Data for
Development Impact

Exportando tablas en Stata III
Programando comandos | RDD

Replicación de Papers

Rony Rodriguez-Ramírez

February 19, 2022

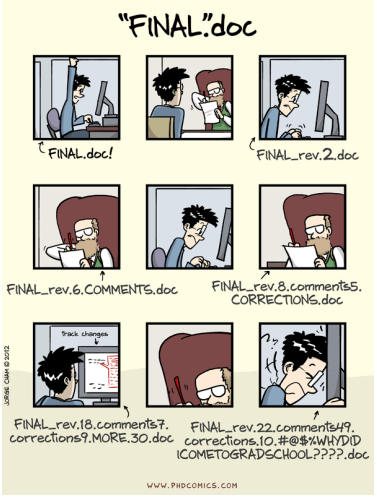
LAMBDA

Intro to Git

Antes de la sesion:

- Confirmemos que tienes una cuenta en GitHub: https://github.com/join
- Instalar GitKraken: https://www.gitkraken.com/

Intro to Git


"FINAL".doc
FINAL.doc!
FINAL_rev.2.doc
FINAL_rev.6.COMMENTS.doc
FINAL_rev.8.comments5.
CORRECTIONS.doc
Track changes
FINAL_rev.18.comments7.
corrections9.MORE.30.doc
FINAL_rev.22.comments49.
corrections.10.#@$%WHYDID
ICOMETOGRADSCHOOL????.doc
JORGE CHAM © 2012
WWW.PHDCOMICS.COM

¿Por qué Git?

- Nos permite hacer rastreo de una documento.
- ¿Quién lo hizo? ¿Qué modificaciones pasaron? ¿Por qué se hicieron estos cambios?
- Nos facilita colaboración entre personas.
- Podemos eliminar el problema de “copia conflictiva” en Dropbox.
- Podemos configurar nuestro flujo de trabajo para replicaciones.

Intro to Git

Tres conceptos claves en Git:

- Clonar (Clone)
- Cometer (Commit)
- Rama (Branch)

¿Qué es clonar

Clonar es similar a descargar un repositorio en su computadora

La diferencia entre clonar y descargar radica en que Git clona una repositorio y recuerda de donde se ha descargado este repositorio. Esto es necesario ya que Git conoce donde se debe compartir las ediciones que se hacen a una archivo específico de una repositorio.

Clonar un repositorio

¿Cómo clonar un repositorio?
Live Preview

Explorando el clon

Dos puntos a tomar en cuenta en torno a los repositorios:

- Commits
- Branches

¿Qué es una commit?

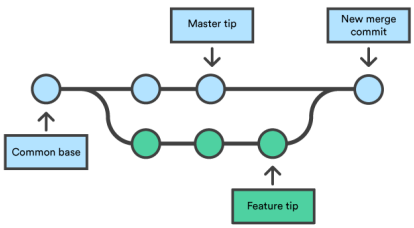
- En lugar de tener una lista de cada versión guardada de un archivo, en Git usamos commits para indicar cuál es cada diferencia significativa entre dos versiones de nuestra carpeta de proyecto.
- Cada commit es una snapshot de todos los archivos de un proyecto. Git lista cada snapshot y lo compara con el snapshot anterior.
- Cada commit tiene una tiempo y rastreo de quién hizo el commit.

¿Como hacer un commit?

Live Preview

Introduciendo branches (ramas)

- La rama es la ***característica clave*** clave de Git.
- Las ramas nos permite crea una copia del código donde nosotros podemos experimentar. Si nos gusta el resultado podemos unir nuestro experimento con el projecto principal.



Más materiales

- Lamentablemente por tiempo no podremos introducir más features de Git.

Regresión discontinua

Intro: RDD

- Presentado por primera vez para estudiar el impacto del reconocimiento al mérito por Thistlethwaite & Campbell (1960).
- Sin embargo, solo comenzó a llamar la atención en economía desde finales de la década de 1990.
- Pero ¿qué es una discontinuidad?
  - Una ruptura brusca en los valores de una función.
  - Matemáticamente, estamos hablando de una ecuación por partes (piecewise equation):

$$f(x) = \begin{cases} \frac{1}{2}x, & x < 5 \\ 2 + \frac{1}{2}x, & x \geq 5 \end{cases}$$

Uso de RDD

- No tenemos un RCT y nos preocupan las variables endógenas.
- RDD utiliza la asignación de discontinuidad exógena para estimar los efectos causales; es decir, las observaciones de un lado de la ruptura terminan siendo muy similares a las del otro lado.
- Esencialmente, terminamos con grupos equilibrados pero no tenemos una asignación de tratamiento aleatorizada.

Prinicipios de RDD

Randomización local:

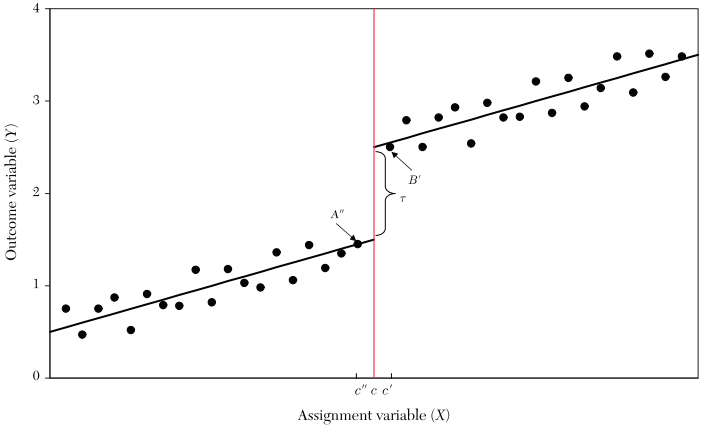
- El estado de tratamiento es una función determinista de una variabla a, por lo cual cuando conocemos X conocemos el estado de tratamiento D_x.
- El estado de tratamiento es una función discontinua de a porque no importa que tan cerca X llegue al corte, D_x continua sin cambios hasta que el corte se ha alcanzado.

$$D_x = \begin{cases} 1 & \text{if } X \geq \text{cutoff} \\ 0 & \text{if } X < \text{cutoff} \end{cases}$$

Prinicipios de RDD

- Llamamos a nuestra variable de asignación X, variable en ejecución (forcing or running variable).
- Un punto importante, es que esta variable no es ortogonal a:
  - Las características observables de los individuos.
  - Las características no observables de los individuos.

RD como una randomización local



Dos condiciones claves en RDDs

1. El tratamiento en la población debe depender en que si la variable observada excede el valor crítico denota como c.
2. Los individuos no tiene un control preciso de la variable de asignación.
   - "Sin control preciso" significa que entre los que puntúan cerca del umbral, es cuestión de "suerte" en cuanto a qué lado del umbral aterrizan.
   - Las personas que marginalmente pasan el corte o quedan atras, son asumidos como identicos (bastante similares).

Programación en Stata

Creando nuestros comandos

Cuando escribe un comando que Stata no reconoce, Stata primero busca en su memoria un programa con ese nombre. Si Stata lo encuentra, Stata ejecuta el programa.

Si Stata no encuentra el comando, debemos definirlo o instalarlo. Por ejemplo:

```
. hello
command hello is unrecognized
r(199);
```

Así es como funciona la programación en Stata. Un programa se define por:

```
program define progname
  Stata commands
end
```

Creando nuestros comandos

Si el comando que deseamos crear usa la sintaxis estándar de Stata. Es decir, argumentos como una lista de variables, posiblemente un peso, tal vez una cláusula "if" o "in", y tal vez un montón de opciones, podemos aprovechar el propio analizador de Stata, que almacena convenientemente todos estos elementos en macros locales listos para que los use.

Entre program y end, debemos utilizar el comando *syntax*. El cual tendrá podría tener la siguiente lógica:

```
capture program drop test
program define test
  syntax varname, options
  ...
end
```

Creando nuestros comandos

Con varname estaremos especificando el uso de variables después de nuestro comando. Este es nuestro primer elemento.

- Puede especificar mínimos y máximos, por ejemplo, un programa que requiera exactamente dos variables diría varlist(min=2 max=2).
- Cuando solo tiene una variable podemos escribir varname, que es la abreviatura de varlist(min=1 max=1).

Stata luego se asegurará de que su programa se llame con exactamente un nombre de una variable existente, que se almacenará en una macro local llamada *varlist*.

Creando nuestros comandos: [Opciones]

Los elementos de sintaxis opcionales están encerrados entre corchetes [ ]. Los argumentos no opcionales van simplemente después de una comma. En este caso tendremos nuestro comando de la siguiente forma:

```
capture program drop generate_name
program define generate_name
  syntax varname, Generate(name)
  ...
end
```

Ahora nosotros tendremos dos macros que podemos utilizar en nuestro comando: ”varlist” y ”generate”.

Creando nuestros comandos: [Opciones]

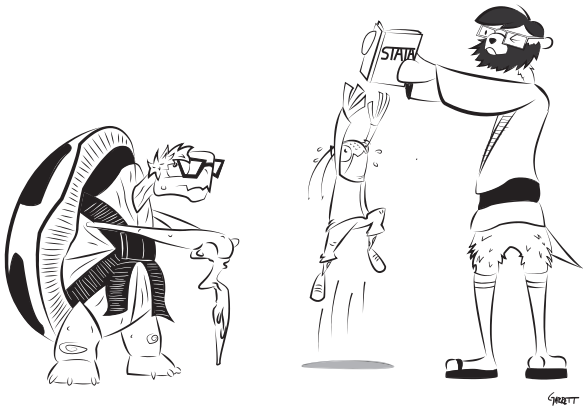
Aqui es donde ya encontraremos las cosas más interesantes de los comandos. Por ejemplo:

```
capture program drop generate_name
program define generate_name
  syntax varname, generate(name)
    gen ‘generate’ = ‘varlist’

end
```

También podríamos agregar si queremos hacer cambios extras a esta nueva variable. Pasaremos a Stata ahora para realizar más ejemplos.

STATA TIME



End.