**Coursera – Data Analytics Cert Specialization (John Hopkins)**

Course 1 – Data Scientist Toolbox

**Week 1**

**ELOs**

1. Introduction to the key ideas behind working with data in a scientific way that will produce new and reproducible insight
2. Introduction to the tools that will allow you to execute on a data analytic strategy, from raw data in a database to a completed report with interactive graphics
3. Hands on practice so you can learn the techniques for yourself

**Tools**

Git -

GitHub – Collaborative tool for code sharing and editing

R –

Rstudio –

**Week 2: Installing the Toolbox**

**Intro to the Command Line Interface (CLI)**

* Working for files and folders typing in commands as opposed to using the mouse
* For Windows we will be using Git Bash, Mac/Linux is the terminal
* Used for
  + Navigate folders
  + Create and edit files, folders, and programs
  + Run computer programs – main use in data science
* Basics of Directory (Folder)
  + Organized like a tree
  + Directories can be inside of another directory
  + Can be navigated using CLI
  + Root (/) directory is at the top of the tree
  + Your home (~) directory is where you log in, where most of your personal stuff is
    - Your ~ is a sub directory of / or your / is up from you ~
  + Find git bash by looking it up on your start menu
  + Prompt – User name
  + CLI opens on your working directory
  + To find your path type PWD (Print Working Directory) this will show you the path to your working directory
* CLI Commands (Commands > Flags > Arguments)
  + Commands – Such as PWD
  + Flags – Parameters passed to the command to tell it what kind of behavior to take, preceded by a - (-a, -l, -al)
  + Arguments – What the command will modify, like a file
  + Depending on the command you may or may not need flags and arguments

**List of commands**

* **pwd –** print working directory
* **clear –** will clean the CLI
* **ls –** list files and folders in the current (working) directory
* **la -a –** will list all files and folders including hidden ones, this hidden files and folders will be preceded by .
* **ls -al** – Will list details of the hidden folders
* **cd –** changed directory takes you to the home directory, CD and an argument takes you where you want to visit, CD .. takes you one level above your current directory
* **mkdir –** make directory follow by an argument (folder name), same as right click > create new folder
* **touch –** creates a file follow by an argument (file name)
* **cp –** copy, follow by an argument of what to copy and an argument of where to copy to
* **cp -r –** to copy an entire directory into another directory
* **rm –** remove or delete files
* **rm -r –** removes entire directory, no undo be careful
* **mv –** used to move files from one directory to another, can also be used to rename files use command then give it a file to rename and then a new name for that file
* **echo –** will print out whatever you tell it to
* **date –** prints out the date

**GIT Vision Control System**

Records all the changes that you make to a file or set of files overtime. Used to keep track of everything being done to these files and to manage intermediate files in between drafts and final version.

**GitHub**

Web based hosting service for software development people to collaborate on projects on a bigger scale, uses git revision control as its driving force. You can contribute to projects and other people can contribute on your projects, by pulling and pushing to remote repositories on the web. The repositories you have on GitHub are backed up on the server in case something happens to your copy. A strength of GitHub is its social aspect, which allows people to share, contribute, and follow one another.

**GitHub Repository**

Two ways of creating a repo; crate it form scratch or “fork” another user’s repo. To crate a repo from scratch:

* Go to your GitHub account
* “Cerate a new repo”, give it a name and description. Note: make it googleable so that it can be searched by others
* Select either public or private
* Check the “Initialize this repository with a README”
* Click “create repo” at the bottom

To create a copy of your newly created GitHub repo to Git you need to:

* Open GitBash
* Create a directory on your computer where to store it
  + $ mkdir ~/”directory name”
* Navigate to your ne directory
  + $ cd ~/”directory name”
* Initialize a local Git repository in this directory
  + $ git init
* Link up your local repository to your GitHub repository
  + $ git remote add origin https:/github.com/username/reponame.git

To “fork” a repo from another user, this is mostly done to develop software with other people;

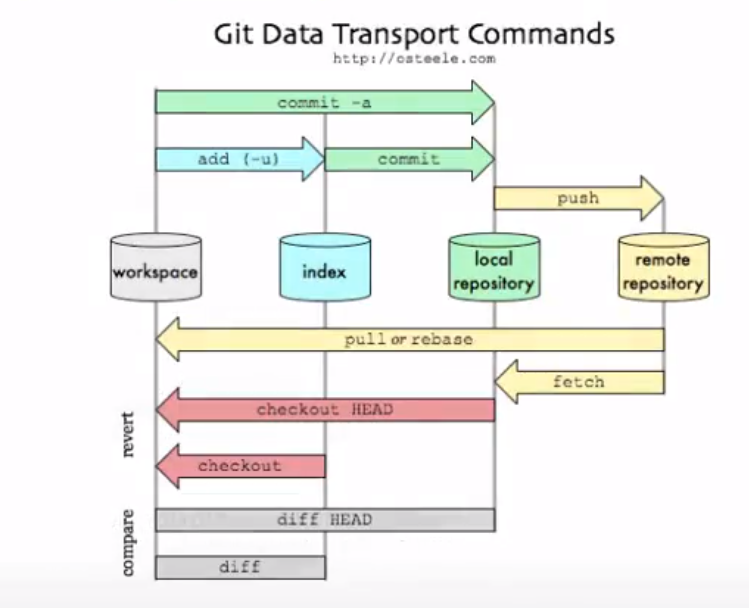
* Find a repo that you are interested in and click the “fork” button up top
* This will create a copy to your GitHub, which you can clone to Git on your machine
  + $ git clone https:/github.com/username/reponame.git

**Pushing and Pulling from GitHub**

**Terms**

* **Workspace** **–** The directory where files are being worked on (e.g Eclipse Jee Neon)
* **Index –** Tells Git what are the files it should be controlling under version control
* **Local Repo –** These are the files that are stored or version controlled on your local computer
* **Remote Repo –** GitHub in our case

You start by crating a new file on your workspace > add it to you index, so that git knows to monitor that particular file and track all of its changes (-u) > commit that file to your local repository so that it can be stored and update it > after you have made a few commits you’ll want to issue a “push” command to your remote repository



**Adding**

If you are working a directory that is a repository under version controlled by Git, you need to add them to the index:

* **git add .** will add all new files to the directory where you are working
* **git add -u**  update tracking for files that have changed names or were deleted
* **git add -A** does both of the previous commands
* This needs to be done before committing to your local repository

**Committing**

Once the files have been added to the index you can commit them to your local repository by using the **git commit -m “message”** command. The message is a useful description of what was done to the file, this only update the local repository not GitHub.

**Push**

Using the command **git push** will push your changes to the remote directory.

**Branches**

If in a project where there is a version used by other people and you might want to create a branch so that you can make changes independently. Use the command **git checkout -b branchname**, this will crate a new branch. To see what branch you are working on use the command **git branch**, to switch back to the master branch use the command **git checkout master** (master is the default name for the branch for all the repositories created with GitHub).

**Pull Request**

Once you have made a push of your changes up to your repository on GitHub and if you are working on a different branch (independent) or on a forked repository, then you might need to merge your changes to the original/forked repository. To do this you’ll need to, issue a pull request on GitHub > go to the branch you are interested in pulling > click on “compare and pull request” > this will issue a pull request to the individual who owns the branch.

**R Packages**

To obtain R packages go to CRAN, for some biological and big data packages go to Bioconductor Project. To obtain information on CRAN using the **a <- available.packages()** argument, this will probably return a search of maybe thousands. The command **head (rownames (a), #)** this would return a set number you give it (#), in alphabetical order. To narrow down your search you can and you know the topic of the package you are looking for go to the **Task Views** link on CRAN this groups together R packages related to a given topic/area.

**Installing R Packages**

Use the command **install.packages(“packagename”)** function in R. To install multiple packages at once use **install.packages(c(“packagename”,”packagename”,”packagename”))**. To install packages into RStudio go to tools > install packages > pick the repository > pick the package you want to install > install.

**Installing from Bioconductor**

Give R the **source(“**[**http://bioconductor.org/biocLite.R**](http://bioconductor.org/biocLite.R)**”)** command, this will load the biocLite function. Then type the biocLite**()** command, this will install the basic version of Bioconductor. To install further R packages use the same commands as with packages from CRAN but substitute the **install.packages** with the **biocLite** command.

**Loading R Packages**

Even after installing the packages they are not yet available, you need to command R to load them using the following command **library(packagename)**. To see all the function associated with a package you just installed, use the following commands **library(packagename)** then **search()**.

**Installing R Tools**

Necessary for building R packages in Windows, go to cran.r-projec.org/bin/wondows/Rtools/ , then install latest version. Select the .exe link corresponding to your version of R. Once the Rtools is installed, then proceed to install devtools from RStudio. If you are not sure if you have already installed it use the **find.packages(“devtools”)** command in the console. If you do not have it, to install it use the **install.packages(“devtools”)** command.

**Verifying Intallation**

Load the packages using the **library(devtools)** command, then type the **find\_rtools()**, this should return **TRUE** printed out in the console and this verifies proper installation.

**Week 3: Conceptual Issues**

**Types of Questions**

In approximate order of difficulty;

* **Descriptive Analysis** 
  + **Goal:** To describe a set of data
    - First type of analysis performed
    - Commonly applied to census data
    - First describe the data and then interpret what you’ve seen
    - Description can’t be generalized without further statistical modeling
    - **Example:** US Census describes the US population, but do not predict anything about other people of other countries
* **Exploratory Analysis**
  + **Goal:** Look at data and find new relationships and connections, but not necessarily confirm them.
    - Good for defining future data projects
    - Not the final saying
    - Should not be use to generalize or predict when a standalone analysis
    - Correlation does not imply causation
    - **Example:** Looking at how the brain reacts to different stimulus and finding relationships
* **Inferential Analysis** 
  + **Goal:** Use a relatively small amount of data and extrapolate the results to a larger population.
    - Most common goal of statistical modeling
    - Involves estimating the quantity you care about and the uncertainty over said estimate
    - Inference depends on the population you are looking at and the sampling scheme
    - **Example:** The effect of air pollution control on life expectancy in the US by analysis a sub-set of the US population and extrapolate the results to make an inference between the air pollution and life expectancy
* **Predictive Analysis** 
  + **Goal:** To use the data collected on some objects to predict the values of another object
    - Even if *X* predicts *Y* it doesn’t mean that *X* causes *Y*
    - Accurate prediction depends on measuring the right variables
    - More data and a simple model tend to work really well
    - Prediction is very hard, especially about the future
    - **Example:** Look into Nate Silver’s (538 forecast) prediction on US elections based on previous polling data
* **Causal Analysis** 
  + **Goal:** What happens to one variable when you change the value of another variable
    - Usually requires the use of randomize controlled studies to identify causation
    - There are other methods, but these are susceptible to assumptions
    - Identified as average effects: “On average if we do this, then on average this will happen”
    - Usually the “Gold Standard” for data analysis
    - **Example:** Randomized medical studies to analyze what happens when a medicine/procedure is applied to a patient
* **Mechanistic Analysis** 
  + **Goal:** Understand the exact changes in variables that lead to changes in other variables for individual objects
    - Rarely covered in data science
    - Incredibly hard to infer, except in very simple situations
    - Usually modeled by a deterministic set of equations
    - Mostly used in the physical/engineering science
    - Generally, the only random component is measurement error
    - **Example:** Difference in changes that can be made to pavement design and what differences would this lead to in the functioning of this pavement

**What is Data**

“Data are values of qualitative or quantitative variables, belonging to a set of items”

**Set of items** – is sometimes called the population in statistical inference, what we are trying to discover something about.

**Variables** – Are measurements of characteristics of an item, qualitative or quantitative.

**Qualitative** – Not necessarily order, such as sex or country of origin.

**Quantitative** – Usually measured on a continuous scale such as weight or height

**What data Look like** – Most starts in raw form, such as fast queue files, API, medical records, video, audio files, access through open data websites (data.gov).

Data is the second most important factor in data science, the most important is the question. Often the data will limit or enable the questions you are trying to ask. When limiting, you will need to modify your question to try to answer a sub-question or related question.

**Experimental Design**

**Pre**

* First thing to keep in mind is the analysis plan, to foresee key issues in the study design that can become a problem.
* Formulate a plan for data a code sharing.

**Key Ideas**

Formulate your question in advance

* A scientific discipline, such as data analysis, requires that you answer a specific question when using data

**Statistical Inference**

* Use a **probability argument** to choose a **sample** of the **population** then collect **descriptive statistics** statement and use **inferential statistics** to decide if the results collected from the **sample** apply to the larger **population**

**Confounding –** I you measure 2 variables and find a correlation and might be inclined to think that one causes the other, however there might be a 3rd variable, the confounder, for the relationship between the 2 variables. Pay attention to other variables that might be the ones causing the correlation, sometimes called spurious correlation. **Correlation is not causation**.

* **Fixes**
  + If able to, make one of the variables fixed
  + Stratify a variable
  + Randomize it to assign people to groups

**Prediction Study –** Collect information on observations from subjects in one group and from subject in another group. Then build a predictive function (f(x)) so that one a new subject appears we can predict in which of the two groups it would fall under. Same issues with probability, sampling, and confounding need to be taken into consideration.

**Prediction Vs. Inference**

Prediction is slightly more challenging than inference, you need the distribution between populations to be relatively more separated, their means need to be far part enough that one can distinguish where an observation came from.

**Prediction Key Quantities (Condition (+ or -) and Test (+ or -))**

* True positive (++)
* False positive (+-)
* False negative(-+)
* True negative (--)
* Sensitivity
* Specificity
* Positive predictive value
* Negative predictive value
* Accuracy

**Data Dredging**

The use of data mining to uncover patterns in data that can presented as statistically significant

Course 1 – R Programming

**Week 1: Getting Started with R Nuts and Bolts**

**Setting Working Directories and Editing R Code Files**

Figure out which working directory are you on by using the ***getwd()*** Function. This is important because when you read data or write thing out these will be done in your working directory. If you try to red a file which is not in the working directory you will get an error, you can either change the file to your working directory or change your working directory to where the file is.

In order to write R code, you will need to use a text editor, got to File>New Script, to load up R’s rudimentary text editor. To lead the code into the console, you can either do a copy paste from the R editor to the console, this works for simple functions. Or go to File>Save As>save to your working directory as a .r file on the R editor window. Then use the ***dir*** function in the console window you’ll see your file with the function. Then load it into the console using the ***source (filename.r)*** function to load all the code that is the file.

**Overview and History of R**

R is a dialect of S, which was initiated as an internal statistical analysis environment – originally implemented as Fortran libraries, to implement repetitive routines (1976). In 1988 the system was rewritten in C (V3) language to make it more portable across systems. V4 was also release in 1988 which we use today, R is an implementation of the S language.

R was created in 1991 by Ross Ihaka and Robert Gentleman and licensed in 1995 under the GNU General Public License to make it a free software (Martin Machler). In 1996 the public mailing list was formed (R-help and R-devel). In 1997 the R Core Group was formed which has control over the source code for R.

**Features –** R’s syntax is very similar to S which made the transition easy, not so relevant now. Semantics are superficially similar. R runs on most OS and frequent releases including bug fixes. The core software is lean with fundamental functions and you can add functionalities as you need them. Its graphics capabilities are very sophisticated, useful for interactive work, but contains a powerful programming language for dev new tools which helps to ease the transition from the programmer to the user.

**Free Software –** The program can be run for any purpose (0), the freedom to study the how it works and adapt it to your needs (access to the source code is a precondition) (1), freedom to redistribute copies (2), freedom to improve and release these to the public (3).

**Drawbacks –** Essentially based on 40 yr old technology, little support for dynamic or 3-D graphics (improving), functionality is based on consumer demand and contribution, objects worked on must be stored in the physical memory of the machine.

**Design of the R System –** Divided into two conceptual parts, the base system you get from CRAN and everything else. Some of the base packages include utils, stats, graphics, splines, tcltk, grDevices, and others. There are about 4000 packages available on CRAN developed by users and programmers. There are also packages with the Bioconductor project for genomics data analysis.

**Getting Help**

Search on forum archives for similar questions, manuals, CRAN FAQ. Think about the following when asking questions in the forum; what steps will reproduce the problem? What is the expected output? What do you see instead? What version are you using? OS? Header: “Version. Function not working OS – problem”.

**Console Input and Evaluation**

**Entering Input –** Expressions are typed into the R prompt. <- symbol is the assignment operator use to assign a value to a symbol > x <- 1 or > msg <- “hello”. You can also use the ## symbols to add comments or notes to yourself as the R engine will ignore anything to the right of the # > msg <- “hello” ## introduction will > print (msg) or msg = hello