlTreeParse( fileUrl, useInternal=TRUE )

scores <- xpathSApply( doc, “//li[@class =’score’] “, xmlValue )

teams <- xpathSApply( doc, “//li[@class = ‘team-name’]”, xmlValue)

scores

XML Tutorials for the XML R package.

> library(XML)

> fileURL2 <- "http://d396qusza40orc.cloudfront.net/getdata%2Fdata%2Frestaurants.xml"

> doc2 <- xmlTreeParse(fileURL2, isURL=TRUE, useInternalNodes=TRUE)

> rootNode2 <- xmlRoot(doc2)

> zipcodes2 <- xpathSApply(rootNode2, "//zipcode", xmlValue)

> zipcodes2

[1] "21206" "21231" "21224" "21211" "21223" "21218" "21205" "21211" "21205" …

> count2 <- length(which(zipcodes2 == "21231"))

> count2

[1] 127

## Reading JSON

* structured format used a lot on internet
* Javascript Object Notation
  + lightweight data storage
  + common format for data from Application Programming Interfaces (APIs)
  + similar to XML but different syntax and format
  + Data stored as:
    - Numbers (double)
    - Strings (double quoted)
    - Boolean (true or false)
    - Array (ordered, comma separated enclosed in square [] )
    - Object (unordered, commas separated collection of key:value pairs in {} )

library (jsonlite)

jsonData <- fromJSON(https://the site)

names(jsonData) ## top level names

names(jsonData$owner) ## data frame that contains the owner data structure tree

jsonData$owner$login ## login for all frames

* turn into JSON

myjson <- toJSON( iris, pretty=TRUE )

cat(myjson)

* back to data fram

iris2 <- fromJSON(myjson)

head(iris2)

* JSON refs:
  + <http://www.json.org>
  + Good tutorial: <http://www.r-bloggers.com/new-package-jsonlite-a-smarter-json-encoderdecoder/>
  + jsonlite vignette

## The data.table Package

* faster, more compact version of data frames
* data.table
  + written in C so is faster
  + all functions accept data.frame work on data.table.
  + much faster at subsetting, group, and updating

library( data.table) ## see video for more example.

* see all tables in memory

tables()

NAME, NROWS, NCOLS, KEY

* good to summarize data by passing functions as a list to the data table subset
  + summarize means, max, etc.
* No copies made, so you have to make an explicity copy with copy() function.
* multiple operations

DT[, m:= {tmp <- (x+z); log2(tmp + 5)} ]

DT[,a := x>0] # add column with TRUE/FALSE if x > 0

DT[,b:= mean(x+w), by=a) # aggregate function over the variable given to ‘by’

DT[, .N, by=x ] # counts with .N, over the x variable given to by

* keys in data.tables, allows easier sorting.

setkey (DT, x)

DT[‘a’] # quick subset to the key value match

* Fast reading

system.time(fread(file)) # get system time to read in a file

### QUIZ EXAMPLE:

> fileURL3 <- "https://d396qusza40orc.cloudfront.net/getdata%2Fdata%2Fss06pid.csv"

> install.package("data.table")

> library(data.table)

> system.time(DF1 <- read.csv(url(fileURL3))

+ )

user system elapsed

24.26 6.75 31.17

> system.time(DT<- fread(fileURL3))

% Total % Received % Xferd Average Speed Time Time Time Current

Dload Upload Total Spent Left Speed

0 0 0 0 0 0 0 0 --:--:-- --:--:-- --:--:-- 0 62 10.9M 62 6940k 0 0 6725k 0 0:00:01 0:00:01 --:--:-- 7046k100 10.9M 100 10.9M 0 0 7784k 0 0:00:01 0:00:01 --:--:-- 8047k

user system elapsed

0.22 0.03 1.61

Warning message:

closing unused connection 3 (https://d396qusza40orc.cloudfront.net/getdata%2Fdata%2Fss06pid.csv)

> class(DT)

[1] "data.table" "data.frame"

> dim(DT)

[1] 14931 239

broken down by sex. Using the data.table package, which will deliver the fastest user time?

DT[,mean(pwgtp15),by=SEX]

## Data Sharing: How to share data with a statistician

# How to share data with a statistician

This is a guide for anyone who needs to share data with a statistician. The target audiences I have in mind are:

* Scientific collaborators who need statisticians to analyze data for them
* Students or postdocs in scientific disciplines looking for consulting advice
* Junior statistics students whose job it is to collate/clean data sets

The goals of this guide are to provide some instruction on the best way to share data to avoid the most common pitfalls and sources of delay in the transition from data collection to data analysis. The [Leek group](http://biostat.jhsph.edu/~jleek/) works with a large number of collaborators and the number one source of variation in the speed to results is the status of the data when they arrive at the Leek group. Based on my conversations with other statisticians this is true nearly universally.

My strong feeling is that statisticians should be able to handle the data in whatever state they arrive. It is important to see the raw data, understand the steps in the processing pipeline, and be able to incorporate hidden sources of variability in one's data analysis. On the other hand, for many data types, the processing steps are well documented and standardized. So the work of converting the data from raw form to directly analyzable form can be performed before calling on a statistician. This can dramatically speed the turnaround time, since the statistician doesn't have to work through all the pre-processing steps first.

# What you should deliver to the statistician

For maximum speed in the analysis this is the information you should pass to a statistician:

1. The raw data.
2. A [tidy data set](http://vita.had.co.nz/papers/tidy-data.pdf)
3. A code book describing each variable and its values in the tidy data set.
4. An explicit and exact recipe you used to go from 1 -> 2,3

Let's look at each part of the data package you will transfer.

### The raw data

It is critical that you include the rawest form of the data that you have access to. Here are some examples of the raw form of data:

* The strange [binary file](http://en.wikipedia.org/wiki/Binary_file) your measurement machine spits out
* The unformatted Excel file with 10 worksheets the company you contracted with sent you
* The complicated [JSON](http://en.wikipedia.org/wiki/JSON) data you got from scraping the [Twitter API](https://twitter.com/twitterapi)
* The hand-entered numbers you collected looking through a microscope

You know the raw data is in the right format if you:

1. Ran no software on the data
2. Did not manipulate any of the numbers in the data
3. You did not remove any data from the data set
4. You did not summarize the data in any way

If you did any manipulation of the data at all it is not the raw form of the data. Reporting manipulated data as raw data is a very common way to slow down the analysis process, since the analyst will often have to do a forensic study of your data to figure out why the raw data looks weird.

### The tidy data set

The general principles of tidy data are laid out by [Hadley Wickham](http://had.co.nz/) in [this paper](http://vita.had.co.nz/papers/tidy-data.pdf) and [this video](http://vimeo.com/33727555). The paper and the video are both focused on the [R](http://www.r-project.org/) package, which you may or may not know how to use. Regardless the four general principles you should pay attention to are:

1. Each variable you measure should be in one column
2. Each different observation of that variable should be in a different row
3. There should be one table for each "kind" of variable
4. If you have multiple tables, they should include a column in the table that allows them to be linked

While these are the hard and fast rules, there are a number of other things that will make your data set much easier to handle. First is to include a row at the top of each data table/spreadsheet that contains full row names. So if you measured age at diagnosis for patients, you would head that column with the name AgeAtDiagnosis instead of something like ADx or another abbreviation that may be hard for another person to understand.

Here is an example of how this would work from genomics. Suppose that for 20 people you have collected gene expression measurements with [RNA-sequencing](http://en.wikipedia.org/wiki/RNA-Seq). You have also collected demographic and clinical information about the patients including their age, treatment, and diagnosis. You would have one table/spreadsheet that contains the clinical/demographic information. It would have four columns (patient id, age, treatment, diagnosis) and 21 rows (a row with variable names, then one row for every patient). You would also have one spreadsheet for the summarized genomic data. Usually this type of data is summarized at the level of the number of counts per exon. Suppose you have 100,000 exons, then you would have a table/spreadsheet that had 21 rows (a row for gene names, and one row for each patient) and 100,001 columns (one row for patient ids and one row for each data type).

If you are sharing your data with the collaborator in Excel, the tidy data should be in one Excel file per table. They should not have multiple worksheets, no macros should be applied to the data, and no columns/cells should be highlighted. Alternatively share the data in a [CSV](http://en.wikipedia.org/wiki/Comma-separated_values) or [TAB-delimited](http://en.wikipedia.org/wiki/Tab-separated_values) text file.

### The code book

For almost any data set, the measurements you calculate will need to be described in more detail than you will sneak into the spreadsheet. The code book contains this information. At minimum it should contain:

1. Information about the variables (including units!) in the data set not contained in the tidy data
2. Information about the summary choices you made
3. Information about the experimental study design you used

In our genomics example, the analyst would want to know what the unit of measurement for each clinical/demographic variable is (age in years, treatment by name/dose, level of diagnosis and how heterogeneous). They would also want to know how you picked the exons you used for summarizing the genomic data (UCSC/Ensembl, etc.). They would also want to know any other information about how you did the data collection/study design. For example, are these the first 20 patients that walked into the clinic? Are they 20 highly selected patients by some characteristic like age? Are they randomized to treatments?

A common format for this document is a Word file. There should be a section called "Study design" that has a thorough description of how you collected the data. There is a section called "Code book" that describes each variable and its units.

### How to code variables

When you put variables into a spreadsheet there are several main categories you will run into depending on their [data type](http://en.wikipedia.org/wiki/Statistical_data_type):

1. Continuous
2. Ordinal
3. Categorical
4. Missing
5. Censored

Continuous variables are anything measured on a quantitative scale that could be any fractional number. An example would be something like weight measured in kg. [Ordinal data](http://en.wikipedia.org/wiki/Ordinal_data) are data that have a fixed, small (< 100) number of levels but are ordered. This could be for example survey responses where the choices are: poor, fair, good. [Categorical data](http://en.wikipedia.org/wiki/Categorical_variable) are data where there are multiple categories, but they aren't ordered. One example would be sex: male or female. [Missing data](http://en.wikipedia.org/wiki/Missing_data) are data that are missing and you don't know the mechanism. You should code missing values as NA. [Censored data](http://en.wikipedia.org/wiki/Censoring_(statistics)) are data where you know the missingness mechanism on some level. Common examples are a measurement being below a detection limit or a patient being lost to follow-up. They should also be coded as NA when you don't have the data. But you should also add a new column to your tidy data called, "VariableNameCensored" which should have values of TRUE if censored and FALSE if not. In the code book you should explain why those values are missing. It is absolutely critical to report to the analyst if there is a reason you know about that some of the data are missing. You should also not [impute](http://en.wikipedia.org/wiki/Imputation_(statistics))/make up/ throw away missing observations.

In general, try to avoid coding categorical or ordinal variables as numbers. When you enter the value for sex in the tidy data, it should be "male" or "female". The ordinal values in the data set should be "poor", "fair", and "good" not 1, 2 ,3. This will avoid potential mixups about which direction effects go and will help identify coding errors.

Always encode every piece of information about your observations using text. For example, if you are storing data in Excel and use a form of colored text or cell background formatting to indicate information about an observation ("red variable entries were observed in experiment 1.") then this information will not be exported (and will be lost!) when the data is exported as raw text. Every piece of data should be encoded as actual text that can be exported.

### The instruction list/script

You may have heard this before, but [reproducibility is kind of a big deal in computational science](http://www.sciencemag.org/content/334/6060/1226). That means, when you submit your paper, the reviewers and the rest of the world should be able to exactly replicate the analyses from raw data all the way to final results. If you are trying to be efficient, you will likely perform some summarization/data analysis steps before the data can be considered tidy.

The ideal thing for you to do when performing summarization is to create a computer script (in R, Python, or something else) that takes the raw data as input and produces the tidy data you are sharing as output. You can try running your script a couple of times and see if the code produces the same output.

In many cases, the person who collected the data has incentive to make it tidy for a statistician to speed the process of collaboration. They may not know how to code in a scripting language. In that case, what you should provide the statistician is something called [pseudocode](http://en.wikipedia.org/wiki/Pseudocode). It should look something like:

1. Step 1 - take the raw file, run version 3.1.2 of summarize software with parameters a=1, b=2, c=3
2. Step 2 - run the software separately for each sample
3. Step 3 - take column three of outputfile.out for each sample and that is the corresponding row in the output data set

You should also include information about which system (Mac/Windows/Linux) you used the software on and whether you tried it more than once to confirm it gave the same results. Ideally, you will run this by a fellow student/labmate to confirm that they can obtain the same output file you did.

# What you should expect from the analyst

When you turn over a properly tidied data set it dramatically decreases the workload on the statistician. So hopefully they will get back to you much sooner. But most careful statisticians will check your recipe, ask questions about steps you performed, and try to confirm that they can obtain the same tidy data that you did with, at minimum, spot checks.

You should then expect from the statistician:

1. An analysis script that performs each of the analyses (not just instructions)
2. The exact computer code they used to run the analysis
3. All output files/figures they generated.

This is the information you will use in the supplement to establish reproducibility and precision of your results. Each of the steps in the analysis should be clearly explained and you should ask questions when you don't understand what the analyst did. It is the responsibility of both the statistician and the scientist to understand the statistical analysis. You may not be able to perform the exact analyses without the statistician's code, but you should be able to explain why the statistician performed each step to a labmate/your principal investigator.

# Contributors

* [Jeff Leek](http://biostat.jhsph.edu/~jleek/) - Wrote the initial version.
* [L. Collado-Torres](http://bit.ly/LColladoTorres) - Fixed typos, added links.
* [Nick Reich](http://people.umass.edu/nick/) - Added tips on storing data as text.

## Practical R Exercises in swirl Part 1

1. Install swirl

Since swirl is an R package, you can easily install it by entering a single command from the R console:

install.packages("swirl")

If you've installed swirl in the past make sure you have version 2.2.21 or later. You can check this with:

packageVersion("swirl")

2. Load swirl

Every time you want to use swirl, you need to first load the package. From the R console:

library(swirl)

3. Install the Getting and Cleaning Data course

swirl offers a variety of interactive courses, but for our purposes, you want the one called Getting and Cleaning Data. Type the following from the R prompt to install this course:

install\_from\_swirl("Getting and Cleaning Data")

4. Start swirl and complete the lessons

Type the following from the R console to start swirl:

swirl()

Then, follow the menus and select the Getting and Cleaning Data course when given the option.

### Notes

* Space an object is occupying in memory: object.size(plants)
* Return a vector of column names: names(plants)
* get summary stats of the columns: summary(plants)
  + summary() displays the minimum, 1st quartile, median, mean, 3rd quartile, and maximum
  + For categorical variables (called 'factor' variables in R), summary() displays the number of times each value (or 'level') occurs in the data.

can see how many times each value actually occurs in the data with

| table(plants$Active\_Growth\_Period).

> table(plants$Active\_Growth\_Period)

Fall, Winter and Spring Spring Spring and Fall

15 144 10

Spring and Summer Spring, Summer, Fall Summer

447 95 92

Summer and Fall Year Round

24 5

### 1: Manipulating Data with dplyr

* install.package(“dplyr”)

| In this lesson, you'll learn how to manipulate data using dplyr. dplyr is a fast and powerful R package written by Hadley Wickham and Romain Francois that provides a consistent and concise grammar for manipulating tabular data.

| One unique aspect of dplyr is that the same set of tools allow you to work with tabular data from a variety of sources, including data frames, data tables, databases and multidimensional arrays. In this lesson, we'll focus on data frames, but everything you learn will apply equally to other formats.

Load data into a data frame tbl:

| The first step of working with data in dplyr is to load the data into what thepackage authors call a 'data frame tbl' or 'tbl\_df'. Use the following code tocreate a new tbl\_df called cran:

| cran <- tbl\_df(mydf).

Printing is much more compact and readable.

| According to the "Introduction to dplyr" vignette written by the package authors, "The dplyr philosophy is to have small functions that each do one thing well." Specifically, dplyr supplies five 'verbs' that cover most fundamental data manipulation tasks: select(), filter(), arrange(), mutate(), and summarize().

select(): get columns

filter(): get rows

mutate(): add columns, aggregate

arrange(): order and sort

summarize(): summary stats

| Okay, ready to put all of this together? Use filter() to return all rows of cran for which r\_version is NOT NA. Hint: You will need to use !is.na() as part of your second argument to filter().

> filter(cran, !is.na(r\_version))

| Sometimes we want to order the rows of a dataset according to the values of a particular variable. This is the job of arrange()

arrange(cran2, country, desc(r\_version), ip\_id)

| It's common to create a new variable based on the value of one or more variables already in a dataset. The mutate() function does exactly this.

| The size variable represents the download size in bytes, which are units of computer memory. These days, megabytes (MB) are a more common unit of measurement. One megabyte is equal to 2^20 bytes. That's 2 to the power of 20, which is approximately one million bytes!

| We want to add a column called size\_mb that contains the download size in megabytes. Here's the code to do it:

mutate(cran3, size\_mb = size / 2^20)

| An even larger unit of memory is a gigabyte (GB), which equals 2^10 megabytes. We might as well add another column for download size in gigabytes!

| Use mutate(cran3, size\_mb = size / 2^20, size\_gb = size\_mb / 2^10) to create two new columns called size\_mb and size\_gb, respectively.

> mutate(cran3, size\_mb = size / 2^20, size\_gb = size\_mb / 2^10

| The last of the five core dplyr verbs, summarize(), collapses the dataset to a single row. Let's say we're interested in knowing the average download size. summarize(cran, avg\_bytes = mean(size)) will yield the mean value of the size variable. Here we've chosen to label the result 'avg\_bytes', but we could have named it anything. Give it a try.

> summarize(cran, avg\_bytes = mean(size))

### 2: Grouping and Chaining with dplyr

| In the last lesson, you learned about the five main data manipulation 'verbs' in dplyr: select(), filter(), arrange(), mutate(), and summarize(). The last of these, summarize(), is most powerful when applied to grouped data.

| The main idea behind grouping data is that you want to break up your dataset into groups of rows based on the values of one or more variables. The

group\_by() function is responsible for doing this.

> library(dplyr)

> cran <- tbl\_df(mydf)

> rm("mydf")

> by\_package <- group\_by(cran, package)

now any operation we apply to the grouped data will take place on a per package basis.

> summarize(by\_package, mean(size))

| Instead of returning a single value, summarize() now returns the mean size for EACH package in our dataset.

n() # num observations; only use inside summarize, mutate, filter

n\_distinct() # efficient count of number distinct elements in set

# faster than length(unique(x))

| We need to know the value of 'count' that splits the data into the top 1% and bottom 99% of packages based on total downloads. In statistics, this is called the 0.99, or 99%, sample quantile. Use quantile(pack\_sum$count, probs = 0.99) to determine this number.

> quantile(pack\_sum$count, probs=0.99)

99%

679.56

> top\_counts <- filter(pack\_sum, count > 679)

> top\_counts\_sorted <- arrange(top\_counts, desc(count))

> View(top\_counts\_sorted)

| Our final metric of popularity is the number of distinct countries from which each package was downloaded. We'll approach this one a little differently to introduce you to a method called 'chaining' (or 'piping').

# Don't change any of the code below. Just type submit()

# when you think you understand it.

# We've already done this part, but we're repeating it

# here for clarity.

by\_package <- group\_by(cran, package)

pack\_sum <- summarize(by\_package,

count = n(),

unique = n\_distinct(ip\_id),

countries = n\_distinct(country),

avg\_bytes = mean(size))

# Here's the new bit, but using the same approach we've

# been using this whole time.

top\_countries <- filter(pack\_sum, countries > 60)

result1 <- arrange(top\_countries, desc(countries), avg\_bytes)

# Print the results to the console.

print(result1)

| We'd like to accomplish the same result as the last script, but avoid saving our intermediate results. This requires embedding function calls within one another.

# Don't change any of the code below. Just type submit()

# when you think you understand it. If you find it

# confusing, you're absolutely right!

result2 <-

arrange(

filter(

summarize(

group\_by(cran,

package

),

count = n(),

unique = n\_distinct(ip\_id),

countries = n\_distinct(country),

avg\_bytes = mean(size)

),

countries > 60

),

desc(countries),

avg\_bytes

)

print(result2)

| In this script, we've used a special chaining operator, %>%, which was originally introduced in the magrittr R package and has now become a key component of dplyr. You can pull up the related documentation with ?chain. The benefit of %>% is that it allows us to chain the function calls in a linear fashion. The code to the right of %>% operates on the result from the code to the left of %>%.

# Read the code below, but don't change anything. As

# you read it, you can pronounce the %>% operator as

# the word 'then'.

#

# Type submit() when you think you understand

# everything here.

result3 <-

cran %>%

group\_by(package) %>%

summarize(count = n(),

unique = n\_distinct(ip\_id),

countries = n\_distinct(country),

avg\_bytes = mean(size)

) %>%

filter(countries > 60) %>%

arrange(desc(countries), avg\_bytes)

# Print result to console

print(result3)da

### 3: Tidying Data with tidyr

tidyr was automatically installed (if necessary) and loaded when you started this lesson. Just to build the habit,

(re)load the package with library(tidyr):

> library(tidyr)

The author of tidyr, Hadley Wickham, discusses his philosophy of tidy data in his 'Tidy Data' paper: http://vita.had.co.nz/papers/tidy-data.pdf

This paper should be required reading for anyone who works with data, but it's not required in order to complete this lesson.

Tidy data is formatted in a standard way that facilitates exploration and analysis and works seamlessly with other tidy data tools. Specifically, tidy data satisfies three conditions:

1) Each variable forms a column

2) Each observation forms a row

3) Each type of observational unit forms a table

Any dataset that doesn't satisfy these conditions is considered 'messy' data. Therefore, all of the following are characteristics of messy data:

1: Column headers are values, not variable names

2: Variables are stored in both rows and columns

3: A single observational unit is stored in multiple tables

4: Multiple types of observational units are stored in the same table

5: Multiple variables are stored in one column

When headers are variables and not the names:

Gather takes multiple columns and collapses into key-value pairs, duplicating all other columns as needed. You usegather() when you notice that you have columns that are not variables.

> students

grade male female

1 A 1 5

2 B 5 0

3 C 5 2

4 D 5 5

5 E 7 4

> gather(students, sex, count, -grade)

grade sex count

1 A male 1

2 B male 5

3 C male 5

4 D male 5

5 E male 7

6 A female 5

7 B female 0

8 C female 2

9 D female 5

10 E female 4

The key and value arguments -- sex and count, respectively -- give the column names for our tidy dataset. The final argument, -grade, says that we want to gather all columns EXCEPT the grade column (since grade is already a proper column variable.)

tidyr offers a convenient separate() function for the purpose of separating one column into multiple columns. Pull up the help file for separate() now.

# Repeat your calls to gather() and separate(), but this time

# use the %>% operator to chain the commands together without

# storing an intermediate result.

#

# If this is your first time seeing the %>% operator, check

# out ?chain, which will bring up the relevant documentation.

# You can also look at the Examples section at the bottom

# of ?gather and ?separate.

#

# The main idea is that the result to the left of %>%

# takes the place of the first argument of the function to

# the right. Therefore, you OMIT THE FIRST ARGUMENT to each

# function.

#

students2 %>%

gather( sex\_class, count, -grade ) %>%

separate( col=sex\_class, into=c("sex", "class")) %>%

print

The first variable, name, is already a column and should remain as it is. The headers of the last five columns, class1 through class5, are all different values of what should be a class variable. The values in the test column, midterm and final, should each be its own variable containing the respective grades for each student.

# Call gather() to gather the columns class1

# through class5 into a new variable called class.

# The 'key' should be class, and the 'value'

# should be grade.

#

# tidyr makes it easy to reference multiple adjacent

# columns with class1:class5, just like with sequences

# of numbers.

#

# Since each student is only enrolled in two of

# the five possible classes, there are lots of missing

# values (i.e. NAs). Use the argument na.rm = TRUE

# to omit these values from the final result.

#

# Remember that when you're using the %>% operator,

# the value to the left of it gets inserted as the

# first argument to the function on the right.

#

# Consult ?gather and/or ?chain if you get stuck.

#

students3 %>%

gather( class, grade, class1:class5 , na.rm= TRUE) %>%

print

The next step will require the use of spread(). Pull up the documentation for spread() now.

# This script builds on the previous one by appending

# a call to spread(), which will allow us to turn the

# values of the test column, midterm and final, into

# column headers (i.e. variables).

#

# You only need to specify two arguments to spread().

# Can you figure out what they are? (Hint: You don't

# have to specify the data argument since we're using

# the %>% operator.

#

students3 %>%

gather(class, grade, class1:class5, na.rm = TRUE) %>%

spread( test, grade ) %>%

print

Lastly, we want the values in the class column to simply be 1, 2, ..., 5 and not class1, class2, ..., class5. We can use the extract\_numeric() function from tidyr to accomplish this. To see how it works, try extract\_numeric("class5").

# We want the values in the class columns to be

# 1, 2, ..., 5 and not class1, class2, ..., class5.

#

# Use the mutate() function from dplyr along with

# extract\_numeric(). Hint: You can "overwrite" a column

# with mutate() by assigning a new value to the existing

# column instead of creating a new column.

#

# Check out ?mutate and/or ?extract\_numeric if you need

# a refresher.

#

students3 %>%

gather(class, grade, class1:class5, na.rm = TRUE) %>%

spread(test, grade) %>%

### Call to mutate() goes here

mutate( class = extract\_numeric(class)) %>%

print

| The fourth messy data problem we'll look at occurs when multiple observational units are stored in the same table. students4 presents an example of this. Take a look at the data now.

# Complete the chained command below so that we are

# selecting the id, name, and sex column from students4

# and storing the result in student\_info.

#

student\_info <- students4 %>%

select( id, name, sex ) %>%

print

Duplicate rows:

# Add a call to unique() below, which will remove

# duplicate rows from student\_info.

#

# Like with the call to the print() function below,

# you can omit the parentheses after the function name.

# This is a nice feature of %>% that applies when

# there are no additional arguments to specify.

#

student\_info <- students4 %>%

select(id, name, sex) %>%

### Your code here

unique %>%

print

| The fifth and final messy data scenario that we'll address is when a single

| observational unit is stored in multiple tables. It's the opposite of the fourth problem.

> passed <- passed %>% mutate( status="passed")

> failed <- failed %>% mutate( status = "failed")

> bind\_rows(passed, failed)

Source: local data frame [10 x 5]

name class final tatus status

(chr) (int) (chr) (chr) (chr)

1 Brian 1 B passed passed

2 Roger 2 A passed passed

3 Roger 5 A passed passed

4 Karen 4 A passed passed

5 Brian 5 C NA failed

6 Sally 1 C NA failed

7 Sally 3 C NA failed

8 Jeff 2 E NA failed

9 Jeff 4 C NA failed

10 Karen 3 C NA failed

# Accomplish the following three goals:

#

# 1. select() all columns that do NOT contain the word "total",

# since if we have the male and female data, we can always

# recreate the total count in a separate column, if we want it.

# Hint: Use the contains() function, which you'll

# find detailed in 'Special functions' section of ?select.

#

# 2. gather() all columns EXCEPT score\_range, using

# key = part\_sex and value = count.

#

# 3. separate() part\_sex into two separate variables (columns),

# called "part" and "sex", respectively. You may need to check

# the 'Examples' section of ?separate to remember how the 'into'

# argument should be phrased.

#

sat %>%

select(-contains("total")) %>%

gather(key=part\_sex, value=count, -score\_range) %>%

separate( col="part\_sex", into=c("part","sex")) %>%

print

# Append two more function calls to accomplish the following:

#

# 1. Use group\_by() (from dplyr) to group the data by part and

# sex, in that order.

#

# 2. Use mutate to add two new columns, whose values will be

# automatically computed group-by-group:

#

# \* total = sum(count)

# \* prop = count / total

#

sat %>%

select(-contains("total")) %>%

gather(part\_sex, count, -score\_range) %>%

separate(part\_sex, c("part", "sex")) %>%

group\_by(part, sex) %>%

mutate(total = sum(count),

prop = count /total ) %>%

print

### 4: Dates and Times with lubridate

In this lesson, we'll explore the lubridate R package, by Garrett Grolemund and Hadley Wickham. According to the package authors, "lubridate has a consistent, memorable syntax, that makes working with dates fun instead of frustrating." If you've ever worked with dates in R, that statement probably has your attention.

> Sys.getlocale("LC\_ALL")

> library(lubridate)

> help(package = lubridate)

> today()

[1] "2016-03-13"’

extract any of these components using the year(), month(), or day() function,

> year(this\_day)

[1] 2016

such that 1 = Sunday, 2 = Monday, 3 = Tuesday, etc.

> wday(this\_day)

[1] 1

> wday(this\_day, label=TRUE)

[1] Sun

Levels: Sun < Mon < Tues < Wed < Thurs < Fri < Sat

> this\_moment <- now()

> this\_moment

[1] "2016-03-13 23:47:11 EDT"

Just like with dates, we can extract the year, month, day, or day of week. However, we can also use hour(), minute(), and second() to extract specific time information. Try any of these three new functions now to extract one piece of time information from this\_moment.

> nyc <- now("America/New\_York")

| For a complete list of valid time zones for use with lubridate, check out the following Wikipedia page:

| http://en.wikipedia.org/wiki/List\_of\_tz\_database\_time\_zones

> how\_long <- interval(last\_time, arrive)

That's not exactly what I'm looking for. Try again. Or, type info() for more options.

as.period(how\_long) will tell you how many years, months, days, etc. have passed since you last saw your friend.

> how\_long

[1] 2008-06-17 SGT--2016-03-17 21:24:46 SGT

| You're close...I can feel it! Try it again. Or, type info() for more options. as.period(how\_long) will tell you how many years, months, days, etc. have passed since you last saw your friend.

> as.period(how\_long)

[1] "7y 9m 0d 21H 24M 46.9014120101929S"

| This is where things get a little tricky. Because of things like leap years, leap

| seconds, and daylight savings time, the length of any given minute, day, month, week,

| or year is relative to when it occurs. In contrast, the length of a second is always

| the same, regardless of when it occurs.

...

|============================================================================ | 97%

| To address these complexities, the authors of lubridate introduce four classes of time related objects: instants, intervals, durations, and periods. These topics are beyond the scope of this lesson, but you can find a complete discussion in the 2011 Journal of Statistical Software paper titled 'Dates and Times Made Easy with lubridate'.

## Week 2

### Reading from MySQL

* widely used in internet based applications
* open source and supported
* data structured
  + databases
  + tables within databases
  + fields with tables
* each table is sort of like an R data.frame
* Common IDs (key) links
* Install mysql database:
  + <http://dev.mysql.com/doc/refman/5.7/en/installing.html>
* Install RmySql
  + <http://biostat.mc.vanderbilt.edu/wiki/Main/RMySQL>
  + <http://www.ahschulz.de/2013/07/23/installing-rmysql-under-windows>
* UCSC database
  + <http://genome.ucsc.edu>
  + Genome Bioinformatics
* Connect to database through R
  + uses RMySQL package

***see the databases available:***

ucscDb <- dbConnect(MySQL(), user=”genome”,

h host=”genome-mysql.cse.ucsc.edu”)

result <- dbGetQuery(ucscDb, “show databases;”); dbDisconnect(ucscDb);

***get a particular database***

hg19 <- dbConnect(MySQL(), user=”genome”, db=”hg19”,

host=”genome-mysql.cse.ucsc.edu”)

allTables <- dbListTables(hg19)

length(allTables)

***get the fields in a table:***

dbListFields(hg19, “affyU133Plus2”)

dbGetQuery(hg19, “select count(\*) from affyU133Plus2”)

***get a dataframe from the table***:

affyData <- dbReadTable(hg19, “affyU133Plus2”)

head(affyData)

***select a specific subset:***

query <- dbSendQuery(hg19, “select \* from affyU133Plus2 where mismatches between 1 and 3”)

affyMis <- fetch(query); quantile(affyMis$misMatches)

***get a small query (not the whole table):***

affyMisSmall <- fetch(query, n=10); dbClearResult(query)

***REMEMBER TO CLOSE THE CONNECTION!!!***

dbDisconnect(hg19)

* Further Resources:
  + RMySQL vignette: <http://cran.r-project.org/web/packages/RMySQL/RMySQL.pdf>
  + List of commands: <http://www.pantz.org/software/mysql/mysqlcommands.html>
  + Blog post: <http://www.r-bloggers.com/mysql-and-r>

### Reading from HDF5

* HDF5 used for storing large data sets
* HDF = Hierarchical Data Format
* ***Groups*** contain zero or more data sets and metadata
  + group header with name and attribute
  + group symbol
* ***Datasets*** are multidimensional array of data elements with metadata
  + ***header*** with name, datatype, dataspace, and storage layout
  + ***data array*** with the data
* Install R HDF5 package:

source(<http://bioconductor.org/biocLite.R>)

biocLIte(“rhdf5”)

library(rhdf5)

created = h5createFile(“example.h5”)

created

* Bioconductor: [www.bioconductor.org/](http://www.bioconductor.org/) # good big data packages, mostly genomics focused, but good
* RHDF5 tutorial: <http://www.bioconductor.org/packages/release/bioc/vignettes/rhdf5/inst/doc/rhdf5.pdf>
  + ***See this tutorial for rest of the lesson***

### Reading from The Web

* Webscrapping: programmatically extracting data from HTML code of websites.
  + Netflix reverse engineered Hollywood
  + Some websites do not want to be scrapped.
  + You may be IP blocked if you read too many pages to quickly
* ***Example:***

con = url(<http://scholar.google.com/citations?user=HI-I6C0AAAAJ&hl=en>)

htmlCode = readLines(con)

close(con)

htmlCode

Result is one long string of html text.

* HTML parsing:

library(html)

url <- <http://scholar.google.com/citations?user=HI-I6C0AAAAJ&hl=en>

html <- htmlTreeParse(url, useInteralNodes=T)

xpathSApply(html, “//title”, xmlValue)

xpathSApply(html, “//td[@id=’col-citedby’]]”, xmlValue)

library(httr); html2 = GET(url)

content2 = content(html2, as=”text”)

parsedHtml = htmlParse(content2, asText=TRUE)

xpathSApply(parsedHtml, “//title”, xmlValue)

* httr package allows you to authenticate with website:
  + authenticate(“user”, “passwd”)
* using a handle() allows subsequent access with authenticate via cookies.

### Reading From APIs

* Application Programming Interfaces
* create an account with the organization (Developer account)
  + creates an ***application*** that can be authenticated
* ***See lecture notes for an example with twitter***
  + JSON file is used by twitter
* httr works well with Facebook, Google, Twitter, Github, etc.
  + good demo on GitHub and other sites.

### Reading From Other Sources

* “there is a package for that”
* Google to find the package: “data type R package”
* Example useful ones:
  + file # open connection to a text file
  + url # open connection to a url
  + gzfile # open connection to .gz file
  + bzfile # open connection to .bz file
  + ? connections more info on connecting to files
* ***foreign package***
  + loads data from other stats programs
* ***RPostgresSQL***
  + <https://code.google.com/p/rpostgresql/>
* ***RODBC*** 
  + provides interfaces to multiple databases (PostgreSQL, MySQL, MS Access, SQLite)
* Reading images (for processing)
  + jpeg
  + readbitmap
  + png
  + EBImage (in bioconductor)
* Read GIS data
  + rdgal
  + rgeos
  + raster
* Read music data
  + tuner http://cran.r-project.org/web/packages/tuneR/
  + seewave http://rug.mnhn.fr/seewave/

### Quiz: Week 2 - Due March 20, 10:59 PM PT

1.

Register an application with the Github API here <https://github.com/settings/applications>. Access the API to get information on your instructors repositories (hint: this is the url you want "https://api.github.com/users/jtleek/repos"). Use this data to find the time that the datasharing repo was created. What time was it created?

**Client ID**

584d7b9cdf216ccf598b

**Client Secret**

bd5dc31fccc5ff3e25c7104c41869fa8f92cfee2

This tutorial may be useful (<https://github.com/hadley/httr/blob/master/demo/oauth2-github.r>). You may also need to run the code in the base R package and not R studio.



2012-06-20T18:39:06Z



2013-11-07T13:25:07Z



2014-02-06T16:13:11Z



2014-03-05T16:11:46Z

> library(jsonlite)

> library(httpuv)

> library('httr')

> myapi <- oauth\_app("github",key="584d7b9cdf216ccf598b", secret="9413536677322cedb7fec4fc71daccd73aeb68dc")

> request1 <- GET("https://api.github.com/users/jtleek/repos", config(token = github\_token))

> myjson <- content(request1)

> myjson2 <- jsonlite::fromJSON(toJSON(myjson))

> created <- myjson2[myjson2$name=="datasharing",]

> created$created\_at

[[1]]

[1] "2013-11-07T13:25:07Z"

2.

The sqldf package allows for execution of SQL commands on R data frames. We will use the sqldf package to practice the queries we might send with the dbSendQuery command in RMySQL.

Download the American Community Survey data and load it into an R object called

acs

<https://d396qusza40orc.cloudfront.net/getdata%2Fdata%2Fss06pid.csv>

Which of the following commands will select only the data for the probability weights pwgtp1 with ages less than 50?



sqldf("select \* from acs")



sqldf("select \* from acs where AGEP < 50")



sqldf("select pwgtp1 from acs")



sqldf("select pwgtp1 from acs where AGEP < 50")

3.

Using the same data frame you created in the previous problem, what is the equivalent function to unique(acs$AGEP)



sqldf("select AGEP where unique from acs")



sqldf("select distinct AGEP from acs")



sqldf("select distinct pwgtp1 from acs")



sqldf("select unique AGEP from acs")

4.

How many characters are in the 10th, 20th, 30th and 100th lines of HTML from this page:

http://biostat.jhsph.edu/~jleek/contact.html

(Hint: the nchar() function in R may be helpful)



43 99 7 25



45 31 7 31



45 31 7 25



45 92 7 2



43 99 8 6



45 31 2 25



45 0 2 2

> con <- url("http://biostat.jhsph.edu/~jleek/contact.html")

> htmlCode = readLines(con)

> ?readLines

> close(con)

> str(htmlCode)

chr [1:180] "<!DOCTYPE html PUBLIC \"-//W3C//DTD XHTML 1.0 Strict//EN\" \"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd\">" ...

> dim(htmlCode)

NULL

> class(htmlCode)

[1] "character"

> htmlCode[1]

[1] "<!DOCTYPE html PUBLIC \"-//W3C//DTD XHTML 1.0 Strict//EN\" \"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd\">"

> htmlCode[3]

[1] "<html xmlns=\"http://www.w3.org/1999/xhtml\" xml:lang=\"en\" lang=\"en\">"

> htmlCode[2]

[1] ""

> nchar(htmlCode[10])

[1] 45

> nchar(htmlCode[20])

[1] 31

> nchar(htmlCode[30])

[1] 7

> nchar(htmlCode[100])

[1] 25

5.

Read this data set into R and report the sum of the numbers in the fourth of the nine columns.

<https://d396qusza40orc.cloudfront.net/getdata%2Fwksst8110.for>

Original source of the data: <http://www.cpc.ncep.noaa.gov/data/indices/wksst8110.for>

(Hint this is a fixed width file format)



35824.9



32426.7



222243.1



101.83



28893.3



36.5

## Week 3

### Subsetting and Sorting

* subset example in video
* Subset by column:

X[,1] or X[,”var1”]

* Subset with logical ANDS and ORS

X[X$var1 <= 3 & X$var2 > 11 ]

* Subsetting on NAs will not return those rows

X[which(X$var2 >8 ), ]

* Sort on variable

sort(X$var1)

sort(X$var1, decreasing = TRUE)

sort(x$var1, na.last = TRUE)

* Ordering

X[ order(X$var1),] # reorder column one in increasing order

X[ order{X$var2, X$var3),]

* Order with plyr

library(plyr)

arrange(X, var1)

arrange(X, desc(var1))

* Add column of data

X$var4 <- rnorm(5)

Y <- cbind(X, rnorm(5) ) # bind right side

Y <- cbind (rnorm(5), X ) # bind on left side

Lecture Notes for more subsetting:

<http://www.biostat.jhsph.edu/~ajaffe/lec_winterR/Lecture%202.pdf>

### Summarizing Data

* example dataset: <https://data.baltimorecity.gov/Community/Restaurants/k5ry-ef3g>
  + get link to the csv data set
  + same as reading data in
* Look at the data

head(restData, n=3)

tail( restData, n=3 )

summar(restData) # give counts, min, median, mean 1st/3rd Quartile, Max

str(restData) # get the class, dims, and variable names and classes

quantile( restData$councilDistrict, na.rm=TRUE) # quantile info of a particular variable column

quantile( restData$councilDistrict, probs=c(0.5,0.75,0.9))

* Table

table (restData$zipCode, useNA=”ifany”) # default does not show missing values

* Two dimensional count table

table ( restData$councilDistrict, restData$zipCode )

* Check for missing values

sum (is.na ( restData$councilDistrict) # sums the number of TRUEs

any ( is.na( restData$councilDistrict) # gives a TRUE / FALSE

all ( restData$zipCode > 0 ) # check all values at once for a condition

* Row and Column Sums

colSums ( is.na (restData ))

all (colSums ( is.na(restData ) == 0)

* Find values with specific characteristics

table ( restData$zipCode %in% c(“21212”)) # get a table of T/F for the values listed

table ( restData$zipCode %in% c(“21212”,”21213”) )

* return the value rows with characteristics

restData[ restData$zipCode %in% c(“21212”), ]

* Cross Tabs

xt <- xtabs (Freq ~ Gender + Admit, data=DF )

xt

ftable(xt) # summarize in smaller compact form, looks like a pivot table

* Size of data set

object.size(fakeData) # number of bytes

print (object.size (fakeData), units=”Mb”) #different units

### Creating New Variables

* Raw data may not have what you are looking for
* Data should be transformed to be more useful
* Usually add the new variables to the data frame
  + Important for predictive analysis
* Common variables
  + Missingness indicators – point out where data is missing
  + Cutting up quantitative variables – factor versions of a variable (bins)
  + Applying transforms – to get strange distributions in useable format
* Creating ***sequences***
  + need to index your data set

s1 <- seq(1,10, by=2)

s2 <- seq(1,10, length=3 )

x <- c(1,3,8,25,100); seq(along = x)

* Setting variables

restData$nearMe = restData$neighborhood %in% c(“Roland Park”, “Homeland”)

* Create binary variables
  + Using ***ifelse()***

restData$zipWrong = ifelse( restData$zipcode < 0 , TRUE, FALSE )

* Creating categorical variables
  + using ***cut()***

restData$zipGroups = cut ( restData$zipCode, breaks = quantile(restData$zipCode))

* Create factor variables
  + using ***factor()***

restData$zcf <- factor (restData$zipCode ) #create a factor class variable

* + Levels of factor variables
    - Use ***relevel ()***
    - Use ***as.numeric()***
* Using the mutate function

library(Hmisc); library(plyr)

restData2 = mutate(restData, zipGroups=cut2(zipCode,g=4))

* ***Common Tranforms:***
  + absolute value: abs(x)
  + square root: sqrt(x)
  + ceiling: ceiling(x)
  + floor: floor(x)
  + round(x, digits=n)
  + signif(x,digits=n)
  + cosine, sine cos(x), sin(x)
  + natural logarithm log(x)
  + logarithm log2(x), log10(x)
  + exponential exp(x)

Reference: <http://statmethods.net/management/functions.html>

**Built-in Functions**

Almost everything in **R** is done through functions. Here I'm only refering to numeric and character functions that are commonly used in creating or recoding variables.

**Numeric Functions**

|  |  |
| --- | --- |
| **Function** | **Description** |
| **abs(***x***)** | absolute value |
| **sqrt(***x***)** | square root |
| **ceiling(***x***)** | ceiling(3.475) is 4 |
| **floor(***x***)** | floor(3.475) is 3 |
| **trunc(***x***)** | trunc(5.99) is 5 |
| **round(***x***, digits=***n***)** | round(3.475, digits=2) is 3.48 |
| **signif(***x***, digits=***n***)** | signif(3.475, digits=2) is 3.5 |
| **cos(***x***), sin(***x***), tan(***x***)** | also acos(*x*), cosh(*x*), acosh(*x*), etc. |
| **log(***x***)** | natural logarithm |
| **log10(***x***)** | common logarithm |
| **exp(***x***)** | e^*x* |

**Character Functions**

|  |  |
| --- | --- |
| **Function** | **Description** |
| **substr(***x***, start=***n1***, stop=***n2***)** | Extract or replace substrings in a character vector. x <- "abcdef"  substr(x, 2, 4) is "bcd"  substr(x, 2, 4) <- "22222" is "a222ef" |
| **grep(***pattern***,***x***, ignore.case=**FALSE**, fixed=**FALSE**)** | Search for *pattern* in *x*. If fixed =FALSE then *pattern* is a [regular expression](http://regexlib.com/CheatSheet.aspx). If fixed=TRUE then *pattern* is a text string. Returns matching indices. grep("A", c("b","A","c"), fixed=TRUE) returns 2 |
| **sub(***pattern***,***replacement***,***x***, ignore.case =**FALSE**, fixed=**FALSE**)** | Find *pattern* in *x* and replace with *replacement* text. If fixed=FALSE then *pattern* is a regular expression[.](http://www.ilovejackdaniels.com/regular_expressions_cheat_sheet.pdf) If fixed = T then *pattern* is a text string.  sub("\\s",".","Hello There") returns "Hello.There" |
| **strsplit(***x***,***split***)** | Split the elements of character vector *x* at *split*.  strsplit("abc", "") returns 3 element vector "a","b","c" |
| **paste(..., sep="")** | Concatenate strings after using *sep* string to seperate them. paste("x",1:3,sep="") returns c("x1","x2" "x3") paste("x",1:3,sep="M") returns c("xM1","xM2" "xM3") paste("Today is", date()) |
| **toupper(***x***)** | Uppercase |
| **tolower(***x***)** | Lowercase |

**Statistical Probability Functions**

The following table describes functions related to probaility distributions. For random number generators below, you can use set.seed(1234) or some other integer to create reproducible pseudo-random numbers.

|  |  |
| --- | --- |
| **Function** | **Description** |
| **dnorm(***x***)** | normal density function (by default m=0 sd=1) # plot standard normal curve x <- pretty(c(-3,3), 30) y <- dnorm(x) plot(x, y, type='l', xlab="Normal Deviate", ylab="Density", yaxs="i") |
| **pnorm(***q***)** | cumulative normal probability for q  (area under the normal curve to the left of q) pnorm(1.96) is 0.975 |
| **qnorm(***p***)** | normal quantile.  value at the p percentile of normal distribution  qnorm(.9) is 1.28 # 90th percentile |
| **rnorm(***n***, m=**0**,sd=**1**)** | n random normal deviates with mean m  and standard deviation sd.  #50 random normal variates with mean=50, sd=10 x <- rnorm(50, m=50, sd=10) |
| **dbinom(***x***,***size***,***prob***) pbinom(***q***,***size***,***prob***) qbinom(***p***,***size***,***prob***) rbinom(***n***,***size***,***prob***)** | binomial distribution where size is the sample size  and prob is the probability of a heads (pi)  # prob of 0 to 5 heads of fair coin out of 10 flips dbinom(0:5, 10, .5)  # prob of 5 or less heads of fair coin out of 10 flips pbinom(5, 10, .5) |
| **dpois(***x***,***lamda***) ppois(***q***,***lamda***) qpois(***p***,***lamda***) rpois(***n***,***lamda***)** | poisson distribution with m=std=lamda #probability of 0,1, or 2 events with lamda=4 dpois(0:2, 4) # probability of at least 3 events with lamda=4  1- ppois(2,4) |
| **dunif(***x***, min=**0**, max=**1**) punif(***q***, min=0, max=**1**) qunif(***p***, min=**0**, max=**1**) runif(***n***, min=**0**, max=**1**)** | uniform distribution, follows the same pattern  as the normal distribution above.  #10 uniform random variates x <- runif(10) |

**Other Statistical Functions**

Other useful statistical functions are provided in the following table. Each has the option na.rm to strip missing values before calculations. Otherwise the presence of missing values will lead to a missing result. Object can be a numeric vector or data frame.

|  |  |
| --- | --- |
| **Function** | **Description** |
| **mean(***x***, trim=**0**, na.rm=**FALSE**)** | mean of object x # trimmed mean, removing any missing values and  # 5 percent of highest and lowest scores  mx <- mean(x,trim=.05,na.rm=TRUE) |
| **sd(***x***)** | standard deviation of object(x). also look at var(x) for variance and mad(x) for median absolute deviation. |
| **median(***x***)** | median |
| **quantile(***x***,***probs***)** | quantiles where x is the numeric vector whose quantiles are desired and probs is a numeric vector with probabilities in [0,1]. # 30th and 84th percentiles of x y <- quantile(x, c(.3,.84)) |
| **range(***x***)** | range |
| **sum(***x***)** | sum |
| **diff(***x***, lag=***1***)** | lagged differences, with lag indicating which lag to use |
| **min(***x***)** | minimum |
| **max(***x***)** | maximum |
| **scale(***x***, center=**TRUE**, scale=**TRUE**)** | column center or standardize a matrix. |

**Other Useful Functions**

|  |  |
| --- | --- |
| **Function** | **Description** |
| **seq(***from***,***to***,***by***)** | generate a sequence indices <- seq(1,10,2) #indices is c(1, 3, 5, 7, 9) |
| **rep(***x***,***ntimes***)** | repeat *x* *n* times y <- rep(1:3, 2) # y is c(1, 2, 3, 1, 2, 3) |
| **cut(***x***,***n***)** | divide continuous variable in factor with *n* levels  y <- cut(x, 5) |

### Reshaping Data

* The goal is tidy data
  + each variable own column
  + each observation forms a row
  + each file, table contains information about one type of observation
* Melting Data Frame
  + using ***melt()***
  + give the “id” variables and “measure” variables

mtcars$carname <- rownames(mtcars)

carMelt <- melt(mtcars, id=c(“carname”, “gear”, “cylc”), measure.vars=c(“mpg”,”hp”))

* Casting data frames
  + using ***dcast()***

cylData <- dcast( carMelt, cyl ~ variable )

cylData <- dcast( carMelt, cyl ~ variable, mean )

* Averaging values
  + using ***tapply()***

tapply( InsectSprays$count, InsectSprays$spray, sum )

* + using plyr package

library(plyr)

ddply(InsectSprays, .(spray), summarize, sum=sum(count))

* Arrange = faster reordering without using order()

### Managing Data Frames with dplyr – Introduction

* designed to work with data frames
* Functions:
  + arrange
  + filter
  + mutate
  + select
  + rename
* Assumptions made about data frames:
  + one observation per row
  + each column represents a variable or measure
  + primary implementation is default R
    - can use with data table package or database systems
  + Well formed and annotated (levels) data frame
* dplyr is optimized and distilled version of plyr
* Select – return subset of columns of a data frame
* Filter – extract a subset rows from data frame based on logical conditions
* Arrange – reorder rows of data frame
* Rename – rename variables in data frame
* Mutate – add new variables/columns or transform existing variables/columns
* Summarize – generate summary statistics of variable
* Print – condensed format print function

### Managing Data Frames with dplyr - Basic Tools

* Select
  + select a subset of columns
  + using dplyr ***select(***)

select( Chicago, city:dptp )

select( chicago, -(city:dptp) )

* + using base:R

I <- match(“city”, names(Chicago) )

j <- match(“dptp”, names(Chicago))

Chicago[, -(i:j) ]

* Filter
  + select a subset of rows
  + using ***filter()***

filter( Chicago, pm25tmean2 > 30 )

filter( Chicago, pm25tmean2 > 30 & tmpd > 80 )

* Arrange
  + using ***arrange()***

arrange ( Chicago, data )

arrange (Chicago, desc(date) )

* Rename
  + hard to do in R, easier using ***rename()***

rename( Chicago, pm25 = pm25tmean2, dewpoint = dptp )

* Mutate
  + using mutate()
  + center the variable (subtract the mean)

mutate ( Chicago, pm25detrend = pm25 – mean(pm25, na.rm = TRUE ))

* + Group by factor

chicago <- mutate(Chicago, tempcat = factor (1\* (tmpd > 80 ), labels = c(“cold”, “hot”) ))

hotcold <- group\_by(Chicago, tempcat)

mummarize( hotcold, pm25 = mean(pm25, rm.na=TRUE), o3 = max(o3tmean2), no2 = median( no2tmean2))

* + Summarize by year

Chicago <- mutate( Chicago, year = as.POSIXlt(date)$year + 1900)

years <- group\_by( Chicago, year )

summarize(years, pm25 = mean(pm25, na.rm=TRUE), o3 = max(o3tmean2), no2 = median(no2tmean2)))

* + Use a pipline operator

Chicago %>% mutate(month = as.POSIXlt(date)$mon + 1 ) %>% group\_by(month) %>% summarize(pm25 = mean(pm25, na.rm=TURE), filter( Chicago, pm25tmean2 > 30 )

* Other interfaces: data.table package for large fast tables; SQL interface via DBI package

### Merging Data

* Merge data sets based on an id (same as with linked database tables)
* Merge
  + using ***merge()***
  + can use plyr package join() - only joins based on the ids that are the same (can’t left or right join)
    - it is good for merging multiple tables all at once

### Assignment: swirl Lesson 1: Manipulating Data with dplyr

> install.packages("swirl")

> library(swirl)

> rm(list=ls())

> swirl()

> install\_from\_swirl("Getting and Cleaning Data")

| One unique aspect of dplyr is that the same set of tools allow you to work with tabular data from a variety of sources, including data frames, data tables, databases and multidimensional arrays. In this lesson, we'll focus on data frames, but everything you learn will apply equally to other formats.

| The first step of working with data in dplyr is to load the data into what the package authors call a 'data frame tbl' or 'tbl\_df'. Use the following code to create a new tbl\_df called cran:

> cran <- tbl\_df(mydf)

From ?tbl\_df, "The main advantage to using a tbl\_df over a regular data frame is the printing." Let's see what is meant by this. Type cran to print our tbl\_df to the console.

| According to the "Introduction to dplyr" vignette written by the package authors, "The dplyr philosophy is to have small functions that each do one thing well." Specifically, dplyr supplies five 'verbs' that cover most fundamental data manipulation tasks: select(), filter(), arrange(), mutate(), and summarize().

> filter(cran, package== "swirl")

> filter(cran, r\_version == "3.1.1", country == "US")

> filter(cran, !is.na(r\_version))

> arrange(cran2, country, desc(r\_version), ip\_id)

| It's common to create a new variable based on the value of one or more variables already in a dataset. The mutate() function does exactly this.

| The size variable represents the download size in bytes, which are units of computer memory. These days, megabytes (MB) are a more common unit of measurement. One megabyte is equal to 2^20 bytes. That's 2 to the power of 20, which is approximately one million bytes!

> mutate(cran3, size\_mb = size/ 2^20 , size\_gb = size\_mb / 2^10 )

| The last of the five core dplyr verbs, summarize(), collapses the dataset to a single row. Let's say we're interested in knowing the average download size. summarize(cran, avg\_bytes = mean(size)) will yield the mean value of the size variable. Here we've chosen to label the result 'avg\_bytes', but we could have named it anything. Give it a try.

### Assignment: swirl Lesson 2: Grouping and Chaining with dplyr

| In the last lesson, you learned about the five main data manipulation 'verbs' in dplyr: select(), filter(), arrange(), mutate(), and summarize(). The last of these, summarize(), is most powerful when applied to grouped data.

| The main idea behind grouping data is that you want to break up your dataset into groups of rows based on the values of one or more variables. The group\_by() function is reponsible for doing this.

> cran <- tbl\_df(mydf)

| At the top of the output above, you'll see 'Groups: package', which tells us that this tbl has been grouped by the package variable. Everything else looks the same, but now any operation we apply to the grouped data will take place on a per package basis.

> summarize(by\_package, mean(size))

Source: local data frame [6,023 x 2]

package mean(size)

(chr) (dbl)

1 A3 62194.96

2 abc 4826665.00

3 abcdeFBA 455979.87

4 ABCExtremes 22904.33

5 ABCoptim 17807.25

6 ABCp2 30473.33

7 abctools 2589394.00

8 abd 453631.24

9 abf2 35692.62

10 abind 32938.88

.. ... ...

# Compute four values, in the following order, from

# the grouped data:

#

# 1. count = n()

# 2. unique = n\_distinct(ip\_id)

# 3. countries = n\_distinct(country)

# 4. avg\_bytes = mean(size)

#

# A few thing to be careful of:

#

# 1. Separate arguments by commas

# 2. Make sure you have a closing parenthesis

# 3. Check your spelling!

# 4. Store the result in pack\_sum (for 'package summary')

#

# You should also take a look at ?n and ?n\_distinct, so

# that you really understand what is going on.

pack\_sum <- summarize(by\_package,

count = n(),

unique = n\_distinct(ip\_id),

countries = n\_distinct(country),

avg\_bytes = mean(size)

)

| We need to know the value of 'count' that splits the data into the top 1% and bottom 99% of packages based on total downloads. In statistics, this is called the 0.99, or 99%, sample quantile. Use quantile(pack\_sum$count, probs = 0.99) to determine this number.

> quantile(pack\_sum$count, probs=0.99)

99%

679.56

| Now we can isolate only those packages which had more than 679 total downloads. Use filter() to select all rows from pack\_sum for which 'count' is strictly greater (>) than 679. Store the result in a new variable called top\_counts.

> filter(pack\_sum, count > 679) -> top\_counts

| Let's take a look at top\_counts. Print it to the console.

> top\_counts

Source: local data frame [61 x 5]

package count unique countries avg\_bytes

(chr) (int) (int) (int) (dbl)

1 bitops 1549 1408 76 28715.046

2 car 1008 837 64 1229122.307

3 caTools 812 699 64 176589.018

4 colorspace 1683 1433 80 357411.197

5 data.table 680 564 59 1252721.215

6 DBI 2599 492 48 206933.250

7 devtools 769 560 55 212932.640

8 dichromat 1486 1257 74 134731.938

9 digest 2210 1894 83 120549.294

10 doSNOW 740 75 24 8363.755

.. ... ... ... ... ...

| There are only 61 packages in our top 1%, so we'd like to see all of them. Since dplyr only shows us the first 10 rows, we can use the View() function to see more.

| View all 61 rows with View(top\_counts). Note that the 'V' in View() is capitalized.

> View(top\_counts)

| arrange() the rows of top\_counts based on the 'count' column and assign the result to a new variable called top\_counts\_sorted. We want the packages with the highest number of downloads at the top, which means we want 'count' to be in descending order. If you need help, check out ?arrange and/or ?desc.

> top\_counts\_sorted <- arrange(top\_counts, desc(count))

| Now use View() again to see all 61 rows of top\_counts\_sorted.

> View(top\_counts\_sorted)

Error in editor(file = file, title = title) :

argument "name" is missing, with no default

| Leaving swirl now. Type swirl() to resume.

> options(editor = "internal")

> swirl()

| Chaining allows you to string together multiple function calls in a way that is compact and readable, while still accomplishing the desired result. To make it more concrete, let's compute our last popularity metric from scratch, starting with our original data.

# Don't change any of the code below. Just type submit()

# when you think you understand it. If you find it

# confusing, you're absolutely right!

result2 <-

arrange(

filter(

summarize(

group\_by(cran,

package

),

count = n(),

unique = n\_distinct(ip\_id),

countries = n\_distinct(country),

avg\_bytes = mean(size)

),

countries > 60

),

desc(countries),

avg\_bytes

)

print(result2)

| In this script, we've used a special chaining operator, %>%, which was originally introduced in the magrittr R package and has now become a key component of dplyr. You can pull up the related documentation with ?chain. The benefit of %>% is that it allows us to chain the function calls in a linear fashion. The code to the right of %>% operates on the result from the code to the left of %>%.

# Read the code below, but don't change anything. As

# you read it, you can pronounce the %>% operator as

# the word 'then'.

#

# Type submit() when you think you understand

# everything here.

result3 <-

cran %>%

group\_by(package) %>%

summarize(count = n(),

unique = n\_distinct(ip\_id),

countries = n\_distinct(country),

avg\_bytes = mean(size)

) %>%

filter(countries > 60) %>%

arrange(desc(countries), avg\_bytes)

# Print result to console

print(result3)

# arrange() the result by size\_mb, in descending order.

#

# If you want your results printed to the console, add

# print to the end of your chain.

cran %>%

select(ip\_id, country, package, size) %>%

mutate(size\_mb = size / 2^20) %>%

filter(size\_mb <= 0.5) %>%

# Your call to arrange() goes here

arrange(desc(size\_mb))

### Assignment: swirl Lesson 3: Tidying Data with tidyr

| In this lesson, you'll learn how to tidy your data with the tidyr package

> library(tidyr)

| The author of tidyr, Hadley Wickham, discusses his philosophy of tidy data in his 'Tidy Data' paper:

|

| http://vita.had.co.nz/papers/tidy-data.pdf

|

| This paper should be required reading for anyone who works with data, but it's not required in order to complete this lesson.

| Tidy data is formatted in a standard way that facilitates exploration and analysis and works seamlessly with other tidy data tools. Specifically, tidy data satisfies three conditions:

|

| 1) Each variable forms a column

|

| 2) Each observation forms a row

|

| 3) Each type of observational unit forms a table

| Any dataset that doesn't satisfy these conditions is considered 'messy' data. Therefore, all of the following are characteristics of messy data, EXCEPT...

1: Column headers are values, not variable names

2: Variables are stored in both rows and columns

3: A single observational unit is stored in multiple tables

4: Multiple types of observational units are stored in the same table

5: Multiple variables are stored in one column

6: Every column contains a different variable

> res <- gather(students2, sex\_class, count, -grade)

> res

grade sex\_class count

1 A male\_1 3

2 B male\_1 6

3 C male\_1 7

4 D male\_1 4

5 E male\_1 1

6 A female\_1 4

7 B female\_1 4

8 C female\_1 4

9 D female\_1 0

10 E female\_1 1

11 A male\_2 3

12 B male\_2 3

13 C male\_2 3

14 D male\_2 8

15 E male\_2 2

16 A female\_2 4

17 B female\_2 5

18 C female\_2 8

19 D female\_2 1

20 E female\_2 7

st

> separate(data=res, col=sex\_class, into=c("sex","class"))

grade sex class count

1 A male 1 3

2 B male 1 6

3 C male 1 7

4 D male 1 4

5 E male 1 1

6 A female 1 4

7 B female 1 4

8 C female 1 4

9 D female 1 0

10 E female 1 1

11 A male 2 3

12 B male 2 3

13 C male 2 3

14 D male 2 8

15 E male 2 2

16 A female 2 4

17 B female 2 5

18 C female 2 8

19 D female 2 1

20 E female 2 7

| Conveniently, separate() was able to figure out on its own how to separate the sex\_class column. Unless you request otherwise with the 'sep' argument, it splits on non-alphanumeric values. In other words, it assumes that the values are separated by something other than a letter or number (in this

| case, an underscore.)

# Repeat your calls to gather() and separate(), but this time

# use the %>% operator to chain the commands together without

# storing an intermediate result.

#

# If this is your first time seeing the %>% operator, check

# out ?chain, which will bring up the relevant documentation.

# You can also look at the Examples section at the bottom

# of ?gather and ?separate.

#

# The main idea is that the result to the left of %>%

# takes the place of the first argument of the function to

# the right. Therefore, you OMIT THE FIRST ARGUMENT to each

# function.

#

students2 %>%

gather(sex\_class ,count, -grade ) %>%

separate(col=sex\_class , into=c("sex", "class")) %>%

print

Variables in both rows and columns

# This script builds on the previous one by appending

# a call to spread(), which will allow us to turn the

# values of the test column, midterm and final, into

# column headers (i.e. variables).

#

# You only need to specify two arguments to spread().

# Can you figure out what they are? (Hint: You don't

# have to specify the data argument since we're using

# the %>% operator.

#

students3 %>%

gather(class, grade, class1:class5, na.rm = TRUE) %>%

spread( test , grade ) %>%

print

name class final midterm

1 Brian class1 B B

2 Brian class5 C A

3 Jeff class2 E D

4 Jeff class4 C A

5 Karen class3 C C

6 Karen class4 A A

7 Roger class2 A C

8 Roger class5 A B

9 Sally class1 C A

10 Sally class3 C B

# We want the values in the class columns to be

# 1, 2, ..., 5 and not class1, class2, ..., class5.

#

# Use the mutate() function from dplyr along with

# extract\_numeric(). Hint: You can "overwrite" a column

# with mutate() by assigning a new value to the existing

# column instead of creating a new column.

#

# Check out ?mutate and/or ?extract\_numeric if you need

# a refresher.

#

students3 %>%

gather(class, grade, class1:class5, na.rm = TRUE) %>%

spread(test, grade) %>%

### Call to mutate() goes here %>%

mutate( class=extract\_numeric(class)) %>%

print

| The fourth messy data problem we'll look at occurs when multiple

| observational units are stored in the same table. students4

| presents an example of this. Take a look at the data now.

# Add a call to unique() below, which will remove

# duplicate rows from student\_info.

#

# Like with the call to the print() function below,

# you can omit the parentheses after the function name.

# This is a nice feature of %>% that applies when

# there are no additional arguments to specify.

#

student\_info <- students4 %>%

select(id, name, sex) %>%

### Your code here

unique%>%

print

| Now, pass as arguments the passed and failed tables (in order) to

| the dplyr function bind\_rows(), which will join them together into

| a single unit. Check ?bind\_rows if you need help.

|

| Note: bind\_rows() is only available in dplyr 0.4.0 or later. If you

| have an older version of dplyr, please quit the lesson, update

| dplyr, then restart the lesson where you left off. If you're not

| sure what version of dplyr you have, type packageVersion('dplyr').

# Accomplish the following three goals:

#

# 1. select() all columns that do NOT contain the word "total",

# since if we have the male and female data, we can always

# recreate the total count in a separate column, if we want it.

# Hint: Use the contains() function, which you'll

# find detailed in 'Special functions' section of ?select.

#

# 2. gather() all columns EXCEPT score\_range, using

# key = part\_sex and value = count.

#

# 3. separate() part\_sex into two separate variables (columns),

# called "part" and "sex", respectively. You may need to check

# the 'Examples' section of ?separate to remember how the 'into'

# argument should be phrased.

#

sat %>%

select(-contains("total")) %>%

gather( part\_sex, count, -score\_range) %>%

### <Your call to separate()>

separate(part\_sex, into=c("part","sex")) %>%

print

Source: local data frame [36 x 4]

score\_range part sex count

(chr) (chr) (chr) (int)

1 700-800 read male 40151

2 600-690 read male 121950

3 500-590 read male 227141

4 400-490 read male 242554

5 300-390 read male 113568

6 200-290 read male 30728

7 700-800 read fem 38898

8 600-690 read fem 126084

9 500-590 read fem 259553

10 400-490 read fem 296793

.. ... ... ... ...

# Append two more function calls to accomplish the following:

#

# 1. Use group\_by() (from dplyr) to group the data by part and

# sex, in that order.

#

# 2. Use mutate to add two new columns, whose values will be

# automatically computed group-by-group:

#

# \* total = sum(count)

# \* prop = count / total

#

sat %>%

select(-contains("total")) %>%

gather(part\_sex, count, -score\_range) %>%

separate(part\_sex, c("part", "sex")) %>%

### <Your call to group\_by()>

group\_by(part, sex) %>%

mutate(total=sum(count), prop=count/total ) %>%

print

Source: local data frame [36 x 6]

Groups: part, sex [6]

score\_range part sex count total prop

(chr) (chr) (chr) (int) (int) (dbl)

1 700-800 read male 40151 776092 0.05173485

2 600-690 read male 121950 776092 0.15713343

3 500-590 read male 227141 776092 0.29267278

4 400-490 read male 242554 776092 0.31253253

5 300-390 read male 113568 776092 0.14633317

6 200-290 read male 30728 776092 0.03959324

7 700-800 read fem 38898 883955 0.04400450

8 600-690 read fem 126084 883955 0.14263622

9 500-590 read fem 259553 883955 0.29362694

10 400-490 read fem 296793 883955 0.33575578

.. ... ... ... ... ... ...

### Quiz: Week 3 Quiz

The American Community Survey distributes downloadable data about United States communities. Download the 2006 microdata survey about housing for the state of Idaho using download.file() from here:

<https://d396qusza40orc.cloudfront.net/getdata%2Fdata%2Fss06hid.csv>

and load the data into R. The code book, describing the variable names is here:

<https://d396qusza40orc.cloudfront.net/getdata%2Fdata%2FPUMSDataDict06.pdf>

Create a logical vector that identifies the households on greater than 10 acres who sold more than $10,000 worth of agriculture products. Assign that logical vector to the variable agricultureLogical. Apply the which() function like this to identify the rows of the data frame where the logical vector is TRUE.

which(agricultureLogical)

What are the first 3 values that result?



153 ,236, 388



236, 238, 262



125, 238,262



59, 460, 474

|  |
| --- |
| > fileUrl <- "https://d396qusza40orc.cloudfront.net/getdata%2Fdata%2Fss06hid.csv"  > getwd()  [1] "C:/Users/rich/Desktop/NEXT/course - coursera data specialization/r\_projects"  > download.file(fileUrl, destfile="2006\_microdata\_survey\_housing\_idaho.csv")  trying URL 'https://d396qusza40orc.cloudfront.net/getdata%2Fdata%2Fss06hid.csv'  Content type 'text/csv' length 4246554 bytes (4.0 MB)  downloaded 4.0 MB  > dt <- read.csv("2006\_microdata\_survey\_housing\_idaho.csv")  > agricultureLogical = ifelse(dt$TYPE==1 & dt$ACR==3 & dt$AGS==6, TRUE, FALSE)  > str(agricultureLogical)  logi [1:6496] FALSE NA FALSE FALSE FALSE FALSE ...  > which(agricultureLogical)  [1] 125 238 262 470 555 568 608 643 787 808  [11] 824 849 952 955 1033 1265 1275 1315 1388 1607  [21] 1629 1651 1856 1919 2101 2194 2403 2443 2539 2580  [31] 2655 2680 2740 2838 2965 3131 3133 3163 3291 3370  [41] 3402 3585 3652 3852 3862 3912 4023 4045 4107 4113  [51] 4117 4185 4198 4310 4343 4354 4448 4453 4461 4718  [61] 4817 4835 4910 5140 5199 5236 5326 5417 5531 5574  [71] 5894 6033 6044 6089 6275 6376 6420 |

2.

Using the jpeg package read in the following picture of your instructor into R

<https://d396qusza40orc.cloudfront.net/getdata%2Fjeff.jpg>

Use the parameter native=TRUE. What are the 30th and 80th quantiles of the resulting data? (some Linux systems may produce an answer 638 different for the 30th quantile)



-15259150 -10575416



-16776430 -15390165



10904118 -594524



-10904118 -10575416

> library(jpeg)

> imgUrl <- "https://d396qusza40orc.cloudfront.net/getdata%2Fjeff.jpg"

> download.file(imgUrl, destfile="quizimg.jpg", mode="wb")

> img <- readJPEG("quizimg.jpg", native=TRUE)

> str(img)

'nativeRaster'

> dim(img)

[1] 180 180

> min(img)

[1] -16776430

> max(img)

[1] -594524

> hist(img)

> # comparing nativeRaster values is not easy, so let's do write/read again

> img2 <- readJPEG(writeJPEG(img, raw()))

> dim(img2)

[1] 180 180 3

> class(img2)

[1] "array"

> class(img)

[1] "nativeRaster"

> min(img2)

[1] 0

> max(img2)

[1] 1

> quantile(img, probs=c(.30,.80))

30% 80%

-15259150 -10575416

3.

Load the Gross Domestic Product data for the 190 ranked countries in this data set:

<https://d396qusza40orc.cloudfront.net/getdata%2Fdata%2FGDP.csv>

Load the educational data from this data set:

<https://d396qusza40orc.cloudfront.net/getdata%2Fdata%2FEDSTATS_Country.csv>

Match the data based on the country shortcode. How many of the IDs match? Sort the data frame in descending order by GDP rank (so United States is last). What is the 13th country in the resulting data frame?

Original data sources:

<http://data.worldbank.org/data-catalog/GDP-ranking-table>

<http://data.worldbank.org/data-catalog/ed-stats>



189 matches, 13th country is St. Kitts and Nevis - not sure why this was answer?



189 matches, 13th country is Spain



234 matches, 13th country is St. Kitts and Nevis



234 matches, 13th country is Spain



190 matches, 13th country is Spain



190 matches, 13th country is St. Kitts and Nevis

|  |
| --- |
| > gdpUrl <- "https://d396qusza40orc.cloudfront.net/getdata%2Fdata%2FGDP.csv"  > edUrl <- "https://d396qusza40orc.cloudfront.net/getdata%2Fdata%2FEDSTATS\_Country.csv"  > download.file(gdpUrl, destfile="gdp.csv")  trying URL 'https://d396qusza40orc.cloudfront.net/getdata%2Fdata%2FGDP.csv'  Content type 'text/csv' length 9630 bytes  downloaded 9630 bytes  > download.file(edUrl, destfile="ed.csv")  trying URL 'https://d396qusza40orc.cloudfront.net/getdata%2Fdata%2FEDSTATS\_Country.csv'  Content type 'text/csv' length 59792 bytes (58 KB)  downloaded 58 KB  > df1 <- read.csv("gdp.csv", skip=4, nrows=232); df2 <- read.csv("ed.csv",blank.lines.skip=TRUE)  > gdp <- tbl\_df(df1); ed <- tbl\_df(df2)  > mdat <- merge(ed,gdp,by.x="CountryCode",by.y="X",all=TRUE)  > mdat2 <- merge(gdp,ed,by.x="X",by.y="CountryCode", all=TRUE)  > sum(!is.na(unique(mdat2$X.1)))  [1] 189  > head(gdp)  Source: local data frame [6 x 10]  X X.1 X.2 X.3 X.4 X.5 X.6 X.7 X.8  (fctr) (int) (lgl) (fctr) (fctr) (fctr) (lgl) (lgl) (lgl)  1 USA 1 NA United States 16,244,600 NA NA NA  2 CHN 2 NA China 8,227,103 NA NA NA  3 JPN 3 NA Japan 5,959,718 NA NA NA  4 DEU 4 NA Germany 3,428,131 NA NA NA  5 FRA 5 NA France 2,612,878 NA NA NA  6 GBR 6 NA United Kingdom 2,471,784 NA NA NA  Variables not shown: X.9 (lgl)  > tail(gdp)  Source: local data frame [6 x 10]  X X.1 X.2 X.3 X.4 X.5 X.6  (fctr) (int) (lgl) (fctr) (fctr) (fctr) (lgl)  1 MNA NA NA Middle East & North Africa 1,540,807 NA  2 SAS NA NA South Asia 2,286,093 NA  3 SSA NA NA Sub-Saharan Africa 1,289,813 NA  4 HIC NA NA High income 49,717,634 NA  5 EMU NA NA Euro area 12,192,344 NA  6 NA NA NA  Variables not shown: X.7 (lgl), X.8 (lgl), X.9 (lgl)  > merged\_gdp\_ranked$X  [1] TUV KIR MHL PLW STP FSM TON DMA COM WSM VCT GRD KNA VUT GNB GMB SLB  [18] SYC ATG LCA TMP BLZ LBR BTN CPV CAF MDV LSO BDI ABW GUY ERI SWZ SLE  [35] TGO FJI MRT BRB MWI MNE SUR BMU GIN MCO KSV KGZ NER TJK RWA MDA BEN  [52] HTI BHS MLT LAO MKD ZWE ARM MDG MNG MLI BFA MUS NIC ALB TCD NAM ISL  [69] COG KHM SEN MOZ BWA JAM PNG GEO BRN ZAR BIH GNQ GAB HND NPL UGA AFG  [86] ZMB EST CYP TTO SLV CIV CMR PRY BOL TZA LVA BHR JOR TKM YEM PAN SRB  [103] KEN GHA ETH LTU LBN MAC CRI SVN TUN URY GTM BGR UZB LUX SDN DOM HRV  [120] LKA BLR AZE CUB OMN SYR ECU SVK MAR PRI AGO BGD HUN VNM KWT NZL QAT  [137] UKR ROM CZE KAZ PER DZA IRQ IRL PRT PAK FIN GRC PHL ISR NGA EGY HKG  [154] CHL SGP MYS DNK ARE THA COL VEN ZAF AUT ARG BEL POL NOR IRN SWE CHE  [171] SAU NLD TUR IDN KOR MEX ESP AUS CAN IND ITA RUS BRA GBR FRA grDEU JPN  [188] CHN USA ADO ASM CHI CYM DJI EAP ECA EMU FRO GRL GUM HIC IMY LAC LBY  [205] LIC LIE LMC LMY MIC MMR MNA MNP NCL PRK PYF SAS SMR SOM SSA TCA UMC  [222] VIR WBG WLD  229 Levels: ABW ADO AFG AGO ALB ARE ARG ARM ASM ATG AUS AUT AZE BDI ... ZWE  > levels(merged\_gdp\_ranked$Long.Name)  [234] |
|  |
| |  | | --- | |  | |

4.

What is the average GDP ranking for the "High income: OECD" and "High income: nonOECD" group?



30, 37



133.72973, 32.96667



23, 30



32.96667, 91.91304



23.966667, 30.91304



23, 45

> merged\_gdp\_ranked %>% filter( Income.Group == "High income: OECD" | Income.Group == "High income: nonOECD") %>% group\_by(Income.Group) %>% summarize(mean(X.1, na.rm=TRUE))

Source: local data frame [2 x 2]

Income.Group mean(X.1, na.rm = TRUE)

(fctr) (dbl)

1 High income: nonOECD 91.91304

2 High income: OECD 32.96667

5.

Cut the GDP ranking into 5 separate quantile groups. Make a table versus Income.Group. How many countries

are Lower middle income but among the 38 nations with highest GDP?



0



5



18



3

|  |
| --- |
| breaks <- quantile(mdat2$X.1, probs=seq(0,1,0.20), na.rm=TRUE)  > breaks  0% 20% 40% 60% 80% 100%  1.0 38.8 76.6 114.4 152.2 190.0  > merged\_gdp\_ranked$quantileGDP <- cut(merged\_gdp\_ranked$X.1, breaks=breaks)  > table(merged\_gdp\_ranked$Income.Group, merged\_gdp\_ranked$quantileGDP)    (1,38.8] (38.8,76.6] (76.6,114] (114,152] (152,190]  0 0 0 0 0  High income: nonOECD 4 5 8 4 2  High income: OECD 17 10 1 1 0  Low income 0 1 9 16 11  Lower middle income 5 13 12 8 16  Upper middle income 11 9 8 8 9 |
|  |
| |  | | --- | |  | |

## Week 4

### Editing Text Variables

* programmatically manipulate text data
  + upper/lower case letters in names
    - tolower()
    - toupper()
  + period separated names
    - splitNames = strsplit(names(cameraData), “\\.”)
    - splitNames[[6]][1]
    - firstElement <- function(x){x[1]}
    - sapply(splitNames, firstElement)
  + underscores
    - names(reviews)
    - sub(“\_”,””,names(reviews),) # one substitution
    - gsub(“\_”,””, testName) # many substitution
* lists
  + named variables
  + subset by name, number
* Search for value names
  + grep(“Alameda”, cameraData$intersection) # returns the indices of variable
  + table(grepl(“Alameda”,cameraData$intersection) # returns table of FALSE, TRUE
  + cameraData2 <- cameraData(!grepl(“Alameda”,intersection))
  + cameraData2 <- grep(“Alameda”, cameraData$intersection, value = TRUE )
  + length(grep(“JeffStreet”,cameraData$intersection) # quick check for existience in list
* String functions
  + library(stringr)
  + nchar(“Jeffrey Leek”) # number characters
  + substr(“Jeffery Leek”, 1,7)
  + paste(“Jeffery”,”Leek”) # can set the sep arg
  + paste0(“Jeffery”,”Leek”) #no space in between
  + str\_trim(“Jeffery “) # take out the extra spaces

Text in Data sets (for R)

* Names of variables should be
  + all lowercase when possible
  + Descriptive (Diagnosis versus Dx)
  + Not duplicated
  + Not have underscores or dots or white space
* Variables with character values
  + Should usually be made into factor variables (depends on application)
  + Should be descriptive (use TRUE/FALSE instead of 0/1, Male/Female instead of M/F)

### Regular Expressions

* set of characters that can be used to apply to a larger set
* Regular Expressions can be thought of as combination of ***literals*** and ***metacharacters***
* in ***natural language***:
  + ***literal*** text forms the *words*
  + ***metacharacters*** define the *grammar*
* Simple pattern is an exact set of literals
* Other expressions
  + whitespace boundaries
  + sets of literals
  + logical combinations

^I think # start of the line

$match # end of the line

[Bb][Uu][Ss][Hh] # any combination of upper/lower characters

[0-9][a-zA-Z] # range of letters

[^?.]$ # carrot in the set is a NOT (exception)

### Regular Expressions

9.11 # “.” references any character in that position

flood|fire # or combine the alternatives

flood|fire|earthquake|coldfire

^[Gg]ood|[Bb]ad #combination (good at start, bad anywhr)

^([Gg]ood|[Bb]ad) # combination, parantheses constrain to # either must be at beginning

[Gg]eorge( [Ww]\.)? [Bb]ush # the ? makes ([Ww]\.) optional

Metacharacters

* “.”, “\*” and “+”
* \* means repetition any number of times to include none
* + at least one repetition
* (.\*) repeat any character any number of times

[Bb]ush( +([^ ]+ +){1,5} debate # locate space followed by not a space followed by a space from 1 to 5 times

m, n # at least m but no more than n matches

m # exactly m matches

m, # at least m matches

### Working with Dates

d1 = date() # date and time at the time of the date() call

# returns a character string

d1 = Sys,date() # returns a date class object

**Formatting dates:**

* %d = day as number (0-31)
* %a = abbreviated weekday
* %A = unabbreviated weekday
* %m = month (00-12)
* %b = abbreviated month
* %B = unabbreviated month
* %y = 2 digit year
* %Y = four digit year

format(d2, “%a %b %d”) # “Sun Jan 12”

**Creating dates:**

x = c(“1jan1960”, “2jan1960”, “31mar1960”, “30jul1960”); z = as.Date(x, “%d%b%Y)

weekdays(d2) # “Sunday”

months(d2) # “January”

Julian(d2) # 16082

attr((,”origin”) #”1970-01-01”

Lubridate

library(lubridate)

ymd(“20140108”)

mdy(“08/04/2013”)

**BLUF**

* **all dates and times should be in class “Date” or “POSIXct” or “POSIXlt”**

### Data Resources

Open Government Sites

* United Nations: <http://data.un.org/>
* U.S. <http://www.data.gov/>
  + List of cities and states with open data
* United Kingdom: <http://data.gov.uk/>
* France: <http://www.data.gouv.fr/>
* Ghana: <http://data.gov.gh/>
* Australia: <http://data.gov.au/>
* Germany: <https://www.govdata.de/>
* Hong Kong: <http://www.gov.hk/en/theme/psi/datasets/>
* Japan: <http://www.data.go.jp/>
* Many More: <http://www.data.gov/opendatasites>

Gapminder: <http://www.gapminder.org> human health data

Survey from U.S. :<http://www.asdfree.com/> help with access and analysis in R

Data Marketplace: <http://www.infochimps.com/marketplace>

Kaggle: <http://www.kaggle.com/> Data science competitions

Collections by data scientists:

* Hilary Mason: <http://bitly.com/bundles/hmason/1>
* Peter Skomoroch: <https://delicious.com/pskomoroch/dataset>
* Jeff Hammerbacher: <http://www.quora.com/Jeff-Hammerbacher/Introduction-to-Data-Science-Data-Sets>
* Gregory Piatetsky-Shapiro: <http://www.kdnuggets.com/gps.html>
* <http://blog.mortardata.com/post/67652898761/6-dataset-lists-curated-by-data-scientists>

Specialized Collections:

* Stanford Large Network Data
* UCI Machine Learning
* KDD Nugets Datasets
* CMU Statlib
* Gene expression omnibus
* ArXiv Data
* Public Data Sets on Amazon Web Services

API’s with R interfaces

* twitter (and twitter package)
* figshare and rfigshare
* PLoS and rplos
* rOpenSci
* Facebook and RFacebook
* Google maps and RGoogleMaps

### Assignment: swirl Lesson 1: Dates and Times with lubridate

| Dates are represented by the 'Date' class and times are represented by the 'POSIXct' and 'POSIXlt' classes. Internally, dates are stored as the number of days since 1970-01-01 and times are stored as either the number of seconds since 1970-01-01 (for 'POSIXct') or a list of seconds, minutes, hours, etc. (for 'POSIXlt').

| Let's start by using d1 <- Sys.Date() to get the current date and store it in the variable d1. (That's the letter 'd' and the number 1.)

> d1 <- Sys.Date()

| You are really on a roll!

|====== | 8%

| Use the class() function to confirm d1 is a Date object.

> class(d1)

[1] "Date"

| You are amazing!

|======== | 11%

| We can use the unclass() function to see what d1 looks like internally. Try it out.

> unclass(d1)

[1] 16893

| What if we need to reference a date prior to 1970-01-01? Create a variable d2 containing as.Date("1969-01-01").

> d2 <- as.Date("1969-01-01")

| Now, let's take a look at how R stores times. You can access the current date and time using the Sys.time() function with no arguments. Do this and store the result in a variable called t1.

> Sys.time()

[1] "2016-04-02 21:05:45 EDT"

| By default, Sys.time() returns an object of class POSIXct, but we can coerce the result to POSIXlt with as.POSIXlt(Sys.time()). Give it a try and store the result in t2. By default, Sys.time() returns an object of class POSIXct, but we can coerce the result to POSIXlt with as.POSIXlt(Sys.time()). Give it a try and store the result in t2.

> t2 <- as.POSIXlt(Sys.time())

[1] "2016-04-02 21:07:06 EDT"

> unclass(t2)

$sec

[1] 16.50706

$min

[1] 7

$hour

[1] 21

$mday

[1] 2

$mon

[1] 3

$year

[1] 116

$wday

[1] 6

$yday

[1] 92

$isdst

[1] 1

$zone

[1] "EDT"

$gmtoff

[1] -14400

attr(,"tzone")

[1] "" "EST" "EDT"

| t2, like all POSIXlt objects, is just a list of values that make up the date and time. Use str(unclass(t2)) to have a more compact view.

> str(unclass(t2))

List of 11

$ sec : num 16.5

$ min : int 7

$ hour : int 21

$ mday : int 2

$ mon : int 3

$ year : int 116

$ wday : int 6

$ yday : int 92

$ isdst : int 1

$ zone : chr "EDT"

$ gmtoff: int -14400

- attr(\*, "tzone")= chr [1:3] "" "EST" "EDT"

| If, for example, we want just the minutes from the time stored in t2, we can access them with t2$min. Give it a try.

> t2$min

[1] 7

| Now that we have explored all three types of date and time objects, let's look at a few functions that extract useful information from any of these objects -- weekdays(), months(), and quarters().

| Use difftime(Sys.time(), t1, units = 'days') to find the amount of time in DAYS that has passed since you created t1.

| Unfortunately, due to different date and time representations, this lesson is only guaranteed to work with an "en\_US.UTF-8" locale. To view your locale, type Sys.getlocale("LC\_TIME").

> Sys.getlocale("LC\_TIME")

[1] "English\_United States.1252"

| lubridate was automatically installed (if necessary) and loaded upon starting this lesson. To build the habit, we'll go ahead and (re)load the package now. Type library(lubridate)

| to do so.

> library(lubridate)

| lubridate contains many useful functions. We'll only be covering the basics here. Type help(package = lubridate) to bring up an overview of the package, including the package DESCRIPTION, a list of available functions, and a link to the official package vignette.

> help(package=lubridate)

| In addition to handling dates, lubridate is great for working with date and time combinations, referred to as date-times. The now() function returns the date-time representing this exact moment in time. Give it a try and store the result in a variable called this\_moment.

| Fortunately, lubridate offers a variety of functions for parsing date-times. These functions take the form of ymd(), dmy(), hms(), ymd\_hms(), etc., where each letter in the name of the function stands for the location of years (y), months (m), days (d), hours (h), minutes (m), and/or seconds (s) in the date-time being read in.

| For a complete list of valid time zones for use with lubridate, check out the following Wikipedia page: http://en.wikipedia.org/wiki/List\_of\_tz\_database\_time\_zones

| Create an interval() that spans from last\_time to arrive. Store it in a new variable called how\_long.

> how\_long <- interval(last\_time, arrive)

| Now use as.period(how\_long) to see how long it's been.

> as.period(how\_long)

[1] "7y 10m 20d 21H 24M 18.0946130752563S"

| This is where things get a little tricky. Because of things like leap years, leap seconds, and daylight savings time, the length of any given minute, day, month, week, or year is relative to when it occurs. In contrast, the length of a second is always the same, regardless of when it occurs.

| To address these complexities, the authors of lubridate introduce four classes of time related objects: instants, intervals, durations, and periods. These topics are beyond the scope of this lesson, but you can find a complete discussion in the 2011 Journal of Statistical Software paper titled 'Dates and Times Made Easy with lubridate'.

### Quiz: Week 4 Quiz 5 questions

Due April 3, 11:59 PM PDT

1.

The American Community Survey distributes downloadable data about United States communities. Download the 2006 microdata survey about housing for the state of Idaho using download.file() from here:

<https://d396qusza40orc.cloudfront.net/getdata%2Fdata%2Fss06hid.csv>

and load the data into R. The code book, describing the variable names is here:

<https://d396qusza40orc.cloudfront.net/getdata%2Fdata%2FPUMSDataDict06.pdf>

Apply strsplit() to split all the names of the data frame on the characters "wgtp". What is the value of the 123 element of the resulting list?



"wgtp"



"" "15"



"wgt" "15"



"wgtp" "15"

> setwd("C:/Users/rich/Desktop/NEXT/course - coursera data specialization/r\_projects")

> getwd()

[1] "C:/Users/rich/Desktop/NEXT/course - coursera data specialization/r\_projects"

> fileUrl <- "https://d396qusza40orc.cloudfront.net/getdata%2Fdata%2Fss06hid.csv"

> download.file(fileUrl, destfile="2006\_micro\_survey\_data.csv")

> dt <- read.csv("2006\_micro\_survey\_data.csv")

> dt\_split <- strsplit(names(dt), "wgtp")

> class(dt\_split)

[1] "list"

> dt\_split[123]

[[1]]

[1] "" "15"

2.

Load the Gross Domestic Product data for the 190 ranked countries in this data set:

<https://d396qusza40orc.cloudfront.net/getdata%2Fdata%2FGDP.csv>

Remove the commas from the GDP numbers in millions of dollars and average them. What is the average?

Original data sources:

<http://data.worldbank.org/data-catalog/GDP-ranking-table>



381668.9



387854.4



377652.4



381615.4

> fileUrl <- "https://d396qusza40orc.cloudfront.net/getdata%2Fdata%2FGDP.csv"

> download.file(fileUrl, destfile="gdp\_data.csv")

trying URL 'https://d396qusza40orc.cloudfront.net/getdata%2Fdata%2FGDP.csv'

Content type 'text/csv' length 9630 bytes

downloaded 9630 bytes

> gdp <- read.csv("gdp\_data.csv")

> gdp\_vals <- as.numeric(gsub(",","", gdp[5:194,5]))

> class(gdp\_vals)

[1] "numeric"

> mean(gdp\_vals)

[1] 377652.4

3.

In the data set from Question 2 what is a regular expression that would allow you to count the number of countries whose name begins with "United"? Assume that the variable with the country names in it is named countryNames. How many countries begin with United?



grep("^United",countryNames), 3



grep("\*United",countryNames), 2



grep("United$",countryNames), 3



grep("\*United",countryNames), 5

> united <- gdp[grep("^United",gdp$X.2),4]

> united

[1] United States United Kingdom United Arab Emirates

230 Levels: East Asia & Pacific Euro area ... Zimbabwe

4.

Load the Gross Domestic Product data for the 190 ranked countries in this data set:

<https://d396qusza40orc.cloudfront.net/getdata%2Fdata%2FGDP.csv>

Load the educational data from this data set:

<https://d396qusza40orc.cloudfront.net/getdata%2Fdata%2FEDSTATS_Country.csv>

Match the data based on the country shortcode. Of the countries for which the end of the fiscal year is available, how many end in June?

Original data sources:

<http://data.worldbank.org/data-catalog/GDP-ranking-table>

<http://data.worldbank.org/data-catalog/ed-stats>



15



16



7



13

> fileUrl1 <- "https://d396qusza40orc.cloudfront.net/getdata%2Fdata%2FGDP.csv"

> fileUrl2 <- "https://d396qusza40orc.cloudfront.net/getdata%2Fdata%2FEDSTATS\_Country.csv"

> download.file(fileUrl1, destfile="gdp\_data\_ranked.csv")

trying URL 'https://d396qusza40orc.cloudfront.net/getdata%2Fdata%2FGDP.csv'

Content type 'text/csv' length 9630 bytes

downloaded 9630 bytes

> download.file(fileUrl2, destfile="ed\_data.csv")

trying URL 'https://d396qusza40orc.cloudfront.net/getdata%2Fdata%2FEDSTATS\_Country.csv'

Content type 'text/csv' length 59792 bytes (58 KB)

downloaded 58 KB

> gdp <- read.csv("gdp\_data\_ranked.csv")

> ed <- read.csv("ed\_data.csv")

> gdp\_countryCodes <- as.character(gdp[5:194,1])

> class(gdp\_countryCodes)

[1] "character"

> length(gdp\_countryCodes)

[1] 190

> gdp\_countryCodes <- as.character(gdp[5:194,1])

> ed\_countryCodes <- as.character(ed[,1])

> gdp\_countryCodes2[gdp\_countryCodes2==""] <- NA

> gdp\_countryCodes2 <- gdp\_countryCodes2[is.na(gdp\_countryCodes2)== 0]

> notes <- as.character(ed[,10])

> grep("Fiscal year end",notes)

[1] 3 11 18 31 33 60 65 76 90 92 94 96 103 105 112 126 143 151 154 162 163 166

[23] 173 186 188 199 200 206 216 220 231 234

> grep("Fiscal year end: June",notes)

integer(0)

> grep("Fiscal year end: June",notes)

[1] 11 18 31 60 76 105 112 166 173 188 199 216 234

> matches <- as.character(ed[grep("Fiscal year end: June",notes),1])

> matches

[1] "AUS" "BGD" "BWA" "EGY" "GMB" "KEN" "KWT" "PAK" "PRI" "SLE" "SWE" "UGA" "ZWE"

> length(matches)

[1] 13

5.

You can use the quantmod (<http://www.quantmod.com/>) package to get historical stock prices for publicly traded companies on the NASDAQ and NYSE. Use the following code to download data on Amazon's stock price and get the times the data was sampled.

library(quantmod)

amzn = getSymbols("AMZN",auto.assign=FALSE)

sampleTimes = index(amzn)

How many values were collected in 2012? How many values were collected on Mondays in 2012?



250, 47



250, 51



252, 50



251, 47

|  |
| --- |
| > library(lubridate)  > years <- sampleTimes[which(year(sampleTimes)==2012)]  > mondays <- sampleTimes[which(wday(years, labels=TRUE)=="Mon")]  Error in wday(years, labels = TRUE) : unused argument (labels = TRUE)  > mondays <- sampleTimes[which(wday(years, label=TRUE)=="Mon")]  > mondays  [1] "2007-01-09" "2007-01-23" "2007-01-30" "2007-02-06" "2007-02-13" "2007-02-27"  [7] "2007-03-06" "2007-03-13" "2007-03-20" "2007-03-27" "2007-04-03" "2007-04-10"  [13] "2007-04-17" "2007-04-24" "2007-05-01" "2007-05-08" "2007-05-15" "2007-05-22"  [19] "2007-06-05" "2007-06-12" "2007-06-19" "2007-06-26" "2007-07-03" "2007-07-10"  [25] "2007-07-17" "2007-07-24" "2007-07-31" "2007-08-07" "2007-08-14" "2007-08-21"  [31] "2007-08-28" "2007-09-11" "2007-09-18" "2007-09-25" "2007-10-02" "2007-10-09"  [37] "2007-10-16" "2007-10-23" "2007-11-02" "2007-11-09" "2007-11-16" "2007-11-23"  [43] "2007-11-30" "2007-12-07" "2007-12-14" "2007-12-21" "2007-12-28"  > length(mondays)  [1] 47  > length(years)  [1] 250 |
|  |
| |  | | --- | |  | |

### Assignment: Getting and Cleaning Data Course Project (30 min)

Due April 3, 11:59 PM PDT

Instructions

The purpose of this project is to demonstrate your ability to collect, work with, and clean a data set.

**Review criteria**

1. The submitted data set is tidy.
2. The Github repo contains the required scripts.
3. GitHub contains a code book that modifies and updates the available codebooks with the data to indicate all the variables and summaries calculated, along with units, and any other relevant information.
4. The README that explains the analysis files is clear and understandable.
5. The work submitted for this project is the work of the student who submitted it.

**Getting and Cleaning Data Course Project**

The purpose of this project is to demonstrate your ability to collect, work with, and clean a data set. The goal is to prepare tidy data that can be used for later analysis. You will be graded by your peers on a series of yes/no questions related to the project. You will be required to submit: 1) a tidy data set as described below, 2) a link to a Github repository with your script for performing the analysis, and 3) a code book that describes the variables, the data, and any transformations or work that you performed to clean up the data called CodeBook.md. You should also include a README.md in the repo with your scripts. This repo explains how all of the scripts work and how they are connected.

One of the most exciting areas in all of data science right now is wearable computing - see for example [this article](http://www.insideactivitytracking.com/data-science-activity-tracking-and-the-battle-for-the-worlds-top-sports-brand/). Companies like Fitbit, Nike, and Jawbone Up are racing to develop the most advanced algorithms to attract new users. The data linked to from the course website represent data collected from the accelerometers from the Samsung Galaxy S smartphone. A full description is available at the site where the data was obtained:

<http://archive.ics.uci.edu/ml/datasets/Human+Activity+Recognition+Using+Smartphones>

Here are the data for the project:

<https://d396qusza40orc.cloudfront.net/getdata%2Fprojectfiles%2FUCI%20HAR%20Dataset.zip>

You should create one R script called run\_analysis.R that does the following.

1. Merges the training and the test sets to create one data set.
2. Extracts only the measurements on the mean and standard deviation for each measurement.
3. Uses descriptive activity names to name the activities in the data set
4. Appropriately labels the data set with descriptive variable names.
5. From the data set in step 4, creates a second, independent tidy data set with the average of each variable for each activity and each subject.

Good luck!

**Wearable Computing**

* data science used to be domain of large companies with resources and know-how
* today there is a leveling of the capability
  + data collection – open source data stacks
  + data analysis – open source data stacks
  + driving cost of insights and predictive modeling down
* more data inputs
  + social
  + mobile
  + sensor-driven
* Every day 2.5 quintillion (2,500,000,000,000,000) 2.5 million billion bytes of data are being created.
  + cameras
  + phones
  + credit cards
  + web surfing
  + GPS devices
  + sensors in devices (accelerometers, thermometers, etc.)
* Wearable computing
  + Apparel that monitors, syncs blood pressure, heart rate, and other vitals every 10 minutes
  + prescription medicines
* Well done data analysis can
  + enable insights and actions that improve our lives (consumers)
  + create insights into emerging behaviors that enable business rules and applications that make business more efficient and create competitive advantages
* Innovation - Availability of data, granularity of data, and ability to draw insightss
  + identity linking of any product purchase
  + identity linking to other social media information (facebook)
  + identity linking to purchase histories (credit card or digital wallet)
  + Devices aren’t the value proposition – it is the data stream that comes from them (iPhone)
* Analyst Skills
  + real-time analytics
  + real-time predictive modeling
  + complex data processing
  + application development (in real time) to process the data

### Review Classmates: Getting and Cleaning Data Course Project (30 min)

Due April 6, 11:59 PM PDT

### Post-Course Survey

# Exploratory Data Analysis

Starts: 4 April Ends: 11 May

**Exploratory Data Analysis.**

This course covers the essential exploratory techniques for summarizing data. These techniques are typically applied before formal modeling commences and can help inform the development of more complex statistical models. Exploratory techniques are also important for eliminating or sharpening potential hypotheses about the world that can be addressed by the data. We will cover in detail the plotting systems in R as well as some of the basic principles of constructing data graphics. We will also cover some of the common multivariate statistical techniques used to visualize high-dimensional data.

Before you officially start the course, you may be interested in obtaining the book [Exploratory Data Analysis with R](https://eventing.coursera.org/redirectSigned/eyJrZXkiOiJlbWFpbC5saW5rLm9wZW4iLCJ2YWx1ZSI6eyJ1cmwiOiJodHRwczovL2xlYW5wdWIuY29tL2V4ZGF0YS8_.PWUKh134YK6Zg6a1STUMgf8LmiesoPEktomB9S6iLEk), which is the official textbook for this course.

## Background Material

### Welcome to Exploratory Data Analysis

I'm very excited to start Exploratory Data Analysis, and I hope you are too. This is the fourth course in the [Data Science Specialization](https://eventing.coursera.org/api/redirectStrict/dDEpfcE82Tifg8jjEcrJUogDCPpKUnggpCGXOjbEmXY3iXMfyNXgLMi7qRnwIBXUVUErV_Xtkzun95kyQtRtOw.ZUlU2LYOHxbf2R-7QdUpmQ.xMruQ5A81IPW_rae6HKYXt8RyZnYLfrl9X7ufpDS-njU-zJspQFFv7WRYmZudvGDGpJhB2XvwjzHNNG3ARGsxIR1sqFKm_ExxcRR3hC9MQO1ffxKAdAg70zafxxr3eIJpVe_wMGcE1GIeyQTiF774c7UFsWQeLV_qsrxFb0-wXzmYGjn7EkP_aoQf9bZCU253JYeWB8NyQWXJjKoUlcLXMjPWf2uSNRYML5Zzg0wZUNpXEjV2eV44r9eGJXEAssa-nTAKsqX_038szHz_6qc9_Va6OFML43OT412s6Au7hzCciKzHqeLBx9ueYVwpB86XPdB7g0JhJIj9EAuC1GLXDaBI3Wx7ZyVPPY8823JGp-9dnorXmnTBZBGgoaK9Z4_). Exploratory data analysis (EDA) is a key element of data science because it allows you to develop a rough idea of what your data look like and what kinds of questions might be answered by them. EDA is often the "fun part" of data analysis, where you get to play around with the data and, well, explore!

The recommended background courses are [The Data Scientist's Toolbox](https://eventing.coursera.org/api/redirectStrict/0XbUZXERVuoMmbW-13SythZoTkUzriehpJOm83AivATFjI4HDAfx9mB9wLMhi3_shwEsgtmgRMYmso6A3y3dSg.UvdjLKddvJkuAFQL3zz99g.__aHd6WylOV3iGWlGWby2_PCZu3V4_JPZEOLIOFk0iHH24Qs4zoF4UYh6YmiSTeNBW7b0ybHmdJ31VOdUc3zLNX1lckkHnbmFZU0rruoaYsZ6XLMeMfe5aa2WZIHHjA7vTS0VOyNd1sndpODKemeT87arHiKdedqYud8kAVB_6vlBxzjIn0zJpmbf7yFBiQHiyrG1MvKDs7QHip3aV54RyTQQaUPkclQnmmPw0ASvwU2eGRHC33dBC9tOZtL0v-dRPiCV5ApMpmedc_N1Q-XLb-YQ4CxdYGVI-lAVdxxc_EUkuKDYra2vRHOXByQfcIRyxeUWR6qtfj1JQeG9Lqou8RzQqAqrITRhZ52VYcSXfA) and [R Programming](https://eventing.coursera.org/api/redirectStrict/tH36r4OH_83zfnK0vGgLl9g2exmHcjtDupLePF70MhtM7-3Sker5ZfYRpBuXa2_H4RnTwjnKScv0ftzOlj9c1w.AYBP85-qn3o3PkVnN3nrrA.bXqoFZio-JyosT6eim1o1MCQgjMc2YUBdquFHNyp-lxeo6sFsZnjDDY0ckbx02RTng0y_7AqaYI2UXKvsWhpVJH-nJ9c7_8f2mWn4B0TaT-GTcJLxwxF9j37557OKKH1-jYUxho6HETSaPI96N23X6FarjADLkr0J5yyB_5SHHE1HLq2DXsbEPIrZWYwfFUoSrZGFEjp0JLjtdQDrQYc4LVulZvYepEd7Mbxydtd7HNTWyYv72ja9L_wW5fBlYUxellWUovRNk-Zq8jlY1YLiKT1mXnAhbTGIzKEPTpH_njr0kF9w8qHhoBeGArDNXFd). It is possible to take this class concurrently with that class but you may have to read ahead in the prerequisite class to get the relevant background for this class. For a complete set of course dependencies in the Data Science Specialization please see the [course dependency chart](https://eventing.coursera.org/api/redirectStrict/z2f8YRaFvs3QiHrRmc1t3zt0eRMCPA8s92ZnK27cy9MOdx3EAfboa9yWzYHdbzjNWubm0JfgjaOhF7Tj-c2Q1g.vXcc9sLSxjsUD9-8W9sttg.01CSf9vNYqasdHsTM6CdK560zr3TRs-zQSC9nKZVc8gIkRlfAdh_gCKhEg8SB4j_4MlSvOWA9HyILsQ-bSXbDcbP449r3nRrijEGSzQek9vARXj_1l7hJiliOe5fWU7bjrQ3RzONiYxm7osvlpHcZk9nrhEC-9COaA2bc22LCG3FNyNZx_XrF8CfWXYp_Uj5I1p90j2fo5ouaY9x0iKZAZROJeoCf0ccwxTfOlkdhdr7YLw60xfPrz0zqfR-ZLdcFBQyw5OfQStSwdbbISlWA7ezf-ADeQId75ThpZy9tUADJtYrXPkvaThsJPIT7vIRjRfOL7L4AUcdwlsbmqHSqlXyfXdxqDOHbHABmuDDVHeB79RRo8dZavUBeYBi3qAwwyx4jgg17fRMdunTywkCyMK341NxE4oxD5AYH_biTSo).

### Syllabus

**Course Title:** Exploratory Data Analysis

**Course Instructor:**Roger D. Peng

Course Description:

This course covers the essential exploratory techniques for summarizing data. These techniques are typically applied before formal modeling commences and can help inform the development of more complex statistical models. Exploratory techniques are also important for eliminating or sharpening potential hypotheses about the world that can be addressed by the data. We cover in detail the plotting systems in R as well as some of the basic principles of constructing data graphics. We also cover some of the common multivariate statistical techniques used to visualize high-dimensional data.

Course Content:

* Making exploratory graphs
* Principles of analytic graphics
* Plotting systems and graphics devices in R
* The base, lattice, and ggplot2 plotting systems in R
* Clustering methods
* Dimension reduction techniques

Course Textbook

The book [*Exploratory Data Analysis with R*](https://eventing.coursera.org/api/redirectStrict/R-WE54gjZX_4jfQia3rfXIIWX3IUtMtnGFA0BIwPG6vI5_MlRaanNMQKFfGn6IJvuCb_rY6eqCIG0djpSygfIA.c49DuNz1bqeUPFA85Y2uDA.pYxeMqZtiSC0fIVHP4VHxsmbati4Or3NDG5Eli-WcBLuwEmtv_ehURLMSsD3ZyRY3b6ZLEZdAorEJj4-zYZNEgz_2yiji7jRE-TfeaTiXQX0Kx64KMYwYzNCP4-PLHg5ESej_-JlYKdHJgsf2BIrq2x1x1ecMXncf53NR5dqlIIN7rt5-oftL7o4S0XyZ-IP2zcBzBDFS83Ox6B_W7kNFvGYNQLpuENaEYJmcXv7RUoKWMTjj66ejH0ZEamBI_KwTdMacLYp5hDfsLAM81D9hOq50KBacEn-bpQVfVsmw9o) covers the lecture material in this course.

Weekly quizzes

* There are two quizzes (Weeks 1 and 2).
* You must earn a grade of at least 80% to pass a quiz.
* You may attempt each quiz up to 3 times in 8 hours.
* The score from your most successful attempt will count toward your final grade.

The Course Projects

The two course projects will be assessed via peer assessment. In these projects, you will be asked to construct or reproduce certain plots, the purpose of which is to make you familiar with various plotting options. You will be evaluated on the plot that you produce and the code that you write to construct the plot. Course projects evaluated via peer assessment will make use of your GitHub account.

For each Course Project, you are required to evaluate and grade at least four of your classmates' projects.

Grading policy

You must score at least 80% on all assignments (Quizzes & Projects) to pass the course.

Your final grade will be calculated as follows:

* Quiz 1 = 20%
* Quiz 2 = 20%
* Course Project 1 = 25%
* Course Project 2 = 35%

**Differences of opinion**

Keep in mind that currently data analysis is as much art as it is science - so we may have a difference of opinion - and that is ok! Please refrain from angry, sarcastic, or abusive comments on the message boards. Our goal is to create a supportive community that helps the learning of all students, from the most advanced to those who are just seeing this material for the first time.

**Plagiarism**

Johns Hopkins University defines plagiarism as "...taking for one's own use the words, ideas, concepts or data of another without proper attribution. Plagiarism includes both direct use or paraphrasing of the words, thoughts, or concepts of another without proper attribution." We take plagiarism very seriously, as does Johns Hopkins University.

We recognize that many students may not have a clear understanding of what plagiarism is or why it is wrong. Please see the JHU referencing guide for more information on plagiarism.

It is critically important that you give people/sources credit when you use their words or ideas. If you do not give proper credit -- particularly when quoting directly from a source -- you violate the trust of your fellow students.

The Coursera Honor code includes an explicit statement about plagiarism:

I will register for only one account. My answers to homework, quizzes and exams will be my own work (except for assignments that explicitly permit collaboration). I will not make solutions to homework, quizzes or exams available to anyone else. This includes both solutions written by me, as well as any official solutions provided by the course staff. I will not engage in any other activities that will dishonestly improve my results or dishonestly improve/hurt the results of others.

**Reporting plagiarism on course projects**

One of the criteria in the project rubric focuses on plagiarism. Keep in mind that some components of the projects will be very similar across terms and so answers that appear similar may be honest coincidences. However, we would appreciate if you do a basic check for obvious plagiarism and report it during your peer assessment phase.

It is currently very difficult to prove or disprove a charge of plagiarism in the MOOC peer assessment setting. We are not in a position to evaluate whether or not a submission actually constitutes plagiarism, and we will not be able to entertain appeals or to alter any grades that have been assigned through the peer evaluation system.

But if you take the time to report suspected plagiarism, this will help us to understand the extent of the problem and work with Coursera to address critical issues with the current system.

### Introduction

* should have basics in data analysis toolbox
* familiar with R programming
* familiar with variety of means to reach out and get data
* Exploratory Data Analysis
  + Before forming a model (no preconceptions)
  + Look at data and get a sense of what is going on
  + Before any sort of prediction or inference

### Exploratory Data Analysis with R Book (LeanPub)

I'm happy to announce that [*Exploratory Data Analysis with R*](https://eventing.coursera.org/api/redirectStrict/_9l69wHZyrBEtdrCNLxspNSFcuCAMXtz0aPlYHi4afI79lpRtA-s1VAEkdmVg4t8691H18xEhRimDZfYq2p_Nw._eqHE6MZ7E3N0XGTOwbP2w.8x83Y2nBkfPsax-FyLtwf7L9RPLYty0QDqpvyyUYxnTw57SzKNrs4hg6BsuCP3h30HE30pbjxYByABKXAOU2qFx2j1Cp0JzGDC28kkhtidpWQ0r3i8R5GROiW0iR_7OHnLbkttlgAhjXkO13aNmqUdIDHt3MhAK0VnN42zJU-HX_a-cD8iUc25ihSSux7jT78izteGY49vNwjTddWAqGIFQSzfUjvwlu0qzanZ2pvq5aN7JLrIM982JjWpkjsTrL7uLhOIRcxLS1AainkwPsIXQUU1avXP1SfS12hMc7rAO6EVyV7XUDNCJSq2_Tgf0SIUN7N74cLGUgwdp8MXmXQ4gRgl_pJZQqEDkbdeENrcMxDhvYLYlCS8l1msFFIrjttKfF0xLXx9XWfK6Aax2CIKHCua-sF1xDhKrkgF73ZJHtngkQrZPpEYo3HOvvfFh8chiG2WS-5oR4TqCa9XHipg) is available through Leanpub. This e-book is specifically designed for use with this course and compiles together all of lecture notes for the course in one place. I think you'll find it useful both for completing this course and for completing the rest of the Data Science Specialization. In addition to allowing you to have all the lecture content in one place, the e-book is a way for you to access the course materials after the class has finished. If you are interested in printed paper copies, they are[available from Lulu](https://eventing.coursera.org/api/redirectStrict/SoEiev1pH1sFcE1uvE5a79Lmq7WgfpdLWm9lDMGDTNraELXw3uEcnoIMulP7iUhfx_q1k4UXh0IdbHmXdX5ZmQ.URrx4tIwpVsrba_48i1UGw.5LLFsSMYPH1gXi0dejmFf618rW6L8-0q5Yf-znm8Qy7jEBpR74T7HG_6uYXTbGcctZhoBjNAg6_DHyMiVOOt8LfC-d-RHlmqdNTE75N9QGu-2R7pHzhMWLGPQpcBI10zYndWMRkr0TYD8T7T2NhvfdnFQ-KkSOn4eJLpHpZK2prZSOGXx60rKPo-51aQo77yjT3Gb0SSLTyt0JEyp8qRB5cIowwkOuFLryuyY1QOqIfQ4WdZVejhA67l7dxyPyOroPVrIae7Gd_sST8h3zGLXbGgXdofgZr3IgnVWu8LF-SpvjyCW7w9C75yNPDuxqJhBMYTXN0iCWVFJFpu_fgY0rm6ibBnxdzBGQHVwnpgNVZb_TadMxcu1zEiNGE8SebIw4-otP3dMZjzDLmsYRhBw3l5ozDveB1NAN1t9pLYSG8VuUICjjHvU2GEQQlcQEmC).

### The Art of Data Science (LeanPub)

As you make your way through the world of data science, learning R programming and other important skills, it's important to remember that data science isn't just a collection of tools. It requires a person to apply those tools in a smart way to produce results that are useful to people.

I'm delighted to announce that my latest book, [*The Art of Data Science*](https://eventing.coursera.org/api/redirectStrict/vbrp1ubAdlwSAwxP5NhgED2Fz_qO_8Z-YQeQrwf5No0t7XZP3EIwYoaj8ERph66CCs4ahJmHBMD5xGZzuya1AA.Pwn8ctmkbnm3HPOiXHYIug.9a5KQxbt2P0jw1Ge-fjhWpu0NsG6osm1gQtahT6QnA8wIGYmmmjEHuBwgpyt2IT2ePuLgim9YPYYnQLc0kQ8UhzKc2OVny1GekDbtmgF0qyaw6i7Q7h0Pym8g3QhmA4naAsnRoXT31UVurb2nEHwPZicmBEKO_29yM_HrjLqJRMV_DB06GGSPjIytwdBCAEiV9onQycJA5T5DBQYhwXjB9NzL90mWU7PunCTnzYGHxtKyYpLyp9QpejhvGA6MTxmfBURF5PcjS7j-bCRrV-MngN1jAK4EZ8VBvmm4l8wjTqEGc6uZNpH3GFO9ah_4eJDBRGbW9TpO8VgqfUHgZ6qNQ), is available from Leanpub (printed paper copies are [available from Lulu](https://eventing.coursera.org/api/redirectStrict/qr2tFKtx9cvkWGakCrA_-vwyxWOAFXITf9T7YYyoTqLsaE_Xto0HciGMDb6AmuvI8pXGwZbX2W9xzmnYKhzhaQ.-5IWqRCWsF5mzeMHGUyEyg.y46P6zgBIIZDrNyklX8L5GrAKF0hAboirV53lkiPgjVRw_E3cwWA01uJrCWnhddztNFXM1zocqiI1LhBnXBP-2u3Efw_mZJ9faYOY64WnC_qt8eTsFKYpb6hu-AbHgvyP7vCxcRZ4I6rS-13imJPoXPvfnFqf6_Jufx0C4dL779wjRojznU4axNeOE7ZccZkk7p4FEgNkQikm-yPGU3SVUbiuAvOTCuqgaVkaKBjjcvscQUetXNokCh8V90Q7dCYvOv0IjOaB3sYTHoBIgbzSTGGBgY_gUpFDAuoaJ2CMW-Pojq-0oAaRZ854znUZu7ApEUEDGPU1dRdcGjla6HMXHDi-CSJHw3YX_Zai8jFuOF7w6W8_qza_1SP9h66L_gSmbT4ncrBp9_AMizNZlvQx2XgQDrB_qDvRznmnqVfr_I)). This book addresses the problem of how to think about any data analysis, and how to assemble the tools you've learned and apply them from start to finish. This is the book I wish I'd had when I was first starting out.

I hope you enjoy the book and find it useful. I welcome any feedback as we are continuously working to improve the book.

### Installing R on Windows (3.2.1)

### Installing R on a Mac (3.2.1)

### Installing R Studio (Mac)

### Setting Your Working Directory (Windows)

### Setting Your Working Directory (Mac)

getwd()

setwd()

dir()

ls()

source(“myScript.R”)

## Lesson 1: Graphs

### Principles of Analytic Graphics

* **Show Comparisons**
  + evidence for a hypothesis is always relative to another competing hypothesis
  + Always ask: “Compared to What?” when looking at summary of evidence
    - compare to a ***control*** case
  + usually show comparison in a plot to show the relative values
* Show causality, mechanism, explanation, systematic structure
  + What is your causal framework for thinking about the question?
  + Show the causal relationship also between control and test case
* Show multivariate data
  + Multivariate = more than two variables
  + The real world is multivariate
  + Need to escape flat land – 2D plots
  + Small multiples to show the more than 2-dim (seasons)
* Integrate Evidence
  + Completely integrate words, numbers, images, diagrams
  + Data graphics should make use of many modes of data presentation
  + Don’t let the tools drive the analysis
* Describe and document the evidence with labels, scales, sources, etc.
  + graphic should be credible
  + show sources
  + show source code and calculations
* Content is King
  + Analytical presentations ultimately stand or fall depending on quality, relevance and integrity of evidence

### Exploratory Graphs (part 1)

* Graphs made for internal use to look at data and think
  + understand data properties
  + find patterns in data
  + suggest modeling strategies
  + “debug” analyses
  + NOT NECESSARILY GOING TO BE USED to communicate results
* Characteristics of exploratory graphs
  + Quickly made
  + Large number are made
  + Goal is to get personal understanding
  + Axes/Legends are generally cleaned up later
  + Color/size are primarily used for information (not communication ready)
* Example: US EPA sets national ambient air pollution standards for outdoor air pollution
  + Fine Particle Pollution (PM2.5) the annual mean averaged over 3 years cannot exceed 12µg/m3
  + U.S. National Ambient Air Quality Standards
  + EPA Air Quality System
  + Question: are there any counties in United States that exceed the national standard for fine particle pollution?

**Simple Summaries of Data**

* Five Number Summary: Minimum, 1st Quartile, Median, 3rd Quartile, Maximum

summary() # adds the Mean in R

summary(pollution$pm25)

* Box Plot

boxplot(pollution$pm25, col=”blue”)

* Histogram
  + Don’t want too many bars (too noisy) or too few

hist(pollution$pm25, col=”green”)

rug(pollution$pm25) # goes under histogram with where the data is

hsit(pollution$pm25, col=”green”, breaks = 100) # change the breaks (bins)

* Overlaying Features

boxplot( pollution$pm25, col=”blue”)

abline ( h = 12 ) # adds a line at a value for comparison

hist(pollution$pm25, col=”green”)

abline(v =12, lwd = 2) # add line to histogram (median)

abline(v=median(pollution$25), col = “magenta”, lwd = 4) # adds colored line

* Barplot

barplot(table(pollution$region), col = “wheat”, main = “Number of Counties in Each Region”)

* Two Dimensions
  + Mulitple/Overlay 1-D plots (Lattice/ggplot2)
  + Scatterplots
  + Smooth scatterplots
* 2 Dimensions
  + overlayed/multiple 2-D plots; coplots
  + Use color, size, shape to add dimension
  + Spinning plots
  + Actual 3-D plots (not that useful)
* Multiple Boxplots

boxplot(pm25 ~ region, data = pollution, col = “red” )

* Multiple Histograms

par(mfrow = c(2,1), mar = c (4,4,2,1)

hist ( subset (pollution, region == “east”)$pm25, col = “green” )

hsit (subset (pollution, region == “west” )$pm25, col = “green” )

* Scatterplot

with ( pollution, plot ( latitude, pm25) )

abline( h = 12, lwd = 2, lty = 2 )

with ( pollution, ,plot (latitude, pm25, col = region ) ) # color the data dots according to region

abline ( h=12, lwd = 2, lty = 2 )

* Multiple Scatterplots

par ( mfrow = c(1,2), mar = c(5,4,2,1) )

with (subset (pollution, region = “west”), plot ( latitude, pm25, main = “West”) )

with ( subset(pollution, region = “east”), plot ( latitude, pm25, main = “West:” ))

### Resources for graphing in R

* R Graph Gallery
* R Bloggers

## Lesson 2: Plotting

### Plotting Systems in R

### Base Plotting System (part 1)

### Base Plotting System (part 2)

### Base Plotting Demonstration

## Lesson 3: Graphics Devices

### Graphics Devices in R (part 1)

### Graphics Devices in R (part 2)

## Practical R Exercises in swirl (unit 4, part 1)

### Assignment: swirl Lesson 1: Principles of Analytic Graphs

### Assignment: swirl Lesson 2: Exploratory Graphs

### Assignment: swirl Lesson 3: Graphics Devices in R

### Assignment: swirl Lesson 4: Plotting Systems

### Assignment: swirl Lesson 5: Base Plotting System

### Quiz: Week 1 Quiz10 questions

Due in 10 April

### Assignment: Course Project

Due on 10 April

### Review Classmates: Course Project 1

# Reproducible Research

# Statistical Inference

# Regression Models

# Practical Machine Learning

# Developing Data Products

# Data Science Capstone

# Data Analysis Style Guide (R)

* Download the data
  + check if it exists
  + give location
  + download the file
  + read the file (csv, html)
* Create a data structure
* Explore data
  + sizeof()
  + head()
  + tail()
  + names()
  + str()
  + class()
  + Range of data: summary statistics (min, max, median, std dev)
* Clean and prepare data for analysis
  + missing values
  + Not available (N/A)
  + not a number (NAN)
* Analysis branches
* Run Script
  + source(“file.R”)

**Download**

if(!file.exists(“./data”)){dir.create(“./data”)}

fileUrl <- “https://data.baltimorecity.gov/api/views/etc”

download.file(fileUrl, destfile=”./data/cameras.csv”, method=”wb”)

cameraData <- read.csv(“./data/cameras.csv”)get

Example to set the column classes:

pollution <- read.csv(“data/avgpm25.csv”, colClasses = c(“numeric”,

“character”, “factor”, “numeric”, “numeric”) )

**Introduction to swirl**

* Statistics With R Learning
* Optional for the course.
* Install swirl: install.packages(“swirl”)
* Check version: packageVersion(“swirl”)
* Load swirl: library(swirl)
* Install R Programming course in swirl: install\_from\_swirl(“R Programming”)
* Start swirl: swirl()

**Git Repository Commands**

* build a remote repository on GitHub then:

echo "# cleaningdataproject" >> README.md

git init

git add README.md

git commit -m "first commit"

git remote add origin https://github.com/rbmorrison/cleaningdataproject.git

git push -u origin master

# CodeBook Examples

Getting and Cleaning Data: Course Project

=========================================

Introduction

------------

This repository contains my work for the course project for the Coursera course "Getting and Cleaning data", part of the Data Science specialization.

What follows first are my notes on the original data.

About the raw data

------------------

The features (561 of them) are unlabeled and can be found in the x\_test.txt.

The activity labels are in the y\_test.txt file.

The test subjects are in the subject\_test.txt file.

The same holds for the training set.

About the script and the tidy dataset

-------------------------------------

I created a script called run\_analysis.R which will merge the test and training sets together.

Prerequisites for this script:

1. the UCI HAR Dataset must be extracted and..

2. the UCI HAR Dataset must be availble in a directory called "UCI HAR Dataset"

After merging testing and training, labels are added and only columns that have to do with mean and standard deviation are kept.

Lastly, the script will create a tidy data set containing the means of all the columns per test subject and per activity.

This tidy dataset will be written to a tab-delimited file called tidy.txt, which can also be found in this repository.

About the Code Book

-------------------

The CodeBook.md file explains the transformations performed and the resulting data and variables.

# TODO and LOOKUPS

* Heritage Health Prize
* Knit to HTML
* <https://www.springboard.com/blog/free-public-data-sets-data-science-project/>
* Books.google.com/ngrams
* [www.census.gov/census2010](http://www.census.gov/census2010)
* <http://www.sdss.org/>
* DarwinTunes – audio files
* <http://www.data.gov>
* <http://www.data.gov/safety/results-second-annual-safety-datapalooza/>
* UCI Machine Learning Repository: <http://archive.ics.uci.edu/ml/datasets/Flags>
* Data sets that come with R: > library(datasets) > data(iris)
* <http://www.netlib.org/lapack>
* <http://adv-r.had.co.nz/Profiling.html#profiling>
* <http://www.r-tutor.com/content/r-tutorial-ebook>
* United States Department of Agriculture's PLANTS Database: <http://plants.usda.gov/adv_search.html>
* There is a school of thought that this approach is backwards, that we should teach ggplot2 first. See <http://varianceexplained.org/r/teach_ggplot2_to_beginners/>
* <https://data.baltimorecity.gov>
* <http://www.stat.berkeley.edu/~statcur/Workshop2/Presentations/XML.pdf>
* avatar update; gravatar update
* JSON refs:
  + <http://www.json.org>
  + Good tutorial: <http://www.r-bloggers.com/new-package-jsonlite-a-smarter-json-encoderdecoder/>
  + jsonlite vignette
* <http://vita.had.co.nz/papers/tidy-data.pdf>
* <http://research.collegeboard.org/programs/sat/data/archived/cb-seniors-2013>’
* http://en.wikipedia.org/wiki/List\_of\_tz\_database\_time\_zones
* 2011 Journal of Statistical Software paper titled 'Dates and Times Made Easy with lubridate'.
* Original data sources:
* <http://data.worldbank.org/data-catalog/GDP-ranking-table>
* <http://data.worldbank.org/data-catalog/ed-stats>

Open Government Sites

* United Nations: <http://data.un.org/>
* U.S. <http://www.data.gov/>
  + List of cities and states with open data
* United Kingdom: <http://data.gov.uk/>
* France: <http://www.data.gouv.fr/>
* Ghana: <http://data.gov.gh/>
* Australia: <http://data.gov.au/>
* Germany: <https://www.govdata.de/>
* Hong Kong: <http://www.gov.hk/en/theme/psi/datasets/>
* Japan: <http://www.data.go.jp/>
* Many More: <http://www.data.gov/opendatasites>

Gapminder: <http://www.gapminder.org> human health data

Survey from U.S. :<http://www.asdfree.com/> help with access and analysis in R

Data Marketplace: <http://www.infochimps.com/marketplace>

Kaggle: <http://www.kaggle.com/> Data science competitions

Collections by data scientists:

* Hilary Mason: <http://bitly.com/bundles/hmason/1>
* Peter Skomoroch: <https://delicious.com/pskomoroch/dataset>
* Jeff Hammerbacher: <http://www.quora.com/Jeff-Hammerbacher/Introduction-to-Data-Science-Data-Sets>
* Gregory Piatetsky-Shapiro: <http://www.kdnuggets.com/gps.html>
* <http://blog.mortardata.com/post/67652898761/6-dataset-lists-curated-by-data-scientists>

Human activity using smartphone data sets

<http://archive.ics.uci.edu/ml/datasets/Human+Activity+Recognition+Using+Smartphones>

Specialized Collections:

* Stanford Large Network Data
* UCI Machine Learning
* KDD Nugets Datasets
* CMU Statlib # famous canonical data sets
* Gene expression omnibus
* ArXiv Data
* Public Data Sets on Amazon Web Services

API’s with R interfaces

* twitter (and twitter package)
* figshare and rfigshare
* PLoS and rplos
* rOpenSci
* Facebook and RFacebook
* Google maps and RGoogleMaps

You can use the quantmod (<http://www.quantmod.com/>) package to get historical stock prices for publicly traded companies on the NASDAQ and NYSE. Use the following code to download data on Amazon's stock price and get the times the data was sampled.

library(quantmod)

amzn = getSymbols("AMZN",auto.assign=FALSE)

sampleTimes = index(amzn)

Resources for graphing in R

* R Graph Gallery
* R Bloggers

# References

*The Elements of Data Analytic Style* [https://leanpub.com/datastyle/](https://eventing.coursera.) [other R, and data science books in leanpub]

R-Studio: <http://www.rstudio.com/>

R reference card: <http://cran.r-project.org/doc/contrib/Short-refcard.pdf>

The Comprehensive R Archive Network: <https://cran.r-project.org/>

The R Journal: <https://journal.r-project.org/>

RSeek: <http://www.rseek.org> - custom front-end to Google to find R help.

GitHub:

Tidy Data: <http://vita.had.co.nz/papers/tidy-data.pdf>

Lecture Notes for more subsetting:

<http://www.biostat.jhsph.edu/~ajaffe/lec_winterR/Lecture%202.pdf>