Lab 1 Getting Started with R

The goal of this lab is to become familiar with the R programing language. You will learn how to enter command line code, install packages, import and save data, and manipulate your data set. For those of you familiar with R, this will be a very basic review. For those of you new to R, this will be an introduction to the basics that we will build on throughout the course. We will be using *RStudio* to interface with R.

# Set up R session

## R project

Working through an R project provides a efficient framework for keeping files organized. I suggest you utilize this option for this course. Start by creating a folder called “Multivariate\_2018” somehwere on your computer. Create subfolders called “Code”, “Data”, and “Outputs”. Next, open *RStudio* and click on the *File* tab in the upper left corner of the *RStudio* screen and select “New Project”. Next, select the “New Directory” option and then select “New Project” again. Come up with a creative name for the directory and browse to find your “Multivariate\_2018” folder. Finally, select “Create Project”

## Console and scripts

Now that you have opened *RStudio*, you see a window that is labeled console. You can enter code directly into this console. Let’s try by making sure 1 + 1 = 2.

Simply enter:

1 + 1

and hit Enter.

The way I prefer to use *RStudio* is to create a script. Go to *File* on the tool bar and click on *New Script*. A blank box labeled *Untitled1* should appear. This is your new script and you can name/save it by clicking on “File” and “Save”. Save it to your subfolder named “Code”. To run a command from the script enter:

1 + 1

and hit *Ctrl + Enter* or select the *Run* tab with the upper right corner of the script.

## Getting help

If you know the name of what you are asking about, you can simply type:

?read.table

and the information sheet for reading in a table, which we will be doing soon, pops up.

If you are unsure of the precise name of the function you are looking for, but know the general subject (in this case data input). You can use the help.search function as such:

help.search("data input")

and click on read.table.

If you want to know what package something is in, use the find function. For example, to find the package with Analysis of Variance (ANOVA):

find("anova")

You should see that anova is in the *stats* package.

## Worked examples

You can also access worked examples of a function. For example, a worked example of a linear model (lm) can be found using:

example(lm)

## Demonstrations

Demonstrations are also available in R. Check out the graphics demo:

demo(graphics)

## Setting your working directory

Because you are working in a “R Project” your working directory is already set.

## Downloading packages

Next, you are going to install and load the packages you need to run your analyses. To install the packages, click on the packages tab on the tool bar and select the Set CRAN mirror. This should result in a list of CRAN mirrors (ftp and web servers) located around the world. Select the closest CRAN mirror to your location. Here in Gainesville, the closest CRAN mirror is in Tennessee (USA(TN)). Next, use the install function to install the package MVA:

install.packages("MVA")

Load the packages using *library*.

library(MVA)

To find out information about the package:

library(help = MVA)

# Data!

## Importing Data

For your next step, you will load your data. You will be loading data from two different places today. First, you will download a data set from the package *MVA* on pollution in the USA called USairpollution. Second, you will download a modified version of this data set from your working directory. You are going to give each data set a new name that you can call going forward.

The MVA US air pollution data set:

usAir <- USairpollution  
# To see the data frame, enter:  
usAir

## SO2 temp manu popul wind precip predays  
## Albany 46 47.6 44 116 8.8 33.36 135  
## Albuquerque 11 56.8 46 244 8.9 7.77 58  
## Atlanta 24 61.5 368 497 9.1 48.34 115  
## Baltimore 47 55.0 625 905 9.6 41.31 111  
## Buffalo 11 47.1 391 463 12.4 36.11 166  
## Charleston 31 55.2 35 71 6.5 40.75 148  
## Chicago 110 50.6 3344 3369 10.4 34.44 122  
## Cincinnati 23 54.0 462 453 7.1 39.04 132  
## Cleveland 65 49.7 1007 751 10.9 34.99 155  
## Columbus 26 51.5 266 540 8.6 37.01 134  
## Dallas 9 66.2 641 844 10.9 35.94 78  
## Denver 17 51.9 454 515 9.0 12.95 86  
## Des Moines 17 49.0 104 201 11.2 30.85 103  
## Detroit 35 49.9 1064 1513 10.1 30.96 129  
## Hartford 56 49.1 412 158 9.0 43.37 127  
## Houston 10 68.9 721 1233 10.8 48.19 103  
## Indianapolis 28 52.3 361 746 9.7 38.74 121  
## Jacksonville 14 68.4 136 529 8.8 54.47 116  
## Kansas City 14 54.5 381 507 10.0 37.00 99  
## Little Rock 13 61.0 91 132 8.2 48.52 100  
## Louisville 30 55.6 291 593 8.3 43.11 123  
## Memphis 10 61.6 337 624 9.2 49.10 105  
## Miami 10 75.5 207 335 9.0 59.80 128  
## Milwaukee 16 45.7 569 717 11.8 29.07 123  
## Minneapolis 29 43.5 699 744 10.6 25.94 137  
## Nashville 18 59.4 275 448 7.9 46.00 119  
## New Orleans 9 68.3 204 361 8.4 56.77 113  
## Norfolk 31 59.3 96 308 10.6 44.68 116  
## Omaha 14 51.5 181 347 10.9 30.18 98  
## Philadelphia 69 54.6 1692 1950 9.6 39.93 115  
## Phoenix 10 70.3 213 582 6.0 7.05 36  
## Pittsburgh 61 50.4 347 520 9.4 36.22 147  
## Providence 94 50.0 343 179 10.6 42.75 125  
## Richmond 26 57.8 197 299 7.6 42.59 115  
## Salt Lake City 28 51.0 137 176 8.7 15.17 89  
## San Francisco 12 56.7 453 716 8.7 20.66 67  
## Seattle 29 51.1 379 531 9.4 38.79 164  
## St. Louis 56 55.9 775 622 9.5 35.89 105  
## Washington 29 57.3 434 757 9.3 38.89 111  
## Wichita 8 56.6 125 277 12.7 30.58 82  
## Wilmington 36 54.0 80 80 9.0 40.25 114

The modified US air pollution data set from your working directory is a csv file:

usAir\_mod <- read.csv("../Data/lab\_1/usAir\_mod.csv", row = 1, header = TRUE)  
# header=TRUE is stating that the first row is column names. row=1 is  
# stating that the first column is not data (i.e. sample ids; e.g, city  
# names in this case). Use your own file path. you don't need the '..'  
# preceeding the first slash

The read.csv command imports your data into a data frame. Throughout this course you will be working with data frames. I like using csv files because you can read them in Excel and easily import them into R. More often than not, you will be working with imported data as opposed to data from R packages (e.g.,USairpollution) .

## Checking out your data

The dimensions of your data tell you the number of row, columns, and matrices. Let’s look at the dimensions of your data:

dim(usAir)

## [1] 41 7

# dim(usAir\_mod)

You see that usAir and usAir\_mod have 41 rows and 7 columns.

To further assess your data set, let’s look at the structure of your data:

str(usAir)

## 'data.frame': 41 obs. of 7 variables:  
## $ SO2 : int 46 11 24 47 11 31 110 23 65 26 ...  
## $ temp : num 47.6 56.8 61.5 55 47.1 55.2 50.6 54 49.7 51.5 ...  
## $ manu : int 44 46 368 625 391 35 3344 462 1007 266 ...  
## $ popul : int 116 244 497 905 463 71 3369 453 751 540 ...  
## $ wind : num 8.8 8.9 9.1 9.6 12.4 6.5 10.4 7.1 10.9 8.6 ...  
## $ precip : num 33.36 7.77 48.34 41.31 36.11 ...  
## $ predays: int 135 58 115 111 166 148 122 132 155 134 ...

# str(usAir\_mod)

You can see that your data is indeed a data frame and there are 41 observations and 7 variables. The structure command also shows the variable names and their format. Here, you have numerical and integer variables. **More on checking out your data in the next lab!**

## Manipulating data

At times, you may want to subset your data set by extracting columns, rows or individual values. The first thing that you should know and always remember when it comes to data frames or matrices is that an element in a matrix is identified by **[ROW, COLUMN]**.

### Extracting elements

To extract the VALUE in the first row and first column from “usAir”:

usAir[1, 1]

## [1] 46

\*Practice extracting various elements out of “usAir”. Check the value with the original data matrix to make sure you have your rows and column numbers correct.

### Extracting single rows and columns

You can extract an entire row or column by leaving the column or row position blank, respectively.

To extract the first row from “usAir”:

usAir[1, ]

To extract the first column from “usAir”:

usAir[, 1]

You can also extract columns with a $ and the column name:

usAir$SO2

\*Practice extracting various rows and columns out of “usAir”. Check the value with the original data matrix to make sure you have your row and column numbers correct.

### Dropping single rows and columns

You can also drop columns or rows from a data frame:

To drop the first row from “usAir”:

usAir[-1, ]

To drop the first column from “usAir”:

usAir[, -1]

### Extracting multiple rows and columns

You can also extract multiple rows or columns. To extract the first five rows from “usAir”:

usAir[1:5, ]

To extract the first five columns from “usAir”:

usAir[, 1:5]

### Dropping multiple rows and columns

Just like with single rows and columns, you can also drop multiple rows or columns. To drop the first five rows from “usAir”:

usAir[-(1:5), ]

To drop the first five columns from “usAir”:

usAir[, -(1:5)]

### Selecting columns and rows with logical operators

You may be interested in extracting rows or columns that meet a certain criteria. For example, in our “usAir” data we may only want sites with an average temperature of less than 50 degrees. To achieve this we use a logical operator:

usAir[usAir[, 2] < 50, ]

In the syntax of this line, we maintain the [row, column] designations. However in our row position we have a logical operator that says *if a value in column 2 (the temperature column) is less than 50, keep the entire row.*

### Transposing data

You can transpose a data frame:

t(usAir)

but to transpose a row or column, you must first convert it from a data frame into a matrix:

matrix\_usAir <- as.matrix(usAir$SO2)

then transpose it:

t(matrix\_usAir)

### Sorting data

Let’s take a look at three functions that can reorder data; **rank, sort,** and **order**. For this example, we will use the temperature column of usAir:

temp <- usAir$temp  
  
# Let’s apply each of these functions:  
  
ranks <- rank(temp)  
sorted <- sort(temp)  
ordered <- order(temp)  
  
# and combine them with the original temp values in a data table:  
  
table <- data.frame(temp, ranks, sorted, ordered)

The temp are in their original order. The **ranks** column contains the value that is the rank of the particular data point (i.e., value of temp), where 1 is assigned to the lowest temp and 41 is assigned to the highest. The **sorted** column simply sorts the temp values in ascending order. To sort in descending order, use:

des\_sort <- rev(sort(temp))

Order may be the most useful for sorting multivariate data. The **ordered** column contains an integer vector that will sort temp into ascending order. So the first value in the ordered column is 25, and we can go to the temp column row 25 and see that this observation has the lowest temperature value. Conversely, the last value in the ordered column is 23, and you can go to the temp column row 23 and see that this observation has the highest temperature value.

Let’s take order a step further and sort the entire data frame:

temp\_ordered <- usAir[order(temp), ]

You can see here we are using the **[row, column]** format again. We are ordering by the row values in the column temp, and coupling the values form the other columns with their temperature values, by leaving the column place blank. Order does not allow for values from the same observation (in this case city) to become decoupled.

## Mathematical operations on:

### Rows and Columns

To calculate **mean, median, variance,** or **sum** of a specific column, in this case column 3:

mean(usAir[, 3])

## [1] 463.0976

median(usAir[, 3])

## [1] 347

var(usAir[, 3])

## [1] 317502.9

sum(usAir[, 3])

## [1] 18987

To calculate **mean, median, variance,** or **sum** of a specific row, in this case row 3, we need to transpose the row first:

mean(t(usAir[3, ]))

## [1] 160.42

median(t(usAir[3, ]))

## [1] 61.5

var(t(usAir[3, ]))

## Atlanta  
## Atlanta 37048.96

sum(t(usAir[3, ]))

## [1] 1122.94

### Data frames and matrices

You can also calculate sums and means for all the rows or columns at once:

columnSum <- colSums(usAir)  
rowSums(usAir)

## Albany Albuquerque Atlanta Baltimore Buffalo   
## 430.76 432.47 1122.94 1793.91 1126.61   
## Charleston Chicago Cincinnati Cleveland Columbus   
## 387.45 7040.44 1170.14 2073.59 1063.11   
## Dallas Denver Des Moines Detroit Hartford   
## 1685.04 1145.85 516.05 2831.96 854.47   
## Houston Indianapolis Jacksonville Kansas City Little Rock   
## 2194.89 1356.74 926.67 1102.50 453.72   
## Louisville Memphis Miami Milwaukee Minneapolis   
## 1144.01 1195.90 824.30 1511.57 1689.04   
## Nashville New Orleans Norfolk Omaha Philadelphia   
## 973.30 820.47 665.58 732.58 3930.13   
## Phoenix Pittsburgh Providence Richmond Salt Lake City   
## 924.35 1171.02 844.35 744.99 504.87   
## San Francisco Seattle St. Louis Washington Wichita   
## 1334.06 1202.29 1659.29 1436.49 591.88   
## Wilmington   
## 413.25

colMeans(usAir)

## SO2 temp manu popul wind precip   
## 30.048780 55.763415 463.097561 608.609756 9.443902 36.769024   
## predays   
## 113.902439

rowMeans(usAir)

## Albany Albuquerque Atlanta Baltimore Buffalo   
## 61.53714 61.78143 160.42000 256.27286 160.94429   
## Charleston Chicago Cincinnati Cleveland Columbus   
## 55.35000 1005.77714 167.16286 296.22714 151.87286   
## Dallas Denver Des Moines Detroit Hartford   
## 240.72000 163.69286 73.72143 404.56571 122.06714   
## Houston Indianapolis Jacksonville Kansas City Little Rock   
## 313.55571 193.82000 132.38143 157.50000 64.81714   
## Louisville Memphis Miami Milwaukee Minneapolis   
## 163.43000 170.84286 117.75714 215.93857 241.29143   
## Nashville New Orleans Norfolk Omaha Philadelphia   
## 139.04286 117.21000 95.08286 104.65429 561.44714   
## Phoenix Pittsburgh Providence Richmond Salt Lake City   
## 132.05000 167.28857 120.62143 106.42714 72.12429   
## San Francisco Seattle St. Louis Washington Wichita   
## 190.58000 171.75571 237.04143 205.21286 84.55429   
## Wilmington   
## 59.03571

## Exporting Data

To export data to a readable Excel file or CSV for the “columnSum” result above:

write.csv(columnSum, "Output/filename.csv")

This will save to the working directory *Output* file you created earlier.

\*NOW, RUN THROUGH THE ABOVE EXERCISES WITH YOUR OWN DATA (or use the pitcher plant data set provided)!