# Basics of programming

Mauricio Romero

## Basics of programming

For loops
(Based on notes by Nick C. Huntington-Klein)

Conditional statements

Functions

Hidden curriculum: Things no one teaches you, but you should know
(Based on Scott Cunningham's notes)

## Basics of programming

For loops
(Based on notes by Nick C. Huntington-Klein)

Conditional statements

Functions

Hidden curriculum: Things no one teaches you, but you should know
(Based on Scott Cunningham's notes)

## For loops

- Sometimes we want to subset things in many different ways
- Typing everything out over and over is a waste of time!
- You don’t understand how powerful computers are until you’ve written a ‘for‘ loop
- This is an incredibly standard programming tool
- R has another way of writing loops using the ‘apply‘ family, but we’re not going to go there

## For loops

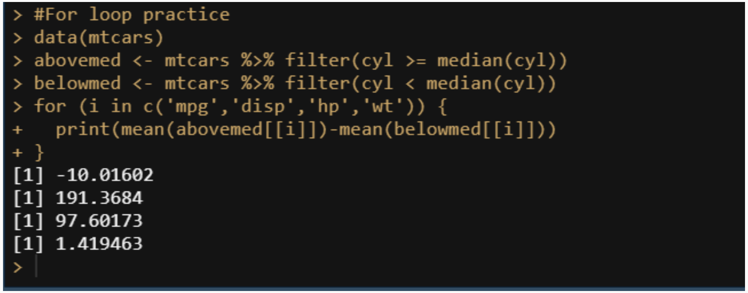
- Basic idea: you have a vector of values, and you have an "iterator" variable
- You go through the vector, setting the iterator variable to each value one at a time
- Then you run a chunk of that code with the iterator variable set

```
for (iteratorvariable in vector) {
  code chunk
}
```

## For loops

- Let's rewrite our 'hp' and 'mpg' differences with a for loop. Heck, let's do a bunch more variables too. (don't forget to get 'data(mtcars)' again - why?)
- Note if we want it to display results inside a loop we need 'print()'
- Also if we're looping over variables, '$' won't work - we need to use '[[]]'

```
data(mtcars)
abovemed <- mtcars %>% filter(cyl >= median(cyl))
belowmed <- mtcars %>% filter(cyl < median(cyl))
for (i in c('mpg','disp','hp','wt')) {
  print(mean(abovemed[[i]])-mean(belowmed[[i]]))
}
```



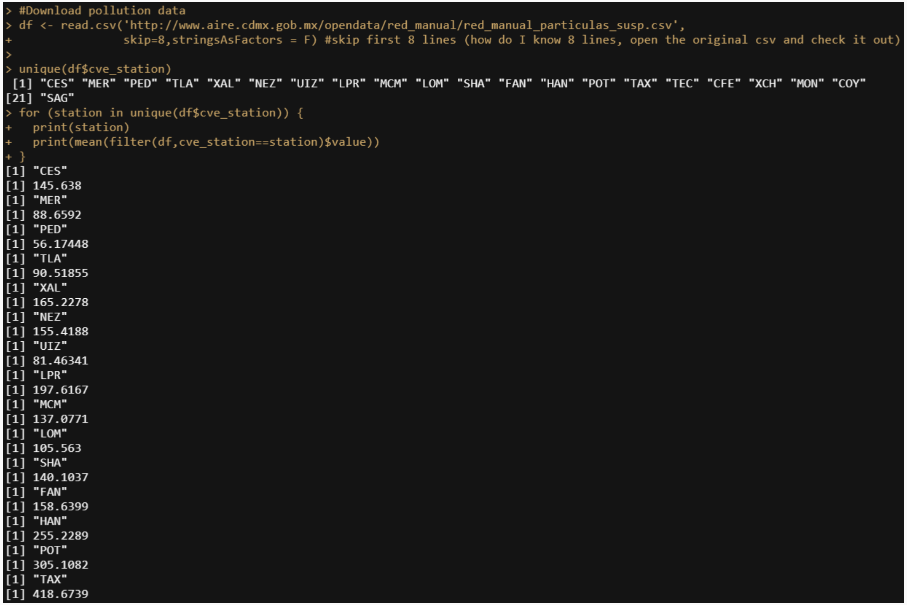
```
> #For loop practice
> data(mtcars)
> abovemed <- mtcars %>% filter(cyl >= median(cyl))
> belowmed <- mtcars %>% filter(cyl < median(cyl))
> for (i in c('mpg','disp','hp','wt')) {
+   print(mean(abovemed[[i]])-mean(belowmed[[i]]))
+ }
[1] -10.01602
[1] 191.3684
[1] 97.60173
[1] 1.419463
> 
```

## For loops

- It's also sometimes useful to loop over different values
- Let's get the average pollution by station
- ‘unique()‘ can give us the levels to loop over

```
df <- read.csv(
'http://www.aire.cdmx.gob.mx/opendata/red_manual/red_manual_particulas_susp.csv',
skip=8,stringsAsFactors = F)

unique(filter(df,cve_parameter=="PM10")$cve_station)
for (station in unique(filter(df,cve_parameter=="PM10")$cve_station)) {
  print(station)
  print(mean(filter(df,cve_station==station,cve_parameter=="PM10")$value))
}
```



```
> #Download pollution data
> df <- read.csv('http://www.aire.cdmx.gob.mx/opendata/red_manual/red_manual_particulas_susp.csv',
+               skip=8,stringsAsFactors = F) #skip first 8 lines (how do I know 8 lines, open the original csv and check it out)
>
> unique(df$cve_station)
 [1] "CES" "MER" "PED" "TLA" "XAL" "NEZ" "UIZ" "LPR" "MCM" "LOM" "SHA" "FAN" "HAN" "POT" "TAX" "TEC" "CFE" "XCH" "MON" "COY"
[21] "SAG"
> for (station in unique(df$cve_station)) {
+   print(station)
+   print(mean(filter(df,cve_station==station)$value))
+ }
[1] "CES"
[1] 145.638
[1] "MER"
[1] 88.6592
[1] "PED"
[1] 56.17448
[1] "TLA"
[1] 90.51855
[1] "XAL"
[1] 165.2278
[1] "NEZ"
[1] 155.4188
[1] "UIZ"
[1] 81.46341
[1] "LPR"
[1] 197.6167
[1] "MCM"
[1] 137.0771
[1] "LOM"
[1] 105.563
[1] "SHA"
[1] 140.1037
[1] "FAN"
[1] 158.6399
[1] "HAN"
[1] 255.2289
[1] "POT"
[1] 305.1082
[1] "TAX"
[1] 418.6739
```

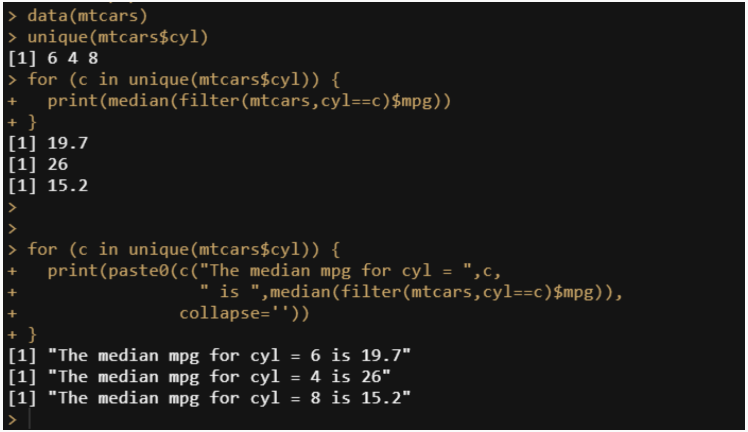
## For loops practice

- Get back the full ‘mtcars‘ again
- Use ‘unique()‘ to see the different values of ‘cyl‘
- Use ‘unique‘ to loop over ‘cyl‘ values and get median ‘mpg‘ for each level of ‘cyl‘
- Use ‘paste0‘ to print out “The median mpg for cyl = # is X” where # is the iterator number and X is the answer.

## For loops practice answers

```
data(mtcars)
unique(mtcars$cyl)
for (c in unique(mtcars$cyl)) {
  print(median(filter(mtcars,cyl==c)$mpg))
}
```

```
for (c in unique(mtcars$cyl)) {
  print(paste0(c("The median mpg for cyl = ",c,
                 " is ",median(filter(mtcars,cyl==c)$mpg)),
                collapse=''))
}
```



```
> data(mtcars)
> unique(mtcars$cyl)
[1] 6 4 8
> for (c in unique(mtcars$cyl)) {
+   print(median(filter(mtcars,cyl==c)$mpg))
+ }
[1] 19.7
[1] 26
[1] 15.2
>
>
> for (c in unique(mtcars$cyl)) {
+   print(paste0(c("The median mpg for cyl = ",c,
+                " is ",median(filter(mtcars,cyl==c)$mpg)),
+              collapse=''))
+ }
[1] "The median mpg for cyl = 6 is 19.7"
[1] "The median mpg for cyl = 4 is 26"
[1] "The median mpg for cyl = 8 is 15.2"
>
```

## Basics of programming

For loops
(Based on notes by Nick C. Huntington-Klein)

Conditional statements

Functions

Hidden curriculum: Things no one teaches you, but you should know
(Based on Scott Cunningham's notes)

## Basics of programming

## For loops practice answers

- Sometimes, you want R to do something only in some cases
- The basic syntax to do this is an “if” statement

```
if (test_expression) {
statement
}
```

- The “test_expression” usually depends on logical operators
  - ‘&‘ is AND
  - ‘|‘ is OR
  - To check equality use ‘==‘, not ‘=‘
  - ‘>=‘ is greater than OR equal to, similarly for ‘<=‘

## For loops practice answers

- Sometimes is useful to “break” (stop) a loop, or skip an iteration
- Do this with “break” and “next”
- Often “break” and “next” are nested within an “if” statement

## Example: for loop combined with if statement and next

```
#printing odd numbers
m=20
for (k in 1:m){
  if (!k %% 2)
    next
    print(k)
}
```

## Example: for loop combined with if statement and break

```
#finding the first number divisible by 13 greater than 100
for (k in 100:100000){
  if (!k %% 13){ #checks if number is odd
    break
  }
}
print(k)
```

## Example: finding prime numbers

```
for(num in 1:100){
  # Program to check if the input number is prime or not
  flag = 0
  # prime numbers are greater than 1
  if(num > 1) {
    # check for factors
    flag = 1
    for(i in 2:(num-1)) {
      if ((num %% i) == 0) {
        flag = 0 #if number is divisible, then not prime
        break #and we can break the loop
      }
    }
  }
  if(num == 2)    flag = 1
  if(flag == 1) {
    print(paste(num,"is a prime number"))
  } else {
    print(paste(num,"is not a prime number"))
  }
}
```

## Basics of programming

For loops
(Based on notes by Nick C. Huntington-Klein)

Conditional statements

Functions

Hidden curriculum: Things no one teaches you, but you should know
(Based on Scott Cunningham's notes)

## Basics of programming

For loops
(Based on notes by Nick C. Huntington-Klein)

Conditional statements

Functions

Hidden curriculum: Things no one teaches you, but you should know
(Based on Scott Cunningham's notes)

## Functions

- Functions are used when you have to repeat the same operation multiple times
- Avoids coding mistakes (and if there is one you only need to fix it once)
- Let's you break code into simpler parts which become easy to maintain and understand
- It's pretty straightforward to create your own function in R programming.

```
func_name <- function (argument) {
statement
return(whatyouwant)
}
```

## Example: finding prime numbers

```
# Identifying prime number
prime_num<-function(num){
  # Program to check if the input number is prime or not
  flag = 0
  # prime numbers are greater than 1
  if(num > 1) {
    # check for factors
    flag = 1
    for(i in 2:(num-1)) {
      if ((num %% i) == 0) {
        flag = 0 #if number is divisible, then not prime
        break #and we can break the loop
      }
    }
  }
  if(num == 2)    flag = 1
  return(flag)
}
```

## Example: finding prime numbers

```
prime_num(2)
prime_num(3)
prime_num(4)
prime_num(6131)
prime_num(4684561123)
```

## Functions

- Functions can have multiple arguments
- Functions can have default values for arguments
- Functions can output vectors, list, or any other object

## Example: power function

```r
pow <- function(x, y = 2) {
  result1 <- x^y
  return(result1)
}
pow(3)
pow(3,3)
```

## Basics of programming

For loops
(Based on notes by Nick C. Huntington-Klein)

Conditional statements

Functions

Hidden curriculum: Things no one teaches you, but you should know
(Based on Scott Cunningham's notes)

## Basics of programming

For loops
(Based on notes by Nick C. Huntington-Klein)

Conditional statements

Functions

Hidden curriculum: Things no one teaches you, but you should know
(Based on Scott Cunningham's notes)

## If you are going to take away one thing...take away this

"A rule of research is that you will end up running every step more times than you think. And the costs of repeated manual steps quickly accumulate beyond the costs of investing once in a reusable tool."
Gentzkow and Shapiro, 2014

## Some basic guiding principles

- Automate everything that can be automated.
- Design your workflow so that when you add new data or tweak something you don't need to manually recreate anything
- Avoid copying and pasting figures/tables from one software into another
    - e.g., impossible to automate: copying a graph from R into Word
- General workflow: raw data $\rightarrow$ clean data $\rightarrow$ analysis $\rightarrow$ (pretty) table or figure

## Basics of programming

For loops

(Based on notes by Nick C. Huntington-Klein)

Conditional statements

Functions

Hidden curriculum: Things no one teaches you, but you should know

(Based on Scott Cunningham's notes)

- Workflow workflow workflow
- Folder structure (also based on Sharnic Djaker and Alejandro Ganimian's notes)
- Naming conventions
- R programs (and other programming scripts)
- Data Visualization (also based on J-PAL data visualization guide)
- Soft skills

"Happy families are all alike; every unhappy family is unhappy in its own way."
Leo Tolstoy

"Happy families are all alike; every unhappy family is unhappy in its own way."
Leo Tolstoy

"Good empirical work is all alike; every bad empirical work is bad in its own way."
Scott Cunningham

## Empirical workflow

- Fixed set of routines you always do to identify most common errors

- Think of it as your morning routine: alarm goes off, shower, coffee, check Twitter,...

## Why do we use checklists?

- Before a trip: checklist to make sure you have everything you need
  - Charger (check)
  - Underwear (check)
  - Toothbrush (check)
  - Passport (oops)
- Empirical checklist: step between “getting the data” and “analyzing the data”
- Focuses on ensuring data quality for the most common situations

## Simple checks

- Checklist should be a few simple, yet non-negotiable, exercises to check for errors
- We will cover some ideas (but you can add more)

## Time

- People often think empirical research is about “getting data” and “analyzing data”

## Time

- People often think empirical research is about “getting data” and “analyzing data”
- If only....

## Time

- People often think empirical research is about “getting data” and “analyzing data”
- If only....
- 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 (cleaning the data)

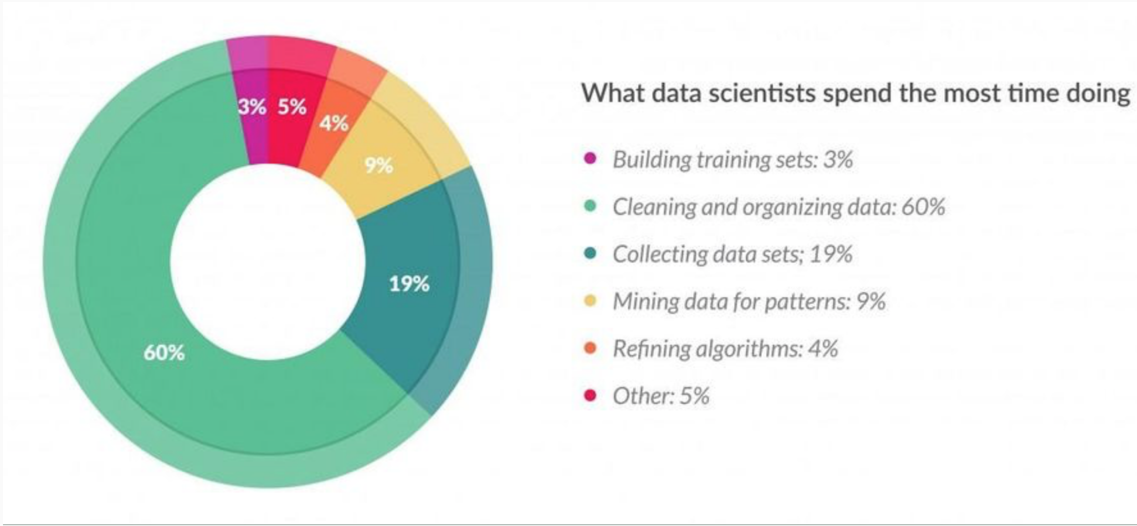


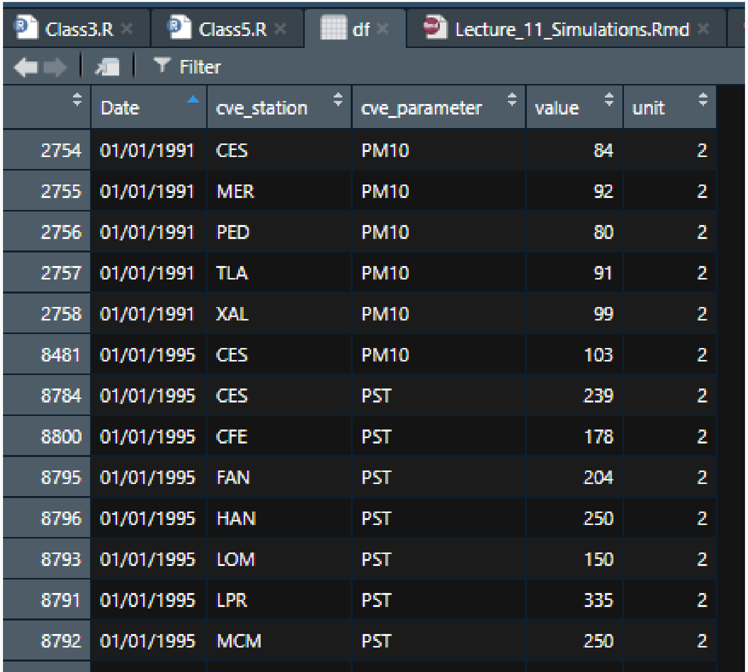
Image from Wenfei Xu at Columbia

## Always read the codebook

- We stand on the shoulders of giants
- Few like reading the codebook (tends to be tedious and boring)
- But the codebook explains how to interpret the data you have acquired
    - THIS IS NOT A STEP YOU CAN SKIP
- Set aside time to study it, and keep it so you can regularly return to it
- This goes for the `readme` that accompanies some datasets, too.

## Don’t forget to look at the data

- The eyeball is not nearly appreciated enough for its ability to spot problems
- Use `the R studio browser` or excel
- Scroll through the variables and accompany yourself with what you’ve got visually



| | Date | cve_station | cve_parameter | value | unit |
|---|---|---|---|---|---|
| 2754 | 01/01/1991 | CES | PM10 | 84 | 2 |
| 2755 | 01/01/1991 | MER | PM10 | 92 | 2 |
| 2756 | 01/01/1991 | PED | PM10 | 80 | 2 |
| 2757 | 01/01/1991 | TLA | PM10 | 91 | 2 |
| 2758 | 01/01/1991 | XAL | PM10 | 99 | 2 |
| 8481 | 01/01/1995 | CES | PM10 | 103 | 2 |
| 8784 | 01/01/1995 | CES | PST | 239 | 2 |
| 8800 | 01/01/1995 | CFE | PST | 178 | 2 |
| 8795 | 01/01/1995 | FAN | PST | 204 | 2 |
| 8796 | 01/01/1995 | HAN | PST | 250 | 2 |
| 8793 | 01/01/1995 | LOM | PST | 150 | 2 |
| 8791 | 01/01/1995 | LPR | PST | 335 | 2 |
| 8792 | 01/01/1995 | MCM | PST | 250 | 2 |

## Look for missing observations

- Check the size of your dataset using `dim`
- Check the structure of your data using `str`
- Check for missing variables
    - Sometimes the data has -99 for missing
    - Others is truly missing (i.e., NA)
    - Others is an empty string

## Check different layers

- Data often comes in layers
    - Many observations for the same unit across time
    - Many observations per state/country/firm/household
- Start with `aggregating` the data to some manageable level and `list/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?
    - If pollution for some station is much higher than for others on the same date

## Always check your merges (i.e., combining data sets)

- 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
- Also make sure you understand what type of merge you are using/need
    - many to one
    - one to many
    - one to one
    - many to many (please no!!!)

## Good practices

- Beyond checking the data, here are a few things that will prevent errors
  1. Organized subdirectories
  2. Automation
  3. Naming conventions
  4. Version control
- I'll discuss each but 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`

## Basics of programming

For loops

(Based on notes by Nick C. Huntington-Klein)

Conditional statements

Functions

Hidden curriculum: Things no one teaches you, but you should know

(Based on Scott Cunningham's notes)

- Workflow workflow workflow
- Folder structure (also based on Sharnic Djaker and Alejandro Ganimian's notes)
- Naming conventions
- R programs (and other programming scripts)
- Data Visualization (also based on J-PAL data visualization guide)
- Soft skills

## No correct organization

- There is no correct way to organize your directories
- But below are some organizing principles I suggest
- Always remember: your future self is lazy, distracted, disinterested and busy
- Make things easy for your future self

## Suggested structure

- The parent project folder should have 7 sub-folders

  1. **Admin** for all admin-related issues (e.g., IRB applications, contracts, memos, etc.)
  2. **Data collection** (e.g., sampling protocol, instruments, field plan, etc.)
  3. **Data storage** for all raw, temp, and clean datasets
  4. **Data codes** for setup/cleaning and analysis
  5. **Writeup** for manuscripts and presentations
  6. **Articles** for all literature you cite/use
  7. **Miscellaneous** for all the files that cannot be placed above

## The “Admin” folder: All administrative related documents go here

- This folder should include the following sub-folders:
    - External – all external facing documents is stored here (e.g., donor communications and website content)
    - Internal – includes internal memos, reimbursements, and IRB

## The “Data collection” folder (in case you collect your own data)

- This folder should include one sub-folder for each round of data collection
- Each sub-folder should include at least two sub-folders:
  - Instruments
  - Fieldwork (field plan, training materials, etc.)

## The “Data storage” folder

- Should include at least three sub-folders:
  - Raw (data **as you get it**: more on this next)
  - Created (you should be able to delete this folder and re-create its content using your scripts)

## The “Data storage” folder

- Should include at least three sub-folders:
    - Raw (data **as you get it**: more on this next)
    - Created (you should be able to delete this folder and re-create its content using your scripts)

**ALL (and ONLY) data files sit in this folder.**

## The “Data storage” folder

- Should include at least three sub-folders:
    - Raw (data **as you get it**: more on this next)
    - Created (you should be able to delete this folder and re-create its content using your scripts)

**ALL (and ONLY) data files sit in this folder.**

**Programming scripts are NEVER saved here.**

## The “Raw” folder

- Should have data **as you get it**
- Add a readme file that explains how you got the data
- Should have documentation for the data as well (e.g., codebooks)
- If there are different sources, use subfolder for each source
- You should never **NEVER EVER** save data you have already edited/manipulated
- No! Don’t save it here even if just manipulated it a tiny bit

## The “Data codes” folder

- May include some sub-folders (depending on needs)
    - Programs
        - When you write little functions for things you do over and over again
        - For example, I have a little program to format tables/figures as I like them
    - Folder for each language you use
        - Sometimes you will use multiple programming languages (e.g., R, Stata, Python)
        - Keep one folder per programming language

## The “Data codes” folder

- May include some sub-folders (depending on needs)
    - Programs
        - When you write little functions for things you do over and over again
        - For example, I have a little program to format tables/figures as I like them
    - Folder for each language you use
        - Sometimes you will use multiple programming languages (e.g., R, Stata, Python)
        - Keep one folder per programming language

**ALL (and ONLY) codes sit in this folder**

## The “Data codes” folder

- May include some sub-folders (depending on needs)
    - Programs
        - When you write little functions for things you do over and over again
        - For example, I have a little program to format tables/figures as I like them
    - Folder for each language you use
        - Sometimes you will use multiple programming languages (e.g., R, Stata, Python)
        - Keep one folder per programming language

**ALL (and ONLY) codes sit in this folder**

**Data is NEVER saved here**

## The “Writeup” folder

Include the following sub-folders:

- Manuscripts and presentations
  - Include 3 subfolders
    1. Archive: to put old versions
    2. Tables: to put all tables outputted by Stata/R/etc.
    3. Figures: to put all figures outputted by Stata/R/etc.
- Comments
  - For all comments received on the manuscript
  - To be saved under the name of the person providing the comments + date
  - e.g. “Pepito Perez – 2020-07-03.docx”
- To do list (for a running list of pending things to do)

## The “Miscellaneous” folder

- This folder should ideally have nothing in it.
    - If you really, really, really, cannot figure out an appropriate location of a file/folder you can add it here
    - Please place all files inside folders, and make sure folder names are intuitive

## Important reminders

- Folders should only include the latest version of a file
    - All previous versions should be saved in a "Archive" sub-folder with the date of the version on the name of the file (file_20191203 instead of file_v2)
- Data/scripts related folder names and files **SHOULD NOT** have spaces, use underscores instead (e.g. "data_collection" instead of "data collection").

## Basics of programming

## Different elements

- Things to keep in mind when naming:
    1. variables,
    2. datasets
    3. scripts
- Avoid
    - meaningless words (e.g., `lmb2`)
    - dating (e.g., `temp05012020`), except to save old version in the archive
    - numbering (e.g., `outcome25`)
    - any of these will confuse your future self

## Naming conventions for variables

- Variables should be readable to a stranger
    - Say that you want to create the product of two variables. Use an underscore and mash the two together
    - price_mpg <- price * mpg
- Name the variable exactly what it is when possible
    - bmi <- weight / (height*height * 703)

## Naming datasets and do files

- The goal is always to name things so that a stranger can know what they are
- One day you will be the stranger on your own project!
- Choose some combination of simplicity and clarity and be consistent
- Do number scripts in the order in which they should be used
- Do not number datasets unless the numbers correspond to some meaningful thing

## Basics of programming

For loops

(Based on notes by Nick C. Huntington-Klein)

Conditional statements

Functions

Hidden curriculum: Things no one teaches you, but you should know

(Based on Scott Cunningham's notes)

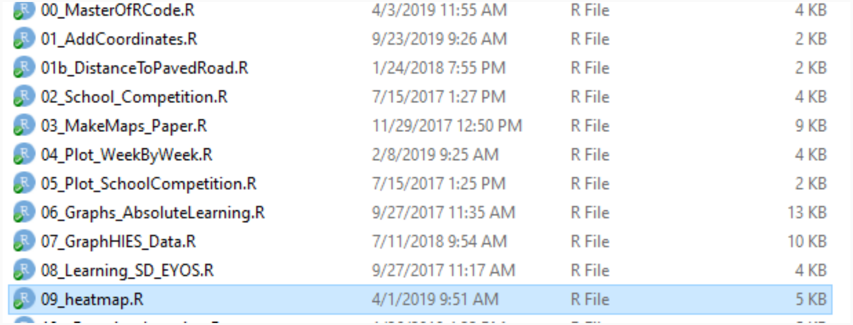
- Workflow workflow workflow
- Folder structure (also based on Sharnic Djaker and Alejandro Ganimian's notes)
- Naming conventions
- R programs (and other programming scripts)
- Data Visualization (also based on J-PAL data visualization guide)
- Soft skills

## Always use scripting programs NOT GUI

- Your future self doesn't even remember making script files, tables or figures
- Much less, how to do things in the GUI
- Throw yourself a bone, and walk your future self exactly through everything
- Which means you've got to have replicable scripting files

## Naming files

- You should have a master file that sets the path, calls the libraries, etc.
- Then you should have scripts that clean the data
- Then scripts that “analyze” the data (and produce tables/figures)
- I use a numbering structure that tells me what comes first (if I sort them by name)



| | | | |
|---|---|---|---|
| 00_MasterOfRCode.R | 4/3/2019 11:55 AM | R File | 4 KB |
| 01_AddCoordinates.R | 9/23/2019 9:26 AM | R File | 2 KB |
| 01b_DistanceToPavedRoad.R | 1/24/2018 7:55 PM | R File | 2 KB |
| 02_School_Competition.R | 7/15/2017 1:27 PM | R File | 4 KB |
| 03_MakeMaps_Paper.R | 11/29/2017 12:50 PM | R File | 9 KB |
| 04_Plot_WeekByWeek.R | 2/8/2019 9:25 AM | R File | 4 KB |
| 05_Plot_SchoolCompetition.R | 7/15/2017 1:25 PM | R File | 2 KB |
| 06_Graphs_AbsoluteLearning.R | 9/27/2017 11:35 AM | R File | 13 KB |
| 07_GraphHIES_Data.R | 7/11/2018 9:54 AM | R File | 10 KB |
| 08_Learning_SD_EYOS.R | 9/27/2017 11:17 AM | R File | 4 KB |
| 09_heatmap.R | 4/1/2019 9:51 AM | R File | 5 KB |

## The master file

- Installs/Loads packages
- Sets the working directory
- Call all other files
- If you run the master, you should replicate all your data cleaning, analysis, results

## Master: Install packages

```
rm(list=ls())
list.of.packages <- c("rstan","foreign")
new.packages <- list.of.packages[!(list.of.packages %in% installed.packages()[,"Package"])]
if(length(new.packages)) install.packages(new.packages,dependencies=TRUE)
```

## Master: load packages

```
library(rstan)
library(foreign)
```

## Master: set folder structure

```
if(Sys.info()["user"]=="Mauricio") {
        setwd("C:/Users/Mauricio/Dropbox/Research/Liberia PSL/")
}
if(Sys.info()["user"]=="MROMEROLO"){
        setwd("C:/Users/MROMEROLO/Dropbox/Research/Liberia PSL/")
}
```

## Master: call other scripts (in order!)

```
#ADDS COORDINATES TO THE SCHOOL LIST
source("Analysis/code/RCode/01_AddCoordinates.R")
#ESTIMATES DISTANCE TO THE NEAREST PAVED ROAD
source("Analysis/code/RCode/01b_DistanceToPavedRoad.R")
#FINDS THE NUMBER OF NEARBY SCHOOLS
source("Analysis/code/RCode/02_School_Competition.R")
#MAKES THE MAPS WE SHOW IN THE PAPER
source("Analysis/code/RCode/03_MakeMaps_Paper.R")
```

## Good text editor

- Remember: the goal is to make beautiful programs
- Stata and Rstudio come with built-in text editors, which use slick colors for various types of programming commands
- Some people prefer to invest in a good text editor which has bundling capabilities that will integrate with Stata, R or LaTeX
    - Mac users sometimes prefer Textmate 2
    - PC users sometimes prefer Sublime
    - I just use the Rstudio/Stata buil-in editors

## Headers

- Always start your programming scripts with a header
- The header is a big comment on the code and says (at a minimum):
  - The purpose of the script
  - The author(s)
  - The date the script was created
  - The last date in which it was updated/modified (and by whom)

## Be VERY generous with your comments and speak clearly

- Your future self is time constrained, so explain *everything* with comments
- Even if something seems obvious now, it will likely not be obvious in the future
- Time used to document your programs is never wasted

## Version control

- You want to avoid having multiple versions of the same file
- You also want to save old versions in case you decide to undo some change
- Version control software lets you do that (I use git)
- Dropbox also has some version history (but I don't fully trust it)
- At least keep only the current version of the file in the “main folder” and old versions in an “archive” folder

## Basics of programming

For loops

(Based on notes by Nick C. Huntington-Klein)

Conditional statements

Functions

Hidden curriculum: Things no one teaches you, but you should know

(Based on Scott Cunningham's notes)

- Workflow workflow workflow
- Folder structure (also based on Sharnic Djaker and Alejandro Ganimian's notes)
- Naming conventions
- R programs (and other programming scripts)
- Data Visualization (also based on J-PAL data visualization guide)
- Soft skills

## Automating Tables and Figures (more on visualization below)

- Goal is to make “beautiful tables/figures” that are never edited post-production
- Tables/figures should be readable on their own (with the table/figure notes)
- Large fixed costs learning commands like “stargazer” in R or “estout” in R (but incur them because marginal costs are zero)

## LaTeX

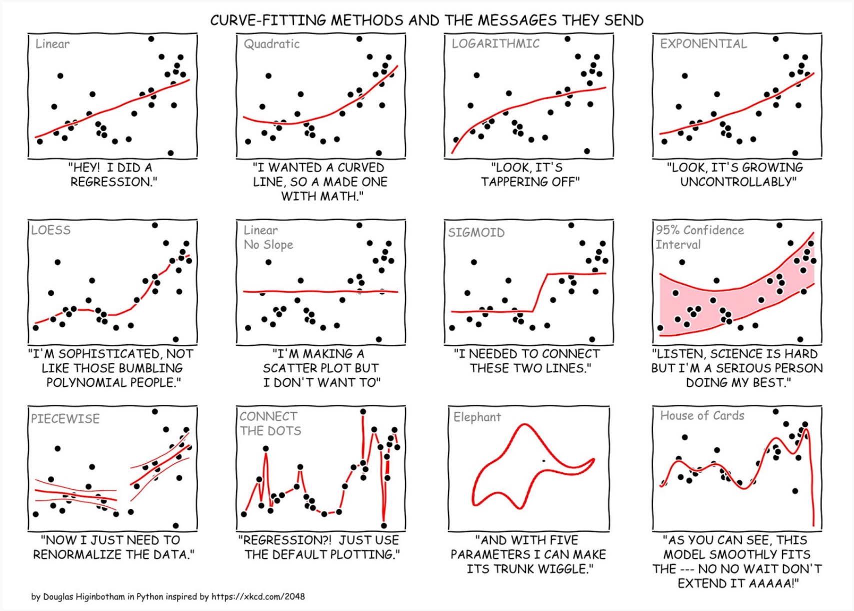
- I use LaTeX to write almost everything these days
- It's aesthetically appealing
- Let's you add figures/tables directly from the file R/Stata produces
- Great at organizing references/literature
- But also has a large fixed cost

## Present only human-interpretable information

- Most tables and figures should be self-contained and self-explanatory
- Watch out for common hard-to-interpret numbers that appear in tables
  - Logit: We don't think in log-odds; transform to predicted probabilities
  - Goodness of fit measures often don't mean much even to an expert reader
  - Remove clutter
  - Avoid jargon and technical abbreviations (pixels are free!)
- Use color and transparency to help make your point
- Include interpretive text such as figure captions and clear, **descriptive titles**

## Present data responsibly by providing scale and context

- Avoid inadvertently distorting the reader or viewer's perception of the data
- If the axes labels do not start at zero, consider whether this distorts the relative size of differences or trends depicted
- Present enough context: Summary statistics may not capture the whole story


CURVE-FITTING METHODS AND THE MESSAGES THEY SEND
Linear
"HEY! I DID A REGRESSION."
Quadratic
"I WANTED A CURVED LINE, SO A MADE ONE WITH MATH."
LOGARITHMIC
"LOOK, IT'S TAPPERING OFF"
EXPONENTIAL
"LOOK, IT'S GROWING UNCONTROLLABLY"
LOESS
"I'M SOPHISTICATED, NOT LIKE THOSE BUMBLING POLYNOMIAL PEOPLE."
Linear No Slope
"I'M MAKING A SCATTER PLOT BUT I DON'T WANT TO"
SIGMOID
"I NEEDED TO CONNECT THESE TWO LINES."
95% Confidence Interval
"LISTEN, SCIENCE IS HARD BUT I'M A SERIOUS PERSON DOING MY BEST."
PIECEWISE
"NOW I JUST NEED TO RENORMALIZE THE DATA."
CONNECT THE DOTS
"REGRESSION?! JUST USE THE DEFAULT PLOTTING."
Elephant
"AND WITH FIVE PARAMETERS I CAN MAKE ITS TRUNK WIGGLE."
House of Cards
"AS YOU CAN SEE, THIS MODEL SMOOTHLY FITS THE --- NO NO WAIT DON'T EXTEND IT AAAAA!"
by Douglas Higinbotham in Python inspired by https://xkcd.com/2048

## Eliminate junk

- Maximize the data-to-ink ratio by using as little ink as possible to show your data
- Remove non data ink, e.g., extra gridlines
- Remove redundant data
- Remove indicators you don't need
- Display only as many decimal places as are relevant

## Represent uncertainty with care

- You DO want to represent uncertainty
- Represent uncertainty without creating visual clutter
- Represent uncertainty in a way that will make sense to your audience (consider their statistical background)

## Choose the type of visualization based on the information you want to convey

- Use a table only if you can't use a figure. Usually when
  - You need to show exact numerical values
  - You want to allow for multiple localized comparisons
  - You have relatively few numbers to show
- Avoid pie charts
  - It is difficult for the audience to visually distinguish the size of each wedge
  - A good alternative for showing shares of a whole is a stacked bar chart

## Consider your audience

- What is the level of technical knowledge of your audience?
- What are the language skills of your audience?
- Avoid jargon, especially if it may not be familiar to the audience
- Always define any technical abbreviations (but avoid using them)
- Will they view images on a computer, as printed materials, or another medium?

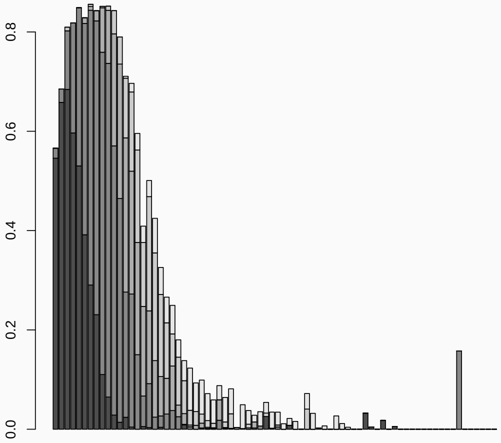
## Other random tips

- Make the font larger than the Stata/R preset
- **LABEL YOUR AXIS**
- The Y-axis tickmarks ideally should be horizontal (R puts them sideways)
- This is a color the scheme I like: `http://www.mulinblog.com/a-color-palette-optimized-for-data-visualization/`
- Use legends if you have multiple lines
- Avoid having two-axis
- **Avoid pie-charts**

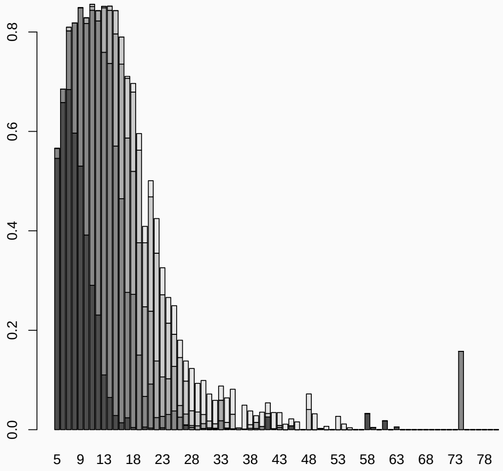
## An example

- Data from the 2015 Household Expenditure and Income Survey in Liberia
- Proportion of people enrolled in different levels of education by age
- Goal: Understand how enrollment looks like for children

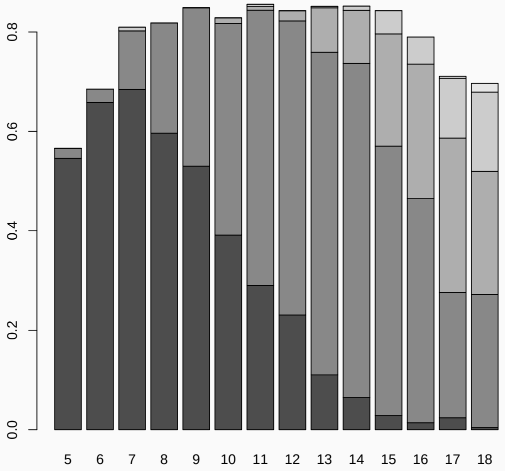
## The figure as R spits it out



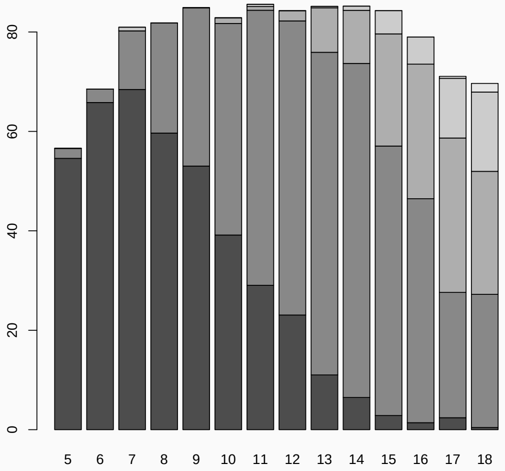
## Let's add x-axis tick marks



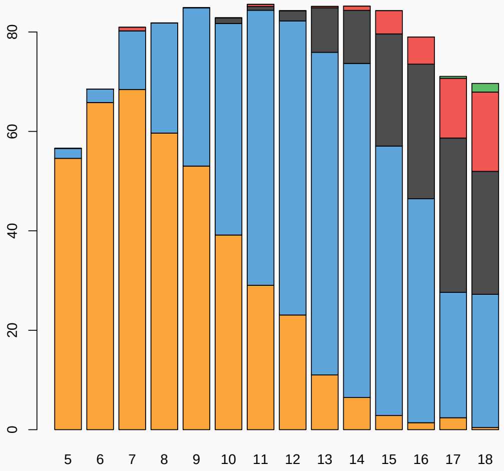
## We only want people aged 5-18



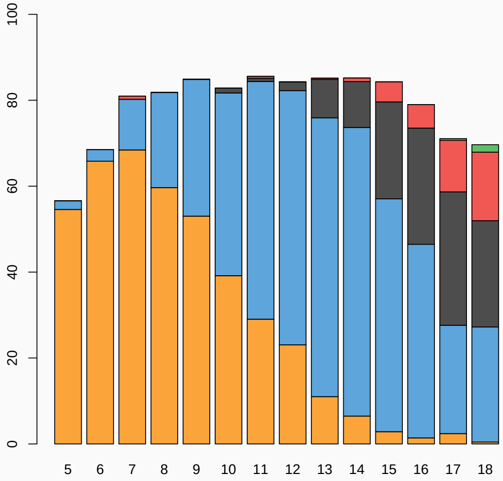
## Lets make the y-axis in % terms



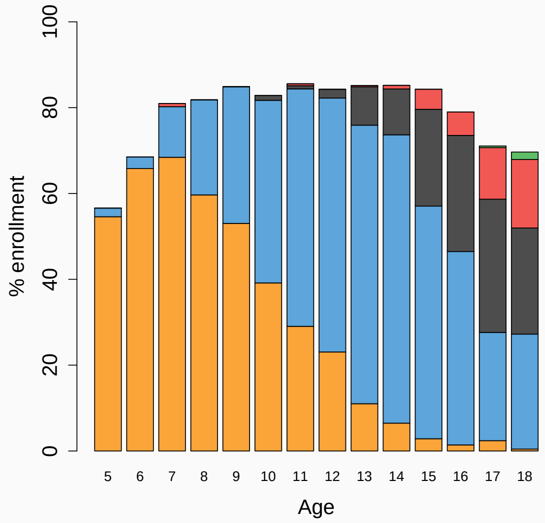
## Use a more appealing color scheme



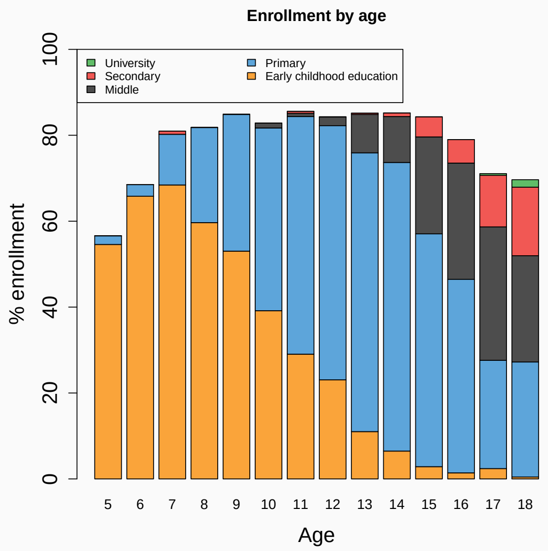
## The y-axis is misleading, lets make it go to 100



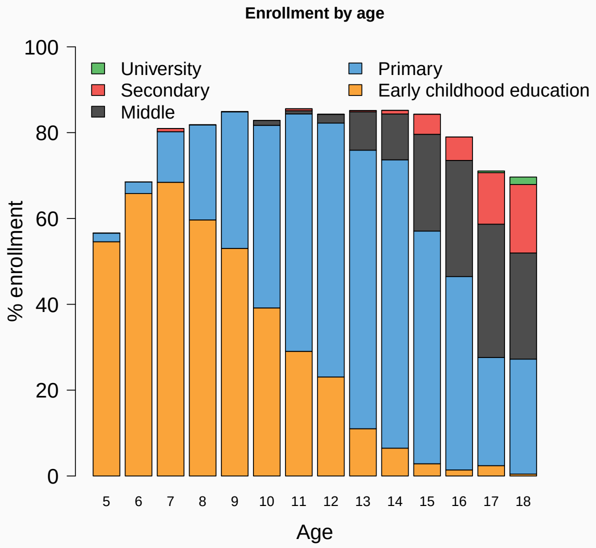
## Add axis labels



## Add a legend to explain what the colors mean



## Make the numbers in the y-axis easier to read and remove legend box



## Basics of programming

## Be critical, but with caution

"It takes an architect to build a cathedral, but anyone can take it down with a sledgehammer"
Jim Hamilton

## Be critical, but with caution

"It takes an architect to build a cathedral, but anyone can take it down with a sledgehammer"
Jim Hamilton

- Don't ever criticize a paper if you don't know how to make it better.
- Remember building a cathedral is about science, but the end product needs to be tasteful/aesthetic (more on that below)

## Selling your work

“Researchers are entrepreneurs in the world of ideas”
Karthik Muralidharan

## Selling your work

“Researchers are entrepreneurs in the world of ideas”
Karthik Muralidharan

- If you don’t advocate for your work, *no one will*
- Network, network, network
- Selling is very important
- Making your research presentable is almost as important as the work itself
- Study the effective rhetoric of economists who successfully communicate their work to others

## More readings (all in the website)

- Jesse Shapiro's "How to Present an Applied Micro Paper"
- Gentzkow and Shapiro's coding practices manual
- Rachael Meager on presenting as an academic
- 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`