Setting Up a Data Science Project

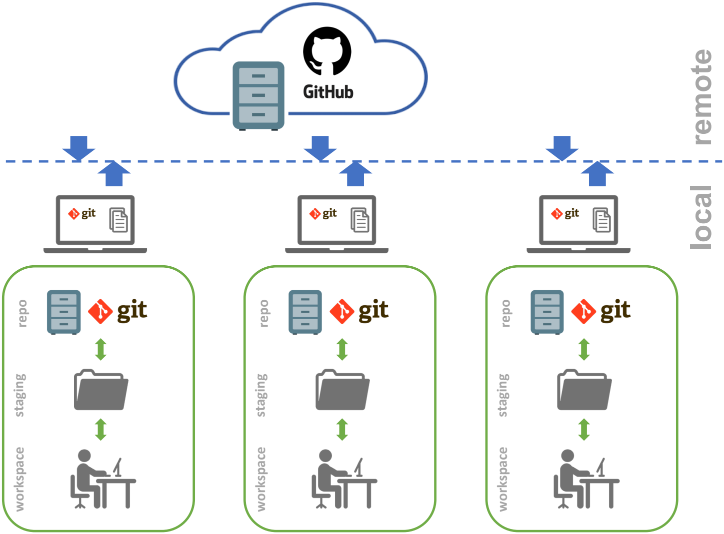
Marco Morales
marco.morales@columbia.edu

Nana Yaw Essuman
nanayawce@gmail.com

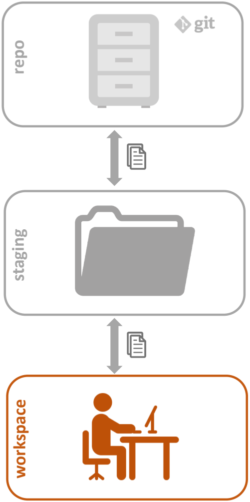
GR5069
Topics in Applied Data Science
for Social Scientists

Spring 2020
Columbia University

Last week we introduced DS projects in a broad context



Today we'll go deep into how to organize your workspace...



RECAP: A Data Science Project

- Three **aims** of a data science project
  a) **reproducibility**
     - anyone should be able to arrive to your **same results**
  b) **portability**
     - anyone should be able to **pick up where you left off** on any machine
- crucial tenets for **collaborative work**
  c) **scalability**
     - your project should also work for **larger data sets** and/or be on the path of **automation**

Structuring DS projects

some basic principles...

1. use **scripts for everything** you do
   - **NEVER** do things **manually**
2. organize your scripts in a sequence
   - **separate activities** in sections
   - keep an early section for **definitions**
   - call **other scripts** when necessary
3. write **efficient** (aka lazy) code
   - turn code used multiple times into **functions**
   - **re-use functions:** make them generic enough
4. rely on **version control** (Git)

Structuring DS projects

portability tricks...

- use a sensible **folder structure** (more later)
  - create folder clusters aligned with purposes
- use **relative paths** in your scripts
  - `"data//external//ARCH535.csv"` as opposed to `"C://users//data//external//ARCH535.csv"`
- take advantage of tools like `here()` package to ease your life

Structuring DS projects

a thin layer...

```
project\
|
| -- src                      <- Code
|
| -- data                     <- Inputs
|
| -- reports                  <- Outputs
|
| -- references               <- Data dictionaries,
|                                explanatory materials.
|
| -- README.md
| -- TODO                     <- (opt)
| -- LabNotebook              <- (opt)
```

Structuring projects

a thin layer...

```
project\
|
| -- src
|      |-- data             <- code to read/munge raw data
|      |-- features         <- code to transform/append data
|      |-- models           <- code to analyze data
|      |-- visualizations   <- code to create visualizations
|
| -- data
|
| -- reports
|
| -- references
|
| -- README.md
| -- TODO
| -- LabNotebook
```

- **principle:** separate function definition and application

Structuring projects

a thin layer...

- use `src` to organize your code
- use **one script per purpose**
- use **version control to "update"** your scripts
- use code to document **"manual" changes**
- call **additional scripts** as needed
- if too many functions, keep a **script with functions**

Structuring projects

a thin layer...

```
#   ######################################################################
#       File-Name:      MakeGraphs_CongressRollCall_160603.R
#       Version:        R 3.3.1
#       Date:           June 03, 2016
#       Author:         MM
#       Purpose:        Exploratory graphs of congressional roll call
#                       data for the 112th US Congress. Simple initial
#                       visualizations to find patterns and outliers.
#       Input Files:    ProcessedRollCall_160225.csv
#       Output Files:   Graph_RollCall_112Congress.gif
#       Data Output:    NONE
#       Previous files: MakeGraphs_CongressRollCall_160524.R
#       Dependencies:   GatherData_CongressRollCall_160222.R
#       Required by:    NONE
#       Status:         IN PROGRESS
#       Machine:        personal laptop
#   ######################################################################


library(ggplot2)
library(dplyr)
```

- **principle:** include all relevant information for each script

Structuring projects

a thin layer...

```
project\
 |
 | -- src
 |
 | -- data
 |      |-- raw              <- original, immutable data dump
 |      |-- external         <- data from third party sources
 |      |-- interim          <- intermediate transformed data
 |      |-- processed        <- final processed data set(s)
 |
 | -- reports
 |
 | -- references
 |
 | -- README.md
 | -- TODO
 | -- LabNotebook
```

- **principle:** input raw data and its format is always immutable

Structuring projects

a thin layer...

- ALWAYS keep your **raw data as immutable**
- keep **external data** separate and immutable
- if/when needed **keep interim data for validation**
- **processed data is ALWAYS replaceable!**
- all data should be linked to a script in `src`
- **document** origin of **raw & external data**

Structuring projects

a thin layer...

```
project\
|
| -- src
|
| -- data
|
| -- reports
|      |-- documents       <- documents synthesizing the analysis
|      |-- figures         <- images generated by the code
|
| -- references
|
| -- README.md
| -- TODO
| -- LabNotebook
```

- **principle:** outputs are disposable

Structuring projects

a thin layer...

- use whichever **document works best** for your purpose
  - reports (R Markdown, Jupyter notebooks)
  - decks
  - papers
- **reports can be updated** and are **subject to change**
- use reports to document **deeper analysis/visualizations** in detail

Structuring projects

a thin layer...

```
project\
|
| -- src
|
| -- data
|
| -- reports
|
| -- references           <- data dictionaries, explanatory materials
|
| -- README.md
| -- TODO
| -- LabNotebook
```

- **principle:** keep as much documentation as possible for your (future) reference and others'

Structuring projects

a thin layer...

```
R version 3.4.3 (2017-11-30)
Platform: x86_64-apple-darwin15.6.0 (64-bit)
Running under: macOS High Sierra 10.13.2

Matrix products: default
BLAS: /System/Library/Frameworks/Accelerate.framework/Versions/(...)/A/libBLAS.dylib
LAPACK: /Library/Frameworks/R.framework/Versions/3.4/Resources/lib/libRlapack.dylib

locale:
[1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods   base

other attached packages:
 [1] bindrcpp_0.2    reshape2_1.4.3  stringr_1.2.0   lubridate_1.7.1 magrittr_1.5
 [6] dplyr_0.7.4     readxl_1.0.0    readr_1.1.1     here_0.1        tidyr_0.7.2

loaded via a namespace (and not attached):
 [1] Rcpp_0.12.14     rprojroot_1.3-1  assertthat_0.2.0 plyr_1.8.4       cellranger_1.1.0
 [6] backports_1.1.2  stringi_1.1.6    rlang_0.1.6      tools_3.4.3      glue_1.2.0
[11] hms_0.4.0        yaml_2.1.16      rsconnect_0.8.5  compiler_3.4.3   pkgconfig_2.0.1
[16] bindr_0.1        tibble_1.3.4
```

- ... and document as much as you can about your session

Structuring projects

a thin layer...

```
project\
|
| -- src
|      |-- data             <- code to read/munge raw data
|      |-- features         <- code to transform/append data
|      |-- models           <- code to analyze data
|      |-- visualizations   <- code to create visualizations
|
| -- data
|      |-- raw              <- original, immutable data dump
|      |-- external         <- data from third party sources
|      |-- interim          <- intermediate transformed data
|      |-- processed        <- final processed data set
|
| -- reports
|      |-- documents        <- documents synthesizing the analysis
|      |-- figures          <- images generated by the code
|
| -- references             <- data dictionaries, explanatory materials
|
| -- README.md              <- high-level project description
| -- TODO                   <- future improvements, bug fixes (opt)
| -- LabNotebook            <- chronological records of project (opt)
```

Sources: **Cookiecutter for Data Science**, **ProjectTemplate**

Structuring projects

yet another layer for naming conventions...

`FinalProject_final_ThisOneForReal_LastOne.R`

- may not be easy to remember, or scalable for reproducibility

- A few pointers:
  - create a specific structure for your filenames

    `[FUNCTION]_[PROJECT]_[VERSION]`
  - use same function names consistently across projects

    i.e. `GatherData` for ETL, `MakeGraphs` for visualizations...
  - no special characters, replace spaces with underscores

Setup Tools for Class

what will be necessary in class...

- Getting Started with Apache Spark:
  - an open-source distributed cluster-computing framework
  - provides an interface for programming entire clusters with implicit data parallelism and fault tolerance
  - has 5 main components:
    - Spark Core
    - Spark SQL
    - Spark Streaming
    - MLlib
    - GraphX
  - Databricks was formed by the creators of Apache Spark to make it more efficient to utilize Spark and manage Spark Clusters

Setup Tools for Class

Let's set up Databricks:

Setup Tools for Class

what will be necessary in class...

- Getting Started with Cloud Computing "The Cloud":
  - on-demand availability of computer system resources, especially data storage and computing power, without direct active management by the user
  - ability to scale elastically - for example, more or less computing power, storage, bandwidth-right when they're needed, and from the right geographic location
  - three main deployment models of Cloud Computing:
    - Private Cloud
    - Public Cloud
    - Hybrid cloud

Setup Tools for Class

Let's set up AWS Educate:

Setting Up a Data Science Project

Marco Morales
marco.morales@columbia.edu

Nana Yaw Essuman
nanayawce@gmail.com

GR5069
Topics in Applied Data Science
for Social Scientists

Spring 2020
Columbia University