Coding Workshop: Empirical Organization[1]

Kyle Coombs (Columbia)

February 24, 2022

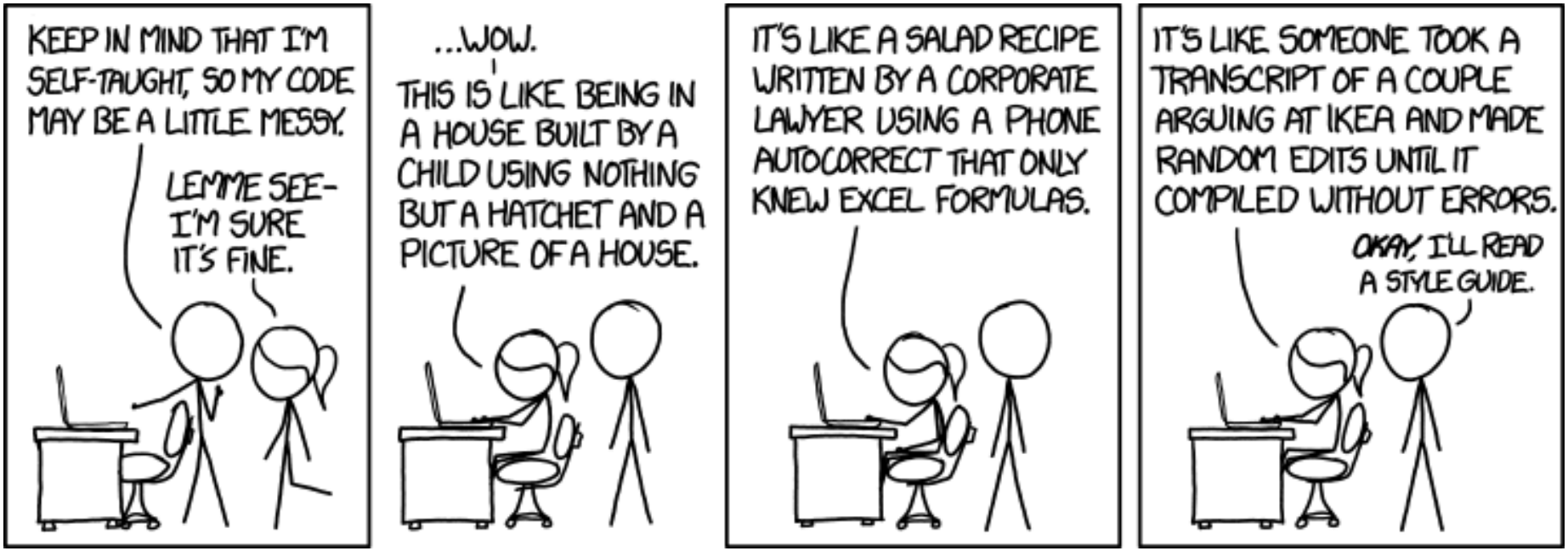


Figure: xkcd

[1] Adapted from Causal Inference and Research Design by Scott Cunningham (Baylor), Lucas Husted (Columbia), Len Goff (Columbia)

Welcome

- This is a walkthrough for Columbia economics students
- Plan for today:
  1. How to organize a folder
  2. Where to get started with Stata and how to use a *.do* file
     Stata organization basics (how to use a *.do* file)
  3. Stata commands everyone should know
  4. R workflow and commands everyone should know
  5. Python commands everyone should know

About me

- 5th year economics PhD
- Graduated in 2014 from Macalester College
- Lived in Peru for a year working with an Ag nonprofit and then making content about artisans for a Fair Trade nonprofit
- RA'd at the Federal Reserve Board in Consumer Finance
- I've dabbled in labor, public finance, political economy and behavioral
- Projects on: charitable giving, discrimination, formal and informal unemployment insurance, schooling & catholic scandals
- I am a great resource for specific project questions – during office hours (Mon, Wed, Thurs. at 2-3pm and Tues. 10:30-11:30am)

Goal Today

- Teach the toolkit to keep your empirical workflow organized
- Suggest some best practices for writing your code
- Give syntax for common data tasks across the different languages
- Most importantly: Give you the toolkit to effectively learn how to debug your own code[2] and read code written by other researchers
- Not the goal: Fluency or proficiency in any language[3]

[2] Or help a colleague/TA/professor more efficiently help you write your code
[3] Some days, I barely have that!

Textbooks: Smarter people than me

Helpful Textbooks

1. Cunningham (2021) Causal Inference: The Mixtape (Also, free version on his website)
2. Huntington-Klein (2022) The Effect
3. Angrist and Pischke (2009) Mostly Harmless Econometrics (MHE)
4. Morgan and Winship (2014) Counterfactuals and Causal Inference (MW)
5. Sweigart (2019) Automate The Boring Stuff With Python

Non-textbook readings

- The help documentation associated with your language (no really)
- Jesse Shapiro's "How to Present an Applied Micro Paper"
- Gentzkow and Shapiro's coding practices manual
- Ljubica "LJ" Ristovska's language agnostic guide to programming for economists
- Grant McDermott on Version Control using Github `https://raw.githack.com/uo-ec607/lectures/master/02-git/02-Git.html#1`

Helpful for troubleshooting

- The help documentation associated with your language (no really)
- All languages `https://stackoverflow.com` `https://stackexchange.com`
- Stata-specific (all hail Nick Cox) `https://www.statalist.org/forums/forum/general-stata-discussion/general`
- Cheatsheets! Stata Rstudio Python
- Me. Sign up for office hours cause it is the best part of this job.

Learn by Immersion

- Just like learning a real language, no amount of talking today will teach you how to use any program.
  - You have to need to use it (immersion) to learn it.
  - Google is your dictionary.
  - Help files are your grammar books
  - A great way to start coding is to see lots of other people's code and copy what you read
- You must learn how to ask the "right" question:
  - Never: "Importing csv file into stata not working"
  - Better: "import csv stata [specific error message]"
  - Better still: "import delimited using csv [specific error message]"
  - set trace on/off around a buggy command can help reveal where the error happens inside the black box

Reducing empirical chaos

"Sad story"

- Once upon a time there was a boy who was writing a job market paper on unemployment insurance during the pandemic
- This boy presented the findings a half dozen times, spoke to the media some, and generally thought he had cool results
- Several people suggested he look at a handful of other outcome series and try changing his analysis unit frequency from monthly to weekly
- He also knew that he needed to restrict his sample to reduce noise

The horror!

- But then after making these changes and re-running his code that took two days, his new sample dropped by 50 percent!
- He was, understandably, terrified.
- The young boy spent a week looking for the fix weeding through six different versions of the .do and .dta files with suffixes like _*v1* and _*test* and _*test2* and _*final_I_swear* and _*okay_i_lied*
- Finally he discovered the phrase:

```
drop if insample_new==1
```

instead of

```
keep if insample_new==1
```

- The boy was very frustrated and decided to work on these slides while re-running his code.

Cunningham Empirical Workflow Conjecture

- The cause of most of your errors is **not** due to insufficient knowledge of syntax in your chosen programming language
- The cause of most of your errors is due to a poorly designed empirical workflow

Empirical workflow

- Workflow is a fixed set of routines you bind yourself to which when followed identifies the most common errors
    - Think of it as your morning routine: alarm goes off, go to wash up, make your coffee/tea, check Twitter, repeat *ad infinitum*
- Finding the outlier errors is a different task; empirical workflows catch typical and common errors created by the modal data generating processes
- Empirical workflows follow a checklist

Why do we use checklists?

- I am going to Kenya in March[4] for a wedding and I need a visa
- So I have prepared checklist of things that I needed:
  - Passport, flight information, travel itinary, hotel bookings, clear photograph, $51
- The empirical checklist is solely referring to the intermediate step between "getting the data" and "analyzing the data"
- It largely focuses on ensuring data quality for the most common, easiest to identify, situations you'll find yourself in
- They'll make you a better coauthor

[4] Yes, this is a flex. Learn to code and you can also be this cool.

Step 1: Organize your directories

- How do coding error fiascos happen?
- In part because of four problems related to
  1. organized subdirectories
  2. automation
  3. naming conventions
  4. version control
- I'll discuss each but I highly recommend that you read Gentzkow and Shapiro's excellent resource "Code and Data for the Social Sciences: A Practitioner's Guide" `https://web.stanford.edu/~gentzkow/research/CodeAndData.pdf`

No correct organization

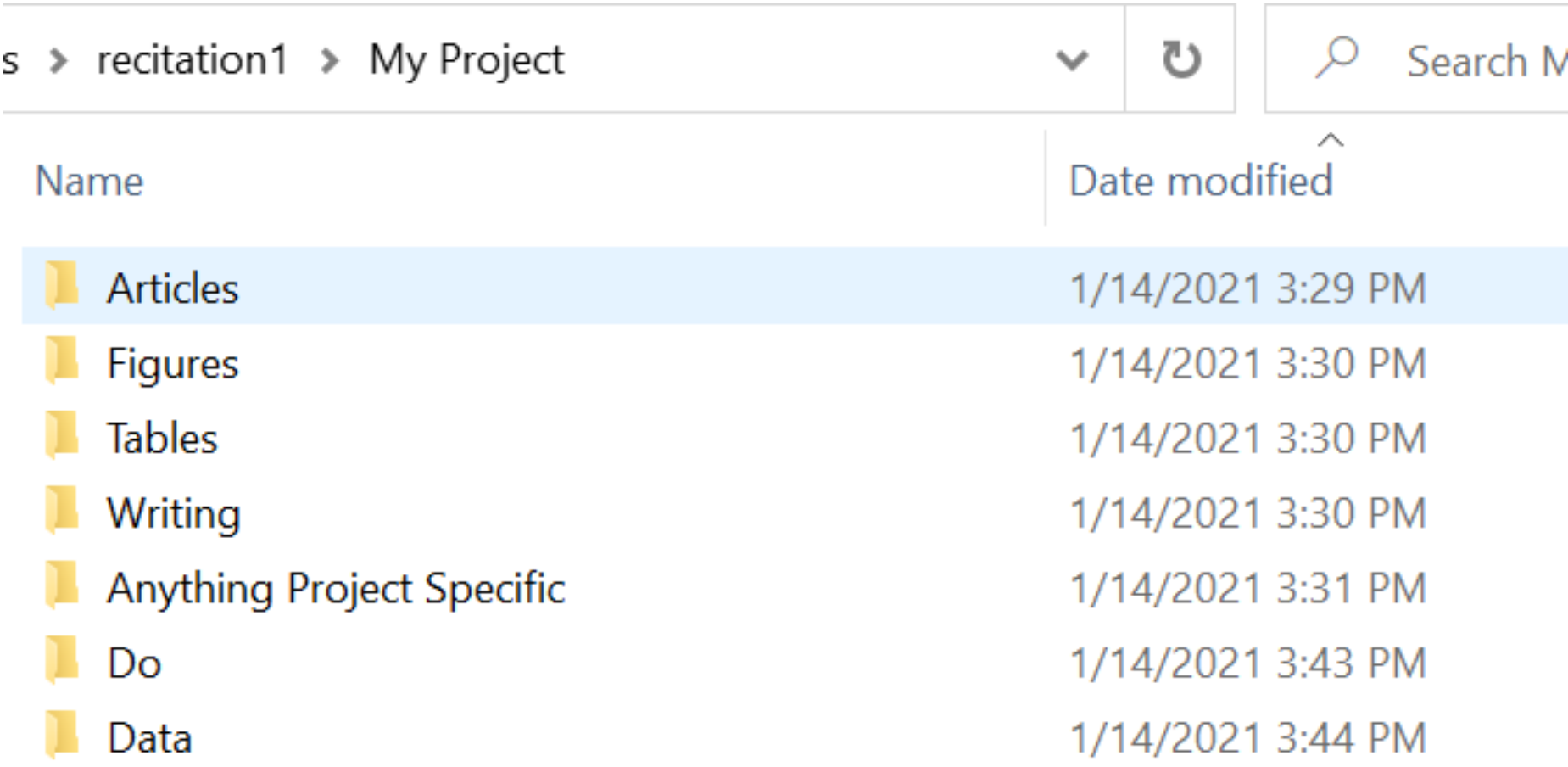
- There is no correct way to organize your directories,
- But all competent empiricists have adopted an intentional philosophy of how to organize their directories
- Why? Because you're writing for your future self, and your future self is lazy, distracted, disinterested and busy[5]
- This will also make you a better coauthor

[5] Also, my future self is generally not a fan of my past or present self's work.

Directories

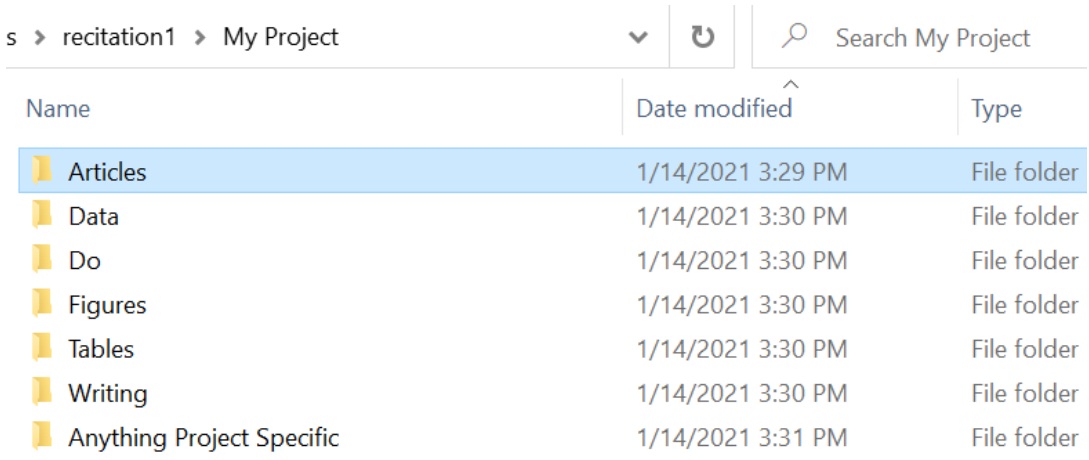
- The typical applied micro project may have hundreds of files of various type and will take *years* just to finish not including time to publication
- So simply finding the files you need becomes more difficult if everything is stored in the same place
- When starting a new project, it is best to create something like the following directories

Subdirectory organization



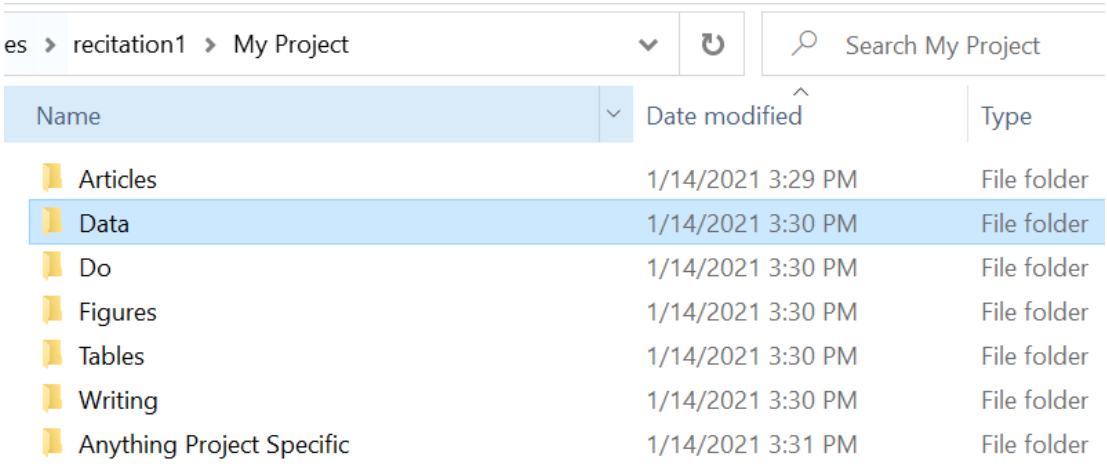
1) Name the project (“My Project”)

Subdirectory organization

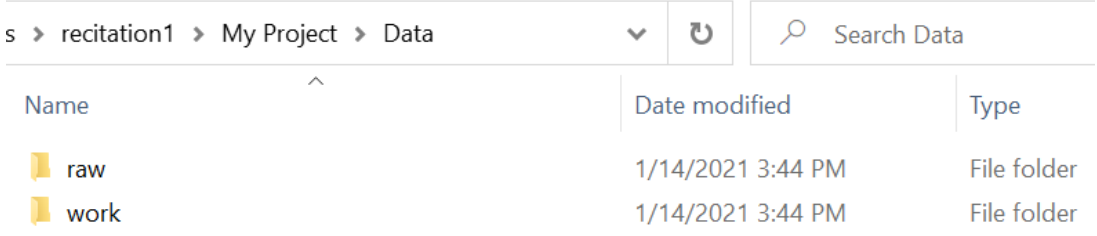


2) A subdirectory for all articles you cite in the paper

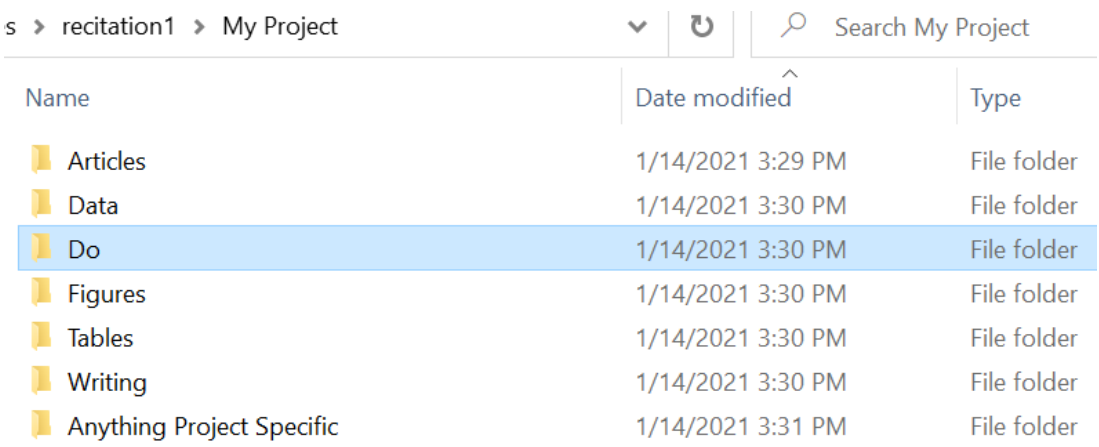
Subdirectory organization



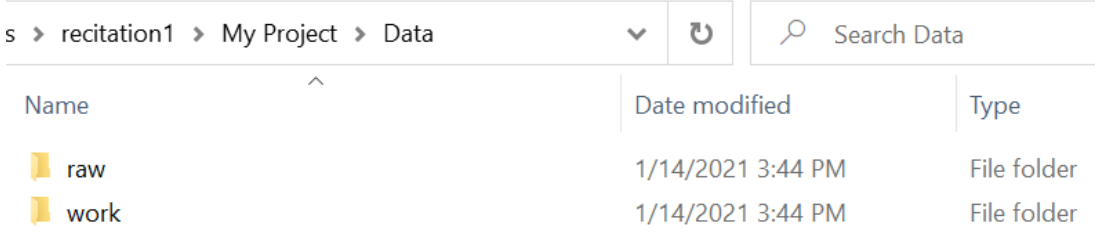
3) Data subdirectory containing all datasets, may contain /raw/ and /work/ folders



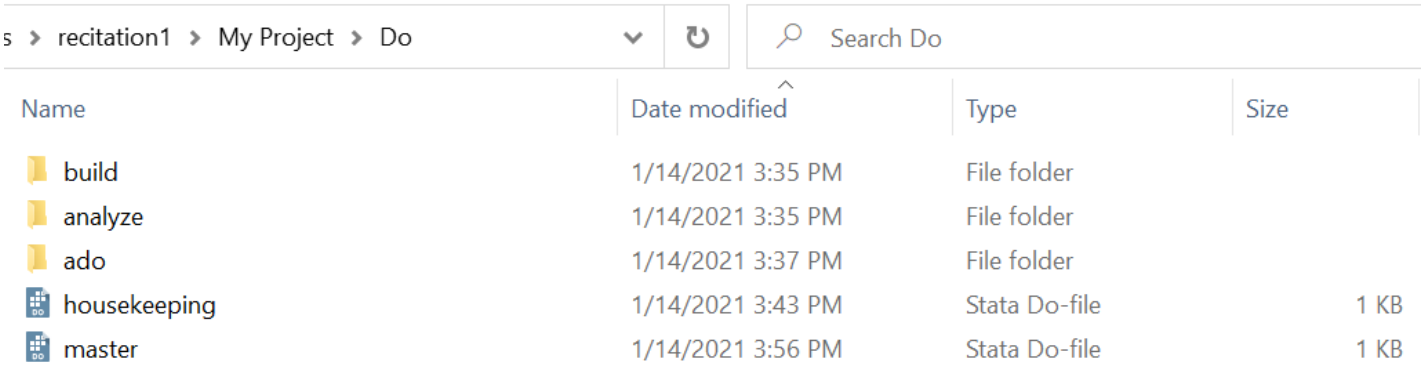
Subdirectory organization



4) A subdirectory for all do files and log files



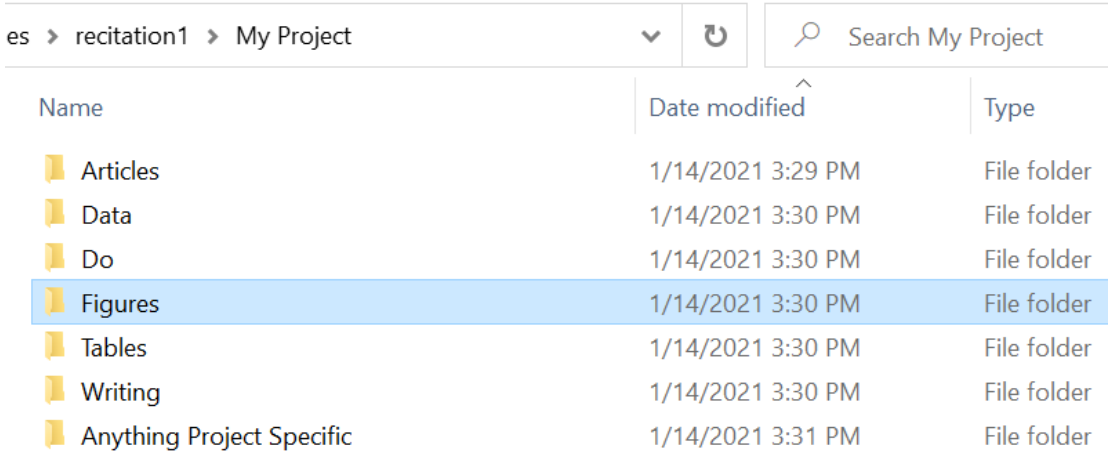
Subdirectory organization



4) Within Do, you may create:

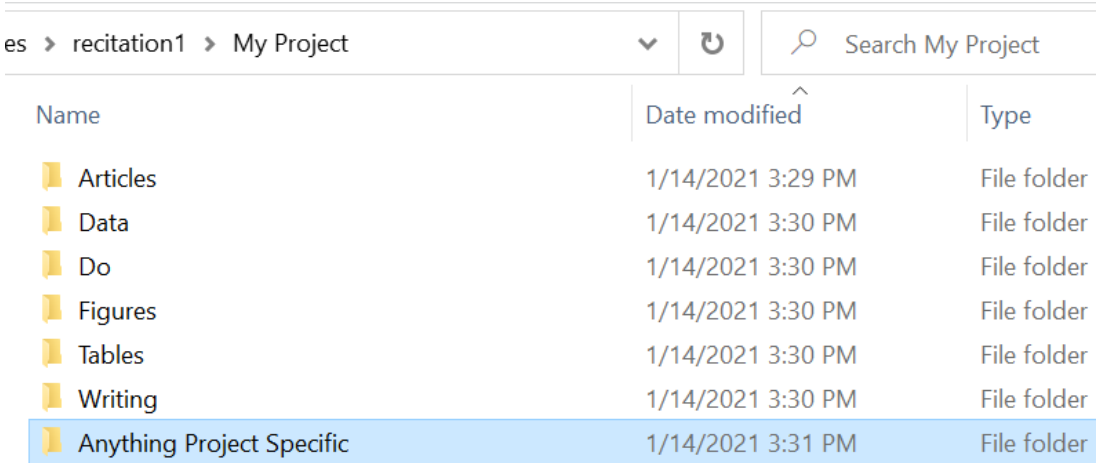
a. /build/ – for importing, cleaning, merging, appending
b. /analyze/ – for analyzing the data
c. /ado/ – a folder for utility programs/functions that are not directly part of the workflow
d. Housekeeping & Master files (we'll get to that)

Subdirectory organization



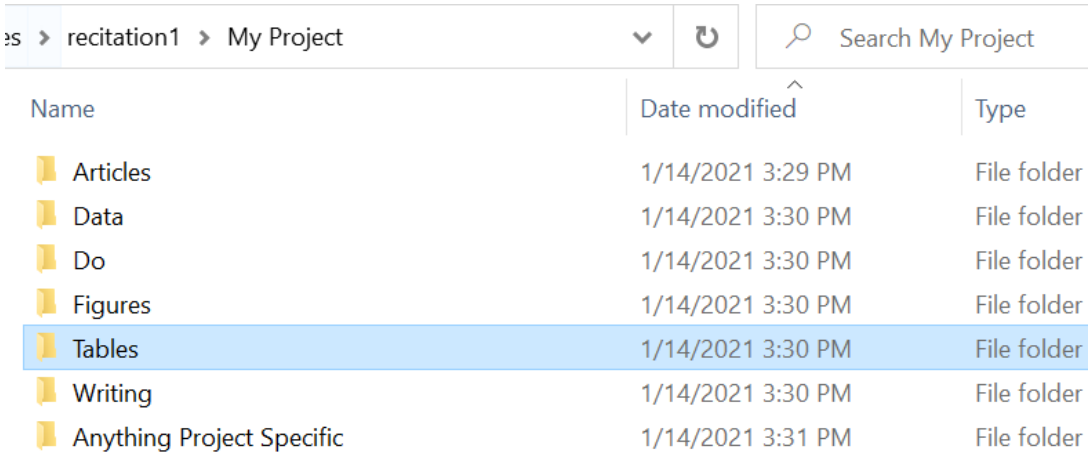
5) All figures produced by Stata or image files

Subdirectory organization



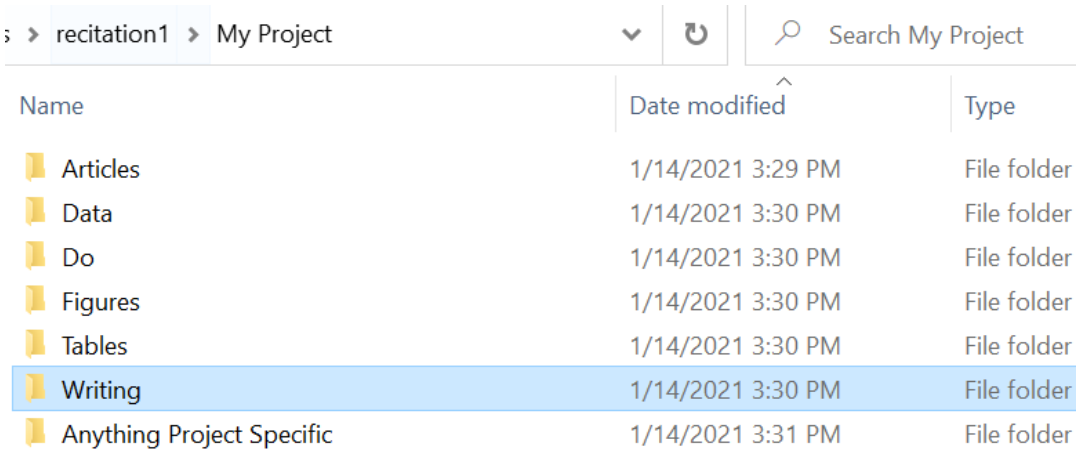
6) Project-specific heterogeneity (e.g., “Inference”, “Grants”, “Interview notes”, “Presentations”, “Misc”)

Subdirectory organization



7) All tables generated by Stata (e.g., .tex tables produced by -estout-)

Subdirectory organization



8) A subdirectory reserved only for writing

The housekeeping file – automate the boring stuff

```
*******************************************************************************
*File: housekeeping.do
*By: Kyle Coombs
*What: Creates globals, ado path, and installs relevant data
*Date: 2021/1/14
*******************************************************************************

global project = "/PROJECT DIRECTORY/"

global data     = "$project/Data/"
global raw      = "$data/raw"
global work     = "$data/work"
global code     = "$project/Do/"
global build    = "$code/build/"
global analyze  = "$code/analyze/"
global ado      = "$code/ado/"

global tables   = "$project/Tables"
global figures  = "$project/Figures"

adopath ++ $ado

ssc install esttab
ssc install gtools
gtools, upgrade

**Anything else that you might need to pre-specify at the start of the project
```

Figure: Some argue that you should have no absolute directory paths and instead define all projects in a "profile" file: `https://julianreif.com/guide/`.

Master File

```
*******************************************************************************
*File: housekeeping.do
*By: Kyle Coombs
*What: Runs project from start to finish
*Date: 2021/1/14
*******************************************************************************

run housekeeping.do

*Process and import data
run $build/import_rawfile1.do
run $build/import_rawfile2.do
run $build/import_rawfile3.do

*Clean raw data
run $build/clean_rawfile1.do
run $build/clean_rawfile2.do
run $build/clean_rawfile3.do

*Merge files 1 and 2
run $build/merge_file1_file2.do

*Append merged file 1 and 2 to file 3
run $build/append_file1and2_file3.do

*Analysis

run $analyze/data_verification_tests.do
run $analyze/summary_statistics.do
run $analyze/basic_regression_tables.do
run $analyze/cool_figure_everyone_remembers.do
run $analyze/robustness_checks.do
```

Figure: Create a master file to run the project's code from start to finish

Master Files: Python & R

```r
#File: master.R
#By: Kyle Coombs
#What: Runs the project from start to finish in Python
#Date: 2022/02/02

#Install packages with housekeeping. Also put together paths.
source('housekeeping.R')
#User written functions can be sourced -- or you could write a package, your call
source(paste0(build,'clean_functions.R'))
source(paste0(analysis,'analysis_functions.R'))

#Import files
df1 <- read_csv(paste0(raw,'file1.csv'))
df2 <- read_parquet(paste0(raw,'file2.parquet'))
df3 <- read_dta(paste0(raw,'file3.dta'))

#Clean files
cleaned_df1  <- clean_df1(df1)
cleaned_df2  <- clean_df2(df2)
cleaned_df3  <- cf.clean_df3(df3)

#Merge files 1 to 2
merged_df1_df2 = merge(cleaned_df1, cleaned_df2, on=c('merge','vars'))

#Append file 1 to
append_df1_df2_df3 = rbind(merged_df1_df2, cleaned_df2)

#Analysis
sum_stats=summary_stats(append_df1_df2_df3,stats=['mean','median','max'])
reg_results=basic_regression(append_df1_df2_df3)

#Tables will likely be made with a host of R packages
make_sum_figures(sum_stats)
make_figures(reg_results)
make_sum_tables(sum_stats)
make_tables(reg_results)
```

(a) Master.R

```python
'''
File: master.py
By: Kyle Coombs
What: Runs the project from start to finish in Python
Date: 2022/02/02
'''

import housekeeping as hp
import import_functions as imp
import clean_functions as cf
import utility_functions as utils
import analysis_functions as af
import pandas as pd
import numpy as np

#Import files
df1 = imp.read_file(hp.paths('file1'))
df2 = imp.read_file(hp.paths('file2'))
df3 = imp.read_file(hp.paths('file3'))

#Clean files
cleaned_df1 = cf.clean_df1(df1)
cleaned_df2 = cf.clean_df2(df2)
cleaned_df3 = cf.clean_df3(df3)

#Merge files 1 to 2
merged_df1_df2 = cleaned_df1.merge(cleaned_df2)

#Append file 1 to
append_df1_df2_df3 = pd.concat(merged_df1_df2, df1, axis=1)

#Analysis
sum_stats=af.summary_stats(append_df1_df2_df3,stats=['mean','median','max'])
reg_results=af.basic_regression(append_df1_df2_df3)
af.make_sum_figures(sum_stats)
af.make_figures(reg_results)
af.make_sum_tables(sum_stats)
af.make_tables(reg_results)
```

(b) Master.py

Figure: R and Python master files often call functions instead of files in a workflow. Other folks may have a different organization scheme for these languages. There are millions of guides and examples online.

Checklist Part 3: Simple data checks

- Your data checks should be a few simple, yet non-negotiable, programming commands and exercises to check for coding errors
- I will mostly use Stata commands for expositional simplicity – you can and should do the same things in Python, R, Julia, Matlab, even... SAS

Time

- People often think empirical research is about "getting the data" and "analyzing the data"
- They have an "off to the races" mindset
- Just like running a marathon involves far far more time training than you ever spend running the marathon, doing empirical research involves far far more time doing tedious, repetitive tasks
- Since you do the tedious tasks repeatedly, they have the *most* potential for error which can be catastrophic
- How can we minimize these errors through a checklist?

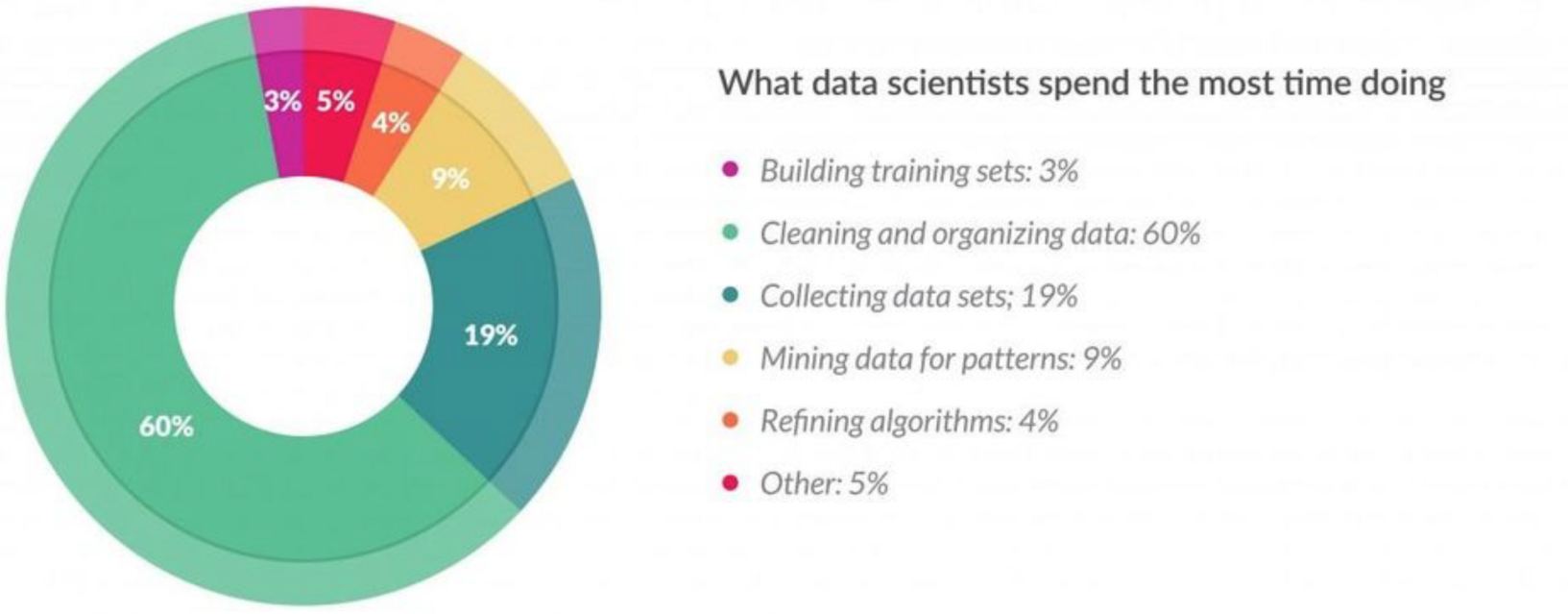


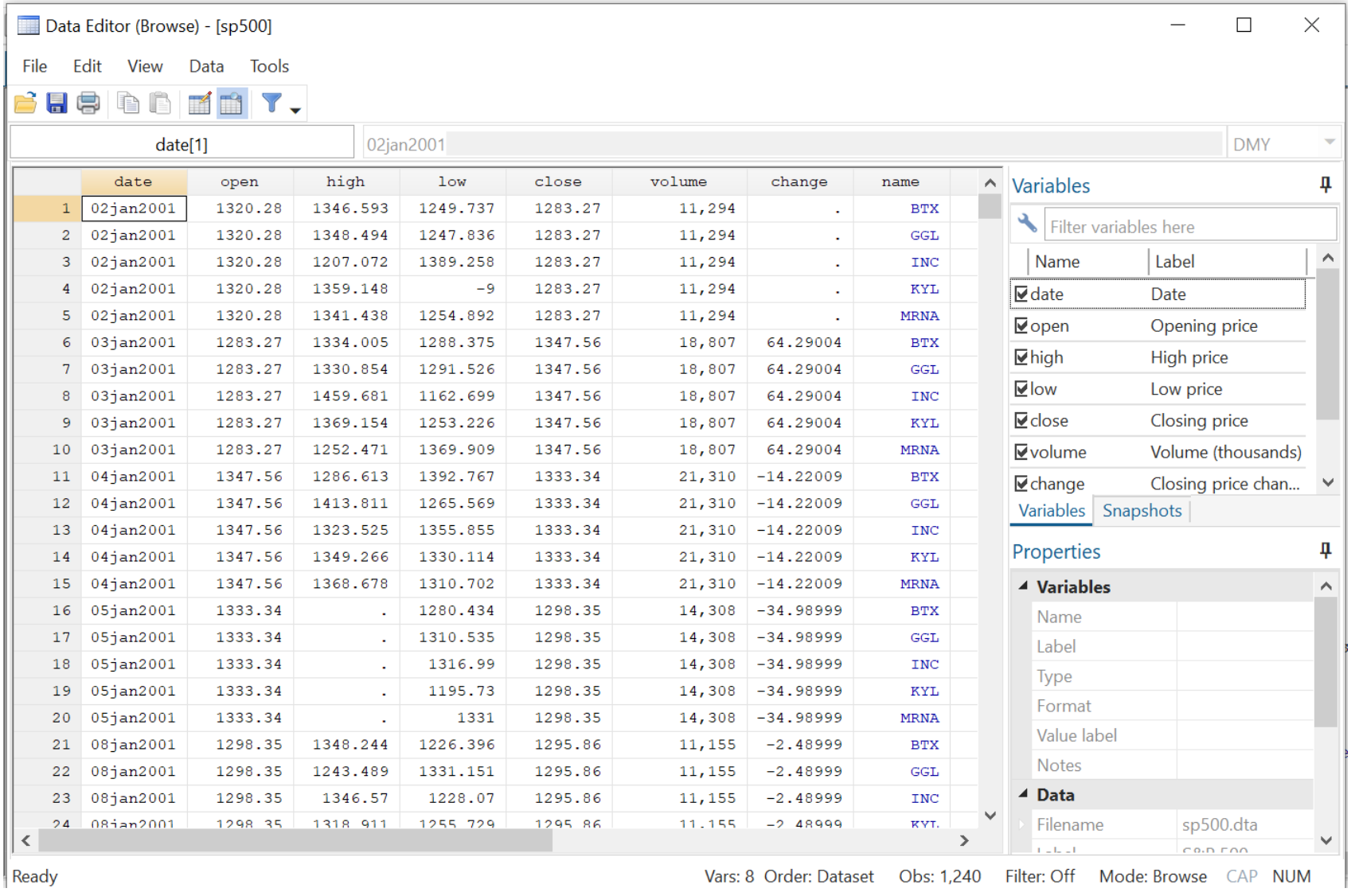
Figure: Image from Wenfei Xu at Columbia GSAPP

Read the codebook

- Few like reading the codebook as it is not gripping literature
- But the codebook explains how to interpret the data you have acquired and it is not a step you can skip
- Set aside time to study it, and have it in a place where you can regularly return to it
- This goes for the `readme` that accompanies some datasets, too.

Look at the data

- "Real eyes realize real lies" –Troy Ave via some dude from my high school
- The eyeball is not nearly appreciated enough for its ability to spot problems
- Use `browse` to just read the spreadsheet with your eyes.
- Scroll through the variables and familiarize yourself with what you've got visually
- Plot the sums/averages over time or some other relevant dimension



	date	open	high	low	close	volume	change	name
1	02jan2001	1320.28	1346.593	1249.737	1283.27	11,294	.	BTX
2	02jan2001	1320.28	1348.494	1247.836	1283.27	11,294	.	GGL
3	02jan2001	1320.28	1207.072	1389.258	1283.27	11,294	.	INC
4	02jan2001	1320.28	1359.148	-9	1283.27	11,294	.	KYL
5	02jan2001	1320.28	1341.438	1254.892	1283.27	11,294	.	MRNA
6	03jan2001	1283.27	1334.005	1288.375	1347.56	18,807	64.29004	BTX
7	03jan2001	1283.27	1330.854	1291.526	1347.56	18,807	64.29004	GGL
8	03jan2001	1283.27	1459.681	1162.699	1347.56	18,807	64.29004	INC
9	03jan2001	1283.27	1369.154	1253.226	1347.56	18,807	64.29004	KYL
10	03jan2001	1283.27	1252.471	1369.909	1347.56	18,807	64.29004	MRNA
11	04jan2001	1347.56	1286.613	1392.767	1333.34	21,310	-14.22009	BTX
12	04jan2001	1347.56	1413.811	1265.569	1333.34	21,310	-14.22009	GGL
13	04jan2001	1347.56	1323.525	1355.855	1333.34	21,310	-14.22009	INC
14	04jan2001	1347.56	1349.266	1330.114	1333.34	21,310	-14.22009	KYL
15	04jan2001	1347.56	1368.678	1310.702	1333.34	21,310	-14.22009	MRNA
16	05jan2001	1333.34	.	1280.434	1298.35	14,308	-34.98999	BTX
17	05jan2001	1333.34	.	1310.535	1298.35	14,308	-34.98999	GGL
18	05jan2001	1333.34	.	1316.99	1298.35	14,308	-34.98999	INC
19	05jan2001	1333.34	.	1195.73	1298.35	14,308	-34.98999	KYL
20	05jan2001	1333.34	.	1331	1298.35	14,308	-34.98999	MRNA
21	08jan2001	1298.35	1348.244	1226.396	1295.86	11,155	-2.48999	BTX
22	08jan2001	1298.35	1243.489	1331.151	1295.86	11,155	-2.48999	GGL
23	08jan2001	1298.35	1346.57	1228.07	1295.86	11,155	-2.48999	INC
24	08jan2001	1298.35	1318.911	1255.729	1295.86	11,155	-2.48999	KYL

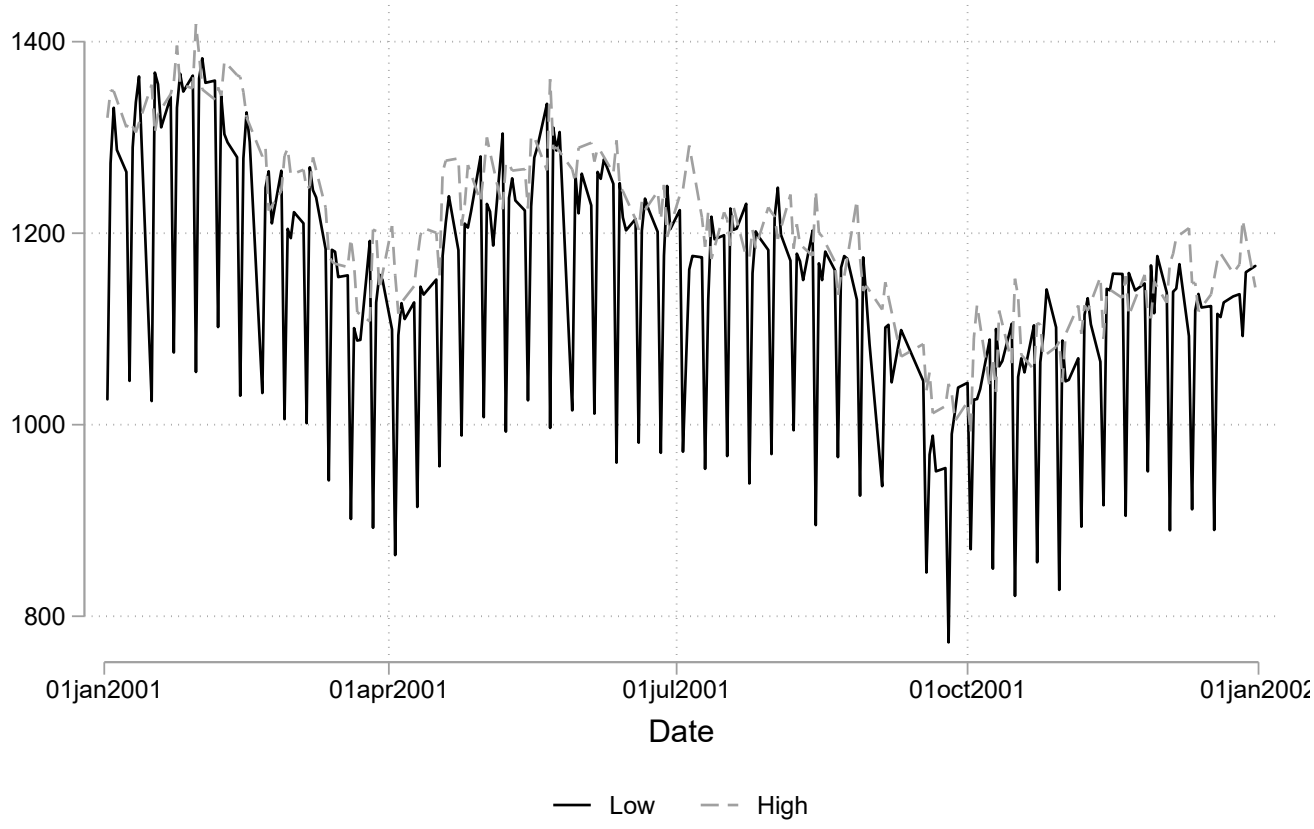


Figure: `collapse (sum) low high, by(date)`

```
replace high=0 if mi(high)
replace low=0 if mi(low)
twoway (line low high date)
```

Missing observations

- Check the size of your dataset in Stata using `count`
- Check the number of observations per variable in Stata using `summarize`
  - String variables will always report zero observations under `summarize` so `count if X==""` will work
- Use `tabulate` also because oftentimes missing observations are recorded with a -9 or some other illogical negative value

Missing time indicators

- Panel data can be overwhelming bc looking at each state/city/firm/county borders on the impossible
- Start with `collapse` to the national level by year/day/month and simply `browse` to see if anything looks strange
  - What's "strange" look like?
  - Well wouldn't it be strange if national unemployment rates were zero in any year?
- You can use `xtline` or `twoway, by()` to see time series for panel identifiers, with or without the subcommand of `overlay`

	date	high	low
1	02jan2001	1320.549	1026.545
2	03jan2001	1349.233	1273.147
3	04jan2001	1348.379	1331.001
4	05jan2001	.	1286.938
5	08jan2001	1310.691	1263.949
6	09jan2001	1316.471	1045.79
7	10jan2001	1311.422	1289.618
8	11jan2001	1305.224	1336.686
9	12jan2001	.	1363.732
10	16jan2001	1355.048	1024.71
11	17jan2001	1304.598	1367.732
12	18jan2001	1324.576	1355.544
13	19jan2001	.	1310.544

Figure: `collapse (mean) low high, by(date)`

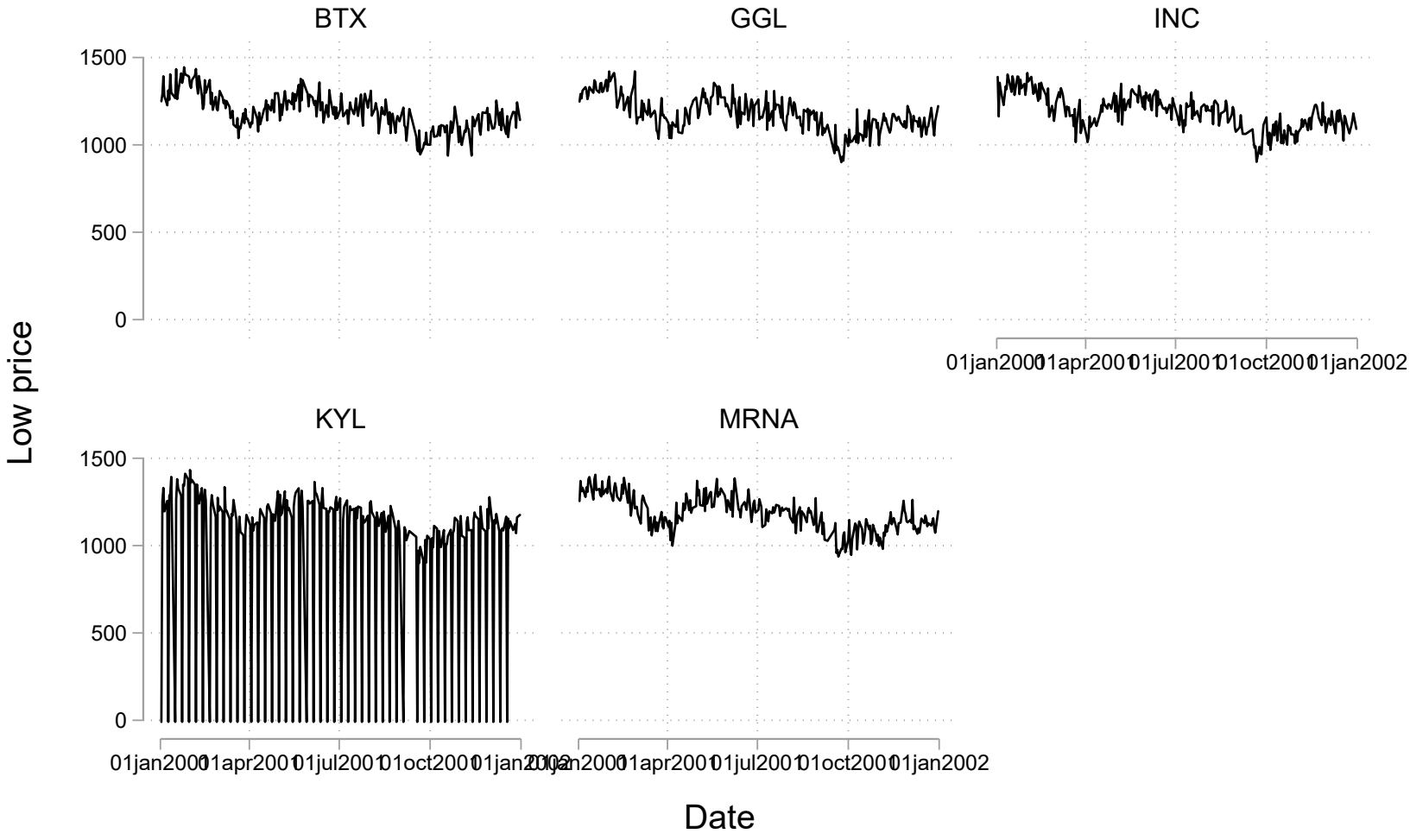


Figure: `twoway high date, by(names)`

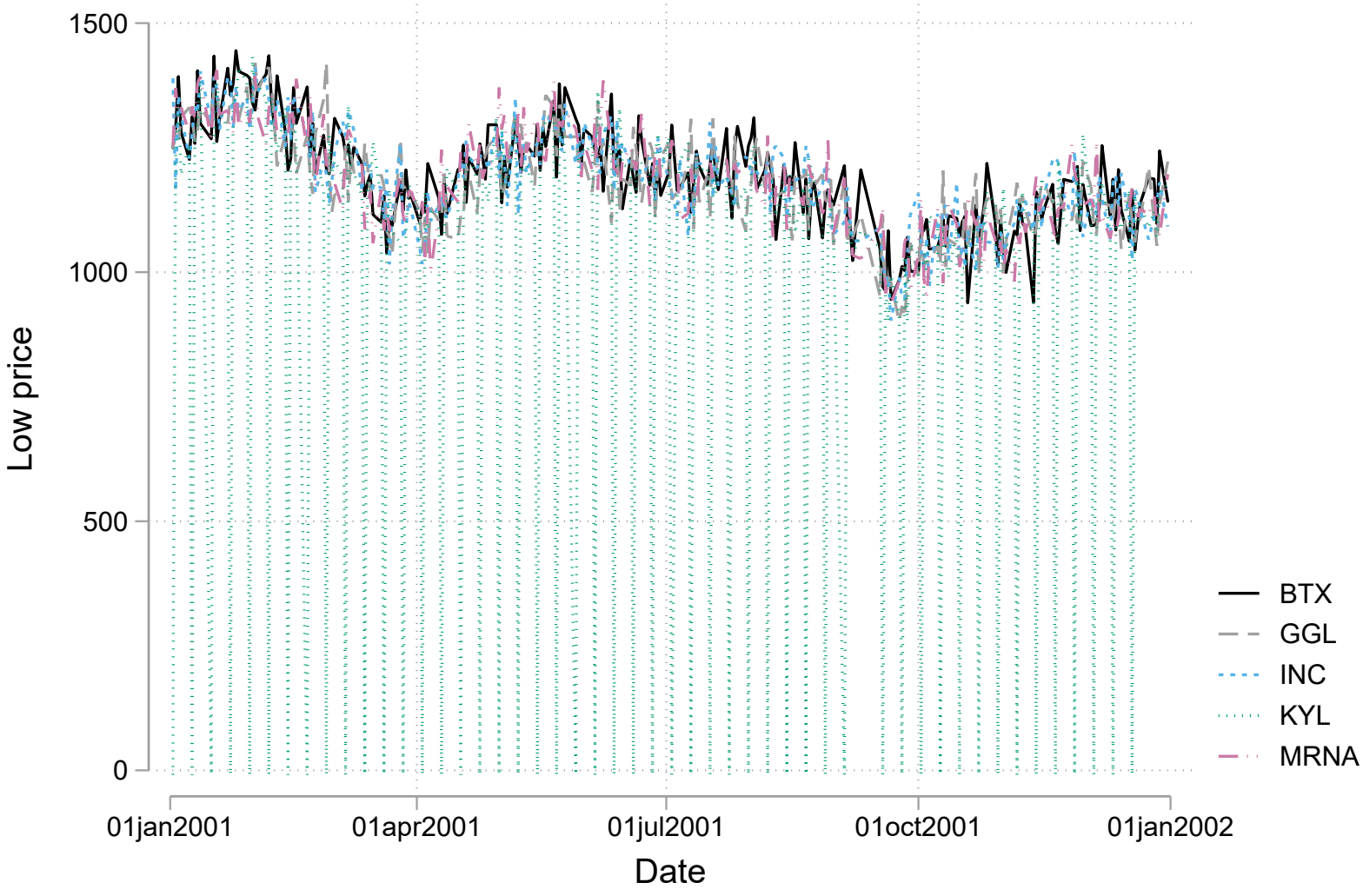


Figure: `xtset name date`
`xtline high date, overlay`

Panel observations are $N \times T$

- Say you have 51 state units (50 states plus DC) and 10 years
- $51 \times 10 = 510$ observations
- If you do not have 510 observations, then you have an unbalanced panel; if you have 510 observations you have a balanced panel
- Check the patterns using `xtdescribe` and simple counting tricks

Group counts

```
. gen one = 1

. bysort county_group: egen count=sum(one)

. ta count
```

count	Freq.	Percent	Cum.
24	48	0.42	0.42
36	36	0.31	0.73
48	48	0.42	1.15
96	96	0.84	1.99
120	480	4.19	6.18
156	312	2.72	8.90
180	10,440	91.10	100.00
Total	11,460	100.00	

If there's uneven counts, visualize them

Results

```
.
. tsset id date
       panel variable:  id (unbalanced)
        time variable:  date, 1995m1 to 2009m12
                delta:  1 month

.
.
end of do-file

. xtdescribe

      id:  4, 5, ..., 270                                    n =         69
    date:  1995m1, 1995m2, ..., 2009m12                      T =        180
           Delta(date) = 1 month
           Span(date)  = 180 periods
           (id*date uniquely identifies each observation)

Distribution of T_i:   min      5%     25%       50%       75%     95%     max
                        24      48     180       180       180     180     180

     Freq.  Percent    Cum. |  Pattern
 ---------------------------+---------------------------------------------------------------------------------------------------------------------------------------------------------------------
> _______________
       58     84.06   84.06 |  111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
> 111111111111111
        3      4.35   88.41 |  ............................................................111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
> 111111111111111
        2      2.90   91.30 |  ............................................................................................................................................................111111111
> 111111111111111
        2      2.90   94.20 |  ........................111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
> 111111111111111
        1      1.45   95.65 |  ................................................................................................................................................111111111111111111111
> 111111111111111
        1      1.45   97.10 |  ....................................................................................................................................111111111111111111111111111111111
> 111111111111111
        1      1.45   98.55 |  ....................................................................................111111111111111111111111111111111111111111111111111111111111111111111111111111111
> 111111111111111
        1      1.45  100.00 |  111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111.............................................
> ...............
 ---------------------------+---------------------------------------------------------------------------------------------------------------------------------------------------------------------
> _______________
       69    100.00         |  XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
> XXXXXXXXXXXXXXX

.
```

log on (text)

Don't forget the question

- "Exploring the data" is intoxicating to the point of distracting
- "All you can do is write the best paper on the question you're studying" – Mark Hoekstra
  - Note he didn't say "Write the best paper you're capable of writing"
  - He said **the best paper**
  - Important therefore to choose the right questions with real upside
- Slow down, think big picture, force yourself to figure out exactly what your question is, who is in your sample (and importantly who won't be) and what time periods you'll pull

Explain What You Want The Code To Do In Words And Work Backwards

- Recently a student came to OH and with a 100-line Stata *.do* file with nested loops, if statements, and "hardcode"
- At first the student asked how to debug a few broken locals, which I could fix
- Eventually, I asked, "What do you actually want this to do?"
- Answer: "Rename each of the variables and set the negative values in each equal to missing"
- The complex code worked, but it was hard to follow without extensive documentation
- With the goal in mind, we went to the help documentation shortened 100 lines to five crisp lines
- Moral: Describe you want in words and work backwards from that

Speak clearly

"Be conservative in what you do; be liberal in what you accept from others." - Jon Postel

- Smart sounding quote about both programming and relationships
- Your future self is time constrained, so explain *everything* to her as well as write clear code
- Optimally document your programs
- But speak your future self's love language so she understands

Always use scripting programs NOT GUI

- Guess what - your future self doesn't even remember making do files, tables or figures, let alone typing into GUI command line
- Therefore throw future you a bone, hold your hand and walk yourself exactly through everything
- Which means you've got to have replicable scripting files*
  - * Sure, sometimes use the the command line for messing around
  - But then put that messing around in the program

Good text editor

- Remember: the goal is to make beautiful programs
- Invest in a good text editor which has bundling capabilities that will integrate with Stata, R or LaTeX
- Textmate 2 is great for Mac and in addition to a Stata and R bundle, it also allows for *column* editing
- PC users tend to love Sublime/VS Code for the same reasons
- Stata and Rstudio also come with built-in text editors, which use slick colors for various types of programming commands

Headers

```
*************************************************************
* name: texas.do
* author: scott cunningham (baylor university)
* description: estimates the causal effect of prison capacity
*              expansion on incarceration rates using synth
* date: march 19, 2018
*************************************************************
```

Setup

```
*******************************************************************************
* Setup and Necessary Installs
*******************************************************************************
clear all               // gets rid of all data in memory
set more off            // useful so code runs without pausing.

* I set up path to the general folder as local variable for later (see below)

run housekeeping.do

cd $code

/*
This is a block of comment
So we can write lots of stuff in here or block it out when we run a file
*/
```

Figure: A common way to start a do file

Assert your truth

- All languages have some kind of "assert" syntax, in all three languages the syntax is `assert` *condition*[6]
- Remember how you confirmed certain things about your in the checklists? Assert can help you make sure those things stay true (if tey should)
- You assert a test, if the data pass the test, the code advances. If not, it breaks
- For example, make sure data only have adults `assert age>=18 & !missing(age)` in Stata or `assert data["age"]>=18` in Python
- Other useful in Stata: `isid` to confirm you have an ID variable

[6] R also has `stopifnot()`

Commenting Commenting Commenting

- All languages let you add comments in various ways:
  - Stata: $*$, //, /$**$/
  - R: %
  - Python: #
- Write informative comments explaining what role a command serves (don't just restate what a command does)
  - Bad: `mean price //this takes the mean`
  - Good: `mean price //record average price to normalize price`
- Where possible name functions and objects informatively

Get Help

- What happens if you get a bug you don't understand? Or you can't remember the syntax for a command or function?
- What if you pull some random user-written command off the internet?
  1. Try things until you're blue in the face
  2. Get pre-written help
- Good packages/languages have official documentation
- Stata: `help` *command*, R: `?`*command*, Python: `help()`
- Read help files to learn how to accomplish a specific task
- Don't peruse help files like they're a Pulitzer novel or a Buzzfeed listicle

Minimal reproducible example

- Many skilled users will ask you to create a minimal reproducible example of your problem to help debug
- That means cutting out project-specific stuff to isolate your bug
- It involves a small dataset (or program to generate it) and the lines of code that fail
- `dataex` is a great Stata tool for building an example
- https://stackoverflow.com/questions/5963269/how-to-make-a-great-r-reproducible-example
- https://stackoverflow.com/questions/20109391/how-to-make-good-reproducible-pandas-examples

Different elements

- Everyone needs a system for naming
  1. variables,
  2. datasets, and
  3. do files
- As these are the three things you repeatedly use, you need to have a system, even if not mine

Naming conventions for variables

- Variables should be readable to a stranger
  - Say that you want to create the product of two variables. Name it the two variables with an underscore
  - `gen price_mpg = price * mpg`
- Otherwise name the variable exactly what it is
  - gen bmi = weight / (height^2 * 703)
- Avoid meaningless words (e.g., `lmb2`), dating (e.g., `temp05012020`) and numbering (e.g., `outcome25`) as your future self will be confused

Naming datasets and do files

- The overarching goal is always to name things so that a stranger seeing them can know what they are
- One day you will be the stranger on your own project! Make it easy on your future self!
- Choose some combination of simplicity and clarity but whatever you do, be consistent
- Avoid numbering datasets unless the numbers correspond to some meaningful thing, like randomization inference where each file is a set of coefficients and numbered according to FIPS index

Version control

- People swear by `git`, particularly Gentzkow and Shapiro
- I'm slowly learning `git` and after a long journey have managed to figure out how to use it in the command line
- Ideally your system allows you to revert to earlier versions without having ten billion files with names like `prison_03102019_sc.do`, etc.
- I do not have time to teach `git` today, but there are several useful tutorials
- You know how Dropbox/OneDrive/Google Drive, etc. saves changes as you make them to files?
- Effectively `git` lets you group how those changes are saved, so you can save explicit versions of code that you know worked and can track exactly which changes made your code break

Merge

- During a stage of arranging datasets, you will likely merge – oftentimes a lot
- Make sure you count before and after you merge so you can figure out what went wrong, if anything
- In Stata, make sure you're using the contemporary `1:1`, `1:m`, `m:1`[7] syntax as many an excellent empiricists have been hurt by merge syntax errors
- Merging is a little clunky in Stata, but R's `merge` and Python's `pandas.join` and `pandas.merge` are smoother
- Check documentation for syntax to keep observations based on if they are matches, unmatched in either, or some combination therein

[7] Avoid m:m at all costs

```
. count
  48,600

. do "/Users/scott_cunningham/Dropbox/Indy/Do/.tm-stata-55642.do"

. merge 1:1 id date using ../data/seer.dta
(note: variable month was byte, now float to accommodate using data's values)

    Result                           # of obs.
    -----------------------------------------
    not matched                       517,044
        from master                       384  (_merge==1)
        from using                    516,660  (_merge==2)

    matched                            48,216  (_merge==3)
    -----------------------------------------

. ta _merge

                _merge |      Freq.     Percent        Cum.
-----------------------+-----------------------------------
       master only (1) |        384        0.07        0.07
        using only (2) |    516,660       91.40       91.47
           matched (3) |     48,216        8.53      100.00
-----------------------+-----------------------------------
                 Total |    565,260      100.00

.
.
end of do-file

. count
  565,260
```

Appending

- Raw data files are often arranged by day, month, year, state, or some other group
- Analysis usually usually compares observations across these groups, so you need them in one dataset!
- You'll need to append these data files – `append` in Stata, `pandas.concat` in Python, `rbind` in R
- If there is an issue, it is typically because the variables are different types across the files (cannot append a string column to a numeric variable without forcing it)

Preserve/Restore Note

- Until Stata 16, Stata could not have multiple dataframes in memory and now it is still confusing functionality
- People often use `preserve`, `restore` and `tempfiles` as work arounds
- Preserve the data in that moment, do stuff to it, or open a new one, then restore back to before any of what you just did
- During that process you may save a file as a tempfile (this will be a local only usable within a session) that is deleted when you close Stata
- R and Python do not have this problem cause they can have multiple dataframes in memory at once

Long vs. Wide

- Data tables/dataframes come in two shapes, "wide" and "long"
- "wide" means there is a column for each distinct piece data point for the unit of analysis
- "long" has a column that indexes the data points for a common series
- Switching between these shapes is called "reshaping"

ID	INCOME2000	INCOME2001
1	10000	11000
2	10000	11000

ID	YEAR	INCOME
1	2000	10000
1	2001	11000
2	2000	10000

Table: Wide vs. long data tables

Reshaping

- In Stata, do `reshape long` for wide $\rightarrow$ long and `reshape wide` for long $\rightarrow$ wide
- R and Python have many reshape functions/methods each with their own time and place

```
153 ********************************************************************************
154 * Reshaping Data
155 ********************************************************************************
156
157 sysuse bpwide, clear
158 reshape long bp_, i(patient) j(timestring) s
159 rename bp_ bp
160 gen time = cond(timestring=="after",1,0)
161
162 xtset patient time
163
164 gen l_bp = l.bp // since we "xtset", lagged operators work within group (good!)
```

Figure: Inevitably reshaping is one of the hardest parts of data cleaning.

Creating New Variables

- Each language has its own way to generate new variables
- Stata:
  - `gen price2 = price^2`
  - `egen mean_price = mean(price), by(id)` – egen provides a variety of statistical methods `mode`, `median`, `mean`, etc.
- Variable creation in R depends on the "package" you use
- Python's Pandas also gives a variety of ways to create variables

Generating Variables From Aggregated Data

When you run regressions or tabulate or summarize variables, Stata stores these values and you can reference them later.

```
195  ********************************************************************************
196  * Getting Data From Memory
197  ********************************************************************************
198  sysuse auto, clear
199
200  * Generate dummy variables automatically
201  tab rep78, gen(reps)
202
203  * Getting 99% and 1% intervals to Winzorize data
204  sum price, d
205  replace price = cond(price>`r(p95)',`r(p95)',price)
206  replace price = cond(price<`r(p5)',`r(p5)',price)
207  sum price // we changed the top 3 and bottom 3 values!
208
209  * storing regression results for later use
210  reg price mpg foreign, r
211  local beta = _b[mpg]
212  local se = _se[mpg]
213  disp `beta'/`se'
```

Renaming Variables

- Sometimes you need a variable to have a more informative name
- Survey data often has names like q1423e for "question 1423 part (e)," which actually refers to price
- In Stata, `rename`
- In R, it once again depends on the package
- In Python (Pandas), there are a handful of methods

Dates & Times Variables

- Almost every language has built-in a way to deal with dates
- Stata likes to know if it is working with timestamp, days, months, quarters, etc. Each integer value is [seconds/days/months/quarters/etc] since 1/1/1960.
- Converting strings into dates: `date("1/15/08","MDY",2019) = 17546`
- Changing integers into dates: `format date %td` $>$ `17546 = 15jan2008`
- Other formats: (%tm, %td, %tc) – you aren't changing the underlying data, just how it is depicted
- Converting between formats:
  - Daily to Monthly: `gen month = mofd(date)`
  - Monthly to Quarterly: `gen quarter = qofd(dofm(month))`
- R packages `lubridate` and `zoo` provide similar functionality
- Python module `datetime` provides similar functionality

Local and Global Variables

- "locals" and "globals" are variables that store information for a function/method
- "locals" can only be referenced within the files or function where they are specified
- "globals" can be used across files and functions until the software is closed
- These are used in different ways across Stata, R, and Python
- In Stata, they can be used to save "code" as a string – not as useful for this purpose in R/Python
- Note: Stata locals are put in ``, while globals are preceded by $ – to use a local/global a string, you need to put it in ""[8]

[8] This is tedious. I have no intuition for you on it. If locals throw a bug, try futzying with the quotes.

Stata Locals/Globals

```
19  ********************************************************************************
20  * An Example of Local Variables
21  ********************************************************************************
22  local a = "a string"
23  local b = 5
24  local c price
25  local d = 2
26
27  sysuse auto, clear
28  gen outcome_2 =   price * 2
29
30  reg `c' mpg
31  reg outcome_`d' mpg
32
33  disp "This is `a'"
34  disp `b'*2
```

Figure: Can use locals in Stata to write up code.

Automate Iterative Tasks with Loops!

- Coding often involves two kinds of loops to repeat things:
  1. For loops : loop over a list in series
  2. While loops : loop through tasks until some condition is met
- The syntax is relatively similar across languages
- In Stata you can loop over variables/names with `foreach` and loop over numbers in a series with `forvalues` – the loop iterates over locals
- Loops are discouraged in R and Python:
  - R: `apply`, `lapply`, `sapply`, etc.
  - Python: List comprehensions, `map`, `apply`

For Loop

```
88  ********************************************************************************
89  * Examples of for loops
90  ********************************************************************************
91  clear all   // clearing everything from before
92  set obs 100 // starting fresh with 100 blank observations and no variables
93
94  * Looping over values
95  forvalues i = 1/5 {
96      gen var_`i' = `i'*4
97  }
98
99  forvalues j = 5(5)25 {
100     local i = `j'*5
101     gen var_`i' = `i'*15
102 }
103
104 * looping over a local list
105 local vars_to_tabulate var_1 var_3 var_5
106 foreach x of local vars_to_tabulate {
107     tab `x'
108 }
109
110 * using wildcards to simplify our loops
111
112 foreach x of varlist var_* {
113     summarize `x'
114 }
115
116 foreach x of varlist var_5-var_100 {
117     if "`x'" ~= "var_50" {
118         summarize `x'
119     }
120     else {
121         disp "var 50 we don't care about"
122     }
123 }
```

Figure: For loops in Stata

While Loop

```
*************************************************************************************
* Example while loops
*************************************************************************************

local n =1
local x =333

*Two conditions: your test condition on `x' and a counter `n' to keep the loop
*from running forever.
while `x'>.001 & `n'<1000 {
    disp `x' " after " `n' " steps"
    local x=`x'^1/3
    local n=`n'+1
}
```

Figure: While loops in Stata

Summarizing Data

Automating Tables and Figures

- Goal: make "beautiful tables" that are never edited post-production
- Large fixed costs learning commands like `estout` or `outreg2`, ut zero marginal costs
- I use `estout` because Jann has written an excellent help file at `http://repec.org/bocode/e/estout/hlp_esttab.html` but many like `outreg2` and `asdoc`
- Learn `twoway` in `Stata` or `ggplot2` in `R` or `seaborn` in `Python` and make "beautiful pictures" too
- Other great resources from Luke Stein: `https://lukestein.github.io/stata-latex-workflows/`
- The packages `stargazer` and `broom` in R do the same, `PyLaTeX` comes close

Making Tables

- General workflow:
  - Run *n* regressions (store with eststo for estout).
  - Use command esttab or asdoc to reshape them.
  - Decide whether you want to print in window or print to file.

Table made entirely in Stata

	est1	est2	est3	est4	est5	est6
smoker	-175.4**	-177.0**	-177.1**	-175.4**	-178.4**	-178.2**
	(26.83)	(27.37)	(27.01)	(26.83)	(26.69)	(27.21)
alcohol	-21.08	-19.79	-14.68	-19.60	-9.421	3.942
	(72.99)	(72.91)	(72.94)	(92.87)	(69.74)	(90.75)
nprevist	29.60**	29.75**	29.79**	29.60**	32.12**	32.09**
	(3.58)	(3.60)	(3.59)	(3.59)	(4.25)	(4.25)
unmarried	-187.1**	-189.8**	-199.5**	-187.1**	-199.1**	-206.9**
	(27.68)	(29.03)	(30.64)	(27.69)	(28.54)	(31.30)
educ		-1.875				1.828
		(5.23)				(5.54)
age			-2.460			-2.143
			(2.31)			(2.46)
drinks				-0.494		-3.027
				(14.78)		(16.43)
Trip FE	No	No	No	No	Yes	Yes
Joint F Test	56.09	44.92	45.14	44.93	33.92	23.88
Adj. R-Square	0.09	0.09	0.09	0.09	0.09	0.09
N	3000	3000	3000	3000	3000	3000

Standard errors in parentheses
Dependent Variable is Birthweight.
$^{+}$ $p < 0.10$, * $p < .05$, ** $p < .01$

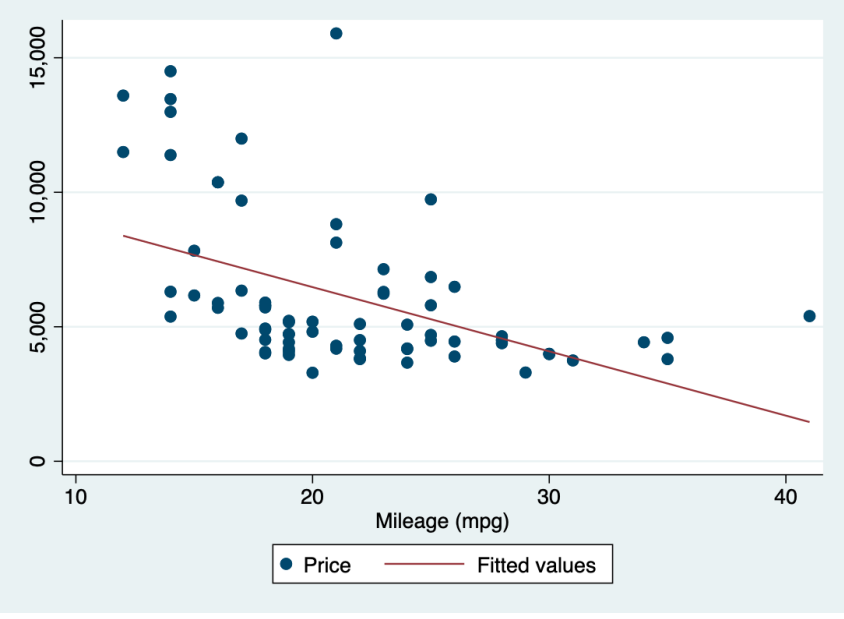
Making Figures (Without That Blue Stata Background)

- Stata graphs have annoying blue backgrounds and other bad presets
- There are two ways to get rid of those:
  - Change your presets/options with `twoway` in Stata
  - Use another language – R's `ggplot` and Python's `seaborn` both offer pros and cons relative to `twoway`
- Within Stata, the main thing to do is to change the graph region, remove grids, and give specific axes

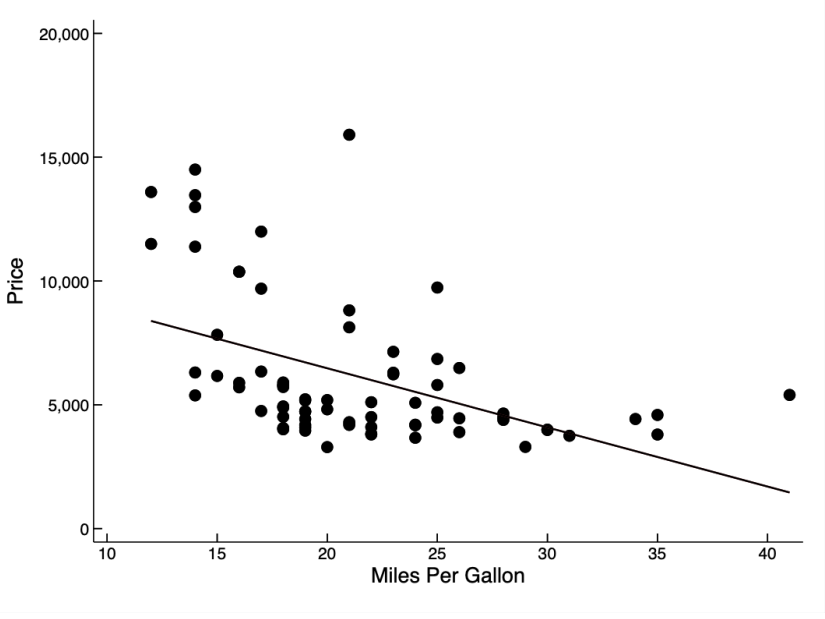
Toss that blue background

```
217  ********************************************************************************
218  * Graphs that don't suck
219  ********************************************************************************
220
221  sysuse auto, clear
222
223  twoway (scatter price mpg) (lfit price mpg)
224  graph export before.png
225
226  // I am going to have a lot of options, so I am changing new line delimiter to ;
227  #delimit ;
228  twoway  (scatter price mpg, color(black)) (lfit price mpg)
229          (lfit price mpg, lcolor(black) lpattern(solid) lwidth(medium)),
230  ytitle("Price", size(medsmall))
231  ylabel(0(5000)20000, labsize(small) angle(0) tposition(inside) nogrid)
232  ytick(0(5000)20000, nolabels tposition(inside) nolabels nogrid)
233  xtitle("Miles Per Gallon")
234  xlabel(10(5)40, labsize(small) tposition(inside))
235  xtick(10(5)40, nolabels tposition(inside) nolabels nogrid)
236  graphregion(color(white)) plotregion(color(white))
237  legend(off);
238  # delimit cr
239  graph export after.png
```

Much cleaner



(a) Before



(b) After

Many languages: which to choose?

- I've brought up several languages today
- Which should you pick?
- Two answers:
  1. The one that makes the most sense right now
  2. Whichever fits the task at hand
- With experience you'll get better at picking up other languages
- You'll also learn how to determine which language is the best for a problem
- Stata is often the easiest for the work you do in an economics class
- Python/R are the best for being more employable

Stata: Overview of Use

- Stata is essentially a very powerful tool to edit a single spreadsheet of data at a time.
  - Syntax *explicitly written* for analyzing a dataset
  - Tools for: matrix calculcations, variables in memory, programs[9]
- Pro: Battle-tested for econometrics and has regular quality control by StataCorp
- Con: Not used much outside academia and has awkward syntax when you want to look at multiple spreadsheets
- Mixed bag: user-written programs available for `ssc install`
- Shortcuts:
  - New *.do* file: ctrl-N or cmd-N or (legacy) cmd-9
  - Data editor: *browse* or **almost never** *edit*
  - Run (selection) of *.do* file: (highlight text) cmd-shift-d

[9] Use only if you need them

Warnings and Quirks

Familiar problems arise in Stata. Make sure to google error codes when you get them. Here are some common errors to know:

- "no; dataset in memory has changed since last saved"
  - You need to clear the data you have since Stata will not overwrite it.
- "variable [x] does not uniquely identify observations in the [master/using] data."
  - Your merge is messed up because you have repeated values of variables when trying to merge.
- "not sorted"
  - For some reason if your data are not sorted you can't use by. As a solution just always use bysort.
- The missing value comparison curse!
  - **BEWARE:** missing values will evaluate in Stata as if they are infinity. See example.

Other Useful Tools

Stata can be clunky with bigger data. Here are a handful of useful tools to try:

- `Gtools` by Mauricio Caceres
  `https://gtools.readthedocs.io/en/latest/index.html`
  largely replaces egen, called `gegen`. Has two-step install:
  1. `ssc install gtools`
  2. `gtools, upgrade`
- `ftools` by Sergio Correia
  `https://github.com/sergiocorreia/ftools` - also similar to egen, but called `fegen`. `fmerge` is faster than merge
- With regressions that have many fixed effects use `areg`, `xtset`, or http://scorreia.com/software/reghdfe/ (by Sergio Correia)
- Use strLs to reduce the memory:
  - `recast strL stringvar`
  - `gen strL stringvar = "string"`

R: Overview of use

1. Everything is an object.
2. Everything has a name.
3. You do things using functions.
4. Functions come pre-written in packages (i.e. "libraries"), although you can — and should — write your own functions too.

Points 1. and 2. can be summarised as an https://en.wikipedia.org/wiki/Object-oriented_programming (OOP) approach.

R vs. Stata

If you're coming from Stata, some additional things:

- Multiple objects (e.g. data frames) can exist happily in the same workspace.
  - No more `keep, preserve, restore` hackery. (https://www.stata.com/new-in-stata/multiple-datasets-in-memory/ added a fix).
  - This is a direct consequence of the OOP approach.
- Load packages at the start of every new R session. Make peace with this.
  - "Base" R comes with tons of useful in-built functions and the tools to write your own functions.
  - However, many of R's best data science functions and tools come from external packages written by users (`tidyverse`, `data.table`, `feols`) that you include with `install.packages()`
- R easily and infnitely parallelizes. For free.
- You need a https://www.stata.com/statamp/ license to parallelize and you pay per core!
- Check `https://stata2r.github.io/` for translation tips!

Python: Overview of use

- Python is an incredible useful tool for doing complex data tasks with data of any size
- The main modules (i.e. packages) for data analysis are `pandas` and `numpy`
- Again everything is an object
- Pro: For bigger data tasks, it is much easier to parallelize tasks
- Con: Not all modules are stable and object-oriented programming makes for clunkier coding
- Mixed: Documentation sometimes seems written with data analysis as an afterthought – makes for frustrating work
- In contrast to Stata and R to some extent, you have a lot more control over how the sausage gets made

Before showing you how to program, how do you download Python?

- There are two approaches: pip installation (command line) and Anaconda (command line or point and click)
- Both allow you to create "environments" of packages that are self-contained
- This can be useful because user-written packages can have conflicting dependencies that are hard to track
- Stata avoids this by being a company
- R also has environments you can use with Renv, but it does not lead with those
- Can use Anaconda or pip

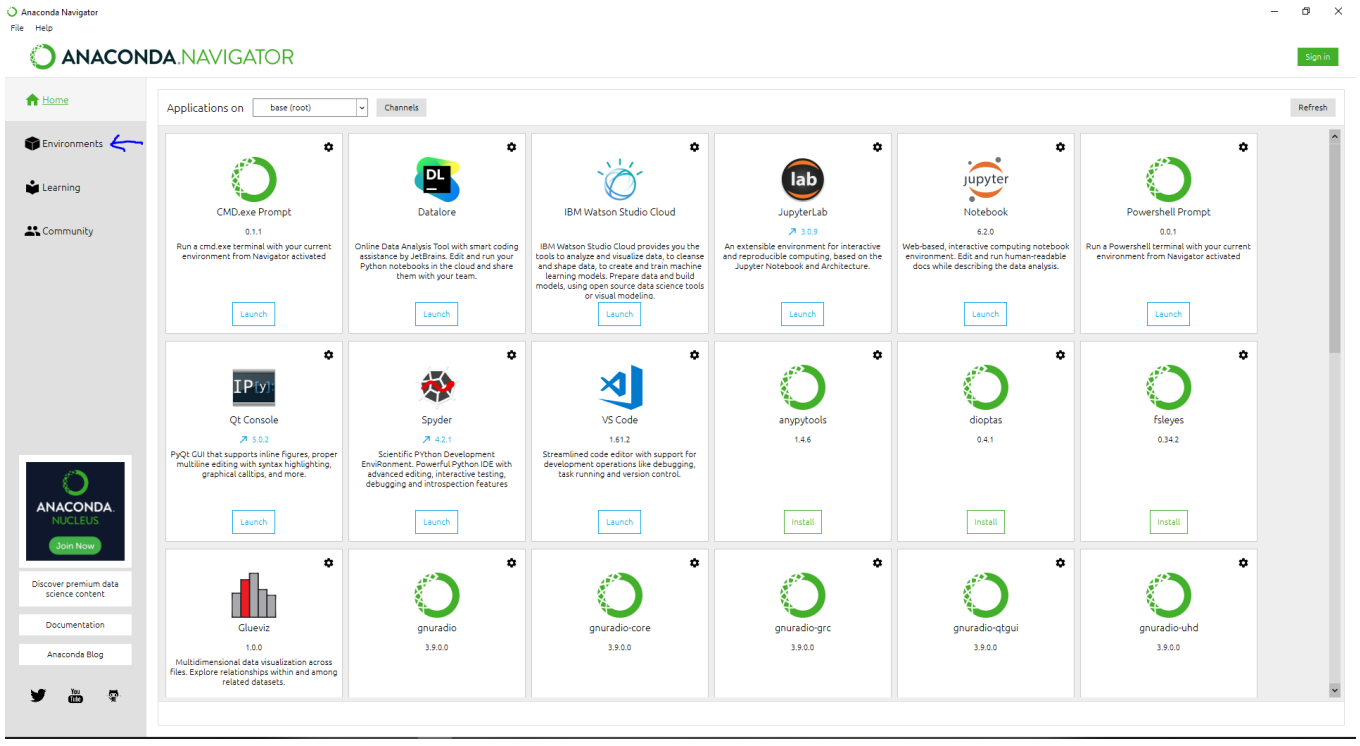


Figure: The anaconda home page. Go to environments to install packages

- Python packages are called modules and live in repositories on the internet
- Communities of programmers vet these repositories
- You have to first install them on your computer, then you use them in a program with the syntax 'import moduleX'
- With Anaconda Navigator, it is pretty straight-forward to find an install packages
- You will mostly use `pandas`, `numpy`, `requests`, `zipfile`, `matplotlib`, `seaborn`, or `geopandas` – all of which are in Anaconda
- Other packages may exist somewhere else and require some work to access

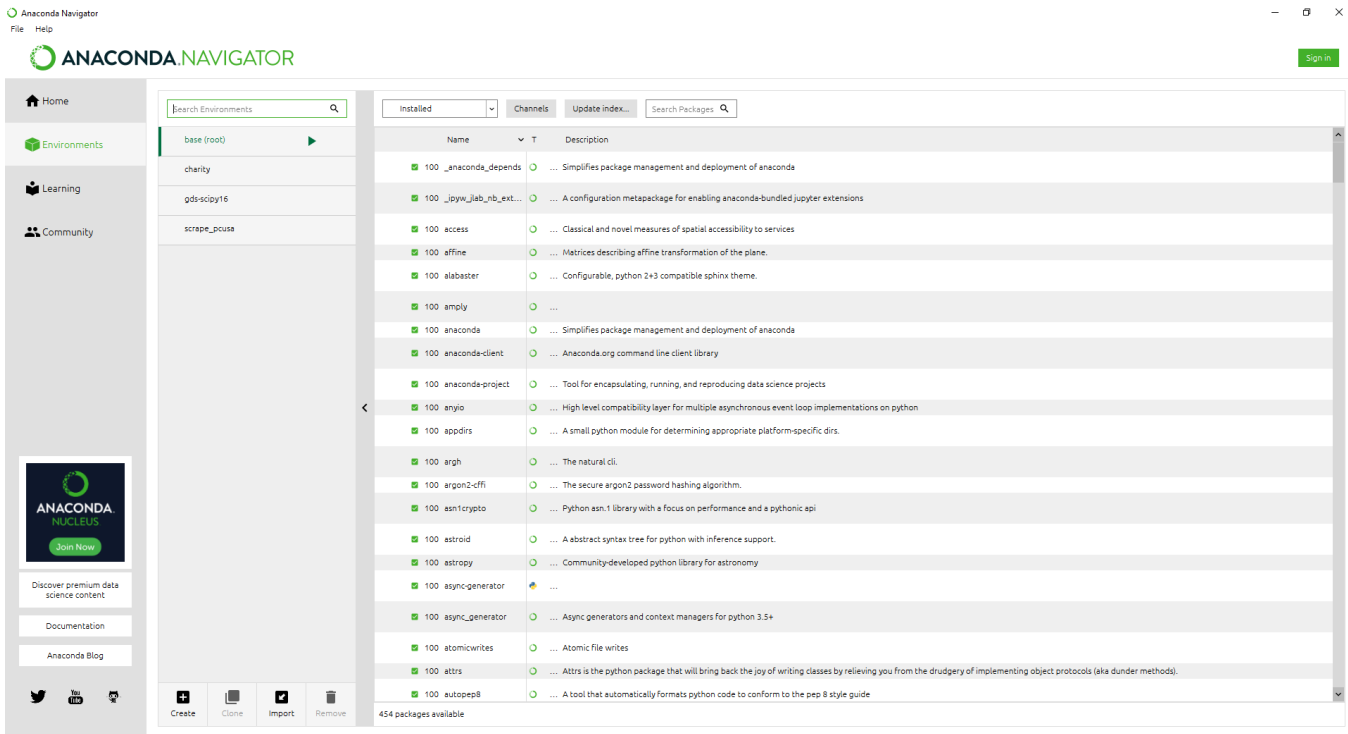


Figure: Install packages

Welcome to the Wonderful, Frustrating World of Coding

- Covered a lot today, but here are the big takeaways:
  - Organize your folders, files, and code to be understandable first, fast second – do not compromise on this
  - Always have a checklist of tasks before you dive into analysis
  - Clearly state the problem you want to solve in words and then work backwards from that to code
  - Actually look at your data before and after you run code to make sure your work does what you think it does!
  - The coding learning curve flattens the more exposure you have to coding, so just start learning!

Office Hours

Please come to my office hours in-person or online[10]:

- Where: On Zoom or Room 1006A
- When:
  - Mondays, Wednesdays, Thursdays from 3-4pm
  - Tuesdays from 10:30-11:30am
- What: Coding help, strategize research project, dataset tips
- Why: Best way for me to assist you is to see what you're working with

[10] I will be in Kenya March 21-March 31, so OH during that period will be moved around so I can do them remotely at a reasonable hour for both of us