Getting Started

Joseph Nathan Cohen
Department of Sociology
Program in Data Analytics and Applied Social Research
City University of New York, Queens College
www.josephnathancohen.info

January 29, 2020

Contents

Important Tasks **1**

Preliminary Comment on Teaching & Learning Philosophy **2**

Introducing R and RStudio **3**
Install the Programs 3
Learn the RStudio Interface 3
Updating R 4
Getting Help 4

Markdown Documents **5**
Start a Markdown Document 5
Writing and Coding in a Markdown Document 5

Starting Your Script **5**
Clear the Memory 5
Setting a Working Directory 7
Packages 7

Basic Operations in R **8**
Arithmetic 8
Creating and Manipulating Objects 8
Vectors 9
Data Frames 9
Lists 12

Important Tasks

Your checklist of things to do:

- Ensure that you are in the right class,
- Ensure that you are registered for this class,
- Access the course readings,
- Review the course syllabus,
- Ensure that you have sent me your email address to be included on the class roster,

- Ensure that you have been invited to the class's DataCamp team. Only students who have sent me their email addresses have been added,
- Make sure you have RStudio and R installed on your personal device,
- Get started with your first DataCamp course

Preliminary Comment on Teaching & Learning Philosophy

In this course, my goal is to ensure that everyone who gets a Master's degree with us has demonstrated the ability to implement and make sense of a reasonably wide array of advanced quantitative analytics operations. If I do my job right, then our graduates will:

- Be reasonably fluent in R, the industry-standard statistical programming platform in the field of data science,
- Have a level of statistical analysis skills that clearly and meaningfully surpass what is offered by leading social science degree programs (in any discipline, from any school)
- Have a level of analytical proficiency that compares well with the typical social science academic (who is not a quantitative methodology specialists),
- Have developed the capacity to serve as the quantitative expert in any environment,
- Have developed the capacity to problem-solve and develop their skills after leaving campus.

More Like Learning a Language or Sport. Learning to do data anlaysis is a lot like learning a language or a sport. It's not as if I someone can take you aside for two hours, teach you a list of vocabulary words and a grammatical conjugation scheme, and then you're off to have French conversations with strangers on the streets of Paris. It's more like learning ice skating – you spend a few weeks feeling tense, awkward and afraid of falling. After a few more weeks, you make your peace with it. Give it a few months, and you are able to do fun things and try things out, and then it becomes fun. So students should embrace that awkward period, and not get too fixated and frustrated that they aren't learning things immediately. You will remember how to do something after you get sick of making same mistake, or forgetting same thing, for the 20th time.[1]

Getting Your Reps. One half of your grade is given for completing DataCamp modules. I consider this part of the course to be equivalent to training or practice for an athlete. Just as a swimming trainer might require an athlete to practice their strokes for several hours a week at the pool, I ask you to practice your coding on DataCamp. I promise that anyone who completes all of these courses will be happy to have done so by the semster's end.

Reading. There's a ton of reading. The only required reading are my class notes. The additional reading is there in case you want to take a deeper dive (or so that you have something to read if you want to come back to this stuff in the future.) The only thing I care about is (1) completed DataCamp courses, (2) quality assignment submissions, and (3) confidence that you understand what we are doing and will perform competently in the field.

Learn to Learn Independently. One of my goals is that you become an independent learning in analytics. For this reason, I am not a hand-holder. If you have a question, try to solve it independently first via Google. That's how I do it. That's how everyone does it. I recommend StackOverflow as a source of answers. If your problem is particularly complicated, then come visit me during office hours or by appointment.

How to Study? There is no substitute for reading the class notes, doing the DataCamp exercises, and making an effort to deliver polished assignments. Besides doing these basic things, I recommend the following:

- Put all of the class notes and handouts into a binder. Get a hole punch so that they can be included in the binder. Put a divider for each week of class materials.
- During class and while you are doing your assignments, take notes on three hole-punched paper. Save those notes in your binder.

[1]Side note: this view on learning analytics is partly why I force students to solve their own problems using StackOverflow, etc. instead of immediately answering their programming questions. It incentivizes investing the mental energy into memorizing how to perform an operation.

- The best notes have explanations and recipes for performing operations that are *in your own words*. Sometimes, material seems obvious when you are first learning it, but then your memory fades. Good notes capture as much as possible during that moment of clarity, so that you can recall later.

Introducing R and RStudio

This semester, most of our work will be done with two programs, **R** and **RStudio**. **R** is an open-source platform for analyzing statistics. **RStudio** is a user interface for R. We use R *through* RStudio. RStudio makes it easier to use R (and adds some functionalities as well).

Install the Programs

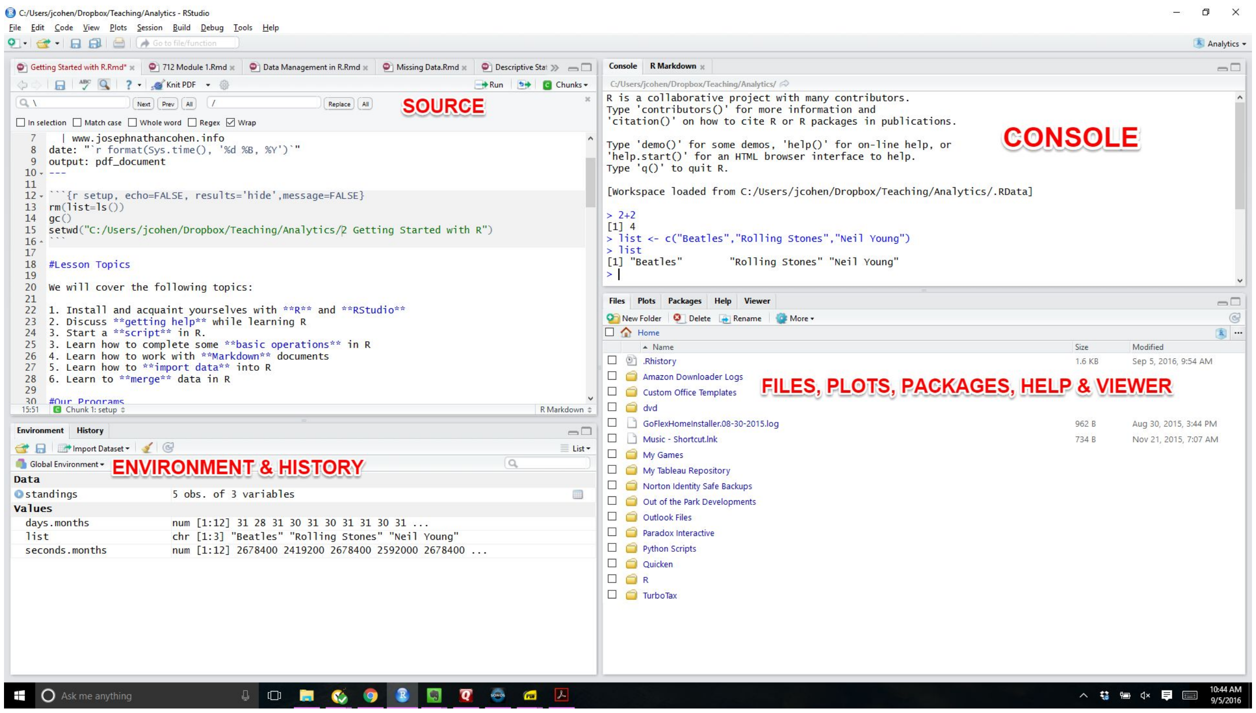
You can download these programs from the Internet:

- Download R from the Comprehensive R Archive Network at https://cran.r-project.org/.
- Download RStudio from the R Studio web site at https://www.rstudio.com/.

32-bit or 64-bit version?. Your choice depends on your computer's CPU. If you have a newer computer, chances are that it has a 64-bit processor. For more information on how to determine your system type, visit this helpful page at http://www.computerhope.com/issues/ch001121.htm

Learn the RStudio Interface

The RStudio interface has several panes (see Figure 1 above).


C:/Users/jcohen/Dropbox/Teaching/Analytics - RStudio
File Edit Code View Plots Session Build Debug Tools Help
Getting Started with R.Rmd* | 712 Module 1.Rmd | Data Management in R.Rmd | Missing Data.Rmd | Descriptive Stat
Knit PDF | Run | Chunks
SOURCE
CONSOLE
FILES, PLOTS, PACKAGES, HELP & VIEWER
ENVIRONMENT & HISTORY
Console | R Markdown
Files | Plots | Packages | Help | Viewer
Environment | History
Import Dataset | Global Environment | List
Ask me anything
10:44 AM 9/5/2016

~~~
  | www.josephnathancohen.info
date: "`r format(Sys.time(), '%d %B, %Y')`"
output: pdf_document
---

```{r setup, echo=FALSE, results='hide',message=FALSE}
rm(list=ls())
gc()
setwd("C:/Users/jcohen/Dropbox/Teaching/Analytics/2 Getting Started with R")
```

#Lesson Topics

We will cover the following topics:

1. Install and acquaint yourselves with **R** and **RStudio**
2. Discuss **getting help** while learning R
3. Start a **script** in R.
3. Learn how to complete some **basic operations** in R
4. Learn how to work with **Markdown** documents
5. Learn how to **import data** into R
6. Learn to **merge** data in R

#Our Programs
~~~

```
C:/Users/jcohen/Dropbox/Teaching/Analytics/
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[Workspace loaded from C:/Users/jcohen/Dropbox/Teaching/Analytics/.RData]

> 2+2
[1] 4
> list <- c("Beatles","Rolling Stones","Neil Young")
> list
[1] "Beatles"        "Rolling Stones" "Neil Young"
> 
```

```
Data
standings          5 obs. of 3 variables
Values
  days.months      num [1:12] 31 28 31 30 31 30 31 31 30 31 ...
  list             chr [1:3] "Beatles" "Rolling Stones" "Neil Young"
  seconds.months   num [1:12] 2678400 2419200 2678400 2592000 2678400 ...
```

Name	Size	Modified
.Rhistory	1.6 KB	Sep 5, 2016, 9:54 AM
Amazon Downloader Logs		
Custom Office Templates		
dvd		
GoFlexHomeInstaller.08-30-2015.log	962 B	Aug 30, 2015, 3:44 PM
Music - Shortcut.lnk	734 B	Nov 21, 2015, 7:07 AM
My Games		
My Tableau Repository		
Norton Identity Safe Backups		
Out of the Park Developments		
Outlook Files		
Paradox Interactive		
Python Scripts		
Quicken		
R		
TurboTax		

Figure 1: The RStudio Window

- **Source.** Programs that you author to run in R. It is where you write and save your code. Similar to the Do File Editor in Stata.
- ***Console.*** Displays the results of processed R commands. Similar to Results window in Stata.
- ***Environment.*** An *environment* is a set of associated objects (data, functions, etc.) that are loaded into R. For more on environments, click here

- ***History.*** A list of commands that have been run in your R session.
- ***Files.*** Lists files in a folder.
- ***Plots.*** Shows graphs that you've created.
- ***Packages.*** *Packages* are libraries of commands. You can install and load packages in this window. For more on packages, see below.
- ***Help.*** For information on packages and commands.
- ***Viewer.*** Shows local web content.

Updating R

R updates regularly. You can update your version of R through RStudio's console (see below) by using this code:

```
install.packages("installr")
library(installr)
updateR()
```

Getting Help

Learning to do statistical analyses in R is like learning a foreign language. You can't read a French-English dictionary and immediately be fluent in French. Likewise, and you can't read a list of R commands and be fluent in R. It takes practice. You have to practice things over and over again until you've memorized them. That is how you will learn.

At first, the learning process is a struggle. You will constantly have to look up commands, and you'll be frustrated at how difficult it is to do things that you can already do on Excel or Stata. Honestly, the first few weeks R are brutal.

Do not be discouraged if you have to look up EVERYTHING when you are getting started. It's like learning a language. Once you look something up enough times, you'll have it memorized. Once you look up enough commands enough times, you'll have a basic grasp of R programming

As you get the ball rolling, keep in mind that you have many resources to help.

1. **Your class notes and reading assignments.** That's what they're for.
2. **O'Reilly Books.** If the assigned readings aren't working out for you, look into O'Reilly books. They're reasonably priced and pretty good, straightforward "how to" books. I recommend the *R Cookbook* by Paul Teetor, *R in a Nutshell* from Joseph Adler, and *R Graphics Cookbook* by Winston Chang. They're all great.
3. **Google.** There's lots of help online. For example, if you want to know how to get the mean value of a variable, then (1) go to Google and (2) do a query along the lines of "get the mean of a variable in R". I did this query while writing these notes. The answer was in the first query return. If the answer isn't in the first few returns or on the first page of your query, try scrolling down or looking at page 2. I recommend checking out any Google returns from the following sites: Stack Overflow, R-Bloggers, inside-R, and any university sites.
4. **R's Help Function.** The answer to your question might be in R's internal help functions. In the console, type two question marks (??) and the keyword. For example:

```
??mean
```

This will bring up a list of possibly useful commands in the RStudio Help window.

If you know the name of the command and want to call up the help page directly, only use one question mark:

```
?mean
```

Markdown Documents

Scripts are prewritten sequences of R code, which can be executed as a group. Stata *.do files are an example of a script. Scripts are useful to track and execute extended sequences of code. To start a script, click **File > New File > R Script**

To add notes to your script that will not run, place a hashtag (#) before the line you're writing. Lines without hashtags will be run as part of a script.

```
# put a hashtag before a line to create a comment
# comments will not be run in R
```

Markdown is a kind of script that allows you to conduct analyses and create reports as part of the same document. It is like a Word document and a do file rolled into one. I prepared this document in Markdown.^[***Creating PDF Documents Using Markdown.*** If you want to produce PDF documents (like this one) using R Markdown, you need to install additional programs to help RStudio complete the task. I recommend installing MiKTeX, which can be downloaded at http://miktex.org/download. For an illustrated tutorial, see Soren L. Kristiansen's article on *Medium*.[^10*Note:* Restart RStudio after installing MiKTeX.]

Start a Markdown Document

1. On the menu bar, click "File" > "New File" > "R Markdown…"
2. A popup window will come up. Select "DOcument" in the left panel of the pop-up window. In the right window, enter a title and your name, and choose which type of document you wish to produce (see footnote for setting up PDF printing)
3. A generic Markdown document will appear. You can edit it to do whatever you want.

Writing and Coding in a Markdown Document

A Markdown document allows you to perform statistical analysis and reporting in the same document. Let's try playing with the reporting function. In your document, write the following lines:

```
# Introduction
What does our baseball team need to do in order to win more games?  This analysis
suggests that batting average, low strikeouts, and premium players are related with
a team's winning percentage.
```

Then click the "Run All" button, depicted below (or, on a PC, hit CTRL + ALT + R). If you did it correctly, you should see a Microsoft Word document pop up with a title, your name, the date, an "Introduction" header, and the passage you typed above. You can do much more in terms of report writing. Check out the Markdown Cheat Sheet for more. You can also download these notes in Markdown to see what I did. But we are going to quickly focus on coding.

In Markdown, you code in chunks. Chunks tell RStudio to transition from document preparation to R coding. When you close the chunk, RStudio reverts to document prreparation. Figure 2 (below) depicts a chunk that begins on line 6 and ends at line 12.

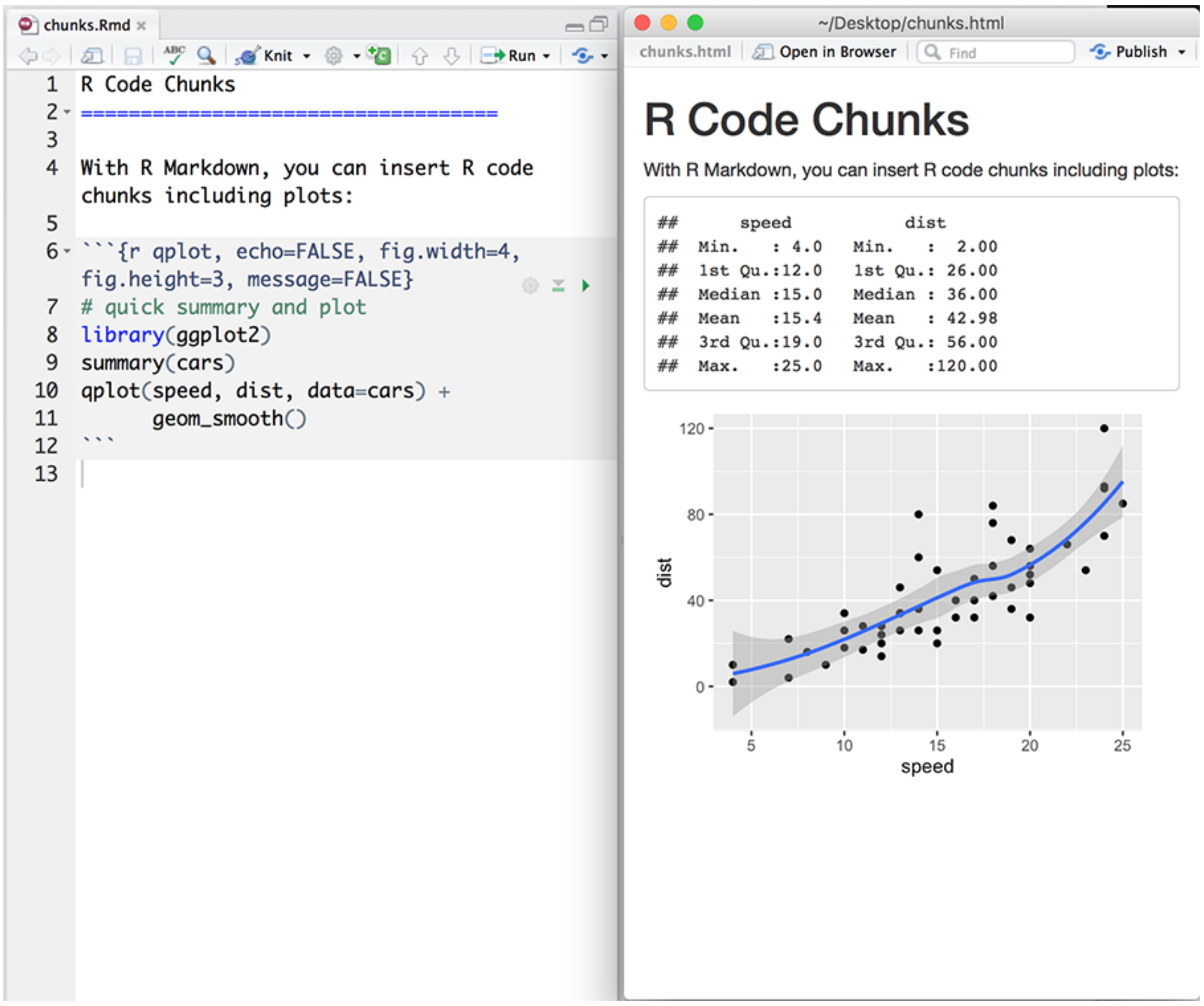
Note that you can also insert Python chunks into a Markdown document with some setup.

Starting Your Script

Clear the Memory

I always start my Markdown files with the following commands, which clear R's memory:

```
rm(list=ls())
gc()
```



~~~
R Code Chunks
=====================================

With R Markdown, you can insert R code
chunks including plots:

```{r qplot, echo=FALSE, fig.width=4,
fig.height=3, message=FALSE}
# quick summary and plot
library(ggplot2)
summary(cars)
qplot(speed, dist, data=cars) +
      geom_smooth()
```
~~~

R Code Chunks

With R Markdown, you can insert R code chunks including plots:

```
##      speed           dist       
##  Min.   : 4.0   Min.   :  2.00  
##  1st Qu.:12.0   1st Qu.: 26.00  
##  Median :15.0   Median : 36.00  
##  Mean   :15.4   Mean   : 42.98  
##  3rd Qu.:19.0   3rd Qu.: 56.00  
##  Max.   :25.0   Max.   :120.00  
```

Figure 2: A Markdown R Chunk

```
##          used (Mb) gc trigger (Mb) max used (Mb)
## Ncells 401173 21.5     841297   45   609057 32.6
## Vcells 747830  5.8    8388608   64  1606118 12.3
```

When you put "include = F" in the code to open a chunk, you are telling R not to publish the results of that operation in the report.

Setting a Working Directory

A *working directory* is the data that stores the files with which R will be working and those that R produces over the course of an analysis. Setting a working directory is an important part of writing an R script.

To set the working directory, use the **setwd()** command. Just cut and paste the path to the directory in which you are working into the command. Note that you have to reverse the slashes in the path - it won't work if you cut and paste without replacing the slashes.

To set this folder as the working directory:

```
#directory <- "C:/Users/jcohen/Dropbox/Teaching/SOC 712/Module 1"
directory <- "E:/Dropbox/Teaching/SOC 712/Module 1"
setwd(directory)
```

Note the quotation marks and that I changed the slashes (from the one above the "enter" key to the one beside the lower-right "shift" key on my keyboard)

To see the files in your working directly, use the **list.files()** command:

```
list.files()
```

Packages

Unlike Stata, R doesn't load up all of its commands when you first open the program. It only loads up basic commands. You will have to ask R to load up the libraries of commands (called **"packages"**) before you use them.

Different packages do different things. For example, **dplyr** has commands that help you manipulate data. **Amelia** is for missing data imputation. **ggplot2** is a package for graphing. There are many packages. For more on packages, click here

RStudio lists the installed packages under the "Packages" tab. You can install more packages by clicking the "Install" button in that tab.

Install a Package

To install a package for the first time, use the [**install.packages()**] command.

For example, to install the **ggplot2** package:

```
install.packages("ggplot2")
```

Troubleshooting. If you get an error related to "trying to use CRAN without setting a mirror", you may have to specify a respository from which to download the package:

```
install.packages("ggplot2",repos="http://cran.rstudio.com/")
```

You can set your respository in RStudio through **Tools > Global Options... > Packages**

Loading a Package

To load a package, use the **library()** command.

For example, to load the **ggplot2** package:

```
library(ggplot2)
```

Basic Operations in R

Arithmetic

R performs basic calculations as in other commonly-used programs. It follows the PEDMAS order of operations, in which operations in parentheses are calculated first, followed by division and multiplication, then addition and subtraction, from left to right.

```
2+2
```

```
## [1] 4
```

```
3*5
```

```
## [1] 15
```

```
10^2
```

```
## [1] 100
```

```
log(10)
```

```
## [1] 2.302585
```

```
exp(2.302585)
```

```
## [1] 9.999999
```

```
5+5/2
```

```
## [1] 7.5
```

```
(5+5)/2
```

```
## [1] 5
```

Sequences

For simple sequences, you can use two numbers separated by a colon:

```
2:5
```

```
## [1] 2 3 4 5
```

For more complicated sequences, use **seq()** command. In that command, you designate (1) the beginning and (2) end of the sequence, then (3) the size of each step. For more, type ?seq in the console. For example:

```
seq(0,100,10)
```

```
##  [1]   0  10  20  30  40  50  60  70  80  90 100
```

Creating and Manipulating Objects

You can assign data to objects using an arrow (the left triangular bracket followed by a dash (<-)) to assign numbers or text to an object or variable name. For example, I want to assign the value 2 to the letter z:

```
z <- 2
z*10
```

```
## [1] 20
```

Note that the object "z" is listed in your environment tab once you have created it. To get rid of the object, use the **rm()** command, as below:

```
rm(z)
```

Vectors

You can assign multiple elements to a variable. To do that, you list the objects in a collector operator **c()**. This operator tells R that you are presenting a list:

```
months <- c("January","February","March","April","May","June","July","August",
            "September","October","November","December")
months
```

```
##  [1] "January"   "February"  "March"     "April"     "May"       "June"
##  [7] "July"      "August"    "September" "October"   "November"  "December"
```

```
days.in.month <- c(31,28,31,30,31,30,31,31,30,31,30,31)
days.in.month
```

```
##  [1] 31 28 31 30 31 30 31 31 30 31 30 31
```

When you apply a calculation to a variable with multiple elements, the calculation will be performed on each element. For example:

```
days.in.month * 100
```

```
##  [1] 3100 2800 3100 3000 3100 3000 3100 3100 3000 3100 3000 3100
```

You can create new objects based on old ones:

```
hours.in.month <- days.in.month * 24
hours.in.month
```

```
##  [1] 744 672 744 720 744 720 744 744 720 744 720 744
```

Data Frames

Data frames are rectangularized tables of data. They are the kinds with which we typically work in statistics.

Create a Frame from Vectors

You can collect these vectors[2] into the kind of data table with which you worked in Stata. Here, each row is a subject, and each column is a variable. To do that, we use the **data.frame()** operation, which tells R we are creating a data table. For example:

```
calendar <- data.frame(months, days.in.month, hours.in.month)
calendar
```

```
##       months days.in.month hours.in.month
## 1    January            31            744
## 2   February            28            672
## 3      March            31            744
## 4      April            30            720
## 5        May            31            744
## 6       June            30            720
## 7       July            31            744
## 8     August            31            744
## 9  September            30            720
```

[2] *Vectors* are these objects with multiple elements. Vectors have the same type of data (e.g., numbers, strings)

```
## 10   October             31            744
## 11  November             30            720
## 12  December             31            744
```

View Top of a Data Frame

For big tables, you can look at the first few rows by using the **head()** command. To see the first three rows of the data frame "calendar":

```
head(calendar, 3)
```

```
##     months days.in.month hours.in.month
## 1  January            31            744
## 2 February            28            672
## 3    March            31            744
```

See a Data Frame's Column Names

Find the names of the variables in the data frame using the **names()** command:

```
names(calendar)
```

```
## [1] "months"         "days.in.month"  "hours.in.month"
```

Calling Columns from a Data Frame

You can call up specific variables in the data frame using a dollar sign ($) designator:

```
calendar$hours.in.month
```

```
##  [1] 744 672 744 720 744 720 744 744 720 744 720 744
```

```
hours <- calendar$hours.in.month
hours
```

```
##  [1] 744 672 744 720 744 720 744 744 720 744 720 744
```

You can add variables to a data frame using this same designator:

```
calendar$month.num <- 1:12
head(calendar)
```

```
##     months days.in.month hours.in.month month.num
## 1  January            31            744         1
## 2 February            28            672         2
## 3    March            31            744         3
## 4    April            30            720         4
## 5      May            31            744         5
## 6     June            30            720         6
```

To call particular elements of a data frame, use a square bracket with a row then column designation. To call up the second row:

```
calendar[2,]
```

```
##     months days.in.month hours.in.month month.num
## 2 February            28            672         2
```

To call out the second column:

```
calendar[,2]
```

```
##  [1] 31 28 31 30 31 30 31 31 30 31 30 31
```

To call out the second row at the second column:

```
calendar[2,2]
```

```
## [1] 28
```

To call out the second through fifth rows:

```
calendar[2:5,]
```

```
##     months days.in.month hours.in.month month.num
## 2 February            28            672         2
## 3    March            31            744         3
## 4    April            30            720         4
## 5      May            31            744         5
```

Summarize Variables in a Data Frame

You can get summary information or perform operations on data frame columns. For example, to get the mean and median number of hours in a month, plus the standard deviation:

```
mean(calendar$hours.in.month)
```

```
## [1] 730
```

```
median(calendar$hours.in.month)
```

```
## [1] 744
```

```
sd(calendar$hours.in.month)
```

```
## [1] 21.60808
```

Pick Out a Subset

To select the observations in the data frame "calendar" in which "days.in.month" is equal to 31subset

```
subset(calendar, days.in.month == 31)
```

```
##      months days.in.month hours.in.month month.num
## 1   January            31            744         1
## 3     March            31            744         3
## 5       May            31            744         5
## 7      July            31            744         7
## 8    August            31            744         8
## 10  October            31            744        10
## 12 December            31            744        12
```

Sort by a Variable

To sort by "hours.in.month" variable:

```
#order() returns the vector positions in ascending or descending rank
order(calendar$hours.in.month)
```

```
##  [1]  2  4  6  9 11  1  3  5  7  8 10 12
```

```
#Use it to call out the order of rows in a data frame:
calendar[order(calendar$hours.in.month),]
```

```
##       months days.in.month hours.in.month month.num
## 2   February            28            672         2
## 4      April            30            720         4
## 6       June            30            720         6
## 9  September            30            720         9
## 11  November            30            720        11
## 1    January            31            744         1
## 3      March            31            744         3
## 5        May            31            744         5
## 7       July            31            744         7
## 8     August            31            744         8
## 10   October            31            744        10
## 12  December            31            744        12
```

Lists

Lists are sets of objects that are of different types. For example, you might bind a data table, regression results, and tables of regression diagnostic results into a common object. You would assemble them in an object of this sort.

Creating Lists

For example, imagine I were to create a table summarizing our "hours.in.month" variable:

```
results <- summary(calendar$hours.in.month)
results
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     672     720     744     730     744     744
```

And I wanted to bind these findings to the data used to create it:

```
#Create a new object "data", which is just a copy of the
#Bind the data and the results, with labels
calendar.findings <- list(data = calendar,
                          results = results)

calendar.findings
```

```
## $data
##       months days.in.month hours.in.month month.num
## 1    January            31            744         1
## 2   February            28            672         2
## 3      March            31            744         3
## 4      April            30            720         4
## 5        May            31            744         5
## 6       June            30            720         6
## 7       July            31            744         7
## 8     August            31            744         8
## 9  September            30            720         9
## 10   October            31            744        10
## 11  November            30            720        11
## 12  December            31            744        12
##
## $results
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     672     720     744     730     744     744
```

Calling List Elements

```
head(calendar.findings$data, 6)
```

```
##     months days.in.month hours.in.month month.num
## 1  January            31            744         1
## 2 February            28            672         2
## 3    March            31            744         3
## 4    April            30            720         4
## 5      May            31            744         5
## 6     June            30            720         6
```