POL 345: Quantitative Analysis and Politics

Precept Handout 1

Week 2 (Verzani Chapter 1: Sections 1.2.4–1.4.31)

Remember to complete the entire handout and submit the precept questions to the Blackboard DropBox 24 hours before precept. In this handout, we cover the following new materials:

- Using `c()` to create and combine vectors
- Vector operations
- Using `[]` to access elements of vectors and subset data
- Basic functions: `length()`, `min()`, `max()`, `range()`, `mean()`, `median()`, and `sum()`
- Summarizing objects with `summary()`
- Using `seq()` and `:` to generate sequences
- Using `names()` to assign or access names of objects
- Obtaining help for **R** with `help()`, `?`, and `??`
- Changing the working directory with `setwd()`
- Displaying the working directory with `getwd()`
- Loading data with `read.csv()`, `read.table()`, and `load()`
- Accessing variables from data frames and creating new variables with `$`
- Obtaining the dimension of data with `dim()`, `nrow()`, and `ncol()`
- Listing objects in the workspace with `ls()`
- Saving objects with `save()` and the workspace with `save.image()`

1 Numeric Vectors

We begin by the simplest and yet inefficient way of entering data into **R**. The following table contains voter turnout data for New Jersey and the United States for the past 5 presidential elections. The data are taken from `http://elections.gmu.edu/voter_turnout.htm`.

Year	State	Voting Eligible Population	Voter Turnout
2008	United States	212,702,354	131,304,731
2008	New Jersey	5,848,620	3,868,237
2004	United States	203,483,455	122,294,978
2004	New Jersey	5,663,201	3,611,691
2000	United States	194,331,436	105,375,486
2000	New Jersey	5,601,788	3,187,226
1996	United States	186,347,044	96,262,935
1996	New Jersey	5,456,334	3,075,860
1992	United States	179,655,523	104,405,155
1992	New Jersey	5,398,559	3,343,594

- To store data, we use a **vector**, which is simply a string of information stored in a specific order. Use `c()` to enter a data vector with multiple entries.

```
> ## remember to use informative labels
> us.vep <- c(212702354, 203483455, 194331436, 186347044, 179655523)
> us.vto <- c(131304731, 122294978, 105375486, 96262935, 104405155)
> nj.vep <- c(5848620, 5663201, 5601788, 5456334, 5398559)
> nj.vto <- c(3868237, 3611691, 3187226, 3075860, 3343594)
> ## check to see if correctly entered
> us.vep

[1] 212702354 203483455 194331436 186347044 179655523
```

- Simple arithmetic operations can be done easily with numeric vectors where an operation will be repeated for each element of the vector.

```
> us.vep.thousands <- us.vep / 1000
> us.vep.thousands

[1] 212702 203483 194331 186347 179656
```

- Arithmetic operations can be done using multiple vectors too.

```
> ## turnout rate for US and NJ
> us.veptr <- us.vto / us.vep
> nj.veptr <- nj.vto / nj.vep
> us.veptr

[1] 0.617317 0.601007 0.542246 0.516579 0.581141

> nj.veptr
```

```
[1] 0.661393 0.637747 0.568966 0.563723 0.619349

> nj.veptr - us.veptr # difference between the two

[1] 0.0440763 0.0367403 0.0267196 0.0471440 0.0382085
```

- To access specific elements of a vector, use square brackets `[ ]` may be used to access specific elements of a vector. You can also change the value of a specific element with the usual assignment operator `<-`.

```
> nj.veptr[5] # NJ's VEP turnout rate from the 2008 election

[1] 0.619349

> nj.veptr[5] <- 0.619 # Round it to 3 decimal places
> nj.veptr

[1] 0.661393 0.637747 0.568966 0.563723 0.619000
```

- To combine multiple vectors into a single vector, use `c()`.

```
> ## additional data for 1984 and 1988 elections
> nj.veptrold <- c(0.5702941, 0.6050096)
> nj.veptr <- c(nj.veptr, nj.veptrold) # combine vectors and replace the original
> nj.veptr

[1] 0.661393 0.637747 0.568966 0.563723 0.619000 0.570294 0.605010
```

2 Functions

Functions are important objects in **R** and perform a wide range of tasks. A function takes often multiple input objects and returns an output object. In **R**, the general use of functions is done by `foo(bar)` where `foo` is the function name and `bar` is the name of the input object. When multiple inputs are given, the syntax becomes for example `foo(bar1, bar2)`. The order of inputs matter. That is, `foo(bar1, bar2)` is in general different from `foo(bar2, bar1)`. To avoid this problem, every input of a function comes with a specific name so that we can do `foo(input1 = bar1, input2 = bar2)` gives the same result as `foo(input2 = bar2, input1 = bar1)` where `input1` and `input2` are the input names in this example.

RStudio: Find Function Names

Use **RStudio** to look up the names of functions. If you can remember just the first letter, type the letter in the **RStudio** text editor (top-left screen) and press tab. **RStudio** should automatically give you a list of functions that start with that letter. To get more specific, type two or more letters.

- Some basic functions useful for summarizing data include `length()`, `min()`, `max()`, `range()`, `median()`, `mean()`, and `sum()`.

```
> length(nj.veptr)  # length of a vector

[1] 7

> min(nj.veptr)     # minimum value

[1] 0.563723

> max(nj.veptr)     # maximum value

[1] 0.661393

> range(nj.veptr)   # range

[1] 0.563723 0.661393

> median(nj.veptr)  # median value

[1] 0.60501

> mean(nj.veptr)    # mean value

[1] 0.603733

> sum(nj.veptr) / length(nj.veptr) # alternative way of calculating mean

[1] 0.603733
```

- The `summary()` function returns the basic summary of any object including a numeric vector.

```
> summary(nj.veptr)

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  0.564   0.570   0.605   0.604   0.628   0.661
```

- The `seq()` function may be used to generate a increasing or decreasing sequence. The first argument `from` specifies the number to start from; the second argument `to` specifies the number at which to end the sequence; the last argument `by` indicates the interval to increase or decrease by.

```
> ## sequence of presidential election years from 1984 to 2008
> seq(from = 1984, to = 2008, by = 4)   # increasing sequence

[1] 1984 1988 1992 1996 2000 2004 2008

> seq(from = 2008, to = 1984, by = - 4) # decreasing sequence
```

```
[1] 2008 2004 2000 1996 1992 1988 1984

> seq(2008, 1984, -4)                    # does the same thing
> seq(1984, 2008, -4)                    # gives an error
> seq(to = 1984, from = 2008, by = -4)   # this works fine
```

- The colon operator `:` creates a simple numeric sequence, beginning with the first number specified and increasing or decreasing by one integer to the last number specified.

```
> 1998:2008

 [1] 1998 1999 2000 2001 2002 2003 2004 2005 2006 2007 2008

> 2008:1998

 [1] 2008 2007 2006 2005 2004 2003 2002 2001 2000 1999 1998
```

- The `names()` function may be used to access and assign names to elements of a vector.

```
> names(nj.veptr) # no names are assigned yet

NULL

> ## assign corresponding year
> names(nj.veptr) <- seq(from = 2008, to = 1984, by = -4)
> names(nj.veptr)

[1] "2008" "2004" "2000" "1996" "1992" "1988" "1984"

> nj.veptr        # each element is now labeled by year

    2008     2004     2000     1996     1992     1988     1984
0.661393 0.637747 0.568966 0.563723 0.619000 0.570294 0.605010
```

3 Help Files

Learning **R** can be a challenging process. In addition to the resources provided through this course (Office hours, Study halls, DiscussionBoard) and online resources at the Comprehensive R Archive Network at `http://cran.r-project.org`, **R** comes with help files about functions that can be easily accessed from the command line.

RStudio: Help with Functions

In **RStudio**, you can use the help window to look up details on any **R** function. Access the help window by going to the bottom-right screen and clicking on the `help` tab. Then type in the name of the function in the search bar on the top right of the help window. Again, if you forget the name of the function, you can start with the first letter and **RStudio** will let you scroll through the relevant function names.

- If you know the exact name of function you are looking for, then simply type `?foo` or `help("foo")` where `foo` is the name of the function.

```
> ## two ways to get the help file for the mean() function
> help("mean")
> ?mean
```

- If you do not know the name, then type `help.search("xxx")` or `??xxx`, which will generate a list of all functions matching the keyword `xxx`. You may access individual help files by double clicking on the desired function listed.

```
> ## two ways to get the list of help files that match with the keyword "mean"
> help.search("mean")
> ??mean
```

R Tip: Use Google to Find Functions

Another way to get help with functions is to use Google. The online community of **R** users has done a pretty good job of assisting beginners with R coding. For example, if you could not remember the exact function for taking the mean of your data, you could google "R take the mean", or something like that. In general, we recommend starting your search with the letter "R", followed by some terms relevant to the function you are trying to find.

4 Loading and Saving Data

4.1 The Working Directory

The working directory is the default directory where **R** loads data from and saves data to. Although **R** can access to any file on your computer if their full path is specified, setting the working directory allows you to avoid some typing. For example, you may work to create a folder called `Handout1` and then specify it as the working directory when working on this handout.

RStudio: Change the Working Directory

To change the working directory click on

`"Tools > Set Working Directory > Choose Directory..."`

and then pick the folder from where you want to work.

- There are two ways to change the working directory:

  1. From command line: use `setwd()` and specify the full path of the folder you wish to load files from or save files to.

```
> # Set the working directory to a folder in the H drive
> setwd("H:/POL345/Handout1")
```

2. From drop down menu: choose `Change dir...` under `Files` for Windows and `Change Working Directory` under `Misc` for Macs.

- To display the current working directory, use `getwd()`.

```
> # This will display the path of the currently set working directory
> getwd()
```

4.2 Loading Data

We next show how to load data into **R**. For the purposes of this course, we will work with three types of data files. Download the example data files from Blackboard.

> **RStudio: Loading Data**
>
> You can load .csv and .txt files by going to the drop down menu and clicking
>
> `"Workspace > Import Dataset > from text file"`

1. Comma separated value (or csv) files: `turnout.csv`
2. Space delimited files: `turnout.txt`
3. **R** data file: `turnout.RData`

Each file contains the identical data with the following variables:

Variable	Description
`year`	The presidential election year
`State`	State name
`VEP`	Voter Eligible Population
`VAP`	Voting Age Population
`total`	Total turnout

For each data file, a different command should be used to load data. The `.RData` file comes with an object name for data and thus you do not need to assign an object name. If the first row contains variable names, as is the case in the turnout data, then specify `header = TRUE`.

```
> ## load the turnout data as the object "turnout"
> turnout <- read.csv("turnout.csv", header = TRUE)   # csv file
> turnout <- read.table("turnout.txt", header = TRUE) # space delimited file
> load("turnout.RData")                               # .RData file
```

4.3 Summarizing Data

Here, we introduce several basic functions that can be used for summarizing data.

- The `names()` function returns variable names from the data. Additionally, as with vectors, this command may be used to assign variables names to the data.

```
> names(turnout)

[1] "year"  "State" "VAP"   "VEP"   "total"
```

- The `$` symbol is used to access an individual variable from within a data set using its name. It returns a vector containing the specified variable

```
> states <- turnout$State
```

- The `dim()` function gives the dimensions of the data, i.e., the number of rows (observations) and the number of columns (variables). It returns a numeric vector containing the number of rows followed by number of columns

```
> dim(turnout)

[1] 416   5
```

- The `nrow()` and `ncol()` functions give the number of rows and columns of the data.

```
> nrow(turnout)

[1] 416

> ncol(turnout)

[1] 5
```

- The `summary()` function will provide basic statistics for each variable in the data frame.

```
> summary(turnout)

      year             State          VAP                VEP
 Min.   :1980   Alabama   :  8   Min.   :2.77e+05   Min.   :2.70e+05
 1st Qu.:1987   Alaska    :  8   1st Qu.:1.00e+06   1st Qu.:9.54e+05
 Median :1994   Arizona   :  8   Median :2.68e+06   Median :2.56e+06
 Mean   :1994   Arkansas  :  8   Mean   :7.57e+06   Mean   :7.05e+06
 3rd Qu.:2001   California:  8   3rd Qu.:4.71e+06   3rd Qu.:4.47e+06
 Max.   :2008   Colorado  :  8   Max.   :2.31e+08   Max.   :2.13e+08
                (Other)   :368
     total
 Min.   :1.63e+05
 1st Qu.:4.58e+05
 Median :1.35e+06
```

```
 Mean   :3.20e+06
 3rd Qu.:2.72e+06
 Max.   :1.33e+08
 NA's   :1.06e+02
```

- As with vectors, square brackets may be used to call specific portions of the data frame. Using brackets with a comma `[rows,columns]` allows users to call specific rows and columns by either row/column numbers or row/column names. Use sequencing functions covered above, i.e., `:` and `c()`. Here are some examples (the output suppressed).

```
> turnout[1:10, ]            # extract first ten rows
> turnout[, c("year", "State")] # extract two variables
> turnout[1:10, c(1:3, 5)]  # extract first ten rows, first five and last columns
> turnout$State[1:10]        # extract first ten obs. for "State" variable
```

- In **R**, missing values are represented by `NA`. When applying functions to an object with missing values, they often automatically remove those values before performing operation.

```
> # Missing total turnout for Connecticut
> turnout[8, ]

  year       State     VAP     VEP total
8 2008 Connecticut 2696127 2455684    NA
```

5 Workspace

The workspace is the "sandbox" where **R** temporarily saves everything you've created or loaded during the current session. The function `ls()` displays the names of all objects currently stored in the workspace.

```
> ls()

 [1] "nj.vep"           "nj.veptr"         "nj.veptrold"
 [4] "nj.vto"           "states"           "turnout"
 [7] "us.vep"           "us.vep.thousands" "us.veptr"
[10] "us.vto"
```

RStudio: Saving and Loading the Workspace

The workspace can be saved by using the dropdown menu and clicking on

`"Workspace > Save Workspace As..."`

and then picking a location to save the file. To load the same workspace next time you start **RStudio**, the dropdown menu can be similarly used

`"Workspace > Load Workspace..."`.

6 Saving Objects

It is important to remember that objects created during an **R** session are only temporarily stored to the workspace. Unless you save objects from the workspace before exiting **R**, you will lose all objects and changes to data.

- To save any object, use `save(xxx, file = "yyy.RData")` function where `xxx` is the object name and `yyy.RData` is the file name. Multiple objects can be listed and they will be stored as a single `.RData` file. The extension `.RData` should always be used for the file name. Unless the full path is specified, objects will be saved to the working directory.

```
> ## specify the .RData file name
> save(turnout, file = "turnout.RData")
> ## save multiple objects in a folder other than the working directory
> save(turnout, nj.veptr, us.veptr, file = "H:/POL345/Handout1/turnout.RData")
```

- To save the entire workspace (all objects), use the `save.image()` function with a `.RData` file name.

```
> save.image("H:/Handout1.RData")
```

  When you quit **R**, you will be asked whether you would like to save the workspace. If you answer Yes, the workspace will be saved as an invisible file `.RData` in the working directory. The above function allows users to specify the file name.

- To access the objects saved in the `.RData` file, simple double click the file or use the `load()` function as above.

7 Precept Questions

In preparation for precept, please answer the following questions based on the `turnout2.csv` file (available at Blackboard) and submit your code following the instruction given in the syllabus.

1. Begin by downloading the `turnout2.csv` file from Blackboard.
2. Load the data into **R** from the working directory.
3. How many variables are in the data set? What additional variables are available, as compared to the `turnout.csv` file? What variables are missing, as compared to the `turnout.csv` file?
4. How many observations are in the data?
5. Obtain a summary of the VEP variable.
6. Calculate the VAP turnout rate for each year by dividing the total turnout by the VAP.
7. Calculate the VEP turnout rate for presidential election years only by dividing the total turnout by the VEP. Compare these with the corresponding VAP turnout rates. **Hint**: Try using a sequence command to subset the relevant data.