

Learning Goals

1. Sign up for RStudio Cloud (<https://rstudio.cloud/>)
2. Download and Install R and RStudio.
3. Open and familiarize yourself with RStudio's windows and menus.
4. Learn to manage files on your computer(watch videos on course overview page on sakai).
5. Learn to control your working directory with the commands `getwd()` and `setwd()`.
6. Saving your code to a script file.
7. Learn syntax to perform standard algebraic calculations with *R*.
8. Naming variables and objects in *R* and removing names (avoid names such as `c`, `q`, `t`, `C`, `D`, `F`, `I`).
9. Create numerical and character vectors with `c(... , ... , ...)`.
10. Vector algebra for numerical vectors.
11. Create a matrix with `cbind()`.
12. Assign row and column names to a matrix.
13. Subsetting a matrix, viewing the *k*th column or the *k*th row of the matrix and also viewing a named row or column.
14. Adding a column to a matrix.
15. Creating a data frame (possible to use mixed data types).
16. subsetting data frames.
17. Manipulating data in a data frame to create a new column/variable in the data frame.

Topic 1: Intro To R and R Studio

hyperlinks are in blue

We will use the open source (free) software R in this class. The easiest way to keep your work together is to use this software is through RStudio. You should follow the directions given below to first install R on your computer and then install RStudio.

Downloading R and R Studio (You must install R on your computer first)

To download R:

1. Go to the website <https://www.r-project.org/>
2. Click on CRAN under Download in the left bar.
3. Choose a mirror site near you to download from in the United States.
4. Under Download and Install R click on the link that corresponds to your operating system.
5. For Windows:
 - Then click on Install R for the first time
 - Click on Download R 3.5.2 for Windows

For Mac:

- Click on the apple in the upper left hand corner of your screen and choose About This Mac from the menu to check which version of Mac OS X you are running,

- Click on the appropriate version; R-3.3.3 pkg for MAC OS X 10.9 and higher or R-3.5.2 pkg for MAC OS X 10.11 and higher.

6. Open the download on your computer to begin installation.

7. Follow setup Wizard.

To download RStudio:

1. Go to the website <https://www.rstudio.com/products/rstudio/download3/>
2. Click on RStudio
3. Click on Download under RStudio Desktop(Free License)
4. Under Installers select the appropriate installer for your system.
5. Open the download on your computer to begin installation.

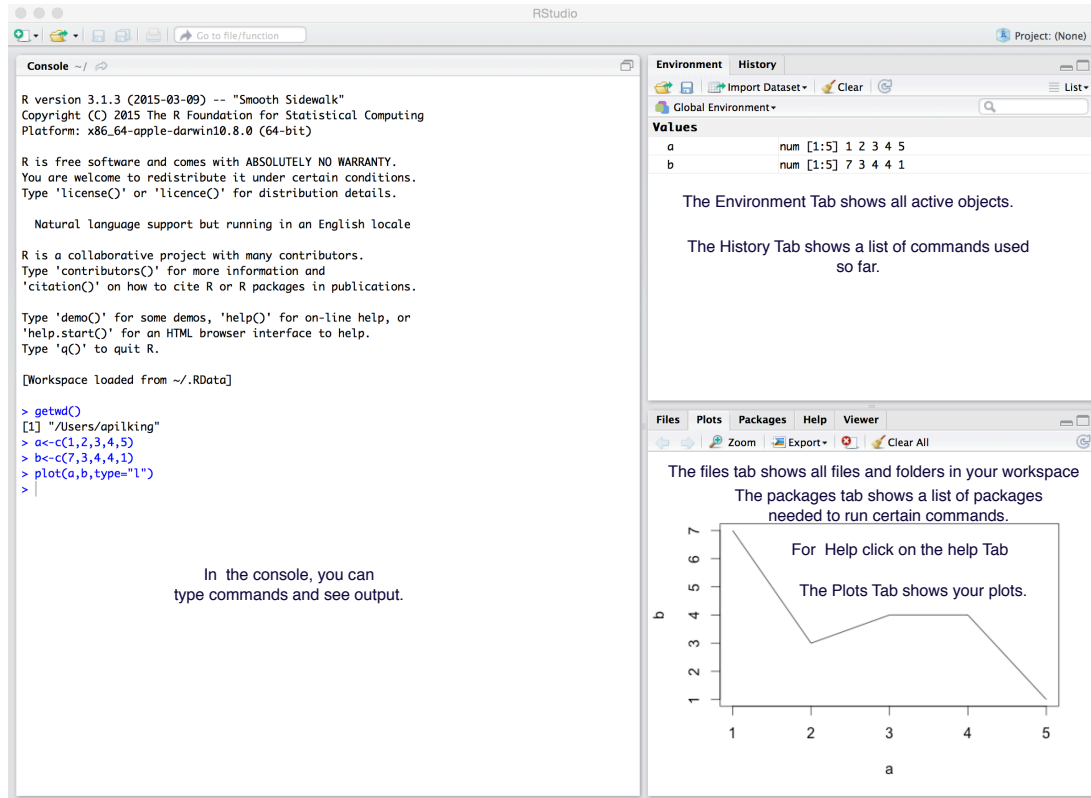
Getting Started The pdf [A \(very\) short introduction to R](#). will help get us started. You will also find the following introduction from Princeton's site helpful:

[Introduction to RStudio](#).

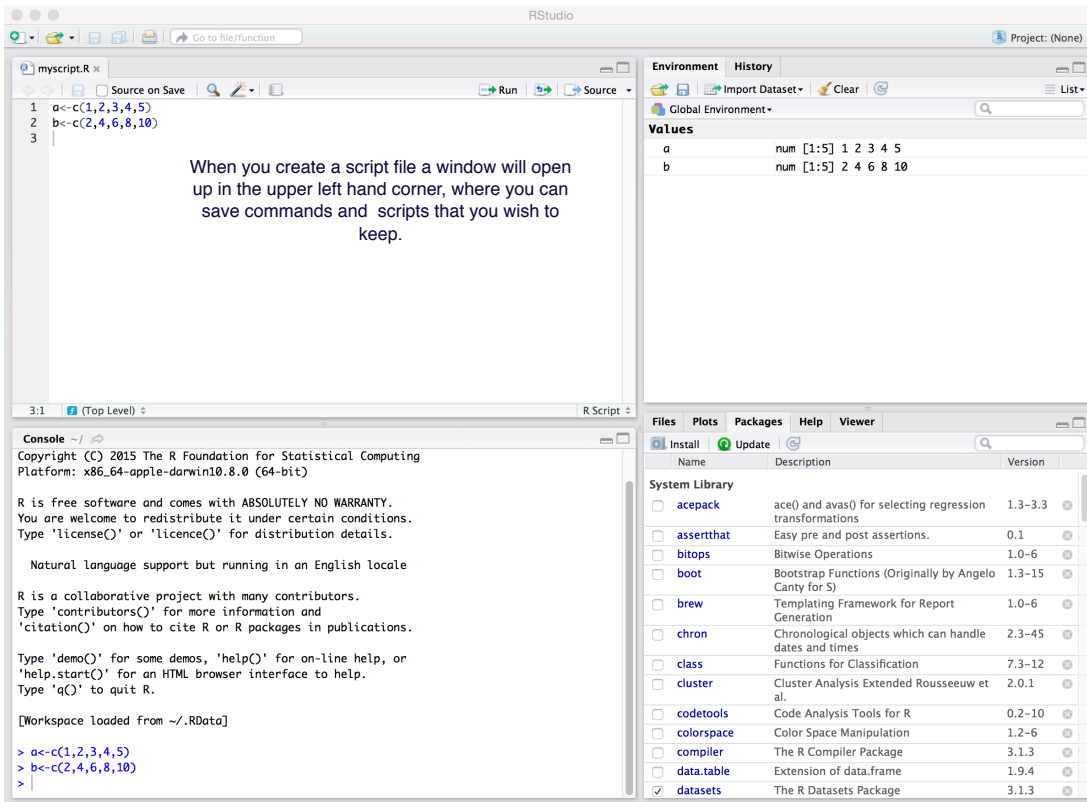
In addition, I have put the following introductions/references in sakai:

A First Course in Statistical Programming with R by Braun and Murdoch, Ch 1 and *The Undergraduate Guide to R* by Trevor Martin.

Let's Get Started: When you open R studio, you will see a window similar to the one shown below:



If you create or open a script file, you will see a fourth window:



Finding/Changing your Working Directory in RStudio To find your current working directory you should type the command `getwd()`. This displays your current working directory where any files you create and save will be saved. You can change your working directory using the command `setwd()`. You should type a command of the form `setwd("directoryname")` to switch to a new directory e.g. `setwd("Desktop/MyRFiles")`. Make sure that the slashes are forward slashes and that you don't forget the apostrophes. Alternatively, you can click on a directory under the **files** tab in the lower right hand window in RStudio and choose **Set As Working Directory** from the menu under **More** in the menu bar. Another option for setting your working directory is to choose **Set Working Directory** under **Session** in the menu bar at the top of your screen.

Calculations in RStudio You can use R as a calculator. Addition and subtraction are denoted by `+` and `-` respectively. Multiplication and division are denoted by `*` and `/` respectively and a power is denoted by `^`. For example $5^2 + 3(19) - 47$ could be calculated with the command (typed at the command symbol `>` in the console window) `5^2 + (3*19) - 47`, R should return the answer 35 when you press **Enter**. You must be careful to use brackets to eliminate ambiguity. For example `(2 - 3) * 9` should give a different answer than `2 - 3 * 9`.

Note the following formula:

For N players in a round robin tournament, where every player plays every other player exactly once, the number of games we need to arrange is:

$$\frac{N \cdot (N - 1)}{2}$$

For example for 6 players, we see that the number of games to be played is half of the number of X's in the table below;

Player	1	2	3	4	5	6
1	–	X	X	X	X	X
2	X	–	X	X	X	X
3	X	X	–	X	X	X
4	X	X	X	–	X	X
5	X	X	X	X	–	X
6	X	X	X	X	X	–

There are 5 X's in each row in this table, thus the number of matches necessary for the tournament is

$$\frac{6 \cdot 5}{2} = 15 \text{ matches.}$$

Example 1. There are 240 teams in a tournament. They are split into 6 groups (by region) for the first round. Three of the groups have 50 teams in them and three of the groups have 30 teams in them. Each group will play a round robin in the first round of the tournament. Calculate the number of games that must be played in the first round of the tournament in R.

If you use brackets and forget to add the closing bracket (RStudio automatically adds closing brackets for a number of computations), the `>` on the command line changes into a `+`. The `+` can also mean that R is still busy with some heavy computation. If you want R to quit what it was doing and give back the `>`, press ESC.

Creating a script file and saving your commands You can save your commands by creating a script file.

- Under the **file** menu in RStudio, choose **New File/R Script**.
- in the new file type **# Script Topic 1** (the **#** symbol in a script file means that line is ignored when you run the file in R).
- Under the **file** menu in RStudio, choose **Save As** and save your file with the name **Script_Topic_1**.
- R will save the file in your current working directory.
- To save a command to your script file, choose **History** in the upper right hand panel of RStudio.
- Click on the command you want to save to your script file, then click on **To Source** in the menu bar to put the command in the script file and then save the script file.

Variables: Naming and storing objects Quite often in a calculation, we need to store intermediate results. We can create symbolic variables to store such objects and then refer to the object by name in calculations. For example if you type the command

```
a<-5
```

and press enter, the R remembers that **a** is 5. If you then type a command using **a** in calculations R will substitute 5 for **a** in the calculations. For example if you type

```
a + 10
```

and press ENTER, R should return the value 15.

If you assign a different value to **a**, R will forget the previous assignment. If you wish to remove the variable **a**, type **rm(a)**.

Some names should be avoided since they are already assigned by the system to variables or functions and this will cause confusion. The main culprits are : **c**, **q**, **t**, **C**, **D**, **F** and **I**.

Example 2 There are 100 teams in a tournament. They are split into 4 groups (by region) for the first round each of which will play a round robin.

- Group A: 20 teams
- Group B: 15 teams
- Group C: 32 teams
- Group D: 33 teams

Calculate the number of games that must be played in the first round of the tournament in R by first calculating





- **ga** = number games played in round 1 by the teams in Group A
- **gb** = number games played in round 1 by the teams in Group B
- **gc** = number games played in round 1 by the teams in Group C
- **gd** = number games played in round 1 by the teams in Group D

and then adding these numbers together.

Types of objects in R: Scalars, Vectors and Matrices R stores data in different types of objects. A *vector* is just a list of numbers and can be created and named using the command **c(...)**.

Example The following set of data shows results for the round robin tournament for Group A in the first round of the FIFA world cup in 2014:

GROUP A

TEAMS	MP	W	D	L	GF	GA	+/-	Pts
 BRAZIL	3	2	1	0	7	2	5	7
 MEXICO	3	2	1	0	4	1	3	7
 CROATIA	3	1	0	2	6	6	0	3
 CAMEROON	3	0	0	3	1	9	-8	0

To create a vector named `gf` showing the goals for each team, we type the command

```
gf <- c(7,4,6,1)
```

and press **ENTER**. To view this vector, we type `gf` and press **ENTER**. We get the following output:

```
[1] 7 4 6 1
```

To create a vector named `ga` showing the goals against each team, we type the command

```
ga<- c(2,1,6,9)
```

and press **ENTER**. To view this vector, we type `ga` and press **ENTER**. We get the following output:

```
[1] 2 1 6 9
```

Vector Algebra for numerical vectors You can perform calculations with vectors just as you with ordinary numbers as long as the vectors have the same length. In this case the symbols `+`, `-`, `*`, and `/` perform the calculations on corresponding components of the vector. For example, we can create a goal differential vector using the command

```
gd<-gf-ga
```

and when we execute the command `gd` we get

```
[1] 5 3 0 -8
```

If we were interested in the value of the statistic $\frac{GF}{GA}$ for each team, we could calculate its value for each team simultaneously by creating the new vector

```
p<-gf/ga
```

If we then type `p` and press **ENTER**, we get the output:

```
[1] 3.5000000 4.0000000 1.0000000 0.1111111
```

If you just want to see 3 decimal places of the data, you can type

```
print(p, digits=3)
```

To multiply a vector by a scalar(number), we use the `*` symbol. For example the command `2*gf` gives us the vector

```
[1] 14 8 12 2
```

Example 3 (a) For the above set of data for Group A, create a vector called `w` showing the number of wins(W) for each team, create a vector called `l` showing the number of losses(L) for each team and create a vector called `d` showing the number of draws(D) for each team.

(b) Create a vector called `wins.minus.losses` showing the number of wins minus the number of losses for each team.

(c) Create a vector called `gp` showing the number of games played for each team by adding the vectors `w`, `d` and `l`.

(d) Counting a draw as half of a win, create a vector called `wp` showing the winning percentage for each team, counting a draw as a half win and a half loss i.e. using the formula for winning percentage

$$\frac{\text{Wins} + (1/2)(\text{Draws})}{\text{Games Played}}$$

Character vectors The data contained in the above vectors is **numeric** data. We can also create vectors containing data of other types. For example, we could create a vector of team names where the components are strings of characters with the following command:

```
teams<-c("Brazil","Mexico", "Croatia", "Cameroon")
```

```
teams
```

```
## [1] "Brazil" "Mexico" "Croatia" "Cameroon"
```

We can see the output by typing `teams` as above.

Data Types in Vectors and Matrices Vectors (and Matrices below) must have entries of the same type when stored in R. If we create a vector with two different types of data, say a mix of character and numeric types, R will convert all of the data to the same a common type, which in this case will be character data (as in the example shown below).

```
mixed<-c("Brazil",1, 2, 3)
```

```
mixed
```

```
## [1] "Brazil" "1" "2" "3"
```

Matrices A rectangular collection of values of the same type can be stored in a **matrix** which A matrix is a rectangular array requiring two numbers (dimensions) to describe its size; the number of rows and the number of columns. For example one could create a matrix with 4 rows and 5 columns showing wins (`w`), draws (`d`), losses (`l`), goals for (`gf`) and goals against (`ga`) from the data in the FIFA example above.

There are a number of ways to create this matrix, one is to create a vector for each column and bind them together with the command `cbind()` as follows:

```
gf<-c(7,4,6,1)
ga<-c(2,1,6,9)
w<-c(2,2,1,0)
d<-c(1,1,0,0)
l<-c(0,0,2,3)
datamatrix<-cbind(w,d,l,gf,ga)
datamatrix
```

```
##      w d l gf ga
## [1,] 2 1 0 7 2
## [2,] 2 1 0 4 1
## [3,] 1 0 2 6 6
## [4,] 0 0 3 1 9
```

Row names Notice that R retains the names of the columns. We can also name the rows using the command `rownames()`, where we specify a vector of row names for the matrix.

```
rownames(datamatrix)<- c("Brazil", "Mexico", "Croatia", "Cameroon")
datamatrix
```

```
##      w d l gf ga
## Brazil 2 1 0 7 2
## Mexico 2 1 0 4 1
## Croatia 1 0 2 6 6
## Cameroon 0 0 3 1 9
```

Subsetting To look at the *k*th column of the matrix, we can use the subsetting command `datamatrix[,k]` and to view the *k*th row, we use the command `datamatrix[k,]`. You can also request columns and rows by name as shown below.

```
datamatrix[,1]
```

```
##   Brazil   Mexico   Croatia Cameroon
##      2       2       1       0
```

```
datamatrix[2, ]
```

```
## w d l gf ga
## 2 1 0 4 1
```

```
datamatrix[, "w"]
```

```
##   Brazil   Mexico   Croatia Cameroon
##      2       2       1       0
```

```
datamatrix["Cameroon",]
```

```
## w d l gf ga
## 0 0 3 1 9
```


Adding a column to a matrix. We can add a column to a matrix by first defining the column and using the `cbind()` command or we can add it directly as shown below.

```
gd<-datamatrix[, "gf"]-datamatrix[, "ga"]
datamatrix<-cbind(datamatrix,gd)
datamatrix

##           w d l gf ga gd
## Brazil   2 1 0 7  2  5
## Mexico   2 1 0 4  1  3
## Croatia  1 0 2 6  6  0
## Cameroon 0 0 3  1  9 -8
```

Example 4 (a) Create the following matrix using R:

$$a = \begin{pmatrix} 2 & 3 & 4 & 5 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix}$$

- (b) Label the rows as A, B and C respectively using the command `rownames()`.
(c) Label the columns as W, X, Y and Z respectively using the command `colnames()`.
(d) Add a fifth column called U which is the product of Columns 1 and 2.

Algebra of matrices Later we will look into the algebra of matrices and learn how to manipulate them to solve systems of equations and produce sports rankings.

Data Frames We will need the matrix format for data which requires an algebra of matrices, however it is easier to manipulate data when we create an object called a **data frame** and this type of object can contain data of different types. We can create a data frame using the `data.frame()` command by listing the columns we wish to put in the data frame inside the brackets or by converting our matrix to a data frame using the same command.

```
dataframefifa<-data.frame(w,d,l,gf,ga,gd)
dataframefifa
```

```
##           w d l gf ga gd
## Brazil   2 1 0 7  2  5
## Mexico   2 1 0 4  1  3
## Croatia  1 0 2 6  6  0
## Cameroon 0 0 3  1  9 -8
```

```
dataframefifa<-data.frame(datamatrix)
dataframefifa
```

```
##           w d l gf ga gd
## Brazil   2 1 0 7  2  5
## Mexico   2 1 0 4  1  3
## Croatia  1 0 2 6  6  0
## Cameroon 0 0 3  1  9 -8
```

We can use the $\$$ symbol to look at columns of a data frame and to manipulate them. Below we create a new column for Wins minus Losses and call it $WminusL$. Note we can name, define and include the new column in the data frame with a single command.

```
dataframefifa$w
```

```
## [1] 2 2 1 0
```

```
dataframefifa$l
```

```
## [1] 0 0 2 3
```

```
dataframefifa$WminusL<-dataframefifa$w-dataframefifa$l  
dataframefifa
```

```
##           w d l gf ga pd WminusL  
## Brazil   2 1 0  7  2  5         2  
## Mexico   2 1 0  4  1  3         2  
## Croatia  1 0 2  6  6  0        -1  
## Cameroon 0 0 3  1  9 -8        -3
```

Example 5: Offensive Efficiency Basketball In Shea and Baker [1], they discuss a statistic designed to measure Offensive efficiency for basketball players. The formula is as follows

$$OE = \frac{FG + A}{FGA - ORB + A + TO},$$

where

OE = Offensive Efficiency,

FG = Field Goals Made,

FGA = Field Goals Attempted,

A = Assists

ORB = Offensive Rebounds,

TO = Turnovers.

The following table shows some game statistics for games played in Oct. 2016 by four NBA players:

NAME	FG	FGA	A	TO	ORB
DeMar DeRozan	11	24	0	0	2
Kevin Durant	11	21	6	2	0
James Harden	7	11	11	5	0
Stephen Curry	12	21	2	2	2

- Create 5 vectors called NAME, FG, FGA, A, TO, ORB with the appropriate data in each.
- Create a data frame called nba.data using the `data.frame()` command and the vectors from part (a).
- Using one command define, create and add a new column for OE.

Packages and RMarkdown(very useful for combining code and output in homework) R's functionality is extended greatly by a large number of packages available on the CRAN website. You can find documentation on the CRAN website with each package explaining what it does and what

commands it uses. The `rmarkdown` package is very useful if you want to present your code and output in a single document (this is an ideal way to submit homework). To install this package type the command `install.packages("rmarkdown")`. You will get a message saying the package has been downloaded.

To print your work to a Word/pdf document, you create a new R Markdown document by choosing **File** from the menu bar and the choosing **New File/R Markdown** from the pull down menu. The file will already have some text and r-code in it for the purposes of demonstration. You should save your file in your workspace by giving it a name. and then click on the `knit` button at the top to see how it prints the sample material. You should read through the basics of using `Knit` and `RMarkdown` under the following link: https://kbroman.org/knitr_knutshell/pages/Rmarkdown.html.

Create a Word(or pdf) Document Keep the `title`, `author`, `date` and `output` options, filling in your own name etc.... You can replace the text with whatever text you please and replace the R code (enclosed by ````\{r}` and `````) by the R code you used to create the data frame in the previous example. Then click on the `knit Word` button to create your document.

Input File:

```
---  
title: "MyWord"  
author: "Annette Pilkington"  
date: "1/13/2019"  
output:  
  pdf_document: default  
---
```

Example 5

```
```\{r cars}  
NAME<-c("DeMar DeRozan","Kevin Durant","James Harden", "Stephen Curry")
NAME
FG<-c(11,11,7,12)
FGA<-c(24,21,11,21)
A<-c(0,6,11,2)
TO<-c(0,2,5,2)
ORB<-c(2,0,0,2)
nba.data<-data.frame(NAME,FG,FGA,A,TO,ORB)
nba.data
```\
```

Output File after Knitting

MyWord

Annette Pilkington
1/13/2019

Example 5

```
NAME<-c("DeMar DeRozan","Kevin Durant","James Harden", "Stephen Curry")
NAME

## [1] "DeMar DeRozan" "Kevin Durant" "James Harden" "Stephen Curry"

FG<-c(11,11,7,12)
FGA<-c(24,21,11,21)
A<-c(0,6,11,2)
TO<-c(0,2,5,2)
ORB<-c(2,0,0,2)
nba.data<-data.frame(NAME,FG,FGA,A,TO,ORB)
nba.data

##           NAME FG FGA  A TO ORB
## 1 DeMar DeRozan 11  24  0  0  2
## 2 Kevin Durant  11  21  6  2  0
## 3 James Harden   7  11 11  5  0
## 4 Stephen Curry 12  21  2  2  2
```

Warning If you use named objects/variables in your markdown file, but do not include the definitions of the variables, your document will not compile and you will get an error message. You can hide the background material that you do not wish to show in the final document with the `include=FALSE` command as follows:

Input File:

```
---
title: "Including all | named variables in your knitting"
author: "Annette Pilkington"
date: "1/13/2019"
output: pdf_document
---
```

The first block of code will not be printed in the knitted document, notice I have set it up so that there is no output, but if there was, it would not be printed

```
```{r background, include=FALSE}
gf<-c(7,4,6,1)
ga<-c(2,1,6,9)
w<-c(2,2,1,0)
l<-c(0,0,2,3)
d<-c(1,1,0,0)
teams<-c("Brazil","Mexico","Croatia","Cameroon")
datamatrix<-cbind(w,d,l,gf,ga)
rownames(datamatrix)<-teams
```
```

The second block of code gives me this output:

```
```{r }
gd<-datamatrix[,"gf"]-datamatrix[,"ga"]
datamatrix<-cbind(datamatrix,gd)
datamatrix
```
```

The file would not run without the first block of code, because the second block uses variables defined in the first block.

Output File after Knitting

Including all named variables in your knitting

Annette Pilkington

1/13/2019

The first block of code will not be printed in the knitted document, notice I have set it up so that there is no output, but if there was, it would not be printed

The second block of code gives me this output:

```
gd<-datamatrix[, "gf"]-datamatrix[, "ga"]
datamatrix<-cbind(datamatrix,gd)
datamatrix
```

```
##          w d l gf ga gd
## Brazil  2 1 0 7  2  5
## Mexico  2 1 0 4  1  3
## Croatia 1 0 2 6  6  0
## Cameroon 0 0 3 1  9 -8
```

The file would not run without the first block of code, because the second block uses variables defined in the first block.

R commands

1. `getwd()` : get working directory.
2. `setwd()` : set working directory.
3. `a<-....` : a equals
4. `rm(a)`: remove the variable `a`.
5. `c(1, 2, 3)`: the vector (1, 2, 3).
6. `c("fee", "fie", "fo", "fum")` : the character vector (fee, fie, fo, fum).
7. `print(w, digits=k)`: show the vector `w` with entries rounded to `k` decimal places.
8. `m<-cbind(w, x, y, z)` : make a matrix `m` whose columns are the vectors `w`, `x`, `y` and `z` respectively.
9. `rownames(m)<-c("A", "B", "C", "D")` : give the rows of `m` the names `A`, `B`, `C` and `D` respectively.
10. `colnames(m)<-c("E", "F", "G", "H")` : give the columns of `m` the names `E`, `F`, `G` and `H` respectively.
11. `m[2,]`: view the second row of `m`.
12. `m[, 2]` : view the second column of `m`.
13. `m[, "H"]` : view the column of `m` named `H`.
14. `m["D",]` : view the row of `m` named `D`.
15. `dfm<-data.frame(w, x, y, z)` : create a data frame (called `dfm`) with columns given by the vectors `x`, `y`, `z` and `w` respectively.
16. `dataframem<-data.frame(m)` : create a data frame (called `dataframem`) from the matrix `m`.
17. `dfm$w` : view the column called `w` in the data frame `dfm`.
18. `dfm$v<-dfm$w - dfm$x` : create new column of the data frame `dfm` called `v` obtained by subtracting the column `x` from the column `w`.
19. `install.packages("rmarkdown")` : Install the `rmarkdown` package.

References

- [1] Shea, Stephen M., and Baker, Christopher E. Basketball Analytics. Advanced Metrics, LLC, Lake St. Louis, MO, 2013.