Using R Volume I: Introduction and Univariate Statistical Analyses

Dev K. Dalal, PhD

2026-01-05

Table of contents

1 Introduction to the Series

Reasoning with data requires, among other things, producing and accurately interpreting the output of statistical analyses. That is, effectively using data to draw accurate conclusions - therein informing research and practice - requires knowing how to analyze your data, how to get your statistical software to run the appropriate analysis, and how to correctly interpret

the output. To this end, myriad books are available to learn how to model your data and how to produce and interpret your output. For instance, Tabachnick and Fidell (2021) present different ways to model multivariate data using SPSS. Likewise, Field, Miles, and Field (2012) similarly show readers how to understand and produce statistical analyses using *R*.

Given these different resources, it is reasonable to ask why this book series is even necessary. In some ways, it isn't–a reader of this book series can easily gather the information contained within these books across other resources. The main impetus for this series is not to replace these other books–indeed, these other sources provide rich detail that this book series intentionally does not. Instead, the goal of this book series is to provide readers with a set of resources to facilitate transitioning to using *R* to run their statistical analyses, and to collect these in one location–this book series.

To this end, this book series intentionally does not discuss issues of when to use which modeling approach. Rather, it focuses on helping readers already familiar with the theoretical aspects of data modeling use *R* (R Core Team, 2022) to model their data. You can think of it as the answer to the question: "How do I get *R* to run a [INSERT YOUR FAVORITE STATISTICAL ANALYSIS HERE][1]?;" you, the reader, already knows what you'd would like *R* to do, but need to know how to get *R* to do it.

Likewise, this book series intentionally excludes in depth discussions on interpreting output. Again, I assume you are familiar with how to interpret the output of "a [YOUR FAVORIATE STATISTICAL ANALYSIS HERE][2]." You just need to learn how to get *R* to run it. In this way, this book series is directed at readers who know what they want to run and how to understand the output, but not in *R*. If you, as a reader, do not count yourself as part of this group of readers (i.e., you are still unsure about how to model data, or how to interpret the output of data), you may want to consider a different resource for learning *R*.[3] For the others, this series should, hopefully, get you over the hurdle of using *R*.

This book series is three volumes. Volume I, *Using* R*: Introduction and Univariate Statistical Analyses,* presents readers with an introduction to *R* as a software and language, basic operations in *R*, and then how to use *R* to run simple univariate statistical analyses (e.g., central tendency, variability, *t*-tests, etc.). Volume II, *Using* R*: Multivariate Statistical Analyses,* shows readers how to use *R* to run different multivariate and advance data modeling methods (e.g., moderated regression analysis, structural equation models, multilevel models, etc.). Volume III, *Using* R*: Psychometrics Analyses*, presents readers with information on how

[1] You have a favorite, right? If you don't, you should. Which analysis gets your most excited to think about? That's your favorite! Figure that out, and get it tattooed on your body! (Maybe not that, unless tattoos are your thing. In that case, have at it! Certainly a lower back tattoo of the General Linear Model would be very popular, no?)

[2] Look, if you have a favorite statistical analysis and you don't know how to interpret it, what are we doing here? Why would you tattoo something on your body that you don't know how to read!?!?

[3] For these readers, I recommend the book by Field and colleagues (2012). It is the book I used to teach myself how to use *R*. Also, yes, not every footnote in this book series is a bad joke, most of them actually provide important information. Whether or not a footnote has important information can't be known until you read it–therefore, you have to read all of my bad jokes. You're welcome, and I'm sorry.

to use R to run different psychometrics models (e.g., Cronbach's α, Generalizability theory, Item Response Theory, etc.).[4]

One last note before transition to this specifics volume: Practicing using R is of paramount importance. Like any skill, your facility with R will improve the more you use it and practice (yes, we're talkin' 'bout practice!). It's important to stick to using R even when it gets painful. But, as Tony "Duke" Evers told Rocky Balboa before the start of the third round of his fight with Ivan Drago: NO PAIN!

2 Introduction to this Volume

My partner, a developmental psychologist and gifted scholar, once mentioned to me that getting started in R is hard because it's not clear how to do some of the most basic things that SPSS makes very easy with the point-and-click interface. Indeed, I struggled mightily with initially using R because I couldn't even figure out how to get my data set into R, let alone figure out how to manipulate and analyze it.

Volume I of this book series addresses a lot of these initial hurdles that people face when they want to switch from an existing software (e.g., SAS) to R. The steps and operations outlined in this document represent some of the initial issues that researchers have faced when switching to R (admittedly, based on my experiences and conversations). This includes things like understanding R is both a software *and* language. Being that it's a language, like all languages, it has its own rules that have to be followed. In addition, we'll cover how to install R, and how to work with R as an object based program.

Following this initial information, we will cover how to load data into R, and then some basic operations to use on your data once it's in R. Finally, this volume discusses how to get R to produce some basic univariate statistics. This includes measures of central tendency and variability, as well as the output of basic statistical models like t-tests, correlations and regression, χ^2, and Analysis of Variance (ANOVA).

2.1 What is *R?*

Before using R, it is relevant to discuss a little about what R is as a program. In fact, it is somewhat inaccurate to call R just a program; indeed, R is a programming language. This means that learning R is like learning a new language with its own syntax, structure, and rules. This is usually what makes learning R – or any computer language – particularly challenging for most people. Learning R has many benefits that, although you may not see right away, can aide in your research and practice beyond just computing correlations and t-values. Learning

[4] I remind readers here, as I will throughout the books, that interpretation of the output is purposely limited in this book series. The books are intended to simply help readers produce output using R to facilitate transitioning from using other software programs (e.g., SPSS) to using R.

to program in *R* means you can use *R* to do all sort of other things like scrap text, pull data from webpages into a database, simulate data, and much more. Also, learning to code in *R* can help you develop skills for coding in other programs whose rules might be different, but easily learned once you know *R* (e.g., Python). In addition, the *R* users community is extensive with contributions from disparate fields of study. This means that you can get lots of help with issues you face, and that researchers are continuously adding to the library of packages available in *R*.

2.1.1 R as Object Based

One important thing to know about *R* is that *R* is an object-based environment. This means that if an *R* user wants to store something in the *R* environment to review later, manipulate, or use in subsequent operations, they will have to store it as an object.

For example, if I ran the following *R* code to read in a data file (i.e., open a data set to analyze) as it is written: `read.csv("BBall 22-23 Working.csv", header=TRUE)`, *R* would just print on the screen an N x k matrix of data, where N is the number of rows (e.g., participants in a study) and k is the number of columns (e.g., variables collected) in the data set—in this case a 470 x 27matrix, but I've truncated it to be only the first 10 rows and first 10 columns.

```
read.csv("BBall 22-23 Working.csv", header=TRUE)[1:10,1:10]
```

```
    Rk           Player Pos Age FcBc  Tm Conf  G GS   MP
1   81     Clint Capela   C  28   FC ATL    1 65 63 26.6
2  372  Onyeka Okongwu   C  22   FC ATL    1 80 18 23.1
3   96    John Collins  PF  25   FC ATL    1 71 71 30.0
4  213   Aaron Holiday  PG  26   BC ATL    1 63  6 13.4
5  276      Vit Krejci  PG  22   BC ATL    1 29  0  5.7
6  536      Trae Young  PG  24   BC ATL    1 73 73 34.8
7  105  Jarrett Culver  SF  23   FC ATL    1 10  1 13.7
8  182      AJ Griffin  SF  19   FC ATL    1 72 12 19.5
9  224 De'Andre Hunter  SF  25   FC ATL    1 67 67 31.7
10 244   Jalen Johnson  SF  21   FC ATL    1 70  6 14.9
```

Although *R* would print the matrix, you would not be able to run any statistical analyses on it because it's not stored as an object in the *R* environment.

Instead, if I ran

```
Data <- read.csv("BBall 22-23 Working.csv", header=TRUE)
```

R would store the N x k matrix as an object named `Data` that I could refer to later as a data frame on which I can perform operations (e.g., analyze the data). In other words, by storing the data set as an object, it is stored in the *R* environment to be used later. Because it's stored as an object, for instance, running `head(Data)` prints the first six rows of data:

```
head(Data)
```

```
   Rk         Player Pos Age FcBc  Tm Conf  G GS   MP  FG FGPerc X3P X3Perc X2P
1  81   Clint Capela   C  28   FC ATL    1 65 63 26.6 5.4  0.653 0.0  0.000 5.4
2 372 Onyeka Okongwu   C  22   FC ATL    1 80 18 23.1 4.0  0.638 0.1  0.308 3.9
3  96   John Collins  PF  25   FC ATL    1 71 71 30.0 5.1  0.508 1.0  0.292 4.1
4 213  Aaron Holiday  PG  26   BC ATL    1 63  6 13.4 1.5  0.418 0.6  0.409 0.9
5 276     Vit Krejci  PG  22   BC ATL    1 29  0  5.7 0.5  0.405 0.2  0.238 0.3
6 536     Trae Young  PG  24   BC ATL    1 73 73 34.8 8.2  0.429 2.1  0.335 6.1
  X2Perc  FT FTPerc ORB DRB  TRB  AST STL BLK TOV  PF  PTS
1  0.654 1.2  0.603 4.0 7.1 11.0  0.9 0.7 1.2 0.8 2.1 12.0
2  0.647 1.9  0.781 2.7 4.5  7.2  1.0 0.7 1.3 1.0 3.1  9.9
3  0.619 2.0  0.803 1.1 5.4  6.5  1.2 0.6 1.0 1.1 3.1 13.1
4  0.424 0.4  0.844 0.4 0.8  1.2  1.4 0.6 0.2 0.6 1.3  3.9
5  0.625 0.0  0.500 0.2 0.7  0.9  0.6 0.2 0.0 0.2 0.6  1.2
6  0.476 7.8  0.886 0.8 2.2  3.0 10.2 1.1 0.1 4.1 1.4 26.2
```

Importantly, there are some operations you will run in *R* that will not require assigning the operation to an object. For example, you don't need to define an object for setting a working directory,

```
setwd("C:/Users/devda/Desktop/Using R")
```

installing packages,

```
install.packages("psych", dependencies=TRUE)
```

loading packages (all discussed in further detail below)

nor asking for the first six rows of data (which is a great way to check if your data sets loaded correctly):

```
head(Data)
```

```
   Rk         Player Pos Age FcBc  Tm Conf  G GS   MP  FG FGPerc X3P X3Perc X2P
1  81   Clint Capela   C  28   FC ATL    1 65 63 26.6 5.4  0.653 0.0  0.000 5.4
2 372 Onyeka Okongwu   C  22   FC ATL    1 80 18 23.1 4.0  0.638 0.1  0.308 3.9
3  96   John Collins  PF  25   FC ATL    1 71 71 30.0 5.1  0.508 1.0  0.292 4.1
4 213  Aaron Holiday  PG  26   BC ATL    1 63  6 13.4 1.5  0.418 0.6  0.409 0.9
5 276     Vit Krejci  PG  22   BC ATL    1 29  0  5.7 0.5  0.405 0.2  0.238 0.3
```

```
6 536     Trae Young  PG  24   BC ATL    1 73 73 34.8 8.2  0.429 2.1  0.335 6.1
  X2Perc  FT FTPerc ORB DRB  TRB  AST STL BLK TOV  PF  PTS
1  0.654 1.2  0.603 4.0 7.1 11.0  0.9 0.7 1.2 0.8 2.1 12.0
2  0.647 1.9  0.781 2.7 4.5  7.2  1.0 0.7 1.3 1.0 3.1  9.9
3  0.619 2.0  0.803 1.1 5.4  6.5  1.2 0.6 1.0 1.1 3.1 13.1
4  0.424 0.4  0.844 0.4 0.8  1.2  1.4 0.6 0.2 0.6 1.3  3.9
5  0.625 0.0  0.500 0.2 0.7  0.9  0.6 0.2 0.0 0.2 0.6  1.2
6  0.476 7.8  0.886 0.8 2.2  3.0 10.2 1.1 0.1 4.1 1.4 26.2
```

You can also print descriptive statistics if you don't want to save them. Specifically,

```
Descriptives <- describe(Data)
```

will run the `describe()` function from the *psych* package (Revelle, 2023) on all of the variables in the data frame named `Data`, but store the descriptive statistics in the object `Descriptives`. In order to view them you have to call the object `Descriptives`:

```
Descriptives
```

```
        vars   n   mean     sd median trimmed    mad   min    max  range  skew
Rk         1 470 264.65 155.87 266.50  264.05 197.19  1.00 539.00 538.00  0.02
Player*    2 470 235.50 135.82 235.50  235.50 174.21  1.00 470.00 469.00  0.00
Pos*       3 470   3.04   1.47   3.00    3.06   1.48  1.00   5.00   4.00 -0.03
Age        4 470  25.84   4.33  25.00   25.52   4.45 19.00  38.00  19.00  0.58
FcBc*      5 470   1.59   0.49   2.00    1.61   0.00  1.00   2.00   1.00 -0.37
Tm*        6 470  17.65   9.40  18.00   18.05  13.34  1.00  31.00  30.00 -0.21
Conf       7 470   1.72   0.70   2.00    1.65   1.48  1.00   3.00   2.00  0.45
G          8 470  54.37  19.55  59.00   55.91  22.24 10.00  83.00  73.00 -0.56
GS         9 470  26.12  27.74  12.00   23.24  17.79  0.00  83.00  83.00  0.66
MP        10 470  21.07   9.01  20.20   21.09  11.19  2.50  37.40  34.90  0.04
FG        11 470   3.62   2.43   3.00    3.31   1.93  0.20  11.20  11.00  1.05
FGPerc    12 470   0.47   0.08   0.46    0.46   0.06  0.15   0.82   0.67  0.70
X3P       13 470   1.06   0.88   0.90    0.96   0.89  0.00   4.90   4.90  0.99
X3Perc    14 462   0.33   0.11   0.35    0.34   0.06  0.00   1.00   1.00 -0.37
X2P       15 470   2.56   2.00   1.90    2.28   1.48  0.10  10.50  10.40  1.32
X2Perc    16 470   0.54   0.08   0.54    0.54   0.07  0.07   0.82   0.75 -0.53
FT        17 470   1.56   1.60   1.00    1.26   0.89  0.00  10.00  10.00  2.12
FTPerc    18 467   0.76   0.12   0.77    0.77   0.11  0.00   1.00   1.00 -1.13
ORB       19 470   0.92   0.75   0.70    0.80   0.59  0.00   5.10   5.10  1.77
DRB       20 470   2.87   1.76   2.50    2.65   1.48  0.20   9.60   9.40  1.21
TRB       21 470   3.80   2.34   3.30    3.48   1.93  0.40  12.50  12.10  1.24
AST       22 470   2.19   1.93   1.40    1.86   1.19  0.10  10.70  10.60  1.62
STL       23 470   0.64   0.36   0.60    0.61   0.30  0.10   1.90   1.80  0.82
```

```
BLK       24 470   0.41   0.39   0.30    0.34   0.30  0.00   3.00   3.00  2.64
TOV       25 470   1.17   0.81   0.90    1.06   0.59  0.00   4.10   4.10  1.21
PF        26 470   1.78   0.72   1.70    1.76   0.74  0.10   3.80   3.70  0.25
PTS       27 470   9.86   6.87   7.95    8.94   5.41  0.40  33.10  32.70  1.16
        kurtosis   se
Rk         -1.20 7.19
Player*    -1.21 6.26
Pos*       -1.40 0.07
Age        -0.44 0.20
FcBc*      -1.87 0.02
Tm*        -1.30 0.43
Conf       -0.91 0.03
G          -0.75 0.90
GS         -1.18 1.28
MP         -1.14 0.42
FG          0.42 0.11
FGPerc      2.03 0.00
X3P         0.92 0.04
X3Perc      6.71 0.01
X2P         1.53 0.09
X2Perc      2.84 0.00
FT          5.41 0.07
FTPerc      3.65 0.01
ORB         4.15 0.03
DRB         1.47 0.08
TRB         1.48 0.11
AST         2.61 0.09
STL         0.35 0.02
BLK        10.08 0.02
TOV         0.94 0.04
PF         -0.36 0.03
PTS         0.77 0.32
```

The following, without defining it as an object, will just print the descriptive statistics on the screen:

```
describe(Data)
```

```
        vars   n   mean     sd median trimmed    mad   min    max  range  skew
Rk         1 470 264.65 155.87 266.50  264.05 197.19  1.00 539.00 538.00  0.02
Player*    2 470 235.50 135.82 235.50  235.50 174.21  1.00 470.00 469.00  0.00
Pos*       3 470   3.04   1.47   3.00    3.06   1.48  1.00   5.00   4.00 -0.03
```

Age	4	470	25.84	4.33	25.00	25.52	4.45	19.00	38.00	19.00	0.58
FcBc*	5	470	1.59	0.49	2.00	1.61	0.00	1.00	2.00	1.00	-0.37
Tm*	6	470	17.65	9.40	18.00	18.05	13.34	1.00	31.00	30.00	-0.21
Conf	7	470	1.72	0.70	2.00	1.65	1.48	1.00	3.00	2.00	0.45
G	8	470	54.37	19.55	59.00	55.91	22.24	10.00	83.00	73.00	-0.56
GS	9	470	26.12	27.74	12.00	23.24	17.79	0.00	83.00	83.00	0.66
MP	10	470	21.07	9.01	20.20	21.09	11.19	2.50	37.40	34.90	0.04
FG	11	470	3.62	2.43	3.00	3.31	1.93	0.20	11.20	11.00	1.05
FGPerc	12	470	0.47	0.08	0.46	0.46	0.06	0.15	0.82	0.67	0.70
X3P	13	470	1.06	0.88	0.90	0.96	0.89	0.00	4.90	4.90	0.99
X3Perc	14	462	0.33	0.11	0.35	0.34	0.06	0.00	1.00	1.00	-0.37
X2P	15	470	2.56	2.00	1.90	2.28	1.48	0.10	10.50	10.40	1.32
X2Perc	16	470	0.54	0.08	0.54	0.54	0.07	0.07	0.82	0.75	-0.53
FT	17	470	1.56	1.60	1.00	1.26	0.89	0.00	10.00	10.00	2.12
FTPerc	18	467	0.76	0.12	0.77	0.77	0.11	0.00	1.00	1.00	-1.13
ORB	19	470	0.92	0.75	0.70	0.80	0.59	0.00	5.10	5.10	1.77
DRB	20	470	2.87	1.76	2.50	2.65	1.48	0.20	9.60	9.40	1.21
TRB	21	470	3.80	2.34	3.30	3.48	1.93	0.40	12.50	12.10	1.24
AST	22	470	2.19	1.93	1.40	1.86	1.19	0.10	10.70	10.60	1.62
STL	23	470	0.64	0.36	0.60	0.61	0.30	0.10	1.90	1.80	0.82
BLK	24	470	0.41	0.39	0.30	0.34	0.30	0.00	3.00	3.00	2.64
TOV	25	470	1.17	0.81	0.90	1.06	0.59	0.00	4.10	4.10	1.21
PF	26	470	1.78	0.72	1.70	1.76	0.74	0.10	3.80	3.70	0.25
PTS	27	470	9.86	6.87	7.95	8.94	5.41	0.40	33.10	32.70	1.16

	kurtosis	se
Rk	-1.20	7.19
Player*	-1.21	6.26
Pos*	-1.40	0.07
Age	-0.44	0.20
FcBc*	-1.87	0.02
Tm*	-1.30	0.43
Conf	-0.91	0.03
G	-0.75	0.90
GS	-1.18	1.28
MP	-1.14	0.42
FG	0.42	0.11
FGPerc	2.03	0.00
X3P	0.92	0.04
X3Perc	6.71	0.01
X2P	1.53	0.09
X2Perc	2.84	0.00
FT	5.41	0.07
FTPerc	3.65	0.01

```
ORB         4.15 0.03
DRB         1.47 0.08
TRB         1.48 0.11
AST         2.61 0.09
STL         0.35 0.02
BLK        10.08 0.02
TOV         0.94 0.04
PF         -0.36 0.03
PTS         0.77 0.32
```

2.1.2 Installing R

The last bit of introduction to *R* is installing *R*. Naturally, the first step in using *R* is to install *R* on your computer. Users have two options when picking which type of *R* to install: Basic *R* or RStudio.[5] Both *R* and RStudio offer largely the same functionality, but RStudio has some advantages. For example, RStudio has more user-friendly displays, and allows you to work in *R* Projects and use RMarkdown and Quarto; these latter two options allow for creating documents and/or webpages that integrate interpretation with analysis. In fact, this book series was written using Quarto. Which version of *R* you use is a personal preference, but I will say that most people like RStudio. In addition, you will need to install the base *R* even if you are using RStudio.

To install *R*, follow the instructions on the following webpage: https://www.r-project.org/. From here, you will be asked to select a "CRAN Mirror" from which to download *R*. CRAN stands for Comprehensive *R* Archive Network, and this network is a collection of internet hosts that contain the same information, but allow large numbers of individuals to access *R* simultaneously without overloading one server. You usually select a CRAN nearest you. To install RStudio, you follow the instructions here: https://www.rstudio.com/products/rstudio/download/.

Having covered some of the initial issues, we can now turn to some of the basic operations you'll need to get started with *R*.

2.2 Basic Operations

In this section, I outline some initial operations that are typically done before any analysis. These include: setting a working directory, loading data sets into *R*, and installing and loading *R* packages. Admittedly, this section does not cover *all* of the operations you might need. Rather, I selected these as these tend to be the ones that most early users need immediately, with more nuanced or advanced operations (e.g., reshaping data) being needed later. For

[5] I note that there is also *R* commander which allows for a point-and-click functionality. This book series does not consider this option.

example, nearly everyone will need to know how to set a variable in their data set to a factor in order to run an ANOVA early in their transition to *R*. Knowing how to convert a data set from wide to long format to conduct multilevel modelings, though, is something that new users need after mastering the initial operations and basic analyses. As such, I demonstrate these more advanced operations in later volumes of the series.

2.2.1 Setting a Working Directory

A "working directory" is the folder on your computer in which you are going to work. The reason for setting a working directory is to facilitate opening and saving files. The working directory you set is going to be the folder in which the data you're working with is stored. This is also where your script files will be saved, which you should be saving frequently!

By setting a working directory, you can save yourself the need to include the full path directory information when trying to read in to or save out of the *R* environment. For example, if I needed to read in the above data set before setting a working directory, I would have to write the following:

```
Data <- read.csv("C:/Users/Dev Dalal/Desktop/Using R/BBall 22-23 Working.csv",
header=TRUE)
```

And to save something from *R* to the same folder I would have to write:

```
write.table(coefficients,"C:/Users/Dev Dalal/Desktopp/Using R/"Descriptive
Statistics.csv", sep=",")
```

The above line of code will save an object in the *R* environment I named `coefficient` as the file named "Descriptive Statistics.csv" as a comma separated values file in the same folder as the data.

If I set the working directory from the start, the previous two lines of coding is shortened considerably . If I set the working directory for the above example as:

```
setwd("C:/Users/Dev Dalal/Desktop/Using R/")
```

R knows to read all objects from and save all objects to the *Using R* subfolder of the *Desktop*[6] folder on my computer. Then, to read in the data and/or save the coefficients, I simply need to code:

```
Data <- read.csv("BBall 22-23 Working.csv", header=TRUE)
```

```
write.table(coefficients, "Descriptive Statistics.csv", sep=",")
```

As you can see, this line of code is significantly shortened because of the setting of a working directory earlier–something we only need to do once at the start of our code.

[6]This is for all of your monsters who save everything to your Desktop. It's just anarchy on that Desktop! It's not safe for anyone!

To set the working directory for a Windows computer, you use the following general code:

```
setwd("[DRIVE LETTER]:/Users/[YOUR USER NAME]/[MAIN FOLDER]/[SUBFOLDER 1 (IF APPLICABLE)/")
```

For example:

```
setwd("C:/Users/Dev Dalal/Documents/Using R")
```

To set the working directory for a Mac, you use the following general code:

```
setwd("/Users/[YOUR USER NAME]/[MAIN FOLDER]/[SUBFOLDER 1 (IF APPLICABLE)/")
```

For example:

```
setwd("/Users/Dev Dalal/Documents/Using R")
```

You'll know you've successfully set a working directory if the code runs without an error message. If you are following along with coding while reading this volume, now would be a good time to set your working directory before moving forward.

2.2.2 Getting Data into *R*

After setting a working directory, the next step to analyzing data is actually having data in *R* to analyze. There are many ways in which you can have your data saved and read into *R*. Naturally, the first way is to input the data yourself; that is, you can enter the data into *R* manually. This would work easily if you only have a small amount of data to enter.

The other way is to read in a data set created in some other program. Any text file with any delimiter (e.g., tab delimited, comma separated values) can be handled by *R*. A delimiter is a character that separates values in a file. So, for example, a comma separated values (csv) file – the type of file used in these volumes – separates the data points using commas.[7]

2.2.2.1 Inputting Data into *R*

To input data into *R*, you have to define the data as an object. Although the following demonstration is unnecessarily broken into multiple steps, it will help you understand how data can be entered into *R*. In the below code, the `<-` code assigns/stores the vector of numbers on the right into an object named whatever is to the left of the `<-`. The `c()` code tells *R* to combine these into a single object of different elements (i.e., numbers). The below code will create two vectors, `Y` and `X1`, with ten entries each:

[7] You can read directly into your *R* environment SPSS data files, and .xls/.xlsx, Excel data. However, I have personally, and have heard from others, that reading in files this way can result in some issues. So, my general recommendation is to save data files into .csv format. Any existing statistics software (e.g., SPSS, SAS), spreadsheet application (e.g., Google Sheets, Excel), and survey software (e.g., Qualtrics, SurveyMonkey) can save data as .csv files with variable names as the first row.

```
Y <- c(71, 67, 69, 56, 53, 52, 59, 70, 75, 53)
X1 <- c(16, 18, 30, 18, 24, 12, 30, 32, 26, 14)
Y
```

```
 [1] 71 67 69 56 53 52 59 70 75 53
```

```
X1
```

```
 [1] 16 18 30 18 24 12 30 32 26 14
```

To make these two vectors a single data set to analyze, we use the `data.frame()` function. A data matrix in *R* is typically referred to as a data frame; note, though, that some packages will require data in matrix format–I will point this out as we encounter these situations.

The `data.frame()` function allows us to specify a new object as a data frame with Y and X1 relabeled "PTG" and "SS," respectively. We will then have 10 cases on 2 variables in a 10 x 2 data frame.

```
Example1 <- data.frame(PTG = Y, SS = X1)
Example1
```

```
   PTG SS
1   71 16
2   67 18
3   69 30
4   56 18
5   53 24
6   52 12
7   59 30
8   70 32
9   75 26
10  53 14
```

From here, we could start the performance operations on PTG and SS (e.g., correlate them; print descriptive statistics, etc.).

2.2.2.2 Loading Data Sets into *R*

As noted earlier, it is more likely that you will be reading in a data set from an external source (e.g., Qualtrics, Excel) than inputting your own data. Reading a .csv file into *R* is straightforward thanks to the `read.csv()` function. Furthermore, assuming you have set a working directory, it just requires the name of the file. Remember, though, that because *R* is an object-based language, you have to assign the data to an object; otherwise, *R* will just print the data matrix and you won't be able to do anything with it.

Although we did this earlier, we will read in *BBall 22-23 Working.csv* again using the `read.csv()` function. This is some data from the 2022-2023 season of the National Basketball Association; we will use this data later to perform some univariate statistics, and I explain the data in more detail in that section.

```
Data1 <- read.csv("BBall 22-23 Working.csv", header=TRUE)
```

Note three things with this code. First, the file is being assigned to the object `Data1` using the `<-` part of the code. Second, `header= TRUE` tells *R* that the first row of the file is variable names. Finally, `TRUE` is all capitalized. *R*, like all languages, computer or otherwise, has specific rules, and one of *R*'s rules is that commands that require a `TRUE/FALSE` response must be all capitalized.

```
Data1 <- read.csv("BBall 22-23 Working.csv", header=TRUE)
```

After running this code, you should have a data frame in your *R* environment named `Data1`. If you are using RStudio, the *Environment* window pane in your interface should show an object under the *Data* section named *Data1.* Alternatively, you can using the `objects()` function to print a list of the different objects you've defined in your *R* environment. In addition to the other objects we've defined earlier, your environment should also have an object named `Data1` if the `read.csv()` function worked:

```
objects()
```

```
[1] "Data"         "Data1"        "Descriptives" "Example1"     "X1"
[6] "Y"
```

`Data1` will have 254 cases on 33 variables.

To make sure your data read in correctly, you can print the first and/or last six rows of data using the `head()` and/or `tail()` functions, respectively, including within the parentheses the name of the data frame object:

```
head(Data1)
```

```
   Rk         Player Pos Age FcBc  Tm Conf  G GS   MP  FG FGPerc X3P X3Perc X2P
1  81   Clint Capela   C  28   FC ATL    1 65 63 26.6 5.4  0.653 0.0  0.000 5.4
2 372 Onyeka Okongwu   C  22   FC ATL    1 80 18 23.1 4.0  0.638 0.1  0.308 3.9
3  96   John Collins  PF  25   FC ATL    1 71 71 30.0 5.1  0.508 1.0  0.292 4.1
4 213  Aaron Holiday  PG  26   BC ATL    1 63  6 13.4 1.5  0.418 0.6  0.409 0.9
5 276     Vit Krejci  PG  22   BC ATL    1 29  0  5.7 0.5  0.405 0.2  0.238 0.3
6 536     Trae Young  PG  24   BC ATL    1 73 73 34.8 8.2  0.429 2.1  0.335 6.1
  X2Perc  FT FTPerc ORB DRB  TRB  AST STL BLK TOV  PF  PTS
1  0.654 1.2  0.603 4.0 7.1 11.0  0.9 0.7 1.2 0.8 2.1 12.0
2  0.647 1.9  0.781 2.7 4.5  7.2  1.0 0.7 1.3 1.0 3.1  9.9
3  0.619 2.0  0.803 1.1 5.4  6.5  1.2 0.6 1.0 1.1 3.1 13.1
4  0.424 0.4  0.844 0.4 0.8  1.2  1.4 0.6 0.2 0.6 1.3  3.9
5  0.625 0.0  0.500 0.2 0.7  0.9  0.6 0.2 0.0 0.2 0.6  1.2
6  0.476 7.8  0.886 0.8 2.2  3.0 10.2 1.1 0.1 4.1 1.4 26.2
```

```
tail(Data1)
```

```
     Rk        Player Pos Age FcBc  Tm Conf  G GS   MP  FG FGPerc X3P X3Perc
465 349  Monte Morris  PG  27   BC WAS    1 62 61 27.3 4.0  0.480 1.3  0.382
466 532  Delon Wright  PG  30   BC WAS    1 50 14 24.4 2.8  0.474 0.8  0.345
467  16   Deni Avdija  SF  22   FC WAS    1 76 40 26.6 3.3  0.437 0.9  0.297
468 268 Corey Kispert  SF  23   FC WAS    1 74 45 28.3 3.9  0.497 2.2  0.424
469  35  Bradley Beal  SG  29   BC WAS    1 50 50 33.5 8.9  0.506 1.6  0.365
470 111  Johnny Davis  SG  20   BC WAS    1 28  5 15.1 2.4  0.386 0.6  0.243
    X2P X2Perc  FT FTPerc ORB DRB TRB AST STL BLK TOV  PF  PTS
465 2.7  0.543 1.0  0.831 0.4 3.0 3.4 5.3 0.7 0.2 1.0 1.2 10.3
466 1.9  0.564 1.0  0.867 1.2 2.4 3.6 3.9 1.8 0.3 0.9 1.2  7.4
467 2.4  0.530 1.6  0.739 1.0 5.4 6.4 2.8 0.9 0.4 1.6 2.8  9.2
468 1.7  0.637 1.0  0.852 0.4 2.4 2.8 1.2 0.4 0.1 0.7 1.3 11.1
469 7.3  0.552 3.8  0.842 0.8 3.1 3.9 5.4 0.9 0.7 2.9 2.1 23.2
470 1.8  0.485 0.5  0.519 0.4 2.0 2.3 1.0 0.4 0.3 0.6 1.7  5.8
```

As this point, you should be able to read in any .csv file into *R* as long as you know in what working directory it is stored! Once the data are in *R*, you can start performing operations with your data.

2.2.3 Installing and Loading Packages

One of the main benefits of using *R* is that members of the *R* community have taken the time to develop packages that have different functions that run different analyses and/or commands. Using the functions within these packages makes it easier to run all of your analyses because users do not have to program a function for each thing they want *R* to do. This is particularly

helpful for users of *R* who are interested in producing statistical output. Importantly, if a package/function to conduct an analysis does not already exist, *R* can be used to program this function. If you stick with *R* long enough, you will find that it's a very quick transition from using packages to programming your own functions.

Fortunately, installing and loading packages is straightforward. The code to install a package is `install.packages()` and the code to load a package is `library()`. So, for instance, if we wanted to install the *psych* package (Revelle, 2023), we would run:

`install.packages("psych", dependencies=TRUE)`.

Importantly, package developers can, at times, rely on other packages, developed by other people, to make their package simpler. The `dependencies=TRUE` part of the code will install any package on which the needed package depends to run analyses. For instance, the *psych* package makes use of the *boot* package to run some of its analyses–the `dependencies=TRUE` part of the `install.packages()` function will install the *boot* package as part of installing *psych.*

To make the functions within a package available, you have to load the package to your *R* environment's library. To do this, we use the `library()` function with the name of the package within the parentheses:

`library(psych)`.

When you install the package, you may notice some warnings telling you about the dependent packages that were not available. This is okay as they are likely replaced by other packages. When you load the *psych* package, you might encounter warning messages. These warnings can likely be ignored as it simply tells the user that the package was developed under a specific version of *R*.

To ensure that the *psych* package was loaded successfully, you can run the `search()` function which will display all packages and tools that are currently active in your *R* environment. If your desired package loaded correctly, in this case the *psych* package, it should show as `package:NAME` in the output. As we can see, `package:psych` is shown among the packages operating.

```
search()
```

```
 [1] ".GlobalEnv"        "package:psych"     "package:stats"
 [4] "package:graphics"  "package:grDevices" "package:utils"
 [7] "package:datasets"  "package:methods"   "Autoloads"
[10] "package:base"
```

2.2.4 Basic *R* Commands

Next, we turn to performing some basic operations in *R*. These operations, though not exhaustive, should set you up well for performing analyses in *R*.

2.2.4.1 References a Specific Part of an Object

First, referencing a specific part of an object in *R* is done with the use of the `$` sign. For example, if I wanted to add the data in column 1 to the data in column 2 for each person (akin to making a scale sum-score for a two-item measure), I would use the `$` to reference each column:

```
Data1$Column3 <- Data1$Column1 + Data1$Column2
```

Note that if Column3 doesn't exist in the data set, *R* knows to create it, and that the values will be whatever each row of data has in Column1 added to Column2 data. If Column3 does exist, though, *R* will overwrite whatever is in that column, so make sure you are careful about creating new variables.

Using *Data1*, you can create the variable `NegPlay` as the sum of `TOV` and `PF` to create. Although these data are described in further detail below, `TOV` refers to the average number of times a player gave the ball away to the opposing team a game, and `PF` refers to the average number of fouls a player committed a game. The sum of these two would be, then, the average number of negative plays the player had in a game during the season.

```
Data1$NegPlay <- Data1$TOV + Data1$PF
```

If you were successful, using the `head()` command, you should see a new variable at the end named `NegPlay`.

```
head(Data1)
```

```
   Rk         Player Pos Age FcBc  Tm Conf  G GS   MP  FG FGPerc X3P X3Perc X2P
1  81   Clint Capela   C  28   FC ATL    1 65 63 26.6 5.4  0.653 0.0  0.000 5.4
2 372 Onyeka Okongwu   C  22   FC ATL    1 80 18 23.1 4.0  0.638 0.1  0.308 3.9
3  96   John Collins  PF  25   FC ATL    1 71 71 30.0 5.1  0.508 1.0  0.292 4.1
4 213  Aaron Holiday  PG  26   BC ATL    1 63  6 13.4 1.5  0.418 0.6  0.409 0.9
5 276     Vit Krejci  PG  22   BC ATL    1 29  0  5.7 0.5  0.405 0.2  0.238 0.3
6 536     Trae Young  PG  24   BC ATL    1 73 73 34.8 8.2  0.429 2.1  0.335 6.1
  X2Perc  FT FTPerc ORB DRB  TRB  AST STL BLK TOV  PF  PTS NegPlay
1  0.654 1.2  0.603 4.0 7.1 11.0  0.9 0.7 1.2 0.8 2.1 12.0     2.9
2  0.647 1.9  0.781 2.7 4.5  7.2  1.0 0.7 1.3 1.0 3.1  9.9     4.1
3  0.619 2.0  0.803 1.1 5.4  6.5  1.2 0.6 1.0 1.1 3.1 13.1     4.2
4  0.424 0.4  0.844 0.4 0.8  1.2  1.4 0.6 0.2 0.6 1.3  3.9     1.9
```

```
5  0.625 0.0  0.500 0.2 0.7  0.9  0.6 0.2 0.0 0.2 0.6  1.2     0.8
6  0.476 7.8  0.886 0.8 2.2  3.0 10.2 1.1 0.1 4.1 1.4 26.2     5.5
```

The use of the $ will become more valuable when you want to print a specific part of an output. For example, if we wanted to print the column for the new sum scores we just created, we can use the $ to call that specific column only:

```
Data1$NegPlay
```

```
  [1] 2.9 4.1 4.2 1.9 0.8 5.5 2.1 1.8 4.2 2.2 2.8 1.4 0.2 3.6 2.3 2.5 1.6 2.0
 [19] 2.9 3.4 3.1 1.6 5.1 5.5 1.6 0.4 5.1 0.7 3.4 4.1 2.8 1.6 5.6 2.5 4.6 1.8
 [37] 2.4 2.1 2.5 2.1 0.3 2.8 3.9 1.7 3.0 3.5 4.6 2.2 3.5 4.6 0.8 2.6 2.1 3.3
 [55] 3.1 1.6 4.1 6.9 2.7 3.8 3.4 4.2 2.3 1.9 4.0 3.7 1.7 1.9 4.6 0.8 2.4 5.0
 [73] 1.5 2.4 2.7 2.3 1.7 3.9 5.1 2.2 3.7 4.3 1.4 3.0 6.1 1.6 2.6 2.5 3.8 2.3
 [91] 2.0 1.2 6.1 3.0 2.0 3.3 2.6 2.6 3.8 1.8 3.9 3.0 0.6 1.8 3.0 0.9 3.0 4.1
[109] 4.1 4.2 2.6 6.1 5.2 2.3 0.6 3.5 3.0 6.5 1.5 1.8 3.2 5.9 2.7 3.7 5.3 5.7
[127] 2.1 0.9 3.1 3.7 4.2 3.4 2.1 1.4 2.4 0.8 6.0 3.5 4.2 2.5 5.8 1.4 2.9 4.9
[145] 1.8 4.3 3.2 3.4 5.2 1.7 2.2 3.7 3.3 2.3 3.7 4.2 4.0 4.1 1.1 4.4 2.5 2.5
[163] 3.0 1.2 4.1 1.5 1.4 5.9 3.3 2.8 3.8 4.8 2.8 4.8 3.9 2.2 1.1 3.2 2.1 4.2
[181] 5.3 2.3 2.7 3.3 1.9 1.7 2.3 0.9 1.3 5.0 4.7 2.1 2.6 4.8 1.1 5.3 2.2 3.1
[199] 2.9 1.5 2.3 2.0 4.5 3.7 3.1 2.5 3.0 3.9 4.5 2.1 4.0 7.0 0.9 1.9 2.8 3.0
[217] 4.6 2.3 1.6 2.8 1.7 4.3 2.6 1.2 2.0 4.7 1.7 4.0 3.6 3.6 6.8 1.5 4.8 0.9
[235] 5.7 1.4 0.7 2.6 1.9 2.1 2.7 2.6 5.1 4.5 5.6 3.3 2.7 1.4 4.4 5.9 2.8 2.8
[253] 0.8 3.4 3.4 2.3 5.8 1.6 4.3 1.3 1.5 4.7 2.5 3.5 3.2 3.1 1.8 4.7 3.0 2.1
[271] 2.7 5.6 2.4 4.5 1.9 1.2 2.4 4.1 4.7 3.6 5.0 2.9 0.9 2.0 4.1 4.5 1.6 2.1
[289] 4.4 2.5 2.2 4.1 6.5 1.9 2.5 2.6 3.0 5.3 3.2 1.7 1.1 3.5 3.8 2.8 1.1 4.6
[307] 2.7 2.7 3.2 4.0 3.5 2.4 5.7 2.7 2.5 2.4 1.8 1.3 3.1 5.9 1.2 4.2 3.0 4.6
[325] 5.2 1.9 1.7 2.2 2.7 4.4 1.5 1.7 6.4 2.3 2.2 1.7 1.3 0.9 4.9 2.2 0.7 2.8
[343] 2.4 0.7 3.7 3.5 3.3 5.2 2.8 2.5 1.7 2.0 4.2 3.0 4.1 2.7 2.7 2.9 2.3 3.0
[361] 3.6 3.0 1.6 2.5 2.4 1.4 5.6 2.4 1.7 0.7 1.7 4.8 5.0 4.2 2.3 2.7 1.6 2.3
[379] 1.5 2.3 2.7 2.5 1.9 0.9 2.0 2.0 2.7 4.2 4.4 3.5 5.4 3.1 1.9 2.9 2.7 1.9
[397] 1.1 3.6 2.7 2.0 1.3 3.6 2.2 1.8 3.2 3.4 2.3 4.6 5.7 1.9 4.9 2.5 2.1 1.5
[415] 4.1 2.0 3.8 1.5 2.3 3.2 3.0 1.0 2.2 2.3 3.4 1.9 2.4 1.9 2.4 2.6 2.4 2.0
[433] 2.2 0.9 1.5 1.8 2.7 1.4 3.1 2.2 1.7 3.7 4.2 1.4 3.1 5.9 1.9 4.0 4.2 4.1
[451] 1.2 2.0 2.4 5.0 3.6 0.9 0.7 3.5 2.2 5.1 2.1 1.5 5.3 2.4 2.2 2.1 4.4 2.0
[469] 5.0 2.3
```

Importantly, you can use this basic command on any number of columns and/or performing any mathematical operation. For example, creating a product or polynomial term to use in moderated multiple regression and/or polynomial regression requires multiplication and exponentiation, respectively.

```
Data$Interaction <- Data$Var1 *  Data$Var2
```

```
Data$Squared <- Data$Var1^2
```

The first line creates the product of *Var1* and *Var2* and names it *Interaction* in the *Data* data frame. The second line creates the variable *Squared* by squaring *Var1*. These new variables can now be used as part of moderated and/or polynomial regression analysis—discussed in Volume II of this series.

2.2.4.2 Creating Scale Scores from Means

Researchers in the behavioral sciences typically use means of individual item-responses to index construct standing. That is, rather than using sum scores, they use mean scores to create scores on measures of study variables. The process, for the most part, is the same for creating mean scores: you have to tell *R* that you need a new variable that is the mean of existing columns of data. The difference is the function to do this directly is `rowMeans()`. 9Note, that the M in `rowMeans()` is capitalized.) In addition, we have to nest within the `rowMeans()` function the `cbind()` function. The `cbind()` function tells *R* to collect, bind, the columns included in the parentheses.

For example, `cbind(Data1$X3Perc, Data1$X2Perc, Data1$FTPerc)` tells *R* to bind together columns `X3Perc` (the average percentage of three point shots made in a game), `X2Perc` (the average percentage of two point shots made in a game), and `FTPerc` (the average percentage of free throw shorts made in a game). If we ran just this line of code, *R* would just print the 470 rows of these three columns. (I demonstrate this below, but truncate the output to only the first 20 rows to save space.)

```
cbind(Data1$X3Perc, Data1$X2Perc, Data1$FTPerc)[1:20,]
```

```
      [,1]  [,2]  [,3]
 [1,] 0.000 0.654 0.603
 [2,] 0.308 0.647 0.781
 [3,] 0.292 0.619 0.803
 [4,] 0.409 0.424 0.844
 [5,] 0.238 0.625 0.500
 [6,] 0.335 0.476 0.886
 [7,] 0.083 0.516 0.692
 [8,] 0.390 0.536 0.894
 [9,] 0.350 0.521 0.826
[10,] 0.288 0.587 0.628
[11,] 0.406 0.506 0.831
[12,] 0.000 0.431 0.667
[13,] 0.143 0.500 1.000
[14,] 0.344 0.514 0.832
[15,] 0.348 0.625 0.656
```

```
[16,] 0.446 0.539 0.714
[17,] 0.231 0.701 0.821
[18,] 0.250 0.500 1.000
[19,] 0.000 0.751 0.610
[20,] 0.395 0.546 0.770
```

This is because we did not define it as an object like we discussed earlier. If, however, I nested the above code within the `rowMeans()` function, it would perform `rowMeans()`on the three columns. In short, this would produce, for each row of data, the average percentage of successful scoring plays across the season:

```
Data1$ScoreMean <- rowMeans(cbind(Data1$X3Perc, Data1$X2Perc, Data1$FTPerc),
na.rm=TRUE)
```

Here, *R* will create a new column of data called `ScoreMean` in `Data1`, and it will be the mean of the three variables. The `na.rm=` part is how you tell *R* what to do with missing data. (Unfortunately, the `na.rm=` code is not universal. One of the growing pains of learning *R* will be learning package/function specific coding for missing data). By setting `na.rm=TRUE`, we are telling *R* to remove all missing data before computing the mean. If the above code ran successfully, the first six rows of your data would look like:

```
Data1$ScoreMean <- rowMeans(cbind(Data1$X3Perc, Data1$X2Perc, Data1$FTPerc), na.rm=TRUE)
head(Data1)
```

```
   Rk          Player Pos Age FcBc  Tm Conf  G GS   MP  FG FGPerc X3P X3Perc X2P
1  81   Clint Capela   C  28   FC ATL    1 65 63 26.6 5.4  0.653 0.0  0.000 5.4
2 372 Onyeka Okongwu   C  22   FC ATL    1 80 18 23.1 4.0  0.638 0.1  0.308 3.9
3  96   John Collins  PF  25   FC ATL    1 71 71 30.0 5.1  0.508 1.0  0.292 4.1
4 213  Aaron Holiday  PG  26   BC ATL    1 63  6 13.4 1.5  0.418 0.6  0.409 0.9
5 276     Vit Krejci  PG  22   BC ATL    1 29  0  5.7 0.5  0.405 0.2  0.238 0.3
6 536     Trae Young  PG  24   BC ATL    1 73 73 34.8 8.2  0.429 2.1  0.335 6.1
  X2Perc  FT FTPerc ORB DRB  TRB  AST STL BLK TOV  PF  PTS NegPlay ScoreMean
1  0.654 1.2  0.603 4.0 7.1 11.0  0.9 0.7 1.2 0.8 2.1 12.0     2.9 0.4190000
2  0.647 1.9  0.781 2.7 4.5  7.2  1.0 0.7 1.3 1.0 3.1  9.9     4.1 0.5786667
3  0.619 2.0  0.803 1.1 5.4  6.5  1.2 0.6 1.0 1.1 3.1 13.1     4.2 0.5713333
4  0.424 0.4  0.844 0.4 0.8  1.2  1.4 0.6 0.2 0.6 1.3  3.9     1.9 0.5590000
5  0.625 0.0  0.500 0.2 0.7  0.9  0.6 0.2 0.0 0.2 0.6  1.2     0.8 0.4543333
6  0.476 7.8  0.886 0.8 2.2  3.0 10.2 1.1 0.1 4.1 1.4 26.2     5.5 0.5656667
```

Note, there is now two new columns in `Data1`: `NegPlay` and `ScoreMean`.

It is also worth noting here that if there are a lot of variables to average together, the above code can get unwieldy to write. There are, fortunately, shortcuts that can be used, I discuss one shortcut next.

2.2.4.3 Referencing Columns using Column Numbers

One shortcut to coding in *R* is to reference the columns in a function using column numbers rather than column names. Although this option is not available for all function, it is helpful for any function in which the coding requires the names of columns.

In addition, this does require knowing the column numbers associated with your variable names–which can be difficult for larger data sets. However, when you become proficient at using this approach to coding, you will see your coding moves quickly. Indeed, in the remaining parts of the book series, I will make use of this shortcut whenever possible.

The first step in using the shortcut would be to determine the column numbers associated with the variables in the data frame using the `colnames()` function. By putting the name of the data frame in this function, *R* will print the names and numbers of each column.

```
colnames(Data1)
```

```
 [1] "Rk"        "Player"    "Pos"       "Age"       "FcBc"      "Tm"
 [7] "Conf"      "G"         "GS"        "MP"        "FG"        "FGPerc"
[13] "X3P"       "X3Perc"    "X2P"       "X2Perc"    "FT"        "FTPerc"
[19] "ORB"       "DRB"       "TRB"       "AST"       "STL"       "BLK"
[25] "TOV"       "PF"        "PTS"       "NegPlay"   "ScoreMean"
```

Admittedly, this is not the most visually appealing way to display this information, but you do get the critical information needed. On the left, in brackets, are the column numbers that start each row of variables, so columns 1 through 6 are in the `[1]` row followed by columns 7 through 12 in the `[7]` row, and so on.

From this output, then, we know in which column *number* is each variable: variable `Player` is in column 2, and variable `Conf` is in column 7. With this information, we can use column numbers from a data frame to reference columns/variables in our functions.

For example, suppose we wanted to rerun the `rowMeans()` function from above, but using the column numbers. The code would replace the `cbind()` part of the code with the code to reference specific columns. To do that, we need to use the `[ , ]` next to the data frame names.

Recall that data frames are just matrices of data. Data matrices are oriented in Row x Column structure. The left and right sides of the comma in the `[ , ]` code represent rows of the data frame and columns of a data frame, respectively. As such, putting value numbers to the left and right of the comma will reference specific cells of the matrix. For example:

1. `Data1[10,5]` will print the data point in row 10, column 5 of the data frame `Data1`

   ```
   Data1[10,5]
   ```

```
[1] "FC"
```

2. `Data1[5,10]` will print the data point in row 5, column 10 of the data frame `Data1`

```
Data1[5,10]
```

```
[1] 5.7
```

In addition to a single row or column, you can specify sets by including more than one number to the left and/or right of the column. When the row/column numbers are consecutive, you can indicate them using the `:`. So, for instance, `Data1[1:20,10:15]` will print the first 20 rows on columns 10 through 15:

```
Data1[1:20, 10:15]
```

```
     MP  FG FGPerc X3P X3Perc X2P
1  26.6 5.4  0.653 0.0  0.000 5.4
2  23.1 4.0  0.638 0.1  0.308 3.9
3  30.0 5.1  0.508 1.0  0.292 4.1
4  13.4 1.5  0.418 0.6  0.409 0.9
5   5.7 0.5  0.405 0.2  0.238 0.3
6  34.8 8.2  0.429 2.1  0.335 6.1
7  13.7 1.7  0.395 0.1  0.083 1.6
8  19.5 3.4  0.465 1.4  0.390 2.0
9  31.7 5.7  0.461 1.5  0.350 4.2
10 14.9 2.3  0.491 0.4  0.288 1.8
11 27.9 5.1  0.447 2.7  0.406 2.4
12 12.0 1.1  0.417 0.0  0.000 1.1
13  4.1 0.6  0.391 0.1  0.143 0.5
14 36.4 8.3  0.464 1.8  0.344 6.5
15 13.9 1.5  0.485 0.6  0.348 1.0
16 30.5 3.6  0.476 2.3  0.446 1.3
17 11.7 1.6  0.665 0.0  0.231 1.6
18  7.4 0.5  0.458 0.0  0.250 0.4
19 23.5 3.6  0.747 0.0  0.000 3.6
20 25.9 2.7  0.454 1.5  0.395 1.3
```

If the rows and/or columns are not consecutive, you have to "collect" them using the `c()` function nested within the `[ , ]`. To display rows, 2, 4, 6, 8, and 10, and columns 1, 3, 5, 7, and 9, you would have to "collect" them together as `c(2,4,6,8,10)` and `c(1,3,5, 7,9)` within the `[ , ]`:

```
Data1[c(2,4,6,8,10), c(1,3,5,7,9)]
```

```
    Rk Pos FcBc Conf GS
2  372   C   FC    1 18
4  213  PG   BC    1  6
6  536  PG   BC    1 73
8  182  SF   FC    1 12
10 244  SF   FC    1  6
```

Finally, and most useful for nesting this coding into other functions, if either side of the comma is left blank, *R* will retain all entries for whichever side is left blank. So, `Data1[,1:5]` will print columns 1 through 5 for all 470 entries, and `Data1[1:20,]` will print all 29 columns for the first 20 rows.

Therefore, if we want to perform an operation, say computing the mean from three columns, for all entries in the data frame, we can leave the left side of the comma blank, and just indicate the column numbers we average.

Taking the above example of the mean of `X3Perc`, `X2Perc`, and `FTPerc`, we can include columns 14, 16, and 18 into the right side `[ , ]` and left the left side blank to apply to all rows:

```
Data1$ScoreMean2 <- rowMeans(Data1[,c(14,16,18)], na.rm=TRUE)
```

This code creates a new column of data in `Data1` named `NASMean2`. If we use the coding we just learned, we can print the first 15 rows of data (i.e, `1:15`) on the last two columns (i.e., `29:30`)[8], and see that the two columns, `ScoreMean` and `ScoreMean2`, are identical for each row:

```
Data1[1:15,29:30]
```

```
   ScoreMean ScoreMean2
1  0.4190000  0.4190000
2  0.5786667  0.5786667
3  0.5713333  0.5713333
4  0.5590000  0.5590000
5  0.4543333  0.4543333
6  0.5656667  0.5656667
7  0.4303333  0.4303333
8  0.6066667  0.6066667
9  0.5656667  0.5656667
10 0.5010000  0.5010000
11 0.5810000  0.5810000
12 0.3660000  0.3660000
```

[8] I know that these are the two columns of interest without needing to rerun `colnames(Data1)` because when I ran it earlier, there were 29 columns with `ScoreMean` being the last column. As such, `ScoreMean2` has to be the 30th column.

```
13 0.5476667  0.5476667
14 0.5633333  0.5633333
15 0.5430000  0.5430000
```

Admittedly, the example of nesting the referencing of column numbers rather than names was not a large reduction in actual coding. Keep in mind, though, that running operations on large numbers of variables is more common than not, so using this approach will become helpful quickly. As noted earlier, I use this approach to coding whenever possible moving forward.

2.2.4.4 Creating Factors

Factors are nominal variables that are used to group similar categories of observations together, such as experimental conditions or demographic groups. When variables are set as factors, *R* will treat them as groups in analyses. To create factors, we would either need to add a new variable to the data frame as a factor or turn an existing variable into a factor. In both cases we would use the `factor()` function. In the former case, we would first add a variable; in the latter case we would convert an existing variable into a factor.

If we are adding a new variable to our data frame that is a factor, we first add a column of data to our data frame that contains the value designating to which level of the factor each row belongs:

```
Example1$Cond <- c(1,1,1,1,1,2,2,2,2,2)
```

The above code adds to the data frame `Example` the column named `Cond` that is a series of 1s and 2s–the first five rows are 1s and the second five rows are 2s. To convert these 1s and 2s to a factor with two levels, we use the `factor()` function. As with other functions, though, you have to assign it an object. In this case, we are going to take the existing column, `Cond` and reassign it as a factor.

```
Example1$Cond <- factor(Example1$Cond, levels = c(1,2), labels = c("Control",
"Treatment"))
```

Here we see that the `Cond` variable in `Example1` will be converted to a factor from the existing data in `Example1$Cond`. In addition, there are two `levels=` to this factor, collected as `c(1,2)`. Finally, we are assigning `labels=` for levels 1 and 2, to `c("Control", "Treatment")` , respectively.

```
Example1$Cond <- c(1,1,1,1,1,2,2,2,2,2)
Example1$Cond <- factor(Example1$Cond, levels = c(1,2), labels = c("Control", "Treatment"))
```

We now see that the data frame we created has five control and five treatment labels.

```
Example1
```

```
   PTG SS      Cond
1   71 16   Control
2   67 18   Control
3   69 30   Control
4   56 18   Control
5   53 24   Control
6   52 12 Treatment
7   59 30 Treatment
8   70 32 Treatment
9   75 26 Treatment
10  53 14 Treatment
```

Some notes on the code. First, in both the `levels=` and `labels=` parts, the entries have to be "collected" together using the `c()` function. Second, the `labels=` input have to be in quotation makes for the text to be accepted. Finally, if your data already contain a column of data for the factor, you can just use the `factor()` part of the example.

As a hypothetical example, suppose your data set contained responses to a gender identity question. Further suppose that respondents were given one of four options to select: 1) male; 2) female; 3) non-binary; 4) self-described. To ensure that this variable is treated as a factor in *R* you would need to use the `factor()` function:

```
Data$GenID <- factor(Data$GenID, levels=c(1,2,3,4), labels=c("Male", "Female",
"Non-Binary", "Self.Desc")
```

This code would convert the entries of 1, 2, 3, and 4 to labels of "Male," "Female," "Non-Binary," and "Self.Desc," respectively, and treat the `GenID` variable as a factor.

2.2.4.5 Saving *R* Objects

The last basic operation discussed here is how to save objects in your *R* environment into your working directory. You may do this to save the output of an analysis you ran, such as a table of descriptive statistics, correlation matrices, and/or coefficients. You may also alter a data frame (e.g., making new variables), and want to save the data after the alterations.

To save an object in *R*, you use the `write.table()` function. Because we are saving an object out of the *R* environment, we do not need to assign this function to an object–rather we can just run the line of code directly, and *R* will save the object to our working directory. To use the `write.table()` function, you need to tell *R* three things: (1) the name of the object to save, (2) the name of the file, with extension, you want to give this object, and (3) what delimiter (recall, a delimiter is a chracter used to seperate values in a file) to use to separate the values in the data file. As noted above, I like to use commas to separate values as these files tend to read well across software platforms.

For example, suppose we wanted to save the data frame `Data1` after computing the sum scores and the mean scores. Having set our working directory earlier, we can simply specify the object name to save, the file name we want to give it, and the fact that it is a comma separated values file:

```
write.table(Data1, "Dataset 1--NEW.csv", sep=",")
```

This code tells *R* to save the object `Data1` as `"Dataset 1--NEW.csv"`, as a comma separated values file, in the working directory. Note that the file name is in quotation makes and includes the file extension (i.e., .csv). Also note that the delimiter is also in quotation makes (i.e., `=","`).

3 Univariate Statistical Analyses

When I first started to use *R*, I went to a colleague, Jay, and ask for help. At this time, *R* was relatively new, so there were not a ton of packages available, but there was a package written to run multidimensional item response theory. The fact that a free program was avaialble to run this analysis was exciting because, at the time, the only program available for it was rather expensive. The specifics of this analysis/technique is not important. What is important is what Jay said to me when I walked into his office to ask for help. He commented that no one ever asked him for help with how to use *R* to get descriptive statistics; people always want to jump into the complex analyses without learning the basics first.

Jay's comment proved prescient insofar as, despite his dedicated help, I never fully understood what I was doing in *R*.[9] However, once I learned how to use *R* to produce descriptive statistics and conduct univariate statistical analyses, I was able to rapidly expand my use of *R* to the more complex applications. As such, although your research and/or consulting might necessitate more advanced statistical methods, I encourage you to work through these next few sections slowly to fully grasp how to get *R* to produce the output from univariate statistics. Once you have a grasp of these, you will find the next volumes considerably easier to follow.

Before turning to these analyses, I want to remind you that we will be covering how to produce to the output for each of these, but not necessarily elaborating in depth the interpretation of each. I leave the detailed interpretation of the output to other textbooks. Here, I will explain how to code the analysis in *R* to produce the output for interpretation. Also, whenever possible, I rely on existing packages to produce output rather than trying to individually code functions. Whereas some instructors may view this as a disservice to students who may better appreciate statistics by coding the analysis themselves, rather than relying on others' coding, the goal of these volumes is to help individuals transition to using *R*.

[9] Full disclosure–there are days when I still don't think I know what I'm doing…

3.1 About the Data Sets

Throughout these volumes, I make use of existing data sets available online, from my own research papers, or data that I simulate to demonstrate some technique. In each case, I'll note the source of the data. Although this means you get to practice on real data, it also means that you may have to load into the *R* environment different data sets for different analyses. It also means that not all of the analyses we perform will be statistically significant–that's not how real data work; real data fail sometimes (read: most of the time). Furthermore, I'll explain or describe each data set each time, but admittedly only briefly.

To demonstrate univariate statistics we will use the end of season player statistics for National Basketball Association (NBA) players in the 2022-2023 season for those players who played in at least ten games.[10] In addition to truncated game statistics, I added a column of data indicating if the player plays in a back court versus front court position (i.e., BcFc), and if the team played for a team in the Eastern Conference, Western Conference, or multiple teams (i.e., Conf).[11]

Knowing that we already set a working directory, we start by importing this data file into *R* and assigning it an object–for simplicity, we can name the object `bball`.

```
bball <- read.csv("BBall 22-23 Working.csv", header=TRUE)
```

We now have an object in our *R* environment named `bball` that is a data frame of 470 rows (players) by 27 columns (variables/player statistics). The table below briefly explains each variable in the data set.[12]

[10] These data were obtained from Basketball Reference (https://www.basketball-reference.com/leagues/NBA_2023_per_game.html) retrieved 1/18/2024.

[11] For readers unfamiliar with the NBA and/or Basketball I will provide a brief overview of these two variables. First, back court versus front court can, in an admittedly oversimplification, designate positions on the basketball court that have, traditionally, played closer to the scoring basket (i.e., front court) versus those that play further way from the scoring basket (i.e., back court). Front court positions include the Small Forward, Power Forward, and Center; the back court positions include the Shooting Guard and Point Guard.

The 30 teams in the NBA are divided into two subgroups of 15 teams, known as Conferences, for the purposes of rank ordering teams for positions in the NBA end of season Playoff Tournament. Ten teams from each conference make the final playoff brackets (with four teams in each conference needing to play in an entry sub-tournament known as the play-in tournament). Some players changed teams in the middle of the season, and this change could have been within or across conferences. As such, these players are indicated as a different group.

[12] If you are not a basketball fan some of these variables may seem confusing. Fortunately, you do not need to really like or understand basketball to use these data in this demonstration. Just follow along, and trust me that the analyses make at least a little sense. The only things you really need to know about basketball are (1) the Chicago Bulls are the best basketball team in all the land, and (2) Michael Jordan is the best basketball player of all time.

If you are an avid basketball fan, you may find some of the comparisons/"research questions" we run a little wanting. Go with it as we are just trying to demonstrate the techniques, not break into basketball analytics to predict the best basketball player or team–indeed, we established that the answer to these questions are (1) Michael Jordan and (2) the Chicago Bulls, respectively. We can still analyze the data, though.

Table 1: Brief Descriptions of Variables in "BBall 22-23 Working.csv" Data File. Data from the 2022-2023 NBA season.

Column Number	Variable Name	Brief Description
1	Rk	Player ID based on last name alphabetizing.
2	Player	Name of player.
3	Pos	Player primary position: C-Center, PF-Power Forward, SF-Small Forward, SG-Shooting Guard, PG-Point Guard.
4	Age	Player age during season.
5	FcBc	Whether player's primary position was front court (FC) or back court (BC) position.
6	Tm	Player team: TOT representing total across multiple teams.
7	Conf	Player's team as Eastern (1) or Western (2) conference, or played across multiple teams (3).
8	G	Total games played in during season.
9	GS	Total games in which player started the game in season.
10	MP	Total minutes player played in season.
11	FG	Average number of field goals made by player in season.
12	FGPerc	Average percentage of field goals made by player in season.
13	X3p	Average number of 3-point shots made by player in season.
14	X3Perc	Average percentage of 3-point shots made by player in season.
15	X2p	Average number of 2-point shots made by player in season.
16	X2Perc	Average percentage of 2-point shorts made by player in season.
17	FT	Average number of free throws made by player in season.
18	FTPerc	Average percentage of free throws made by player in season.
19	ORB	Average number of offensive rebounds by player in season.
20	DRB	Average number of defensive rebounds by player in season.

Column Number	Variable Name	Brief Description
21	TRB	Average number of total rebounds by player in season.
22	AST	Average number of assists per game by player in season.
23	STL	Average number of steals per game by player in season.
24	BLK	Average number of blocks per game by player in season.
25	TOV	Average number of turnovers per game by player in season.
26	PF	Average number of personal fouls per game by player in season.
27	PTS	Average number of points scored per game by player in season.

3.2 Descriptive Statistics

Descriptive statistics are used to get a sense of your data. Typical descriptive statistics include central tendency and variability, and are helpful for things like data cleaning, checking assumptions, and giving readers a sense of your data.

Producing descriptive statistics in *R* is straightforward thanks to the `describe()` and `describeBy()` functions in the *psych* package. Whereas the `describe()` function produces the descriptive statistics for all referenced variables, `describeBy()` subsets the descriptive statistics *by* some grouping variable–that is, you can produce descriptive statistics separately for different groups (e.g., conditions of an experiments, gender, etc.).

With our data in *R*, we can use the different `describe()` functions in the *psych* package. First, we'll use `describe()` to ask for the full list of descriptive statistics. To use this function, we need to specify the variables we want to describe.

```
describe(bball$PTS, na.rm=TRUE)
```

Here, we are asking for the descriptive statistics for just the `PTS` variable. We are also telling *R* to remove missing values before computing the descriptive statistics. Before producing the output, I will share four variations of this code:

1) `describe(bball[,27], na.rm=TRUE)`

2) `describe(cbind(bball$PTS, bball$TOV, bball$MP), na.rm=TRUE)`

3) `describe(bball[,c(27,25,10)], na.rm=TRUE)`

4) `describe(bball, na.rm=TRUE)`

Variation #1 of the code does the same as the original code, but does so by referencing a specific column of data, rather than referring to a specific variable name. As such, the original code and variation #1 will produce the same output: the descriptive statistics for `PTS`.

Variations #2 and #3 do the same thing: they both produce descriptive statistics for three different variables: `PTS`, `TOV`, and `MP`. Whereas Variation #2 references the specific variable names, Variation #3 references the specific columns. Note that Variation #2 nests the `cbind()` function within the `describe()` function–this tells *R* to bind the columns that follow together. In Variation #3, this is accomplished by nesting the `c()` function inside of the `describe()` function–this tells *R* to collect/combine the three (or however many) variables together.

Finally, Variation #4 will print the descriptive statistics for every column in the data frame. When the number of columns in the data frame is relatively small Variation #4 might be the best approach; however, this variation becomes cumbersome when the number of variables to analyze is large (as is the case here, so we will not produce the actual output for Variation #4).

```
describe(bball$PTS, na.rm=TRUE)
```

```
   vars   n mean   sd median trimmed  mad min  max range skew kurtosis   se
X1    1 470 9.86 6.87   7.95    8.94 5.41 0.4 33.1  32.7 1.16     0.77 0.32
```

```
describe(bball[,27], na.rm=TRUE)
```

```
   vars   n mean   sd median trimmed  mad min  max range skew kurtosis   se
X1    1 470 9.86 6.87   7.95    8.94 5.41 0.4 33.1  32.7 1.16     0.77 0.32
```

```
describe(cbind(bball$PTS, bball$TOV, bball$MP), na.rm=TRUE)
```

```
   vars   n  mean   sd median trimmed   mad min  max range skew kurtosis   se
X1    1 470  9.86 6.87   7.95    8.94  5.41 0.4 33.1  32.7 1.16     0.77 0.32
X2    2 470  1.17 0.81   0.90    1.06  0.59 0.0  4.1   4.1 1.21     0.94 0.04
X3    3 470 21.07 9.01  20.20   21.09 11.19 2.5 37.4  34.9 0.04    -1.14 0.42
```

```
describe(bball[,c(27,25,10)], na.rm=TRUE)
```

```
    vars   n  mean   sd median trimmed   mad min  max range skew kurtosis   se
PTS    1 470  9.86 6.87   7.95    8.94  5.41 0.4 33.1  32.7 1.16     0.77 0.32
TOV    2 470  1.17 0.81   0.90    1.06  0.59 0.0  4.1   4.1 1.21     0.94 0.04
MP     3 470 21.07 9.01  20.20   21.09 11.19 2.5 37.4  34.9 0.04    -1.14 0.42
```

As the output shows, there is a lot of descriptive statistics produced for these data, and most of these are self-explanatory based on the column names. Two pieces of output that might not be clear are *trimmed* and *mad*. The former refers to the trimmed mean of the variable. How much the mean is trimmed is something that you can set within the `describe()` function– see the help files associated with this function for how to do this. *mad* refers to the median absolute deviation from the median.

To pare down the amount of output, you can add `fast=TRUE` to your `describe()` function. Setting this to `TRUE`, note all letter capitalized in TRUE, the function will only print *n, mean, sd, min, max,* and *range*:

```
describe(bball[,10:15], na.rm=TRUE, fast=TRUE)
```

```
        vars   n  mean   sd median  min   max range  skew kurtosis   se
MP         1 470 21.07 9.01  20.20 2.50 37.40 34.90  0.04    -1.14 0.42
FG         2 470  3.62 2.43   3.00 0.20 11.20 11.00  1.05     0.42 0.11
FGPerc     3 470  0.47 0.08   0.46 0.15  0.82  0.67  0.70     2.03 0.00
X3P        4 470  1.06 0.88   0.90 0.00  4.90  4.90  0.99     0.92 0.04
X3Perc     5 462  0.33 0.11   0.35 0.00  1.00  1.00 -0.37     6.71 0.01
X2P        6 470  2.56 2.00   1.90 0.10 10.50 10.40  1.32     1.53 0.09
```

In this output, I have requested the truncated amount of descriptive statistics for `mp`, `FG`, `FGPerc`, `X3Pp`, `X3Perc`, and `X2p`. Note that in the code I specified all six columns by using the `:` between the first and last column to display; using this type of coding will tell *R* to apply the function across all columns from the first number through the last number. As such, this short hand of coding can only be used for consecutive columns.

Finally, we can produce descriptive statistics for different subgroups in our data using the `describeBy()` function. This function works similarly to the `describe()` function with the added code for indicating the variable name for the different groups:

```
describeBy(bball[,c(10:15)], group=bball$Pos, fast=TRUE)
```

```
 Descriptive statistics by group
group: C
        vars  n  mean   sd median  min   max range  skew kurtosis   se
MP         1 99 18.57 8.57  18.30 2.50 34.60 32.10  0.26    -1.09 0.86
FG         2 99  3.29 2.22   2.60 0.20 11.00 10.80  1.16     1.04 0.22
FGPerc     3 99  0.57 0.10   0.55 0.25  0.82  0.57 -0.13     0.74 0.01
X3P        4 99  0.37 0.53   0.10 0.00  2.30  2.30  1.64     1.97 0.05
X3Perc     5 91  0.28 0.20   0.33 0.00  1.00  1.00  0.67     1.94 0.02
X2P        6 99  2.92 2.05   2.50 0.20 10.00  9.80  1.21     1.33 0.21
```

```
------------------------------------------------------------
group: PF
       vars  n  mean   sd median  min   max range  skew kurtosis   se
MP        1 90 21.46 8.71  20.70 5.60 37.40 31.80  0.08    -1.16 0.92
FG        2 90  3.62 2.53   2.80 0.50 11.20 10.70  1.28     0.88 0.27
FGPerc    3 90  0.48 0.06   0.47 0.38  0.66  0.28  0.61    -0.14 0.01
X3P       4 90  0.95 0.69   0.80 0.00  3.00  3.00  0.91     0.28 0.07
X3Perc    5 90  0.32 0.09   0.34 0.00  0.44  0.44 -1.92     4.28 0.01
X2P       6 90  2.66 2.21   1.80 0.30 10.50 10.20  1.52     1.94 0.23
------------------------------------------------------------
group: PG
       vars  n  mean   sd median  min   max range  skew kurtosis   se
MP        1 82 23.28 9.43  24.35 5.20 37.40 32.20 -0.20    -1.26 1.04
FG        2 82  4.18 2.78   3.70 0.50 10.90 10.40  0.70    -0.60 0.31
FGPerc    3 82  0.43 0.05   0.43 0.24  0.57  0.32 -0.27     1.29 0.01
X3P       4 82  1.36 1.01   1.15 0.00  4.90  4.90  1.10     1.07 0.11
X3Perc    5 82  0.34 0.07   0.34 0.00  0.47  0.47 -1.77     6.25 0.01
X2P       6 82  2.82 2.10   2.10 0.10  9.50  9.40  1.00     0.40 0.23
------------------------------------------------------------
group: SF
       vars  n  mean   sd median  min   max range  skew kurtosis   se
MP        1 89 22.03 8.86  20.80 3.90 36.90 33.00 -0.13    -1.08 0.94
FG        2 89  3.60 2.33   3.10 0.20 10.10  9.90  0.95     0.20 0.25
FGPerc    3 89  0.44 0.07   0.45 0.15  0.57  0.42 -1.43     4.33 0.01
X3P       4 89  1.31 0.87   1.10 0.00  4.40  4.40  0.87     0.77 0.09
X3Perc    5 89  0.34 0.07   0.35 0.00  0.45  0.45 -1.90     5.37 0.01
X2P       6 89  2.29 1.87   1.70 0.10  8.30  8.20  1.28     1.14 0.20
------------------------------------------------------------
group: SG
       vars   n  mean   sd median min   max range  skew kurtosis   se
MP        1 110 20.57 9.02  20.05 4.1 36.40 32.30  0.07    -1.09 0.86
FG        2 110  3.50 2.30   3.05 0.4 10.00  9.60  0.98     0.20 0.22
FGPerc    3 110  0.44 0.05   0.44 0.3  0.59  0.29 -0.13     0.73 0.00
X3P       4 110  1.33 0.83   1.30 0.0  3.60  3.60  0.49    -0.51 0.08
X3Perc    5 110  0.35 0.08   0.37 0.0  0.50  0.50 -1.82     5.17 0.01
X2P       6 110  2.18 1.70   1.70 0.1  7.80  7.70  1.26     1.03 0.16
```

Here, we see that the basic descriptive statistics for `MP`, `FG`, `FGPerc`, `X3Pp`, `X3Perc`, and `X2p` are printed separately for each of the five different positions. If we omitted `fast=TRUE`, then we would print the full set of descriptive statistics for each group. Like the variables to be analyzed, we can specify the grouping variable using the column number referencing method, `group=bball[,3]`. Finally, you can look at descriptive statistics across collections of grouping variables by including more than one grouping variable nesting `c()` functions within the

`describeBy()` function. For instance, this code will produce the basic descriptive statistics for the cross sections of `Conf` and `FcBc`:

```
describeBy(bball[,c(10:15)], group=bball[,c(5,7)], fast=TRUE)
```

```
 Descriptive statistics by group
FcBc: BC
Conf: 1
       vars  n  mean   sd median  min   max range  skew kurtosis   se
MP        1 81 22.93 9.44  24.70 4.10 36.80 32.70 -0.26    -1.17 1.05
FG        2 81  3.90 2.43   3.50 0.50 10.00  9.50  0.67    -0.64 0.27
FGPerc    3 81  0.44 0.04   0.43 0.34  0.57  0.23  0.25     0.39 0.00
X3P       4 81  1.36 0.91   1.30 0.00  4.00  4.00  0.53    -0.34 0.10
X3Perc    5 81  0.34 0.08   0.36 0.00  0.44  0.44 -2.06     5.48 0.01
X2P       6 81  2.55 1.79   1.90 0.30  7.30  7.00  0.91    -0.19 0.20
------------------------------------------------------------
FcBc: FC
Conf: 1
       vars   n  mean   sd median  min   max range skew kurtosis   se
MP        1 120 21.25 9.04  19.90 2.50 37.40 34.90 0.06    -1.11 0.83
FG        2 120  3.59 2.50   2.80 0.20 11.20 11.00 1.04     0.39 0.23
FGPerc    3 120  0.49 0.09   0.48 0.18  0.78  0.59 0.25     1.20 0.01
X3P       4 120  0.88 0.82   0.70 0.00  3.60  3.60 0.86     0.11 0.07
X3Perc    5 116  0.30 0.14   0.33 0.00  1.00  1.00 0.22     4.16 0.01
X2P       6 120  2.72 2.20   1.80 0.10 10.50 10.40 1.22     1.12 0.20
------------------------------------------------------------
FcBc: BC
Conf: 2
       vars  n  mean   sd median  min   max range  skew kurtosis   se
MP        1 81 21.00 9.34  20.20 4.40 36.30 31.90  0.05    -1.19 1.04
FG        2 81  3.89 2.77   3.10 0.40 10.90 10.50  0.89    -0.23 0.31
FGPerc    3 81  0.44 0.05   0.44 0.30  0.54  0.24 -0.33    -0.24 0.01
X3P       4 81  1.31 0.96   1.00 0.00  4.90  4.90  1.23     1.60 0.11
X3Perc    5 81  0.35 0.07   0.36 0.12  0.50  0.38 -1.07     1.82 0.01
X2P       6 81  2.59 2.09   2.00 0.10  9.50  9.40  1.20     1.01 0.23
------------------------------------------------------------
FcBc: FC
Conf: 2
       vars   n  mean   sd median  min   max range  skew kurtosis   se
MP        1 120 20.89 8.90  20.80 3.90 35.70 31.80 -0.09    -1.20 0.81
FG        2 120  3.61 2.35   2.90 0.20 11.10 10.90  1.07     0.51 0.21
FGPerc    3 120  0.50 0.10   0.49 0.15  0.82  0.67  0.32     1.47 0.01
```

```
X3P       4 120  0.87 0.83   0.70 0.00  4.40  4.40  1.33     2.00 0.08
X3Perc    5 118  0.32 0.13   0.35 0.00  1.00  1.00  0.10     6.72 0.01
X2P       6 120  2.74 2.03   2.30 0.20  9.60  9.40  1.35     1.62 0.19
------------------------------------------------------------
FcBc: BC
Conf: 3
       vars  n  mean   sd median  min   max range  skew kurtosis   se
MP        1 30 20.45 8.50  18.20 7.20 37.40 30.20  0.36    -0.98 1.55
FG        2 30  3.22 2.10   2.45 0.50  9.90  9.40  1.27     1.42 0.38
FGPerc    3 30  0.43 0.06   0.43 0.24  0.59  0.35 -0.27     2.40 0.01
X3P       4 30  1.40 0.77   1.20 0.30  3.10  2.80  0.51    -0.71 0.14
X3Perc    5 30  0.37 0.04   0.37 0.26  0.50  0.24  0.97     2.58 0.01
X2P       6 30  1.82 1.53   1.40 0.10  6.80  6.70  1.59     2.15 0.28
------------------------------------------------------------
FcBc: FC
Conf: 3
       vars  n  mean   sd median  min   max range  skew kurtosis   se
MP        1 38 17.73 7.35  15.50 6.50 35.60 29.10  0.61    -0.61 1.19
FG        2 38  2.83 1.75   2.40 0.80 10.30  9.50  2.03     6.17 0.28
FGPerc    3 38  0.48 0.08   0.46 0.36  0.68  0.32  0.65    -0.40 0.01
X3P       4 38  0.79 0.65   0.75 0.00  2.50  2.50  0.74    -0.29 0.11
X3Perc    5 36  0.33 0.11   0.34 0.00  0.59  0.59 -1.14     2.53 0.02
X2P       6 38  2.04 1.57   1.55 0.20  8.30  8.10  1.93     4.69 0.25
```

Here you see the descriptive statics within the six combinations that are produced when you cross the two levels of `FcBc` with the three levels of `Conf`.

3.3 Variances, Covariances, and Correlations

Having produced descriptive statistics, we can now turn to measures of association–namely, correlations. In addition to computing the correlation between two variables, we can compute variance of a variable, and the covariance of two variables.

To compute the variance of a variable, we use the `var()` function. To utilize this function, we need to specify a column of data for which we would like to compute the variance. In addition, we need to specify what to do with missing values: `na.rm=`; setting this to `TRUE` will remove missing data before computing the variance. To compute the variance of average points per game scored in the season, we would include `bball$PTS` in the `var()` function.

```
var(bball$PTS, na.rm=TRUE)
```

```
[1] 47.19653
```

To compute the covariance of two variables, we use the `cov()` function. Here, we need to specify two variables on which to compute the covariance, and a `use=` command. The `use=` code tells *R* what observations to use. Specifying `use="pairwise.complete.obs"`, in quotation marks, tells *R* to engaged in pairwise deletion–this means any case for which both variables are non-missing data will be used to compute the covariance. If we want to compute the covariance of average points per game scores in a season and the average minutes a game played in a season:

```
cov(bball$PTS, bball$MP, use="pairwise.complete.obs")
```

```
[1] 54.27879
```

Finally, we can compute correlations among variables in multiple ways, but I'll demonstrate two specific approaches: `cor.test()`, and `rcorr()` from the *Hmisc* pacakge (Harrell, 2022).

The `cor.test()` function requires specifying (1) the two variables to be correlated, (2) and `alternative=` representing if the null hypothesis significance testing should be two sided, less than 0, or greater than 0, and, among other specifications not detailed here, (3) the `method=` as either a Pearson correlation, or a Kendall or Spearman rank correlation. For our demonstration, we will simply compute the Pearson Product Moment Correlation between `PTS` and `MP` with a `"two.sided"` test:

```
cor.test(bball$PTS, bball$MP, alternative = "two.sided", method = "pearson")
```

```
    Pearson's product-moment correlation

data:  bball$PTS and bball$MP
t = 39.486, df = 468, p-value < 2.2e-16
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 0.8543198 0.8963478
sample estimates:
      cor
0.8770007
```

Here, we see that the correlation between average minutes a game played and average points per game score is, as one might expect, fairly strong, $r\ (468) = .88$, $p < .001$. If we wanted to use a one-tailed test, we can change `"two.sided"` to either `"greater"` or `"less"`. In addition, we could change from `"pearson"` to either `"kendall"` or `"spearman"`. It is important, of course, for you, as the analyst, to know which correlation coefficient is most appropriate for your data.

Most researchers are less interested in computing single bivariate correlations; rather, they want to compute a full matrix of correlations among their variables. The `rcorr()` function from the *Hmisc* package (Harrell, 2022) can be used for this. This function will compute all sets of bivariate correlations among the columns of a matrix. To that end, the `rcorr()` function requires specifying a matrix format of your data–this can be achieved by wrapping the `rcorr()` function around the `as.matrix()` function. This latter function will tell *R* to convert the data included within the parentheses to a matrix format. Other than specifying the matrix of data to analyze, you will need to tell the `rcorr()` function what `type=` of correlation to run, `"pearson"` or `"spearman"`.

Let's compute the correlation matrix for the variables starting from `MP` to `PTS`. Based on the code above about referencing columns, we know that this would be columns 10 through 27. Therefore, we can tell *R* to compute the correlations of columns in `bball` 10-27 as a matrix. First, though, we have to install[13] and load the *Hmisc* library.

```
library(Hmisc)

rcorr(as.matrix(bball[,10:27]), type="pearson")
```

```
          MP   FG FGPerc   X3P X3Perc   X2P X2Perc   FT FTPerc   ORB   DRB   TRB
MP      1.00 0.88   0.14  0.67   0.16  0.78   0.11 0.72   0.26  0.33  0.72  0.65
FG      0.88 1.00   0.21  0.63   0.13  0.94   0.13 0.87   0.24  0.31  0.73  0.65
FGPerc  0.14 0.21   1.00 -0.28  -0.11  0.38   0.79 0.17  -0.28  0.63  0.43  0.52
X3P     0.67 0.63  -0.28  1.00   0.40  0.32  -0.17 0.45   0.42 -0.22  0.25  0.12
X3Perc  0.16 0.13  -0.11  0.40   1.00 -0.01  -0.15 0.09   0.29 -0.28 -0.05 -0.12
X2P     0.78 0.94   0.38  0.32  -0.01  1.00   0.23 0.86   0.11  0.47  0.77  0.73
X2Perc  0.11 0.13   0.79 -0.17  -0.15  0.23   1.00 0.09  -0.21  0.46  0.32  0.39
FT      0.72 0.87   0.17  0.45   0.09  0.86   0.09 1.00   0.26  0.26  0.62  0.54
FTPerc  0.26 0.24  -0.28  0.42   0.29  0.11  -0.21 0.26   1.00 -0.28 -0.01 -0.10
ORB     0.33 0.31   0.63 -0.22  -0.28  0.47   0.46 0.26  -0.28  1.00  0.69  0.84
DRB     0.72 0.73   0.43  0.25  -0.05  0.77   0.32 0.62  -0.01  0.69  1.00  0.97
TRB     0.65 0.65   0.52  0.12  -0.12  0.73   0.39 0.54  -0.10  0.84  0.97  1.00
AST     0.72 0.72  -0.01  0.51   0.13  0.64  -0.08 0.65   0.23  0.05  0.46  0.36
STL     0.74 0.62   0.03  0.46   0.10  0.55  -0.03 0.51   0.19  0.20  0.44  0.39
BLK     0.34 0.33   0.50 -0.08  -0.13  0.44   0.38 0.28  -0.18  0.64  0.59  0.65
TOV     0.79 0.87   0.13  0.51   0.09  0.83   0.02 0.81   0.19  0.25  0.66  0.58
PF      0.72 0.60   0.32  0.28   0.01  0.60   0.26 0.49   0.06  0.55  0.69  0.69
PTS     0.88 0.99   0.16  0.68   0.17  0.91   0.09 0.91   0.29  0.25  0.69  0.60
          AST   STL   BLK  TOV   PF  PTS
MP       0.72  0.74  0.34 0.79 0.72 0.88
FG       0.72  0.62  0.33 0.87 0.60 0.99
```

[13] As I've already installed the packages we need for this series of books, I skip the actual code for installing any packages moving forward. Refer to the previous sections on how to install packages if you need a refresher.

```
FGPerc -0.01  0.03  0.50 0.13 0.32 0.16
X3P     0.51  0.46 -0.08 0.51 0.28 0.68
X3Perc  0.13  0.10 -0.13 0.09 0.01 0.17
X2P     0.64  0.55  0.44 0.83 0.60 0.91
X2Perc -0.08 -0.03  0.38 0.02 0.26 0.09
FT      0.65  0.51  0.28 0.81 0.49 0.91
FTPerc  0.23  0.19 -0.18 0.19 0.06 0.29
ORB     0.05  0.20  0.64 0.25 0.55 0.25
DRB     0.46  0.44  0.59 0.66 0.69 0.69
TRB     0.36  0.39  0.65 0.58 0.69 0.60
AST     1.00  0.67  0.05 0.84 0.41 0.72
STL     0.67  1.00  0.20 0.61 0.53 0.62
BLK     0.05  0.20  1.00 0.24 0.53 0.29
TOV     0.84  0.61  0.24 1.00 0.60 0.87
PF      0.41  0.53  0.53 0.60 1.00 0.57
PTS     0.72  0.62  0.29 0.87 0.57 1.00

n
        MP  FG FGPerc X3P X3Perc X2P X2Perc  FT FTPerc ORB DRB TRB AST STL BLK
MP     470 470    470 470    462 470    470 470    467 470 470 470 470 470 470
FG     470 470    470 470    462 470    470 470    467 470 470 470 470 470 470
FGPerc 470 470    470 470    462 470    470 470    467 470 470 470 470 470 470
X3P    470 470    470 470    462 470    470 470    467 470 470 470 470 470 470
X3Perc 462 462    462 462    462 462    462 462    459 462 462 462 462 462 462
X2P    470 470    470 470    462 470    470 470    467 470 470 470 470 470 470
X2Perc 470 470    470 470    462 470    470 470    467 470 470 470 470 470 470
FT     470 470    470 470    462 470    470 470    467 470 470 470 470 470 470
FTPerc 467 467    467 467    459 467    467 467    467 467 467 467 467 467 467
ORB    470 470    470 470    462 470    470 470    467 470 470 470 470 470 470
DRB    470 470    470 470    462 470    470 470    467 470 470 470 470 470 470
TRB    470 470    470 470    462 470    470 470    467 470 470 470 470 470 470
AST    470 470    470 470    462 470    470 470    467 470 470 470 470 470 470
STL    470 470    470 470    462 470    470 470    467 470 470 470 470 470 470
BLK    470 470    470 470    462 470    470 470    467 470 470 470 470 470 470
TOV    470 470    470 470    462 470    470 470    467 470 470 470 470 470 470
PF     470 470    470 470    462 470    470 470    467 470 470 470 470 470 470
PTS    470 470    470 470    462 470    470 470    467 470 470 470 470 470 470
       TOV  PF PTS
MP     470 470 470
FG     470 470 470
FGPerc 470 470 470
X3P    470 470 470
X3Perc 462 462 462
```

```
X2P    470 470 470
X2Perc 470 470 470
FT     470 470 470
FTPerc 467 467 467
ORB    470 470 470
DRB    470 470 470
TRB    470 470 470
AST    470 470 470
STL    470 470 470
BLK    470 470 470
TOV    470 470 470
PF     470 470 470
PTS    470 470 470

P
       MP     FG     FGPerc X3P    X3Perc X2P    X2Perc FT     FTPerc ORB
MP             0.0000 0.0020 0.0000 0.0007 0.0000 0.0221 0.0000 0.0000 0.0000
FG     0.0000         0.0000 0.0000 0.0038 0.0000 0.0048 0.0000 0.0000 0.0000
FGPerc 0.0020 0.0000         0.0000 0.0217 0.0000 0.0000 0.0001 0.0000 0.0000
X3P    0.0000 0.0000 0.0000         0.0000 0.0000 0.0002 0.0000 0.0000 0.0000
X3Perc 0.0007 0.0038 0.0217 0.0000         0.7593 0.0017 0.0462 0.0000 0.0000
X2P    0.0000 0.0000 0.0000 0.0000 0.7593         0.0000 0.0000 0.0160 0.0000
X2Perc 0.0221 0.0048 0.0000 0.0002 0.0017 0.0000         0.0599 0.0000 0.0000
FT     0.0000 0.0000 0.0001 0.0000 0.0462 0.0000 0.0599         0.0000 0.0000
FTPerc 0.0000 0.0000 0.0000 0.0000 0.0000 0.0160 0.0000 0.0000         0.0000
ORB    0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
DRB    0.0000 0.0000 0.0000 0.0000 0.3267 0.0000 0.0000 0.0000 0.7647 0.0000
TRB    0.0000 0.0000 0.0000 0.0107 0.0084 0.0000 0.0000 0.0000 0.0305 0.0000
AST    0.0000 0.0000 0.7929 0.0000 0.0050 0.0000 0.0693 0.0000 0.0000 0.3265
STL    0.0000 0.0000 0.4652 0.0000 0.0302 0.0000 0.4874 0.0000 0.0000 0.0000
BLK    0.0000 0.0000 0.0000 0.0727 0.0048 0.0000 0.0000 0.0000 0.0000 0.0000
TOV    0.0000 0.0000 0.0056 0.0000 0.0675 0.0000 0.6032 0.0000 0.0000 0.0000
PF     0.0000 0.0000 0.0000 0.0000 0.8451 0.0000 0.0000 0.0000 0.1791 0.0000
PTS    0.0000 0.0000 0.0007 0.0000 0.0003 0.0000 0.0499 0.0000 0.0000 0.0000
       DRB    TRB    AST    STL    BLK    TOV    PF     PTS
MP     0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
FG     0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
FGPerc 0.0000 0.0000 0.7929 0.4652 0.0000 0.0056 0.0000 0.0007
X3P    0.0000 0.0107 0.0000 0.0000 0.0727 0.0000 0.0000 0.0000
X3Perc 0.3267 0.0084 0.0050 0.0302 0.0048 0.0675 0.8451 0.0003
X2P    0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
X2Perc 0.0000 0.0000 0.0693 0.4874 0.0000 0.6032 0.0000 0.0499
FT     0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
```

```
FTPerc 0.7647 0.0305 0.0000 0.0000 0.0000 0.0000 0.1791 0.0000
ORB    0.0000 0.0000 0.3265 0.0000 0.0000 0.0000 0.0000 0.0000
DRB           0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
TRB    0.0000        0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
AST    0.0000 0.0000        0.0000 0.3201 0.0000 0.0000 0.0000
STL    0.0000 0.0000 0.0000        0.0000 0.0000 0.0000 0.0000
BLK    0.0000 0.0000 0.3201 0.0000        0.0000 0.0000 0.0000
TOV    0.0000 0.0000 0.0000 0.0000 0.0000        0.0000 0.0000
PF     0.0000 0.0000 0.0000 0.0000 0.0000 0.0000        0.0000
PTS    0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
```

The first thing you may notice that *R* is telling you that certain objects are "...masked from..." then a package. What is happening here is that the *Hmisc* package has functions with the same name as *psych*, `describe()`, and the *base* package, `format.pval()` and `unit()`. In this case, if you call any of these three functions, it will use the function for *Hmisc* because it is the most recent package loaded.[14]

Next, you'll see that *R* prints the correlation matrix (as a full square matrix, with 1.0 along the diagonals) for all 18 variables. This is followed by a matrix of sample sizes used to compute each variable. If there were no missing data, this matrix of sample sizes would reduce to a single number representing the sample size used to compute all of the correlations. Lastly, you'll see the p-values associated with each bivariate correlation.

So, for example, the bivariate correlation between average minutes played per game, `MP`, and average number of turnovers committed a game, `TOV`, is $r = .79$, which was computed using 470 respondents, and the p-value associated with this correlation is $p < .0001$. Knowing that the degrees of freedom for the statistical test of whether or not a Pearson correlation is significantly different than 0 is $df = N - 2$, we can report that $r(468) = .79$, $p < .0001$.

3.4 Regression

From computing measures of association, we can shift to conducting regression analyses. Regression is a technique wherein one or more variables are used to predict standing on another variable. For instance, we might use a combination of minutes played (`MP`), average rebounds per game (`TRB`), and average assists per game (`AST`) to predict points per game (`PTS`). To do this, we will make use of the linear model, `lm()`, function. We will first discuss simultaneous entry regression, followed by hierarchical entry. Finally, I will show you how to integrate categorical variables into regression models using dummy coding.

[14] If you want to use a function from a specific package, you can tell *R* to call a specific package and use the function from that package by including the name of the package, followed by `::`, and then the function as you would normally specify it. So, for instance, if I wanted to use the *psych* package to compute descriptive statistics on `PTS`, I would use `psych::describe(bball$PTS)`.

3.4.1 Simultaneous Entry

To use this function, one must specify a linear equation relating the outcome variable to the predictor(s). This is accomplished by specifying the outcome variable, the thing to be predicted, on the left of a `~`, and the predictor variables to the right of it. In *R*, we use the `~` symbol as an equal sign therein telling *R* that the variable to the left is equal to the expression of variables on the right.

```
PTS ~ MP + TRB + AST
```

Note that the predictors are specified as additive. Therefore, *R* tries to compute the linear model in which average points per game equals, `~`, the linear, weighted sum of average minutes played, average total rebouds per game, and average assists per game. For simple linear regression (i.e., regression with only one predictor), we would just include only one predictor variable to the right of the `~`.

In addition to specifying the linear equation, we need to tell *R* what data frame contains our variables, `data=`, and how to handle missing data, `na.action=`, wherein `na.exlude` will remove cases for which there is missing data on any of the variables. Lastly, we need to define the `lm()` output as an object so that we can review the `summary()` of the output:

```
Model1 <- lm(PTS ~ MP + TRB + AST, data=bball, na.action=na.exclude)
```

After running the above line of code, your *R* environment should not change, except for a new object named `Model1`. To print the output, we utilize the `summary()` function:

```
summary(Model1)
```

```

Call:
lm(formula = PTS ~ MP + TRB + AST, data = bball, na.action = na.exclude)

Residuals:
     Min       1Q   Median       3Q      Max
-11.0922  -1.7311  -0.0342   1.5452  13.1040

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -3.50637    0.38531  -9.100  < 2e-16 ***
MP           0.50470    0.02877  17.541  < 2e-16 ***
TRB          0.27679    0.08277   3.344 0.000893 ***
AST          0.76582    0.10972   6.980 1.02e-11 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
Residual standard error: 3.139 on 466 degrees of freedom
Multiple R-squared:  0.7926,    Adjusted R-squared:  0.7913
F-statistic: 593.6 on 3 and 466 DF,  p-value: < 2.2e-16
```

Here, we see that all three predictors are positively and significantly related to average points per game. The "Estimate" column provides the unstandardized regression coefficients for each predictor and the intercept. The "Std. Error" column presents the standard errors for each estimate. Finally, the "t value" and "Pr (>|T|)" columns present the t-values and p-values for each predcitor (wherein each t value is distributed on N-k-1 degrees of freedom, and N represents sample size and k represents number of predictors and is printed in the lower portion of the output).

We also see that 79.3% of the variance in average points per game is predicted by the three variables. This is displayed in the latter part of the output under "Multiple R-squared." We also see the F value associated with this R^2 in the "F-statistic" section. As such, we can see that the overall model is significant, R^2=.793, F (3, 466) = 593.60, $p < .001$.

For publication and presentation purposes, it is helpful to present the standardized regression coefficients, or β-values, and the confidence intervals for the regression coefficients. To do this, we can utilize the `lm.beta()` function from the *QuantPsyc* package (Fletcher, 2022), and the `confint()` function from base *R*. To do this, we simply specify the name of the object containing the output from the `lm()`, in this case `Model1`. In addition, for the `confint()` function, we need to specify a `level=` corresponding to the width of the confidence interval with .95 being the standard in the behavioral sciences referencing a 95% confidence interval for the raw regression coefficients.

```
library(QuantPsyc)

lm.beta(Model1)
```

```
        MP        TRB        AST
0.66184330 0.09424634 0.21532764
```

```
confint(Model1, level=.95)
```

```
                 2.5 %     97.5 %
(Intercept) -4.2635211 -2.7492197
MP           0.4481614  0.5612425
TRB          0.1141308  0.4394409
AST          0.5502190  0.9814196
```

Here we see that the `lm.beta()` produces three standardized regression coefficients for the three predictors, and the `confint()` function produces the 95% confidence intervals for the unstandardized regression coefficients.

3.4.2 Hierarchical Entry

Whereas simultaneous entry regression allows you to assess the predictive capacity of a set of variables together, hierarchical entry allows you to add variables/predictors sequentially, and assess the improvement in model fit due to the inclusion of the new predictor.

The only real difference between hierarchical entry and simultaneous entry regression is that the former estimates more than one model. Otherwise, both approaches use the same structures of `lm()`, `summary()`, `lm.beta()`, and `confint()`.

An added benefit of using this hierarchical or sequential entry approach is that we can compute an index of improved model fit, namely the change in model R^2, or ΔR^2. This index tells you the additional amount of variance in an outcome variable predicted with the inclusion of additional predictor(s). At the time of writing this edition of the book, there is no function written to automatically compute ΔR^2, so I will quickly write one here for you to use moving forward. Note, that I do not discuss in detail how to write function in *R*, but please contact me if you have any questions.

```
Delta.R2 <- function(x, y){
  test <- anova(y,x)
  ifelse(test$Df[2] < 0, print("Your model comparison ANOVA will result in negative degrees o
  Delta.R2 <- summary(x)$r.squared - summary(y)$r.squared
  R2 <- paste("Delta R^2 =", round(Delta.R2, digits=3))
  ANO <- paste("F(", test$Df[2], ",", test$Res.Df[2],")", "=", round(test$F[2], digits=3), "p
  print(R2)
  print(ANO)
  }
```

To be sure, we do not need this function to compute the change in R^2, nor to test if the change is significant, but this function will make it easier in the future.

Suppose that a researcher is interesting in answering the question of whether or not average total rebounds per game, `TRB`, and average assists per game, `AST`, significantly add to the prediction of average points per game, `PTS`, above and beyond simply average minutes played, `MP`. In short, do all three predictors do better than just the one?

Earlier, we ran the model with all three predictors, so we kind of already know the answers. However, if we suppose we didn't, we would start by running a model predicting `PTS` from just `MP`–a simple linear regression.

```
ModelMP <- lm(PTS ~ MP, data=bball, na.action=na.exclude)
summary(ModelMP)
```

```
Call:
```

```
lm(formula = PTS ~ MP, data = bball, na.action = na.exclude)

Residuals:
    Min      1Q  Median      3Q     Max
-9.3872 -1.9246 -0.1052  1.6807 14.1938

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -4.23342    0.38804  -10.91   <2e-16 ***
MP           0.66877    0.01694   39.49   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 3.304 on 468 degrees of freedom
Multiple R-squared:  0.7691,    Adjusted R-squared:  0.7686
F-statistic:  1559 on 1 and 468 DF,  p-value: < 2.2e-16
```

Here we see that average minutes played per game positively and significantly predicts average points per game, $b = .67$, $t\ (468) = 39.49$, $p < .001$, and that average minutes played per game along accounts for about 77% of the variance in average points per game, $R^2 = .77$, $F\ (1, 468) = 1559.00$, $p < .001$.

To assess if `TRB` and `AST` add to the prediction of average points per game, we run a model with all three predictors (the same model as above, but we will rerun it anyways). In addition, we can use the `Delta.R2()` function we wrote above to test to see if the addition of the two predictors significantly improves the model fit, and how much additional variance in average points per game is predicted by the additional two variables. To utilize this function, we specify the model with more predictors first, followed by model with fewer predictors.

```
ModelAll <- lm(PTS ~ MP + TRB + AST, data=bball, na.action=na.exclude)
Delta.R2(ModelAll, ModelMP)
```

```
[1] ":)"
[1] "Delta R^2 = 0.023"
[1] "F( 2 , 466 ) = 26.374 p = 1.41e-11"
```

Here we see that the inclusion of the additional two predictors increases the amount of variance predicted in average points per game by about 2.3%. We see that this is a significant increase insofar as the F value is significantly different from 0, $F\ (2, 466) = 26.37$, $p < .001$. We can now, like above, use the `summary()`, `lm.beta()`, and `confint()` function as we did above to interpret either or both models–since we did this earlier, we won't redo it here.

Before moving on to how to include categorical predictors in regression, I want to note a few things about using hierarchical entry regression with which you are likely already familiar. First, the demonstration showed how to add two predictors to a single predictor. However, any number of predictors (within reason) can be add to a model with any number of existing predictors (e.g., adding three new variables to a model that already has four variables in it). Second, the order of entry into the hierarchical entry will affect the interpretation of results. Variables entered into the model first get the account for more variance than later variables– the difference being attributable to multicollinearity (Dalal, 2023). Lastly, also because of multicollinearity, the amount of additional variance accounted for by a predictor or set of predictors is not necessarily the same as the amount of variance accounted for by a predictor alone (Dalal, 2023).

3.4.3 Categorical Predictors in Regression

Categorical predictors can be readily included in regression models by converting the nominal codes to dummy, effect, or orthogonal codes. Although the discussion of each of these types of codes is beyond the scope of this volume, I will show you have to create either dummy codes or effect codes. Then, I will show you have to include these into regression models. To simplify the discussion, we will utilize a simple linear regression model to see if the average number of foul committed per game, `PF`, is significantly predicted by whether or not a player is a front court or back court player, `FcBc`. In short, the question is if certain positions on the court lead to committing more fouls than other positions.

First, we have to convert the nominal codes of FC and BC to either dummy codes or effect codes. Dummy codes are created by specifying a reference group and assigning them a "0" on each code. All other groups are given a code of "1" in one code, and a "0" on each other code. Effect codes are created by specifying a reference group and assigning them a "-1" on each code. All other groups are given a code of "1" in one code, and a "0" on each other code.[15]

To do this, we can use *R*'s conditional statements function, `ifelse()`. To use this function, we specify three things: (1) the conditional test to evaluate, (2) what happens if the test is passed, and (3) what happens if the test is failed. In this case, the test will be if a case has a specific nominal code (e.g., `FC`). If the test is passed, this case will get a "1" on the dummy code; otherwise, the row will get a "0." For effect coding, if the test is passed, they'll get a "1" in the effect code, otherwise a "-1" in the effect code.

Note that the test in the conditional contains `==` to signify if the entry in `FcBC` is equal to `"FC"`. The conditional formatting requires the use of two equal signs - this is an *R* specific rule - otherwise, the expression cannot be validated. Some other expressions that you might use in the conditional statements include greater (`>`) or less (`<`) than, greater/less than or equal to (`>=`, `<=`), not equal to (`!=`).

[15] Recall that with dummy, effect, or orthogonal codes, you will need to create j - 1 codes, where j is the number of categories in the nominal variable.

```
bball$D1 <- ifelse(bball$FcBc == "FC", 1,0)
bball$E1 <- ifelse(bball$FcBc == "FC", 1, -1)
head(bball)[25:29]
```

```
  TOV  PF  PTS D1 E1
1 0.8 2.1 12.0  1  1
2 1.0 3.1  9.9  1  1
3 1.1 3.1 13.1  1  1
4 0.6 1.3  3.9  0 -1
5 0.2 0.6  1.2  0 -1
6 4.1 1.4 26.2  0 -1
```

Notice that we have a created two new variables in the `bball` data frame, `D1` and `E1`. We can now include one of these in a regression model to assess if front court versus back court position predicts average fouls committed per game using the `lm()` function.

```
ModelD <- lm(PF ~ D1, data=bball, na.action=na.exclude)
ModelE <- lm(PF ~ E1, data=bball, na.action=na.exclude)
summary(ModelD)
```

```

Call:
lm(formula = PF ~ D1, data = bball, na.action = na.exclude)

Residuals:
     Min       1Q   Median       3Q      Max
-1.69568 -0.51406 -0.01406  0.48594  1.90432

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  1.61406    0.05086  31.733  < 2e-16 ***
D1           0.28162    0.06614   4.258 2.49e-05 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.7048 on 468 degrees of freedom
Multiple R-squared:  0.0373,    Adjusted R-squared:  0.03524
F-statistic: 18.13 on 1 and 468 DF,  p-value: 2.491e-05
```

```
summary(ModelE)
```

```
Call:
lm(formula = PF ~ E1, data = bball, na.action = na.exclude)

Residuals:
     Min       1Q   Median       3Q      Max
-1.69568 -0.51406 -0.01406  0.48594  1.90432

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  1.75487    0.03307  53.069  < 2e-16 ***
E1           0.14081    0.03307   4.258 2.49e-05 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.7048 on 468 degrees of freedom
Multiple R-squared:  0.0373,    Adjusted R-squared:  0.03524
F-statistic: 18.13 on 1 and 468 DF,  p-value: 2.491e-05
```

Here we see that front court versus back court position does predict average fouls committed per game. You also see that, except for the regression coefficient, the results are the same. This is because with dummy coding the regression coefficient reflects the mean difference between the comparison group and the group coded "1" on that dummy code for the outcome variable. As such, we see that the average number of fouls committed per game is about .28 fouls per game higher for front court players than back court players. The regression coefficient for the intercept represents the mean of the outcome variable for the reference group–the average number of fouls per game for back court players is about 1.61 fouls per game.

The regression coefficient for the effect coding represents the difference in the mean on the outcome variable for the group coded "1" and the grand mean. As such, we see that the average number of fouls committed per game is about .14 fouls per game higher than the grand mean. The regression coefficient for the intercept represents the grand mean of the outcome variable–the average number of fouls committed per game across all players in about 1.75 fouls per game.

Although a simplistic example, this demonstration can be readily extended to situations in which more than two categories. Also, the categorical predictors can be included in a multiple regression with continuous variables as well. And, in a later volume, we will see the use of categorical variables in moderated regression.

3.5 *t*-Tests

When needing to compare one or two means, we can turn to *t*-tests. We will cover how to conduct single sample *t*-tests, independent samples *t*-tests, and matched/paired-samples *t*-tests. Fortunately, conducting all three utilizes the same function, `t.test()`.

3.5.1 Single Sample *t*-Test

Single sample *t*-tests are used to compare a sample mean value to some comparison value. For example, we might compare the average score on a test for a class of students to say 70%, and determine if the class, on average, scored significantly differently than 70%.

To demonstrate a single sample *t*-test, suppose we are interested in knowing if the average per game field goal percent across the players is significantly different from 50%. A field goal percent of 50% means a player makes as many shots as he misses, so if in the 2022-2023 season NBA players had, on average, a per game field goal percentage significantly greater than 50%, then they made more shots than they missed.

To test this, we will use the `t.test()` function to specify the variable in the sample, and we have to specify the population mean, `mu=`, referring to the comparison value.

```
t.test(bball$FGPerc, mu=.50)
```

```
    One Sample t-test

data:  bball$FGPerc
t = -7.5639, df = 469, p-value = 2.082e-13
alternative hypothesis: true mean is not equal to 0.5
95 percent confidence interval:
 0.4631390 0.4783418
sample estimates:
mean of x
0.4707404
```

Here, we see that the average per game field goal percentage in the '22-'23 season was about 47% (`mean of x = .471`). We also see that the this percentage is significantly less than 50%, t (469) = -7.56, $p < .001$. We can use the *lsr* (Navarro, 2015) package to compute the Cohen's D using the `cohensD()` function. Like the single sample *t*-test, we simply need to specify the variable in our data frame, and the population mean.

```
library(lsr)
```

```
cohensD(bball$FGPerc, mu=.5)
```

```
[1] 0.3488955
```

As such, we see that the average per game field goal percentage, $\overline{x} = .47$, is significantly less than 50%, $t(469) = -7.56$, $p < .001$, Cohen's $D = .35$.

3.5.2 Independent Samples *t*-Test

An independent samples *t*-Test is used to compare the means on a variable between two different groups. To demonstrate this, we can look to see if the average number of personal fouls committed per game, `PF`, differs significantly between front court players and back court players. This is the same research question we asked earlier when including categorical predictors in regression; we will just use a *t*-Test this time. We can do this in two ways: specifying columns or formula-based. For consistency with `lm()`, I will demonstrate the formula based approach.

Like the `lm()` function, we need to specify the outcome variable to the left of the `~` and the variable associated with the different groups to the right. We also need to specify the data frame, whether or not the test is a `paired= FALSE`, and whether or not the variances are equal, `var.equal =`.

To determine if the variances are equal, we can use Levene's test for homogeneity of variances from the *car* package (Fox & Weisberg, 2019), `leveneTest()`. Fortunately, the specification for this function is the same as `lm()`: outcome on the left of the `~`, grouping variable on the right, and a data frame.

```
library(car)
```

```
leveneTest(PF ~ FcBc, data=bball)
```

```
Warning in leveneTest.default(y = y, group = group, ...): group coerced to
factor.
```

```
Levene's Test for Homogeneity of Variance (center = median)
       Df F value Pr(>F)
group   1  1.5109 0.2196
      468
```

Based on the Levene's test, we see that the variances are not significantly different from each other, $F\ (1, 468) = 1.51$, $p = .220$. As such, we can specify `var.equal = TRUE`. We can also use the `cohensD()` function, with the same formula specification, to compute the effect size of the mean difference.

```
t.test(PF ~ FcBc, data=bball, var.equal=TRUE)
```

```
    Two Sample t-test

data:  PF by FcBc
t = -4.2583, df = 468, p-value = 2.491e-05
alternative hypothesis: true difference in means between group BC and group FC is not equal t
95 percent confidence interval:
 -0.4115799 -0.1516620
sample estimates:
mean in group BC mean in group FC
        1.614062         1.895683
```

```
cohensD(PF ~ FcBc, data=bball)
```

```
[1] 0.3995833
```

As we can see, there is a significant difference between front court and back court player on their average personal fouls per game, t (468) = -4.26, $p < .001$, Cohen's $D = .40$. We also see that the average number of fouls per game is greater among front court players, $\overline{x}_{FC} = 1.90$, than back court players, $\overline{x}_{BC} = 1.61$.

3.5.3 Paired Samples *t*-Test

Finally, a paired sample t-test is used to see if there is a significant difference between two variables within people. To demonstrate this, we might assess if the average percentage of three-point field goals made is significantly different (likely lower given they are more difficult to make) than the average percentage of free throws made. To run this analysis, we need to specify the two variables, not in formula notation, and specify `paired = TRUE`. If we want to specify the an `alternative=` we can do that as well. We can also specify the `cohensD()` function using the same general approach. One difference, though, is that we have to specify the `method="paired"`.

```
t.test(bball$X3Perc, bball$FTPerc, paired=TRUE, alternative = "two.sided")
```

```
    Paired t-test

data:  bball$X3Perc and bball$FTPerc
t = -66.597, df = 458, p-value < 2.2e-16
```

```
alternative hypothesis: true mean difference is not equal to 0
95 percent confidence interval:
 -0.4445771 -0.4190917
sample estimates:
mean difference
     -0.4318344
```

```
cohensD(bball$X3Perc, bball$FTPerc, method="paired")
```

```
[1] 3.108469
```

Here we can see that there is a significant difference between the average per game percentage of three point shots made compared to the average per game percentage of free throw shots made, $t(458) = -66.60$, $p < .001$, Cohen's $D = 3.11$.

3.6 Analysis of Variance (ANOVA)

When comparing means across three or more groups, Analysis of Variance (ANOVA) is a viable analysis strategy.[16] ANOVA compares the ratio of variance of an outcome within groups to the variance of the outcome across groups–if the latter is larger than the former, the groups differ more from each other than group members differ from one another.

3.6.1 One-Way ANOVA

When your study has only one factor on which three or more groups differ, a one-way ANOVA is appropriate. For instance, suppose we think that the average percentage of two point shots made per game differs among the basketball positions (e.g., center, point guard, etc.). In this case, we have five groups: point guard, shooting guard, small forward, power forward, and center.

Numerous packages allow you to conduct ANOVAs. For simplicity, I will show you have to use to the `ezANOVA()` function from the *ez* package (Lawrence, 2016). To be sure, there are a lot of options to specify within the `ezANOVA()` function; although we will cover a number of them, we cannot cover all of them.

For an one-way ANOVA, we will need to specify five things: the `data=`, the `dv=`, the ID variable distinguishing different cases, `wid=`, the `between=` factor, and the `type=` of sums of squares.[17]

[16] I note that ANOVA can be used to compare the means between two groups, but this is equivalent to a *t*-test, so most people will use a *t*-test. As described later, if one is comparing the means across more than one factor, a factorial ANOVA, even if there are only two groups on each factor, is necessary.

[17] A detailed discussion of the different types of sums of squares is beyond the scope of this volume. Interested readers are directed to Field et al. (2012) page 475-476.

One additional specification I will add is `detailed=TRUE` so that the function prints the full ANOVA table.

Lastly, I am setting `return_aov=FALSE`. I am setting this to false because of a very specific bug in the *ez* package that the developer has decided not to fix.[18] The bigger issue, though, is if `return_aov=TRUE`, there is a possibility that results will not match across functions in *R* when cell sizes are unequal. Since the `return_aov=` aspect of the code is not necessary, we can circumvent any issues by setting it to `FALSE`.

```
library(ez)
```

```
ezANOVA(data=bball, wid=Rk, dv=X2Perc, between=Pos, type=3, detailed=TRUE, return_aov = FALSE
```

```
Warning: Converting "Rk" to factor for ANOVA.
```

```
Warning: Converting "Pos" to factor for ANOVA.
```

```
Warning: Data is unbalanced (unequal N per group). Make sure you specified a
well-considered value for the type argument to ezANOVA().
```

```
Coefficient covariances computed by hccm()
```

```
$ANOVA
       Effect DFn DFd         SSn      SSd           F            p p<.05
1 (Intercept)   1 465 134.5030986 2.556134 24468.17412 0.000000e+00     *
2         Pos   4 465   0.8058795 2.556134    36.65046 1.209594e-26     *
        ges
1 0.9813501
2 0.2397014

$`Levene's Test for Homogeneity of Variance`
  DFn DFd         SSn      SSd         F         p p<.05
1   4 465 0.003435191 1.282825 0.3112981 0.8704353
```

Naturally, the independent variables need to be factors–we can either do that ourselves by specifying the variables `as.factor()`, or allow `ezANOVA()` to do it automatically. The other thing you will notice is that there is a warning about the unequal sample sizes, and that you should make sure your `type=` specification is "well-considered." Briefly, ANOVAs with unequal sample sizes in the conditions can be problematic, and `ezANOVA()` is just reminding you of that.

[18] This issue is described in detail here: https://github.com/jasp-stats/jasp-issues/issues/609

As the output shows, we see that, based on Levene's test, the variances on average percent of two point shots made is not significantly different among the groups, $F(4, 465) = .31$, $p > .05$. We further see that there is a significant mean difference among the groups in terms of the average percentage of two point shots made in a game, $F\ (4, 465) = 36.65$, $p < .001$, and that the effect size for this (i.e., generalized eta-squared, `ges`) is about $\eta^2 = .24$.

A follow-up question to ask is which groups are significantly different from each other? We can determine this by using the `PostHocTest()` function from the *DescTool* package (Signorell, 2024). To use this function, we need to specify an `aov` model and the `method=`. To obtain an `aov` model, we can actually rerun the `ezANOVA()` requesting `return_aov=TRUE`. To demonstrate, I'll rerun the ANOVA I just ran, but with three adjustments: (1) I will save the output of the ANOVA as an object, `Model2Pointers<-`, and (2) I will set `return_aov=TRUE` to ensure the output has an *aov* object for `PostHocTest()`. The third change I am going to make is to ensure that `Pos` is a factors using the `as.factor()` function. I am making this third change because the `PostHocTest()` function needs the between subject factors are specified as factors, and does not do so automatically like `ezANOVA()`.

After making these changes, I will specify `PostHocTest()` to run `method=bonferroni` corrected pairwise comparisons across the different conditions of the study. You will see that specifying the *aov* model in the `PostHocTest()` function requires referencing, using the `$`, the `aov` part of the `ezANOVA()` output. This is because the output from `ezANOVA()` is stored as a list in *R*; one of those list entries is the *aov* output, and this is the part we need in the `PostHocTest()` function.

```
library(DescTools)
```

```
Model2Pointers <- ezANOVA(data=bball, wid=Rk, dv=X2Perc, between=Pos, type=3, detailed=TRUE,
```

```
Warning: Converting "Rk" to factor for ANOVA.
```

```
Warning: Converting "Pos" to factor for ANOVA.
```

```
Warning: Data is unbalanced (unequal N per group). Make sure you specified a
well-considered value for the type argument to ezANOVA().
```

```
Coefficient covariances computed by hccm()
```

```
PostHocTest(Model2Pointers$aov, method="bonferroni")
```

```
  Posthoc multiple comparisons of means : Bonferroni
    95% family-wise confidence level
```

```
$Pos
             diff       lwr.ci       upr.ci    pval
PF-C  -0.04570202 -0.076158701 -0.015245339 0.00028 ***
PG-C  -0.11188630 -0.143111506 -0.080661098 < 2e-16 ***
SF-C  -0.08433560 -0.114881779 -0.053789427 4.5e-13 ***
SG-C  -0.10125859 -0.130228668 -0.072288504 < 2e-16 ***
PG-PF -0.06618428 -0.098108963 -0.034259601 9.4e-08 ***
SF-PF -0.03863358 -0.069894433 -0.007372733 0.00537 **
SG-PF -0.05555657 -0.085279243 -0.025833888 2.1e-06 ***
SF-PG  0.02755070 -0.004459373  0.059560771 0.15579
SG-PG  0.01062772 -0.019881979  0.041137412 1.00000
SG-SF -0.01692298 -0.046737358  0.012891393 1.00000

---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Here, we see that centers (C) have a higher two point shot shooting percentage than any of the other positions. Likewise, power forwards (PF) have a significantly higher two point shot shooting percentage than small forwards (SF), shooting guards (SG), and point guards (PG). The latter three groups do not differ significantly from each other.

Clearly, a missing piece of information are the actual average percentage of two point shots made. To ascertain this information, we can use the `describeBy()` function from *psych* to look at the mean two-point shot percentage across the five groups.

```
describeBy(x=bball$X2Perc, group=bball$Pos, fast=TRUE)
```

```
 Descriptive statistics by group
group: C
   vars  n mean   sd median  min  max range  skew kurtosis   se
X1    1 99 0.61 0.08   0.61 0.25 0.82  0.57 -0.91     3.93 0.01
------------------------------------------------------------
group: PF
   vars  n mean   sd median  min  max range skew kurtosis   se
X1    1 90 0.56 0.06   0.56 0.42 0.74  0.32 0.19     -0.1 0.01
------------------------------------------------------------
group: PG
   vars  n mean   sd median  min  max range  skew kurtosis   se
X1    1 82 0.49 0.08   0.49 0.07 0.66  0.59 -2.04    10.51 0.01
------------------------------------------------------------
group: SF
   vars  n mean   sd median  min  max range  skew kurtosis   se
```

```
X1    1 89 0.52 0.08   0.53 0.19 0.68  0.49 -1.25     3.93 0.01
------------------------------------------------------------
group: SG
   vars   n mean   sd median  min  max range skew kurtosis   se
X1    1 110 0.51 0.08   0.51 0.25 0.75   0.5 -0.3     1.76 0.01
```

For basketball fans, this information is unlikely to be surprising given than these two positions tend to, though not always, take most of their shots from very close to the basket.

3.6.2 Factorial ANOVA

When comparing means among groups that differ on two or more factors, you can use factorial ANOVA. Factorial ANOVA separates the variance of an outcome variable across combinations of levels of two or more independent variables. Fortunately, the `ezANOVA()` function can easily accommodate two or more independent variables.

To demonstrate this, suppose we are interested in knowing if the average percentage of three point shots made in a game differs between front court and back court players (`FcBc`) and between eastern conference and western conference teams (`Conf`). The only differences in the `ezANOVA()` function between one-way ANOVA and factorial ANOVA is that the `between=` aspect of the code will now have to include both factors. We are also changing the outcome variable to `X3Perc`.

One additional step we need to take, though, is how to handle the missing data. In particular, there are eight players who did not attempt a three point shot all season; as such, the data for these individuals is missing. On the one hand, we could replace the missing data with a value, say 0, to reflect that 0% of their three point shots were made. However, this would be inaccurate insofar as they did not attempt a shot. On the other hand, `ezANOVA()` does not handle missing data well. To deal with this issue, we can create a different data frame with just the variables of interest, then use the `na.omit()` function to remove (i.e, omit) anyone with missing data on any of the variables in the new data frame.

```
Threes<- bball[,c(1,5,7,14)]
Threes <- na.omit(Threes)
```

You can see that there is a new data frame in the *R* environment named `Threes` that has 462 rows–470 original data points minus the eight players with missing data. Now we can assess if the average percentage of three point shots made varies between front court and back court players, by conference, and by the combination using the `ezANOVA()` function. You will notice that now the `between=` aspect has two variables, collected within parentheses with a `.` before it–the period is just an *ez* package specific rule.

```
ezANOVA(data=Threes, wid=Rk, dv=X3Perc, between=.(FcBc, Conf), type=3, detailed=TRUE, return
```

```
Warning: Converting "Rk" to factor for ANOVA.

Warning: Converting "FcBc" to factor for ANOVA.

Warning: "Conf" will be treated as numeric.

Warning: Data is unbalanced (unequal N per group). Make sure you specified a
well-considered value for the type argument to ezANOVA().

Coefficient covariances computed by hccm()

Warning: At least one numeric between-Ss variable detected, therefore no
assumption test will be returned.

$ANOVA
       Effect DFn DFd          SSn      SSd            F            p p<.05
1 (Intercept)   1 458 5.8344607633 5.804464 460.36690132 3.371099e-71     *
2        FcBc   1 458 0.0220025431 5.804464   1.73610604 1.882917e-01
3        Conf   1 458 0.0625542108 5.804464   4.93582687 2.679351e-02     *
4   FcBc:Conf   1 458 0.0007654038 5.804464   0.06039403 8.059843e-01
           ges
1 0.5012886469
2 0.0037763100
3 0.0106620110
4 0.0001318473
```

The first thing you will notice is that we have violated Levene's test for homogeneity of variances, $F(5, 456) = 4.50$, $p < .001$. You can address this violation in multiple ways (e.g., transformations, Welch's F, robust standard errors), and I encourage you to evaluate the pros and cons of each. For our purposes here, we will continue to evaluate this ANOVA, without correcting for the violated assumption, as we are simply interested in demonstrating the technique.

Based on the output, we see again the there is a significant difference in the percent of three point shots made per game between front court and back court players, F (1, 456) = 10.19, p < .001, $\eta^2 = .02$. We also see, though, that there is not difference between the conferences nor an interaction.

If the interaction were significant, however, we could follow up the ANOVA with post-hoc tests. To do this, you can use the `PostHocTest()` from the *DescTools* package. Like earlier, we are going to specify the `ezANOVA()` output as an object, `Model3Pointers <-`. We will also

specify `return_aov=TRUE`, and set `FcBc` and `Conf` as factors. After making these changes, I will specify `PostHocTest()` to run `method=bonferroni` corrected pairwise comparisons across the different conditions of the study.

```
Threes$FcBc <- as.factor(Threes$FcBc)
Threes$Conf <- as.factor(Threes$Conf)

Model3Pointers <- ezANOVA(data=Threes, wid=Rk, dv=X3Perc, between=.(FcBc, Conf), type=3, deta
```

```
Warning: Converting "Rk" to factor for ANOVA.
```

```
Warning: Data is unbalanced (unequal N per group). Make sure you specified a
well-considered value for the type argument to ezANOVA().
```

```
Coefficient covariances computed by hccm()
```

```
PostHocTest(Model3Pointers$aov, method="bonferroni")
```

```
  Posthoc multiple comparisons of means : Bonferroni
    95% family-wise confidence level

$FcBc
             diff     lwr.ci      upr.ci   pval
FC-BC -0.03117917 -0.0521012 -0.01025713 0.0036 **

$Conf
          diff       lwr.ci     upr.ci   pval
2-1 0.01664939 -0.010585582 0.04388436 0.4276
3-1 0.03436617 -0.004173835 0.07290618 0.0980 .
3-2 0.01771678 -0.020774593 0.05620815 0.8080

$`FcBc:Conf`
                  diff       lwr.ci     upr.ci   pval
FC:1-BC:1 -0.036960515 -0.085144965 0.01122393 0.3612
BC:2-BC:1  0.008049383 -0.044240514 0.06033928 1.0000
FC:2-BC:1 -0.014367441 -0.062383700 0.03364882 1.0000
BC:3-BC:1  0.033797531 -0.037324418 0.10491948 1.0000
FC:3-BC:1 -0.002580247 -0.069237047 0.06407655 1.0000
BC:2-FC:1  0.045009898 -0.003174552 0.09319435 0.0912 .
FC:2-FC:1  0.022593074 -0.020916339 0.06610249 1.0000
BC:3-FC:1  0.070758046  0.002597651 0.13891844 0.0348 *
```

```
FC:3-FC:1  0.034380268 -0.029107030 0.09786757 1.0000
FC:2-BC:2 -0.022416824 -0.070433082 0.02559944 1.0000
BC:3-BC:2  0.025748148 -0.045373800 0.09687010 1.0000
FC:3-BC:2 -0.010629630 -0.077286430 0.05602717 1.0000
BC:3-FC:2  0.048164972 -0.019876629 0.11620657 0.5592
FC:3-FC:2  0.011787194 -0.051572548 0.07514694 1.0000
FC:3-BC:3 -0.036377778 -0.118640937 0.04588538 1.0000

---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

From these results, we see that the average percentage of three point shots made by back court players is significant higher, by about .03, than front court players. We further see, though, that the three conditions for conferences do not differ, nor are there any meaningful differences among the crossing of the two factors.[19]

3.6.3 Repeated Measures ANOVA

Repeated measures ANOVA is a technique used to assess if the means across three or more measurements differ significantly from each other wherein each measurement is collected within person. This could be, for example, measurements of the same construct at three different times (e.g., pre test, post test, follow-up test), or three different measurements of constructs (importantly, using the same scaling/response scale). For instance, we can assess if the average percentage of three point shots, two point shots, and free throw shots made a game differ significantly from each other across players in the NBA in the 2022-2023 season (we can compare these three variables because all three are percentages).

One data pre-processing step in which we need to engage is to change the format of the data from wide format to long format. Whereas wide format data has a row of a data frame for each participant, long format has a row for each participant's score on a variable.

To reshape the data, we will make use of the *reshape2* package's (Wickham, 2007) `melt()` function. To utilize this function, we will specify the name of a data frame, the `id=` that tells `melt()` what are between subject factors, and then a `measured=` which are the within subjects factors. We will specify this as a new data frame.

To make things a little easier, though, we can start by creating a smaller data frame of only the variables we need for the current analysis. We need to specify: `Rk`, `X3Perc`, `X2Perc`, and `FTPerc`. In addition, we will specify `FcBc` in the new data frame because in the demonstration of mixed-effects ANOVA below we will make sure of the same general research question, but

[19] You may notice that `BC: 3-FC:1` comparison is significantly different from each other. We are unlikely to interpret this, though, as the overall interaction term was not significant, and this is likely not a replicable result.

add a between subjects factor. Finally, we will use the the `na.omit()` function to remove missing data from the data frame before reshaping it because `ezANOVA()` does not hanlde missing data.[20]

```
library(reshape2)

Within <- bball[,c(1,5,14,16,18)]
Within <- na.omit(Within)
RMANOVA <- melt(data=Within, id=c("Rk","FcBc"), measured=c("X3Perc", "X2Perc", "FTPerc"))
```

After running these lines of code, you will see a new object in your *R* environment named `RMANOVA` with 1377 observations on four variables–this is because each respondent has about three rows of data each. You can see, for example, the first 21 rows of the new data frame (sorted on `Rk`) shows three rows for each `Rk`, 1-7, with the same value for `FcBc` for each set of three, but different values for `X3Perc`, `X2Perc`, and `FTPerc`.

```
RMANOVA[order(RMANOVA$Rk),][1:21,]
```

```
     Rk FcBc variable value
358   1   FC   X3Perc 0.269
817   1   FC   X2Perc 0.564
1276  1   FC   FTPerc 0.702
179   2   FC   X3Perc 0.000
638   2   FC   X2Perc 0.599
1097  2   FC   FTPerc 0.364
195   3   FC   X3Perc 0.083
654   3   FC   X2Perc 0.545
1113  3   FC   FTPerc 0.806
444   4   BC   X3Perc 0.355
903   4   BC   X2Perc 0.532
1362  4   BC   FTPerc 0.812
182   5   FC   X3Perc 0.353
641   5   FC   X2Perc 0.591
1100  5   FC   FTPerc 0.750
420   6   BC   X3Perc 0.384
879   6   BC   X2Perc 0.515
1338  6   BC   FTPerc 0.667
222   7   BC   X3Perc 0.399
681   7   BC   X2Perc 0.518
1140  7   BC   FTPerc 0.905
```

[20] In volume II of this series, I will show you how to use multilevel modeling to model repeated measures data like this in which case the missing data can be handled.

Now that we have the data in the format we need, we can use `ezANOVA` to run a repeated measures ANOVA. Like one-way and factorial ANOVA, we specify the `data=`, `dv=`, `detailed=`, and `wid=`. Rather than specifying a `between=.()`, though, we will specify the `within=.()`.

```
ezANOVA(data=RMANOVA, dv=.(value), wid=.(Rk), within=.(variable), type=3, detailed=TRUE, retu
```

```
Warning: Converting "Rk" to factor for ANOVA.
```

```
$ANOVA
       Effect DFn DFd       SSn      SSd         F p p<.05       ges
1 (Intercept)   1 458 405.85505 5.285220 35170.077 0     * 0.9640819
2    variable   2 916  42.80902 9.835438  1993.458 0     * 0.7389825

$`Mauchly's Test for Sphericity`
    Effect         W          p p<.05
2 variable 0.9888574 0.07727548

$`Sphericity Corrections`
    Effect       GGe p[GG] p[GG]<.05       HFe p[HF] p[HF]<.05
2 variable 0.9889802     0         * 0.9932456     0         *
```

From this output, we can see Mauchley's test of sphericity is not violated, $w = .99$, $p = .08$. We also see that there is a significant difference among the percentage of types of shots made, $F\ (2, 916) = 1993.46$, $p < .001$, $\eta^2 = .74$.

To follow-up the significant ANOVA, we can use the `pairwise.t.test()` function to assess which measurements are significantly different from each other. To use this function, we will need to specify the outcome variable `x=`, the factor `g=`, how we want to adjust the p-values for family-wise type I error[21], `p.adjust.method=`, and finally we want `paired=TRUE` to account for the within-person nature of the data.

```
pairwise.t.test(x=RMANOVA$value, g=RMANOVA$variable, p.adjust.method="bonferroni", paired=TRU
```

```
    Pairwise comparisons using paired t tests

data:  RMANOVA$value and RMANOVA$variable
```

[21] A full discussion of family-wise type I error is beyond the scope of this volume, and you are encouraged to consult your statistics notes/textbook for a discussion. In addition, there are numerous corrections other than the Bonferroni correction I prefer, as well as lengthy discussions on the appropriate post-hoc analyses to follow a significant ANOVA that I encourage you to read.

```
       X3Perc X2Perc
X2Perc <2e-16 -
FTPerc <2e-16 <2e-16

P value adjustment method: bonferroni
```

The output of this function provides a matrix of p-values associated with each pairwise comparison, and not surprisingly, shows that each of the measurements are significantly different from each other. As such, we can use the `describeBy()` function to look at the means of each variable.

```
describeBy(RMANOVA$value, group=RMANOVA$variable, fast=TRUE)
```

```
 Descriptive statistics by group
group: X3Perc
   vars   n mean   sd median min max range  skew kurtosis   se
X1    1 459 0.33 0.11   0.35   0   1     1 -0.35     6.98 0.01
------------------------------------------------------------
group: X2Perc
   vars   n mean   sd median  min  max range skew kurtosis se
X1    1 459 0.54 0.08   0.54 0.25 0.78  0.53 -0.1     0.78  0
------------------------------------------------------------
group: FTPerc
   vars   n mean   sd median min max range  skew kurtosis   se
X1    1 459 0.76 0.12   0.77   0   1     1 -1.13     4.11 0.01
```

We see that players' average percentage of free throws made in a game was significantly great than, percentage of two point shots made, which was significantly greater than percentage of three point shots made.

3.6.4 Mixed-Model ANOVA

In some instances, you might be interested in understanding how variables differ both within a person, like a repeated-measures design, and across people, like a between-subjects design. In these instances, you can use Mixed-Models ANOVA. For example, earlier we learned that the average percentage of three-point, two-point, and free-throw shots made per game differ significantly. Suppose we were also interested in whether or not these percentages differ along with whether or not a player is a front court or back court player.

We can assess this relation by including both a `within=.()` and `between=.()` factors in the `ezANOVA()` function, assuming that our data are in long format. Since we did that earlier, we can move to modeling the mixed-model ANOVA.

```
ezANOVA(data=RMANOVA, wid=.(Rk), dv=.(value), within=.c(variable), between=.c(FcBc), type=3,

Warning: Converting "Rk" to factor for ANOVA.

Warning: Converting "FcBc" to factor for ANOVA.

Warning: Data is unbalanced (unequal N per group). Make sure you specified a
well-considered value for the type argument to ezANOVA().

$ANOVA
         Effect DFn DFd          SSn      SSd             F            p p<.05
1   (Intercept)   1 457 3.948967e+02 5.280390 3.417698e+04 0.000000e+00     *
2          FcBc   1 457 4.830477e-03 5.280390 4.180616e-01 5.182305e-01
3      variable   2 914 4.188970e+01 9.106124 2.102277e+03 0.000000e+00     *
4 FcBc:variable   2 914 7.293141e-01 9.106124 3.660136e+01 5.113439e-16     *
           ges
1 0.9648494883
2 0.0003356516
3 0.7443588921
4 0.0482483686

$`Mauchly's Test for Sphericity`
         Effect         W         p p<.05
3      variable 0.9988409 0.7676496
4 FcBc:variable 0.9988409 0.7676496

$`Sphericity Corrections`
         Effect       GGe        p[GG] p[GG]<.05      HFe        p[HF]
3      variable 0.9988423 0.000000e+00         * 1.003225 0.000000e+00
4 FcBc:variable 0.9988423 5.300793e-16         * 1.003225 5.113439e-16
  p[HF]<.05
3         *
4         *
```

Here we see that Mauchly's test of sphericity is again not violated for either the main effect, $w = 1.00$, p .77, nor the interaction, $w = 1.00$, $p = .77$. Interestingly, we see a main effect of type of shot, F (2, 914) = 2102.28, $p < .001$, $\eta^2 = .74$, but not for player position, F (1, 457) = .42, $p = .52$, $\eta^2 < .001$. There was, however, a significant interaction between type of shot and player positions, F (2, 914) = 36.60, $p < .001$, $\eta^2 = .05$.

To follow up on this significant interaction, we would want to look at the simple effects. We can do this in multiple ways including looking at the within-person effects within levels of

the between subjects effects (i.e., repeated measures ANOVA for front court players only and for back court players only). Alternatively, you can investigate if each of the repeated measurements (i.e., `X3Perc`, `X2Perc`, and `FTPerc`) differs significantly between the levels of the between factors variables–This would be one-way ANOVAs on each of the three repeated measures. This latter approach is a straightfoward application of one-way ANOVA as discussed above, so we won't redo all of that here.

To utilize the former approach, though, we will need to subset the data using the `subset()` function, specifying the original data frame and how we will subset it–namely into `FC` and `BC` players. That is, we will make a data frame, labeled `FC` and `BC`, of the rows of data wherein the variable `FcBc == FC` or `BC`, respectively. Note the use of `==` to tell *R* to to check to see if the entries equal a condition–a single `=` will not work because of *R* language rules.

```
FC <- subset(RMANOVA, FcBc=="FC")
BC <- subset(RMANOVA, FcBc=="BC")
```

You will now see two new data frames in your *R* environment corresponding to data only for front court players, `FC`, and data corresponding to only back court players , `BC`. With these new data frames we can run the repeated measures ANOVA we ran above across all the players, but for the two groups separately.

```
ezANOVA(data=FC, dv=value, within=.(variable), wid=Rk, type=3, return_aov = FALSE, detailed=
```

```
Warning: Converting "Rk" to factor for ANOVA.
```

```
$ANOVA
       Effect DFn DFd       SSn      SSd         F             p p<.05
1 (Intercept)   1 267 235.59151 3.218955 19541.416 1.002339e-251     *
2    variable   2 534  24.50922 6.156567  1062.924 6.557812e-187     *
        ges
1 0.9617274
2 0.7233114

$`Mauchly's Test for Sphericity`
    Effect        W          p p<.05
2 variable 0.974352 0.03156547     *

$`Sphericity Corrections`
    Effect       GGe         p[GG] p[GG]<.05       HFe         p[HF] p[HF]<.05
2 variable 0.9749934 2.559047e-182         * 0.9820721 1.283466e-183         *
```

```
ezANOVA(data=BC, dv=value, within=.(variable), wid=Rk, type=3, return_aov = FALSE, detailed=
```

```
Warning: Converting "Rk" to factor for ANOVA.
```

```
$ANOVA
       Effect DFn DFd       SSn      SSd         F             p p<.05
1 (Intercept)   1 190 170.26836 2.061435 15693.430 1.433710e-184     *
2    variable   2 380  19.02912 2.949557  1225.788 1.877218e-166     *
        ges
1 0.9714114
2 0.7915570

$`Mauchly's Test for Sphericity`
    Effect         W            p p<.05
2 variable 0.8783126 4.729764e-06     *

$`Sphericity Corrections`
    Effect      GGe         p[GG] p[GG]<.05       HFe         p[HF] p[HF]<.05
2 variable 0.891514 9.676247e-149         * 0.8993831 5.021867e-150         *
```

As you can see, both repeated measures ANOVAs show that the percentage of each shot made differ significantly from each other for both groups. To further investigate this, you can use the `pairwise.t.test()` function as we did above and consider using the one-way ANOVA approach. If you do, you will see that for all three types of shots, the groups differ significantly from each other. More specifically, whereas the back court players have a higher percentage of three point and free throw shots made per game, the front court players have a higher percentage of two point shots made.

3.7 Analysis of Covariance

Analysis of Covariance (ANCOVA) is similar to ANOVA in that we are interested in partitioning the variance in an outcome variable across levels of one or more factors. Where ANCOVA differs is the inclusion of one or more continuous covariates. For instance, we might be interested in knowing if the average minutes played per game differs by position played. However, we might also think that the number of minutes played is likely impacted by the number of games started. In this case, we might want to partition the variance in average minutes played across the different positions on the court, but control for the number of games started; ANCOVA allows us to do this.

One issue, though, is that the `ezANOVA()` implementation of ANCOVA is not fully validated yet (as of 2/9/2024). As such, we will make use of the `Anova()` function from the *car* package (Fox & Weisberg, 2019), and the `aov()` function from the basic install of *R*. Specifically, we will wrap the the `Anova()` function around the `aov()` function, and specify `type=3`, for the sums of squares. Within the `aov()` function, we will specify the formula relating the outcome, `MP`, to the two predictors, `~ Pos + GS`, along with `data=bball`, and `na.action=na.exclude` to handle the missing data.

```
Anova(aov(MP ~ Pos + GS, data=bball, na.action=na.exclude), type=3)
```

```
Anova Table (Type III tests)

Response: MP
             Sum Sq  Df  F value    Pr(>F)
(Intercept) 10844.6   1  545.095 < 2.2e-16 ***
Pos           833.1   4   10.468 3.981e-08 ***
GS          27691.6   1 1391.893 < 2.2e-16 ***
Residuals    9231.2 464
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Here we see that both the predictor, `Pos`, and the covariate, `GS`, significantly predict the average number of minutes played in a game during the 2022-2023 season. Using the `cor.test()` function, we can assess the relation between average number of games started and minutes played.

```
cor.test(bball$MP, bball$GS, type="pearson")
```

```
    Pearson's product-moment correlation

data:  bball$MP and bball$GS
t = 36.084, df = 468, p-value < 2.2e-16
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 0.8317462 0.8798647
sample estimates:
      cor
0.8576719
```

Not surprising, starting more games on average means playing more minutes per game.

To conduct post hoc tests on `Pos` we have to compare the mean minutes played, adjusted for games started. As such, functions like `PostHocTest()`, from the *desc* package, or `pairwise.t.test()` will not work. Instead, we will need to use the `emmeans_test()` function from the the *rstatix* package (Kassambara, 2023). Fortunately, utilizing this function is straightforward; we simply need to specify `data=`, the formula linking the outcome to the factors, input `covariate=`, specify a `p.adjust.method=`, and request `detailed=TRUE` to ensure we get confidence intervals for the comparisons.

```
library(rstatix)
```

```
emmeans_test(data=bball, formula=MP ~ Pos, covariate=GS, p.adjust.method = "bonferroni", deta
```

```
# A tibble: 10 x 14
   term    .y.   group1 group2 null.value estimate    se    df conf.low conf.high
 * <chr>   <chr> <chr>  <chr>       <dbl>    <dbl> <dbl> <dbl>    <dbl>     <dbl>
 1 GS*Pos MP    C      PF              0  -2.62   0.650   464   -3.90    -1.34
 2 GS*Pos MP    C      PG              0  -4.01   0.666   464   -5.32    -2.70
 3 GS*Pos MP    C      SF              0  -2.87   0.652   464   -4.15    -1.59
 4 GS*Pos MP    C      SG              0  -2.90   0.618   464   -4.12    -1.69
 5 GS*Pos MP    PF     PG              0  -1.39   0.681   464   -2.73    -0.0485
 6 GS*Pos MP    PF     SF              0  -0.250  0.667   464   -1.56     1.06
 7 GS*Pos MP    PF     SG              0  -0.284  0.635   464   -1.53     0.963
 8 GS*Pos MP    PG     SF              0   1.14   0.683   464   -0.205    2.48
 9 GS*Pos MP    PG     SG              0   1.10   0.652   464   -0.179    2.38
10 GS*Pos MP    SF     SG              0  -0.0335 0.637   464   -1.29     1.22
# i 4 more variables: statistic <dbl>, p <dbl>, p.adj <dbl>, p.adj.signif <chr>
```

From this output we see that, controlling for games started, players who play the center position tend to play significantly fewer average minutes a game compared to all other positions. However, the remaining four positions do not differ from each other meaningfully.

3.8 Chi-Square χ^2

The final analysis we will cover in volume I is χ^2 analysis. χ^2 is used to assess if categorical variables differ significantly from each other. For instance, suppose we were interested in knowing if the different conferences had different numbers of players at each position. We could use a χ^2 test to assess if the observed number of players in each position across the two conferences is what would be expected, where the expected amount would be the same number across conferences.

To run this analysis, we will use the `CrossTable()` function from the *gmodels* package (Warnes, Bolker, Lumley, & Johnson, 2022). To utilize this function, we can specify the two categorical variables. In addition, we will request a few things: `expected=TRUE` will display the expected values within each cell, `chisq=TRUE` will print the χ^2 statistic, and `format="SPSS"` which will display the results in percentage format rather than proportion format. In addition to these specification, you can also request Fisher's Exact Test, `fisher=TRUE`, and/or McNemar's test, `mcnemar=TRUE`.

```
library(gmodels)
```

```
CrossTable(bball$Pos, bball$Conf, expected = TRUE, chisq = TRUE, format = "SPSS")
```

```
   Cell Contents
|-------------------------|
|                   Count |
|         Expected Values |
| Chi-square contribution |
|             Row Percent |
|          Column Percent |
|           Total Percent |
|-------------------------|

Total Observations in Table:  470
```

	bball$Conf			
bball$Pos	1	2	3	Row Total
C	44	40	15	99
	42.338	42.338	14.323	
	0.065	0.129	0.032	
	44.444%	40.404%	15.152%	21.064%
	21.891%	19.900%	22.059%	
	9.362%	8.511%	3.191%	
PF	36	45	9	90
	38.489	38.489	13.021	
	0.161	1.101	1.242	
	40.000%	50.000%	10.000%	19.149%
	17.910%	22.388%	13.235%	
	7.660%	9.574%	1.915%	
PG	37	32	13	82
	35.068	35.068	11.864	
	0.106	0.268	0.109	
	45.122%	39.024%	15.854%	17.447%
	18.408%	15.920%	19.118%	
	7.872%	6.809%	2.766%	
SF	40	35	14	89
	38.062	38.062	12.877	
	0.099	0.246	0.098	
	44.944%	39.326%	15.730%	18.936%
	19.900%	17.413%	20.588%	
	8.511%	7.447%	2.979%	

```
-------------|-----------|-----------|-----------|-----------|
          SG |        44 |        49 |        17 |       110 |
             |    47.043 |    47.043 |    15.915 |           |
             |     0.197 |     0.081 |     0.074 |           |
             |    40.000% |    44.545% |    15.455% |   23.404% |
             |    21.891% |    24.378% |    25.000% |           |
             |     9.362% |    10.426% |     3.617% |           |
-------------|-----------|-----------|-----------|-----------|
Column Total |       201 |       201 |        68 |       470 |
             |    42.766% |    42.766% |    14.468% |           |
-------------|-----------|-----------|-----------|-----------|


Statistics for All Table Factors


Pearson's Chi-squared test
------------------------------------------------------------
Chi^2 =  4.009374     d.f. =  8     p =  0.8562768



       Minimum expected frequency: 11.86383
```

Not surprisingly, we see that the observed number of players at each position across the three conferences (recall, that conference level "3" refers to players who played in both conferences during the season) does not differ from the expected rate, χ^2 (8) = 4.01, p = .856. This is because there is not necessarily a strategic reason to carry more or fewer types of players on one's team based on the conference one's team is in.

For demonstration purposes, let's look at if the counts of position, `Pos`, differs significantly across front court versus back court designation, `FcBc`. Naturally, this comparison will show significant differences because there are cells of this 5 x 2 matrix that have 0 entries (i.e., centers are not back court players). Running this analysis, though, shows you a significant result.

```
CrossTable(bball$Pos, bball$FcBc, expected = TRUE, chisq = TRUE, format = "SPSS")
```

```
   Cell Contents
|-------------------------|
|                   Count |
|         Expected Values |
```

```
| Chi-square contribution |
|             Row Percent |
|          Column Percent |
|           Total Percent |
|-------------------------|

Total Observations in Table:  470

             | bball$FcBc
   bball$Pos |       BC  |       FC  | Row Total |
-------------|-----------|-----------|-----------|
           C |        0  |       99  |       99  |
             |   40.443  |   58.557  |           |
             |   40.443  |   27.932  |           |
             |    0.000% |  100.000% |   21.064% |
             |    0.000% |   35.612% |           |
             |    0.000% |   21.064% |           |
-------------|-----------|-----------|-----------|
          PF |        0  |       90  |       90  |
             |   36.766  |   53.234  |           |
             |   36.766  |   25.392  |           |
             |    0.000% |  100.000% |   19.149% |
             |    0.000% |   32.374% |           |
             |    0.000% |   19.149% |           |
-------------|-----------|-----------|-----------|
          PG |       82  |        0  |       82  |
             |   33.498  |   48.502  |           |
             |   70.227  |   48.502  |           |
             |  100.000% |    0.000% |   17.447% |
             |   42.708% |    0.000% |           |
             |   17.447% |    0.000% |           |
-------------|-----------|-----------|-----------|
          SF |        0  |       89  |       89  |
             |   36.357  |   52.643  |           |
             |   36.357  |   25.110  |           |
             |    0.000% |  100.000% |   18.936% |
             |    0.000% |   32.014% |           |
             |    0.000% |   18.936% |           |
-------------|-----------|-----------|-----------|
          SG |      110  |        0  |      110  |
             |   44.936  |   65.064  |           |
             |   94.207  |   65.064  |           |
             |  100.000% |    0.000% |   23.404% |
```

```
             |   57.292% |    0.000% |           |
             |   23.404% |    0.000% |           |
-------------|-----------|-----------|-----------|
Column Total |       192 |       278 |       470 |
             |   40.851% |   59.149% |           |
-------------|-----------|-----------|-----------|


Statistics for All Table Factors


Pearson's Chi-squared test
------------------------------------------------------------
Chi^2 =  470     d.f. =  4     p =  2.059248e-100



       Minimum expected frequency: 33.49787
```

Here we see that the observed count of each position as either front court or back court is significantly different than what is expected, χ^2 (4) = 470.00, $p < .001$. For example, we would expect about 65 players who are shooting guards (SG) to be listed as front court players (FC), but the actual count is 0. Again, although this is not a very realistic example of using a χ^2 analysis – given that we created one of the variables, `FcBc`, based on the other variable, `Pos` – you can see how to use this appraoch when you are trying to assess if categorical variables are significantly related to each other. In Volume II of this series, we discuss how to predict categorical variables using binary and multinomial logistic regression.

4 Summary

The goal of Volume I in the *Using R* series is to introduce readers to what is *R,* and how to use *R* to perform some basic univariate statistical techniques. Before moving on to Volumes II or III, it is a good idea to have a good grasp of the topics covered in this volume. Indeed, utilizing *R* to conduct path analysis, model mediations or moderations, and/or run complex psychometric analyses will be significantly easier if you have a solid grasp of the information presented here.

Most importantly, *R* is a language and skill you are trying to learn. Like any skill, you will improve with dedicated practice. As such, utilize the accompanying data and recreate the output in this volume on your own. Utilize variables in the data set we didn't use to practice these techniques. It is not important that the results are significant; rather, the goal is to

practice producing output using these data so that you can produce the output when you are analyzing your own data.

Finally, if you have any issues or run into any errors while using this volume, please do not hesitate to reach out to me (ddalal@albany.edu). Given the dynamic nature of *R* and package development, it is possible that updates to packages or *R* itself will make some of this code not work as written. If this happens to you while using this volume, let me know and I can update things.

5 References

- Dalal, D. (2023, August 23). (Multi)Collinearity in Behavioral Sciences Research. *Oxford Research Encyclopedia of Business and Management.* Retrieved 1 Feb. 2024, from https://oxfordre.com/business/view/10.1093/acrefore/9780190224851.001.0001/acrefore-9780190224851-e-410.
- Field, A., Miles, J., & Field, Z. (2012). *Discovering Statistics Using R.* Washington, DC: SAGE.
- Fletcher, T. D., (2022). *QuantPsyc: Quantitative Psychology Tools.* R package version 1.6. https://CRAN.R-project.org/package=QuantPsyc.
- Fox, J., & Weisberg, S. (2019). *An R Companion to Applied Regression* (3rd. Edition). Thousand Oaks, CA: Sage.
- Harrell Jr, F. (2022). Hmisc: Harrell Miscellaneous. R package version 4.7-0, https://CRAN.R-project.org/package=Hmisc.
- Kassambara, A. (2023). *rstatix: Pipe-Friendly Framework for Basic Statistical Tests.* R package version 0.7.2, https://CRAN.R-project.org/package=rstatix.
- Lawrence, M. A. (2016). *ez: Easy Analysis and Visualization of Factorial Experiments.* R package version 4.4-0, https://CRAN.R-project.org/package=ez.
- Navarro, D. J. (2015) *Learning statistics with R: A tutorial for psychology students and other beginners.* (Version 0.6). University of New South Wales. Sydney, Australia
- R Core Team (2022). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna Austria. URL https://www.R-project.org/.
- Revelle, W. (2023). *psych: Procedures for Psychological, Psychometric, and Personality Research.* Evanston, IL. R package version 2.3.12, https://CRAN.R-project.org/package=psych.
- Signorell, A. (2024). _DescTools: Tools for Descriptive Statistics_. R package version 0.99.54, https://CRAN.R-project.org/package=DescTools.

- Tabachnick, B. G., & Fidell, L. S. (2019). *Using Multivariate Statistics* (7th Edition). New York: Pearson.
- Warnes, G. R., Bolker, B., Lumley, T., & Johnson, R. C. (2022). *gmodels: Various R Programming Tools for Model Fitting.* R package version 2.18.1.1, https://CRAN.R-project.org/package=gmodels.
- Wickham, H. (2007). Reshaping data with the reshape package. *Journal of statistical software*, *21*, 1-20.