Guide to R
Data analysis for Economics

Guide to R
Data analysis for Economics

William A. Sundstrom
Michael J. Kevane

Dedication

The guide is dedicated to the student who wrote (to one of us) the following:

Something terrible happened just now. I normally dont save often in Rstudio because it seems to have no problems staying open, so I hadnt saved the code for my final essay and now it suddenly crashed. Theres just that rainbow wheel and it says application not responding. Do you know of anything I should/could do? I'm totally fucked
Thanks!

We liked the absence of final period, the chipper thank you at the end, and the followup email, “Nevermind!”

- Version 1.0 was prepared by William A. Sundstrom and Bobby Fatemi, September 2012.
- Version 2.4 edited by Michael Kevane
- Versions 1.1 -- 2.3 edited by William A. Sundstrom
- Version 3.0 edited by William A. Sundstrom and Michael Kevane, with assistance from Derran Cheng, December 2015,
- Version 4.0 edited by Michael Kevane December 2016
- Version 7.0 edited ichael kevane August 2017

This is a work in progress. Feedback of any kind is welcomed! Please contact Bill Sundstrom, wsundstrom@scu.edu , or Michael Kevane, mkevane@scu.edu , Dept. of Economics, Santa Clara University.

On-line version with links to downloads: `http://rpubs.com/wsundstrom/home`

Table of Contents

Introduction

This guide was designed for use by Economics majors. It may also be helpful to others who want a hands-on approach to using R for basic econometrics, and for students taking other courses that use R who need a refresher. It consists largely of tutorials, which walk the user through accompanying R scripts (programs). The topics and applications in the tutorials closely follow the content of a typical introductory econometrics class and replicate some of the examples used the best-selling textbook by James Stock and Mark Watson, *Introduction to Econometrics* (Pearson).

What is R?

R is an open source programming language and software environment for statistical computing and graphics. It is widely used by statisticians and other number crunchers for data analysis and related applications. A growing number of large and small organizations, including many companies here in Silicon Valley, use R for a variety of vital functions, from Google and Facebook, to UBER and Orbitz, to the New York Times and the non-profits Benetech and Human Rights Data Analysis Group.

R provides most of the data management and econometric tools needed to manipulate and analyze data. Being open-source, the R development community provides free add-in "packages" that expand and enhance R's capabilities.

R can be used interactively: You can type in commands (such as 2 + 2) and R will output the results for you to see (hopefully 4!). However, R is most powerful as a "scripting language." A script is a sequence of commands that you submit to R and R executes, much as a cook might follow the steps of a recipe. Scripting is very important because it provides an easy way to fix mistakes, to insert or delete commands, to repeat what we have programmed without having to re-enter everything, and to give others an opportunity to replicate our work.

For most people, the goal in using R for data analysis to analyze data, not to write complex computer programs. Using R is about copying, pasting, and modifying. Therefore, to make it easier we provide you with sample scripts that you can modify rather than write from scratch. We will use the interface called RStudio to edit and run R scripts.

How to use this guide

Each chapter in this guide consists of a tutorial that will help you learn how to conduct a variety of statistical analyses and manage your data in R. The tutorials refer to accompanying R scripts and sample data sets, which can be downloaded from a public website (indicated at the beginning of each chapter). For each tutorial, you should open the accompanying script in RStudio and follow the instructions in the tutorial systematically, running the lines of script as called for, and examining the results. The later tutorials build on the earlier ones, so do them in order.

We try keeping the tutorials up to date, but R is constantly evolving. Often there is more than one way to do something, and new packages that offer new and improved features are constantly appearing. We encourage users to search and experiment. Let us know if you find a better mousetrap-- or build one yourself!

1. Getting started

In this tutorial, you will learn how to:

- Know what computer hardware you need
- Install R and RStudio
- Create a working directory (folder) on your computer
- Download the tutorial scripts and data sets to your working directory
- Get to know the RStudio interface
- Write and run a simple R script
- Install packages

Where to find what you need:

- Downloads: R tutorial scripts and data files `http://rpubs.com/wsundstrom/home`

What hardware do you need?

Any standard personal computer will work fine for running R, Mac or Windows. You can often buy a new laptop for less than $300, and an adequate used laptop for less than the cost of a textbook. Some tablet computers (such as the Surface) may be compatible with R Studio. The Ipad cannot be used with RStudio at the time of this writing, but RStudio has an online version that may soon be convenient.

We recommend that you also purchase an inexpensive external "mouse" for your laptop. You will do a lot of highlighting of sections of code, which is much easier with a mouse than with a finger and touchpad. Investing a few dollars in a mouse will repay itself many times over in saved frustration.

Install R

To get started, you need to download and install two different free software packages: R and RStudio. You will actually be interacting with RStudio, even

though R does all the number crunching. R is maintained by CRAN, the Comprehensive R Archive Network. CRAN is a collection of sites that carry identical material, including downloads of R, extensions, documentation, and other R related files. R is available for many platforms, and below are download instructions for Mac and Windows.

YOU SHOULD INSTALL R FIRST BEFORE R STUDIO.

SO DO THE FOLLOWING FIRST!

Download and install the software program for R. Visit the CRAN website

`https://cran.r-project.org/`

Click on the Download link corresponding to the operating system that you will be using to run R—for most of you, that is Mac OS X or Windows.

You may at some point be asked to choose a CRAN "Mirror" from a long list. A mirror is a server that maintains everything R related. It does not matter which one you pick. Follow the instructions on the next screen, depending on your operating system.

MAC users: Figure out which version of the Mac operating system (OS X) you are running, and click on the appropriate version of R. (You can find your version by clicking on the Apple icon in the upper left and then "About This Mac".) After the install package has been downloaded, find it (probably in your Downloads folder) and double click. Let Installer walk you through installation. No customization is necessary. Some newer version Macs will automatically install the software.

Windows users: Click the link that says "install R for the first time." On the next screen, click the link to download R for Windows. After downloading the install package, which probably can be found in your Downloads folder (it will be a file ending with .exe), double click it and follow the installer's instructions. Typically nothing needs to be done other than clicking either 'next' or 'I Agree' every time, accepting the default settings.

Linux users: Installing R and RStudio will often be considerably more complex: consult a Linux expert.

Mac and Windows users: Now that you have installed R, you could access and run it by clicking on the icon that might now be on your desktop or among your programs. But as mentioned above, we will be interacting with R through the RStudio interface, so let us install that now.

Install RStudio

Now that you have R installed on your machine, you should download and install RStudio, which provides a more user-friendly interface and script editor for working with R. Downloading is similar for Mac and Windows.

Go to the RStudio website at https://www.rstudio.com/.

Click Download RStudio. Then click to download the Desktop version that is free and then under the heading "Installers for Supported Platforms" you should be able to find the link for your system, either Windows or Mac. Click on that link and download the installer. After the download, go into your Downloads folder and double click on the downloaded package to install.

If you have an older Mac (running OS X 10.5 or earlier), you may need to install a different version of RStudio. Seek help! Better yet, just buy a cheap Windows laptop. ;-)

The most typical problem when installing is that someone else set up your computer and you do not have "administrator" privileges when you use the computer. If that is the case, logout and re-boot your computer by logging in as the administrator, and then install R and R-Studio.

Another typical problem for Mac users is that you open and run RStudio from your download folder instead of having be a proper installation. When users do this, they end up "installing" RStudio repeatedly. So be careful. Ask for help.

Create a working directory (folder)

When you are first starting to use R, it is desirable to have all your scripts and data in a single working directory (folder). You should create this new folder in whatever folder you use for course work. For example, a folder name might be *Classes/econ42*.

Once you create the working directory, you will need to know its full "path" name. For example, in Windows the name might be:

```
C:\Users\mkevane\Documents\Classes\econ42
```

on a Mac it might be:

```
/Users/mkevane/Documents/Classes/econ42
```

Of course, unless you are Michael Kevane (or Micah Kevane), your username will be different. You can find the full path using Finder on a Mac or Explorer in Windows. Note that the Mac uses forward slashes and Windows uses backwards slashes. In a programming quirk, R only recognizes forward slashes, so that in your R code, the working directory for a Windows machine will be written as, for example:

```
C:/Users/mkevane/Documents/Classes/econ42
```

To set the working directory, type something like this for a Mac or switch the hasttags if you use Windows):

```
# (edit for YOUR folder)
setwd("/Users/mynamehere/econ_42")
# setwd("C:/Users/mynamehere/econ_42")
```

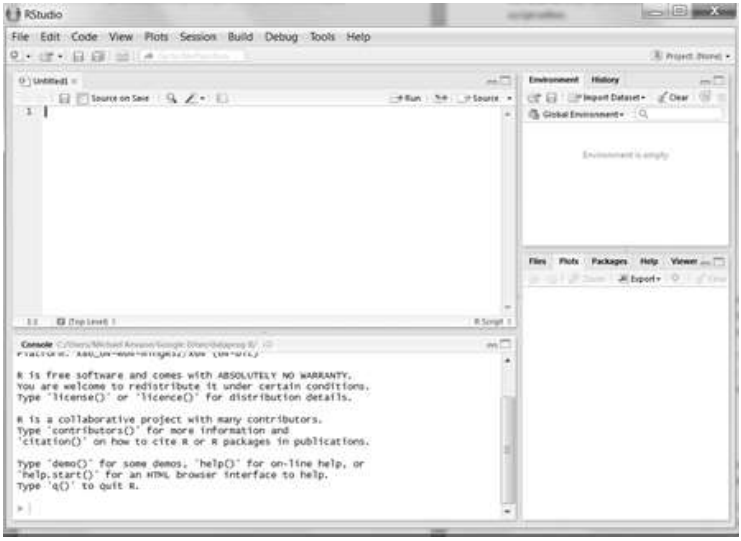
Setting your working directory is one of the hardest things for many students, who are accustomed to having all of their files on the desktop or in the downloads folder. For many reasons that is a TERRIBLE practice. Therefore, this class will break you of that habit, at least once, and force you to create a folder where you keep all the files related to the class.

Download scripts and data

Most of the files you need for the tutorials in this guide are in a compressed (zip) file available for download at http://rpubs.com/wsundstrom/home. Some of the scripts though may be older than the most recent versions discussed in this guide. The files are downloadable in a zipped file named *files42.zip*. A zipped file is a compressed set of files, somewhat smaller than the underlying files. Save this file to your computer. Once the file is downloaded, then unzip the files. In some cases, your computer may unzip the files automatically when you download them. In that case you will have a new folder, probably called *files42*, containing all the data and scripts. If your computer did not unzip the file, you should double click on *files42.zip* to unzip and create the folder with the files. If you have never unzipped a file before, you may need to install a free program such as WinZip or 7-Zip.

After you have the "unzipped" folder with all the script and data files in it (probably called *files42*), move all the files into your working directory (which you created, right? See previous section).

The R scripts all have names like *t_rbasics.R*. The data files are all "comma-separated values" files ending in the extension .csv. The zip file also contains some documentation for the data.

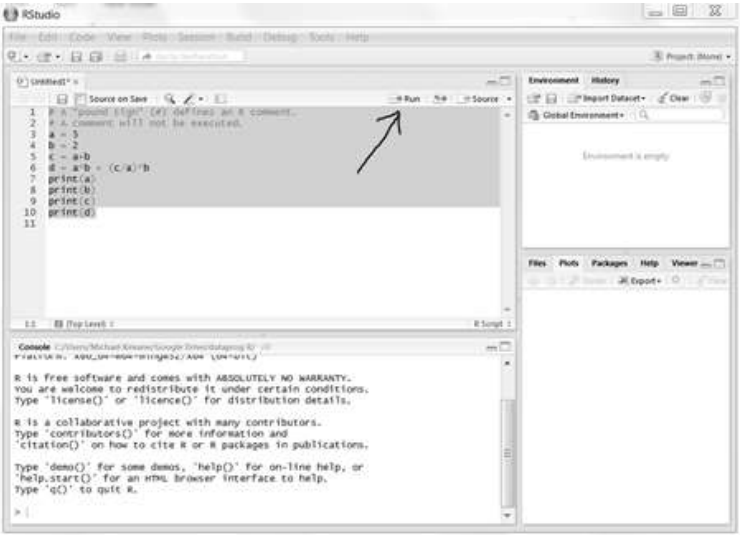


Getting to know your way around RStudio

Open the RStudio program. When you start RStudio for the first time, the screen will look something like the screenshot on this page. The left frame is the R Console. This is where commands could be entered one by one, if you wanted to run R interactively. We will not usually enter commands this way—instead, we will use the script editor.

Let us open the script editor and enter a simple RStudio script. At the top of the window, use the drop-down menus to select *File → New → R Script.* You should see something like the screenshot above. The top left frame that is now opened up is an area where you write and edit scripts: the R Script Editor. A script is simply a set of executable commands that can be run in chunks or all at once. Here's an example. Type the following simple script into the script editor (it works like a simple word processor):

```
# A "pound sign" or hashtag (#)
# defines an R comment.
# A comment will not be executed.
a = 5
b = 2
c = a+b
d = a*b + (c/a)*b
print(a)
print(b)
print(c)
print(d)
```



Notice that when you type the script nothing happens. The script has to be "run." Before we do that, take a look at the commands in the script. The "=" sign defines or creates a new object: it could be a number, like a or b here, or a variable in your data, or whole data set, or output from a graph or statistical procedure.

Many R users use the characters "<-" instead of "=" to define new objects (as an assignment operator). For defining objects, "<-" usually means exactly the same thing as "=". Thus the line:

```
a = 5
```

could be written:

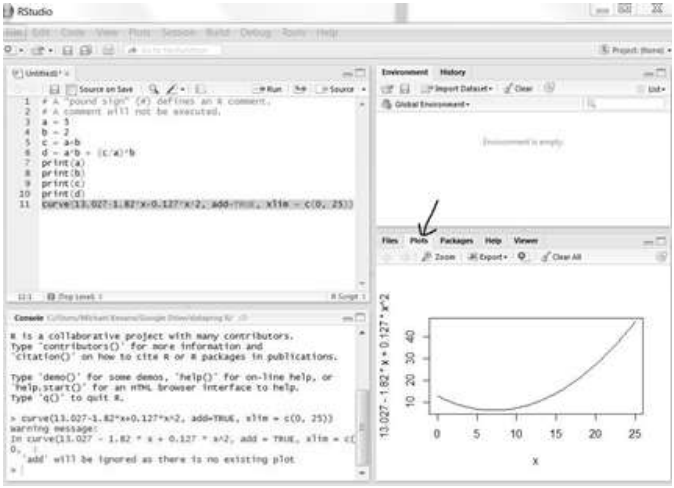
```
a <- 5
```

We use "=" in this guide because it is commonsensical for economists, but be aware that "<-" is the convention among most R users.

Can you predict what R will do once we run this? Let us run the script and see. You can run the script a couple of different ways.

Often we will want to run just a part of the script. To do this, highlight any section of your script you want to run and press the run button located at the top right of your Script Editor (see arrow in the screenshot). Sometimes it is more convenient to click on the line number, which will then highlight the line itself. You can also click and hold to highlight a set of lines by moving over the line numbers. Note: Here is where you will start to see the advantages of using an external mouse and not your touchpad!

Alternatively, if you want to run the whole script, you can go to the Code tab at the top and click on Run Region, then press Run All.



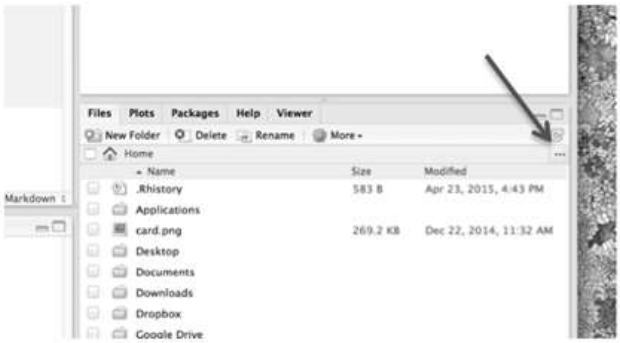
Once you run the script, you should notice a couple of things. First, you should see the output of the script (as well as the code that was run) in the R Console, which is the frame in the lower left directly underneath your script editor. Second, you should also notice that the top right box, which is your Workspace, is no longer blank. Your Workspace shows you everything that you have stored during your current or most recent session using R—from saved variables to imported data sets. Entire workspaces can be saved and reopened later. We encourage you, though, to never save the workspace (always click "no" when prompted when you close RStudio), and instead save your script.)

The bottom right box, the Session Management area, is where you can access your files (saved scripts, plots, etc.). You can also access your installed packages (more on packages later), and you can get additional help if need be. Plots will appear here as well. We prefer not to use interactive menus. You should too.

To see how a plot will appear, note that R can also act as a graphing calculator. Type the following as a new line in your script editor:

```
curve(13.027-1.82*x+0.127*x^2, add=TRUE,
      xlim = c(0, 25))
```

Highlight the line and press the run button. You should see a curve appear in the plot window on the lower right. (You may have to click on the "plots" tab.) Type in a different equation on the following line (you may cut and paste the existing line, and then modify, rather than retyping everything) and see what happens. Does another line appear on the plot?



Install packages

A package in R is like an "add-in" for Excel or a downloadable app for your phone. Some packages come as standard equipment with R, but some must be downloaded and installed. A package has its own set of commands and help documentation. You can type any package name + R into Google, and you will almost always find helpful hints, summaries and documentation.

Among the packages we will use in these tutorials are "sandwich", "lmtest", "car", "stargazer", "AER" , "ggplot2", "WDI", "gdata", "doBy" and "countrycode". We will see what they do later.

Packages need only be installed once onto your computer. This process downloads some files and makes them available to R. If you change computers, the packages will need to be installed again.

Open the script *t_install_packages.R*. To find the script, you can click on the three dots in the lower right window as indicated by the arrow in the screen shot, and then browse through your directories to the folder. Open the *files42* folder and the *tutorials* sub-folder. The R scripts and data sets you downloaded should appear in the file list. Find and click on the file *t_install_packages.R*. The script should open in your script editor (upper left). The script editor works pretty much like a simple word processor.

The commands to install packages look like the following:

```
install.packages(c("sandwich","stargazer,"ggplot"))
```

Notice that c() has in the parentheses a list of packages, each separated by a comma. Notice the double parentheses at the end of the line. An alternative,

```
install.packages("sandwich")
```

will install just one package. The command can be copied and changed to have the whole list of packages installed.

Check that you are connected to the Internet, and then highlight the commands in the script and click on “run” to install the packages. You should see in the Console (lower left) that the packages are being installed. You will see a lot of R text appearing. This does not necessarily indicate a problem. R uses red script sometimes simply to highlight important information. So skim the script that appears; unless it explicitly says that the install failed, you are alright.

Sometimes installing a package does not work. For example, you might receive the following message when installing an R-package (on a Windows computer).

```
‘lib = "C:/Program Files/R/R-3.2.3/library"' is
      not writable
```

You should close R-Studio completely (saving your script), and then go to Menu, click on R-studio and then right-click to “Run as Administrator”. This will restart your R-Studio and should allow you to install the package with no problem.

Quick Reference: The RStudio screen

This screen shot shows the four basic windows or boxes you will be using once you get up and running in RStudio. Refer back to this page if you forget which is which. You can change the size of the different frames by hovering on the line between two frames and then clicking and dragging.

Script Editor (upper left) is where you will view and edit your R scripts (commands).

R Console (lower left) is where R is actually running, and where most of your data analysis results will appear.

Workspace (upper right) shows data sets and variables you have loaded or created. Usually you want to be in the Environment tab.

Session Management (lower right) has tabs for various R-related files; plots (graphics) you have created; packages (program add-ins to do a variety of tasks); and help.

Summary

Key R commands (functions):

- setwd: sets the working directory
- install.packages: downloads and installs add-in packages

Key points/ concepts:

- R is the software that runs the data analysis.
- RStudio is our interface with R, in which we can edit scripts, see output, see variables and datasets, and manage files.
- To do analysis we need to open RStudio only; it will automatically open R
- The RStudio screen has four basic windows:
  - Script editor for editing and running code
  - Workspace, showing open data sets and saved results
  - R console for results and running R commands interactively
  - Session management, with tabs for file list, plots, packages, and help
- Keep all your R scripts and data sets for this course in a single working directory (folder), and make sure you know where it is.
- Scripts can be edited and run from the script editor.
- Packages need only be installed once, but must be loaded (library) every time you start R and want to use that package.

2. Scripts and data

In this tutorial, you will learn how to:

- Open an R script in RStudio
- Organize an R script
- Set the working directory in your script
- Load packages used in the tutorials
- Import data from a spreadsheet file into R
- Clean and manipulate the data
- Deal with a couple of common problems that arise when running R code

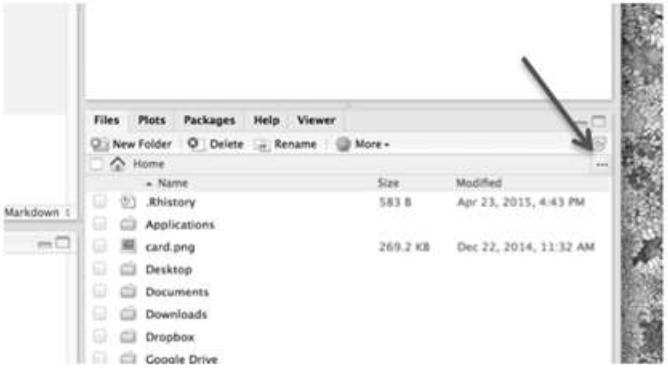
Where to find what you need:

- Downloads: R tutorial scripts and data files `http://rpubs.com/wsundstrom/home`

Introduction

In this tutorial we will look at a basic R script, run some preliminary commands to set up R, and then look at some actual data. Statistical analysis involves manipulating data, so the most important first thing to learn is how to get data into the R programming environment. R stores data in a format that R users call a *dataframe*. For most purposes, this is equivalent to a *dataset*, but R can also associate single numbers (scalars), lists and other vectors, and labels, etc. with a dataset. A dataframe is broader than just a data set.

This guide emphasizes the usage of scripts. Good data analysis practice is to start with a raw, original dataset and then have every single manipulation or transformation of the original dataset stored in a script. That way, any person can take the original data and replicate the analysis. Perhaps they (or you!) will find a mistake. A script is vastly superior to transforming data in Excel. If you use Excel to transform data (by, for example, creating a new variable based on existing data) there is often no



record of the transformation (except in your head). Countless researchers have been embarrassed to admit that they cannot replicate their original analysis (we include ourselves in that list). It was not common practice until the 2000s to insist on a script enabling replication in the social sciences. Most social scientists transformed their data extensively in Excel prior to statistical analysis. They no longer are able to reproduce those transformations.

Open a script

To get started, open RStudio (Note: this also opens R; you do not need to open R separately). Under the Files tab (in the lower right frame called Session Management), go to your working directory. (This is one of the few times we will use this menu, by the way). The working directory is the folder you created in Tutorial #1, named something like *econ42*, which should contain all your R scripts and data sets.

As noted in the previous chapter, to find the script, you can click on the three dots as indicated by the arrow in the screen shot, and then browse through your directories to the folder. Open the *files42* folder and the *tutorials* sub-folder. The R scripts and data sets you downloaded should appear in the file list. Find and click on the file *t_rbasics.R*. The script should open in the script editor (upper left).

Script format

In the editor window, scroll through the *t_rbasics.R* script to see what it looks like. Recall that lines beginning with the "hashtag" symbol # will be ignored by R. We use these lines to insert comments and other information about what the script is doing, and to break the script into sections.

It is good practice always to follow the format of this tutorial for your R scripts, with a brief title, then YOUR NAME and the date, followed by a

short description of what the script does. This is useful information for your future self (a year from now you might want to use a script, but without the comments you may no longer remember what the script is doing).

Every line that is part of a comment must start with a # sign, or else R will treat it as a command and give you an error message when you try to run it.

```
#=======================================
# Data Analysis Tutorial: Script basics,
# import data
#=======================================

# original Bill Sundstrom 9/1/2014
# edits by Michael Kevane 12/28/2014
# Latest version: Bill Sundstrom 8/14/2015

# Description: Script format, install packages,
# set options, import and describe
# CA school district data
```

Following the title and description, the script is organized into three main sections:

1. Settings, load packages, and options
2. Data section
3. Analysis section (empty in this script... see later tutorials)

The Settings section loads packages and locations of datasets that you might use or import for analysis.

In the data section, you will import your data set, create any new variables, and create any sub-samples of the data. Here is also where

In the analysis section, you typically first run some descriptive statistics to see what is in the data. Then commands will create tables and plots to describe the data, and conduct any analysis—in particular, regressions.

Whenever you need to write a script, you can simply modify an existing script and then "save as" using a new filename reflecting the assignment or exercise (e.g. Save as… *Mypractice_script_1*).

Settings: working directory, packages, etc.

The first main section of the script is

```
#=======================================
# 1. Settings, load packages, and options
#=======================================
```

The code in this section sets your working directory, loads the packages we will often use in this course, turns off scientific notation so numbers will be

easier to read, and does a few other useful things. *This section or one very similar should be at the start of each of your scripts.*

Clear the working space: The first little instruction to R will clear your working space, which means it will remove data sets and results that are in R's memory. This is like starting your R session with a clean slate.

```
# Clear the working space
rm(list = ls())
```

Set the working directory: You need to set the working directory so that when you ask R to import, analyze and save data sets, R knows where to find them on your computer. Edit the setwd(…) command for your working directory (folder), replacing the part inside the quotes with the full path name of your folder.

IMPORTANT: whether you use a Mac or Windows machine, the slash symbols in the setwd(…) path must be "forward" (/) slashes, not "backward" (\ ).

```
# Set working directory
# (edit for YOUR folder)
setwd("/Users/mynamehere/econ_42")
```

When setting your directory, it is best not to use the "Set as Working Directory" feature that R-Studio offers in the Files section of the Session Management window. Instead, go to the folder that you have created and right click on one of the files. Then click (right click or left click or double finger on touchscreen) to get "Properties" where you can see the file path or directory; you can then copy and paste your working directory into your script.

Load packages: Whereas you only have to *install* a package onto your computer once, packages you want to use for analysis must be *loaded* into R every time you start a new R session. The library() commands load the packages. By including these commands in all your scripts and running them every time you start up RStudio, you will not need to worry about remembering to do it.

```
# Load the packages
# (must have been installed, as above)
library(sandwich)
library(lmtest)
library(stargazer)
library(ggplot2)
library(WDI)
library(gdata)
library(doBy)
library(reshape)
library(countrycode)
```

The remaining lines of code in Section 1 set the rule for usage of scientific notation (9E7 or 90000000) for numbers and define a function *cse* that will produce correct standard errors for regressions (more on that later).

Now run the entire Settings section of code as a block if you have not run it already. Highlight all the lines from the very top of the script through all of Section 1 (remember you can also highlight by clicking on the line numbers rather than the actual lines), and click the Run button.

You should see in the Console (lower left) that the packages installed are being loaded. You will see a lot of R text appearing. This does not necessarily indicate a problem. R uses red script sometimes simply to highlight important information. So skim the script that appears; unless it explicitly says that the package is not found, you are alright.

Once your packages are loaded, click on the Packages tab in the Session Management box (lower right). You should see the names of the packages you added in the list, with check marks for the ones that are loaded.

Data section

Most data that you will analyze will come in the form of a spreadsheet, which organizes data in rows and columns. Excel format is the most common format for spreadsheets. But a simpler form, called the "comma separated value" or csv format, is loaded into R with ease. Any Excel file that you have can be saved in .csv format, but be careful because a .csv file can only be one worksheet of an Excel file.

The data section of a script reads in and manages the data. R has its own format for data, so we need to read or import the data from its original ".csv" file and create a "dataframe" in R.

How to enter a small data set "by hand"

Before we import the spreadsheet data, note that it is possible to enter data "by hand" into R, putting the numbers directly in the script. The first few lines in the data section show you how:

```
# Data can be entered directly in your script
# Simple example: each variable is created as a vector
age = c(25, 30, 56)
gender = c("male", "female", "male")
weight = c(160, 110, 220)
# Assemble the variables into a data set
mydata = data.frame(age,gender,weight)
# take a look at the data
mydata
```

This data set will have three people in it (N = 3), and three variables: age, gender, and weight. We enter each variable as a vector (that's what the c(...) means), and then put them together into a single data set (last line above). A *vector* is an ordered list. (A *matrix* is an ordered set of vectors.) Note that the first person in the data set is aged 25, male, and weighs 160 pounds. Obviously, it is important to enter the numbers in the correct order.

Highlight the above lines in your script and hit the Run button. Now look under Data in the Workspace (upper right). You should see an entry for the new data frame, mydata. If you click on it, a new window will open in your script editor area, showing you the data in a spreadsheet-like format. If you click on the blue button next to the mydata name, you will see a snapshot of the variables names and data.

You can also create data using a number generator, and this is very useful for running simulations. A simulation is when you create a fake dataset from a process that you determine, add some random "noise" to the data, and then see if your statistical technique for estimating parameters recovers the "true" parameters that you used to generate the data in the first place.

For example, the following code generates a data frame of 1000 observations. (The runif command generates random numbers using the uniform distribution, and the set.seed command will ensure that each time you run the command you will get the same random numbers, unless you change the "seed" that is the number 1534 in this example.)

```
set.seed(1534) # For replication
N = 1000 # Population size
X0 = runif(N) # Value of intercept value
X1 = runif(N) # Value of explanatory value
Xerr = runif(N) # Value of error term value
Y = X0 + 4*X1 +Xerr # Value of outcome Y
samp = data.frame(X0,X1,Y) # data that might "observe"
```

Notice in your environment window that the dataframe *samp* has three variables and 1000 observations.

In most applications, you will be obtaining your data from some external source, often as a spreadsheet.

Importing data from a spreadsheet

Let us import the California test score data used by Stock and Watson's *Introduction to Econometrics*. This data set consists of information on 420 elementary school districts in California, including average 5th-grade standardized test scores, student-teacher ratios, etc. Our goal (eventually) is to determine whether class size reductions increase test scores. A description of the data is provided in the appendix to this booklet.

The spreadsheet we will import is *caschool.csv*. A .csv file is a data set whose entries are separated by commas. As you know, in Microsoft Excel, the default file extension is typically .xls or .xlsx. It is possible to read Excel files directly into R, but when dealing with spreadsheet data, it is best to save a copy as a .csv file, because comma separated value files are simpler and less prone to formatting issues.

If you are in Excel and want to save an Excel spreadsheet as .csv, before saving it make sure the first row contains the names of the variables (no spaces in the names!) and the data start on the second row. Then use "save as" and find the CSV option. This will only save the active worksheet in an Excel file. (Remember, some Excel files have multiple worksheets; when you save as csv, only the open worksheet is saved. Also, the csv format stores cells as numbers/values, and does not save formulas.)

IMPORTANT: Always read the data into R using a read command in your R script.
DO NOT click on the .csv file to open it.

Here is how to read or load a csv file, if the working directory has been properly set to point R to the folder where the csv file is located. The command to read the csv file will only work if you have properly set your working directory and if the file caschool.csv is actually in your working directory. Use Windows Explorer or Mac Finder to double check.

Highlight and run the following line of code in the script:

```
caschool = read.csv("caschool.csv",header=TRUE,sep=",")
```

Alternatively, we can import the data directly from the Stock and Watson website, where it is in Stata .dta version. (Stata is another statistical computing software favored by economists.) Luckily, we have a command read.dta that can read the file into R format.

```
caschool <- read.dta("http://wps.aw.com/wps/media/objects/11422
/11696965/datasets3e/datasets/caschool.dta")
```

Notice the equivalent use of "=" and "<-" to serve as the assignment operator.

Look under Data in the Workspace (upper right). You should now see an entry for the new data frame, *caschool*.

This simple command read.csv(...) line has several features you will see in R over and over. First, the command creates a new "object", which is called caschool. In this case caschool is a data frame, which is what we call data sets in R. Second, in the parentheses of the command that created caschool, we have several arguments separated by commas (this is typical of many R functions):

- "caschool.csv" is the name of the csv file (in quotes)
- header=TRUE tells R that the first row contains the variable names
- sep="," tells R that the data are comma separated (not strictly necessary here).

With the read.dta command we do not need these options since the data is already formatted in Stata.

Under Data in the Workspace (upper right), click on the data name, *caschool*. This should open the data set in what looks like a spreadsheet in the script editor window. Take some time to examine the spreadsheet. (If you click on the blue button, this will display the variable names and a small preview of the data).

The units of observation (school districts) are in the rows, and variables are in the columns. Most of the values taken by variables are "numeric," in the sense that each individual value is a number, but some contain "strings" (words, etc.), such as county and district. When R reads a variable, it takes its best guess as to what kind of variable it is; string variables like county will typically be treated as factor variables. A factor variable is categorical, and qualitative (not quantitative): thus an observation (school district) is either in Santa Clara County, or in Los Angeles County, or in one of the other counties. A factor variable is actually a vector of integer values with a corresponding set of character values, called the levels, which are called whenever the factor variable is displayed.

Making new variables

IMPORTANT: When we refer to a variable in R code, the format is the name of the data frame (caschool), then the $ sign, followed by the variable name (smallclass), all with no spaces. Hence caschool$smallclass is the variable *smallclass* in the dataframe *caschool*. This way R knows which data set to look in, if you have more than one. (An exception is commands that allow you to specify the dataset.)

Now let us create a new variable in our data set. Highlight and run the next line:

What is wrong with this code?

```
# Natural log of earnings
acs$lnearn = log(incwage)
# potential years work experience
exper = acs$age - acs$educ_years - 5
```

Answer: In creating the variable *lnearn*, R will not know where to find the variable *incwage*; in creating *exper*, R will not know what data set to put the new variable in, so it will not be accessible for further calculations.

Here is the fully correct code:

```
# Natural log of earnings
acs$lnearn = log(acs$incwage)
# potential years work experience
acs$exper = acs$age - acs$educ_years - 5
```

Note that the second block of code correctly creates a variable that is the natural log of earning, then a variable measuring "potential experience," and both are properly "addressed" to be in the the acs data frame.

```
casc
hool
$sma
llcl
ass
= ca
scho
ol$s
tr<2
0
```

This creates a new variable in the dataset called smallclass, which takes the value TRUE if the student-teacher ratio (str) is less than 20, and FALSE otherwise. We call such a variable a binary or dummy variable.

Click on caschool again and look at the data. The very last variable is the new variable smallclass. Note that it takes values of "TRUE" or "FALSE". In many applications, R actually treats TRUE as the number 1, and FALSE as the number 0.

You can also create new variables using mathematical formulas, such as `caschool$strsq = caschool$str^2.` Math expressions in R are generally similar to Excel.

Factor variables can be created from numeric variables. Here are a couple of ways to create a factor variable (for a data frame called health):

```
health$California = health$state_abb=="CA"
```

```
health$CA = factor(health$state_abb=="CA", labels = c("R
est of USA", "CA"))
```

Both lines create identical binary variables (TRUE if in state of CA, FALSE otherwise)... the second one just labels the values in a more informative way than FALSE/TRUE. If you wanted to create a dummy variable for the three West Coast states, the following command would work:

```
health$westcoast <- (health$state_abb=="CA" |
                     health$state_abb=="WA" |
                     health$state_abb=="OR")
```

The | symbol stands for “or” (while the & symbol, not used here, stands for “and”).

Obtaining data from a server on the Internet

Very often you will be able to download data directly from a website into R. That is because programmers have written R code packages to interact directly with web servers. As noted earlier, Stock and Watson (and their publisher, Pearson) have posted the data used in examples in the textbook (and in this guide) online. For example, the data on California schools is stored online in the Stata format (with file extension .dta). This file can be read with the read.dta command. If you did not try this command in the script, run it now:

```
caschool <- read.dta("http://wps.aw.com/wps/media/objects/11422/11696965/datasets3e/datasets/caschool.dta")
```

and make sure that the new dataframe shows up in the environment window.

Let us try to obtain data that are more interesting from the Internet. One very easy-to-use R package is the WDI package. This package interacts with the World Bank’s World Development Indicators website, and can be used to extract data directly from the website into R.

The script *t_rbasics.R* contains code to access World Bank data. Find the relevant section of code. The first command (which is very long) defines a set of 12 variables to be downloaded from the World Bank website. The list gives the World Bank’s names for the variables, and then after a hashtag inserts a more descriptive comment of what the variable means. (It is sad to learn some of the acronyms: IMRT stands for the infant mortality rate, a key indicator of development.)

```
# Create a list of variables to import

wdilist <- c("NY.GDP.PCAP.PP.KD", # GDP per capita,
               # PPP (constant 2005 intl $)
 "SP.POP.GROW", # Population growth (annual %)
 "SP.POP.TOTL", # Population, total
 "SP.POP.TOTL.FE.ZS", # Population, female (% of total)
 "SP.URB.TOTL.IN.ZS", # Urban population (% of total)
 "SP.POP.BRTH.MF", # Sex ratio at birth
                   # (females per 1000 males)
 "SP.DYN.LE00.IN", # Life expect at birth, total
                   # (years)
 "SP.DYN.LE00.FE.IN", # Life expect, female (years)
```

```
 "SP.DYN.LE00.MA.IN", # Life expect, male (years),
 "SP.DYN.IMRT.IN", # Infant mortality rate
 "SP.DYN.TFRT.IN") # Fertility rate (births per woman)
```

Notice the first open parenthesis after the c, and the final close parenthesis on the last line before the hashtag. Notice how the hashtags are used for comments to explain each variable.

The next block of code instructs R to go to the Internet via the WDI command and get the values for the list of variables, for all countries and regional aggregates, for the year 2015.

```
# Extract latest version of desired
# variables from WDI.
# This may take a few minutes,
# depending on connection speed

wdim = WDI(country="all", indicator = wdilist,
       extra = TRUE, start = 2015, end = 2015)
```

It may take a minute or two for the WDI data to be downloaded from the web. Obviously, your computer must be connected to the Internet to access data remotely.

The gdata package in R has a command rename.vars that can be used to rename the variables with something more intelligible.

```
# Rename the variables
wdim <- rename.vars(wdim,c("NY.GDP.PCAP.PP.KD",
    "SP.POP.TOTL"), c("GDPpcUSDreal","population"))
wdim <- rename.vars(wdim, c("SP.POP.TOTL.FE.ZS",
     "SP.URB.TOTL.IN.ZS"),c("femaleperc","urbanperc"))
wdim <- rename.vars(wdim, c("SP.POP.BRTH.MF",
    "SP.DYN.LE00.IN"), c("sexratiobirth","lifeexp"))
wdim <- rename.vars(wdim, c("SP.POP.GROW"),
     c("popgrow"))
wdim <- rename.vars(wdim, c("SP.DYN.LE00.FE.IN",
    "SP.DYN.LE00.MA.IN"), c("lifexpfem","lifeexpmale"))
wdim <- rename.vars(wdim, c("SP.DYN.SMAM.MA",
    "SP.DYN.SMAM.FE"), c("smammale","smamfemale"))
wdim <- rename.vars(wdim, c("SP.DYN.IMRT.IN",
    "SP.DYN.TFRT.IN"), c("infmort","fertility"))
```

Each line starts with the data frame name, wdim, followed by the <- sign. Again, R users often prefer <- to =, for reasons that economists cannot explain. Perhaps the idea is that stuff on the right will transform or turn into what is on the left. So the dataframe wdim is created or replaced by the operations to the right of the <- sign.

Notice that we repeat the rename command several times. We could have created one long list, and renamed all the variables, but we might have risked getting the order wrong, and then we would rename a variable into a name that was for another variable. To avoid confusion, we prefer to repeat the command and just rename two variables at a time. Sometimes in coding it pays to be cautious instead of concise.

There is a variable in the data frame called region, which assigns a region for every country. We want to remove observations that did not correspond to a country, but instead give data for the aggregate of the region. So we clean the data frame by taking out the observations that are aggregates for the regions, thus leaving only countries.

```
# Take out the entries that are aggregates
# (eg East Asia) and not countries
wdim <- subset(wdim, !( region=="Aggregates"))
```

This is a good example of the subset command. Here the "not" symbol ! is used to subset the data as long as region is not equal to "Aggregates." The double == sign here is used to specify that *region* must be equal to a specific value.

The region variable is a factor variable in R. Recall that a factor variable is a vector of integer values with a corresponding set of character values, called the levels, which are called whenever the factor variable is displayed. The levels are where the string (text) names for the regions are stored. A factor variable format greatly economizes on size of the data frame. Suppose there were 100 countries in 8 regions each observed for 50 years. That would mean 5,000 observations in the data frame. If the region variable were a regular string variable containing the full text of the region, and supposing the text averaged 20 letters, then just storing the names of the regions for the 5,000 observations would involve 5,000 x 20 = 100,000 characters. But by storing each region as a factor variable, a single-digit number (from one to eight), and then having levels equal to the region names, one for each number, the data frame only needs to have 5,000 + 160 = 5,160 characters. The amount of data stored is reduced by a factor of 20!

Use a subset of observations or variables

Sometimes it is convenient to create a new sample that consists of just a subset of the observations, say with some set of characteristics, or a subset of the variables. The next two commands show examples of this. The first creates a data set consisting of the African countries in the wdim data. The line after this one creates a new data set with just the country names and the two income-related variables.

```
# Create a new data set that is just the African
# countries in wdim
wdi_africa <- subset(wdim, region=="Sub-Saharan
      Africa (all income levels)")
# Create a new data set that is just the income
# variables in wdi
wdim_income <- wdim[c("country", "GDPpcUSDreal", "femalepe
rc")]
```

The Africa subset should have observations for 47 countries. The income subset should have 212 observations for the three variables selected. You can confirm that by looking at the Workspace or Environment frame. The region variable used in the first command is a factor variable with labels for the different levels. How to find out what string corresponds to what region? Use the command:

```
levels(wdim$region)
```

An alternative to naming a new data set is simply to select the desired subsample within the statistical procedure, e.g., lm(Y ~ X, data=subset(…)). Future tutorials will demonstrate this technique.

Head-scratching problems in R

Sometimes R does things that are very hard to understand, and sometimes R does not do things that you think it should be doing. Here are the two most important (and common, in our experience) problems you may need to troubleshoot.

Notice the command prompt > in the Console area (lower left). The most important typical problem is when you run code that contains an open parenthesis (, or open brace {, or open bracket [, or open quote ", and then do not close it in the same block of commands that you have highlighted. R now "thinks" you have opened a long command. If it does not receive the close parenthesis) or bracket] or brace }, it will simply "wait" and will indicate it is waiting with a + sign in the console area instead of the command prompt >. Any other commands you enter by highlighting and running will simply not be recognized. R thinks they are now part of an even longer command, and continues to wait for a close parenthesis.

To solve this problem, move your cursor down into the Console area next to the + sign, and hit the Esc (escape) button in upper left of your keyboard. Alternatively, type and run a close parenthesis (or bracket or brace or ") to get out of limbo. (If you are a fan of the TV show *Lost*, you will remember the command prompt > and perhaps even the numbers that Locke and Desmond had to enter…)

This problem is closely related to another problem that especially affects some Mac users. When you cut and paste code, say from a pdf file, sometimes when you paste it into R Studio, the code has "hidden junk." When you run the code, you will see that "junk" appear in the Console area. Alternatively, you might see text in the Console, or part of the command. You will be tempted to say some junk yourself: *&$%)&%!!!!! Here, you have asked R to "run" junk commands, and R has no idea what to do. You will get an error message or the plus sign.

Unfortunately for Mac users, solving this problem is not so easy, because you cannot "see" the junk. In some apps, holding down Shift+Option+Command+V will paste without formatting. You may also want to paste first into a simple TextEdit program in the Mac, and then cut and paste again. Sometimes that will "strip" away junk hidden formatting. As a last resort, you type the commands again yourself. Sigh. Windows users can use Ctrl-Shift-V, which will paste without formatting, or cut and paste into Notepad (in Accessories) and then cut and paste again.

One last comment. When you include code with quotation marks or double quotation marks, note that R may be picky about straight vs. "curly" symbols. For example, on a Mac:

```
# straight quotation marks work:
(female==1 & race=="White")
# curly quotation marks do not:
(female==1 & race=="White")
```

Clean up a workspace; save and retrieve data

By now you may have a lot of data sets and results in your workspace (upper left window in RStudio). You can remove everything by hitting the Clear button (the little broom). CAUTION: This will remove everything from the workspace! You will then have to re-create any data sets etc. before doing further analysis.

The good thing is that you have a script that can reproduce your data sets exactly. That's why we clear the workspace at the start of every script. So: no worries.

Run the following command in the script, and notice that the datasets disappear from the workspace:

```
rm(list = ls())
```

All of the scripts you write should have this command at the very beginning: it is good practice to make sure you clear all datasets, vectors, and functions before running a script, to prevent your script from accidentally making use of a data frame "left over" from an earlier session.

If you want to get rid of just a few of the data sets in the workspace, you can use a variant of the rm command and list the dataframes you want to remove.

Saving a dataframe in R

Generally, it is better to create datasets from the original csv files every session. This is best practice because it makes results entirely replicable. But you can save a dataset to a file on your computer, and retrieve it later. This

can be useful if you are using big data and the data processing takes a lot of time.

The last few lines of the tutorial show how to save the data to a file and retrieve it using the save and load commands. Note that the data set will have its original name when you retrieve it (wdim in this case).

You can also use the command write.csv to save data in an Excel spreadsheet readable format. This can be helpful if you are sharing the data with other users.

Finishing your R session

When you are finished and go to exit RStudio, it will ask you if you want to "save workspace image." Usually it is not necessary to do this, because each script will import and analyze all the data from scratch each time. Make sure you save your script. A good habit is to update the name of a script with a new number when you have done significant work, so play_script_MK_v1.R then play_script_MK_v2.R and so on.

Summary

Key R commands (functions):

- setwd: sets the working directory
- library: loads packages
- options(scipen = …): sets the number format
- read.csv: reads a .csv data set into R dataframe
- read.dta: reads a .dta Stata data set into R dataframe
- rename.vars: rename variable name in a dataframe
- rm: remove objects from the workspace
- save: save a data frame or other objects to a file
- subset: create a subset of a dataframe according to some condition
- load: retrieve a saved data frame or other object
- write.csv: save a data set in csv format

Key points/ concepts:

- Working directory must be the complete "path" to your folder.
- Packages need only be installed once, but must be loaded (library) every time you start R and want to use that package.
- Variable names format: dataframe$variablename
- Create new variables using = and logical operators.
- Factor variables are categorical, such as county of residence.

- If R “hangs” or gets stuck, try moving the cursor to the console and hit the esc button.

3. Descriptive statistics

In this tutorial, you will learn how to:

- Create tables that describe the data you have loaded in R
- Create frequency tables
- Create tables of means of variables according to categorical variables (crosstab)
- Conduct t-test of difference in means

Where to find what you need:

- Downloads: R tutorial scripts and data files `http://rpubs.com/wsundstrom/home`

Introduction

Once data has been loaded into R, it is time to analyze it. Analysis generally takes three forms: tables of descriptive statistics and comparative statistics, graphical displays in the form of charts (also known as figures or plots), and regression analysis.

Getting started

Open RStudio and find and open the script *t_rdesc.R* in your script editor. Run the Settings section and the Data section. Once again, note that to run this script on your computer you will need to edit the setwd(...) command for your own working directory (folder). That is, change the part in quotations to <u>your</u> working directory folder (and not Sundstrom's or Kevane's) in the following line of code if you are a Mac user or the line with C: if you are a Windows user:

```
setwd("/Users/yournamehere/econ_42")
```

The Settings section, when run, will load the packages and set a few parameters for display of results. The data section will import the data. This script will load three datasets: the CA schools data, the WDI data, and the CPS earnings data. To load the data, highlight and run all of the code through the Data section (to just before the Analysis section). In the Workspace frame (upper right), you should see three data frames, caschool, earn, and wdim. These are the three data sets to be analyzed.

Note that the function cse( ) is used to correct the standard errors (see later chapters).

Descriptive statistics with caschool dataset

A table of descriptive statistics usually includes the mean, standard deviation, median, etc. We can create such a table using stargazer, one of the packages that we loaded. Run the following line of code:

```
stargazer(caschool, type="text", median=TRUE,
     digits=2, title="CA school data set")
```

The option median=TRUE tells stargazer to calculate the medians, as well as the other stats, which are done automatically. If you want to control exactly which summary statistics are calculated, replace `median=TRUE` with `summary.stat=c("n", "mean", "sd")`, to get the number of observations, mean, and standard deviation. You can add other statistics (see stargazer help online).

In the console (lower left) you should see a table of numbers. Take a look, scroll through, and make sure you can confirm that the mean district test score *testscr* is 654.16 and the median student-teacher *str* ratio is 19.72.

IMPORTANT: Note that some variables are not included in the table, such as county. Why not? The county variable is not numerical; it consists of the names of the counties. As noted above, what R has done here is make county into a "factor" variable when the data was read into R. The mean of a factor variable would be meaningless! No pun intended?

The mean of smallclass is 0.57. What does this tell you?

Select specific variables: We can obtain descriptive statistics separately for selected variables. You can do this by running the next command in the script (try it).

Select a subsample: Often we will want to run our analysis on a subset of the observations that have some specific characteristic(s). The easiest way to do this in R is using the subset(...) function. Examine and then run the next line of code:

```
stargazer(subset(caschool, smallclass==1),
      type="text", digits=2,
      title="Schools with student-teacher
```

```
        ratio less than 20")
```

The code replaces the name of the data set (caschool) with the following: subset(caschool, smallclass==1). This uses only the observations that have the property smallclass equal to 1.

The double == sign is used to indicate that you want only the observations where the condition is exactly true.

As mentioned previously, use the symbol "&" for “and” and the "|" for “or” for more complex subsetting, such as

```
acs_new = subset(acs,female==0 & (race=="White" | race==
"Hispanic")).
```

Based on this command, can you tell who will be included in the sample? If you were to run a command like this, note that here a new data frame called acs_new is created; you would see it in the Workspace environment window with fewer observations than the original data frame acs.

For more information on subsetting, see the website `http://rprogramming.net/subset-data-in-r/`.

Frequency tables: The table command will provide a count of the number of observations in each category of a factor variable, such as county. Run the table commands and see what happens.

```
# frequency tables by county
table(caschool$county)
table(caschool$county, caschool$smallclass)
```

Crosstabs: Probably the most common data analysis done in the social sciences is to compute means of variables for different groups. In R the doBy package is useful for this purpose; it has a function called summaryBy that produces crosstabs.

```
summaryBy(testscr ~ smallclass, data=caschool,
      FUN=c(mean, min, max), na.rm=TRUE)
```

The ~ sign (the tilde) indicates that the variable that follows is the categorical variable. FUN=c() indicates the various statistics to compute. na.rm=TRUE tells R to ignore missing values.

Testing for difference in means

Social scientists are always comparing average of variables for two (or more) groups. Is this group different from that group? Just looking at the two means is not enough. If the sample size from which the means are calculated is very small, then the different in the means may just be the

```
Welch Two Sample t-test

data:  testscr by smallclass
t = -4.04, df = 404, p-value = 0.00006333
alternative hypothesis: true difference in mea
ns is not equal to 0
95 percent confidence interval:
 -10.96  -3.79
sample estimates:
mean in group FALSE  mean in group TRUE
              650                  657
```

result of chance: one sample drew a few high values and the other sample for the other group a few low values.

Moreover, the variance of the observations in the samples for the two groups may be very high. In order to determine whether the difference in means is statistically significant, social scientists calculate a t-statistic and examine the probability of observing a t-statistic of the value or higher. The null hypothesis that there is no difference in the means between the two groups is rejected when the probability (p-value) of obtaining a t-statistic that one actually obtains is very low, if the null hypothesis were true.

R can calculate t-statistics and p-values in a snap. Here is the command, for the caschool data.

```
t.test(testscr~smallclass, data=caschool ,
       FUN=c(mean), na.rm=TRUE)
```

The results are pretty clear: smaller classes have higher test scores. The t-statistic is 4.04, with an associated p-value of .00006 which is tiny. The small classes have average score of 657 and the larger classes have average test score of 650.

Descriptive statistics with WDI dataset

Let us turn to the WDI data. The script has already loaded the dataset and transformed some of the variables. You can use stargazer to get a table of descriptive statistics.

```
stargazer(wdim, type="text", median=TRUE,
     digits=2, title="WDI data set")
```

We could use the same command with a subset of the data to get a table of descriptive statistics for just one world region.

```
stargazer(subset(wdim,
       levels(wdim$region)=="Middle East &
       North Africa (all income levels)"),
```

```
       type="text", digits=2,
       title="WDI data for Middle East")
```

Now use summaryBy to create a table of the average of *femaleperc* (Population, female (% of total)) for each region.

```
# Table by region of % female
summaryBy(femaleperc ~ region,  data=wdim,
         FUN=c(mean),na.rm=TRUE)
```

This is a straight average of the percent female for each country in the region, and is not weighted by the population size of each country. You will see something like this (the output has been cleaned up a bit, and the data here is from observations for 2010).

```
region                    femaleperc.mean
1 East Asia & Pacific              50.1
2 Europe & Central Asia            51.2
3 Latin America & Caribbean        50.7
4 Middle East & North Africa       45.8
5 North America                    50.6
6 South Asia                       49.2
7 Sub-Saharan Africa               50.2
```

South Asia has almost two percentage points, and East Asia almost one percentage point, fewer females than Europe, North America or Latin America. Since the populations of both regions are larger than one billion persons, these one or two percentage point differences have been the basis for estimations that there are about 100 million "missing women" in those regions, arising increasingly from sex selective abortion as parents prefer to have sons rather than daughters.

Descriptive statistics with CPS earnings data

The script also has commands to read in the CPS earnings data used in Stock and Watson's Table 3.1. These lines are towards the top of the script in the data section.

```
earn = read.csv("cps92_08.csv", header=TRUE,
       sep=",")
```

You can create a new variable that adjusts 1992 values to 2008 dollars using the CPI.

```
earn$realahe = ifelse(earn$year==2008, earn$ahe,
       earn$ahe*215.2/140.3)
```

Notice this uses an ifelse command to carry out the transformation. Can you figure out what the line of code is doing? You should write some code to carry out some t-tests for differences in means.

Summary

Key R commands (functions):

- stargazer: creates tables—in this case descriptive statistics (requires package stargazer)
- subset: creates a subsample of a data set satisfying some condition(s)
- summaryBy: calculates summary statistics for different groups (requires package doBy)
- t.test: calculates t-statistic and p-value for difference in means

Key points/ concepts:

- Subsample of data: subset(dataset, condition)

4. Graphs

In this tutorial, you will learn how to:

- Create some useful plots:
  - Histograms
  - Box plots
  - Scatter plots

Where to find what you need:

- Downloads: R tutorial scripts and data files `http://rpubs.com/wsundstrom/home`

Introduction

Visualizing your data is an important first step in any analysis. By examining plots, we can often identify important relationships in the data as well as some potential sources of problems, such as outliers. R is renowned for its graphics capabilities.

There are many options for creating plots in R. We are going to use a function called ggplot, which is part of the ggplot2 package. We use the California schools data provided by Stock and Watson and the WDI data from the World Bank to generate histograms, box plots, and scatter plots.

Getting started

Open RStudio and find and open the script *t_graphs.R* in your script editor. Once again, note that to run this script on your computer you will need to edit the setwd(...) command for your own working directory (folder). That is, change the part in quotations to <u>your</u> working directory folder (and not Kevane's or Sundstrom's) in the following line of code:

```
setwd("/Users/yournamehere/econ_42/data")
```

Now you load the packages, run other settings, and import the data. To do that, highlight and run all of the code through the Data section (to just before the Analysis section). Examine the table of descriptive statistics to remind yourself what is in the data set. Note that the function cse( ) is used to correct the standard errors (and will be used when presenting results of regression analysis).

Draw a histogram

A histogram is a chart of the distribution of a variable in the sample, with the height of each bar showing the number or percentage of observations in various ranges of values.

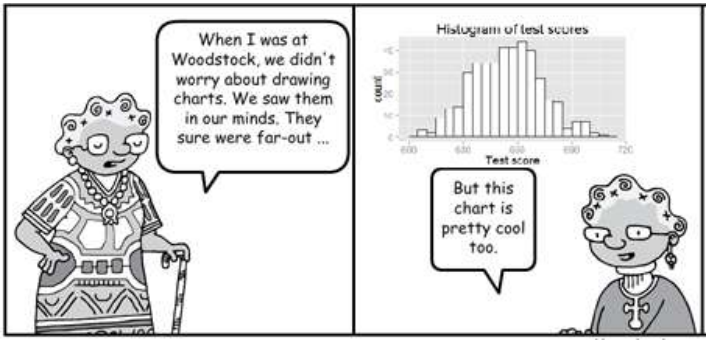
Run the following line of code:

```
ggplot(data=caschool, aes(testscr)) +
geom_histogram()+
labs(title="Histogram of test score") +
labs(x="Test score", y="Count")
```

A plot should appear in the Session Management area (lower right). Click on the "Plots" tab if it does not appear.

Some things to note about the ggplot command here:

- This command has the usual R structure: a command name, with additional information in parentheses.
- Inside the parentheses, the data frame name you want to examine (caschool) comes first. (If you left out the data= part, would the histogram still run?) Then comes the aes() part, where aes stands for *aesthetic*, here interpreted as a mapping of variables to various parts of the plot. The variables to be used in the plot are here. In the histogram case, there is just one variable, *testscr*.
- Since we tell ggplot which data set to use with the data=caschool option, this allows us to name the variables without using the data set as a prefix (e.g., *testscr* instead of *caschool$testscr*).
- Further instructions are added with + signs.
- The title and axis label options create titles for the graph as a whole and for the x- and y- axis respectively. These titles should be in "quotes".



Click on the Zoom button to see a larger, scalable version of your plot. You can then copy and paste into a Word document. Click on the Export tab. You can save your plot as a pdf file or as an image file, such as JPEG, or just copy it to your clipboard to paste into a Word document.

Controlling the appearance of the histogram:

The mass of black created by this histogram is not very informative. The ggplot2 package has many ways to control the appearance of plots. For example, try running the next two lines of code in the script.

```
ggplot(data=caschool, aes(testscr)) +
   geom_histogram(breaks=seq(600, 700, by = 10),
   col="red", fill="green", alpha = .2) +
   labs(title="Histogram of test score") +
   labs(x="Test score", y="Count")
```

The second line of code tells R to use the aes information and apply a particular set of "geometry" choices for a histogram; namely, have bins going from 600 to 700 in intervals of ten, make the outline of each bar red, and have the inside of the bars be light green (change the alpha value to .8 and see what happens). If you want to make graphs look even prettier, visit the ggplot site at http://ggplot2.tidyverse.org/reference/

There are other ways to make histograms. As mentioned earlier, one of the great features of R is that as an open source software users are constantly coming up with packages to improve the software. It can, however, be confusing. The package ggplot2 is one such user written package. R comes bundled with a simpler histogram command. Just type and run:

```
hist(caschool$testscr,
      main="Histogram of test scores",
      xlab="Test score")
```

In the plot window you will see a histogram similar to the one generated by the ggplot command. The hist() function does not have as many options as ggplot, but is pretty mnemonic! Note that in the hist() command there is not a separate option for naming the data frame, so the variable that you want to use to draw the histogram has to be properly addressed, by first indicating the data frame, then the $ sign, then the variable name, or `caschool$testscr`.

Draw a box plot

Recall that a box plot, or "box-and-whiskers" plot, shows the median and interquartile range of the data for a variable. The lines extending from the box show extreme values. Dots are potential outliers. Box plots are most interesting when we compare the distribution between different subsets of the data. In this case, let us compare the distribution of test scores between districts with smaller and larger average class sizes (student-teacher ratio). Run the following command:

```
ggplot(data=caschool,aes(smallclass,testscr,
     fill=smallclass)) +
  geom_boxplot() +
  labs(title="Test score by small class size") +
  labs(x="Average class size under 20", y="Test score")
```

Note:

- The first variable in the aes() is the X-axis variable: the groups we want to compare (smallclass in this case). The second variable is the Y-axis: the variable whose distribution we are examining (testscr). The fill= option colors each box differently for each X-axis category.
- The geom_boxplot() option tells ggplot to make a box plot.
- What does the plot tell you about the relationship between test scores and class size?
- To run a boxplot, the grouping variable for the X-axis must be a factor variable with well-defined levels (and not a numeric or continuous variable).

If your grouping variable is numeric, it will have to be turned into a factor variable. Code like this can be adapted to "factorize" a 0,1 variable:

```
teach$minority <- factor(teach$minority,
        levels <- c(0,1),
        labels <- c("Non-minority", "Minority"))
```

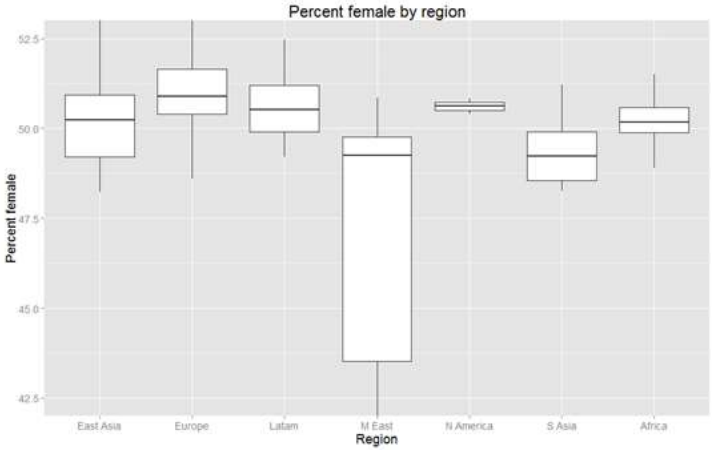
Draw a boxplot using the WDI data

We might also use the WDI package to access data from the World Bank's Development Indicators database and draw a boxplot with the data. The script already contains the code to download the data from the World Bank server. The WDI package will automatically add a variable for region that we will use for the boxplot. Run the commands for the box plot.

```
ggplot(data=wdim, aes(x=region, y=femaleperc,
       fill=region)) +
     geom_boxplot() +
     labs(title="Percent female by region") +
     labs(x="Region", y="Percent female")
```

The boxplot generated has some problems that make it unreadable. Most charts have to be cleaned up, and this boxplot is no exception. Since there are outliers at the low end of percent female for the Middle East, the y-axis scale makes it hard to visualize the box plots for the other regions. Moreover, the labels for the regions, given by the World Bank, include the annoying "(all income levels" statement for each one. Thus, the names are very long. Some code can be written to rename the labels for the regions. Recall that the region variable is a factor variable, with the labels stored in the different levels of region. The question then arises: How to change the levels of a factor variable? Here is the code for the first three changes. The final <- (which is equivalent to the = sign) defines the new content of the level, and the code the left of the <- defines which of the levels of the factor variable region to change.

```
levels(wdim$region)[levels(wdim$region)=="Europe &
       Central Asia (all income levels)"]
       <- "Europe"
levels(wdim$region)[levels(wdim$region)=="Middle
       East & North Africa (all income levels)"]
       <- "M East"
levels(wdim$region)[levels(wdim$region)=="East Asia
       & Pacific (all income levels)"]
       <- "East Asia"
```



There is an alternative way to recode or revalue the levels of a factor variable, and that is through the revalue function of the plyr package. This function can also revalue numeric values. The code is straightforward:

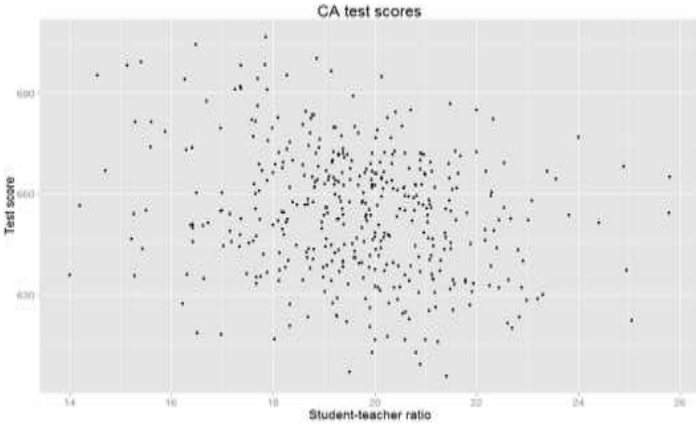
```
wdim$region <-revalue(wdim$region, c("Europe &
Central Asia (all income levels)"="Europe", “Middle
East & North Africa (all income levels)" = “M East"))
```

Another command is recode in the dplyr package.

We are now ready to redraw the boxplot. We want to rescale the y-axis and drop outliers. The way to do this is by creating a new chart, that we call chart0, and storing that as an object in R. Then the next line of code creates a different version of the chart, chart1, that sets the y-axis to be between 42 and 54 percent. You should “talk through” the code with a friend to make sure you understand what each part of the code is doing.

```
# A way to change scale of y-axis and drop outliers
chart0 <- ggplot(aes(y = femaleperc, x = region),
  data = wdim) + geom_boxplot(outlier.size=NA) +
  labs(title="Percent female by region", x="Region",
  y="Percent female")
# scale y limits
chart1 <- chart0 + coord_cartesian(ylim = c(40,54))
# display the resulting chart, which is called chart1
chart1
```

The resulting chart looks quite a bit better. We see that almost all of the countries of North America, Africa and Europe have percent of the population that is female about 50 percent, while the Middle East, South Asia and East Asia have many countries below 50 percent. Why do you think so many countries in the Middle East have such low percent female? The answer is not discrimination resulting in “missing women.” Think about the



oil-rich Gulf States. They attract migrants from around the world to work in the cities and oil fields. Those migrants are predominantly male. For some countries, the resulting population is almost two-thirds male.

Draw a scatter plot

We are usually most interested in the relationship between variables, such as test scores and class size. A scatter plot is a good way to visualize the relationship. Run the scatter plot command in the script to see the relationship between class size and test scores. Notice we now use geom_point.

```
ggplot(data=caschool, aes(x=str, y=testscr)) +
       geom_point(shape=0)+
   labs(title="CA test scores") +
   labs(x="Student-teacher ratio", y="Test score")
```

R scales the plot to the range of values for X and Y. Does the plot suggest that increasing class size (str) leads to lower test scores? There does seem to be a negative relationship.

The additional scatter plot commands show some of the other things you can do with R plots. Run each of these and see what happens. For example, if you want to include a regression line in the scatter plot,

```
ggplot(caschool, aes(x=str, y=testscr)) +
  labs(y = "Test score",x = "Student-teacher ratio")+
  labs(title = "CA test scores") +
  geom_point(shape=1) +    # Use hollow circles
  geom_smooth(method=lm)
```

For a smoothed fitted curve, use `geom_smooth(method=loess)`

We can generate a scatter plot with the WDI data. Run the following command.

```
ggplot(data=wdim, aes(x=GDPpcUSDreal,y=femaleperc)) +
       geom_point()+
       labs(title="Income and percent female") +
       labs(x="GDP per capita", y="Percent female")+
       theme(legend.position="none")
```

Summary

Key R commands (functions):

- ggplot: creates a variety of plots, including histogram, box plot, and scatter plot (requires package ggplot2)
- aes(): subcommand of ggplot, indicates variables to be graphed
- geom: indicates the geometry of the graph in ggplot

Key points/ concepts:

- Options ggplot can be used to control plot types and appearance.

5. Regression analysis

In this tutorial, you will learn how to:

- Run a least-squares regression in R and interpret the results
- Obtain heteroscedasticity-corrected standard errors and test statistics for regression coefficients
- Assemble the estimates into a nice table of results
- Retrieve and plot the regression residuals

Where to find what you need:

- Downloads: R tutorial scripts and data files `http://rpubs.com/wsundstrom/home`

Introduction

It is easy to run regressions in R. However, the standard R regression procedure has some limitations. Therefore, we use some packages that fix a couple of issues: (1) We want to correct the estimated standard errors for heteroscedasticity. (Why?) (2) We want our results presented in a table that makes it easy to compare alternative regression models.

Run a simple regression

Start RStudio and open the script *t_regbasics.R* in your script editor. Once again, note that to run this script on your computer you will need to edit the setwd(...) command for your own working directory (folder). That is, change the part in quotations to your working directory folder in the following line of code:

```
setwd("/Users/yournamehere/econ_42")
```

Then load packages, run other settings, and import the data. Highlight and run all of the code through the Data section (to just before the Analysis section). Examine the table of descriptive statistics to remind yourself what is in the data set.

Before we run the regression, let us take another look at the scatter plot: Run the ggplot command. Note again that the relationship between str and testscr appears to be weakly negative. Now highlight and run the following line:

```
reg1 = lm(testscr ~ str, data=caschool)
```

Note the following:

- lm(...) is the linear model (regression) function. The first variable is the Y variable, then any X variable(s) are included after the ~.
- The X variable, here str, is the explanatory variable or the regressor.
- We gave the regression output a name, *reg1*. In your workspace (upper right) under Values, you should now see reg1.
- Notice data=caschool in the lm command. This allows us to just use the variable names in this function without including the data set name and $ sign.

To see the actual results of the regression in your console, highlight and run the summary(...) command. In the future, we will not do this, but let us take a look. Here is the relevant part of the regression output in your console:

```
lm(formula = testscr ~ str, data = caschool)

Residuals:
    Min      1Q  Median      3Q     Max
-47.727 -14.251   0.483  12.822  48.540

Coefficients:
            Estimate Std. Error t value   Pr(>|t|)
(Intercept) 698.9330     9.4675  73.825    < 2e-16 ***
str          -2.2798     0.4798  -4.751 0.00000278 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 18.58 on 418 degrees of freedom
Multiple R-squared:  0.05124,   Adjusted R-squared:  0.04897
F-statistic: 22.58 on 1 and 418 DF,  p-value: 0.000002783
```

Consult your textbook for interpretation of most of the numbers presented in the output. For now, note that the implied regression equation is:

testscr = 698.9330 – 2.2798 str

The intercept is 699 and the slope is 2.28.

Create a table with corrected standard errors

Economists usually present regression results in a different format. The regression tables in your Stock and Watson textbook (e.g., Table 7.1) are good examples. We can use the stargazer command to make perfect regression tables. In addition, we want to use the corrected standard errors for the coefficients. Let us make the table and see what we get.

Examine the next command, but don't run it yet…

```
stargazer(reg1,
    se=list(cse(reg1)),
    title="CA school district regression",
    type="text",
    df=FALSE, digits=3)
```

Q: How do I know that these four lines are all part of the same command to R? A: Everything within the set of parentheses following the command stargazer is part of the same command.

Within the parentheses, note the following:

- reg1 is the name of the regression(s)
- se=... is how we tell R to use the corrected standard errors. If you omitted this option, stargazer would just use the uncorrected SEs. See below for more on this.
- title=... gives the table a title (put title in quotes)
- type="text" means the table will be printed as text (always use this)
- df=FALSE means the table will not be cluttered with degree of freedom numbers
- digits=3 rounds the numbers in the table to three digits after the decimal (you can change this, of course)

Now highlight and run the stargazer command (all four lines of it... make sure you include all parentheses!). Recall that if you just run part of a command, and it includes an open parenthesis, then R will be waiting for the close parenthesis, and will not give any output. See the earlier chapter for how to resolve this problem.

```
CA school district regression
==============================================
                         Dependent variable:
                     ---------------------------
                                testscr
----------------------------------------------
str                            -2.280***
                                (0.519)

Constant                      698.933***
                                (10.364)

----------------------------------------------
Observations                      420
R2                               0.051
Adjusted R2                      0.049
Residual Std. Error             18.581
F Statistic                    22.575***
==============================================
Note:                *p<0.1; **p<0.05; ***p<0.01
```

Examine the results in the console frame. The slope and intercept coefficient estimates, with standard errors in parentheses, are in the top panel. The significance codes (asterisks) are for the null hypothesis that the coefficient is zero. *** means the p-value is between 0 and 0.01, so the null can be rejected at the 0.01 = 1% level. The coefficient called "Constant" is the estimate of the Y intercept. Observations is N = 420. R-squared and adjusted R-squared give a measure of goodness of fit of the regression. "Residual Std. Error" is also known as the standard error of the regression. The F Statistic is for a test of the null hypothesis that all the slope coefficients are zero. We will discuss this later in the guide.

You can compare these results with the table of regression results we obtained without the robustness correction (above). Note that the point estimates are the same (e.g., the slope is -2.28), whereas the standard errors are different (for the regressor *str*, 0.4798 vs. 0.519). This is due to the correction for heteroscedasticity.

Correction of standard errors for heteroscedasticity:

In many statistical packages (including R), the default regression function calculates coefficient standard errors that are only valid if the error term is homoscedastic—that is, the variance of the error (unexplained part) is the same for all observations. As you learn in chapter 5 of the Stock and Watson textbook, there is no reason to think this would be true, and if it is not, the standard errors are incorrect, and so are the resulting hypothesis tests.

To fix this, a correction recalculates the standard errors: this is the function cse(…) just before the data section. Stargazer uses this to correct

the SEs. The correction seldom makes a large difference, but it is technically correct, and will make all your results completely comparable with the Stock and Watson textbook when we replicate some of their results. Hence, we will always use a correction of the standard errors.

Retrieve and plot the regression residuals

The regression residuals are the difference between the actual value of the dependent variable (Y) and the predicted value from the regression, for each observation. We will discuss in class how the residuals can provide a useful diagnostic.

Highlight and run the following commands:

```
caschool$resid1 = resid(reg1)

ggplot(data=caschool, aes(x=str, y=resid1)) +
 geom_point(shape=0)+
 labs(title="Residual against ST") +
 labs(x="Student-teacher ratio",
      y="Regression residual")
```

Do you see any pattern in the plot?

Run the last lines of code and see how the second regression and third regression can be displayed as new columns in the stargazer table. Make sure you can write out the formula for the estimated regression equations for reg2 and reg3 and explain what they mean.

Run a regression using the WDI data

Let us run a regression using the World Bank's Development Indicators database to see if there is a relationship between the income (measured as per capita real GDP) of a country and the percent of females in the country. We might imagine that poorer countries have more "missing women."

Run the regression and display the results using stargazer. GDPpcUSDreal is the only regressor in this equation.

```
regwdi1 = lm(femaleperc ~ GDPpcUSDreal, data= wdim)

stargazer(regwdim1,
    se=list(cse(reg1)),
    title="Regression of GDP and percent female",
    type="text",
    df=FALSE, digits=5)
```

The df=FALSE option tells stargazer not to display the degrees of freedom for the calculations. The digits=5 option tells stargazer to round results to the fifth decimal.

Take some time to examine the output of the regression. The variable measuring GDP per capita has an estimated coefficient of -.00006 and has

Regression of GDP and percent female

	Dependent variable:
	femaleperc
GDPpcUSDreal	-0.00006*
	(0.00003)
Constant	51.01891***
	(0.44854)
Observations	181
R2	0.15953
Adjusted R2	0.15483
Residual Std. Error	2.83512
F Statistic	33.97546***

Note: *p<0.1; **p<0.05; ***p<0.01

one asterisk, indicating statistical significance at the ten percent level. To interpret the magnitude of the relationship, look at the underlying data.

Under Data in the Workspace frame (upper right), click on the data name, *wdim*. This should open the data set in what looks like a spreadsheet in the script editor frame. The GDP variable is in dollars, while the percent female variable is in percentages (and not in decimals). So the regression coefficient means that a one dollar increase in GDP per capita is associated with a decrease in the percent female of .00006. Alternatively, an increase in GDP of ten thousand dollars per capita reduces the percent female by more than half a percentage point. That seems pretty substantial. Of course, we need to reflect on whether this coefficient might be biased. (Obviously it is!)

An important observation to make here is that just because the slope coefficient here seems to be a "small" number does not necessarily mean the effect is small—in this case it means that the scale of X (per capita GDP) is large. We could make the regression table easier to interpret by rescaling the X variable to be in thousands of dollars. Without re-running the regression, can you tell what the slope coefficient would be in that case?

Summary

Key R commands (functions):

- lm: runs regressions
- cse: function you load at beginning of script to generate corrected standard errors (requires package sandwich)
- stargazer: creates nice tables of regression results and allows you to include corrected standard errors (requires package stargazer)
- resid: obtains the regression residuals

Key points/ concepts:

- The basic format for running a simple regression is lm(Y ~ X).
- ALWAYS report the heteroscedasticity-robust standard errors.

6. Multiple regression

In this tutorial, you will learn how to:

- Run multiple regressions in R
- Assemble the estimates into a nice table of results
- Use regression results to predict Y for hypothetical values of X
- Calculate standardized regression coefficients to judge size of effect
- Conduct a variety of hypothesis tests for joint and other hypotheses about the regression coefficients, including F tests

Where to find what you need:

- Downloads: R tutorial scripts and data files `http://rpubs.com/wsundstrom/home`

Introduction

Multiple regression is regression with more than one regressor (X variable). In R, multiple regression is straightforward. In this tutorial, you will run all the regressions for Table 7.1 of Stock and Watson (p. 238 of original 3rd edition) and put them into a single stargazer table, which will look a lot like Table 7.1. Running and presenting the regressions is the easy part: you will need to make sure you are able to interpret the regression coefficients and understand why the estimated coefficients might be biased.

Run some regressions Start RStudio and open the script *t_multreg.R* in your script editor. Make sure to change the setwd(…) command, as usual. This script loads the now-familiar CA school district data set, and runs several regressions with the district test score (testscr) as the dependent (Y) variable, and various regressors (X variables).

Scroll down to the regression commands. Note that the regressions (2)-(5) are multiple regressions, which have more than one regressor (X

variable). In R we can add regressors simply by including them with a "+" sign, as in testscr ~ str + el_pct.

Examine the stargazer command, which creates the table. Note that we include the names of all the regressions (reg1, reg2, …), followed by the SE correction for all of the regressions, in the same order: se=list(cse(reg1), cse(reg2), …).

It is very important that you pay attention to the parentheses. The parentheses in every R command must balance, in the sense that for every open paren "(" there is a later close paren ")".

Highlight and run all the lines from the beginning of the script through the entire stargazer command. Look at the resulting table in your console.

Compare the results for each column with what is in Table 7.1 of your textbook (p. 238). The two tables should contain identical numbers, at least within rounding error.

```
stargazer(reg1, reg2, reg3, reg4, reg5,
   se=list(cse(reg1),cse(reg2),cse(reg3),
         cse(reg4),cse(reg5)),
   title="Regression Results", type="text",
   column.labels=c("Simple", "ELL", "Lunch",
            "CalWorks", "Full"),
   df=FALSE, digits=3)
```

The code puts column labels in stargazer. It helps the reader. Make sure your labels are in the same order as the corresponding regressions.

Regression Results

	Dependent variable:			
	testscr			
	Simple (1)	ELL (2)	Lunch (3)	CalWorks (4)
str	-2.280***	-1.100**	-0.998***	-1.310***
	(0.519)	(0.433)	(0.270)	(0.339)
el_pct		-0.650***	-0.122***	-0.488***
		(0.031)	(0.033)	(0.030)
meal_pct			-0.547***	
			(0.024)	
calw_pct				-0.790***
				(0.068)
Constant	699.000***	686.000***	700.000***	698.000***
	(10.400)	(8.730)	(5.570)	(6.920)
Observations	420	420	420	420
R2	0.051	0.426	0.775	0.629
Adjusted R2	0.049	0.424	0.773	0.626
Residual Std. Error	18.600	14.500	9.080	11.700
F Statistic	22.600***	155.000***	476.000***	235.000***
Note:			*p<0.1; **p<0.05; ***p<0.01	

Make sure you understand how to interpret each coefficient in the table (the above table only reproduces the first four regression results), and how to do a hypothesis test or confidence interval for each variable.

You can write the table to a text file if you want to save it, rather than cut and paste from the console. In the stargazer command, you would add another option, saving to a file named filename.txt: out="filename.txt"

Prediction using regression results

We can use a regression to predict the Y value for specific hypothetical or actual values of the regressors using the predict function. Let us do this using the results of regression reg3 and predict the average test score for a district with str = 20, el_pct = 20, and meal_pct = 50.

To do this, we first create a new data frame with these values of the X variables:

```
newdata = data.frame(str = 20, el_pct = 20,
    meal_pct = 50)
```

Next, we use the predict command, telling R which regression coefficients to use (reg3) and which data frame (newdata):

```
predict(reg3, newdata)
```

Run these commands and see what the result is. The predict command will also calculate the upper and lower limits of a 95% confidence interval around the prediction. There are two kinds of confidence intervals for predictions. The first is obtained with the option interval="confidence" in the predict command. This gives us the 95% *confidence interval* for E(Y | X): that is, what is the expected test score over a large number of districts with this set of characteristics X? Because our coefficient estimates are uncertain, there is some uncertainty about this expected test score, and the CI gives us a plausible range of values.

The second kind of interval is often called a 95% *prediction interval*, and it is obtained with the option interval="prediction". The prediction interval can be interpreted as a range of values for predicting the test score in a *single school district* with these characteristics. Because of the residual variation around predicted values, the 95% prediction interval is always wider than the confidence interval of the prediction.

Run these two intervals and see what you get. Is the prediction interval in fact wider?

Obtain standardized coefficients to assess effect size

How important is the effect of a regressor (X) on Y? Note that this is not the same as asking whether the coefficient is statistically significant. We want to know how big the point estimate is, not just whether it is different from zero. We will discuss various ways of judging effect size in class, but one common approach is to "standardize" the regression coefficient. Confusingly, standardized coefficients are sometimes called "beta coefficients"—not to be confused with the coefficient symbol β, which we call beta.

The standardized slope coefficient tells us the expected marginal effect of a one-standard-deviation change in X on Y, measured in standard deviations of Y. We rescale the coefficient by measuring both variables in their sample standard deviations rather than their units. In more everyday language, this tells us how much a "typical" change in the X variable will move Y, other things equal. We allow the data to tell us what is a "typical" amount of variation.

The standardized coefficient is just one way of judging magnitude of effect, and is more appropriate for continuous variables than for binary variables.

There are packages that calculate standardized coefficients, but here we will just do it in the script by obtaining the coefficient estimate and multiplying it by sd(X)/sd(Y).

Run these lines and see what happens:

```
coef(summary(reg5))["str","Estimate"]*sd(caschool$str)/s
d(caschool$testscr)
coef(summary(reg5))["el_pct","Estimate"]*sd(caschool$el_
pct)/sd(caschool$testscr)
```

F-tests

Joint hypotheses are hypotheses about more than one "restriction" on the coefficients. For example, we might want to test the null hypothesis that in regression (5), the coefficients on meal_pct and calw_pct are both equal to zero. These variables are intended to control for low-income school districts; if both coefficients were zero, it would suggest that the relative poverty of a district does not have a significant impact on average test score, controlling for the other factors in the regression.

Note: It is incorrect to test this joint hypothesis with two separate t-tests.

To implement a joint hypothesis test, we can use the function lht( ) which is part of the car package. To see how it works, run the next command:

```
lht(reg5, c("meal_pct = 0", "calw_pct = 0"),
      white.adjust = "hc1")
```

Examining this command, do note that

- reg5 tells lht which regression results to use.
- c("meal_pct = 0", "calw_pct = 0") expresses the null hypothesis, with each restriction in quotation marks.
- white.adjust = "hc1" assures that the heteroscedasticity-corrected standard errors are used for the test. Always include this.

```
Linear hypothesis test

Hypothesis:
meal_pct = 0
calw_pct = 0

Model 1: restricted model
Model 2: testscr ~ str + el_pct + meal_pct + calw_pct

Note: Coefficient covariance matrix supplied.

  Res.Df Df      F    Pr(>F)
1    417
2    415  2 290.27 < 2.2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.'
0.1 ' ' 1
```

Interpreting the results requires careful understanding of the nature of the hypothesis tests. For now, note that the null hypothesis is stated for us, and that the F-stat for the test is 280.66, with the p-value next to it (<2.2e-16). Df = 2 tells you the number of restrictions being tested.

Obviously, with such a low p-value we can reject the null hypothesis. We can test a wide variety of hypotheses using lht. Run the remaining lht lines in the script and see if you can understand what each test is doing.

Multiple regression with WDI data

In the previous chapter, we ran a regression of income (measured as per capita real GDP) on the percent of females in the country. The script contains the commands to estimate a multiple regression version of that equation, where we include population, latitude, infant mortality and fertility.

You have already run the commands to import the WDI dataset into R. The script includes the usual commands to clean up the dataset, renaming the variables, etc. Also in the Data section are some commands to rescale two of the variables. The third command transforms the variable latitude from a factor variable to a numeric variable.

```
wdim$GDPpcUSDreal = wdim$GDPpcUSDreal/1000
wdim$population = wdim$population/1000000
wdim$latitude =as.numeric(wdim$latitude)
```

Note: sometimes the latitude variable will be loaded into R as a factor variable. In that case, the levels of the factor do not correspond to the numbers of the latitude (which are instead the labels of the factor). So the following line converts the labels to be a string, and then turns the string into a number for the latitude:

```
wdim$latitude=as.numeric(as.character(wdim $latitude))
```

In the Analysis section, run two regressions and display the results using stargazer.

```
regwdi1 = lm(femaleperc ~
       GDPpcUSDreal+population+
        latitude, data= wdim)
regwdi2 = lm(femaleperc ~ GDPpcUSDreal +
       population+ infmort+ fertility+latitude,
       data= wdim)

stargazer(regwdi1,regwdi2,
       se=list(cse(regwdi1),cse(regwdi2)),
       title="Regression of percent female
       on various correlates", type="text",
       df=FALSE, digits=3)
```

Take some time to examine the output of the regression. The variable measuring GDP per capita is roughly the same magnitude as before (an increase in GDP per capita of $1000 is associated with a decrease in the percentage female of .1). But now it is more statistically significant (there are two and three asterisks). The variable latitude is also statistically significant. Can you interpret the meaning?

Once again, we repeat the admonishment: R has no trouble calculating regression coefficients, but we need to use judgment and common sense to determine whether the coefficients are likely to be unbiased.

Summary

Key R commands (functions):

- lm: can be used for multiple regression too
- stargazer: can combine results from several regressions
- predict: use regression results to predict Y for different data
- lht: flexible function for linear hypothesis tests (requires package car)

Key points/ concepts:

- The format for running a multiple regression is lm(Y ~ X1 + X2 + …)
- ALWAYS use the heteroscedasticity-corrected standard errors.
- We can obtain standardized coefficients to judge effect size.
- For a joint hypothesis test using lht, make sure to include the option white.adjust.
- While R can calculate regression coefficients, only you can interpret the regression to determine if there is bias.

7. Merging and reshaping data sets

In this tutorial, you will learn how to:

- Merge data sets by matching observations
- Reshape a dataset from wide to long

Where to find what you need:

- Downloads: R tutorial scripts and data files `http://rpubs.com/wsundstrom/home`

Introduction

Although the emphasis of this guide is on how to analyze data once you have it, a big challenge in data science is obtaining, managing, and manipulating data before you even start analyzing it. This tutorial covers one of the most important, and dangerous, data operations: merging two datasets.

Very often, we draw data from multiple sources and then want to merge the different data sets by matching on a particular variable or variables. For example, you might have company sales data by zip code and want to match it to census or other data also arranged by zip code. The idea is to create a new data set that has variables from both data sets.

In order to practice the merge command, we merge the WDI data with another dataset from the Internet. Economists frequently merge country-level data from different sources.

The R command merge(…) does the job, but there are several things to be careful about!

There may be some cases where an observation is in one of the data sets but not the other. You need to decide: Do you want to keep only the countries that are in both data sets (intersection), or include all of them (union)?

The variable you merge on must match perfectly. In our case, it will be a country name. Be aware that "USA" will NOT match with "U.S.A." or with "United States"! This is one reason to prefer standard numerical codes for countries or other entities, rather than names.

There may be non-unique matches. For example, suppose you had data on individuals, including their state of residence, and wanted to assign state-level variables to each individual. Then each state would match to more than one individual. This does not happen in our case.

This section also introduces techniques to reshape data. Very often, we have multiple observations per unit. We observe seven test scores for one student. We observe heights of six children in a family. The data can be stored in the "wide" fashion, where each row is a student or family, and we have columns for each test score or for each child. Alternatively, the data might be stored in the "long" fashion, where there are three columns: one for the unit, one for the observation (1-7, or 1-6), and one for the measure (test score or height). In the long version, the identifier for the student or family is repeated in each row that corresponds to that unit. Different R functions will be more or less adapted to data in the wide or long format, so it is useful to know how to reshape a dataset.

Getting started

Start RStudio and open the script *t_merge.R* in your script editor. Make sure to change the setwd(…) command, as usual. Run the commands in the Settings section, and also run the first part of the Data section that loads the now-familiar WDI dataset obtained from the World Bank server. The second data set is coming from a csv file available on the Internet, through the Correlates of War project from the University of Maryland. (If you are a political science major, you should definitely explore the website for this project!) We will use these data to have a measure of the percent of the population of each country of the world belonging to one of the major world religions (Islam, Buddhism, Hinduism, Catholicism and Protestantism). Use the following code that is in the script.

```
religion = read.csv("http://www.correlatesofwar.org/data
-sets/world-religion-data/wrp-national-data-1/at_downloa
d/file",
       header=TRUE, sep=",", strip.white=TRUE,
       stringsAsFactors=FALSE)
```

The first line of code defines a new data frame called religion, and reads in the data from the web server. (The web address changes from time to time, so you might have to explore the website a bit to find the exact address if the one above does not work.)

```
religion = subset(religion,
      religion$year=="2010" )
religion = subset(religion, select =
      c(islmgenpct,budgenpct, hindgenpct,
      chrstcatpct, chrstprotpct, name ) )
```

The second command limits the data frame religion to a subset of the first data frame that just includes observations for the year 2010. The third command further subsets the data frame to just include the percentages of the population from each of the world's major religions.

The following command uses a package that you need to have installed and loaded with the library command (also in the script) called *countrycode*. The associated command will take the name of each country and create a new variable that is the ISO standard three letter abbreviation for the country. The WDI data has these same ISO codes for each country, so the merge will be exact.

```
# convert country code in religion (correlates
# of war - cow) to World Bank code iso3c
# using install.packages("countrycode")
religion$iso3c=countrycode(religion$name,
      "cowc", "iso3c")
# rename to be country
religion <- rename(religion,c('name'='country'))
```

The last line simply renames the variable for country; in the original dataset it is "name" so the code renames the variable to be "country." Makes sense, right?

Merging the two files

Now the merge command is given as follows (there are two options):

```
# If we want to keep only the countries that have #
observations from BOTH data sets:

common = merge(religion, wdim, by="iso3c")

# If we want to keep all the data from both
# data sets:

combine = merge(religion, wdim, by="iso3c",
          all.x = TRUE, all.y = TRUE)
```

Run the merge commands to create new merged data sets, and the descriptive statistics. The data set *common* includes only those countries that are present in BOTH original data sets. The data set *combine* includes all the observations from both original data sets. If a country is in, say, *wdim*, but not in *religion*, it will be included in combined, but all the *religion* variables will take the value NA, since they are not available for that country.

In most applications, you would want all the observations—that is, the combined data set. You can always choose to exclude from analysis observations that are missing some of the variables.

Following the merge are commands to run regressions using the combined data. Try to interpret the output, but bear in mind that we might not want to interpret the coefficients as causal. So when interpreting the regression, it is best to use the language of association or correlation.

Missing values

You have to be careful about missing values. Some R procedures will not run if there are missing values; others will drop observations with missing values, as stargazer did here. Data with missing values for some variables need to be treated with caution! When running a regression, R's default is to run the regression using only those observations for which all variables in the regression are non-missing: the dependent variable and all regressors.

Run the regressions that follow the merge commands, which use the combined data set. Note that N = 180 for the first regression. That is, 180 countries have non-missing values of all the variables in this regression. N = 180 is less than the 217 countries in the combined data. This could be alright if the countries with complete data are representative of all countries. But if the sample of countries with complete data differs systematically from the full sample, there might be some sample selection bias!

If you dropped or added other regressors, this would likely change the sample of countries again. Regressions 2 and 3 have 175 observations. When there are missing values, we must be very careful about comparing alternative specifications, because they may affect the sample.

Extra: web scraping and merging

One of the tasks that R is very good at is scraping website for html coded tables, and loading them into R for use in data analysis. The script *t_merge_extra.R* contains some code that might pique your interest. It is more opaque than the transparent code we have been using so far.

The first line of code in the Data section defines an empty list, sal, that will be filled up as we loop through the scraping of a website table. The website is the Transparent California site that has a section that posts the salaries of every public employee in Santa Clara County.

```
sal <- list()
for (i in 2:50) {
url =
paste0("http://transparentcalifornia.com/salaries/santa-
clara-county/", "?page=", i)
tt <- readHTMLTable(url)
n.rows <- unlist(lapply(tt, function(t) dim(t)[1]))
```

```
sal[[i]] <- ldply(tt, data.frame)
 }
```

The "for (i in 2:50)" part of the next line in the code tells R to loop through the numbers 2 to 50, and do what is after the open brace { until arriving at the close brace } and then loop again with the next number. Inside the braces are the following commands. First, an object url is defined as the address of the website where the table is displayed (different for each page number, indexed by i). Then the command readHTMLTable looks on the Web and reads the html-formatted table on that webpage. The next two lines structure the html table as an R data set.

After the brace, the command

```
sccountysal <- rbind.fill(sal)
```

joins the newly created data sets together into one big dataset. The following line removes the working objects that are no longer needed.

```
# remove some data sets from your workspace
rm(list = c("tt", "sal", "i", "n.rows", "url"))
```

Then there are some commands to clean up the strings that are in each cell and turn them into numbers (the as.numeric command)

```
sccountysal$totalpay = str_replace
(sccountysal$Total.pay…benefits,"[$]", "")
sccountysal$totalpay1 =
       str_replace(sccountysal$totalpay,"[,]","")
sccountysal$totalpay2 =
        as.numeric(sccountysal$totalpay1)
sccountysal$name = as.character(sccountysal$Name)
sccountysal$namelc = tolower(sccountysal$name)

sccountysal$firstname <-
sapply(strsplit(sccountysal$namelc, " "), '[', 1)
sccountysal$lastname <-
sapply(strsplit(sccountysal$namelc, " "), '[', 2)
```

After the cleanup, we are ready for the merge. We first load in a dataset of names and genders that someone has kindly put on the Internet. The list is far from complete, and only includes the several thousand most common male and female names.

```
f = read.table("http://www.cs.cmu.edu/afs/cs/project/ai-
repository/ai/areas/nlp/corpora/names/female.txt",
    sep="\t", col.names=c( "name"), fill=FALSE,
strip.white=TRUE)
f$female=1
m = read.table("http://www.cs.cmu.edu/afs/cs/project/ai-
repository/ai/areas/nlp/corpora/names/male.txt",
                sep="\t", col.names=c( "name"),
fill=FALSE, strip.white=TRUE)
```

```
m$female=0
names <- rbind(f,m)
names$name = as.character(names$name)
names$firstname =tolower(names$name)
```

Some names are both male and female. The data.table package has a command unique that will drop duplicates. Here we do not care whether it drops the male or female, but if this were for research we would care!

```
names <- unique(names, by = "name")
```

We now use a merge command. We use a different one from "merge" here. This is the join command, which comes with the plyr package. There are four options for the join command:

- inner: only rows with matching keys in both x and y
- left: all rows in x, adding matching columns from y
- right: all rows in y, adding matching columns from x
- full: all rows in x with matching columns in y, then the rows of y that don't match x.

```
# merge the datasets using join in library(plyr)
# set type='left' here so that get all of the
# first data frame,
# but only the relevant rows from the names dataset.
sccsal = join(sccountysal, names, by='firstname',
type='left', match='all')

# Create a factor var for female, for box plot
sccsal$femalefac=as.factor(sccsal$female)

table(sccsal$femalefac, useNA="ifany")
```

We can see that only 2/3 of the observations match to a gender. There remain about 1/3 of the salary database names that do not match. We need a better names database, right?

We can however proceed and draw a boxplot of the distribution of salaries by gender.

```
# Boxplot - change scale of y-axis and drop
# outliers
p0 = ggplot(aes(y = totalpay2, x = femalefac), data =
sccsal) + geom_boxplot(outlier.size=NA)+
  labs(title="Total pay according to gender",
x="Female?", y="Total pay, outliers excluded")
# compute lower and upper whiskers
ylim1 = boxplot.stats(sccsal$totalpay2)$stats[c(1, 5)]
# scale y limits based on ylim1
p1 = p0 + coord_cartesian(ylim = ylim1*1.05)
p1
```

The box plots created are only for the top 2500 salary earners in the country. You can redo the analysis by looping from 2 to 500, thus obtaining 25,000 records. But this is very time-consuming (might take 15-30 minutes depending on your connection speed). Alternatively, the Transparent California website lets you download the entire file as a csv file, which you can then import. Just for fun, we can also run a regression. What does it tell us?

Reshaping

We often want to reshape a data set from wide to long or long to wide. Suppose, for example, that we have a data frame *scores* of five test scores for 20 individuals. The data is wide format, where each row is for one student. We have six columns. An id variable, and then five columns for the test scores, labelled ts1, ts2, ts3... ts5. How can we turn this into a long data set? We use the packages tidyr and dplyr (they have to be installed if you have not already installed).

```
ts1 <- runif(20)+2
ts2 <- runif(20)+3
ts3 <- runif(20)+4
ts4 <- runif(20)+6
ts5 <- runif(20)+7
id <- seq(1, 20, by=1)
scores <- data.frame(ts1,ts2,ts3,ts4,ts5,id)
install.packages("tidyr")
library(tidyr)
library(dplyr)
scores_l <- scores %>% gather(test, score,
           ts1:ts5)
```

The syntax makes use of the "pipe" operator (%>%) from dplyr. This operator enables us to write out commands more like sentences, rather than nesting every command inside ever more elaborate parentheses. Search Google for margrittr to learn more (there's an inside joke there, *ceci n'est pas une pipe*, LOL). The data can now be displayed using ggplot, first a scatter plot and then a box plot. But before doing that we create a variable indicating the number of the test.

```
scores_l$testnum=substr(scores_l$test,3,3)
ggplot(data=scores_l, aes(x = testnum,
       y = score, color=id)) +
       geom_point(size = 1.5,alpha = 0.75)
ggplot(data=scores_l, aes(x = testnum, y = score))+
geom_boxplot()
```

Summary

Key R commands (functions):

- merge: merge two data sets by matching on one or more common variables
- unique: from the data.table package, will drop duplicates
- join: from plyr package, brings more precision and customizability to merges
- readHTMLTable: can read simple html tables on the Internet into R format
- str_replace: replace a set of characters (a string) in a variable
- runif: generate simulated random variable by drawing from the uniform distribution
- data.frame: turn a set of vectors into an R dataframe
- gather: used with “piping” to reshape a dataset

Key points/ concepts:

- Merging data is very useful, but must be treated with caution.
- Missing values can create challenges in statistical analysis, because of sample selection bias and difficulties of comparability across specifications.
- Web-scraping and reshaping are common activities of data analysts.

8. Nonlinear regression

In this tutorial, you will learn how to:

- Run nonlinear specifications of regressions, including polynomials and natural logs
- Run regressions with interaction terms
- Replicate Table 8.1 of Stock and Watson (p. 284)
- Examine what happens with perfect multicollinearity

Where to find what you need:

- Downloads: R tutorial scripts and data files `http://rpubs.com/wsundstrom/home`

Introduction

Regression analysis can be used to capture nonlinear and other complex relationships between variables, such as when we examine the effects of binary variables and interactions, which permit effects of some variables to depend on other variables.

Start RStudio and open the script *t_nonlinear.R* in your script editor. Scroll through the script and note that the first part of this case uses the CA school district data set, while the second part uses CPS earnings data. Edit the setwd(…) command and then run the script from the top through the CA school data section.

Run nonlinear regressions using CA school data

In the analysis section for the CA school data we examine the relationship between the district average income (avginc) and the average test score (testscr).

First run the scatter plot: It suggests that the relationship is not a straight line. It is easy then to run quadratic or higher polynomials of this

relationship by adding the square, cube, etc. of the income variable to our regression. One way to do this would be to create new variables for avginc squared, etc. For example, we could include the following line:

```
caschool$avginc2 = caschool$avginc^2
```

Instead of creating new variables, we will use a nifty feature of R, which is that we can generate new variables within the regression command itself, "on the fly" so to speak. For example, examine the command for reg2:

```
reg2 = lm(testscr ~ avginc + I(avginc^2),
      data=caschool)
```

The term I(avginc^2) represents a new variable equal to avginc^2, which is avginc to the 2 power (squared). R will make a new variable by performing whatever calculation is inside the I(…). Note that the rules for math expressions in R are basically the same as Excel.

Now run the regressions *reg1 – reg3* and the *stargazer* command that follows them. The option digits=4 is included in the stargazer command. You may sometimes need more digits if the coefficients are very small, or else they round to zero. Please note that just because the coefficient is a small number this does not mean the size of the effect is small, because that also depends on the scale of the X variable. *Which specification do you prefer, and why?*

Now let us try some log specifications: Run the regressions *reg4 – reg6* and the stargazer command that follows them. Note that log(X) = the natural logarithm of X.

Bear in mind that the dependent variable (Y) is not the same for all three regressions. For *reg4* (lin-log), it is the average income in dollars. For *reg5* and *reg6*, it is the natural log of average income. Therefore, the coefficients are not in the same units for all the regressions, and it is not straightforward to compare them. In particular, the R-squared can only be compared for two regressions with the same dependent variable.

Stargazer tells us what the dependent (Y) variable is for each column. Also, in the stargazer command, the option column.labels= has been added to allow us to add more informative labels to the column heads.

Make sure you can interpret what each slope coefficient is telling you in words (remember that changes in logs are proportional changes).

Regressions with dummies and interactions

Now let us see how to run regressions with dummy variables and interaction terms by replicating Table 8.1 from Stock and Watson (3rd ed., p. 284). This table examines gender differences in earnings and the return to education

based on individual-level data from the March 2009 Current Population Survey (a brief description is in the Appendix).

In the analysis, the dependent variable will be the natural log of average hourly earnings. The log-linear specification is often used in studies of earnings.

The key X variables are years of education, a binary (dummy) variable for female, and years of work experience, which is estimated as the number of years since the individual completed their schooling.

We will also include a regressor that is the interaction of education and female. An interaction variable is an additional regressor that is the product of two other regressors. The coefficient on this variable estimates the difference between the slopes on education for women vs. men.

Run the data section, where we create a couple of new variables. Although we could use I(log(ahe)) as the dependent variable in the regressions, we have created a separate variable *logahe = log(ahe)* for convenience here.

Now run the regressions for *col1 – col3m* and create the first stargazer table showing the alternative approaches to estimating regression (3).

When including an interaction term, you should always include the individual variables as regressors too: yrseduc, female, and yrseduc x female.

Note that *col3alt* and *col3* are two different ways to run the same regression. In *col3alt* we add the interaction variable by adding I(yrseduc*female), whereas in col3 we use another shortcut built into R, where the interaction term is yrseduc:female. The colon (:) does the trick. This way of doing it is more flexible and we will generally use it for interactions.

The use of the female dummy variable and the interaction allows the intercept and the slope to be different for females and males. Yet another way to accomplish the same thing here would be to run separate regressions for the female and male samples, which we do in *col3f* and *col3m*. Note how we create the female subset of the data.

Look at the coefficients in the table, and make sure you understand why together they tell you the same thing as the coefficients for *col3alt* and *col3*.

Now let us complete Table 8.1 by running the specification in column (4), which adds experience, experience squared, and dummies for region. Make sure you understand why we did NOT include a fourth dummy variable for the Northeast region.

Run the regression and generate the table. Compare the results with the textbook's Table 8.1. Aside from rounding, the results should be identical.

Perfect multicollinearity

Recall that perfect multicollinearity occurs when one of the regressors is a linear function of one or more of the other regressors. A common mistake that can cause perfect multicollinearity is the so-called dummy variable trap. Different statistical packages and procedures deal with perfect multicollinearity in different ways. In the case of R, the lm(…) command will run, but will typically drop one or more of the offending regressors. Let us take a look.

Find the section of script with the heading "Perfect multicollinearity: diagnosis," and run the commands.

Why is there perfect multicollinearity in this regression? How did R deal with the problem? For some statistical procedures, perfect multicollinearity may cause R to fail and create an error message. Treatment of the problem in this case is straightforward: drop one of the region dummies. In other cases, the problem may not be so obvious.

A quick point on dummy variables

Recall that when you have m categories, and you need m-1 binary variables to avoid the dummy variable trap. R will create the correct number of dummy variables from a factor variable, but you may want to control which category is the reference (zero) category. Here is how. In this case, we use the race factor variable, and set the reference category to be "White":

```
# determine the names of the categories
levels(acs$race)
# create new variable that sets the reference level as
desired
acs$racew = relevel(acs$race, ref="White")
```

Then use the new variable in the regression instead of the old one.

Summary

Key R commands (functions):

- lm: the linear model command can, in some cases, be used to estimate non-linear relationships, by including polynomial and log terms as regressors
- levels: will return the levels or "labels" of a factor variable
- relevel: can be used to change labels of a factor variable

Key points/ concepts:

- I(...)can be used within a regression to create new variables "on the fly."
- A colon (such as X1:X2) can be used within a regression to create an interaction term between X1 and X2 "on the fly."
- A regression suffering from perfect multicollinearity will result in R dropping one or more perfectly collinear regressors.

9. Binary dependent variables

In this tutorial, you will learn how to:

- Run regression models with a binary (dummy) dependent variable, including the linear probability model, logit, and probit
- Replicate Table 11.2 of Stock and Watson (p. 401)
- Plot predicted probabilities implied by the probit estimates

Where to find what you need:

- Downloads: R tutorial scripts and data files `http://rpubs.com/wsundstrom/home`

Introduction

In many real-world applications, we are interested in explaining or predicting dependent variables that are binary (0-1). Examples could include such decisions as whether or not to get married, to attend college, or to buy a second-hand ambulance; Stock and Watson focus on a lender's decision whether or not to approve a mortgage loan.

Models with a binary dependent variable can be estimated using ordinary least squares regression, treating the dependent (0-1) variable like any other: this is called the linear probability model (LPM) and it is used widely in the social sciences. For reasons discussed, the LPM is not always the best way to estimate a model with a binary outcome. Instead, economists often use a nonlinear model such as the logit or probit. These are easy to estimate in R, although the interpretation of the coefficients can be tricky.

Start RStudio and open the script *t_binary.R* in your script editor. Edit the setwd(…) command as usual.

Run the LPM, logit, and probit regressions

Examine the data section of the script. Following the presentation in Stock and Watson, we are using data on mortgage applications from the Home Mortgage Disclosure Act (HMDA), which is available in the AER package in R. It is already a data frame in R, so we read it directly into R using the data command. Run the entire data section, and note the new variables created.

Let us start by examining the relationship between an applicant's payment-income (P/I) ratio and a binary variable equal to 1 if their mortgage application was denied, 0 if it was accepted. This should be a positive relationship, because someone with a higher P/I will have a harder time affording their mortgage payments, leading the bank to be more reluctant to make the loan. The boxplot shows us that indeed the median P/I ratio is higher for those denied (a "yes" means they were denied), and the distribution is shifted higher compared with those who were not denied, that is, who were accepted for loans.

The scatter plots also examine this relationship. If you run the first ggplot command, you will see that it is hard to tell what the relationship is, because of the binary dependent variable. The second ggplot puts a linear regression line through the scatter, and here the positive relationship does show up quite clearly. This line represents the linear probability model's prediction of the probability of denial. What are some obvious problems with this probability prediction?

We now turn to the regression models in Stock and Watson Table 11.2. The first is the linear probability model, which is just a standard OLS regression. The dependent variable in the data frame is defined as a factor variable, and to get the regression to work we need to convert it to a 0-1 number using I(as.numeric(deny) - 1).

Next we run the equivalent logit and probit models using the command glm(…).We write the regression formula in exactly the same way we do for the lm(…) regression, with the dependent variable followed by ~. To estimate the logit (fm2), we add the family = binomial option, and for the probit we add the family=binomial(link="probit") option. The x=TRUE option is added to allow us to predict the estimated probabilities (see below):

```
# Run logit regression
fm2 = glm(deny ~ afam + pirat + hirat + lvratcat
       + chist + mhist +
       phist + insurance + selfemp,
       family = binomial, x=TRUE, data = HMDA)

# Run probit regressions
fm3 = glm(deny ~ afam + pirat + hirat + lvratcat
       + chist + mhist +
        phist + insurance + selfemp,
        family = binomial(link = "probit"),
       x=TRUE, data = HMDA)
```

Once we have run all the regressions, we can put them into a stargazer table, as usual. In the stargazer command, notice that we use the standard error correction for the LPM to correct for heteroscedasticity, but leave the logit and probit standard errors at their default (NULL) values:

```
se=list(cse(fm1),NULL,NULL,NULL,NULL,NULL).
```

Run the stargazer table and compare the results with Stock and Watson Table 11.2. In the textbook's Table 11.2, the bottom panel runs some joint hypothesis tests. We can do that as well, using the lht command. For example, suppose we want to test the joint null that the coefficients on both *hirat* and *pirat* are zero. The script shows you how. Run the following line. Can you reject this hypothesis?

```
# F tests of joint hypotheses can be conducted
# using lht. For example:
lht(fm4, c("pirat = 0", "hirat = 0"))
```

Predicted probabilities and marginal effects

One problem with the probit and logit models is that the coefficients are not easily interpreted in any "natural" terms. For the LP model of column (1), you can read the coefficient of 0.0837 on *afamyes* (binary variable for black) pretty directly as meaning that the probability of being denied a mortgage is 8.37 percentage points greater for a black person, other variables held constant. But what does the coefficient of 0.389 mean in the probit column (3)?

To aid in interpreting the results, let us look at a couple of ways of "translating" the probit results from column (3).

Marginal effects of X's on predicted probability

First, let us rescale the probit coefficients so that they represent the marginal effect of a change in the regressor on the *predicted probability* of mortgage denial. If you understand the probit model, this will immediately raise a flag. The fact is, the relationship between any X and the probability is nonlinear, so there is no single number representing the marginal effect.

To deal with this we typically calculate some kind of *average* marginal effect. There are two ways to do this. First, calculate the marginal effect at the means of all the X variables. In effect, we create a fictitious person who has the sample mean characteristics, and ask how their predicted probability of denial would change with a one-unit change in one of the variables, holding all the rest constant. Second, calculate the marginal effect of a variable separately for each individual in the sample, and then average it over all the individuals.

The command maBina from the package erer allows us to do it either way. In the following commands, the first (fm3a) uses the first method, evaluating the marginal effect at the mean X values (hence x.mean = TRUE). The second (fm3b) uses the second method, calculating the marginal effect for each observation and averaging. The option rev.dum = TRUE does the calculations treating dummy variables as 0-1, rather than pretending that they are continuous. Note that using maBina requires that we specified the option x=TRUE in the glm estimation of the probit.

```
fm3a = maBina(fm3, x.mean = TRUE,rev.dum = TRUE, digits
= 3)
fm3b = maBina(fm3, x.mean = FALSE,rev.dum = TRUE, digits
= 3)
```

The results can be displayed using the stargazer command that follows in the script. Note that the results are almost identical in this case.

Plot predicted probability of denial by P/I ratio by race

The last little bit of code shows how to plot the predicted probability as a function of one of the X variables, once you have run the maBina command. It uses the maTrend command:

```
preplot = maTrend(fm3b, n=300, nam.c="pirat", nam.d="afa
myes", simu.c = TRUE)
plot(preplot)
print(preplot)
```

Note the following:

- The first argument fm3b is the name of the stored results from maBina.
- n = 300 gives the number of points to plot (more means smoother curve).
- nam.c = "pirat" is the continuous X variable over which to plot the predictions (P/I ratio in this case).
- nam.d = "afamyes" is the binary variable over which to plot separate curves (race in this case).
- simu.c = TRUE means we use simulated values of the X variable, which makes for a smoother curve.

Run the command and see what happens. The resulting graph will look better if you make sure your plot window (in lower right corner) in large rather than small. So resize that window and run again.

The script also includes code that generates the same plot but where you calculate the probabilities “manually” using the coefficients from the regressions. You can also use the pnorm() function to calculate the

probability that someone will be denied, under different assumptions about their characteristics. Thus, two individuals, one white and the other black, but sharing the same average characteristics, will have different probabilities of denial. This is a good example of how the regression output, which is stored in R, can be retrieved and used in displays or to carry out meaningful calculations.

Summary

Key R commands (functions):

- glm: runs generalized linear models, which include logit and probit
- maBina: predicts marginal effects of X on predicted probability in a probit or logit model
- maTrend: predicts probability as a function of X
- plot and curve: make graphs of variables
- pnorm: calculate the probability of observing a value assuming it is a random variable that follows the normal distribution

Key points/ concepts:

- A regression with a binary dependent variable can be run as an OLS regression, but make sure the Y variable is numeric (0-1).
- A better way to estimate such a model is the logit or probit, using R's glm command.
- The predicted probabilities can be obtained and plotted to see the impact of a regressor at different values.

10. Regressions with panel data

In this tutorial, you will learn how to:

- Run regression models for panel data, including:
  - Pooled model
  - Fixed effects (FE) models
  - Models in first differences (FD)
- Correct standard errors using clustering

Where to find what you need:

- Downloads: R tutorial scripts and data files `http://rpubs.com/wsundstrom/home`

Introduction

Panel data sets combine cross-section and time-series elements. Units of observation, such as individuals or states, are observed repeatedly over time. Special regression techniques for analyzing panel data allow us to control for characteristics of the cross-section units that do not vary over time and are not observable as variables in the data. A common way to do this is to estimate a model with *fixed effects.* As we shall see, fixed effects are essentially equivalent to including a separate dummy variable for each cross-section entity (but one).

Our example here looks at the effect of alcohol taxes on traffic fatalities, using state-by-time data. Does raising the tax on beer reduce drunk driving deaths? The tutorial follows closely the analysis presented in Chapter 10 of Stock and Watson, *Introduction to Econometrics.* A convenient package, called *plm*, makes it easy to run fixed-effects (FE) models in R, as well as make some corrections to get correct standard errors on the coefficients.

To get started, open the script *t_panel.R* in your script editor in RStudio. Edit the setwd(…) command for your working directory. Check the

list of packages in the library commands and make sure you have installed all of them. Then run the script from the top through the data section.

In section of the script, note that there are three functions to calculate correct standard errors for different regression procedures. We will be discussing and applying these functions when we create tables of regression results for panel models and instrumental variables (next chapter).

The data set is imported from a package called AER, and is the same data used by S&W throughout chapter 10. Take a look quick look at the data by clicking on Fatalities under Data in the Workspace (upper right panel). Note the structure of the data. The first seven rows are observations for the state of Alabama (state="al"), one observation for each year 1982-1988. Definitions of the variables can be obtained by searching on Fatalities under the Help tab.

To replicate the regression results in S&W, we create a new variable for traffic fatalities per 10,000 persons, which is called *fatality_rate* here.

Now look at the results of the stargazer command (descriptive statistics). Why is N = 336? Why is per capital income so low (average < $14,000)?

Cross-section regressions

Our question is whether fatality rates were affected by the beer tax in a state. Before proceeding, make sure you can offer a simple argument for why an increase in the beer tax might be expected to reduce traffic fatalities. Your explanation should include a supply-and-demand diagram.

Is it true? Let us start, as S&W do, by plotting the relationship between the beer tax (X) and the fatality rate (Y) in levels of the variables, for a cross-section of states at the beginning and ending years of the data set, 1982 and 1988. The ggplot commands do this and add a regression line. Run each plot and look at the relationship in each case. Are you surprised?

Regressions for 1982 and 1988 cross-sections

Let us run the regressions for 1982 and 1988 and put them in a table. Make sure you understand the details, including the data subset command, and the se=list(...) correction in stargazer. Then interpret the results. For example, is the slope significantly different from zero?

```
# Simple regressions for 1982 and 1988 cross
# sections, as in SW
reg1982 = lm(fatality_rate ~ beertax,
             data = subset(Fatalities, year==1982))
reg1988 = lm(fatality_rate ~ beertax,
             data = subset(Fatalities, year==1988))
stargazer(reg1982, reg1988,
  se=list(cse(reg1982),cse(reg1988)),
  title="Simple cross-section regressions",
```

```
type="text", column.labels=c("1982", "1988"),
df=FALSE, digits=4)
```

Pooled and fixed-effect (FE) regressions

We have a puzzling finding here, namely that states with higher beer taxes tend to actually have higher traffic fatality rates. Of course, we have not controlled for anything, so maybe there is omitted variable bias (OVB). Namely, there is something else about the states with high beer taxes that is correlated with traffic deaths. Can you think of what such an OV might be?

We can take advantage of the panel structure of our data to control for any time-invariant characteristics of the states. As Stock and Watson show, if there are characteristics of a state that do not change over time, when we examine the change in our variable(s) of interest-- say, between 1982 and 1988-- those non-changing effects will cancel out.

We can take advantage of the entire data set to control for time-invariant state effects in a couple of different ways. To accomplish this in R, we make use of the package plm, which includes all the tools we need for dealing with panel data.

The panel structure of our data means that an observation is defined by the value of two variables: the state and the year. In general we will refer to the cross-section unit as the "entity" (in our case, the state), and the time variable as the "time" or "period" variable (in our case, the year).

In each plm command, `index=c("state","year")` defines the first variable (state) as the entity and second (year) as the time variable.

Pooled regression

We can use plm to run a straightforward OLS regression on the entire panel, which is usually called a "pooled" model (all the years are pooled into a big data set and treated as separate observations). We could run this using our old friend lm as follows:

```
lm(fatality_rate ~ beertax, ...),
```

but instead we will use the plm command with the option model="pooling" to obtain the pooled estimates:

```
reg1 = plm(fatality_rate ~ beertax,
      data = Fatalities, index =
      c("state","year"), model="pooling")
```

Fixed-effects (FE) regression

Before we look at the results in a stargazer table, let us run the fixed-effects (FE) model. In effect the FE model runs the pooled regression with a complete set of state dummy variables (all but one of the states, of course).

In this case, each state is allowed its own intercept, so that the slope on the beer tax variable can be interpreted as the effect of a change in the tax *within-state*. The model implicitly assumes that within each state, the beer tax has the same *marginal* impact on fatalities.

Here is the code using plm. Note that in this case, model="within" means fixed effects for the entity variable, state. The option effect="individual" is not strictly necessary here, because it is the default option, but we will see what it is about in a second.

```
reg2 = plm(fatality_rate ~ beertax, data =
      Fatalities, index = c("state","year"),
      model="within", effect="individual")
```

To see what this FE regression is doing, let us go ahead and run another regression that explicitly includes a full set of state dummy variables. We can do this by including *state* (a factor variable) as a regressor in OLS, as in reg2a.

Go ahead and highlight and run the three regression commands-- reg1, reg2, and reg2a-- and then the stargazer table to compare reg2 and reg2a. In the table, note that the plm FE results (reg2) include only the slope estimate-- the intercept and state dummy coefficients are not explicitly calculated.

Now compare the two estimates of the slope on the beertax variable. What do you see? Note also the R^2 for each regression. They are quite different. For now, note that they are not measuring the same thing. Examine the dummy variable estimates for reg2a. One state is missing. Which one? Why?

Finally, note that we can also include period (year) FEs as well as state FEs. This would be like including a dummy variable for each year (but one). This is implemented in plm with the effect="twoways" option, includes in reg3. Now you can see what the effect= option is doing. Go ahead and run reg3. We will look at the results below.

Clustered standard errors

An important feature of panel data is that there is often correlation between the errors within entities. This could be due to serial correlation-- essentially, persistent effects through time. In our example, suppose that for some reason, traffic fatalities in Arizona were unusually high in 1983. Serial correlation would mean that fatalities would likely continue to be somewhat higher than expected in 1984, and so on.

When errors are correlated, the conventional estimates of the standard errors are incorrect. In many cases the estimated SEs are too small-- sometimes dramatically so-- which leads to excessive confidence in the precision of the results. Fortunately, we can correct the SEs for the existence

```
Panel regressions, clustered SEs
================================================
                      Dependent variable:
              ---------------------------------
                           fatality_rate
              Pooled OLS   State FE    St-Yr FE
                 (1)         (2)         (3)
------------------------------------------------
beertax       0.3646***   -0.6559**    -0.6400*
               (0.1186)    (0.2888)    (0.3501)

Constant      1.8533***
               (0.1175)

------------------------------------------------
Observations     336         336         336
R2              0.0934      0.0407      0.0361
Adjusted R2     0.0928      0.0348      0.0302
F Statistic  34.3943*** 12.1904*** 10.5133***
================================================
Note:              *p<0.1; **p<0.05; ***p<0.01
```

of correlation across time periods within the entities using "clustered" standard errors.

Implementing clustered SEs with stargazer requires a different correction function from the old heteroscedasticity correction; the clustering function is defined earlier in the script as clse.

Comparing the results

Let us go ahead and run the following stargazer command comparing the pooled and FE models, all with clustered SEs-- here's the code and the results:

```
# put in a table with SEs clustered by state
stargazer(reg1, reg2, reg3,
 se=list(clse(reg1),clse(reg2),clse(reg3)),
 title="Panel regressions, clustered SEs", type="text",
 column.labels=c("Pooled OLS", "State FE", "St-Yr FE"),
 df=FALSE, digits=4)
```

Note that the regression in column (2) replicates equation (10.15) in the S&W textbook, while column (3) replicates (10.21). You can see that including fixed effects reverses the sign of the beer-tax coefficient, suggesting that in fact within states an increase in the tax is associated with reduced traffic fatalities.

Panel regression in first differences

An alternative approach to taking advantage of the full panel is to run the regression in *first differences* (FD). That is, we create new variables that

represent the year-to-year change in each variable within a state: e.g., the change in the fatality rate in Alabama between 1982 and 1983, between 1983 and 1984, and so on. We then run the regression in these differences.

FD is implemented in reg4 using the model="fd" option in plm. The differenced variables are created automatically within the procedure, so we do not have to.

In the absence of bias due to endogeneity, the point estimates of the slopes from the FE and FD regressions should be quite similar, because in each case the regression is "sweeping out" any confounding effects that are constant within-state over time. If you run reg4 and the stargazer command that follows it, you will see that this is not exactly the case here. This is difficult to interpret and may indicate that neither the FE nor FD model is doing a good job controlling for confounding effects.

Adding covariates (Xs)

As with any regression, we can add more variables to a fixed-effects regression. The reason to do so would be similar to why we add regressors in any regression: they may control for omitted effects and therefore mitigate OVB. Note: In FE or FD panel regressions, any additional covariates must be time-varying within the units-- states, in our example. Any variables that are constant over time within each state (such as the total land area of the state) would be perfectly collinear with the fixed effects. The regression reg3a adds a variable for spirits consumption to our FE regression. The regression 4a adds dummy variables for each year, in the first-difference specification. To create the dummy variable we have used the dplyr package function mutate() which is quite versatile and used a lot for more complex data manipulation.

Extracting the fixed effects

Sometimes we would like to retrieve the fixed effects coefficients, as if we had run the regression with dummy variables for state and time. This can be done after running plm, using the function fixef. The end of the tutorial script shows you how.

Summary

Key R commands (functions):

- plm: runs various panel regressions (requires package plm)
- mutate: part of dplyr package. Enables one to easily make changes to variables or add new variables

Key points/ concepts:

- plm works a lot like lm.
- index = c("state","year") is used to specify the entity and time variables.
- plm options can be used to specify fixed effects for entities and/or time periods.
- Clustered standard errors can (and should) be incorporated into stargazer tables.

11. Instrumental variables

In this tutorial, you will learn how to:

- Run regression models with instrumental variables (two-stage least squares)
- Obtain correct standard errors
- Run various diagnostic tests

Where to find what you need:

- Downloads: R tutorial scripts and data files `http://rpubs.com/wsundstrom/home`

Introduction

Instrumental-variables (IV) regression is a widely used technique for dealing with endogeneity problems-- namely, when one or more of the regressors (X) is correlated with the error term (u). When this is true, the estimated OLS coefficient on the endogenous X is biased and inconsistent. IV regression rests on finding a variable (or variables) that is correlated with X but uncorrelated with u. Such a variable is called an instrumental variable (IV), or just an *instrument*. Your textbook discusses in greater detail the requirements for a valid instrument. In this tutorial, we focus on how to run IV regressions in R, using the packages *AER* and *ivpack*. As usual, we will work through some examples from Stock and Watson, *Introduction to Econometrics*-- in this case, from Chapter 12.

The example we will look at is a classic application of IV regression: estimating the price elasticity of demand for a product. In this case the product in question is cigarettes. Recall that the elasticity of demand is the percentage change in quantity demanded for a percentage change in price, moving along the demand curve. This can also be expressed as the slope of the relationship between the natural logs of Q and P. That is,

$$\text{elasticity} = \epsilon = \frac{dQ_d/Q_d}{dP/P} = \frac{d\ln(Q_d)}{d\ln(P)}.$$

It seems that we could simply use data on quantity and price to run a regression between the log-transformed values and obtain an estimate of the elasticity, as follows:

$$\ln Q = \alpha + \beta \ln P + u, \text{ where } \hat{\beta} \text{ is the estimate of the elasticity.}$$

The problem is, *combinations of quantity and price that we observe in the data reflect the forces of both demand and supply.* Therefore, in general the relationship we estimate is neither a demand curve nor a supply curve, but usually a complex mix of shifting demand and supply curves.

In the above regression model, *P* is endogenous: it is correlated with the error term in the demand curve. Unobserved factors that increase demand will tend to increase the price as the equilibrium moves up the supply curve. Thus $\hat{\beta}$ is a biased estimate of the demand elasticity.

To deal with this we will estimate the demand equation using IV. IV regressions are usually estimated using a procedure called *two-stage least squares* (2SLS). The command ivreg estimates 2SLS using a slight modification of our familiar regression syntax.

To get started, open the script *t_ivreg.R* in your script editor. Edit the setwd(…) command for your working directory. Check the list of packages in the library commands and make sure you have installed all of them. Then run the script from the top through the data section.

The data set *CigarettesSW* is imported from the AER package, and is the same data used by S&W throughout chapter 12. We will be working with the 1995 cross-section of states, which is called *cigs* in the tutorial. Note that we divide the prices by the CPI variable to obtain the "real" prices, though this is not strictly necessary when dealing with a single year of data.

The variable we will use as an instrument is *tdiff*, which is the real tax on cigarettes arising from the state's general sales tax. The presumption is that in states with a larger general sales tax, consumers will tend to pay a higher price for cigarettes, but that the general tax is not determined by the cigarette market, so otherwise uncorrelated with cigarette demand.

Running IV regressions in R

As the name suggests and the Stock and Watson textbook explains, 2SLS involves estimating two regressions: In the first stage, the endogenous variable (log price in our example) is regressed on the instrument or instruments (*tdiff*), along with any other exogenous variables (controls). Then the estimated coefficients from the first-stage regression are used to

predict the endogenous variable (log price). If you have a valid instrument, the predicted value of the endogenous variable has essentially been "cleaned" of its correlation with the error term in the outcome equation.

In the second stage, the dependent variable (log quantity) is regressed on the predicted values of the endogenous variable (predicted log price), along with the exogenous controls. If the instrument is valid, the estimated coefficient on the predicted endogenous variable in the second stage is an unbiased estimate of the desired parameter (in our case, the demand elasticity).

In practice, statistical software often just reports the second-stage regression, which is what one is usually interested in. That is the case with the command we use, ivreg.

OLS and first-stage regressions

Before we turn to the IV estimates, let us run the OLS regression (around line 109) and the stargazer table following it. What is the estimated price elasticity? *Remember, this estimate is probably biased because of the endogeneity of the price.*

Now let us take a quick look at what would be the first-stage regression, which is called first1 in the script. Make sure you understand why this is the first-stage regression. Run it and note that the coefficient on the instrument, *tdiff*, is statistically significant (at what level?). This is important, as we shall discuss in class.

Finally, let us run one more regression, called the *reduced form*. The reduced-form regression is a regression of the outcome variable (log quantity) on the instrument and any other exogenous variables, but not the endogenous (price) variable. The reduced-form shows the indirect effect of the instrument on the outcome variable, by way of the endogenous variable.

IV (2SLS) regressions

Now let us turn to the actual IV regression. Note the syntax in the following ivreg command:

```
iv1 = ivreg(lquant ~ lprice | tdiff ,
       data = cigs)
```

The part before the vertical line, lquant ~ lprice, is the (second-stage) regression we want to estimate: the demand curve. You enter it exactly as you would a regression in lm. The part after the vertical line, *tdiff*, is the instrument. If we had more than one instrument, we would enter them using a "+" sign (see below). Instruments should only appear after the vertical line.

Note that in the stargazer command we use the *ivse* function in se=list(ivse(iv1)) to obtain corrected standard errors.

Run the first ivreg and the stargazer and examine the results. You might compare them with the OLS we ran before. What is the estimate of the demand elasticity here? Also, compare with the estimate presented in Stock and Watson (equation 12.11). You'll see a slight difference in the estimates SEs, due to a small difference in the formulas used.

Note that we can have more than one instrument, and we can also include exogenous control variables. Let Y be the outcome (dependent) variable of interest (*lquant* in our example), X be the endogenous variable (*lprice*), W be any exogenous regressors, not including instruments (see below), and Z be the instruments (*tdiff* in our case). Then the syntax for ivreg is:

```
ivreg(Y ~ X + W | W + Z, ... )
```

Important note: Endogenous variables (X) can only appear *before* the vertical line; instruments (Z) can only appear *after* the vertical line; exogenous regressors that are not instruments (W) must appear *both before and after* the vertical line.

Let us add an exogenous regressor: log per capita income. Why does this belong in a demand equation? Let us also add another instrument suggested in the book: the real tax on cigarettes tax/cpi, which we'll create in the regression. This gives us the specification in the textbook (12.16).

Here is the code:

```
iv2 = ivreg(lquant ~ lprice + lincome | lincome
      + tdiff + I(tax/cpi), data = cigs)
```

Run the above and the stargazer that follows it.

Tests and diagnostics

The instrumental variables method must be used with considerable caution. In general, it is crucial to have a compelling case for the validity of your instrument(s). Although there are statistical tests that can help us judge, for the most part IV requires knowledge and judgment about the specific application. Section 12.4 in Stock and Watson offers a good discussion for our example.

You can run some useful diagnostics on your IV regression using the summary command, as shown at the end of the script. Here's the code and the results:

```
summary(iv2, vcov = sandwich, diagnostics = TRUE)

Call:
```

```
ivreg(formula = lquant ~ lprice + lincome | lincome + tdiff +
    I(tax/cpi), data = cigs)

Residuals:
      Min        1Q    Median        3Q       Max
-0.6006931 -0.0862222 -0.0009999  0.1164699  0.3734227

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   9.8950     0.9288  10.654 6.89e-14 ***
lprice       -1.2774     0.2417  -5.286 3.54e-06 ***
lincome       0.2804     0.2458   1.141     0.26

Diagnostic tests:
                 df1 df2 statistic p-value
Weak instruments   2  44   228.738  <2e-16 ***
Wu-Hausman         1  44     3.823  0.0569 .
Sargan             1  NA     0.333  0.5641
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.188 on 45 degrees of freedom
Multiple R-Squared: 0.4294, Adjusted R-squared: 0.4041
Wald test: 17.25 on 2 and 45 DF,  p-value: 0.000002743
```

The top part of the table just presents the coefficients, etc. But the bottom portion, *Diagnostic tests*, contains some new information. Note the following:

- Weak instruments: This is an F-test on the instruments in the first stage. The null hypothesis is essentially that we have weak instruments, so a rejection means our instruments are not weak, which is good.
- Wu-Hausman: This tests the consistency of the OLS estimates under the assumption that the IV is consistent. When we reject, it means OLS is not consistent, suggesting endogeneity is present. If we accept the null, it essentially means that the OLS and IV estimates are similar, and endogeneity may not have been a big problem.
- Sargan: This is a test of instrument exogeneity using overidentifying restrictions, called the J-statistic in Stock and Watson. It can only be used if you have more instruments than endogenous regressors, as we do in iv2. If the null is rejected, it means that at least one of our instruments is invalid, and possibly all of them.

Instrumental variables in panel regression

There is no problem using instrumental variables in panel estimation, with the plm() command. As seen in Chapter 10, the | symbol is used to define the IV variables. It is common for an estimation to include all the explanatory variables as instruments, by using a dot (.) and then use the minus sign (-) to exclude the troublesome endogenous variable (or variables) from the list of instruments, and the plus sign (+) to include the IV variable(s).

The plm package includes a dataset called *Crime* that consists of the crime rates per person for 90 counties in north Carolina observed over the period 1981-87, so 7 years (so there are 630 observations. Explaining crime rates by the number of police per capita seems like an interesting exercise. Our intuition is that more police should reduce crime, but of course places with high crime also have more police officers. The number of police officers is endogenous. If we had a good instrumental variable, we could estimate the magnitude of how much adding additional police officers to a county force reduces the crime rate. There is a growing literature trying to estimate that relationship. We do not pretend that the *Crime* datset has such an IV, but for illustrative purposes, here is the syntax for the estimation, where the general level of taxes per capita is used as an IV:

```
data("Crime", package = "plm")

reg1 =  lm(log(crmrte)~log(polpc)+log(density)
      +factor(year),data = Crime)
reg2 =  plm(formula=log(crmrte)~log(polpc)
      +log(density)+factor(year) | . -log(polpc)
      +log(taxpc),data = Crime,model="within")
reg3 =  plm(formula=log(crmrte)~log(polpc)
      +log(density)+factor(year) | . -log(polpc)
      +log(taxpc),data = Crime,model="random")

stargazer(reg1, reg2, reg2, reg3,
           title="IV Regression", type="text",
           df=FALSE, digits=5)
```

Estimating the standard errors when doing panel estimation with instrumental variables is more complex than in other cases. The plm package was updated in November 2016 to include correct standard errors clustered by place and taking into account an IV estimation. For more information on this consult the following exchange on stackoverflow:
http://stackoverflow.com/questions/40367855/iv-estimation-with-cluster-robust-standard-errors-using-the-plm-package-in-r

Summary

Key R commands (functions):

- ivreg: runs IV (2SLS) regressions (requires package AER)

Key points/ concepts:

- ivreg works a lot like lm.
- The syntax for ivreg is as follows: ivreg(Y ~ X + W | W + Z, ...), where X is endogenous variable(s), Z is instrument(s), and W is exogenous controls (not instruments).
- Corrected standard errors can (and should) be incorporated into stargazer tables, requiring package ivpack.
- Diagnostic tests can be conducted using summary.
- The syntax for IV regression in a panel is the same as for a cross-section.

12. Models with sample selection

In this tutorial, you will learn how to:

- Estimate regression models with corrections for sample selection

Where to find what you need:

- Downloads: R tutorial scripts and data files `http://rpubs.com/wsundstrom/home`

Introduction

Sample-selection bias in OLS regressions arises when the process whereby observations are selected into the sample depends on the dependent (Y) variable in a way that creates a correlation between the error term and the regressors. A classic example, which we explore here, is the estimation of a wage or earnings equation where the wage is a function of observable skills (say, education), and where the decision to work for pay depends on the expected pay itself.

To illustrate, let us consider an admittedly oversimplified scenario, in which people only choose to work for pay if their expected wage would exceed some threshold value. Otherwise, work is not worth it to them, and they rely on other resources (spouse, parents, etc.). Then the sample of workers in the labor force, for whom we observe earnings, is selected from those in the population who are paid above that threshold. Among highly educated workers, most would earn above the threshold regardless, so they are pretty well-represented in the sample. But among the less-educated, usually low-paid workers, the sample is more selective and non-random: only those with unusually high earnings given their low levels of education clear the threshold and are observed in the sample of paid workers. In this scenario, education would be negatively correlated with the error term

among the persons observed in the sample, which biases the OLS coefficient on education downward.

Note that non-random sampling does not *necessarily* create biased estimates, as it does in the above scenario. Selection on the X variables need not cause bias. For example, suppose we had a sample consisting of a randomly chosen 2% of all women and a random 1% of all men. The sample is non-random, because selection depends on X = gender. But a comparison of men's and women's wages based on these samples would be unbiased: the selection is not causing a correlation between X (gender) and the error term of earnings.

Still, there are many reasons that a sample might be non-random in a way that could cause selection bias. For survey data, we know that response rates are low, and the kind of person who responds to a survey may be different from the kind who does not in ways that are correlated with outcome variables, on average. In other cases (such as our wage example), we only observe the dependent variable for people who make a certain decision related to that variable, such as whether to work for pay, or buy a particular product, or migrate.

Sometimes it is possible to estimate a model that corrects for the sample-selection bias that would be present in OLS. Essentially, what we need is a model of the selection process. Then once we estimate the determinants of selection, we can use that to correct for the bias in the regression. Our focus in this tutorial is on the mechanics of how to estimate this kind of model in R, which is straightforward. Understanding the theory behind the procedures and how to interpret the results is discussed in the Stock and Watson textbook.

Getting started

Start RStudio and open the script *t_selection.R* in your script editor. Edit the setwd(…) command for your working directory. Check the list of packages in the library commands and make sure you have installed all of them. Then run the script from the top through the data section.

The data set *Mroz87* is 1975 data on married women's pay and labor-force participation, from a well-known 1987 article by Thomas Mroz.[1] Mroz took his data from the Panel Study of Income Dynamics (PSID), an important national research survey. The data set is available as part of the sampleSelection package, which you should have installed and loaded. To see what variables are in the data, you can search RStudio Help for *Mroz87*.

[1] Mroz, T. A. (1987) "The sensitivity of an empirical model of married women's hours of work to economic and statistical assumptions." Econometrica 55, 765–799.

We will estimate a log wage equation for married women, in which the natural log of wages is a function of education, experience, experience squared, and a dummy variable for living in a big city. The issue here is that we can only observe the wage for women who are working for a wage. A substantial proportion of married women in 1975 were out of the labor force, so an OLS estimate of the wage equation would be estimated using a sample subject to considerable sample selection, potentially resulting in bias of the nature discussed above.

Because we have observations in the data set on non-participants (not working for pay), we can implement models that correct for the selection process. Note also that in the script we have created a new variable that is a dummy for the woman having children under 18. In effect, we will use this as an instrument for labor-force participation.

Estimating procedures: two-step and maximum likelihood

We are going to run three regressions to estimate the log wage equation, and compare the results. The first is simply OLS, using the sample of labor-force participants, for whom we can observe the wage. Here is the code, which should look quite familiar. Make sure you understand every bit:

```
# OLS: log wage regression on LF participants only
ols1 = lm(log(wage) ~ educ + exper + I( exper^2 ) + city, data=
subset(Mroz87, lfp==1))
```

Now let us turn to two alternative methods for estimating the wage equation with selection. The first is the so-called heckit or Heckman two-step correction model, named for the Nobel-prize winning economist, James Heckman. Heckman's idea was to treat the selection problem as if it were an omitted variable problem. A first-stage probit equation estimates the selection process (who is in the labor force?), and the results from that equation are used to construct a variable that captures the selection effect in the wage equation. This correction variable is called the inverse Mills ratio.

Let us look at the code:

```
# Two-step estimation with LFP selection equation
heck1 = heckit( lfp ~ age + I( age^2 ) + kids + huswage +
educ, log(wage) ~ educ + exper + I( exper^2 ) + city, da
ta=Mroz87 )
```

The set-up is in the spirit of the standard lm, but in this case note that we have two equations, separated by a comma.

The first equation in the command is the selection process, which is estimated as a probit:

```
lfp ~ age+I(age^2)+faminc+kids+huswage+educ.
```

The dependent variable in the heckit command is always the binary variable for being selected into the sample-- in this case, it is *lfp*, which is a dummy variable equal to one if the woman is participating in the labor force (and is thus being paid and observed for the wage equation). Then following the comma is the second equation, which is the wage regression of interest:

```
log(wage) ~ exper + I( exper^2 ) + educ + city.
```

The selection equation should include as regressors variables that are likely to affect the selection process. In our example, we want variables that could plausibly affect whether or not a married woman would be in the labor force. Examine the variables included here, make sure you know what they are, and see if you can understand why they would be included in the *lfp* equation.

In principle, the selection equation and the wage equation could have exactly the same set of regressors. But this is usually not a good idea. To get a good estimate of the selection model, *it is very desirable to have at least one variable in the selection equation that acts like an instrument*: that is, a variable that one expects would affect the selection process, but not the wage process, except through selection. In this example, the *kids* variable might play such a role, if we can assume that women with kids may be more likely to be stay-at-home moms, but for working moms having kids would not affect their hourly pay rate.

An alternative statistical method to accomplish the same result as the Heckman model is to estimate the selection model using maximum likelihood (ML) for both equations simultaneously. Note that the syntax for the command is virtually identical to the heckit:

```
# ML estimation of selection model
ml1 = selection( lfp ~ age + I( age^2 ) +
       kids + huswage + educ,
       log(wage) ~ educ + exper + I( exper^2 )
       + city, data=Mroz87)
```

In the tutorial, note that the selection command can be used to estimate the Heckman model as well by adding the option method="2step" to the preceding command.

Let us run the stargazer command to see the output from OLS and the two alternative selection models:

```
# stargazer table
stargazer(ols1, heck1, ml1,
          se=list(cse(ols1),NULL,NULL),
          title="Married women's wage
```

```
Married women's wage regressions
================================================================
                                Dependent variable:
                    ---------------------------------------
                                      log(wage)
                          OLS         Heckman        selection
                                     selection
                          (1)           (2)             (3)
----------------------------------------------------------------
educ                   0.1057***     0.1152***       0.1112***
                        (0.0130)     (0.0204)        (0.0171)

exper                  0.0411***     0.0427***       0.0420***
                        (0.0154)     (0.0134)        (0.0132)

I(exper2)               -0.0008*     -0.0008**       -0.0008**
                        (0.0004)     (0.0004)        (0.0004)

city                     0.0542       0.0447          0.0487
                        (0.0653)     (0.0692)        (0.0684)

Constant               -0.5308***    -0.7524*        -0.6586**
                        (0.2032)     (0.3926)        (0.2962)

----------------------------------------------------------------
Observations             428           753             753
R2                      0.1581       0.1589
Adjusted R2             0.1501       0.1490
Log Likelihood                                      -916.2869
rho                                  0.2234      0.1303 (0.2229)
Inverse Mills Ratio                0.1501 (0.2293)
Residual Std. Error   0.6667
F Statistic          19.8561***
================================================================
Note:                                *p<0.1; **p<0.05; ***p<0.01
```

```
regressions", type="text",
   df=FALSE, digits=4)
```

As you can see, the selection corrections did not have a big effect on any of the parameters of interest. The estimated returns to education and experience are a bit larger in the selection regressions, but the difference is really small. In this case, it appears that the OLS equation on the selected sample may not have been very biased after all.

In the panel below the coefficients, note that there is an estimate for *rho* in the selection models: columns (2) and (3). This is an estimate of the correlation of the errors between the selection and wage equations. In the lower panel, the estimated coefficient on the inverse Mills ratio is given for the Heckman model. The fact that it is *not* statistically different from zero is consistent with the idea that selection bias was not a serious problem in this case.

The final stargazer table in the script includes the option selection.equation=TRUE, which means the table will show you the first-stage probit model for *lfp*, if you are interested in what factors affect the selection process itself.

Summary

Key R commands (functions):

- heckit: runs Heckman two-step selection model (requires package sampleSelection)
- selection: runs maximum likelihood (ML) selection model (requires package sampleSelection)

Key points/ concepts:

- heckit and selection work a lot like lm, but have two equations.
- The syntax for either is as follows: selection(S ~ W1 + W2 | Y ~ X1 + X2 ..., ...), where the first equation is the selection equation, and S is a dummy variable for selection into the sample.
- stargazer can be used to obtain the second-stage estimates or modified to examine the selection equation.

13. Regression discontinuity

In this tutorial, you will learn how to:

- Run and interpret a regression discontinuity study
- Plot the data to show the treatment effect

Where to find what you need:

- Downloads: R tutorial scripts and data files `http://rpubs.com/wsundstrom/home`

Introduction

Does legal access to alcoholic beverages lead to higher mortality of young adults? It's not hard to imagine why it would: Easier access to alcohol could lead to more drunk driving or other forms of dangerous behavior. Using observational data, is there a good way to quantify whether this causal effect exists, and if so, how large it is?

One interesting approach to answering the question is to use a technique called regression discontinuity (RD) design. The idea behind RD is to take advantage of policies or decision rules that depend on a strict threshold in some measurable continuous variable. For example, in the case of alcoholic beverages, the minimum legal drinking age (MLDA) provides such a threshold. Up until a person's 21st birthday, they may not legally purchase or consume alcohol. But this changes the day they turn 21.

We refer to the continuous variable here (age) as the running variable. The policy or treatment (legal consumption of alcohol) is a discontinuous function of the running variable.

The discontinuity of the treatment is the key to RD designs. Consider a comparison of people who are a month shy of their 21st birthday with people who have just turned 21 (just one month older). If we suppose that the underlying health status of young adults who differ in age by only a month is very small, and further assume that the only major influence on

mortality of reaching 21 is legal access to alcohol, then a comparison of the average mortality rates across these two groups is an estimate of the causal effect of legal drinking on deaths.

Of course it could be the case that mortality rates actually do change with the running variable, age. For example, as they move into their early 20s, young adults tend to become more emotionally mature and may generally act more safely. If so, mortality rates could have a somewhat downward trend as a function of age. This could potentially mask the upward effect of alcohol-related deaths caused by crossing the MLDA of 21.

To sort out the causal effect of the treatment (MLDA), RD uses regression to control for the potential relationship between the outcome variable (mortality) and the running variable (age). The regression methodology is quite simple. Suppose we have data on average mortality rates by age. Then the simplest way to estimate the treatment effect of the MLDA is the following regression:

$$\overline{M}_a = \alpha + \rho D_a + \gamma(a - a_0)$$

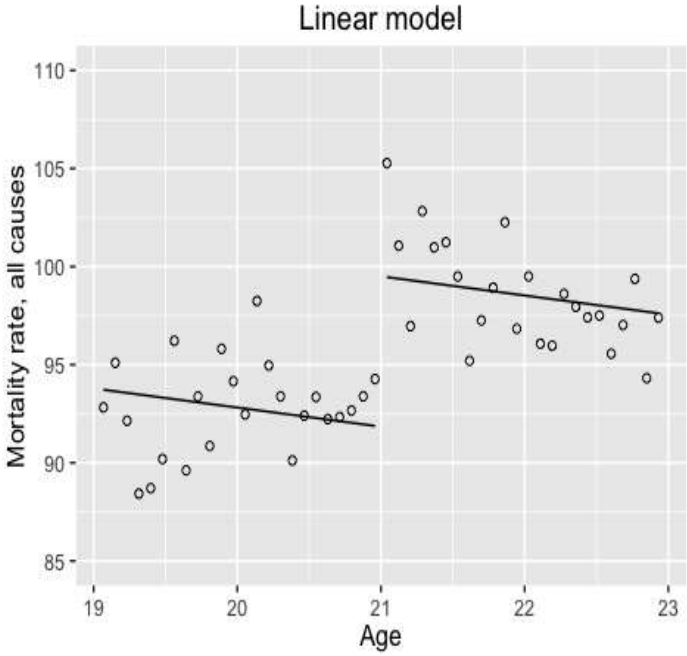
where $\overline{\mathrm{M}}_{\mathrm{a}}$ is average mortality rate at age a, a_0 is the threshold (21 years in our case) and D_{a} is a dummy variable for the treatment and is equal to 0 if a person's age is less than a_0 (21) and 1 if they are 21 or older. We could just use a rather than $(\mathrm{a} - \mathrm{a}_0)$ as the running variable here, but the reason we use $(\mathrm{a} - \mathrm{a}_0)$ will become clear below.

In this equation, our estimate of ρ is the estimate of the treatment effect. It is the amount by which the mortality rate jumps up (or down) at the 21st birthday, controlling for a linear relationship between age and mortality.

RD designs are a powerful tool for estimating a causal treatment effect whenever we have a treatment that is strictly a discontinuous function of a measurable running variable. Another example would be estimating the effect of attending a selective school on educational or economic outcomes, where admission to the school depends on a minimum threshold test score. Applicants with a test score just below the threshold would probably be quite similar in underlying abilities to those just above the threshold, but only the latter would attend the selective school.

Implementing RD in R

Let's take a look at how to implement RD using real data. In this case the data are average mortality rates of Americans who were ages 19-22 between 1997 and 2003. The mortality rate is deaths from all causes per 100,000 individuals, for 30-day increments of age. So there are 48 observations. These data were kindly provided by Professor Carlos Dobkin



of UC-Santa Cruz, who was the co-author of a published RD study on this topic.

The basic code can be found in *tutorial script t_rd.R*, using the data set mortality_data.csv. Open that script and run it through the data section. Note that we defined two new variables here:

```
mort$D <- ifelse(mort$Age>=21,1,0)
```

is the treament dummy, and

```
mort$agec = mort$Age - 21
```

is the running variable, namely age relative to the threshold of 21.

Now run the basic RD regression for the above equation (called parallel) and look at the results in the stargazer table. Note that this regression forces the slope on the running variable to be the same above and below the threshold. From the regression table, what is the estimated effect of crossing the MLDA on mortality? Is the effect statistically significant? Is the magnitude of the effect large, in your judgment?

With RD, it's often a good idea to plot the data with the estimated regression line. One reason to do this is to check for possible nonlinearities in the relationship with the running variable. As we'll see below, it's easy to deal with this if we need to.

To make the plot, we predict mortality rates for the regression we just ran and then add the predicted line separately below and above the age cutoff. Here's the code followed by the plot:

```
# Predict the mortality rate from the regression
mort$pred_par <- predict(parallel)

# Note that we add the before and after
# prediction lines separately to show the
discontinuity
ggplot(mort, aes(x=Age, y=All)) +
 geom_point(shape=1) +
 labs(y = "Mortality rate, all causes",
 x = "Age, title = "Linear model") +
 ylim(85,110) +
 geom_line(data=subset(mort,Age<21),
 aes(x=Age,y=pred_par)) +
 geom_line(data=subset(mort,Age>=21),
 aes(x=Age, y=pred_par))
```

Nonlinearities in RD

In this case the simplest RD specification looks pretty good. But we might suppose that the relationship between mortality and the running variable (age) could be more complex. For one thing, the slope could change across the treatment threshold. Eyeballing the data points in the graph, you might discern that this is happening here, with the slope after age 21 looking more negative. There's a plausible reason to think this might happen in this case: perhaps once a person turns 21 and gains more experience with legally drinking alcohol, they become aware of their limits and start drinking more responsibly, reducing their mortality risk.

The remainder of the tutorial script runs some additional specifications that allow the slope to differ below and above the threshold and allow curvature using a quadratic or cubic in the running variable. Note that we use a coding shortcut here, with the asterisk inside the regression automatically creating an interaction term plus the two "main effects." That is,

```
linear <- lm(All ~ D*agec, data=mort)
```

gives the same results as

```
linear <- lm(All ~ D + agec + D:agec, data=mort).
```

Allowing the slope to differ across the threshold is the reason we use agec = age - 21 as the running variable rather than just age. If you take the regression equations and plug in age = 21 plus or minus a tiny amount, you will see that in each case the estimated coefficient on D represents the vertical "jump" in the prediction. This makes it very easy to judge the size and significance of the discontinuous treatment effect.

Run these specifications and examine the results in the table. Which specification do you prefer? Is there evidence of a change in the slope after age 21? Is there evidence of curvature in the relationships? Explain.

Summary

Key points/ concepts:

- Regression discontinuity involves a treatment or policy rule that is a discontinuous function of some continuous running variable.
- Interaction terms can be used to allow the running variable to have a nonlinear relationship with the outcome variable.
- You should always graph the relationship between the outcome and the running variable to see whether your model has captured the relationship well.

14. Randomized experiments

In this tutorial, you will learn how to:

- Estimate a treatment effect in a randomized control trial (RCT)
- Identify key threats to validity to an RCT

Where to find what you need:

- Downloads: R tutorial scripts and data files `http://rpubs.com/wsundstrom/home`

Introduction

The past several chapters on instrumental variables, sample selection, and regression discontinuity have dealt with corrections to bias in regression estimation. Bias typically arises when an explanatory variable is not independent or exogenous. That is, the explanatory variable of interest is correlated with the error term in the regression. IV, sample selection corrections, and RD are techniques that correct for some types of bias.

Another increasingly common way to correct for bias is to use experimental data, rather than observational data. In an experiment, the economist manipulates the economic environment much the way a natural scientist manipulates the physical environment. By manipulating the environment, a researcher has more confidence that an omitted variable is not responsible for change in the outcome of interest.

This confidence is what enables the researcher to learn about some causal process. A causal process is a process where a researcher thinks something like this: "The outcome changed because the explanatory variable changed." When there is bias, we cannot be very confident that the change in the explanatory variable caused the change in the outcome. We worry that the association might be spurious because something else, an omitted variable, also changed. In an experimental setting, we take care to ensure that there are no likely omitted variables.

Potential outcomes and average treatment effect

In thinking about how to estimate a casual effect, social scientists examine the issue from a perspective known as the "potential outcomes" framework. We imagine that for the units of observation there are values of the outcome variable that could, in principle, be known for different values of the explanatory variable. The simplest case is when the explanatory variable is a dichotomous variable measured as 0 or 1. The variable indicates whether a treatment was delivered, as in medicine. When the treatment equals 1, an observation will have potential outcome Y(1), and when the treatment equals 0, the potential outcome is Y(0). We can then define the Average Treatment Effect (ATE) as the average over the *i* observations of the differences in potential outcomes:

$$ATE = (\frac{1}{n})(\sum_{i=1}^{n}(Y_i\,(1)) - \sum_{i=1}^{n}(Y_i(0)))$$

We would like to make an inference about the value of this treatment effect, using a sample of data. There are several issues, however. First, we do not observe both potential outcomes for each unit of observations, we only observe one outcome. The subjects (individuals, households, regions) either get the treatment or they do not. If they get the treatment we observe Y(1) but not Y(0). If they do not get the treatment we observe Y(0) but not Y(1). In other words, we do not observe the counterfactual for each observation. Another problem is that we measure with error, so there is always room for our sample estimate of the ATE to deviate from the true relationship. Moreover, our sample itself could be an outlier or odd sample, so we need to calculate a standard error for our estimate based on properties of the sample (i.e. how much variation is there among the units of the sample).

There remains of course the possibility that an omitted variable is causing the outcome rather than the treatment. But this is less likely in an experiment. If we assign the treatment randomly, so that the treatment is uncorrelated with omitted variables, then some math will show that the expected value of the difference of the actually observed mean outcome for those receiving the treatment and the actually observed mean outcome for those not receiving the treatment will be equal to the ATE. Let us suppose that m units in the sample receive the treatment, and n-m units are in the control group (treatment=0), then:

$$ATE = E(\sum_{i=1}^{m} \big(Y_i(1)\big)(\frac{1}{m}) - \sum_{i=m+1}^{n-m} \big(Y_i(0)\big)\,(\frac{1}{n-m}))$$

So the difference in actual means will be an unbiased estimate of the ATE.

Of course, several assumptions are needed for the difference in means to be convincingly interpreted as an unbiased estimate of the ATE. These assumptions go by the following labels: non-interference, excludability, response bias, non-compliance, and no systematic attrition.

Non-interference refers to the possibility that people's actions are affected by their treatment status relative to other people in the experiment. That is, everyone else's treatment status might affect what a person does. Suppose we are in a cash transfer experiment, with the objective of seeing how poor people respond over the longer term to receiving cash transfers. Suppose many people in the treatment group share their cash transfer with their neighbors, because they are also very poor but were assigned to the control group. The cash transfers will appear to be ineffective, because there will be no difference between the treatment and control groups. The non-interference assumption rules this out.

Another assumption is that other actors outside the experiment are not responding to the treatment and control. Continuing the cash transfer example, if you are assigned to the control group, but then another development organization offers you the same cash transfer as received by the treatment group, and the experimenters do not know about this, then again the treatment will appear ineffective because there will be no difference between treatment and control. The experimenter must be able to reasonably exclude the possibility of reactions by others to the experiment itself.

Another concern is respondent bias. If I know I have been selected into the treatment group, then I may respond to the experimenter when asked about my economic status in ways that I think the experimenter wants to hear, rather than with the actual outcomes. Likewise, a control group member may feel some spite at not having been selected, and so "underestimate" his or her outcomes. The treatment appears very effective, but the result is due to response bias.

Non-compliance refers to the fact that some individuals may not comply with the treatment. That is, they may receive the treatment but decline to actually carry out or implement the program that constitutes the treatment. This is very common whenever an experiment involves extending an opportunity to avail oneself of a benefit rather than directly administering a treatment. Some who are offered the opportunity will take-up the opportunity, others will not. Conceptually and in practice social scientists often want to distinguish between the effects of a treatment on

those who actually take up the treatment, and the issue of why some people take up a treatment and others do not. To take a rather extreme example, suppose that the treatment group is offered an opportunity to sign up for an innovative but expensive health insurance program. Nobody signs up. There is then no difference in outcomes between the treatment and control group, a year later. The ATE measures the effect of the treatment, and would seem to be zero in this case. But since nobody took-up the insurance, we do not actually know whether it would have changed outcomes. Our interpretation of the ATE needs to be more nuanced.

Attrition from the sample poses a similar problem. Imagine that the same experiment were conducted. Most of the control group who are not offered the insurance leave the program and refuse to be interviewed and so their outcomes are missing. Suppose those in the control group who remained were people who were having really bad outcomes (bankrupt due to unanticipated health expenses) and they hoped maybe by continuing to participate they might get some benefit. Now the ATE will be quite high, with the treatment group (regardless of their take-up) having much better outcomes than the control group. The estimate of the ATE is quite misleading unless the differential attrition is dealt with in the analysis.

Common sense is often a good guide for thinking about these kinds of issues: does the experiment and assignment into treatment and control seem like it might actually generate these kinds of biases? If so, then more thinking needs to be done in order to arrive at a good experiment and better analysis of results.

In the remainder of this chapter, we assume that response bias, interference or spillover effects from one subject to the other, and other actor responses to the experiment are not significant in terms of the size of their effects. We focus instead on corrections for non-compliance and attrition.

To examine these issues, open the script *t_experiment.R* in your script editor in RStudio. Edit the setwd(…) command for your working directory. Check the list of packages in the library commands and make sure you have installed all of them. Then run the script from the top through the data section.

The code in this script is adapted from one of the *Method Guides* produced by the Evidence in Governance and Politics group, entitled "10 Types of Treatment Effect You Should Know About."[2] The EGAP group has numerous other excellent guides, with R code for replicating and learning.

[2]http://egap.org/methods-guides/10-types-treatment-effect-you-should-know-about

Generating a data set

Instead of loading in a data set, the R code generates a simulated dataset.

```
# Generate a dataset
set.seed(1234) # For replication
N  2000 # Population size
b0 <- -1.6
b1 <- 3
b2 = 7.3
b3 = -4.3
x1 <- runif(n=N, min=18, max=60)
x2 <- runif(n=N, min=5, max=25)
x3 <- runif(n=N, min=0, max=1)
err <- rnorm(n=N, mean=0, sd=20)
# Potential outcome under control condition
Y0 = runif(N)
# Potential outcome under treatment condition
Y1 = Y0*10
# Treatment: 1 if treated, 0 otherwise
D = sample((1:N)%%2)
# Outcome observed in sample
Y1 = b0 + D1*(Y0*10) + (1-D1)*Y0 + b1*x1 + b2*x2 + b3*x3
+ err # Outcome in population
```

Recall that the runif command generates random numbers using the uniform distribution, while the set.seed command will ensure that each time you will get the same random numbers. You can get different random numbers by setting a different "seed" instead of 1234, or by not setting a seed. The sample command takes a random sample of specified size (here N) from the elements of x using either with or without replacement. Notice that the code creates an outcome Y1 that depends on the treatment and three covariates (x1, x2 and x3) and also has a normally distributed error term. The magnitude of the treatment effect depends on an unobservable variable Y0, which varies from 0-1 and has mean .5, so the average treatment effect is 5.

Estimating the ATE

A direct way to estimate the ATE is to compute the difference in means between the treatment and control group, and then compute the standard error of that difference (the estimator of the treatment effect), and use the standard error to compute a confidence interval or test the hypothesis that the ATE is equal to zero.

The following two commands estimate the ATE, and the ttest command displays the statistical significance of the difference in means.

```
summaryBy(Y1 ~ D1,  data=samp , FUN=c(mean),na.rm=TRUE)
ATE1 = with(samp,mean(Y1[D1==1])-mean(Y1[D1==0]))
t.test(Y1~D1, data=samp , FUN=c(mean), na.rm=TRUE)
```

Standard error of treatment coefficient falls with controls

	Dependent variable:		
	Y1		
	(1)	(2)	(3)
D1	4.714*	5.555***	5.385***
	(2.679)	(1.039)	(0.884)
x1		2.965***	2.968***
		(0.043)	(0.036)
x2		7.311***	7.258***
		(0.090)	(0.076)
x3			-4.243***
			(0.153)
Constant	203.448***	-22.450***	-0.385
	(1.878)	(2.323)	(2.051)
Observations	2,000	2,000	2,000
R2	0.002	0.850	0.892
Adjusted R2	0.001	0.850	0.891
Residual Std. Error	59.897	23.240	19.759
F Statistic	3.097*	3,765.737***	4,098.280***

Note: *p<0.1; **p<0.05; ***p<0.01Standard error of treatment coefficient falls with controls

When the commands are run, you can see that the difference is statistically significant and about equal to 5, which is what the simulation has as the ATE.

Notice then what happens when regressions are estimated with the x1, x2 and x3 control variables. The estimate of the effect of the treatment is close to 5 for each regression, but the precision of the estimate increases as more controls are added (that is, the standard error of the estimate decreases from 2.68 to .88).

Correcting for non-compliance

The script that creates the simulated dataset also creates two outcome variables, Y2 and Y3, that reflect two different non-compliance scenarios. The first scenario has compliance vary for those who receive the treatment. Some of those offered the treatment do not comply. The variable D2 measures whether they comply or not. Compliance depends on the value of the unobserved variable Y0; those with high Y0 are more likely to comply. C2 indicates whether they are assigned and then "take up" the treatment.

In the second non-compliance scenario, the outcome variable Y3 is constructed by having non-compliance operate at both ends. Some in the treatment group will not take up the treatment, while non-compliers in the control group find ways to take up the treatment. In the Y3 case, the propensity to be a non-complier varies with treatment status itself (getting the treatment means being more likely to comply, so not getting the treatment means being less likely to comply) and with the variable Y0.

The regression commands that follow in the tutorial code use the instrumental variable technique from the earlier chapter to estimate the treatment effect. Since the observed indicators of whether the person took the treatment or not (C2 and C3) are no longer random variables and instead depend on Y0, the estimates of the treatment effect will be biased. Likewise, the variable D1, the treatment assignment, mis-measures whether the person actually received the treatment or not. But the intent-to-treat (ITT) variable D1 is a good instrumental variable. It is correlated with the actual observed indicator of treatment, and since it was randomly assigned it only affects the outcome through the treatment effect itself. As the regression results show, the IV estimator can be quite different form the OLS estimate of the treatment effect.

Dealing with attrition

The script that creates the simulated dataset also creates outcome variables Y1attrit, Y2attrit and Y3attrit that have treatment and control observations missing, for the regular case and the two non-compliance cases. The simulation supposes that observations from the treatment group are more likely to be missing when the Y0 value is low, while observations from the control group are more likely to be missing when the Y0 variable is high. The idea is that Y0 proxies for the person's understanding of the magnitude of the treatment effect. Perhaps a person in the treatment group with a low Y0 thinks the treatment is unlikely to have much effect, and so they drop out. For the control group, a person with a high Y0 perhaps feels their opportunity cost of remaining in the experiment (and answering questions) is high and so they drop out when they are assigned to the control group.

Observations that have attrited out of the sample are no longer observed. There are two methods that are commonly used to deal with these situations. Both involve making assumptions about what the values of the unobserved outcomes might be.

In the extreme bounds method, the assumptions made are those most and least favorable to finding an effect of the treatment. These define an upper and lower bound for the treatment effect. In the upper bound case, the missing outcomes for the treatment group are assumed to be the most favorable to finding a treatment effect, while the missing outcomes for

the control group are assumed to be the worst outcomes. In the lower bound case, those in the treatment group that have attrited are assumed to have had the lowest possible outcome, while those in the control group who have attrited are assumed to have had the best possible outcome. In each case, lower and upper bounds, the bounds are weighted averages of the observed and unobserved (and assumed) outcomes. The weights are the proportions in each sample group that are observed and the proportion missing.

A second method was introduced by Lee in a paper first circulated in 2002, but not published until 2009.[3] The method involves estimating the treatment effect for the responders (that is, not the same ATE considered above). Since the sample of responders is likely to be skewed in some way (i.e. possibly more of the treatment group responding; possibly more of the control group responding), bounds are established by trimming the larger group at the upper end of the outcome variable and then the lower end of the outcome variable. This then enables calculation of an upper and lower bound for the treatment effect for responders. The bounds are valid as long as one assumes that the there is a monotonic relationship (in either direction) between the treatment and the likelihood of attrition.

The code in the script calculates these bounds for the two methods. Run the code and try to interpret the results.

[3] David S. L ee, Training, Wages, and Sample Selection: Estimating Sharp Bounds on Treatment Effects, in Review of Economic Studies, 76(3) 1071-1102.

Summary

Key R commands (functions):

- `runif` generates a random variable using the uniform distribution
- `sample` generates a 0-1 variable that randomly designated observations into one group or another
- `pnorm, rnomr, rbinom` are commands used in simulation to create random variables
- `data.frame` puts vectors together into a data frame
- `max, min, mean, sum` are operations conducted on a vector
- `quantile` finds the value from the elements of a vector in the specified quantile

Key points/ concepts:

- An average treatment effect (ATE) for a program or policy can be estimated as long at the treatment is assigned randomly to the units of observations, and other assumptions hold.
- Other important assumptions are about non-compliance, attrition, non-interference, excludability, and response bias.
- The instrumental variables method can be used when non-compliance is an issue; the intent-to-treat assignment variable can be an instrument for the actual take up (or not) of treatment.
- Bounds analysis can be used to estimate treatment effects where there is non-random attrition.

15. Analysis of statistical power

In this tutorial you will learn how to

Determine the power of a statistical analysis involving
- the test of difference in means from two populations or subgroups
- the test of a hypothesis about a regression coefficient

Introduction

Suppose you think that a variable of interest is important in determining an outcome. Other control variables also matter in explaining variation in the outcome. You estimate a multiple regression model, and obtain an estimate of the coefficient of the variable of interest, along with the standard error of the estimate. The p-value is greater than .10, say, so by normal conventions the null hypothesis that the coefficient is equal to zero cannot be rejected. Are you able then to say confidently that the effect is zero, that this variable does not affect the outcome? (Notice the daring use of the homonyms effect and affect in the same sentence!) The problem is that given the way much analysis is reported, the reader cannot tell whether the null result is due to the null hypothesis of no effect being true, or due to the statistical procedure having low *power* to reject the null hypothesis when the alternative hypothesis is indeed true. The power of a test is the probability that the test correctly rejects the null hypothesis under the assumption that the alternative hypothesis is actually true. One might think of the power of a test as a way of measuring how well a statistical test is able to detect a real difference when applied to a sample from a population.

This chapter introduces the essential ideas and provides R code for doing power analysis. The concept of the power of a statistical test when applied to a sample of given characteristics is often neglected in the social sciences. This has had an unintended outcome: because few social scientists are trained in power analysis, a kind of equilibrium seems to have been

reached in academia, where few academics report on research without statistically significant effects, and so students do not have to be taught power analysis in order to understand published research. Many leading econometrics textbooks, for example, contain no discussion of power analysis. Many published journal articles do not mention or discuss power analysis.

It is a bad equilibrium we hope you will work to remedy. Please take the time to work through the examples in the chapter, using the script *t_power analysis.R*; learning about power early on can save many headaches later on! More importantly, it can really help in designing an empirical study, by forcing the researcher to answer the question of what, exactly, is the hypothesis to be tested and what is a reasonable alternative hypothesis.

What is power analysis?

Before diving in to power analysis in the multiple regression setting, it helps to present the basic idea in a familiar setting, the comparison of two means. Suppose we have data on the sleep patterns of male and female college students. We want to know whether males sleep more hours each night compared with females. We measure the mean hours of sleep for 18 males and determine it is 7.00 hours while the mean hours of sleep for 19 females is measured as 7.13. Suppose that both samples of males and females have the same sample standard deviation of .50. We assume the difference in the means is normally distributed. We carry out the test for whether the difference in means of .13 is statistically significant. We first calculate the standard error of the difference in the means, $\text{sqrt}[(s_1^2/n_1) + (s_2^2/n_2)] = .16$ and then the t-statistic (.13/.16) = .79. Since it is a two-tailed test, the p-value is two times the probability, for the standard normal distribution, of observing a value below -.79. The p-value is then .43. We conclude there is no difference in sleep patterns, because we have set our significance level at α=.05 and the p-value is .43.

A friend says, "Maybe the sample size is too small, though. Your test has no power." Indeed, our sample of 18 males and 19 females does seem small. What is the probability that we would reject the null hypothesis of no difference if in reality males on average slept .20 more hours than females, and there was a common standard deviation of .50 hours when sleeping? This probability is the power of a test. It is important to point out that the power depends on the alternative hypothesized effect size (the .20). For different alternative possible effect sizes, a different power is calculated.

To calculate the power, first note that the cutoff "critical t-statistic" under HO, the null hypothesis that there is no difference, is 1.96 (either 1.96 or -1.96 in a two-sided test). Define a critical value X for the difference of

means that solves the equation .05 = 2*pnorm(-|(X-0)/(*se*)|), where pnorm is the cumulative standard normal probability, and *se* is the standard error of the difference in the means (which we calculated earlier). Solving, we find $X^{critical}$ = .32. If the absolute value of the difference in means (X) in our sample were larger than .32, we would reject the null hypothesis that the difference is equal to 0.

Now, if the true difference were .20, we can ask how often we would commit a Type II error of accepting the HO when the alternative hypothesis HA was really true. That is, what is the chance that we would find an X lower than .32 or greater than -.32 if the true difference were .20 and the standard error were as above? This probability is given by prob(x>-.32 and x<.32) when x~N(.20, se)). In R, this probability is:

```
(pnorm((.32-.20)/(se))-pnorm((-.32-.20)/(se))).
```

This turns out to be .77.

So there is a 77% probability we would accept HO even when the true difference in means is .20. The power of the test is said to be .23, because there is only a 23% probability of rejecting the null hypothesis of no difference when the true effect is .20. If the true effect were .30, the probability of a Type II error would be .56, and the power would then be .44; there would be a 44% chance of rejecting the null hypothesis of no difference when the true difference is .30 and we draw a sample of 37 from the population.

Varying the sample size also changes the power. If the true effect were .20 but we drew a sample of 100, equally spread across the two groups, the power would be .52.

The power of the test for the difference in means depends on assumptions about the true effect size, the standard error of the estimate of the difference in means, and the sample size. An assumption about the true effect size is needed in order to know what the power of the test is. If we think the effect size is huge, then we only need a small sample in order to have high power. If we think the effect size is small, we need a large sample. The assumption about effect size should be guided by common sense understanding of the underlying factor responsible for difference and its likely effect on the outcome, as well as by prior research on the question. For example, to determine whether a small innovation in technique in a sport (like holding a tennis grip at a slightly different angle) would change outcomes, one might need a very large sample, because the effect of the change is likely to be very small. On the other hand, if a brain surgeon slightly changes how she holds a scalpel, the effect of the change might be very large. Context is important for thinking about what is a reasonable effect size.

Let us consider some other examples and variants of the analysis of power. Suppose our null hypothesis is that there is no difference in hours of sleep between males and females, and our alternative hypothesized effect size is that there is a .25 difference in hours of sleep. Then, what is the minimum sample size of males and females (assume equal numbers of each group) needed in order to conduct a one-sided t-test with power = .80? Power = .80 implies rejecting the null hypothesis 80% of the time if the alternative hypothesis is indeed true. Assume that the difference in sample means divided by the standard error (the t-statistic) is distributed normally. The critical value of the t-statistic for a one-tailed test at α=.05 is 1.65. This implies the critical value of the difference in means, X, is given by the equation: $1.65=(X^{critical})/(se)$, where *se*, as before, is given by $sqrt[(s_1^2/(.5n)) + (s_2^2/(.5n))]$. So $X^{critical} = 1.65*se$. The following equation may be solved for n, where we again assume the standard deviation for each sample is equal to .5, that is $s_1 = s_2 = .5$:

$.80 = 1 - \Phi(z<(X^{critical} - .25)/(se))$
$.80 = 1 - \Phi(z<(1.65 - .25/se)$
$.2 = \Phi(z<(1.65 - .25/(se))$
$qnorm(.2) = 1.65 - .25/se$
$se*(qnorm(.2) - 1.65) = -.25$
$se = .25/(1.65 - qnorm(.2))$
$sqrt[(s_1^2/(.5n)) + (s_2^2/(.5n))] = .25/(1.65 - qnorm(.2))$
$[(s_1^2/(.5n)) + (s_2^2/(.5n))] = [.25/(1.65 - qnorm(.2))]^2$
$(.25*2)/(.5n) = [.25/(1.65 - qnorm(.2))]^2,$
$n = (2*(.25*2))/(.25/(1.65 - qnorm(.2)))^2$

When solved we obtain n = 100, or 50 males and 50 females.

Consider another example. Suppose we are reading an academic paper that finds that giving a random sample of schoolchildren an hour of tutoring each day does not significantly affect their grades on a test, compared with a control group. The sample size is 25 students in each group, the difference in means is 4.4 (with the treatment group higher) and the standard error of the difference in means is 3.1. The t-statistic of the difference in means is then 4.4/3.1, which has a p-value of .16, and thus the difference is not statistically significantly different from zero.

Given these numbers, can we determine whether the randomized control trial was adequately powered? To do that we need to make an assumption about what is a reasonable effect size. Let us suppose that a change in the difference in means of .3 standard deviations would be a large effect size. We assume the variance of the test scores for the two groups is the same. We want to first "back out" from the data what the implied standard deviation is of the test scores. Now, the variance of the estimate of the difference in means is given by:

$$
\begin{aligned}
Var(\overline{X}_i - \overline{X}_j) &= \text{var}(\sum X_i / n_i - \sum X_j / n_j), \\
&= \text{var}(\sum X_i / n_i) + \text{var}(\sum X_j / n_j) + \text{cov}(\sum X_i / n_i, \sum X_j / n_j) \\
&= \frac{n_i \,\text{var}(X_i)}{n_i^2} + \frac{n_j \,\text{var}(X_j)}{n_j^2} \\
&= \frac{2\,\text{var}(X_i)}{n_i}
\end{aligned}
$$

The standard error of the estimate of the difference in means (what was reported in the academic paper) is the square root of the variance of the estimator, so we have,

$$
\begin{aligned}
se &= \sqrt{\frac{2\,\text{var}(X_i)}{n_i}} \\
se^2 n_i &= 2\,\text{var}(X_i) \\
\text{var}(X_i) &= \frac{1}{2} n_i se^2 \\
st.dev.(X_i) &= \sqrt{\frac{n_i}{2}} se
\end{aligned}
$$

From this expression we find that the standard deviation of the test scores is

$$\sqrt{12.5} \cdot 3.1 = 10.96 .$$

Our reasonable effect size was assumed to be .3 of a standard deviation, and this equals 3.28.

So if the true effect size were 3.28 for the difference in means, what is the probability of rejecting the null hypothesis? The null hypothesis is rejected if the observed difference in scores is 1.96 times the standard error of the difference in scores; the null is rejected if |X|>1.96*se*. The probability of rejecting the null is Prob(|X|>1.96*se*) when X~N(3.28, 3.1). So the power of the test = 1 - prob(X>-6.08 and X<6.08) when X~N(3.28, 3.1), where 6.08 is 1.96*se*. In R, this probability is:

```
1-(pnorm((6.08-3.28)/(3.1)) -
   pnorm((-6.08-3.28)/(3.1)))
```

which gives a value of .18. So the power is extremely low; even if the true difference in means were 3.28, we would reject the null hypothesis of no difference in only 18% of samples. The reason is that the variance of our estimate of the difference in means is very high, because the sample size is quite small. We can calculate that we would need the sample size to be

about 175 in each group (or 350 overall) to have 80% power to detect an effect size of .3 standard deviations! Our sample of 25 in each group is ridiculously small, and could only detect a huge effect size.

Power is complicated with very small samples

One of the reasons that power is so often ignored in many research projects is that there are many assumptions that must be made. Here, for example, are some of the assumptions that must be made just for considering the power of the test of the simple comparison of means:

- Assumptions about the distribution of the test statistic
    - Normally distributed
    - Student's t distribution
    - Other
- Assumptions about variances of variables for two groups when calculating the standard error of the test statistic
    - Equal variances
    - Unequal variances
- Assumptions about sample sizes for groups
    - Same sample sizes
    - Different sample sizes

So for sample sizes that are small (say under 30 in each group) there are many different power calculations that could be made (practically speaking they will all generate very similar effects with 30 in each group, but will diverge substantially as sample sizes get smaller). Moreover, many textbooks in statistics are more concerned with power for tests that involve non-normal distributions, such as the Student's t-distribution, Chi-squared distribution, and F distribution. These distributions are not symmetric, and the distribution depends on the assumed effect size. Calculating power when the test statistic follows a non-central distribution is thus more complicated.[4]

Power in multiple regression analysis

Most economists are interested in the precision of an estimated effect obtained from regression analysis where covariates are included. Normally, the standard error of the estimate of an effect will be smaller when

[4] See Christopher L. Aberson, *Applied Power Analysis for the Behavioral Sciences* (Routledge, 2011), for a good discussion using SPSS.

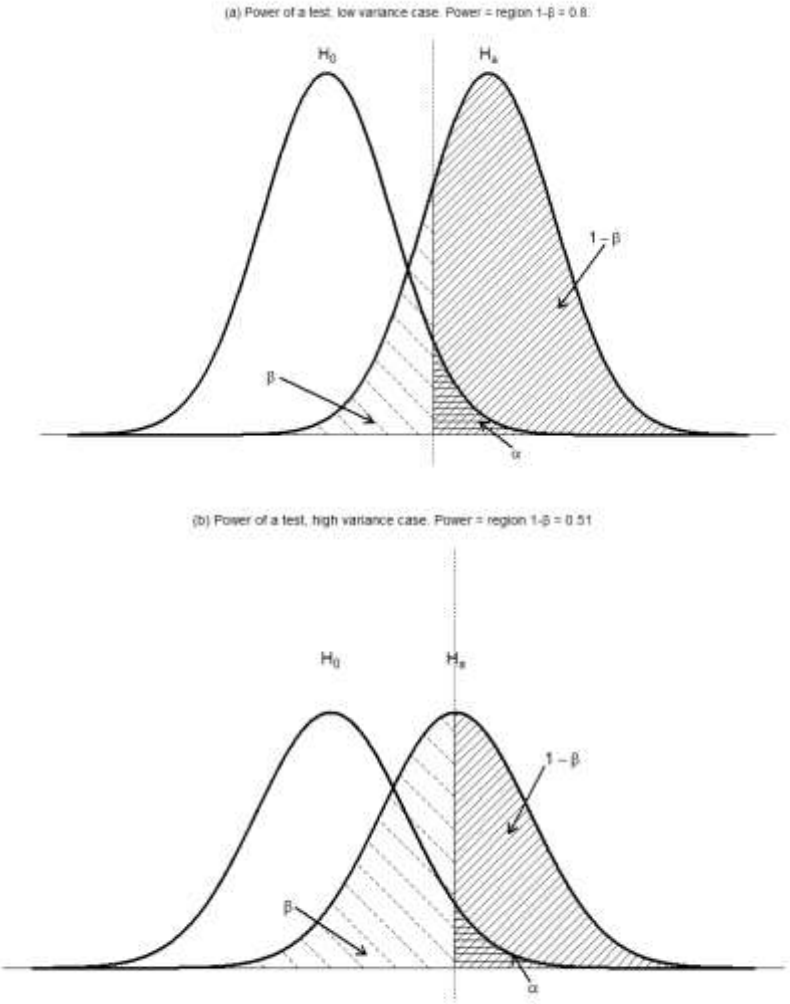
controlling for important covariates, and so the power to detect a given effect size will be greater.

Suppose we continue the example above and imagine a regression that estimates the effect on test scores of a program that gives an extra hour of tutoring per day. The estimate of the effect is 4.1 with a standard error of 2.5 in a regression with some covariates included and the same sample of 50, with 25 in each group. The standard error of the estimate of the effect has fallen from 3.1 to 2.5. The t-statistic on the coefficient then is 4.1/2.5 = 1.64 which is significant at the 10% level but not at the 5% level. Let us stick to the conventional 5% level. Can we conclude that there is no effect? Or is the sample size too small to detect a reasonable effect? We need to conduct power analysis to answer that question.

We can use the estimate of the standard error to determine what the sample size would have to have been to detect an effect size of .3 standard deviations (recall from earlier that would imply a coefficient of 3.28). If an estimate of 3.28 were to be statistically significant at the conventional 95% level, the standard error of the estimate would have to be such that 3.28/*se* > (1.96+.84), or *se*<1.17. Since the actual standard error is 2.5, and the standard error is proportional to $1/\sqrt{n}$, the sample size has to be increase by a factor of $(2.5/1.17)^2 = 4.55$, or from 50 to 228. Note this is substantially smaller than the 350 total sample required for the comparison of means. The calculation also makes clear that the non-statistically-significant finding of the regression is more about the low power of the regression than about the absence of a reasonable effect size in the population from which the sample is drawn.

Figures (a) and (b) illustrates the main idea, for two different levels of the assumed variance of the distribution of outcomes. The H_0 normal density curve represents the distribution of estimates of the treatment effect estimated in the regression under the assumption that the true treatment effect is zero. The H_a density curve represents the distributions of estimates of the treatment effect estimated in the regression under the assumption that the true effect is some reasonable non-zero positive effect. As before, let us assume this effect is 3.28. Marked on the H_0 curve is the value of the difference in means that is 1.96*se*, so that $\alpha/2$ equals .025. The crosshatched area under the curve to the right of this point is the probability that a sample drawn randomly from the population might lead to an estimated effect great than 1.96*se*, and this probability for a one-sided test is 2.5%.

The critical value on the x-axis can then be used to calculate the probability of rejecting the null hypothesis under the assumption that the



alternative hypothesis were actually true. This is the shaded area marked as 1 – β. This value is the power of the test. Note that the value is much lower when the variance is higher.

Determining power through simulation analysis

Another way to determine the power of a test, given an assumption about effect size and standard error, is to run a simulation. Continuing with the example above, R can be programmed to draw 5000 random samples of fixed size (e.g., 50 for each gender, as above) from a distribution (e.g. a

normal distribution with standard deviation of .5 as above, and mean of 7 for men and 7.25 for women). For each of the samples, the one-sided t-test for difference in means can be performed, and the results stored. Then the frequency of rejection of the null hypothesis of zero difference can be calculated from all of the simulated samples.

16. Miscellaneous useful code (unfinished)

In this tutorial, you will learn how to:

- Use various miscellaneous code segments to carry out operations that data analysts often carry out.

Introduction

The hardest part of any data analysis project is usually preparing the data, rather than running regressions or generating tables. Data needs to be merged, cleaned, aggregated, split, appended and collapsed, to note just a few of the terms used by data analysts. This chapter review a number of R features that are useful in preparing data.

Reading in datasets

The fread command is used to read in very large datasets that are stored as ascii text files (usually with filename extension .txt). If the file has the same number of columns for every row (so it is balanced") then fread will be very fast and correctly interpret numerical, factor and logical variables. Fread is part of the package data.table, so the following commands would be used:

```
install.packages("data.table")
library(data.table)
d2=fread("C:/data/datatoinput.txt")
```

Creating leads, lags, percent changes

Creating a new variable that is the percent change from the previous value. Sometimes a data set has multiple observations for a unit of observation (individual, household, region, country). That is, the data is in the form of a

pane. You might want to create a new variable that is the percentage change from one year to the next. Say the data is for countries, and goes from 1960 to 2015. You want the annual percent change in GDP. The following code will do the trick.

```
# sort by country and year
wdim <- wdim[order(wdim$country, wdim$year),]

# duplicate the GDP column as otherwise you lose
# the values when calculating the percent change
wdim$GDP_change<- wdim$GDP

# define function that will calculate the
# percentage change
pct <- function(x) {((x-lag(x))/lag(x))}

# use piping to apply it to the entire column
wdim    <-    wdim    %>%    group_by(country)    %>%
mutate_each(funs(pct), c(GDP_change))
```

Notice the %>% symbol. This is called "piping" and is part of the package called magrittr (the joke is Magritte's pipe, do a Google image search). More information on maggritr is here:
https://cran.r-project.org/web/packages/magrittr/vignettes/magrittr.html
Obviously, the percent change cannot be calculated for 1960 (since you do not have data for 1959!)

Recode missing values

Recode missing and infinite values to be zero or missing.

```
# Recode a missing value to be zero
mydt$score <- ifelse(mydt$score==NA, 0, mydt$score)
# Recode Inf values to be NA
wdim$GDP <- ifelse(wdim$GDP == Inf, NA, wdim$GDP)
```

Collapse a dataset

Very often, a dataset will have multiple observations for a given unit of observation, and you want a single observation for that unit. For example, you may have multiple scores on quizzes, homework scores and hours spent studying for a student, and you want an average of those scores. If the data is in the "long" format, where each student enters as multiple rows, with each row having a quiz, homework score and hours studying for one week, then this dataset needs to be "collapsed" so that a new dataset is created with one row per student. The R function summaryBy of the doBy package does this.

```
# collapse the outcomes by week and test
# first write a function to calculate
```

```
# mean, median and number
# for mean and median, calculate while
# ignoring missing values
fun_v   <-   function(x){c(m=mean(x,   na.rm   =   TRUE),
l=length(x), md=median(x,  na.rm = TRUE), s= sum(x, na.rm
= TRUE ))}

# Now create new dataframe that is
# collapsed version
quiz_clp <- summaryBy(quiz+ hours+ homework ~ studentid,
data=quizdata, FUN = fun_v)
```

Another way to do it is with the aggregate command that is built-in to R.

```
quiz_clp<-aggregate(quizdata, by=list(studentid),
  FUN=mean, na.rm=TRUE)
```

Looping

```
for(i in 1:5) {
      assign(paste0('ts',i), runif(20)+i)
}
crimtype<-c("violent", “murder", "rape",
"aggravated", "robbery", "property",
"burglary","larceny", "autotheft")
for(x in crimtype) {
subcrime <- x
## subset for kind  of crime
crimes_paste(subcrime)<-subset(allcrimes,
CRIM_CODE==paste(subcrime))
}
```

Appendix. Data sets used in the tutorials

The California Test Score Data Set

The California Standardized Testing and Reporting (STAR) dataset contains data on test performance, school characteristics and student demographic backgrounds. The data used here are from all 420 K-6 and K-8 districts in California with data available for 1998 and 1999. Test scores are the average of the reading and math scores on the Stanford 9 standardized test administered to 5th grade students. School characteristics (averaged across the district) include enrollment, number of teachers (measured as "full-time-equivalents"), number of computers per classroom, and expenditures per student. The student-teacher ratio used here is the number of full-time equivalent teachers in the district, divided by the number of students. Demographic variables for the students also are averaged across the district. The demographic variables include the percentage of students in the public assistance program CalWorks (formerly AFDC), the percentage of students that qualify for a reduced price lunch, and the percentage of students that are English Learners (that is, students for whom English is a second language). All of these data were obtained from the California Department of Education.

Downloaded from Stock and Watson website.

File: **caschool.csv**

Variable	Definition
dist_cod	district code
county	county
district	district name
gr_span	grade span of district
enrl_tot	total enrollment
teachers	number of teachers
calw_pct	Percent qualifying for CalWorks
meal_pct	Percent qualifying for reduced-price lunch
computer	number of computers
testscr	avg test score (= (read_scr+math_scr)/2)
comp_stu	computers per student (= computer/enrl_tot)
expn_stu	expenditures per student ($'s)
str	student teacher ratio (enrl_tot/TEACHERS)
avginc	district average income (in $1000's)
el_pct	percent of English Learners
read_scr	avg Reading Score
math_scr	avg Math Score

Current Population Survey Data

Each month the Bureau of Labor Statistics in the U.S. Department of Labor conducts the "Current Population Survey" (CPS), which provides data on labor force characteristics of the population, including the level of employment, unemployment, and earnings. Approximately 65,000 randomly selected U.S. households are surveyed each month. The sample is chosen by randomly selecting addresses from a database comprised of addresses from the most recent decennial census augmented with data on new housing units constructed after the last census. The exact random sampling scheme is rather complicated (first small geographical areas are randomly selected, then housing units within these areas randomly selected); details can be found in the Handbook of Labor Statistics and is described on the Bureau of Labor Statistics website.

The survey conducted each March is more detailed than in other months and asks questions about earnings during the previous year. The file CPS92_08 contains the data for 1992 and 2008 (from the March 1993 and 2009 surveys). These data are for full-time workers, defined as workers employed more than 35 hours per week for at least 48 weeks in the previous year. Data are provided for workers whose highest educational achievement is (1) a high school diploma, and (2) a bachelor's degree. The CPS Data for Table 8.1 in Stock and Watson are from the March 2009 survey. The files may be downloaded from Stock and Watson site.

Files: **cps92_08.csv, ch8_cps.csv (not all variables in each file)**

Variable	*Definition*
female	1 if female; 0 if male
year	Year
ahe	Average Hourly Earnings
bachelor	1 if worker has a bachelor's degree; 0 if worker has a high school degree
yrseduc	Years of Education
northeast	1 if from the Northeast, 0 otherwise
midwest	1 if from the Midwest, 0 otherwise
south	1 if from the South, 0 otherwise
west	1 if from the West, 0 otherwise

Acknowledgments

The authors are grateful to the many students who, over the years, let us try out different possibilities for using R in the classroom. Research assistance from Bobby Fatemi and Derran Cheng was much appreciated!