Econometrics in R's `tidyverse`

by Tyler Ransom, University of Oklahoma
@tyleransom

Basics of R

R can be thought of as a really fancy calculator

Packages:

- R comes with a lot of functionality out-of-the-box
- Other functionality requires the user to load packages
- One-time installation: `install.packages("tidyverse")`
- Each time you open R: `library(tidyverse)`

Commenting:

- Use `#` to make a comment
- This tells R to ignore that code

```
# My name is Tyler
```

Assignment operator:

- Use `<-` to store a calculation, e.g. `x <- 3` ("$x = 3$")

Pipe operator:

- Use `%>%` to "pipe" objects

  `y <- mean(log(x))` becomes `y <- x %>% log %>% mean`
- `%<>%` pipes forward, then backwards

  `x <- mean(log(x))` is same as `x %<>% log %>% mean`

Working with Data

R's fundamental data object is a **data frame**

Like spreadsheets, stores data in columns and rows

`tidyverse` uses **tibbles** (enhanced data frames)

```
df <- as_tibble(mtcars)
```

Reading in data

- Many functions for reading in different types of data

  `df <- read_csv("myfile.csv")` (comma separated)

  `df <- read_fwf("myfile.dat")` (fixed-width)
- More details: see Data Importing Cheat Sheet
- `haven` package: import foreign files (e.g. SAS, Stata, ...)

Accessing columns of data

- To reference a column in a tibble, use `$`

  `df$mpg`

  `mean(df$mpg)` will return sample avg of `mpg` variable

Ignore missing values

- Missing values are indicated by `NA`
- Some commands won't automatically ignore `NA` values
- For these cases, use `na.rm` option

  `mean(df$mpg, na.rm=TRUE)`

  `df$mpg %>% mean(na.rm=TRUE)` (equivalent)
- Otherwise, `R` would say the mean is `NA`

Removing columns and rows from a tibble

- To keep columns in a tibble, use `select()`

  `df1 <- df %>% select(mpg,disp,hp,gear,carb)`
- To keep rows in a tibble, use `filter()`

  `df1 %<>% filter(mpg>=10)`
- To remove columns, put a minus in front

  `df1 <- df %>% select(-mpg,-disp)`

Remove missing values from a tibble

- To remove **all** rows with *any* `NA` values, use `drop_na()`

  `df1 <- df %>% drop_na()`
- Can also drop `NA`'s from particular columns:

  `df1 <- df %>% drop_na(gear,carb)`

Creating new columns in a tibble

- To create a new column in a tibble, use `mutate()`

  `df1 %<>% mutate(mpg.squared = mpg^2)`

Manipulating values of a variable

- To replace (i.e. recode) values of a variable:

  `df %<>% mutate(gear = replace(gear,gear==4,99))`

  Changes all 4's in `gear` to be 99's

  `gear==4` can be any other logical condition
- To specify a series of conditions, use `%in%`

  `df %<>% mutate(hp =`

  `replace(hp,hp %in% c(110,120),99))`

  Changes all 110's or 120's in `hp` to be 99's

Working with discrete variables

- Discrete variables often require special treatment
- In R, declare discrete variables as **factors**

  `df %<>% mutate(gear = as.factor(gear))`

Other data manipulations

- See Data Wrangling Cheat Sheet

Getting to know your data

It's important to know what's in your data by

1. Looking at summary statistics
2. Performing cross-tabulations
3. Visualizing certain variables

Summary statistics

- Report quartiles, min/max, mean, and #`NA`'s:

  `summary(df)`
- Report N (total non-missing), mean, SD, min/max:

  `df %>% as.data.frame %>% stargazer(type="text")`

Cross-tabulations

- Report frequencies of a discrete variable:

  `table(df$gear)`
- Average y by categories of a discrete x variable:

  `df %>% group_by(gear) %>%`

  `summarize(m.mpg = mean(mpg))`

Visualization

- Often helpful to look at a histogram or line graph
- Histogram (continuous x):

  `ggplot(df, aes(mpg)) + geom_histogram()`
- Histogram (factor x):

  `ggplot(df,aes(x=gear)) + geom_bar()`
- Kernel density plot:

  `ggplot(df, aes(mpg)) + geom_density()`
- Simple scatter plot with linear fit:

  `ggplot(df,aes(disp,mpg)) + geom_point() +`

  `geom_smooth(method="lm")`
- More details: see ggplot2 Cheat Sheet

Regression modeling

Basic OLS regression

- Regression:

  `est <- lm(mpg ~ gear + hp, data=df)`
- Examine regression output:

  `summary(est)`

  `tidy(est)`

  `stargazer(est,type="text")`
- Other functional forms:

  `est <- lm(mpg ~ gear + I(gear^2), data=df)`

  `est <- lm(log(mpg) ~ gear + I(gear^2), data=df)`
- Factor variables automatically get separate intercepts

t-statics and F-statistics

- t-stats, p-values reported in regression output
- F-test:

  `linearHypothesis(est,c("gear","hp"))`

  tests $H_0 : \beta_{gear} = 0, \beta_{hp} = 0$

  `linearHypothesis(est,c("gear=5","hp=-1"))`

  tests $H_0 : \beta_{gear} = 5, \beta_{hp} = -1$
- Robust F-test (see next section):

  `linearHypothesis(est.rob,c("gear","hp"))`

Robust standard errors (`estimatr` package)

- Correct for heteroskedasticity:

  `est.rob <- lm_robust(mpg ~ gear + hp, data=df)`
  or

  `stargazer(est,se=starprep(est.rob),type="text")`
- Correct for serial correlation:

  `fixed.est <- est %>% coeftest(vcov=NeweyWest)`

  `stargazer(est,se=list(fixed.est[,2]),type="text")`
- Correct for clustering:

  `est.clust <- lm_robust(mpg ~ gear + hp, data=df,`

  `clusters=df$carb)`
  or

  `stargazer(est,se=starprep(est.clust),type="text")`

Instrumental Variables

- Let `drat` be the endogenous covariate
- Let `wt` be the instrument
- Let `qsec` and `gear` be exogenous covariates

  `est.iv <- ivreg(mpg ~ drat + qsec + gear |`

  `wt + qsec + gear, data=df)`
- Instruments come after the | symbol
- Endogenous covariates come before the | symbol
- Exogenous covariates appear on both sides of the |
- First-stage regression:

  `est.1 <- lm(drat ~ wt + qsec + gear, data=df)`

  `df %<>% mutate(drat.hat = est.1$fitted.values)`
- Second-stage regression:

  `est.2 <- lm(mpg ~ drat.hat + qsec+ gear,data=df)`
- Can also use `estimatr` for robust SEs:

  `est.ivr <- iv_robust(mpg ~ drat + qsec + gear |`

  `wt + qsec + gear, data=df)`

Working with time series data

- Declare a time series data frame

  `df.ts <- zoo(df, order.by=df$year)`
- Time series line plot:

  `ggplot(df.ts, aes(year, inf)) + geom_line()`
- Simple AR(1) model:

  `est <- dynlm(inf ~ L(inf,1), data=df.ts)`
- First-differences model:

  `est.diff <- dynlm(d(inf) ~ unem, data = df.ts)`
- ADF test for unit root:

  `adf.test(df1.ts$inf, k=1)`
- ARIMA model:

  `est.arima <- auto.arima(df.ts$inf)`
- Plot h-period-ahead forecast intervals

  `autoplot(forecast(est.arima, h=2))`
- Extended date and time functions available in `lubridate` package

Working with panel data

- Report number of units and time periods

  `pdim(df)`
- Pooled OLS model

  `est.pols <- plm(lwage ~ exper + I(exper^2) +`

  `year, data = df, index = c("id","year"),`

  `model = "pooling")`
- Random effects model

  `est.re <- plm(lwage ~ exper + I(exper^2) +`

  `year, data = df, index = c("id","year"),`

  `model = "random")`
- Fixed effects model

  `est.fe <- plm(lwage ~ exper + I(exper^2) +`

  `year, data = df, index = c("id","year"),`

  `model = "within")`
- First differences model

  `est.fd <- plm(lwage ~ exper + I(exper^2) +`

  `year, data = df, index = c("id","year"),`

  `model = "fd")`

Limited dependent variable models

Linear probability model (LPM):

- If y is a factor, format it as a numeric

  `est.lpm <- lm(as.numeric(y) ~ x1 + x2, data=df)`

Logit and Probit:

In this case, y should be formatted as a factor

- Logit:

  `est.logit <- glm(y ~ x1 + x2,`

  `family=binomial(link="logit"),data=df)`
- Probit:

  `est.probit <- glm(y ~ x1 + x2,`

  `family=binomial(link="probit"),data=df)`

List of packages

The document requires the following packages:

tidyverse	car	sandwich	tseries
magrittr	estimatr	zoo	lubridate
stargazer	lmtest	dynlm	forecast
broom	clubSandwich	AER	plm

Layout by Winston Chang, http://wch.github.io/latexsheet/