|  |  |  |  |
| --- | --- | --- | --- |
| **PM592: Regression Analysis for Health Data Science** |  |  |  |
| **Lab 1 – Introduction to R**  **Data Needed:** *chs\_individual.csv, okcprofiles\_cleaned.csv* | | | |

**Outline**

* Overview of R and R Studio
* Packages
* Basic programming in R
* Inputting data

1. **Installation**
   1. **Install R**Download R software for your device by visiting <https://www.r-project.org/>. R can be thought of as the “engine” that will run all the analyses that we will be performing. You’ll notice there are several versions of R available on the website, and each has its own “fun” name. Choose the most recent version to download.
   2. **Install R Studio**If R is the engine, RStudio is the dashboard. Although not required to run R, this program will provide you with an “environment” for interfacing with R. Follow the instructions at <https://rstudio.com/products/rstudio/download>
   3. **RStudio Layout**RStudio consists of several windows:

* The *command window* is where you can type simple commands after the “>” prompt. R will execute these commands immediately upon pressing “Enter.”
* The *script window* is where you can write and save scripts, or collections of commands, to be executed when you’d like. To execute a chunk of code in this window, you can highlight it and click “Run” or press CTRL+ENTER.
* The *workspace/history window* is where you can see which data and values R has in its memory. You can view and edit the values by clicking on them. The history window shows what has been entered previously.
* The *files/plots/packages/help window* is where you can open files, view current and previous plots, install and load packages, or use the help function.

1. **Packages**
   1. One of benefits of R is its flexibility. By taking advantage of user-written packages (aka “libraries”) the user can perform a breadth of analyses. Packaged once installed on your computer do not need to be installed again. However, packages do need to be loaded into your workspace before you can use them in any given R session.
   2. To install a package type: install.packages(“packagename”)
   3. To load a package that has been installed, type: library(packagename)
   4. We will be using the “Tidyverse” package in this class. Tidyverse changes how you interact with objects in your workspace to be more intuitive and acts like more traditional statistical software.

* Install Tidyverse and load it into your workspace

1. **Working in R**
   1. **Your Working Directory**. Your working directory is the folder on your computer in which you are currently working. When you ask R to open a file, it will look in your working directory for this file. When you have R save an object like a data set or figure, it will save it in the working directory.
   2. **Setting the Working Directory.** You can set your working directory by going to the “Files” pane and browsing to the folder that has your data file. Once you are in this folder, click on “More” and “Set As Working Directory”.
   3. **Reading Data in R from a File**
      1. We often can’t control the type of data set we will be given. It may be in a text format, an Excel file, a comma-separated values file, or a file from another statistical program like SAS. Tidyverse (or more correctly the readr package within Tidyverse) offers an extensive list of supported file types, and the functions used to read these files all begin with “read\_”.
         1. Text: read\_txt
         2. CSV: read\_csv
      2. The Haven package offers support for additional file types, such as SPSS, Stata, and SAS data sets.

* Use the “read\_csv” function to read-in the chs\_individual.csv file.
  + 1. Assigning objects such as the data sets you read in is accomplished by using the “<-“ operator. In this course we will mostly work with data sets. However there are several other objects such as vectors, matrices, and arrays.  
         
       You can see various data types in R here: <https://www.statmethods.net/input/datatypes.html>
* Store the data set in an object named “kids”  
  kids <- read\_csv("chs\_individual.csv")
  1. **Getting Help**
     1. Typing ? before the name of a function will bring up the help documentation for that function.
     2. Typing ?? before the name of a function will do a broader search for documentation, including on functions that may not be loaded into your workspace.
  2. **A First Look**
     1. To view the contents of an object directly, you can type the name of the object.

> kids

# A tibble: 1,200 x 23

sid townname male race hispanic agepft height weight bmi asthma

<dbl> <chr> <dbl> <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>

1 1 Lancast~ 1 W 0 10.2 123 54 16.2 0

2 2 Lancast~ 1 W 0 10.5 145 77 16.6 0

3 6 Lancast~ 0 B 0 10.1 145 143 30.9 0

4 7 Lancast~ 0 O 0 10.7 156 72 13.4 0

5 8 Lancast~ 0 W 1 9.78 132 61 15.9 0

6 10 Lancast~ 1 O 1 NA NA NA NA 0

7 13 Lancast~ 1 O 1 10.2 140 79 18.3 0

8 16 Lancast~ 0 W 0 10.4 141 74 16.9 1

9 19 Lancast~ 0 W 0 NA NA NA NA 0

10 21 Lancast~ 0 W 1 9.94 126 59 16.9 0

# ... with 1,190 more rows, and 13 more variables: active\_asthma <dbl>,

# father\_asthma <dbl>, mother\_asthma <dbl>, wheeze <dbl>,

# hayfever <dbl>, allergy <dbl>, educ\_parent <dbl>, smoke <dbl>,

# pets <dbl>, gasstove <dbl>, fev <dbl>, fvc <dbl>, mmef <dbl>

* + 1. The data set has 1,200 observations and 23 variables. We say its *dimension* is 1200 x 23 because it is 1200 entries long and 23 entries wide. We can also find the dimension manually:

> dim(kids)

[1] 1200 23

* + 1. **Variables**. Variables are measurements to be made on several subjects. In this dataset, subject, townname, age, sex, race, etc. are all variables.
    2. **Variable Names**. A variable name is a computerized name of a variable. It is good programming practice to abbreviate names when possible, be consistent about capitalization, avoid spaces in variable names, do not use symbols (other than “\_” and “.” in R), and do not begin with a symbol or number. We can find variable names manually:

> names(kids)

[1] "sid" "townname" "male" "race"

[5] "hispanic" "agepft" "height" "weight"

[9] "bmi" "asthma" "active\_asthma" "father\_asthma"

[13] "mother\_asthma" "wheeze" "hayfever" "allergy"

[17] "educ\_parent" "smoke" "pets" "gasstove"

[21] "fev" "fvc" "mmef"

* + 1. **Data Values**. A data value is the actual measurement of a variable on a single subject. E.g., subject 1’s data value for ht is 56.7.
    2. **Observations.** A set of data values for the same subject. E.g., the observation for subject 1 consists of all their data values.
    3. **Rows and Columns.** Because the dataset has a particular dimension to it, we can refer to rows and columns by their index [r, c], which refers to the rth row and the cth column.  
       The data value of the first row and column is given by:

> kids[1,1]

# A tibble: 1 x 1

subject

<dbl>

1 1

We can also extract the entire first row by leaving the column value blank:

kids[1,]

# A tibble: 1 x 23

sid townname male race hispanic agepft height weight bmi asthma

<dbl> <chr> <dbl> <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>

1 1 Lancast~ 1 W 0 10.2 123 54 16.2 0

# ... with 13 more variables: active\_asthma <dbl>, father\_asthma <dbl>,

# mother\_asthma <dbl>, wheeze <dbl>, hayfever <dbl>, allergy <dbl>,

# educ\_parent <dbl>, smoke <dbl>, pets <dbl>, gasstove <dbl>,

# fev <dbl>, fvc <dbl>, mmef <dbl>

Or likewise, extract the entire first column:

> kids[,1]

# A tibble: 1,200 x 1

sid

<dbl>

1 1

2 2

3 6

4 7

5 8

6 10

7 13

8 16

9 19

10 21

# ... with 1,190 more rows

* Obtain the data value of race for the 10th observation.
  1. **Missing Values**
     1. Missing values in similar programming languages follow these conventions:
        1. Missing numeric values are assigned a value of a dot “.”
        2. Missing character values are assigned a blank space “”
     2. In R, missing values are all assigned to <NA>  
        Let’s look at the first 10 observations:

> kids

# A tibble: 1,200 x 23

sid townname male race hispanic agepft height weight bmi asthma

<dbl> <chr> <dbl> <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>

1 1 Lancast~ 1 W 0 10.2 123 54 16.2 0

2 2 Lancast~ 1 W 0 10.5 145 77 16.6 0

3 6 Lancast~ 0 B 0 10.1 145 143 30.9 0

4 7 Lancast~ 0 O 0 10.7 156 72 13.4 0

5 8 Lancast~ 0 W 1 9.78 132 61 15.9 0

6 10 Lancast~ 1 O 1 NA NA NA NA 0

7 13 Lancast~ 1 O 1 10.2 140 79 18.3 0

8 16 Lancast~ 0 W 0 10.4 141 74 16.9 1

9 19 Lancast~ 0 W 0 NA NA NA NA 0

10 21 Lancast~ 0 W 1 9.94 126 59 16.9 0

# ... with 1,190 more rows, and 13 more variables: active\_asthma <dbl>,

# father\_asthma <dbl>, mother\_asthma <dbl>, wheeze <dbl>,

# hayfever <dbl>, allergy <dbl>, educ\_parent <dbl>, smoke <dbl>,

# pets <dbl>, gasstove <dbl>, fev <dbl>, fvc <dbl>, mmef <dbl>

Subject 10 and 19 are missing data on agepft, height, weight, and BMI.

* + 1. We can use the filter function to obtain records that are missing on a particular variable:

> kids %>%

+ filter(is.na(bmi))

# A tibble: 89 x 23

sid townname male race hispanic agepft height weight bmi asthma

<dbl> <chr> <dbl> <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>

1 10 Lancast~ 1 O 1 NA NA NA NA 0

2 19 Lancast~ 0 W 0 NA NA NA NA 0

3 25 Lancast~ 1 B 0 NA NA NA NA 1

4 29 Lancast~ 0 B 0 NA NA NA NA 1

5 37 Lancast~ 1 O 1 NA NA NA NA 0

6 69 Lancast~ 1 W 1 NA NA NA NA 0

7 89 Lancast~ 0 W 0 NA NA NA NA 0

8 111 Lancast~ 0 B 0 NA NA NA NA 0

9 126 Lancast~ 0 O 1 NA NA NA NA 0

10 142 Lancast~ 0 W 0 NA NA NA NA 1

# ... with 79 more rows, and 13 more variables: active\_asthma <dbl>,

# father\_asthma <dbl>, mother\_asthma <dbl>, wheeze <dbl>,

# hayfever <dbl>, allergy <dbl>, educ\_parent <dbl>, smoke <dbl>,

# pets <dbl>, gasstove <dbl>, fev <dbl>, fvc <dbl>, mmef <dbl>

* + 1. If we want just the subject ID values of those missing BMI:

> kids %>%

+ filter(is.na(bmi)) %>%

+ pull(sid)

[1] 10 19 25 29 37 69 89 111 126 142 149 150 241 242

[15] 431 456 496 500 509 581 588 594 622 623 633 638 656 694

[29] 714 723 730 747 752 804 870 936 1020 1024 1026 1031 1034 1067

[43] 1098 1121 1135 1154 1155 1199 1256 1262 1265 1310 1332 1336 1368 1373

[57] 1389 1399 1401 1416 1429 1491 1512 1550 1555 1558 1571 1628 1633 1641

[71] 1658 1702 1708 1724 1782 1806 1854 1862 1863 1868 1889 1903 1908 1915

[85] 1946 1983 2025 2036 2053

1. **Brief Introduction to Tidyverse**
   1. In Base R, the way to display all observations missing BMI values would be:

> kids[is.na(kids$bmi),]

# A tibble: 89 x 24

sid townname male race hispanic agepft height weight bmi asthma

<dbl> <chr> <dbl> <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>

1 10 Lancast~ 1 O 1 NA NA NA NA 0

2 19 Lancast~ 0 W 0 NA NA NA NA 0

3 25 Lancast~ 1 B 0 NA NA NA NA 1

4 29 Lancast~ 0 B 0 NA NA NA NA 1

5 37 Lancast~ 1 O 1 NA NA NA NA 0

6 69 Lancast~ 1 W 1 NA NA NA NA 0

7 89 Lancast~ 0 W 0 NA NA NA NA 0

8 111 Lancast~ 0 B 0 NA NA NA NA 0

9 126 Lancast~ 0 O 1 NA NA NA NA 0

10 142 Lancast~ 0 W 0 NA NA NA NA 1

# ... with 79 more rows, and 14 more variables: active\_asthma <dbl>,

# father\_asthma <dbl>, mother\_asthma <dbl>, wheeze <dbl>, hayfever <dbl>,

# allergy <dbl>, educ\_parent <dbl>, smoke <dbl>, pets <dbl>, gasstove <dbl>,

# fev <dbl>, fvc <dbl>, mmef <dbl>, height\_2 <dbl>

However, we will be using Tidyverse in this class. Tidyverse contains a suite of sub-packages such as dplyr, which makes data manipulation easy.

* 1. The pipe operator %>%
     1. The pipe operator takes the output of one function and feeds it into the first argument of the next function. While most functions use the form f(x), the piping operator will use the form x %>% f. Using the pipe, your code will be clearer and easier to read as you follow the cascade of commands.
     2. Suppose we want to count how many individuals in our dataset have an FEV value greater than 2000. The first method does not use piping, while the second method does. The second method is cleaner and easier to follow.

> count(filter(kids, fev>2000))

> kids %>%

filter(fev > 2000) %>%

count()

* 1. Dplyr Useful Functions
     + select() returns the data frame with only the specified columns.
     + mutate() can create or overwrite existing columns.
     + filter() returns the data frame with rows meeting the specified criteria.
     + arrange() sorts the data frame based on the provided columns.
     + summarize() reduces variables down to certain summary statistics.
     + group\_by() allows for operations based on strata of the specified variable (e.g., males, females).

1. **Manipulating Data**
   1. **Adding Variables**
      1. In Tidyverse we use the “mutate” function to add a variable or change an existing variable.
      2. In this example we create two variables: height-squared, and BMI (body mass index). First, though, we have to change height to inches (it is currently in cm). Then, we get summary statistics for each of these variables to make sure they were computed correctly.

> kids <-

+ kids %>%

+ mutate(height = height\*0.3937

height\_2 = height \* height,

+ bmi = 703\*weight/height \_2)

> summary(kids$height\_2)

Min. 1st Qu. Median Mean 3rd Qu. Max.

2014 2825 2995 3000 3170 4220

> summary(kids$bmi)

Min. 1st Qu. Median Mean 3rd Qu. Max.

11.27 15.75 17.44 18.46 20.30 41.18

**Common Operators for Mathematical Functions**

|  |  |  |  |
| --- | --- | --- | --- |
| **Symbol** | **Operation** | **Example** | **In R** |
| ^ or \*\* | Exponentiation | Y=X3 | Y = X^3 |
| \* | Multiplication | Y=B x C | Y = B\*C |
| / | Division | Y = X/Z | Y = X/Z |
| + | Addition | Y = X + Z | Y = X + Z |
| - | Subtraction | Y = X - Z | Y = X - Z |

For more information on operators in R:  
<https://www.statmethods.net/management/operators.html>

* 1. **Categorizing Continuous Variables**
     1. We can take advantage of different functions in dplyr to convert continuous variables to categorical.
     2. The if\_else function. This function takes in a condition and returns a specified value if the condition is true, and a different value if the condition is false. In this example we create a variable that is equal to “W” if race is white, or “NW” if race is not white. Then we display the race and white variables.

> kids <-

+ kids %>%

+ mutate(white = if\_else(race=="W", "W", "NW"))

> kids %>%

+ select(race, white)

# A tibble: 1,200 x 2

race white

<chr> <chr>

1 W W

2 W W

3 B NW

4 O NW

5 W W

6 O NW

7 O NW

8 W W

9 W W

10 W W

# ... with 1,190 more rows

* + 1. The case\_when function. If we wanted to have several contingencies, it may seem that we would need to nest several if\_else statements. This function goes through the equivalent of several if\_else statements. In this example we create a variable that categorizes BMI into low (<18.5), medium (18.5-25), and high (25+). Then we display the BMI and BMI category variables.

> kids <-

+ kids %>%

+ mutate(bmicat = case\_when(

+ bmi < 18.5 ~ "low",

+ bmi <= 25 ~ "medium",

+ bmi > 25 ~ "high",

+ TRUE ~ NA\_character\_

+ ))

> kids %>%

+ select(bmi, bmicat)

# A tibble: 1,200 x 2

bmi bmicat

<dbl> <chr>

1 16.2 low

2 16.6 low

3 30.9 high

4 13.4 low

5 15.9 low

6 NA NA

7 18.3 low

8 16.9 low

9 NA NA

10 16.9 low

# ... with 1,190 more rows

This function addresses arguments in the order received. The reason we did not need to specify 18.5 <= bmi <= 25 is because all values <18.5 had already been classified as “low”.

We include the value of TRUE to specify what happens if values are not assigned by the above rules. Here we are saying if (for some reason) a value of BMICAT hadn’t been assigned, assign it to a NA character value.

* 1. **Inputting Data Manually**
     1. Tidyverse stores data sets in an object called a Tibble. Tibble objects are essentially interchangeable with data frames.
     2. You can create your own tibble by specifying the values of each of the columns.

> contacts <-

+ tibble(

+ fname = c("Janifer", "Peter", "Wanda", "Mike", "Frank", "Sarah"),

+ lname = c("Rumbagai", "Taluda", "Pettigrew", "Melvae", "Rivolous", "Schombre"),

+ org = c("LACPWD", "Other", "Enrich LA", "LATA", "Other", "Other"),

+ age = c(32, 54, 49, 46, 50, 40),

+ area = c("Health", "Psychiatry", "Education", "Transportation", "Misc", "Nutrition")

+ )

> contacts

# A tibble: 6 x 5

fname lname org age area

<chr> <chr> <chr> <dbl> <chr>

1 Janifer Rumbagai LACPWD 32 Health

2 Peter Taluda Other 54 Psychiatry

3 Wanda Pettigrew Enrich LA 49 Education

4 Mike Melvae LATA 46 Transportation

5 Frank Rivolous Other 50 Misc

6 Sarah Schombre Other 40 Nutrition

* 1. **Writing A Statistical Summary**

Writing a statistical summary takes *practice*. In the summary, you want to be thorough about what you did but also as concise as possible. The following should be contained in the summary:

|  |  |
| --- | --- |
| Research Question | What question did you want to address? |
| Statistical Method | Which method did you use? |
| Findings | What did you find, with regard to your hypotheses? |
| Effects | What were the effect size, test statistics, and p-value? |
| Interpretation | Any additional interpretation that may be required to better understand the effect. |

Please see <https://jgscott.github.io/teaching/writeups/write_ups/> for additional examples.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Lab 1 Exercises** |  |  |  |  |  |  |

|  |  |
| --- | --- |
| Objective(s): | Practice using the R interface, creating and recoding variables, computing descriptive statistics |
| Datasets Required: | okcprofiles\_cleaned |

1. Create a folder on your computer to store the profiles data.
2. Open RStudio and create a new project. Navigate to the folder you created in (1) as the location for your project.
3. Use the read\_csv function to read-in the profiles data.
4. View part of the data by submitting the data set name into the command line.
   1. How many observations are there?
   2. How many variables are there?
   3. Use the names function to answer: what are the variable names?
5. Summarize the data using different methods:
   1. What is the output when you summarize using R’s base summary() command?
   2. What is the output when you use skimr’s skim() command? (You will have to install and load this package!)
   3. What is the output when you use Hmisc’s describe() command?
6. Create a new variable called drink\_yn which equals 1 if the individual drinks any amount, and equals 0 if the individual does not drink at all.
   1. How many people in the data set drink (any amount)?
7. Create a new variable called drugs\_yn which equals 1 if the individual does drugs in any amount, and equals 0 if the individual does not do drugs at all.
   1. How many people in the data set do drugs (any amount)?
8. Use the count function on drink\_yn and drugs\_yn.
   1. How many people in the data set do not do drugs and do not drink?
9. Create a new variable called height\_ft, which equals height/12.
   1. What is the mean of height\_ft?
10. What percent of profiles on OkCupid are from users who are currently seeing someone?
11. Use the following code to obtain the mean height of males and females.

profiles %>% group\_by(sex) %>% summarise(meanht = mean(height))

* 1. What is your output?

1. Use the following code to obtain the mean height of males and females.

profiles %>% group\_by(sex) %>% summarise(meanht = mean(height, na.rm=T))

* 1. What is your output?
  2. Why do you think the method used in this question worked while it did not work in Question 11?

1. There are several ways we could save the data set, but the RDS format retains R objects. We’ll be working on this data again, so let’s save our data as an RDS file.

write\_rds(profiles, "profiles\_new.rds")