Chapter 2

Multi Dimensional Objects in R

2.1 Data Frames

In statistics, we generally work with several pieces of information for a sample; e.g., the amount of savings in dollars, political affiliation, and presidential approval status reported by a sample of 350 randomly selected American adults. Since we will ask questions about these types of information later, we will want to store all the information together.

2.1.1 Creating Data Frames

To do this in R, we create a data frame, which is just a fancy word for a collection of vectors of information. In Example 2.1, we show how to create a data frame in R.

Example 2.1. Assigning Data Frames to Objects in R. Later, in Example 2.3, we'll load data from a file for observations of the invariant natural killer T (iNKT) cell populations in the spleen from 48 mice with three types of SHIP1 deletion (Anderson et al., 2015). In this example we'll create a data frame directly, which is rather tedious, and demonstrate how to ask for simple previews of the data.

```
> r<-data.frame(iNKT=c(1.34,1.19,0.89,0.88,0.98,0.85,0.93,0.87,1.02,0.96,
+                      0.77,1.17,1.12,1.61,1.18,1.00,0.88,1.08,1.12,0.69,
+                      1.29,1.16,0.94,0.94,0.58,0.87,1.11,1.08,0.92,1.23,
+                      0.94,0.41,0.57,0.19,1.13,0.33,0.60,0.16,0.33,0.77,
+                      1.44,0.62,0.70,0.77,0.47,0.44,0.28,0.22),
+                Ship1=c("+/+","+/+","+/+","+/+","+/+","+/+","+/+","+/+",
+                        "+/+","+/+","+/+","+/+","+/+","+/+","+/-","+/-",
+                        "+/-","+/-","+/-","+/-","+/-","+/-","+/-","+/-",
+                        "+/-","+/-","+/-","+/-","+/-","+/-","+/-","+/-",
+                        "-/-","-/-","-/-","-/-","-/-","-/-","-/-","-/-",
+                        "-/-","-/-","-/-","-/-","-/-","-/-","-/-","-/-")
+               )
#See the first 6 rows of data frame we created
> head(r)
iNKT Ship1
1 1.34   +/+
```

```
2 1.19   +/+
3 0.89   +/+
4 0.88   +/+
5 0.98   +/+
6 0.85   +/+
#See the last 6 rows of data frame we created
> tail(r)
iNKT Ship1
43 0.70   -/-
44 0.77   -/-
45 0.47   -/-
46 0.44   -/-
47 0.28   -/-
48 0.22   -/-
#Ask for the number of rows
> nrow(r)
[1] 48
#Ask for the number of columns
> ncol(r)
[1] 2
#Ask for the column names -- what data is available
> names(r)
[1] "iNKT"  "Ship1"
> summary(r) #Summarize data vectors
iNKT        Ship1
Min.   :0.1600   -/-:16
1st Qu.:0.6150   +/-:18
Median :0.9050   +/+:14
Mean   :0.8546
3rd Qu.:1.1125
Max.   :1.6100
```

Here, we see the data is loaded together creating a data frame with two columns, one for each variable or measurement, and forty-eight rows, one for each observation.

2.1.2 Subsetting Data Frames

Example 2.2. Later, we'll want to ask questions about these objects individually. In R, there are several of ways to subset data. We can specify which columns or rows we want to retain or specify which to get rid of. We can ask for columns by name using '$', and we can specify which rows and columns we want using the column/row numbers or names. Like vectors, we use single brackets to ask for elements of data frames; for data frames we ask for elements via this syntax [row(s),column(s)] as demonstrated below.

```
#Ask for the iNKT observations for the 48 mice
> r$iNKT #By column name
[1] 1.34 1.19 0.89 0.88 0.98 0.85 0.93 0.87 1.02 0.96 0.77 1.17 1.12 1.61
[15] 1.18 1.00 0.88 1.08 1.12 0.69 1.29 1.16 0.94 0.94 0.58 0.87 1.11 1.08
[29] 0.92 1.23 0.94 0.41 0.57 0.19 1.13 0.33 0.60 0.16 0.33 0.77 1.44 0.62
```

```
[43] 0.70 0.77 0.47 0.44 0.28 0.22
> r[,1] #By column number
[1] 1.34 1.19 0.89 0.88 0.98 0.85 0.93 0.87 1.02 0.96 0.77 1.17 1.12 1.61
[15] 1.18 1.00 0.88 1.08 1.12 0.69 1.29 1.16 0.94 0.94 0.58 0.87 1.11 1.08
[29] 0.92 1.23 0.94 0.41 0.57 0.19 1.13 0.33 0.60 0.16 0.33 0.77 1.44 0.62
[43] 0.70 0.77 0.47 0.44 0.28 0.22
> r[,"iNKT"] #By column name
[1] 1.34 1.19 0.89 0.88 0.98 0.85 0.93 0.87 1.02 0.96 0.77 1.17 1.12 1.61
[15] 1.18 1.00 0.88 1.08 1.12 0.69 1.29 1.16 0.94 0.94 0.58 0.87 1.11 1.08
[29] 0.92 1.23 0.94 0.41 0.57 0.19 1.13 0.33 0.60 0.16 0.33 0.77 1.44 0.62
[43] 0.70 0.77 0.47 0.44 0.28 0.22
#Ask for the Ship1 status observations for the 48 mice
> r$Ship1 #By column name
[1] +/+ +/+ +/+ +/+ +/+ +/+ +/+ +/+ +/+ +/+ +/+ +/+ +/+ +/+ +/+ +/- +/- +/- +/-
[19] +/- +/- +/- +/- +/- +/- +/- +/- +/- +/- +/- +/- +/- +/- -/- -/- -/- -/-
[37] -/- -/- -/- -/- -/- -/- -/- -/- -/- -/- -/- -/-
Levels: -/- +/- +/+
> r[,2] #By column number
[1] +/+ +/+ +/+ +/+ +/+ +/+ +/+ +/+ +/+ +/+ +/+ +/+ +/+ +/+ +/+ +/- +/- +/- +/-
[19] +/- +/- +/- +/- +/- +/- +/- +/- +/- +/- +/- +/- +/- +/- -/- -/- -/- -/-
[37] -/- -/- -/- -/- -/- -/- -/- -/- -/- -/- -/- -/-
Levels: -/- +/- +/+
> r[,"Ship1"] #By column name
[1] +/+ +/+ +/+ +/+ +/+ +/+ +/+ +/+ +/+ +/+ +/+ +/+ +/+ +/+ +/+ +/- +/- +/- +/-
[19] +/- +/- +/- +/- +/- +/- +/- +/- +/- +/- +/- +/- +/- +/- -/- -/- -/- -/-
[37] -/- -/- -/- -/- -/- -/- -/- -/- -/- -/- -/- -/-
Levels: -/- +/- +/+
```

The output when asking for a column from the data frame is a vector of data, which we can ask new questions about. Note that the iNKT data is of type numeric and class double, and the Ship1 data is treated as a factor vector.

```
#Ask for the observations for mouse #1
> r[1,] #By row number
iNKT Ship1
1 1.34   +/+
#Ask for the iNKT observation for mouse #1
> r$iNKT[1] #By row number and column name
[1] 1.34
> r[1,1] #By row number and column number
[1] 1.34
> r[1,"iNKT"] #By row number and column name
[1] 1.34
#Ask for the Ship1 observation for mouse #1 #By row number and column name
> r$Ship1[1]
[1] +/+
Levels: -/- +/- +/+
> r[1,2]  #By row number and column number
[1] +/+
Levels: -/- +/- +/+
```

```
> r[1,"Ship1"] #By row number and column name
[1] +/+
Levels: -/- +/- +/+
> r[2,]
iNKT Ship1
2 1.19   +/+
#Grabs the first three rows of the ship1dat dataset
> r[c(1,2,3),]
iNKT Ship1
1 1.34   +/+
2 1.19   +/+
3 0.89   +/+
```

We can ask for the frequencies of iNKT cell populations for observations with a particular SHIP1 status using the operators from Table 1.4.4; it is as simple as asking R to run the command on data which have the specified SHIP1 status. We can ask R this question by asking for the frequencies of iNKT cell populations for which the SHIP1 status is "-/-", for example.

```
#Grabs frequencies of iNKT cell populations for -/- SHIP1 status
> r$iNKT[which(r$Ship1=="-/-")] #subset(x=r$iNKT,subset=r$Ship1=="-/-") equivalently
[1] 1.34 1.19 0.89 0.88 0.98 0.85 0.93 0.87 1.02 0.96 0.77 1.17 1.12 1.61
#Grabs frequencies of iNKT cell populations for +/- SHIP1 status
> r$iNKT[which(r$Ship1=="+/-")] #subset(x=r$iNKT,subset=r$Ship1=="+/-") equivalently
[1] 1.18 1.00 0.88 1.08 1.12 0.69 1.29 1.16 0.94 0.94 0.58 0.87 1.11 1.08 0.92 1.23
[17] 0.94 0.41
#Grabs frequencies of iNKT cell populations for +/+ SHIP1 status
> r$iNKT[which(r$Ship1=="+/+")] #subset(x=r$iNKT,subset=r$Ship1=="+/+") equivalently
[1] 1.34 1.19 0.89 0.88 0.98 0.85 0.93 0.87 1.02 0.96 0.77 1.17 1.12 1.61
```

The == indicates that we're asking for comparison and not setting a object. Recall, we read the code as

r$iNKT	'the frequencies of iNKT cell populations'
[	'for'
which(	'which'
r$Ship1=="+/+")]	'SHIP1 status is +/+.'

We can also ask for parts of the data frame based on the iNKT populations observed.

```
#Ask for observations (rows) where iNKT is greater than 1
#leaving the second index blank returns all columns
> r[which(r$iNKT>1),] #subset(x=r,subset=r$iNKT>1) equivalently
iNKT Ship1
1  1.34   +/+
2  1.19   +/+
9  1.02   +/+
12 1.17   +/+
13 1.12   +/+
```

```
14 1.61   +/+
15 1.18   +/-
18 1.08   +/-
19 1.12   +/-
21 1.29   +/-
22 1.16   +/-
27 1.11   +/-
28 1.08   +/-
30 1.23   +/-
35 1.13   -/-
41 1.44   -/-
#Ask for observations with Ship1 status +/+
#and iNKT is greater than 1
#leaving the second index blank returns all columns
> r[which(r$iNKT>1 & r$Ship1=="+/+"),] #subset(x=r,subset=r$iNKT>1 & r$Ship1=="+/+")
iNKT Ship1
1  1.34   +/+
2  1.19   +/+
9  1.02   +/+
12 1.17   +/+
13 1.12   +/+
14 1.61   +/+
```

Going forward, we will use data frames to store our data in. We will ask questions about data frames just as we asked questions about vectors like we did in Examples 1.20, 1.12, and 1.18. Each column of a data frame is treated as a vector in R so all of the functionality discussed for vectors can be applied to the columns of a data frame.

2.1.3 Loading Data as a Data Frame

In statistics we generally have a lot of information to work with. As we explore political polls and research studies throughout the text, like upcoming Example 2.3, we will want to access the information collected. To save time, and our sanity, we will use a shortcut in R that loads, reads and saves the data for us – we just have to tell R where to look.

Very often in the text, we will refer to datasets posted online at https://cipolli.com. Instead of manually entering data, R will visit the website, download the data of interest and save it to whatever object you'd like. We like to play squash with all the free time we gain, you can do whatever you like.

Though we will load data from the internet or use data available in the program files of R you can load data directly from your computer by entering the file path of your data; it is easiest to use a comma or tab separated file but you can consult the vast R documentation for other file types.

Example 2.3. Anderson et al. (2015) explore the frequencies of invariant natural killer T (iNKT) cell populations in the spleen from mice with three types SHIP1 deletion. These researchers hypothesized that SHIP1 deletion would have major effects on iNKT cell development by altering the thresholds for positive and negative selection through changing the status of SHIP1, a part of the immune cell regulatory system.

Research Question: Does altering SHIP1 status in mice affect iNKT cell populations? This is

important as iNKT cells are able to respond rapidly to danger signals in a cell. They are known to play a role in chronic inflammatory diseases like autoimmune disease, asthma and metabolic syndrome. In human autoimmune diseases, their numbers are decreased in peripheral blood.

This research study looked at 48 mice and measured several items including SHIP1 status and frequency of iNKT cell populations. Even just two observations for 48 mice makes 96 data points – we typed all of that into R in Example 2.1. What if there were 100, 1000, or more mice?

```
#Reads in data from the internet
#file gives the location of the file
#sep tells us how each observation is separated.
#header indicates that there are column names to the data
> ship1dat<-read.table(file = "https://cipolli.com/students/data/ship1.txt",
+         sep=",",header=TRUE)
```

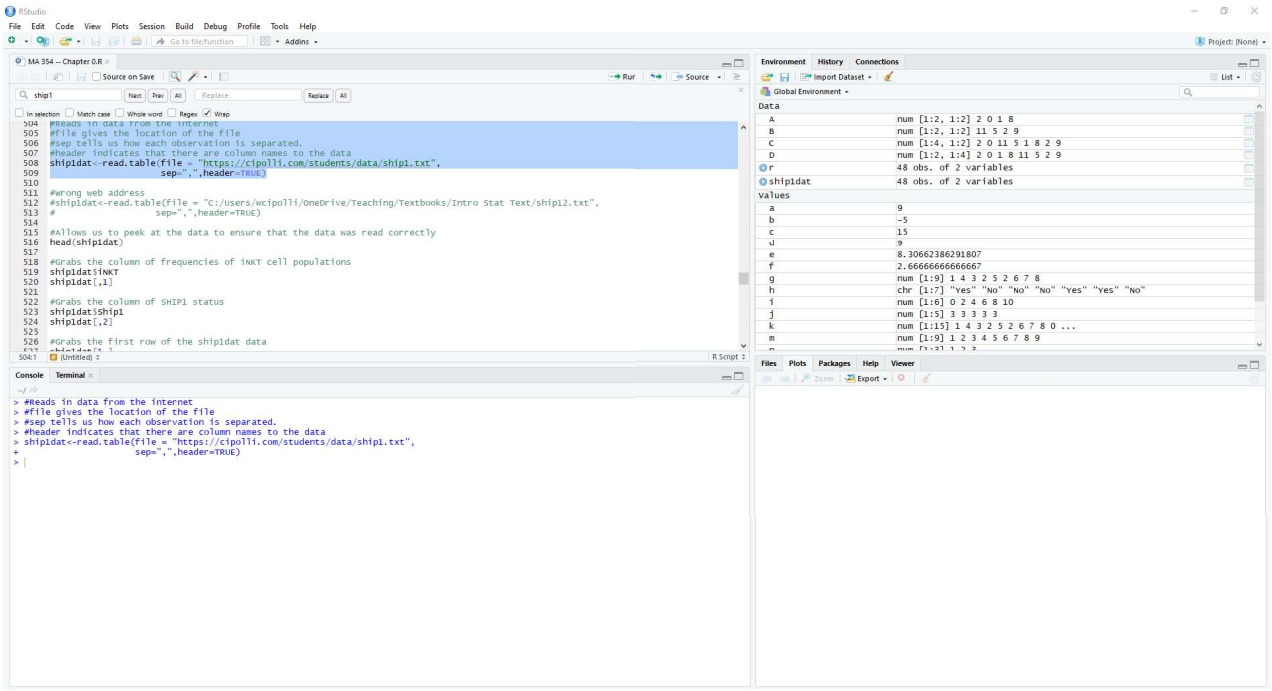

Figure 2.1.1: RStudio – What the ship1.txt data set file looks like. Note that ship1dat is now defined in the environment portion of RStudio.

The `read.table()` function requires us to specify some arguments. Below, we describe the ones used so far, but a more detailed description can be found by running `?read.table()`.

read.table(file=fn,	where fn is the location of the file we want to read
sep=delim,	where delim is the delimiter of the file "," for .csv or "\t" for .tsv
header=TRUE,)	does the first row consist of headers?

If you had a similar data set saved to your computer locally, you could load it directly from there. Assuming you have the file on your desktop and your log on user name is simply username the file location would be as follows.

- **Windows:** "C:/Users/username/Desktop/ship1.txt"
- **Mac OSX:** "/Users/username/Desktop/ship1.txt"

Remark: Above, we've loaded a .txt file, but the code is the same for .csv files, besides the filename change. While there are packages in R to read Microsoft Excel files (Wickham and Bryan, 2019), as well as files from other statistical software such as SPSS, SAS, Stata, Minitab, etc (R Core Team, 2018), we will stick to the plain text format.

Example 2.4. Spotting an Error in R. Let's consider loading the data from of Anderson et al. (2015), but this time there's a mistake.

```
> ship1dat<-read.table(file = "https://cipolli.com/students/data/ship12.txt",
+        sep=",",header=TRUE)
Error in file(file, "rt") : cannot open the connection
In addition: Warning message:
In file(file, "rt") :
cannot open URL        'https://cipolli.com/students/data/ship12.txt':
HTTP status was '404 Not Found'
```

We've run the code but R gives us an error. Here, we see "HTTP status was '404 Not Found'" as part of the error; this indicates that we've supplied the wrong web address – there was a response from the server but the address was not found. If you're loading a file stored on your computer the error is very similar but will read "cannot open file." When we receive an error like this it is important to check the file's location and ensure that we told R to look in the right place.

Example 2.5. Spotting an Error in R. Let's consider loading the data from of Anderson et al. (2015), but this time there's a different mistake.

```
> ship1dat<-read.table(file = "https://cipolli.com/students/data/ship1.txt",
+        sep=",",header=TRUE)
Error in file(file, "rt") : cannot open the connection
In addition: Warning message:
In file(file, "rt") :
InternetOpenUrl failed: 'The server name or address could not be resolved'
```

We've run the code but R gives us an error. Here, we see "InternetOpenUrl failed" as part of the error; this indicates that R wasn't able to check if the file was located in the location we specified. When we receive an error like this it is important to check that we are connected to the internet and that R has permission to use it.

Once we've loaded the data successfully, we can ensure that it has loaded correctly by taking a peek at the top of the data table.

```
#Allows us to peek at the data to ensure that the data was read correctly
> head(ship1dat)
iNKT Ship1
1 1.34   +/+
2 1.19   +/+
```

```
3 0.89   +/+
4 0.88   +/+
5 0.98   +/+
6 0.85   +/+
```

In this case, R returns the first six rows of data; we can see row numbers printed at the beginning of each line followed by the frequencies of iNKT cell populations and SHIP1 deletion status. We can subset the data as we did in Example 2.2 by asking for columns, or rows as follows.

```
#Grabs the column of frequencies of iNKT cell populations
> ship1dat$iNKT
[1] 1.34 1.19 0.89 0.88 0.98 0.85 0.93 0.87 1.02 0.96 0.77 1.17 1.12 1.61 1.18
[16] 1.00 0.88 1.08 1.12 0.69 1.29 1.16 0.94 0.94 0.58 0.87 1.11 1.08 0.92 1.23
[31] 0.94 0.41 0.57 0.19 1.13 0.33 0.60 0.16 0.33 0.77 1.44 0.62 0.70 0.77 0.47
[46] 0.44 0.28 0.22
> ship1dat[,1]
[1] 1.34 1.19 0.89 0.88 0.98 0.85 0.93 0.87 1.02 0.96 0.77 1.17 1.12 1.61 1.18
[16] 1.00 0.88 1.08 1.12 0.69 1.29 1.16 0.94 0.94 0.58 0.87 1.11 1.08 0.92 1.23
[31] 0.94 0.41 0.57 0.19 1.13 0.33 0.60 0.16 0.33 0.77 1.44 0.62 0.70 0.77 0.47
[46] 0.44 0.28 0.22
#Grabs the column of SHIP1 status
> ship1dat$Ship1
[1] +/+ +/+ +/+ +/+ +/+ +/+ +/+ +/+ +/+ +/+ +/+ +/+ +/+ +/+ +/+ +/- +/- +/- +/- +/-
[20] +/- +/- +/- +/- +/- +/- +/- +/- +/- +/- +/- +/- +/- -/- -/- -/- -/- -/- -/-
[39] -/- -/- -/- -/- -/- -/- -/- -/- -/- -/-
Levels: -/- +/- +/+
> ship1dat[,2]
[1] +/+ +/+ +/+ +/+ +/+ +/+ +/+ +/+ +/+ +/+ +/+ +/+ +/+ +/+ +/+ +/- +/- +/- +/- +/-
[20] +/- +/- +/- +/- +/- +/- +/- +/- +/- +/- +/- +/- +/- -/- -/- -/- -/- -/- -/-
[39] -/- -/- -/- -/- -/- -/- -/- -/- -/- -/-
Levels: -/- +/- +/+
#Grabs the first row of the ship1dat data
> ship1dat[1,]
iNKT Ship1
1 1.34   +/+
#Grabs the second row of the ship1dat data
> ship1dat[2,]
iNKT Ship1
2 1.19   +/+
#Grabs the three row of the ship1dat data
> ship1dat[c(1,2,3),]
iNKT Ship1
1 1.34   +/+
2 1.19   +/+
3 0.89   +/+
```

Example 2.6. Spotting an Error in R. Let's consider asking for a column of data in Example 2.3, but this time there's a mistake.

```
#Doesn't return data -- column name is iNKT, not inkt
```

```
> ship1dat$inkt
NULL
```

We've run the code but R doesn't return the data we thought we asked for; here, it just returns NULL. This indicates that there was no column where we told R to look. To fix this we have to ensure that we were telling R to look in the right place; in this case the column we actually wanted was "iNKT," not "inkt;" capitalization matters.

Computers speak a very precise language and we must ask questions in that language – including precise spelling and capitalization. As a human, we might see "inkt" and intuitively figure out that we meant "iNKT" but R *always* answers the exact question we asked.

Example 2.7. Spotting an Error in R. Let's consider asking for a column of data for observations with a specified SHIP1 status in Example 2.3, but this time there's a mistake.

```
#Doesn't return data -- there is not SHIP1 status of "-/+"
> ship1dat$iNKT[which(ship1dat$Ship1=="-/+")]
numeric(0)
```

We've run the code but R doesn't return the data we thought we asked for; here, it just returns numeric(0). This indicates that there were zero observations for which SHIP1 status was "-/+." To fix this we have to ensure that we were telling R to look for the right thing; in this case, it's possible zero observations is the answer we were looking for or that we meant to specify SHIP1 status of "+/-." This shows the importance of ensuring the answer we receive makes sense for the question we asked. Ensuring we're asking the right question is a theme of effective coding.

2.1.4 Writing Data to file

Example 2.8. Writing data to file is largely the same as reading it. For example, if I wanted to print the data from Example 2.3 to a .csv file, instead of a .txt file, I would run the following R code, which saves the file to the desktop.

```
> ship1dat<-write.table(x=ship1dat,file="C:/Users/username/Desktop/ship1.csv",
+                       sep=",",col.names=TRUE) #col.names=TRUE prints column names
```

Remark: If you wanted to do this you would have to reconfigure the file path to the location you wanted to save the file.

2.1.5 Lists

We saw that data frames are stored as a list of equal-length vectors in the previous subsection. Lists can be an important tool for working with complex data sets. For example, we look at data in Example 2.9 where each observation of a certain variable can be of different dimensions.

Example 2.9. Assigning Lists to Objects in R. Suppose we wanted to collect data on professors teaching statistics. We might want to collect their names, subject(s) they hold a degree in, their overall course ratings, their overall difficulty ratings, and the common comments made on their courses. Because the professors teaching statistics can have a different number of subject(s) they

hold a degree in or different number of common comments made on their courses a data frame would be difficult to use. Instead, we create a list.

```
> stats.professors<-list(names=c("Will","Roy","Josh"),
+                        degrees=list(c("Math","Stat","Computer Science"),
+                                     c("Math","Stat","Bio Stat","Applied Math"),
+                                     c("Math","Stat")),
+                        ratings=c(4.6,4.8,3.7),
+                        difficulty=c(3.6,2.7,3.2),
+                        tags=list(c("Accessible","Inspirational","Good Feedback"),
+                                  c("Accessible","Hilarious","Don't Skip","Good Feedback"),
+                                  c("Tough Grader","Don't Skip","Test Heavy",
+                                    "Graded by Few Things","Accessible",
+                                    "Lecture Heavy","Clear Grading Criteria"))
+                    )
> length(stats.professors)
[1] 5
```

Example 2.10. Asking For Subsets of Lists. Recall that we use double the brackets to access elements of a list. For example, below we ask for the attributes of the professors we have data for.

```
#Ask for the professors' names
> stats.professors$names
[1] "Will" "Roy"  "Josh"
> stats.professors[["names"]]
[1] "Will" "Roy"  "Josh"
> stats.professors[[1]]
[1] "Will" "Roy"  "Josh"
#Ask for the data about Will -- the first entries
> stats.professors[["degrees"]][[1]]
[1] "Math"             "Stat"             "Computer Science"
> stats.professors[["ratings"]][1]
[1] 4.6
> stats.professors[["difficulty"]][1]
[1] 3.6
> stats.professors[["tags"]][[1]]
[1] "Accessible"    "Inspirational" "Good Feedback"
```

Since lists are made up of other objects we've talked about, we already know how to ask questions about the elements of a list.

Example 2.11. Spotting an Error in R. The double brackets are important when working with lists. If we aren't careful in asking for the data we want, we won't get what we're looking for; this is particularly problematic when it is part of a larger code file, which would cause downstream consequences.

For example, suppose we wanted to retrieve the name of the first professor for whom we have data. If we use single brackets returns the list of names,

```
> stats.professors["names"]
$names
```

```
[1] "Will" "Roy"  "Josh"
> stats.professors["names"][1]
$names
[1] "Will" "Roy"  "Josh"
```

where we actually wanted the following.

```
> stats.professors[["names"]][1]
[1] "Will"
```

Example 2.12. We will sometimes find it useful to have the `unlist()` function available, which takes a list as input and return a vectorized version. Below, we apply the `unlist()` function to the data from Example 2.9; we set `recursive=FALSE` to indicate that we don't want to unlist the elements of the list as well.

```
> unlist(stats.professors,recursive=FALSE)
$names1
[1] "Will"

$names2
[1] "Roy"

$names3
[1] "Josh"

$degrees1
[1] "Math"             "Stat"             "Computer Science"

$degrees2
[1] "Math"         "Stat"         "Bio Stat"     "Applied Math"

$degrees3
[1] "Math" "Stat"

$ratings1
[1] 4.6

$ratings2
[1] 4.8

$ratings3
[1] 3.7

$difficulty1
[1] 3.6

$difficulty2
[1] 2.7

$difficulty3
```

```
[1] 3.2

$tags1
[1] "Accessible"    "Inspirational" "Good Feedback"

$tags2
[1] "Accessible"    "Hilarious"     "Don't Skip"    "Good Feedback"

$tags3
[1] "Tough Grader"           "Don't Skip"             "Test Heavy"
[4] "Graded by Few Things"   "Accessible"             "Lecture Heavy"
[7] "Clear Grading Criteria"
```

We see that each variable name is present three times, once for each professor. Because we set `recursive=FALSE`, the vectors of information corresponding to a professor are kept together. If we had applied the default version of `unlist()`, where `recursive=TRUE`, `unlist()` would have been applied to those vectors of information as well – we demonstrate this below.

```
> unlist(stats.professors)
names1                   names2                   names3
"Will"                    "Roy"                   "Josh"
degrees1                 degrees2                 degrees3
"Math"                   "Stat"       "Computer Science"
degrees4                 degrees5                 degrees6
"Math"                   "Stat"              "Bio Stat"
degrees7                 degrees8                 degrees9
"Applied Math"                  "Math"                   "Stat"
ratings1                 ratings2                 ratings3
"4.6"                    "4.8"                    "3.7"
difficulty1              difficulty2              difficulty3
"3.6"                    "2.7"                    "3.2"
tags1                    tags2                    tags3
"Accessible"          "Inspirational"          "Good Feedback"
tags4                    tags5                    tags6
"Accessible"              "Hilarious"             "Don't Skip"
tags7                    tags8                    tags9
"Good Feedback"           "Tough Grader"             "Don't Skip"
tags10                   tags11                   tags12
"Test Heavy"   "Graded by Few Things"             "Accessible"
tags13                   tags14
"Lecture Heavy" "Clear Grading Criteria"
```

We see that the information is now collapsed completely and the structure by professor is completely lost. This use of `unlist()` is undesirable for this data.

2.1.6 Data Frame Data Types

Recall Tables 1.3.2 and 1.3.3, where we described the data types and classes of objects in R. We explored the four types of vectors in R, the same as objects – character, numeric, logical, and

complex. Unlike vectors, data frames can consist of different data types – we can think of data frames as a list of equal-length vectors.

When we discuss data types with data frames, we will ask questions about the data type of each column. For example, we can explore the data types of the data frame from Example 2.3.

```
#See the classes of individual objects
> c(class(ship1dat),mode(ship1dat),typeof(ship1dat))
[1] "data.frame" "list" "list"
> c(class(ship1dat$iNKT),mode(ship1dat$iNKT),typeof(ship1dat$iNKT))
[1] "numeric" "numeric" "double"
> c(class(ship1dat$Ship1),mode(ship1dat$Ship1),typeof(ship1dat$Ship1))
[1] "factor"  "numeric" "integer"
#See a summary of the data frame we created
> str(ship1dat)
'data.frame':        48 obs. of  2 objects:
$ iNKT : num  1.34 1.19 0.89 0.88 0.98 0.85 0.93 0.87 1.02 0.96 ...
$ Ship1: Factor w/ 3 levels "-/-","+/-","+/+": 3 3 3 3 3 3 3 3 3 3 ...
```

The frequencies of invariant natural killer T (iNKT) cell populations is treated as numeric data and three types SHIP1 deletion is treated as a factor, but stored numerically. We can use the `as.numeric()` function to observe the corresponding numeric data for factors. For example, we can see the numeric version of the SHIP1 deletion types below; each integer corresponds to a specific level.

```
> ship1dat$Ship1
[1] +/+ +/+ +/+ +/+ +/+ +/+ +/+ +/+ +/+ +/+ +/+ +/+ +/+ +/+ +/- +/- +/- +/- +/-
[20] +/- +/- +/- +/- +/- +/- +/- +/- +/- +/- +/- +/- +/- -/- -/- -/- -/- -/- -/-
[39] -/- -/- -/- -/- -/- -/- -/- -/- -/- -/-
Levels: -/- +/- +/+
> as.numeric(ship1dat$Ship1)
[1]  3 3 3 3 3 3 3 3 3 3 3 3 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1
[40] 1 1 1 1 1 1 1 1 1
```

We see that `R` assigns numbers in the order of the levels – the different values the object could take on. Here, "−/−" is represented by 1, "+/−" is represented by 2,"+/+" is represented by 3. If we wanted, we could reassign the order of the levels, which will be helpful when we start completing statistical analyses on factors.

```
> ship1dat$Ship1.refactored<-factor(ship1dat$Ship1,levels = c("+/+","-/-","+/-"))
> ship1dat$Ship1.refactored
[1] +/+ +/+ +/+ +/+ +/+ +/+ +/+ +/+ +/+ +/+ +/+ +/+ +/+ +/+ +/- +/- +/- +/- +/-
[20] +/- +/- +/- +/- +/- +/- +/- +/- +/- +/- +/- +/- +/- -/- -/- -/- -/- -/- -/-
[39] -/- -/- -/- -/- -/- -/- -/- -/- -/- -/-
Levels: +/+ -/- +/-
> as.numeric(ship1dat$Ship1.refactored)
[1]  1 1 1 1 1 1 1 1 1 1 1 1 1 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 2 2 2 2 2 2 2
[40] 2 2 2 2 2 2 2 2 2
```

Both the character-valued levels and the integer codes for each level are stored in R. There will be little difference when completing statistical analyses between characters and factors, as R will convert categorical objects to factors automatically. There are, however, times when it will be necessary to separate the character-valued levels and integer codes; e.g., when we are naming files, specifying text elements in plots, etc. We can convert factors to characters using `as.character()` for such purposes.

There are examples where factors can be problematic. One example we've come across is "stale" factors sticking around after we take a subset of data – this is common when one factor is removed while cleaning data.

Example 2.13. Suppose we only wanted to run analysis on mice with "+/+" Ship1 deletion status from Example 2.3.

```
> ship1dat.plusplus<-ship1dat[which(ship1dat$Ship1=="+/+"),]
> ship1dat.plusplus$Ship1
[1] +/+ +/+ +/+ +/+ +/+ +/+ +/+ +/+ +/+ +/+ +/+ +/+ +/+ +/+
Levels: -/- +/- +/+
```

When we subset the data the original factors are still present; this would not be an issue if we treated the Ship1 column as a character vector instead of a factor vector. Doing this yields a character vector – notice that no levels are specified.

```
> ship1dat$Ship1<-as.character(ship1dat$Ship1)
> ship1dat[which(ship1dat$Ship1=="+/+"),]$Ship1
[1] "+/+" "+/+" "+/+" "+/+" "+/+" "+/+" "+/+" "+/+" "+/+" "+/+" "+/+" "+/+"
[14] "+/+"
```

We can also simply drop the unused levels using the `droplevels()` function. Doing this yields a factor vector with one level.

```
> ship1dat.plusplus$Ship1<-droplevels(ship1dat.plusplus$Ship1)
> ship1dat.plusplus$Ship1
[1] +/+ +/+ +/+ +/+ +/+ +/+ +/+ +/+ +/+ +/+ +/+ +/+ +/+ +/+
Levels: +/+
```

Example 2.14. We can also ask questions about data types for lists containing more complex data. We demonstrate this below with the data from Example 2.9.

```
> x <- stats.professors # recall stats.professors is a list of data
> c(class(x),mode(x),typeof(x))
[1] "list" "list" "list"
> x <- stats.professors[1]
> c(class(x),mode(x),typeof(x))
[1] "list" "list" "list"
> x <- stats.professors[[1]]
> c(class(x),mode(x),typeof(x))
[1] "character" "character" "character"
```

Again, we see that the use of double brackets provides different output than single brackets which demonstrates the importance of how we ask for data from a list.

2.1.7 Summary of R Commands

Table 2.1.1 summarizes the operators used in this section.

Operator	Functionality	Example From Notes
data.frame(x=c(...),y=c(...))	creates a data frame containing columns x and y with the specified data	See Example 2.1
head(x)	returns a preview of the top rows of x	head(r)
tail(x)	returns a preview of the bottom rows of x	tail(r)
nrow(x)	returns the number of rows of x	nrow(r)
ncol(x)	returns the number of columns of x	ncol(r)
names(x)	returns the names of an object x	names(r)
colnames(x)	returns the column names of a data frame x	colnames(r)
rownames(x)	returns the row names of a data frame x	rownames(r)
summary(x)	returns a summary of the data by column x	summary(r)
x\$$y$	returns the column labeled y of x	r$iNKT
read.table(x)	reads data from a file to an object in R	See Example 2.3
write.table(x)	writes data from an object in R to file	See Example 2.8
list(...)	creates a list	Example 2.9
unlist(...)	simplifies a list to a vector	Example 2.12

Table 2.1.1: A list of basic operators for data frames in R.

2.2 Matrices

2.2.1 Creating Matrices

Understanding statistics, beyond running base code, requires an understanding of the linear algebra mechanisms, which are used to get the results. Our ability to choose the appropriate analysis technique, interpret the results correctly, and evaluate errors properly requires an understanding of the underlying algorithm for computing the statistics of interest. First, we will discuss how to complete basic linear algebra tasks in R.

Example 2.15. Assigning Matrices to Objects in R. Let's create the following 2×2 matrix in R.

$$A = \begin{bmatrix} 2 & 1 \\ 0 & 8 \end{bmatrix} \qquad B = \begin{bmatrix} 11 & 2 \\ 5 & 9 \end{bmatrix}$$

```
#Matrices are created by columns by default
> A<-matrix(data=c(2,0,1,8),nrow=2,ncol=2)
```

```
> A
[,1] [,2]
[1,]    2    1
[2,]    0    8
#We can ask R to build Matrices by row
> A<-matrix(data=c(2,1,0,8),nrow=2,ncol=2,byrow=TRUE)
> A
[,1] [,2]
[1,]    2    1
[2,]    0    8
#Matrices are created by columns by default
> B<-matrix(data=c(11,5,2,9),nrow=2,ncol=2)
> B
[,1] [,2]
[1,]   11    2
[2,]    5    9
#We can ask R to build matrices by row
> B<-matrix(data=c(11,2,5,9),nrow=2,ncol=2,byrow=TRUE)
> B
[,1] [,2]
[1,]   11    2
[2,]    5    9
#What is the number of entries in the matrix?
> length(A)
[1] 4
#Summarize the columns of the matrix
> summary(A)
V1            V2
Min.   :0.0   Min.   :1.00
1st Qu.:0.5   1st Qu.:2.75
Median :1.0   Median :4.50
Mean   :1.0   Mean   :4.50
3rd Qu.:1.5   3rd Qu.:6.25
Max.   :2.0   Max.   :8.00
#What is the number of rows in the matrix?
> nrow(A)
[1] 2
#What is the number of columns in the matrix?
> ncol(A)
[1] 2
#What are the dimensions of the matrix?
> dim(A) #Gives the number of rows and columns
[1] 2 2
#Previewing a matrix
> head(A)  #Shows full matrix because it is small
[,1] [,2]
[1,]    2    1
[2,]    0    8
> tail(A)  #Shows full matrix because it is small
[,1] [,2]
```

```
[1,]    2    1
[2,]    0    8
#What are the diagnoal entries of A?
> diag(A)
[1] 2 8
#What is the transpose of A?
> t(A)
[,1] [,2]
[1,]    2    0
[2,]    1    8
```

Example 2.16. Spotting an Error in R. As you'll see, we generally prefer to enter matrix data as we read a book – left to right, top to bottom. It is important that we carefully specify matrices when we use them. Below, we have all of the pieces to specify the matrix A we would like to assign to an object `A` in `R` but we haven't told `R` to orient it correctly.

```
> A<-matrix(data=c(2,1,0,8),nrow=2,ncol=2)
> A
[,1] [,2]
[1,]    2    0
[2,]    1    8
```

This example shows how important specifying `byrow=TRUE` is when, in fact, we provide the data by row. If we didn't check, we would not see that the matrix was not assigned correctly and any results that followed would be incorrect.

2.2.2 Calculations with Matrices

Example 2.17. Describing Matrices in R. There are many built-in functions in `R` that are helpful for vectors (see Example 1.18) and can be applied to matrices. We will not do this often, but we may find it useful later.

```
#Treats like a vector of size length(A)
> as.vector(A)
[1] 2 1 0 8
#What is the minimum element in g?
> min(A)
[1] 0
#What is the maximum element in g?
> max(A)
[1] 8
#What is the sum of all elements in g?
> sum(A)
[1] 11
#Calculate a cumulative or "running" total.
> cumsum(A)
[1]  2  3  3 11
#What is the average element in g?
> mean(A)
```

```
[1] 2.75
#What is the standard deviation of elements in g?
> sd(A)
[1] 3.593976
#What are the quantiles of elements in g?
> quantile(A)
0%  25%  50%  75% 100%
0.00 0.75 1.50 3.50 8.00
#What is the 90th percentile of elements in g?
> quantile(A,probs=c(0.90))
90%
6.2
```

Example 2.18. Performing Matrix Algebra in R. Let's perform several operations with the matrices created in Example 2.15 in R. Using the basic calculator operators from Example 0.2, applies the operation to each element in the matrix.

```
> A+2
[,1] [,2]
[1,]    4    3
[2,]    2   10
> A-2
[,1] [,2]
[1,]    0   -1
[2,]   -2    6
> A*2
[,1] [,2]
[1,]    4    2
[2,]    0   16
> A/2
[,1] [,2]
[1,]    1  0.5
[2,]    0  4.0
```

For clarity, we complete these calculations "by hand" below.

$$A+2=\begin{bmatrix}2+2 & 1+2\\ 0+2 & 8+2\end{bmatrix}=\begin{bmatrix}4 & 3\\ 2 & 10\end{bmatrix}$$

$$A-2=\begin{bmatrix}2-2 & 1-2\\ 0-2 & 8-2\end{bmatrix}=\begin{bmatrix}0 & -1\\ -2 & 6\end{bmatrix}$$

$$A*2=\begin{bmatrix}2\times 2 & 1\times 2\\ 0\times 2 & 8\times 2\end{bmatrix}=\begin{bmatrix}4 & 2\\ 0 & 16\end{bmatrix}$$

$$A/2=\begin{bmatrix}2/2 & 1/2\\ 0/2 & 8/2\end{bmatrix}=\begin{bmatrix}1 & 0.50\\ 0 & 4\end{bmatrix}$$

When the calculations are done with matrices of the same size, the calculations are done element-wise.

```
> A+B
[,1] [,2]
```

```
[1,]   13    3
[2,]    5   17
> A-B
[,1] [,2]
[1,]   -9   -1
[2,]   -5   -1
> A*B
[,1] [,2]
[1,]   22    2
[2,]    0   72
> A/B
[,1]      [,2]
[1,] 0.1818182 0.5000000
[2,] 0.0000000 0.8888889
```

For clarity, we complete these calculations "by hand" below.

$$A + B = = \begin{bmatrix} 2+11 & 1+2 \\ 0+5 & 8+9 \end{bmatrix} = \begin{bmatrix} 13 & 3 \\ 5 & 17 \end{bmatrix}$$

$$A - B = \begin{bmatrix} 2-11 & 1-2 \\ 0-5 & 8-9 \end{bmatrix} = \begin{bmatrix} -9 & -1 \\ -5 & -1 \end{bmatrix}$$

$$A * B = \begin{bmatrix} 2 \times 11 & 1 \times 2 \\ 0 \times 5 & 8 \times 9 \end{bmatrix} = \begin{bmatrix} 22 & 2 \\ 0 & 72 \end{bmatrix}$$

$$A/B = \begin{bmatrix} 2/11 & 1/2 \\ 0/5 & 8/9 \end{bmatrix} = \begin{bmatrix} 0.\bar{18} & 0.50 \\ 0 & 0.\bar{88} \end{bmatrix}$$

We can also complete matrix algebra.

```
#Matrix multiplication of A and B
> A %*% B
[,1] [,2]
[1,]   27   13
[2,]   40   72
#What is the determinant of A?
> det(A)
[1] 16
#What is the inverse of A?
> solve(A)
[,1]    [,2]
[1,]  0.5 -0.0625
[2,]  0.0  0.1250
#What are the eigen values and vectors for A?
> eigen(A)
eigen() decomposition
$values
[1] 8 2

$vectors
[,1] [,2]
```

```
[1,] 0.1643990    1
[2,] 0.9863939    0
```

The matrix multiplication is carried out "by hand" as follows.

$$A \times B = \begin{bmatrix} 2 & 1 \\ 0 & 8 \end{bmatrix} \times \begin{bmatrix} 11 & 2 \\ 5 & 9 \end{bmatrix} = \begin{bmatrix} 2(11)+1(5) & 2(2)+1(9) \\ 0(11)+8(5) & 0(2)+8(9) \end{bmatrix} = \begin{bmatrix} 27 & 13 \\ 40 & 72 \end{bmatrix}$$

The determinant of A is calculated as follows.

$$\det(A) = \det \begin{bmatrix} 2 & 1 \\ 0 & 8 \end{bmatrix} = 2 \times 8 - 1 \times 0 = 16$$

The inverse of A is calculated as follows using the technique for 2×2 matrices, which makes use of the determinant and adjugate of A.

$$A^{-1} = \frac{1}{\det(A)} \text{ adjugate} \left(\begin{bmatrix} 2 & 1 \\ 0 & 8 \end{bmatrix} \right) = \frac{1}{16} \begin{bmatrix} 8 & -1 \\ 0 & 2 \end{bmatrix} = \begin{bmatrix} 8/16 & -1/16 \\ 0/16 & 2/16 \end{bmatrix} = \begin{bmatrix} 0.5 & -0.0625 \\ 0 & 0.125 \end{bmatrix}$$

To find the eigenvalues of A we solve the following equation,

$$\begin{aligned} \det \left(\begin{bmatrix} 2 & 1 \\ 0 & 8 \end{bmatrix} - \begin{bmatrix} \lambda & 0 \\ 0 & \lambda \end{bmatrix} \right) &= 0 \\ \det \left(\begin{bmatrix} 2-\lambda & 1 \\ 0 & 8-\lambda \end{bmatrix} \right) &= 0 \\ (2-\lambda)(8-\lambda) - 0 &= 0 \\ \implies \lambda_1 = 2, \lambda_2 &= 8 \end{aligned}$$

We solve for V_1, the eigenvector associated with $\lambda_1 = 2$, as follows.

$$\begin{aligned} AV_1 = (2)V_1 \implies AV_1 - (2)V_1 &= \begin{bmatrix} 0 \\ 0 \end{bmatrix} \\ (A-2)V_1 &= \begin{bmatrix} 0 \\ 0 \end{bmatrix} \\ \begin{bmatrix} 2-(2) & 1 \\ 0 & 8-(2) \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} &= \begin{bmatrix} 0 \\ 0 \end{bmatrix} \\ \begin{bmatrix} v_2 \\ 6v_2 \end{bmatrix} &= \begin{bmatrix} 0 \\ 0 \end{bmatrix} \\ \implies v_2 = 0, v_1 &\in \mathbb{R} \end{aligned}$$

We solve for V_2, the eigenvector associated with $\lambda_1 = 8$, as follows.

$$\begin{aligned} AV_2 = (8)V_2 \implies AV_2 - (8)V_2 &= \begin{bmatrix} 0 \\ 0 \end{bmatrix} \\ (A-8)V_2 &= \begin{bmatrix} 0 \\ 0 \end{bmatrix} \\ \begin{bmatrix} 2-(8) & 1 \\ 0 & 8-(8) \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} &= \begin{bmatrix} 0 \\ 0 \end{bmatrix} \\ \begin{bmatrix} -6v_1 + v_2 \\ 0 \end{bmatrix} &= \begin{bmatrix} 0 \\ 0 \end{bmatrix} \\ \implies v_2 = 6v_1 & \end{aligned}$$

Note that we get far more information completing this task by hand. Specifically, we see that the eigenvector associated with $\lambda_1 = 2$ can be any vector of the form $V_1 = (v_1, 0)$ where $v_1 \in \mathbb{R}$ and that the eigenvector associated with $\lambda_2 = 8$ can be any vector of the form $V_2 = (v_1, 6v_1)$ where $v_1 \in \mathbb{R}$. R simply supplies one set of eigenvalues that satisfy this condition.

Example 2.19. Combining Matrices in R. We can also combine the matrices in a couple of ways, as demonstrated below.

```
#Join matrices at the rows
> C<-rbind(A,B)
> C
[,1] [,2]
[1,]    2    1
[2,]    0    8
[3,]   11    2
[4,]    5    9
#Join matrices at the columns
> D<-cbind(A,B)
> D
[,1] [,2] [,3] [,4]
[1,]    2    1   11    2
[2,]    0    8    5    9
```

Example 2.20. Spotting an Error in R. When multiplying matrices in R, it will perform a check to ensure the number of columns of the left matrix and the number of rows in the right matrix match. If this condition is not met, R will throw an error about non-conformable arguments and will not carry out matrix multiplication. If this condition is met, R will perform the matrix multiplication as expected.

```
> A%*%C
Error in A %*% C : non-conformable arguments
> A%*%D
[,1] [,2] [,3] [,4]
[1,]    4   10   27   13
[2,]    0   64   40   72
```

2.2.3 Summary of R Commands

Table 2.2.2 summarizes the operators used in this subsection.

Operator	Functionality	Example From Notes
matrix(data=c(...), nrow=r,ncol=c)	creates a matrix containing the data with r rows and c columns.	matrix(data=c(2,0,1,8), nrow=2,ncol=2)
length(x)	returns the number of entries in x	length(A)
summary(x)	returns a summary of the data by column x	summary(A)
nrow(x)	returns the number of rows of x	nrow(A)
ncol(x)	returns the number of columns of x	ncol(A)
head(x)	returns a preview of the top rows of x	head(A)
tail(x)	returns a preview of the bottom rows of x	tail(A)
diag(x)	returns the diagonal elements of x	diag(A)
t(x)	returns the transpose of x	t(A)
as.vector(x)	returns the vectorized version of x	as.vector(A)
min(x)	returns the minimum element in x	min(A)
max(x)	returns the maximum element in x	max(A)
sum(x)	returns the sum of all elements in x	sum(A)
cumsum(x)	returns the cumulative sum of x top to bottom, left to right	cumsum(A)
mean(x)	returns the average value of elements in x	mean(A)
sd(x)	returns the standard deviation of elements in x	sd(A)
quantile(x,probs=y)	returns the y-th percentile of elements in x	quantile(A,probs=c(0.90))
x %*% y	the result of the matrix multiplication of x and y	A %*% B
det(x)	returns the determinant of X	det(A)
solve(x)	returns the inverse matrix of x	solve(A)
eigen(x)	returns the eigen values and eigen vectors of x	eigen(A)
rbind(x,y)	joins x and y at the rows	rbind(A,B)
cbind(x,y)	joins x and y at the columnss	cbind(A,B)

Table 2.2.2: A list of basic operators for matrices in R.