Department of Economics
Yale University
Prof. Edward Vytlacil

Getting Started

1 Starting a Project

Whenever you start a new project in R, it is recommended that you open a new script file, name it, and write a brief preamble that describes what your code is doing. This is not only helpful for others reading your code, but also allows you to organize your work and thoughts and helps you remember the state of your work when you are coming back to a project that you have not worked on for a while. For example, a preamble could look like below:

```
#------------------------------------------------------------------------#
# Project: Data Task                                                     #
# Date:    03/08/16                                                      #
# File:    Task.R                                                        #
# Author:  md                                                            #
#                                                                        #
# Tasks: a) Graph log mortality rates                                    #
#        b) Fit linear models to the data by income group                #
#        c) Plot fitted lines for the age range 35-76                    #
#------------------------------------------------------------------------#
```

In a second step, you should clear the workspace using the remove command `rm()` applied to the list of user-defined objects `ls()`, and set up your working directory. For example, if your project directory is a folder named "Task" on the desktop of a windows machine or a mac, you probably want to use one of the two `setwd()` lines below in your script:

```
> rm(list = ls())                    # clear workspace of user-defined objects
> setwd("C:\WINDOWS\Desktop\Task")  # set working directory (Windows)
> setwd("/Users/name/Desktop/Task") # set working directory (Mac)
```

Note that windows uses "\" to descibe the folder structure in file paths, while on macs file paths use "/". In order to get the file path of a document on a mac, navigate to the file or folder you wish to copy the path for. Right-click (or Control+Click, or a Two-Finger click on trackpads) on the file or folder in the Mac Finder. While in the right-click menu, hold down the OPTION (alt) key to reveal the "Copy (item name) as Pathname" option, it replaces the standard Copy option. On a windows device, hold down the Shift key, right-click a folder on the right side of the window, and choose Copy

as Path. That puts the full pathname for the folder you right-clicked in the Windows Clipboard.

Whenever you are telling R to load a file (e.g. a dataset) without specifing the whole file path, it will assume that the file is located in the working directory. You can view the current working directory by using the `getwd()` command.

As you can see from the example, it is possible to write comments in R scripts using the `#` sign. R understands that, in a given line of code, everything that follows the `#` sign is a comment. Unfortunately, there is no elegant way of writing multi-line comments in R. You will have to either start every comment line with a `#` or enclose the whole comment with an `if (FALSE) { comment }` block. Similar to a good preamble, thoughtfully written comments make it much easier to understand code and allow you to effectively collaborate. While most of the code that you are required to produce will not be too difficult to understand, we encourage you to start carefully commenting your code as early as possible. Once you have set up your script file, the next step is to import your data into R.

2 Importing Data into R

Importing data into R is fairly simple. Besides the many data sets that are available directly through R packages, you can import data into R from most commonly used data formats. All import commands create a dataframe. Since dataframe objects require unique column names, it is for some formats required to specify where these names should be imported from.

2.1 From a R package

Many datasets are available through R packages. For example, the `Ecdat` package, which we will use for illustration, contains more than a hundred data sets for econometric applications. One of these data sets is `Caschool`, the California Test Score Data Set, a cross section of 420 californian schools that contains information on equipment, teachers and students. To be able to access this data set we simply have to install the `Ecdat` package and load it:

```
> install.packages("Ecdat",dep=TRUE)
> library(Ecdat)
```

You can use the `data()` command to see the set of datasets that are available through the packages that you have currently loaded. Once you have decided which data set to use, you can either create a dataframe manually using the assignment operator, or by using the `data()` command applied to the data set. For example the following two lines of code produce the same dataframe object, once called `Caschool` and once `df.Cs`.

```
> data(Caschool)
> df.Cs <- Caschool
```

2.2 From Stata

The `library` and `readstata` packages allow you to directly import data sets from Stata using the `read.dta()` commands. Since Stata data sets by default save data using unique variable names, no additional information is required. For Stata 13 or 14 .dta files type

```
> install.packages("readstata13",dep=TRUE)
> data <- read.dta13("C:/WINDOWS/Desktop/Task/data.dta")
```

For Stata 12 or earlier versions:

```
> library(foreign)
> data <- read.dta("C:/WINDOWS/Desktop/Task/data.dta")
```

Note that you dont have to install the `foreign` package because it is part of the default R library.

2.3 From a Comma Delimited Text File

If you have a .csv file that uses comma as a seperator and has variable names in the first row, the data can be imported into R using the `read.table()` command.

```
> data <- read.table("C:/WINDOWS/Desktop/Task/data.csv", header=TRUE, sep=",")
```

Note that the command requires you to specify the seperator and whether the .csv file contains variable names in the first row. If the `header` parameter is not specified, R will automatically detect variable names only if the first row has one fewer entry than the number of columns.

2.4 From Excel

If your data is contained in an excel file with variable names in the first row, and you do not want to export it to a comma delimited file first, you can use the `xlsx` package.

```
> install.packages("xlsx",dep=TRUE)
> library(xlsx)
> data <- read.xlsx("C:/WINDOWS/Desktop/Task/data.xlsx", 1)
```

The second argument, `1`, tells R to the number of the worksheet (in this example it is the first worksheet) that should be imported from the .xlsx document.

2.5 From other Formats

If your data is in another format, such as systat, SPSS or SAS, there are also data import packages available. A simple web search will tell you what package to load and the syntax of the import command.

3 Describing Data with R

After setting up your work environment and impoting your data into R, the first step in every empirical project (assuming you already know your research question) is to make efforts to understand your data. This is essential as without a throughout understanding of your data, it is not possible to fully understand the results that you achieve, irrespective of the methods that you are using. For example, you need to understand what your variables measure, how these measurements are coded, what unit of observation they refer to, and what population the data was sampled from.

Often times most of these questions are answered in code books that are provided with the data. However, even then taking the time to study your data is highly recommended, as it improves your intuition for the data and also helps in less obvious ways. For example, you may notice coding errors that would go undetected otherwise, or detect that certain entries are missing systematically. Moreover, looking at the empirical distributions of your variables can give you ideas on how to scale your data in helpful ways, or guide you towards certain methods in describing and analyzing them.

3.1 Loading your data

As a first step, you should always look at the raw data. This ist done easily if your data is available as a dataframe. To demonstrate how you can use R to learn about new data, we will look at the California Test Score Data Set, `Caschool`.

```
> library(Ecdat)
> data(Caschool)
```

3.2 Looking at your data

There are different ways to look at raw data in R. In principle, for small dataframes, one option is to simply print the dataframe by typing

```
> Caschool
```

However, this is not recommended for regular sized or large data sets, as the command trys to print out the whole data frame at once, which is typically not helpful. The recommended alternative is to make use of R's integrated data viewer by typing

```
> View(Caschool)
```

While the default R viewer is not very beautiful, the RStudio viewer looks quite decent. If you prefer to work with the command line interface, you can use the `head()` command to look at a small subset (the first n rows) of your data set. For example,

```
> head(Caschool,n=3)
```

tells R to display the first three rows of the Caschool dataframe. This is often helpful to get a first impression of the data. A slightly modified version of the output the command produces is displayed below:

```
  distcod  county ... enrltot teachers calwpct mealpct  testscr       str   avginc...
1   75119 Alameda ...     195    10.90  0.5102  2.0408    690.8  17.88991 22.690...
2   61499   Butte ...     240    11.15 15.4167 47.9167    661.2  21.52466  9.824...
3   61549   Butte ...    1550    82.90 55.0323 76.3226    643.6  18.69723  8.978...
```

While looking at the raw data you should consult the code book to understand what each of the variables mean and how they are coded. For datasets that were loaded from a R package it is often the case that the codebook is available through the documentation function. For example,

```
> ?Caschool
```

gives you access to the codebook for the California Test Score Data Set.

Just like in the matrix object case, it is also possible to access subsets of a dataframe using square brackets. For example,

```
> Caschool[1:3,1]
[1] 75119 61499 61549
```

Gives us the first three elements of the first column of the data. Analogously, we can access components of individual variables/vectors of the dataframe using square brackets and the `$` symbol. For example,

```
> Caschool$testscr[3]
[1] 643.6
```

outputs the third element of `testscr` variable/third row element of the `testscr` column in the dataframe. Most importantly, we can use logical expressions to access subsets of a dataframe. For example,

```
> Caschool[Caschool$testscr>705,]
    distcod   county ... enrltot teachers calwpct mealpct  ...  testscr
417   69518 Santa Clara ... 3724   208.48  1.0741  1.5038  ...  706.75
```

tells us that there is only one school with average test scores above 705, and that the school is listed in row 417 of the dataframe. Take a second to understand the command above: It asks R to output all rows of the dataframe `Caschool` that satisfy our logical expression `Caschool$testscr>705`.

Moreover, note that we are not restricted to a single logical expression. In fact, we can combine as many logical expressions as we want using the `&` *and* and `|` *or* operators. For example,

```
> Caschool[Caschool$testscr>690 & Caschool$testscr<700,1]
```

asks R to give us all district codes of schools with average test scores in the open interval $(690, 700)$.

3.3 Data Types

In a next step, it is important to learn the data types that are contained in the individual columns of your dataframe. This can be done combining the `sapply()` and `class()` functions. The `sapply()` command is extremely useful in the context of dataframes, as it allows us to apply functions to each column vector of our data individually. For example,

```
> sapply(Caschool,class)
 distcod    county  district    grspan   enrltot  teachers   calwpct   mealpct
"integer"  "factor"  "factor"  "factor" "integer" "numeric" "numeric" "numeric"
 computer   testscr   compstu   expnstu       str    avginc     elpct   readscr
"integer" "numeric"  "numeric" "numeric" "numeric" "numeric" "numeric" "numeric"
 mathscr
"numeric"
```

tells us all the data types of each data vector of the `Caschool` data. As you can see, most variables are numeric (integer is a subclass of numeric). At the same time, the variables `county, district and grspan` are categorical variables and therefore stored as factors.

3.4 Dimensions

In a final step before starting to compute descriptive statistics, it is useful to know the dimensions of your data. This can be done using the `dim()` command that you already know from the matrix context. For example,

```
> dim(Caschool)
[1] 420  17
```

tells us that the `Caschool` dataframe has 420 rows (observations) and 17 columns (variables).

3.5 Summary Statistics

The `summary()` command applied to a dataframe provides descriptive statistics of all variables included in that dataframe. For numerical variables it computes the minimum and maximum, the 25% percentile, the median, the 75 % percentile and the mean. For categorical variables it computes the counts in each category. For example,

```
> summary(Caschool)
    distcod              county      ....       mathscr
 Min.   :61382   Sonoma     : 29             Min.   :605.4
 1st Qu.:64308   Kern       : 27             1st Qu.:639.4
 Median :67760   Los Angeles: 27             Median :652.5
 Mean   :67473   Tulare     : 24             Mean   :653.3
 3rd Qu.:70419   San Diego  : 21             3rd Qu.:665.9
 Max.   :75440   Santa Clara: 20             Max.   :709.5
                 (Other)    :272
```

R has many statistical functions. For example, the functions `mean()`, `median()` `min()`, `max()`, `sd()`, `var()`, `cov()` and `cor()` operate as expected. For example,

```
> cor(Caschool$testscr,Caschool$avginc)
[1] 0.7124308
> cor(Caschool$testscr,Caschool$str)
[1] -0.2263628
```

computes the correlation between average test scores and average income in a school district as well as the correlation between testscores and the student teacher ratio.

A useful command in the context of descriptive statistics is the `sapply()` command. As we have seen in section 3.3 it allows us to apply functions to each column of our data set. For example,

```
> round(sapply(Caschool,mean),3)
  distcod    county  district    grspan   enrltot  teachers   calwpct    ...
67472.810        NA        NA        NA  2628.793   129.067    13.246  ...
  compstu   expnstu       str    avginc     elpct   readscr   mathscr
    0.136  5312.408    19.640    15.317    15.768   654.970   653.343
```

computes the mean of each variable (Can you explain why R outputs missing values for `county`, `disctrict` and `grspan`?). The round also works as expected. The argument `3` tells R to output three decimals.

As an alternative to the `summary()` command, you may also wish to use the `describe()` function to provide an alternative summary of the data. `describe()` is part of the `Hmisc` package.

It is sometimes desired to run functions only on columns of a particular class, for example, on numeric variables. We can use the `sapply()` and `is.numeric` commands for this purpose. For example,

```
> Caschool.numeric <- Caschool[,sapply(Caschool, is.numeric)]
> round(sapply(Caschool.numeric,mean),3)
  distcod   enrltot  teachers   calwpct   mealpct  computer   testscr ...
67472.810  2628.793   129.067    13.246    44.705   303.383   654.157 ...
  compstu   expnstu       str    avginc     elpct   readscr   mathscr
    0.136  5312.408    19.640    15.317    15.768   654.970   653.343
```

Alternatively, we can run functions only on the columns of a particular class without creating a new data.frame. For example:

```
> round(sapply(Caschool[,sapply(Caschool, is.numeric)],mean),3)}
  distcod   enrltot  teachers   calwpct   mealpct  computer   testscr ...
67472.810  2628.793   129.067    13.246    44.705   303.383   654.157 ...
  compstu   expnstu       str    avginc     elpct   readscr   mathscr
    0.136  5312.408    19.640    15.317    15.768   654.970   653.343
```

Likewise, we may wish to run the command `var(Caschool[,sapply(Caschool, is.numeric)])` instead of `var(Caschool)`.

3.6 Dummy Variables

We can use logical expressions to create dummy variables. For example,

```
> Caschool$abovemedian <- Caschool$testscr > median(Caschool$testscr)
```

asks R to create a new variable `Caschool$abovemedian` that equals one if the school has a test score above the median test score, and equals zero otherwise. The mean of a dummy variable is the fraction of observations for which the variable equals one. Thus, taking the mean of the appropriate dummy variable is one way to calculate the fraction of observations (rows) satisfying the logical expression. In this example,

```
> mean(Caschool$abovemedian)
[1] 0.5
```

so that half of the schools have above the median testscore, as must be the case by definition of median. We could also compute the fraction of observations (rows) satisfying the logical expression without creating a new variable by taking the mean of the logical expression directly, for example:

```
> mean(Caschool$testscr > median(Caschool$testscr))
[1] 0.5
```