==========
==========

Welcome to Software Carpentry Etherpad for the November 5-6th. workshop at the Nature Conservancy!

This pad is synchronized as you type, so that everyone viewing this page sees the same text. This allows you to collaborate seamlessly on documents.

Use of this service is restricted to members of The Carpentries community; this is not for general purpose use (for that, try https://etherpad.wikimedia.org/).

Users are expected to follow our code of conduct: https://docs.carpentries.org/topic_folders/policies/code-of-conduct.html

We will use this Etherpad during the workshop for chatting, asking questions, taking notes collaboratively, and sharing URLs or bits of code.

Website: http://tiny.cc/tnc-swc - (https://mickley.github.io/2019-11-05-nature_conservancy/)

Socrative Login: https://b.socrative.com/login/student/
Room: TNCSWC

==========
==========

Instructors:

- Jonathan Borrelli (Rennselaer Polytechnic Institute, borrejj@gmail.com)
- James Mickley (University of Connecticut, james.mickley@uconn.edu)

Helpers:

- Michael Treglia
- Adam Starke

Attendees: (Put your name here):

- Dave Richardson
- Matthew Grasso
- Paul Gallery
- Shannon Thol
- Christina Thorbourne
- Zachary Simek
- Chris Zimmerman
- Kristin France
- Adam Starke
- Mike Treglia
- Becky Shirer
- Marcela Maldonado
- Richard An
- Colleen Lutz
- Stephen Lloyd
- Cathy Gibson

==========
==========

Setup:

1. Sign in up front: Get a name tag and a sticky note of each color
2. Download and install software from our course website: **https://mickley.github.io/2019-11-05-nature_conservancy/**
3. Put your name under Attendees above (You can get here from the Etherpad link on the course website)
4. Fill out the pre-workshop survey if you haven't already: **https://www.surveymonkey.com/r/swc_pre_workshop_v1?workshop_id=2019-11-05-nature_conservancy**
5. **Enter your name at the top of the chat column of this etherpad (top right corner)**
6. Download the datasets here https://www.dropbox.com/sh/huimlzu3ezomugt/AABZu3wxkhkmqKEDHF7b2QFla?dl=0 and save them to your desktop
7. In a web browser sign in to your GitHub.com account. Sign up for one if you don't have an account.
8. In RStudio, click on Tools > Global Options... > Git/SVN > Make sure the grey box under "SSH RSA Key" shows a file name and a "View public key" link exists next to it and clicking on "View public key" opens a window opens which is not blank. If the "SSH RSA Key" box is blank, click "Create RSA key..." and leave the optional Passphrase blank and click "Create".

Please put up a blue sticky on your laptop if you have signed up for Github and have all three programs installed (Github, R and Rstudio)

==========

Link to lessons: http://swcarpentry.github.io/r-novice-gapminder/

DAY 1 - Introduction to R and RSTUDIO

RStudio

- Interface for R that makes your life easier
- You can type code into the console in RStudio, but much more reproducible to save everything in an R script
    - File > New File > R Script
- R Scripts
    - Run code by clicking the run button at the top of the R script box
    - Or, you can use CTRL + Enter (CMD + Enter for macs)
    - You can select multiple lines of code to run
    - Or, you can run the entire .R file using the source button, or source() function
    - If you enter something incomplete, R will wait for you to finish-waiting operation will show instead of regular command prompt
        - Example: "1 + "
        - Press escape to get out of it and back to a ">"
        - Or, finish the incomplete expression
        - You would know if there is an incomplete command if you see a +1==1 on the console instead of a >

R Coding

- Basic Math - R knows mathematical order of operations, and observes parentheses.
    - Has lots of built in mathematical functions
        - sin(), log(), log10(), mean(), sd(), sum(), etc.
    - scientific notatoin is used (e.g., 2e-4 = 0.0002)
    - log() function defaults to NATURAL log rather than base 10.
    - you can change the default # of decimals that it shows if desired
- Typing in a function in RStudio brings up an autocomplete menu with a little help
- Comparing things in R:
    - see if things are equal: 1 == 1 (very important to have 2 equals signs to compare)
    - not equal to: 4 != 2
    - more comparisons: >, <, >=, <=
    - Comparisons can be tricky with non-integers due to precision:
        - e.g., 1.000000000000001 == 1.00000000000000 or 1/3 == 0.33333
        - use all.equal() instead
- Storing things in R
    - use "<-" to save things to a variable that we name. equal sign works but not recommended, can cause confusion
    - x <- 1/40
    - You can also use "=", and some people do. We recommend NOT using "=", can cause problems/confusion
- all.equal(x,x) for decimals is best (floating point precision issues [on any computer])
- 1:5 R knows to make a sequence 1 2 3 4 5
- typing a fcn name without hte parentheses and running it will show what is under the hood
- R will think a capital T is TRUE and F is FALSE, that can lead to problems if you forget

Good practice is to use meaningful variables. Some common conventions for multiple words are:

- "camel case" capitalize beginning of words in middle of variable name...
- periods or underscores between words (e.g., my.variable, my_variable)
- .mass does actually define variable but hidden, won't show in environment
- can use numbers in middle but not beginning of variable name
- generally don't start variable names with a symbol
- also try to avoid super long variable names if you are going to have long stretches of code

Cutting and pasting code/data

- You can easily cut and paste code from another document into R
- Data Pasta
    - Tool for cutting and pasting html table data into R
    - https://cran.r-project.org/web/packages/datapasta/vignettes/how-to-datapasta.html
    - https://cran.r-project.org/web/packages/datapasta/index.html
    - https://github.com/MilesMcBain/datapasta

Vectorization

- R works a lot on whole lists of numbers called "vectors"
- Create a vector: x <- 1:5

    - Now x has 1,2,3,4,5 in it
    - We can do x^2 and square all the numbers at once, great for working with data

Importance of maintaining environment

- ls() function, leave empty, will print out variable names from current environment
- ls(all.names = TRUE)
    - This shows hidden variables
    - all.names is an "argument", something we're telling the fuction. Here, we use the "=" sign
- rm(x) removes the x variable

Packages

- R has many functions baked in, but many more are in 3rd party "packages"
- installed.packages() or the Packages tab on RStudio to see what you have installed
- load a package using library(packagename), e.g., library(ggplot2)
- To install packages
    - install.packages("ggplot2") <- note that we have to use quotes here
    - to tell R which package version of a fcn to use, use dplyr::select(). otherwise it will use whatever has most lately been used and you might not be aware
    - You can set the default folders for libraries install in the Tools>Global Options menu
- best practice to load packages at the top of your code, better for consistency and clarity. not recommended to load them with the checkox in the package window.

Projects in RStudio

- "Projects" packages up some work in RStudio, makes it easy to move around your computer or collaborate with someone
    - To make a project, go to the Project dropdown in the top right corner of your screen
    - Each project has it's own workspace/environment
    - wherever you create it, even if you move it later, it will stay intact and work.
    - best practice recommendation. don't modify data outside of the computing environment. treat data as read only. one exception might want to clean data in R, keep raw data intact, and create a new clean data file.
    - treat any generated output as disposable because if you are using scripts, you can just remake it....
    - folder structure ideas: data, scripts, doc, results
        - James has a skeleton "project template" that he uses for every project, consistency is useful

Getting help:

- help(function_name) OR ?function_name
    - Or, go to the help tab in RStudio, and use the search box
    - The ?function_name only works for packages that have been loaded with library()
        - To search all packages, even unloaded ones, use ??function_name
            - This is a *fuzzy* search, you don't have to have the whole name of the function
                - e.g., ??mean to search for all functions with "mean" in them.
    - Also, *some* packages have vignettes: nice writeups of how to use the package itself

        - vignette(package = "ggplot2")
        - Or, click on the package in the packages tab of RStudio, and click on the vignettes link at the top
        - Lots of TNC-relevant examples - e.g., here's one for Species distribution models in the dismo package - https://cran.r-project.org/web/packages/dismo/vignettes/sdm.pdf
    - CRAN task views
        - https://cran.r-project.org/web/views/
    - Twitter #rstats to get help
    - StackOverflow has a lot of good help too: https://stackoverflow.com/
        - Need a reproducible example: your code, and a bit of example data
            - Another means of sharing reprex (reproducible code for getting help) -https://github.com/tidyverse/reprex
        - sessionInfo() gives all the info on your R and package setup
- All R help pages have the same/similar sections
    - A description
    - Usage notes
    - List of possible arguments (a helper for finding arguments can be accessed with the tab-auto complete when the cursor is inside the function)- within Rstudio
    - Value: whatever the function gives back to you
    - See also: related functions
    - Examples: example code

Getting data into and out of R:

- read.csv(file = "path/filename") - for comma separated values
    - CSV files are most commonly used
    - You could also do tab-separated:
        - read.table() or read.delim() for tab-separated
    - Or, Excel files with the readxl package
        - https://readxl.tidyverse.org/
        - xlsx_example <- readxl_example("datasets.xlsx")
        - read_excel(xlsx_example)
- write.csv(yourRdata, file = "path/filename", row.names = FALSE)

Exploring data in R

- To just get one column of a dataset, use the $
- e.g., cats$coat gets the coat column from the cats dataset
    - This gives you back a vector, and you can do things with that vector
        - cats$weight - 1
        - paste("My cat is", cats$coat)
            - Paste combines two or more bits of text, separated by a space
- Figure out what type of data you have in a variable
    - For dataframes, use str()
        - Shows what's in each column, and the number of rows and columns
        - Each column is its own vector, dataframes are just a bunch of vectors put together, each with a column name
    - use the typeof() function:
        - typeof(3.14) # Double, a floating point number
        - typeof("banana") # Character
        - typeof(TRUE) # Logical
        - typeof(1L) # Integer

        - All things inside of a vector have to be the same type of data. If not, R will make them the same (coerce)
            - Precedence: character > complex numbers > double > integers > logical
    - Factors
        - One way that R represents character/categorical data
            - Stores each "level" or category as a different integer, much more efficient this way computationally
            - e.g.
                - banana = 1
                - apple = 2
                - orange = 3
            - But, if we try to add grapefruit, there's no number/level already there, and R will scream
            - Will order them alphabetically or increasing order--you likely want to make sure control is the first value so R will compare things to it, so make it the first, and you will need to specify to make it so
    - Lists
        - Lists can have things of different types inside of them, unlike vectors
        - Things inside lists can have a name, or not, in which case we refer to them by the position in the list (e.g., 1 for 1st thing)
- More ways to explore variables
    - head() and tail() return the first 6 values/rows
    - names() gives us the column names
    - length() gives us the length of a vector, dataframe, etc.
- Subsetting data
    - cats[1, 1] # First row, first column of a dataframe
    - cats[, 1] # Entire first column
    - cats[1,] # Entire first row
    - cats["coat"] # The column named "coat"
    - cats[[1]] # The first element in cats (usually used with lists more so than dataframes)
- Matrices
    - have rows and columns too, but operate differently from dataframes
    - matrix(0, ncol = 6, nrow = 3)
        - This creates a 6 column x 3 row matrix and fills it with zeros
        - By default, R fills a matrix by column
            - matrix(c(1,2,3,4), ncol = 2, nrow = 2)
    - nrow() to ask how many rows
    - ncol () to ask how many columns
        - These functions also work for dataframes

==========

DAY 1 - Programming in R

TODO:

- Install packages: knitr, formatR, rmarkdown, cowplot, tidyr (double check that they installed)
    - install.packages(c("knitr", "formatR", "rmarkdown", "cowplot", "tidyr"))
- Login to Socrative: https://b.socrative.com/login/student/ - Room = **TNCSWC**

- Follow along with my R script on dropbox (in scripts folder): https://www.dropbox.com/sh/cbzxs8q4l35sh8r/AABmjQ2g57noOA0lKFU7F24Ba?dl=0
- Make comments in your code using #
    - everything after the # is ignored by R
    - these can be on their own line or be inline

In addition to logical comparisons (==, !=, >, < , >=, <=) we can use logical operators representing Or and And;

- For And we use "&"
- For Or we use "|"

A for loop iterates over a vector and performs some task for each value of the vector

Note: A tibble is just a dataframe that has some special properties. If you print a tibble it will only show the first 10 rows, and only as many columns as will fit nicely on your screen

The code for a function follows:

```
function_name <- function(argument1, argument2, ... ){
 stuff you want to do
 return(output)
 }
```

Functions should always end with a return statement

variables that exist within functions are ONLY within functions, outside of the function they don't exist. the function is also working with a copy of the variable/data.

-

Graphing with ggplot

- draws the things at end of code on top
- ggplot does have a preset color palette. you can change that to some other palette. they will load some more info. you can google R colors and it will show you what they correspond to. you can also enter the HTML name/code for colors rather than the ggplot color names
- Adam has created a package that has the TNC branded colors. ADAM CAN YOU POST THAT?
- National Parks color palettes: https://github.com/katiejolly/nationalparkcolors
- Wes Anderson palettes: https://github.com/karthik/wesanderson

If you want to use a mathematical expression in a ggplot axis label you can use the expression function

-
    - for example: labs(x = expression(sqrt(x^3))) added to a ggplot will make the x axis label a square root symbol with x raised to the 3 (as a superscript)

ggplot help:

- ggplot flipbook: https://evamaerey.github.io/ggplot_flipbook/ggplot_flipbook_xaringan.html#1

-
    - goes through a bunch of plots layer by layer building up to the final plot
    - lets you see how the different ggplot pieces come together
- R Graph catalog: **http://shiny.stat.ubc.ca/r-graph-catalog/**
- GGPlot2 online help: **http://docs.ggplot2.org/**
- R Graph Cookbook: **http://www.cookbook-r.com/Graphs/**
- ggplot2 essentials: **http://www.sthda.com/english/wiki/ggplot2-essentials**
- Rstudio cheatsheets: **https://www.rstudio.com/resources/cheatsheets/**
- Cowplot: **https://cran.r-project.org/web/packages/cowplot/vignettes/introduction.html**

Some examples of useful things I (Mike T) have put together for sharing with folks:
* https://tnc-ny-science.github.io/nyc_trees/ownership_exploration_summary_20190807/canopy_ownership_summary_20190807.html
* https://mltconsecol.github.io/misc/commdists_quickAnalytics_interactive_minimal.nb.html

Reports:

- create a new under the File tab

check out Help/Markdown quick reference
check out rmarkdown.rstudio.com for lots of galleries etc
difference between r notebook (rmd) format vs html. the html export won't let people you share it with run the code chunks right from in the html
... R first runs the code chunks and makes them into markdown

- then there is a file in pure markdown
    - uses tool pandoc behind the scenes that converts it to something else as specified, like HTML. under preview-Knit to word, etc. Knit to pdf doesn't work very well FYI.
        - Well, it works, but you need to install a LaTex installation to be able to use it, just another hoop to jump through
            - For macs, MacTex or BasicTex: http://www.tug.org/mactex/morepackages.html
            - For PC, MikTex: https://miktex.org/

R Notebooks use a different environment than the one in main RStudio. Have to define variables and fcns and data etc in the notebook and run in order for subsequent commands to work-you can't just call directly to things you've defined in your studio session

-

==========

DAY 2 - Data Wrangling

TODO:

- Sign up for a Github account if you haven't already: https://github.com
- Login to Socrative: https://b.socrative.com/login/student/ - Room = **TNCSWC**
- Follow along with my R script on dropbox (in scripts folder): https://www.dropbox.com/sh/cbzxs8q4l35sh8r/AABmjQ2g57noOA0lKFU7F24Ba?dl=0
    - Day2_morning.R

- Make sure you have the gapminder_wide.csv file in your Rstudio project's data folder
    - download here: https://www.dropbox.com/sh/huimlzu3ezomugt/AABZu3wxkhkmqKEDHF7b2QFla?dl=0

select(.data, ...) allows you to choose what columns you want to work with in your dataframe

-
    - add column names as arguments with no quotes
    - you can also select a range of column names e.g., column1:column4 will give you column1, column2, column3, and column4
    - select can also remove columns using the syntax -column1

Note: a tibble will show you first 10 rows and only as many columns as fit nicely in data frame, will tell you it is a tibble at the top

filter(.data, ...) allows you to get rows from the dataframe that match a given condition or set of conditions

-
    - can use mulitple conditions by separating with a comma e.g., filter(continent == "Africa", year == 2007)

%>% is called a pipe

-
    - takes the result of the left hand side and uses it as the first argument of the right hand side

1. The traditional way with nested functions
go_to_work(get_breakfast(wake_up()))
2. Using the pipe
wake_up() %>% get_breakfast() %>% go_to_work()

Note that if you are going to comment out lines for special cases or particular runs, or just to explore if something is working, consider adding I() at the end of the code. it just returns the final identiy so that the code won't look for things that aren't there or get hung up waiting for another command from you.

group_by(.data, ...) tells R that you want to do operations to subsets of the data based on groups defined in the columns you give to the function

summarize(.data, ...) allows you to compute things on the groups

the n() function counts the number of samples in a group

mutate(.data, ...) allows you to alter the dataframe by changing a column or adding a new column

Wide format Ex:

-
    -
        -
            - weight length

genus
ursus

felis

LONG which R prefers

- Genus Meas Value
- ursus length
- ursus weight
- felis length
- felis weight

Note: An alternative to read.csv() is the read_csv() function from the readr package. The read_csv() function works the same as read.csv(), but imports the data as a dataframe and a tibble, and it uses the stringsAsFactors = FALSE as default

Gather doesn't remove any columns that aren't involved, but it does remove ones that are.

Cartoon spread/gather https://raw.githubusercontent.com/allisonhorst/stats-illustrations/master/rstats-artwork/tidyr_spread_gather.png
other fun rstats illustrations: https://github.com/allisonhorst/stats-illustrations/tree/master/rstats-artwork

Explainer for joining two dataframes together with dplyr (with animations): https://www.garrickadenbuie.com/project/tidyexplain/

Data resources

- Rstudio cheatsheets: **https://www.rstudio.com/resources/cheatsheets/**
- Package documentation: **https://rdrr.io/cran/dplyr/**, **https://rdrr.io/cran/tidyr/**
- R Cookbook: **http://www.cookbook-r.com/Manipulating_data/**
- Data Wrangling Webinar: **https://www.rstudio.com/resources/webinars/data-wrangling-with-r-and-rstudio/**

There is a community of r users who do #TidyTuesday on twitter. Each week a new data set is posted for people to plot
You can learn more here: https://github.com/rfordatascience/tidytuesday
Here is one very good r-user who does a screencast of how he works through the data https://www.youtube.com/watch?v=6GV9sAD6Pi0&feature=youtu.be

If you are on Slack there is an online learning community R for Data Science: https://medium.com/@kierisi/join-the-r-for-data-science-online-learning-community-842527222ab3

At TNC we have access to LinkedInLearning which has a decent number of R courses: https://www.linkedin.com/learning/topics/r?u=2186177

==========

DAY 2 - Version Control with Git

Working in the terminal (git bash)

- The $ is the prompt here, waiting for us to type something, like > in R
- **pwd** # print working directory, shows what folder you're in
- **ls** # list the contents in the current folder
    - **ls -a** # Shows all the files & folders, even hidden ones
- You can use the up arrow on your keyboard to get the last command

- **ls --help** # Get help for a command like ls
    - **man ls** # This might work if the other one doesn't on your computer
        - to get out of the manual page, press the "q" key
- **cd Desktop** # Change to the desktop folder
    - cd .. brings you back one, ../.. brings you back 2 folders
- **mkdir planets** # Make a new folder within your current folder
- **rm -rf moons/.git** # Remove a file or folder (in this case the .git folder in moons) Be very careful with this; permanent deletions.

Working with Git

- Configuring Git
    - **git config --global user.name "Your Name"**
    - **git config --global user.email "youremail@mail.com"**
- Check your git settings
    - **git config --list**
- Getting help on git functions
    - **git --help config**
        - or **git --config help**
- Create a repository (store all the changes we make to a folder)
    - git init
        - This makes a hidden folder called .git inside your project folder
            - Git stores all of its versioning stuff in here, you don't worry about it
            - don't make a git repository within a git repository
- Ask git about its status
    - **git status**
        - useful command to figure out where we are and what we need to do
        - We'll use this command a lot
- To edit/create a text file
    - **nano mars.txt**
        - To get out of nano:
            - CTRL + x
                - It will ask if you want to save the changes
                    - type "y"
                    - confirm the filename and press enter
- To print out the contents of a file
    - **cat mars.txt**
- To start tracking changes on a file (or put changes into a commit)
    - **git add mars.txt** # This starts tracking the mars.txt file
        - git add puts things into a temporary staging area so we can collect related changes
        - You could also add all the files in the repository with changes
            - **git add --a**
            - Or all of the files in a folder
                - **git add moons**
- To actually save a version of a file we're tracking
    - **git commit -m "Start notes on Mars as a base"**
        - git commit actually puts a set of changes into the repository
        - "commits" are like packages of changes to one or several files that are stored together
        - the -m lets you give git a note on what your version/change was about. want to do this for every commit

        - Good messages should be informative. "changed mars.txt" doesn't tell us much
- Representation of the git workflow
    - file --git add--> [staging area] --git commit--> [permanently saved in repository]
- Look back at the history tracked in git
    - **git log**
        - Lists all of the commits in reverse chronological order.
        - hit enter to keep running through when gets too big for screen, use Q to quit out of it
        - Limit the number of commits to show in the log
            - **git log -1**
                - latest change
        - Show less information
            - **git log --oneline**
- Check how a file has been changed
    - **git diff mars.txt**
        - Show the new changes in mars.txt compared to the last time it was committed
        - Note: you could also just use git diff to show changes made to all files
        - **git diff --staged**
            - This shows the difference betwen last committed change and what's in the staging area
    - You can diff between specific versions in the repository log
        - **git diff HEAD mars.txt** # Compare with the most recent version
        - **git diff HEAD~1 mars.txt** # Compare with the 2nd most recent version
- Show changes in an older commit
    - **git show HEAD~2 mars.txt** # Show the changes in mars.txt in the 2nd most recent version, along with the commit message
- To get rid of a change to a file and revert to the previous version
    - In case you messed something up
    - **git checkout HEAD mars.txt**
    - you have to go back to 1 before the committed change (KF thinks, may not have written this correctly). checkout HEAD will lose anything you have edited and not added or committed, but won't lose your latest commit.
- What if you get a detached head error? this can happen if you checkout HEAD without adding file name after it
    - Fix this with **git checkout master**
- Tell git to ignore certain files
    - Create a file named .gitignore: **nano .gitignore**
        - Inside the file put:
            - ***.dat** # Ignore anything that ends in .dat
            - **results/** #Ignore everything in the results folder
        - Note: you could customize this for whatever files you'd like
    - If you want to see the status of files that you ignored
        - **git status --ignored**
- If you get stuck in the vi editor
    - Press escape
    - type a colon (:)
    - type q!
    - Press enter

- Working with remotes, e.g. github
    - Add an empty remote repository to yours on your computer
        - **git remote add origin https://github.com/mickley/planets.git**
        - **git push -u origin master**
    - Get changes from github to your computer
        - **git pull**
    - Send changes on your computer to github
        - **git push**
- To get someone else's repository onto your computer
    - **git clone https://github.com/mickley/planets.git**
        - Each repository on github has a url: Click the green "Clone or Download" button to get it

Always pull before you start work

A resource I found helpful : https://happygitwithr.com/

Some more git help: https://github.com/k88hudson/git-flight-rules