Data-Driven Asset Management

Possible finally matches vision

Richard G. Lamb

Chapter 3
The "R" Software in Action

Second Edition

Contents

Chapter 3
The “R” Software in Action

To qualify as data-driven, an asset management organization must acquire a critical-mass of skills in the “R” software as its analytic core. The purpose of this chapter is to hand the skills of “R” off to its readers.

The chapter will begin by explaining how the hand-off of skills will be accomplished. It will next explain the process to install “R” on personal computers and then run a session.

The remainder of the chapter will work through a hands-on case in correlation analysis and linear regression. Furthermore, the readers can use the case, as a go-by, to work with the data of their own roles.

The data that was formed to demonstrate the methods of the chapter are available in the file Chap3AnxietyParCorr.csv. The file is available for download at https://analytics4strategy.com/ddassetmgt.

The R script which is explained throughout the chapter is also available from the same webpage as the file Chap3RInAction.R.txt. The extension “.txt” has been added to allow placement on the webpage as a notepad file. The extension must be removed from the file name to make it directly loadable into an R session as explained in section 3.2.

Additionally, the path element in the R script are the author’s. The reader must replace them with their own path. The cases are flagged as `<path>` in the code presented throughout the chapter and the book.

3.1. Approach to Reach Critical Mass

The approach to the chapter is to capitalize on the plentiful published literature that draw upon the ”R” software to explain and demonstrate every type of statistical analysis. Just as important, the same body of literature explains the “R” code to each topic of analysis.

Accordingly, the chapter will draw upon published examples of correlation, partial correlation and linear regression to hand off to its readers the critical-mass skills in the “R” software.

The explanation is not intended to be a full-depth explanation of the principles and practices of the demonstrated analyses. Instead, the demonstration is limited to working with “R” while explaining each introduced analytic to a critical mass of depth. The overarching objective of the chapter is to cause its readers to “run around in R.” Throughout the explanation, readers will be introduced to the most commonly and frequently occurring code in “R.”

The principles and analytics engaged in the case of the chapter are explained to full depth by Chapters 6 and 7 of the book, “Discover Statistics with R, [Fields and Miles, 2012]. There will be section references to full-depth explanations throughout the chapter.

The Fields and Miles text is being drawn upon for a reason besides being well written and a friendly read. It has a quality synonymous to all literature drawn upon by this book. The Fields and Miles book is written to explain what readers should want to know rather than everything there is to know. In contrast, this book is written to explain what readers must know to act on the data-driven methods introduced throughout the book.

Throughout the book, the Fields and Miles text will be referenced for a full grasp of statistical principles, correlation analyses, regression and ANOVA. Other texts will be referenced for time series, survival-hazard and hidden variables analyses.

The published example of the chapter qualifies as a standard of methodology. Although it is not from the domain of asset management, it is exactly the steps we would take to measure and test the correlation and linear fit of the variables that are normal to asset management. As the chapter progresses, the reader can take what is being presented and put it to work in the evaluation of their own operation.

Just as for this chapter, the book will often present published, non-asset-management examples. This is so that the reader can have an explanation beyond the actionable go-by-level explanations of this book.

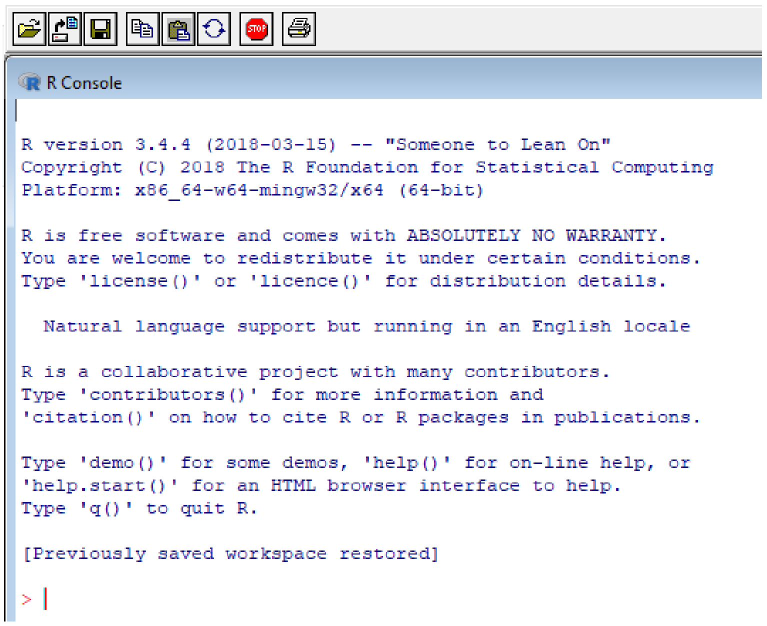
3.2. “R” Installation and Session

There need not be an organization, rather than grassroot, initiative to acquire the “R” software. This is because “R” is freely available for download at https://www.r-project.org/.

Also, at the site are instructions for download, as well as, manuals for working with “R.” Be aware that there are YouTube videos to demonstrate the download and installation process. Furthermore, friendlier texts on the subject of working with “R” are listed in the chapter’s bibliography.

As employees, our computers are integrated within an IT system. Consequently, we will likely need the system administrator to our computer to conduct the download and installation of “R.” The administrator will also be required to load packages into the installed software. Packages will be explained later in the section. In all cases, the administrator’s task is minor and quick.

Like all modern software a session is opened by clicking its icon. Figure 3-1 shows the opening view—the console.



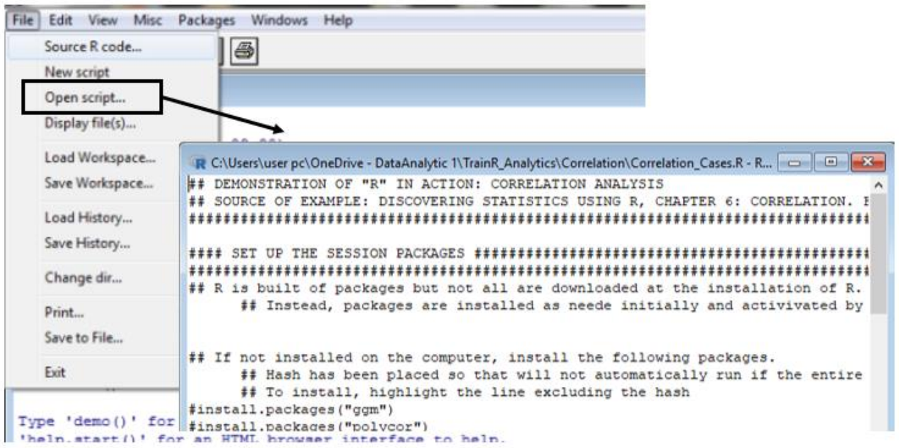
```
R version 3.4.4 (2018-03-15) -- "Someone to Lean On"
Copyright (C) 2018 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

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

[Previously saved workspace restored]

> 
```

Figure 3-1: The console view of “R.”

The nature of the console is that each line of code runs upon pressing the Enter key. At the end of a completed function, the associated output is returned in the console. It is not possible to place a set of commands and then get the collective output. In other words, the console is not a friendly place to work; so we don't.

The hugely more workable view is the script window. It is shown in Figure 3-2. As can be seen, we can start a new script or open an existing one.

Upon clicking the Open Script option, we navigate through our local or server directory to upload the script we want. If we are writing a new script, upon selecting New Script, a blank script view will open. We ultimately save the script as a new file. The procedures are identical to what we are all accustomed to.



```
## DEMONSTRATION OF "R" IN ACTION: CORRELATION ANALYSIS
## SOURCE OF EXAMPLE: DISCOVERING STATISTICS USING R, CHAPTER 6: CORRELATION.
################################################################################

#### SET UP THE SESSION PACKAGES ###############################################
################################################################################
## R is built of packages but not all are downloaded at the installation of R.
      ## Instead, packages are installed as neede initially and activivated by


## If not installed on the computer, install the following packages.
      ## Hash has been placed so that will not automatically run if the entire
      ## To install, highlight the line excluding the hash
#install.packages("ggm")
#install.packages("polycor")
```

```
Type 'demo()' for
'help.start()' for an HTML browser interface to help.
```

Figure 3-2: Script view, were we do our work.

The figure shows the action of opening a script file that was previously developed and saved. A little geek-talk here. A script file is distinguished by its .R extension—seen in the window frame. If we further extend the script name with .txt we would get a Notepad file of the code.

The script window is where we can write, edit and run code instead of working in the console view. In contrast to the console, we can work in a fully flexible manner. We also control when we want the output to be generated relative to the command code.

Along with the code of the analysis, the script can also serve as a document of explanation and paper trail. Notice the hash marks in the code. They convert code to unexecuted text and, thus, allow us to place notes and explanations throughout the script.

We also see the hash marks preceding some command codes. For example, hash marks are placed before the install.packages() function. The placement prevents a one-time command from running indiscriminately as part of every full run. At the same time, we make the command in the working code a paper trail to the code.

There are two ways to run the code. First, highlight the code to be run. Thence, press the run icon on the icon bar at the top of the “R” window for script. The other is to place the cursor in the code line and press the ctrl-r keys.

Chapter 2 explained that everything in “R” is done with functions. In turn, the functions are available from their resident packages. There were approximately 10,000 packages as of 2020.

The most commonly called functions are automatically installed with the initial installation of “R” and automatically opened in all sessions. Others are installed as they are needed for the analytics at hand.

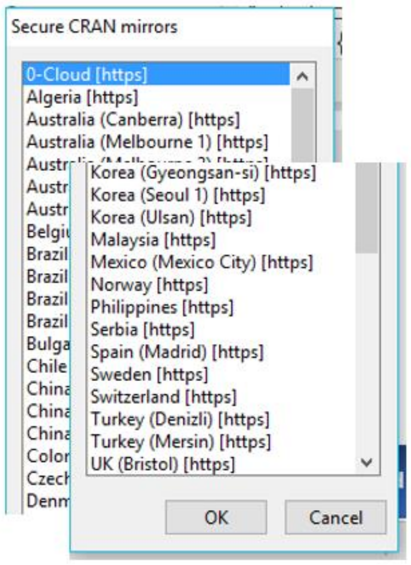
Six packages are required for the analyses demonstrated in this chapter. The code to call them is the `install.packages` function applied as follows:

```
install.packages("ggm")
install.packages("polycor")
install.packages("Hmisc")
install.packages("psych")
install.packages("ggplot2")
install.packages("rlm")
install.packages(MASS)
```

An important detail can be now introduced. “R” is case sensitive. We will get an error message if we violate case. Upon an error message, checking for case is part and partial to checking our code for spelling and punctuation vis-a-vis the go-by—in geek, checking our syntax. Furthermore, we can check each function as we code it and then move to the next.

As we work through the go-by literature to an analytic, the packages will be identified as the code is being presented. Often the packages are arcane. For example, the ggm package is required for "graphical Markov models." However, we have no need to know what that means or what that is—only that we must call it up.

Figure 3-3 shows the process to install a package. For example, the install.packages() function with reference to ggm in the brackets is highlighted and run. The window appears offering choices of mirror servers located around the world. We pick one, click OK and the package will be automatically downloaded and installed in the "R" software. As mentioned, most of us would turn to our system administrator for the task.


Secure CRAN mirrors

0-Cloud [https]
Algeria [https]
Australia (Canberra) [https]
Australia (Melbourne 1) [https]
Korea (Gyeongsan-si) [https]
Korea (Seoul 1) [https]
Korea (Ulsan) [https]
Malaysia [https]
Mexico (Mexico City) [https]
Norway [https]
Philippines [https]
Serbia [https]
Spain (Madrid) [https]
Sweden [https]
Switzerland [https]
Turkey (Denizli) [https]
Turkey (Mersin) [https]
UK (Bristol) [https]

Figure 3-3: Packages are downloaded and installed from a mirror server.

Once installed, it is necessary to open the package in a session. A package only needs to be installed to "R" when it is used for the first time. Thence, for any session using the package, it is only necessary to open it once upstream to its role. The shown `library` function does that for each package with the code:

```
library(ggm); library(Hmisc); library(polycor)
library(psych); library(ggplot2); library(rlm)
library(MASS)
```

3.3. Basics of "R" Demonstrated

Packaged as correlation and regression analyses, the section will introduce the most commonly observed coding. As they emerge in the demonstration, the code will be explained. In this way the reader will become comfortable with following published explanations as actionable go-byes to apply in asset management.

The demonstration will begin with introducing the case and the questions we want to ask and answer with the data. Thence, it will explain how to load data into the session and gain an initial "know-thy-data" perspective of it. Next correlation and partial correlation analyses will be introduced and conducted. Finally, to demonstrate fundamental principles and code to setting up most types of models, the chapter will step lightly into linear regression.

3.3.1. Demonstration Case

Suppose we are given the data set charted in Figure 3-4. They are the cross plots of anxiety to exam performance and revision to exam performance. Revision is a British term for one's approach to studying a topic. The straight line is the linear regression fitted to the cross-plotted points.

The case will follow the published example of Sections 6.5 and 6.6 of Fields and Miles. The data is available for download as Exam Anxiety.dat from the book's website. For the case, the data has been converted to an Excel file, AnxietyParCorr.csv, and made available on the website analytics4strategy.com.

Obviously, the example is not from an asset management operation. However, it is directly relevant whenever our interest is the correlation between variables. We would load our table as directed for the example and change the names of the variables in the code. In other words, the example can be used as our go-by or tool.

In fact, there is a case in asset management literature that explores the comparative correlations of maintenance task types to injuries. The example should be applied to confirm for ourselves the inferences made so confidently in the literature but without the same inspection.

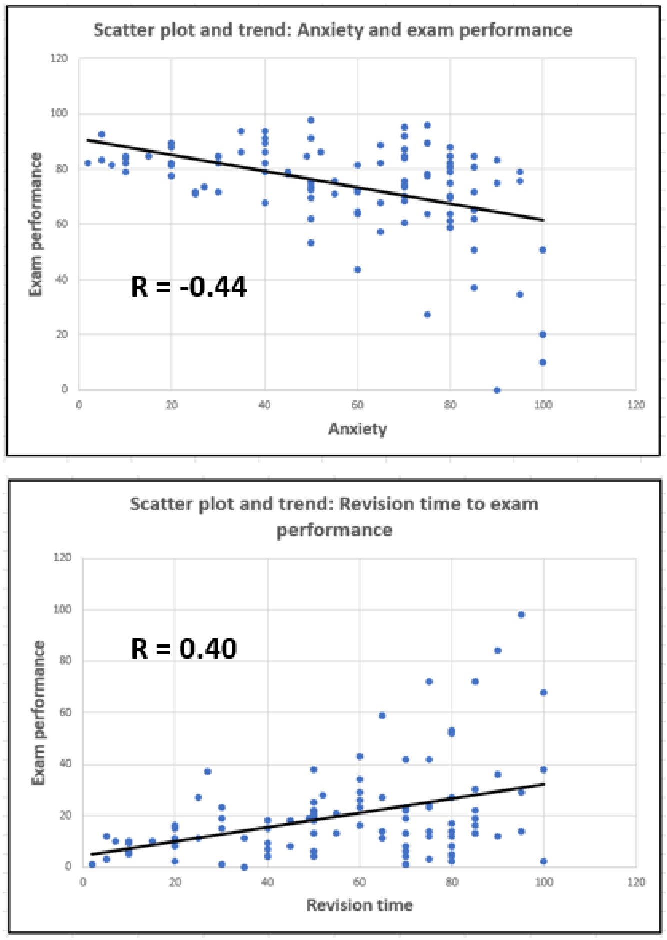


Figure 3-4: Cross plots of paired variables with a fitted line.

Answers to the following questions will emerge as the case unfolds:

- What is the statistical nature of the data—average, median, spread, etc.?
- Is the data normally distributed?
- Are anxiety and revision significantly correlated to exam performance?
- How much of the variance in performance is shared by anxiety and revision time?
- Do the correlations hold if we isolate what is unique between each combination—partial correlation?
- Are anxiety and revision significantly predictive of exam performance?

The code to ask and answer the questions will be presented and explained in the sections to come.

3.3.2. Import Data to the Session

The first action is to import our data into the "R" session. The code to do so is:

```
examData <- ead.csv("C:\\<path>\\Chap3AnxietyParCorr.csv",
   header=TRUE, sep=",")
```

We are showing the case in which our data, a super table, is imported into "R" with the `read.csv` function. The function is used because the super table is a file of comma-separated variables from Excel.

However, there are functions for other types of files. They are `read.xlsx` for .xlsx files and `read.delim` for .txt and .dat files. For Access files we would install and open the RODBC package and set up the location and arguments for importing a query or table from Access.

There are three arguments in the `read.csv` function just as there are for the other read-type functions. The first is the file and its location. The second tells the function if the super table has headers. The third indicates commas or something other as the separator.

The second and third arguments are shown for completeness. However, they are the default arguments. The arguments need not be coded unless our data are exceptions to them.

Let’s talk about the file and location argument. There are options in “R” to identify the files with respect to a “working directory” rather than code out the entire location. However, the full address is shown in the code.

This is preferable because it places a paper trail in the code. This makes it simple to locate the file after we have long forgotten its name and location or to make it possible for recipients of the analytic to find the data without knowing of some working directory and file. In the code, the ellipse represents the complete address for the reader to fill in.

Also note the syntax of the file argument. The location string for Microsoft typically codes with backward slashes. In “R” they must be changed to either a double backward slash as shown in the code or a forward slash.

Another fundamental. Notice the code string `“<-“` in the code line. It assigns what is created by the code to its right to be the “object” to its left. By virtue of the element in the code, the imported table—AnxietyParCorr.csv—is now an object called examData. Notice that throughout the code we refer to the object rather than the csv file.

A comment. Tables can be built from scratch in “R” with standard query language (SQL) by using the `sqldf` function. Beyond what is possible with SQL, we can do much more with “R” code.

As mentioned in Chapter 2, building super tables with SQL and additionally with R coding requires substantial skills. As a measure, the skills for each require six weeks of structured formal training. Therefore, because Access moves SQL coding to the background, we are better served by building super tables in Access and subsequently importing them to R. When additional treatment is called for, the go-by that is being followed for the subject analysis will provide the necessary code.

The data of one or more imported super tables will be manipulated in the conduct of an analytic. However, the code will typically be such that the imported data will remain unchanged—as examData in this case.

The tables built with an analytic can be exported with the `write` function. We may do so to make our super table more super. For example, we would likely want to join the findings of an apparency insight deliverable to the super table.

3.3.3. Know-Thy-Data Analysis

This section will demonstrate a range of descriptive, statistical and graphic insights—know-thy-data. What is demonstrated should be basic to all data to be engaged in insight. It will be apparent that doing so is an easy task of importing the data and subjecting it to the functions of this section as a go-by or tool.

The section will look at data along three dimensions. First is to view the content and type of the variables. Second is to view the data per statistical measures. And third is to view the data graphically.

As always, there are functions to the three dimensions. For the first, we want to be sure we know what is in the table. Three functions immediately come to mind. One, the `head` function, will return the first six rows (records). The other shoe to the function is the `tail` function to return the last six rows. We not are limited to six rows. The argument `n=` allows us to specify a different number of rows.

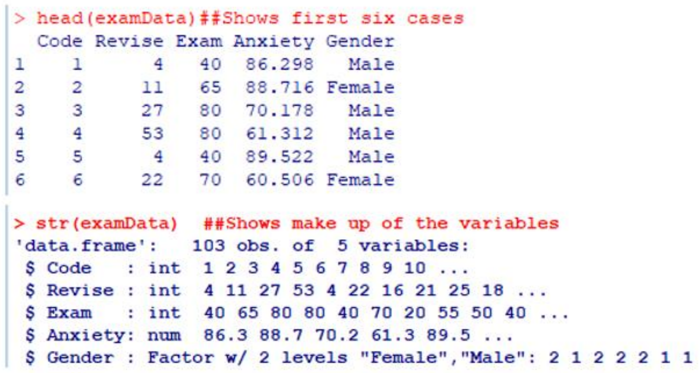
The `str` function, shows the number and nature of the variables and number of records. Its information complements the insight provided by the `head` function.

Both are returned by the following lines of code and the outputs are shown in Figure 3-5:

```
head(examData)        ##Shows first six cases
str(examData)         ##Shows makeup of the variables
```

The head() function returns a small table representative of the parent table. We can see that one, Code, is a house-keeping variable of no use to us. We can also see that we could run our analytics while controlling for gender; which we will not in this example.

From the `str` function we get further descriptions of the variables. We see that we have what is called by “R” a “data frame” with 103 observations (records). Three of the variables are integers, one a numeric and one a factor. For the factor variable, gender, we see that the categories are female and male. For all, we see the first ten records.



```
> head(examData)##Shows first six cases
  Code Revise Exam Anxiety Gender
1    1      4   40  86.298   Male
2    2     11   65  88.716 Female
3    3     27   80  70.178   Male
4    4     53   80  61.312   Male
5    5      4   40  89.522   Male
6    6     22   70  60.506 Female

> str(examData)  ##Shows make up of the variables
'data.frame':   103 obs. of  5 variables:
 $ Code    : int  1 2 3 4 5 6 7 8 9 10 ...
 $ Revise  : int  4 11 27 53 4 22 16 21 25 18 ...
 $ Exam    : int  40 65 80 80 40 70 20 55 50 40 ...
 $ Anxiety: num  86.3 88.7 70.2 61.3 89.5 ...
 $ Gender : Factor w/ 2 levels "Female","Male": 2 1 2 2 2 1 1
```

Figure 3-5: Head and summary views the data.

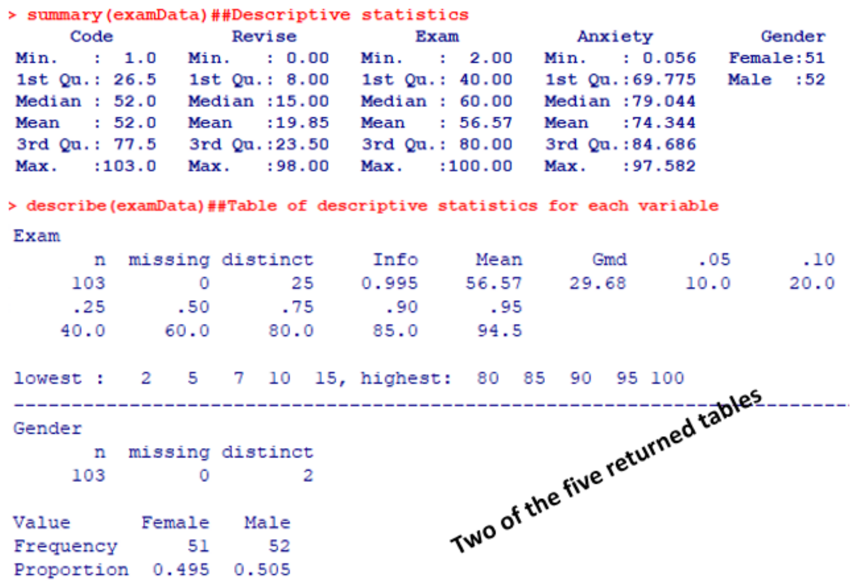
The second dimension of insight gives us a statistical overview of our data set rather than the basic “what it is” presentation. Two such functions are `summary` and `describe`. They are coded as follows and the outputs are shown in Figure 3-6:

```
summary(examData)     ##Descriptive statistics
describe(examData)    ##Table of descriptive statistics
```

The `summary` function provides shape-type insight. They are min-max, mean, median and quartiles for integer and numeric variables. The categories and counts are returned for the factor variable.

The `describe` function provides additional information. Of those returned, the returned non-esoteric insights to the variables are number of observations, number missing, number of distinct cases and quantiles. For the factor variables we receive a count, proportion of the categories and count of missing records (empty cells). The figure has been snipped to show only two of the five variables in the returned table.

Finally, we want graphic insight. Figure 3-4 was a traditional preliminary insight. We will demonstrate two additional cases. First is the case of a pairs-panel as shown in Figure 3-7. Second is a graphical assessment of the distribution of the variables as a normal distribution. It is shown in Figure 3-8.



```
> summary(examData)##Descriptive statistics
      Code           Revise           Exam            Anxiety          Gender
 Min.   :  1.0   Min.   : 0.00   Min.   :  2.00   Min.   : 0.056   Female:51
 1st Qu.: 26.5   1st Qu.: 8.00   1st Qu.: 40.00   1st Qu.:69.775   Male  :52
 Median : 52.0   Median :15.00   Median : 60.00   Median :79.044
 Mean   : 52.0   Mean   :19.85   Mean   : 56.57   Mean   :74.344
 3rd Qu.: 77.5   3rd Qu.:23.50   3rd Qu.: 80.00   3rd Qu.:84.686
 Max.   :103.0   Max.   :98.00   Max.   :100.00   Max.   :97.582

> describe(examData)##Table of descriptive statistics for each variable
Exam
       n  missing distinct     Info     Mean      Gmd      .05      .10
     103        0       25    0.995    56.57    29.68     10.0     20.0
     .25      .50      .75      .90      .95
    40.0     60.0     80.0     85.0     94.5

lowest :   2   5   7  10  15, highest:  80  85  90  95 100
------------------------------------------------------------------------
Gender
       n  missing distinct
     103        0        2

Value       Female   Male
Frequency       51     52
Proportion   0.495  0.505
```

Figure 3-6: Two statistical views of the data set.

First let's look at the graphic from the `pairs.panel` function of the psych package for a tremendous amount of insight. In it, we can inspect the shape or distribution of each variable in the data set. We can inspect the cross-plot relationship of each variable with all other variables. The fitted smooth line gives a sense of pattern to the cross plots. The oval shape is called the correlation ellipse; the more elongated the greater correlation. The large dot indicates the mean value of plot points.

The code to return the pairs panel of Figure 3-7 is as follows:

```
#Figure 3-7
pairs.panels(examData[c("Exam","Anxiety","Revise")])
```

It is good practice to confirm that our data is normally distributed. Doing so is a mandatory step to evaluate the truthfulness of an analytic. If the finding is not a normal distribution, we use an alternative method to the analytic or transform the variables within the analytic.

We were given the cross-plot and line fit (Figure 3-4) of anxiety and revision plotted to exam. Upon inspection, we can see that the points do not fall in a consistent band around the plotted linear fit. This was the first clue that the data are not normally distributed. Furthermore, the histograms of Figure 3-7 do not look like normal distributions.

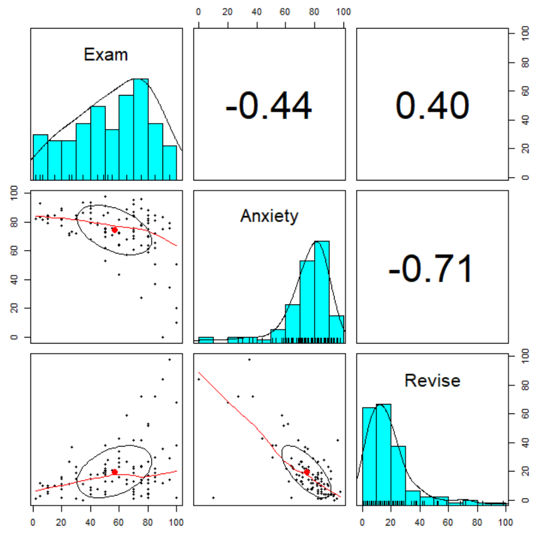


Figure 3-7: Pairs panel perspective of the numeric variables in the data set.

We can test normalcy with graphic insight. However, we should note that the `shapiro.test` function can be used to test if the data is significantly different from the normal distribution.[1]

The graphic method is to build and view a q-q graphic as shown in Figure 3-8. If the points of a variable fall along its line, the data is normally distributed.

In the following code to the generate the figure there are several basic elements of coding to recognize:

```
par(mfrow = c(2, 2))
anx<- qqnorm(examData$Anxiety, main = "Q-Q Anxiety ");
   qqline(examData$Anxiety)
rev<- qqnorm(examData$Revise, main = "Q-Q Revise ");
   qqline(examData$Revise)
```

[1] See Field and Miles, Section 5.6 for a full explanation of the shapiro.test() function.

```
exm<- qqnorm(examData$Exam, main = "Q-Q Exam ");
   qqline(examData$Exam)
par(mfrow = c(1, 1)) #returns to 1 x 1 pictures
```

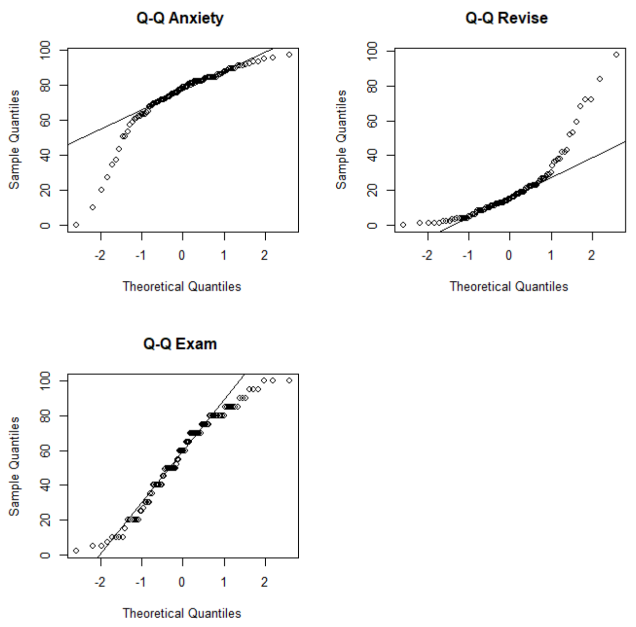


Figure 3-8: A plot point of a normal distributed variable will fall along the straight line.

First, how did we get the three charts in one figure? The answer is the `par` function in the first line and its argument `mfrow = c(2,2))`. The code creates a 2-by-2 matrix of graphics. If we change the `c()` element we can create any combination of rows and columns.

We have built an output sandwich with the `par` function. Note that the `par` function appears again at the end of the block of code. By coding `c(1,1)`, any graphic after the line of code will present in a single frame.

We can also see layered charting in action. Inspect the line of code beginning just after `anx<-`. In it is the code to generate the q-q graph for the anxiety variable.

The code line includes the `qqnorm` and `qqline` functions. Both are individual graphs of the subject variable to the line of code.

If we highlighted and ran the `qqnorm` code, we would get the plotted of points. If we did the same with `qqline` code, we would get a straight line. It represents how a plot of data with the mean and variance of the data should form as quantiles if it were a normal distribution.

As a coded command, the qqline element overlays the straight line on the plotted points. In this case, we can readily conclude the data is not normally distributed.

There is another element to recognize in the code. It is one of the ways we can specify variables to functions.

Notice the syntax `examData$Anxiety` in the code to the first graphic. Also notice the same pattern in the code in which the other two variables are engaged for evaluation. The code line tells the function that we are working with a variable (after the $) from a table (before the $)

There are other ways to identify variables. Some will arise as the chapter progresses. However, the advantage of the `table$variable` syntax is that we leave a paper trail in the code.

3.3.4. Correlation Analysis

Now to conduct two correlation analyses. One between anxiety and exam. The other between revision and exam. However, before jumping in, let's make sure we do not confuse correlation analysis with regression analysis.

Correlation is the extent that two variables move together with respect to their own means. The similarity may have no meaning—be incidental. In contrast, regression quantifies how strongly a predictor variable is related to an outcome variable.

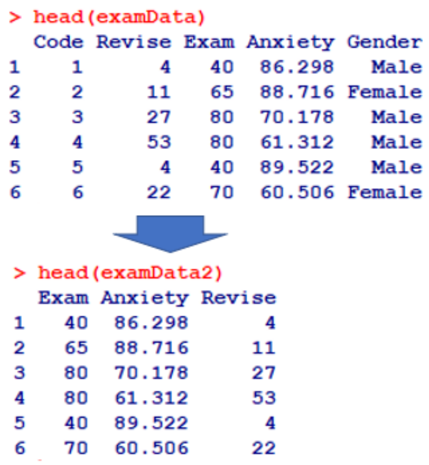
The fitted line in Figure 3-4 is a regression relationship to the cross-plot points. The horizontal axis is the predictor variable to the plots. The R value (correlation coefficient) that is inserted in the figure is a computation upon the collective paired points.[2]

We will take a step here that is a necessary demonstration of "R" more than it is necessary to conduct a correlation analysis. We will filter

[2] See Fields and Miles Section 6.3 for a full explanation of the correlation coefficient.

the examData table to a new table object—examData2—with only the three variables of the correlation analysis to come. We will, thence, subject it to correlation analysis.

When data is assigned to an object—examData—the object looks like a table as shown in the upper part of Figure 3-9. It looks like a table to us, but it is called a "data frame" by "R."



```
> head(examData)
  Code Revise Exam Anxiety Gender
1    1      4   40  86.298   Male
2    2     11   65  88.716 Female
3    3     27   80  70.178   Male
4    4     53   80  61.312   Male
5    5      4   40  89.522   Male
6    6     22   70  60.506 Female
```

```
> head(examData2)
  Exam Anxiety Revise
1   40  86.298      4
2   65  88.716     11
3   80  70.178     27
4   80  61.312     53
5   40  89.522      4
6   70  60.506     22
```

Figure 3-9: Table built by filtering another.

We want to extract the data frame (examData2) in the lower part of the figure from the data frame (examData) in the upper part. The code to return examData2 is:

```
#Figure 3-9 lower
examData2<- examData[,c("Exam","Anxiety","Revise")]
```

We are taking the extraneous step of forming the new data frame object because the code to do so demonstrates two fundamentals of "R." They are the square brackets, `[]`, and the combine function, `c()`.

What can be seen within the brackets is the act of filtering a table by row and column. Notice the comma that divides the bracketed space into right and left sides. The distinction is for filtering rows and columns.

The absence of code to the left of the comma in this case causes all rows of the source table to appear in the newly created data frame.

However, filters placed in the position can be character, numeric or calculated.

In this case, by the code placed to the right of the comma, we are commanding a table with variables and order of exam, anxiety and revision. The enabling combine function, `c()`, plays large in "R." We saw it previously but will explain it now. In this case, the function presents a list of variables. It keeps coding simple. Also notice that the elements, when text, are coded with parenthesis.

As noted already, the order has been changed by which the variables will appear as columns in the new table. We could have referred to the positions of each variable in the source data frame and specified the order in the new data frame with `c(3,4,2)`. The numbers refer to column positions of the source table. The order of appearance in the `c()` function decides the order of appearance in the new table. Also notice that when coding the function with numbers, no parentheses are involved.[3]

This is a good place to explain how to find guidance to any function. First, of course, we need to know there is a function or have an idea of a function. We can start with an open-ended query of the internet from which we will likely discover an appropriate function to work with. Thence, we can get a full-depth explanation of the discovered function using the `help` function. Alternately, we can query the internet for the same detail.

In this case, the `help` function would have taken us to a website with explanation, examples and data for the examples. Calling the code `help(cor)`, we would receive the following explanation of the `cor` function:

cor(x, y, use = "everything", method = "correlation type")

Arguments:

X	A numeric variable, matrix or data frame.
Y	NULL (default) or a vector, matrix or data frame with compatible dimensions to x.

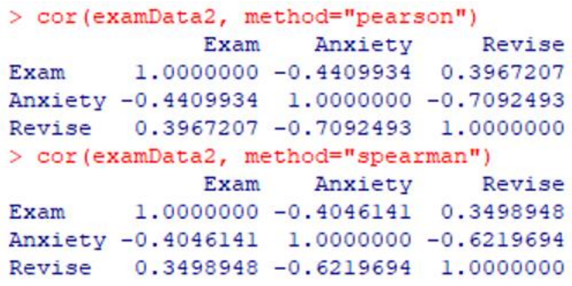
[3] See Field and Miles, Sections 3.5 and 3.9 for additional methods to work with tables.

Use	An optional character string giving a method for computing in the presence of missing values. This must be (an abbreviation of) one of the strings "everything", "all.obs", "complete.obs", "na.or.complete", or "pairwise.complete.obs."[4]
Method	A character string indicating which correlation coefficient is to be computed. One of "pearson" (default), "kendall", or "spearman": can be abbreviated.[5]

Now to put the correlation function, `cor`, to work. Because the earlier know-thy-data stage of the procedure revealed the variables to not be normally distributed, we will use the Pearson and Spearman methods; the latter because of non-normality. The 'Use' argument is omitted because the data frame has no missing data. The 'X' and 'Y' arguments are not necessary because we are using the data frame examData2 as the source data.

The code is as follows and the output is shown in Figure 3-10:

```
#Figure 3-10
cor(examData2, method="pearson")
cor(examData2, method="spearman")
```



```
> cor(examData2, method="pearson")
              Exam    Anxiety     Revise
Exam     1.0000000 -0.4409934  0.3967207
Anxiety -0.4409934  1.0000000 -0.7092493
Revise   0.3967207 -0.7092493  1.0000000
> cor(examData2, method="spearman")
              Exam    Anxiety     Revise
Exam     1.0000000 -0.4046141  0.3498948
Anxiety -0.4046141  1.0000000 -0.6219694
Revise   0.3498948 -0.6219694  1.0000000
```

Figure 3-10: Output of the correlation analysis.

Notice that the tables return different correlation coefficients. This is because our data is not a normal distribution. To deal with the non-

[4] See Field and Miles Section 6.5.3 for an explanation of each case to the "Use" argument.

[5] See Field and Miles, Sections 6.5.4-6 for an explanation of Pearson, Kendall and Spearman correlation.

normal, the Spearman uses a non-parametric method. The strength of the correlations returned by the more realistic Spearman method show to be less than reported with Pearson.

It is always necessary to confirm that the coefficients are significant. How strongly is the correlation between each pair of variables? It is more than just by chance?

This is evaluated with hypotheses testing. The principle is to test the computed correlation against the null hypotheses. It is that there is no correlation between two variables. In other words, the null hypotheses is that the correlation is insignificant; “0.”

As in Figure 3-10, we are given coefficients. However, that does not mean they are significant because they are not “0.” We must always confirm another measure of significance.

We must test that “0” does not appear between the upper and lower limits of a confidence level—e.g., 95 percent. If “0” appears, it means that our correlation is non-existent. We are drowning in a pool with an average depth of one foot.

The test to conduct of the correlation coefficients of Figure 3-10 is what is called the p-value. The smaller the value, the greater the significance. In this example, the p-value for acceptance as significant must be 5 percent or less. This is because we have decided that our statistic much have a confidence of 95 percent.

Before jumping in to the functions to test for significance, it is necessary to introduce another fundamental of “R.” Matrix and vector in contrast to data frame.

A matrix and data frame look alike but are different in nature. A matrix is a table of purely numeric values. As already seen, a data frame is a table of character, integer and numeric values.

A vector can be character or numeric. However, if you saw one you would describe it as a table with a single variable. In fact, the data frame and matrix objects are actually a construct of vectors. However, a vector need not be thought of as a row or a column.

Now we will convert the data frame to a matrix with the as.matrix() function and assign the result to a new table object, examMatrix. The code is as follows:

```
examMatrix<- as.matrix(examData2)
```

We are making the conversion because the function we will use to determine p-values, requires a matrix. Running the code with a data frame would return an error message.

We will generate our p-values with the rcorr() function and the Pearson and Spearmen coefficients. The code is as follows and the output is shown in Figure 3-11:

```
#Figure 3-11
rcorr(examMatrix, type = "pearson")
rcorr(examMatrix, type = "spearman")
```

The output includes the correlation matrices previously returned by `rcor`. The important new information are the p-values.

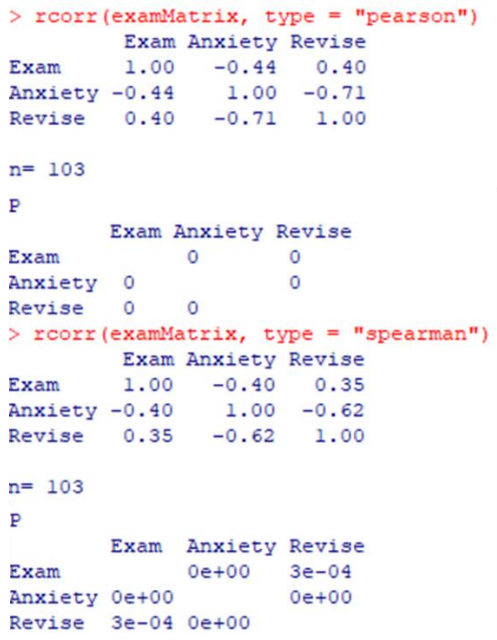
Both Pearson or Spearman return a p-value far smaller than 5 percent. Therefore, the coefficients are highly significant. However, the significance is shown by the Spearman method as weaker than Pearson.

Let's clarify the p-value in the table as compared to an acceptable value of 5 percent. The computed p-value indicates the confidence interval we can give to the correlation as not including "0." As an example, for a p-value of 1.2 percent, our confidence is 98.8 percent.

We would also like to get insight into the upper and lower limits of confidence. Let's state in plain English the meaning of p-value and confidence with respect to the limits.

For our given 95 percent confidence, the limits will not include "0." The confidence from the computed p-value, if it were the basis for the confidence limit, is the limits at which a notch farther would cause "0" to appear within the limits.

We can use the `cor.test` function to get the confidence limits, but only one pair at a time. We could have used the function to get the correlation coefficients and p-values. However, it requires more coding of a lazy SME such as me.

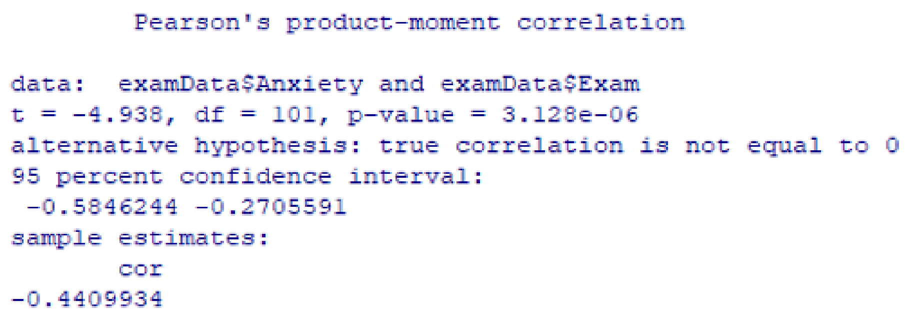


```
> rcorr(examMatrix, type = "pearson")
         Exam Anxiety Revise
Exam     1.00   -0.44   0.40
Anxiety -0.44    1.00  -0.71
Revise   0.40   -0.71   1.00

n= 103

P
        Exam Anxiety Revise
Exam          0       0
Anxiety  0            0
Revise   0    0
> rcorr(examMatrix, type = "spearman")
         Exam Anxiety Revise
Exam     1.00   -0.40   0.35
Anxiety -0.40    1.00  -0.62
Revise   0.35   -0.62   1.00

n= 103

P
        Exam   Anxiety Revise
Exam           0e+00   3e-04
Anxiety 0e+00          0e+00
Revise  3e-04  0e+00
```

Figure 3-11: P-values show significance under Pearson and Spearman.

Note that we can only get an interval for the Pearson method. The code for all three pairs is as follows and the output for the Anxiety, Exam pair is shown in Figure 3-12.

```
#Figure 3-12
cor.test(examData$Anxiety, examData$Exam,
   method="pearson")
cor.test(examData$Anxiety, examData$Revise,
   method="pearson")
cor.test(examData$Revise, examData$Exam, method="pearson")
```



```
        Pearson's product-moment correlation

data:  examData$Anxiety and examData$Exam
t = -4.938, df = 101, p-value = 3.128e-06
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 -0.5846244 -0.2705591
sample estimates:
       cor
-0.4409934
```

Figure 3-12: Confidence limits to the Anxiety-Exam pair.

In the output we receive once again the coefficient and p-value to the pair. Additionally, we now receive the upper and lower limits to 95 percent confidence. If we want to explore the limits for the computed p-value, we would do so by the setting the “conf.level=” argument in the cor.test() function to the confidence level equal to one minus the p-value.

There is another fundamental point to make. Notice that “0” does not appear between the limits of a 95 percent confidence interval. We would expect this because the computed p-value is 5 percent or less.

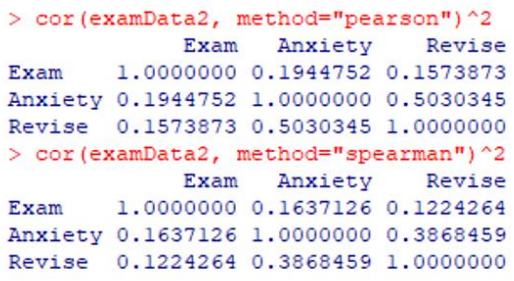
3.3.5. Partial Correlation Analysis

Now is the time to introduce two types of correlation and the difference. They are correlation (R) and correlation-squared (R^2).

As previously defined, correlation is a measure of the extent to which two variables move together with respect to their own means. Correlation-squared is the extent that the variance in one is shared by the other.

The latter of the two is computed as `cor` squared. The code using Pearson and Spearman is simply as follows and the output is shown in Figure 3-13:

```
#Figure 3-13
cor(examData2, method="pearson")^2
cor(examData2, method="spearman")^2
```



```
> cor(examData2, method="pearson")^2
              Exam   Anxiety    Revise
Exam     1.0000000 0.1944752 0.1573873
Anxiety  0.1944752 1.0000000 0.5030345
Revise   0.1573873 0.5030345 1.0000000
> cor(examData2, method="spearman")^2
              Exam   Anxiety    Revise
Exam     1.0000000 0.1637126 0.1224264
Anxiety  0.1637126 1.0000000 0.3868459
Revise   0.1224264 0.3868459 1.0000000
```

Figure 3-13: Correlation squared for shared variance.

Both measures of correlation can be misleading because pairs hide their correlation to a third or more variables. Therefore, we need to determine the unique correlation between two variables rather than disregard the correlation shared with other variables. The concept is shown in Figure 3-14.

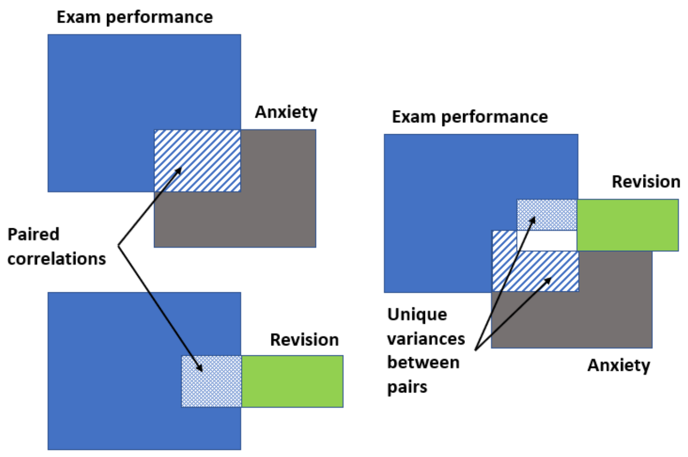


Figure 3-14: Full- and partial-correlation between two variables.

To the left we see the comparative paired correlations between anxiety and exam, and revision and exam. Shown is what the calculation of correlation and correlation-squared have been to this point. It is also the typical perspective of the meaning of correlation.

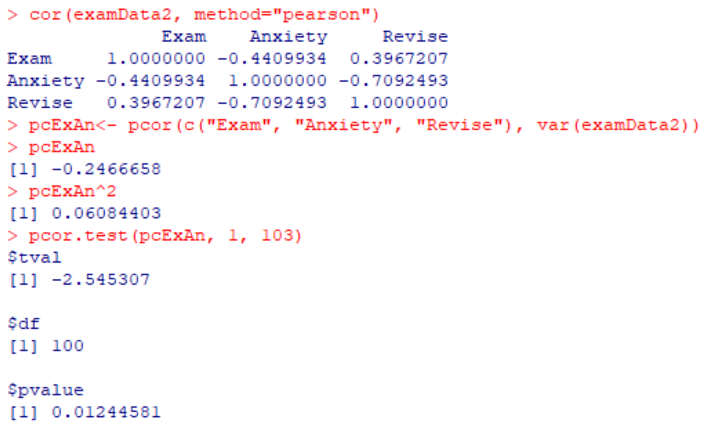
To the right in the figure we can see the lurking issue. Some part of the correlation of the paired set of exam to anxiety is shared with revision.

Therefore, correlation and correlation-squared can be misleading if we disregard associations with other variables. There are, of course, functions for that. We want to determine what is called partial correlation and confirm its significance.

Let's look at the partials of the exam-anxiety pair. The code is as follows and the output is presented in Figure 3-15:

```
#Figure 3-15
cor(examData2, method="pearson")
pcExAn<- pcor(c("Exam", "Anxiety", "Revise"),
   var(examData2))
pcExAn
pcExAn^2
pcor.test(pcExAn, 1, 103)
```

The `cor` line of the code returns the full correlation that was generated with the same function earlier in the chapter. The `pcor` function will return a partial correlation. The output of the `pcor` function is assigned to the pcExAn object.



```
> cor(examData2, method="pearson")
               Exam    Anxiety     Revise
Exam      1.0000000 -0.4409934  0.3967207
Anxiety  -0.4409934  1.0000000 -0.7092493
Revise    0.3967207 -0.7092493  1.0000000
> pcExAn<- pcor(c("Exam", "Anxiety", "Revise"), var(examData2))
> pcExAn
[1] -0.2466658
> pcExAn^2
[1] 0.06084403
> pcor.test(pcExAn, 1, 103)
$tval
[1] -2.545307

$df
[1] 100

$pvalue
[1] 0.01244581
```

Figure 3-15: Partial correlation to exam-anxiety pair.

The partial correlation is returned when the object is called up by the next line of code. In the figure it is apparent that the full- and partial-correlations are very different. The true correlation is far weaker than we would be led to believe with full correlation. In fact, it is below the rule-of-thumb threshold of 30 percent as a strong correlation.

Mathematically we know that the before and after correlation-squared will also be even more disparate. When calling the code line `pcExAn^2`, the correlation-squared measure is returned.

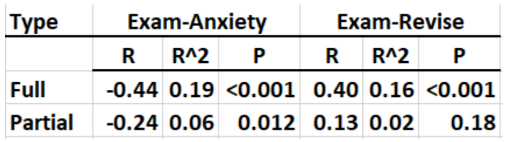
Now let’s look at the `pcor` function. It has two arguments. The second tells the function from which table the variables are taken from. The first shows three variables in the `c()` function. The first two are the variables of interest. The third is the variable for which there is shared correlation and the one we want to exclude from the partial-correlation.

However, know that the `pcor` function does not limit us to sets of three variables. The function allows us to engage any number of variables. The first two in the list are always the partial correlation variables.

The function `pcor.test` returns three insights. The important one is `$pvalue`. It shows that the partial correlation holds. It is less than the 5 percent to judge the correlation as significant. However, because it is substantially greater than the p-value (<0.001) of the full correlation, the significance of correlation is much less.

To determine the confidence limits to the partial correlation, there are three arguments to the `pcor.test` function. The first is the object model `pcExAn`. The second is the number of controlled variables; “1”. And the third is the number of observations.

The Excel table of Figure 3-16 summarizes the findings for the respective pairs. Anxiety has a significant unique correlation to exam performance, whereas, revision does not.



Type	Exam-Anxiety			Exam-Revise		
	R	R^2	P	R	R^2	P
Full	-0.44	0.19	<0.001	0.40	0.16	<0.001
Partial	-0.24	0.06	0.012	0.13	0.02	0.18

Figure 3-16: Findings from full- and partial-correlation analysis.

If we were a subject matter expert in testing, we would ponder why what seems intuitively correlated to us is not. We would have discovered to seek other variables with respect to revision because not all approaches appear to be equally effective.

3.3.6. Regression Analysis

The section will demonstrate regression analysis in contrast to correlation analysis. [6] It is an opportunity to introduce the structure of “formulas” that appears typically in most analytic models.

There are multiple steps to build and evaluate linear regression. Here we will look only at the first step which is to determine the best predictor variables to the model. The step may begin with the correlation analyses of variables as shown throughout the chapter. Thence, we would evaluate the model with different configurations of variables. Finally, we would retain the predictive variables that are significant to the outcome variable.

Although not introduced here, a number of steps take place to discover and work around issues to the model until it is judged as valid. Although not introduced in this chapter, the steps are explained in Chapter 8.

As mentioned earlier, we must not to forget that the coefficients of regression and correlation are different measures. A correlation coefficient indicates the extent to which two variables move together with respect to their own means. A regression coefficient indicates the relationship of a unit change in a predictor variable to the outcome variable. However, the issues of significance and confidence are equally involved.

A reason for demonstrating regression is that, as models, they are set up in much the same way for most types of models and by most software. The `help(lm)` code would take us to the site to present and explain the following code:

```
lm(formula, data, subset, weights, na.action, method =
   "qr", model = TRUE, x = FALSE, y = FALSE, qr = TRUE,
   singular.ok = TRUE, contrasts = NULL, offset, ...)
```

Just as for correlation analysis, the `lm` function is set up by the choices we make for its arguments. Explanations and examples of the

[6] See Field and Miles Chapters 7 and 8 for a full explanation of linear and logistic regression.

options are readily available from the internet and will not be presented here. However, as for many functions, we rarely touch most of the arguments because the defaults typically apply. Accordingly, the shown code may reduce to `lm(formula, data)`.

The model to this example reduces to the following:

```
#Figure 3-17 main effects
examPred <- lm(Exam ~ Anxiety + Revise, examData)
summary(examPred)
confint(examPred)
```

We are left with two arguments in the model function, lm(). One is the reference to the source table of data; examData. The other is the model or formula of variables.

The code is assigned to an object we have named examPred. It is our model. In it, we will learn the strength of relation the predictor variables, anxiety and revision, have with the exam variables.

In the model, the syntax "~" is the equivalent to "=." To the right of the symbol are the predictor variables. To the left is the outcome variable.

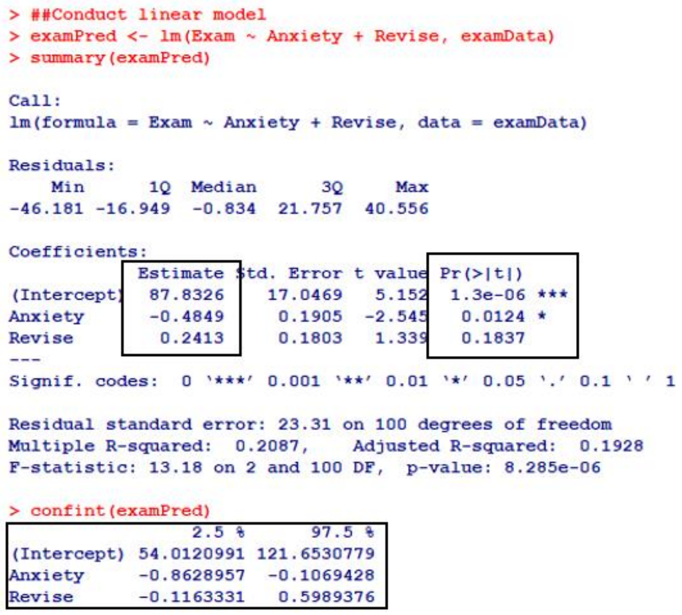
The `summary` function, with our model as its argument, calls up the model shown in Figure 3-17. Also shown, the `confint` function, with the model as its argument, returns the 95 percent confidence interval.

Without getting deeply into the interpretation of regression, the first point of interest is the regression coefficients to the predictor variables. The regression coefficients tell us that for one unit of change in the predictor variable, the outcome will change by the amount of the coefficient.

The second point is significance. It is interpreted through the p-value just as it was for correlation. Third are the upper and lower limits to the 95 percent confidence interval.

Whether the regression coefficients are significant is shown by the p-value and if "0" appears in the 95 percent confidence intervals. Notice that they replicate the partial correlations. Anxiety is significant to the outcome, whereas, revision is not.

In modeling, insignificant variables are removed. Therefore, our regression model would be reduced to anxiety as the predictor variable and exam as the outcome variable.



```
> ##Conduct linear model
> examPred <- lm(Exam ~ Anxiety + Revise, examData)
> summary(examPred)

Call:
lm(formula = Exam ~ Anxiety + Revise, data = examData)

Residuals:
    Min      1Q  Median      3Q     Max 
-46.181 -16.949  -0.834  21.757  40.556 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept)  87.8326    17.0469   5.152  1.3e-06 ***
Anxiety      -0.4849     0.1905  -2.545   0.0124 *  
Revise        0.2413     0.1803   1.339   0.1837    
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 23.31 on 100 degrees of freedom
Multiple R-squared:  0.2087,    Adjusted R-squared:  0.1928 
F-statistic: 13.18 on 2 and 100 DF,  p-value: 8.285e-06

> confint(examPred)
                 2.5 %      97.5 %
(Intercept) 54.0120991 121.6530779
Anxiety     -0.8628957  -0.1069428
Revise      -0.1163331   0.5989376
```

Figure 3-17: The linear regression model of main effects.

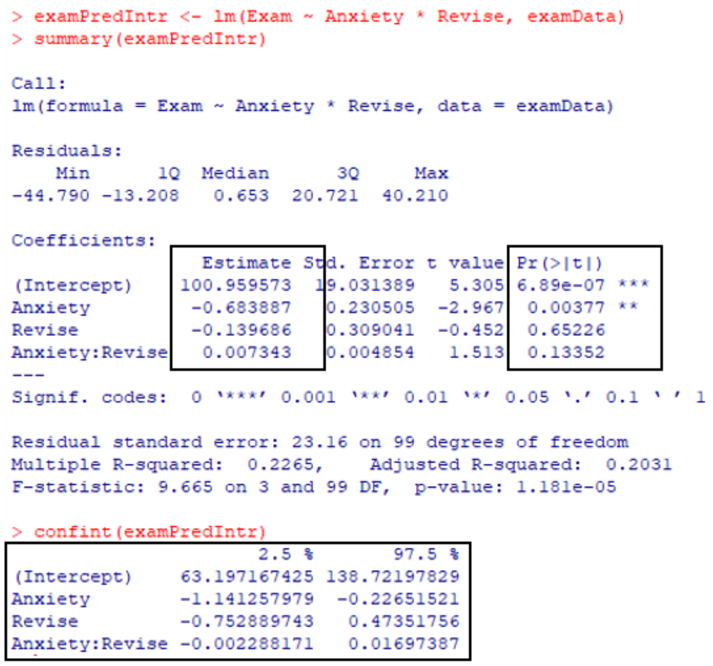
Let's demonstrate another aspect of modeling—interaction between variables. In statistics, an interaction may arise when considering the relationship among three or more variables. It describes a situation in which the effect of one predictor variable on an outcome depends on the state of a second predictor variable.An interaction can entail two or more predictor variables.

Let's look at the set up and interpretation of models with an interaction. Essentially, we run a model with interactive variables and determine their presence by virtue of the significance reported for each. The immediate difference is apparent in the coded formula.

```
#Figure 3-18 with interaction
examPredIntr <- lm(Exam ~ Anxiety * Revise, examData)
summary(examPredIntr)
confint(examPredIntr)
```

Notice that the "+" in the model is replaced with "*." This creates a new variable in the model which is the interactive variable. The substitution would be the case if we had additional predictor variables.

The output of the interaction model to interpret is shown in Figure 3-18.



```
> examPredIntr <- lm(Exam ~ Anxiety * Revise, examData)
> summary(examPredIntr)

Call:
lm(formula = Exam ~ Anxiety * Revise, data = examData)

Residuals:
    Min      1Q  Median      3Q     Max 
-44.790 -13.208   0.653  20.721  40.210 

Coefficients:
                 Estimate Std. Error t value Pr(>|t|)    
(Intercept)    100.959573  19.031389   5.305 6.89e-07 ***
Anxiety         -0.683887   0.230505  -2.967  0.00377 ** 
Revise          -0.139686   0.309041  -0.452  0.65226    
Anxiety:Revise   0.007343   0.004854   1.513  0.13352    
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 23.16 on 99 degrees of freedom
Multiple R-squared:  0.2265,	Adjusted R-squared:  0.2031 
F-statistic: 9.665 on 3 and 99 DF,  p-value: 1.181e-05

> confint(examPredIntr)
                      2.5 %        97.5 %
(Intercept)    63.197167425 138.72197829
Anxiety        -1.141257979  -0.22651521
Revise         -0.752889743   0.47351756
Anxiety:Revise -0.002288171   0.01697387
```

Figure 3-18: Model with interaction of two predictor variables.

There are three things to note. First, a new variable appears in the output as the interactive variable; `anxiety:revise`. Second, the p-value to the interaction is greater than 5 percent and its coefficient shows a weak relationship to exam. Third, the findings for the interactive variable are accentuated by the presence of "0" within the confidence interval. In other words, the signs change for the upper and lower limits.

At this point we need to make a note concerning interpretation. As long as there are interactive variables in a model, we cannot draw meaning from the "main effect" variables—anxiety and revision.

Let’s expand the preceding two findings to three predictor variable models. There would be one three-way interactive variable and three two-way interactive variables. We would evaluate and eliminate interactive and main effect variables downward from the three-way variable. The significance to any lower-level variable to a higher-level variable cannot be interpreted until any associated higher-level variables are removed.

We may choose to run the interaction model first rather than second. Based on finding insignificance we would return to the model of Figure 3-17 in which only main effects are modeled. In turn, the model of Figure 3-17 revealed that we would reduce the model to anxiety as the predictor variable to exam as the outcome variable.

Recall that during the know-thy-data phase of analysis we discovered the three variables to the model are not normally distributed. In such a case, we should go to a “robust” linear regression—meaning the model’s algorithm is not influenced by the distribution. Therefore, let’s subject our data to the `rlm` function for robust models.

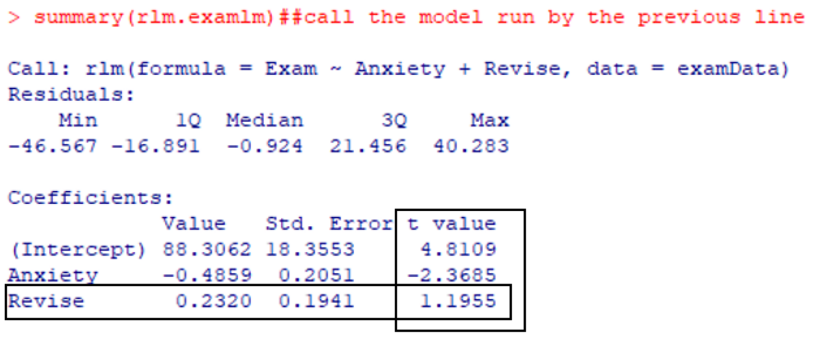
The code is as follows and the output is shown in Figure 3-19:

```
#Figure 3-19
rlm.examlm <- rlm(Exam ~ Anxiety + Revise, data =
   examData)
summary(rlm.examlm)
```

The output does not include a p-value. Instead, the t-value tells the story. We can make our interpretation on the t-value because it resides behind the curtains in the calculation of p-value.

A p-value equal to or less than 0.05 with 100 degrees of freedom is indicated when the t-value is greater than or equal to the cut-off at 1.962 absolute. Anxiety shows to be significant because it exceeds the cut-off. Revise does not.

However, the t-value for regular regression is greater than it will be for robust regression. This demonstrates that robust regression would more realistically deal with the non-normal distribution of the model variables.



```
> summary(rlm.examlm)##call the model run by the previous line

Call: rlm(formula = Exam ~ Anxiety + Revise, data = examData)
Residuals:
    Min      1Q  Median      3Q     Max 
-46.567 -16.891  -0.924  21.456  40.283 

Coefficients:
            Value   Std. Error t value
(Intercept) 88.3062 18.3553     4.8109
Anxiety     -0.4859  0.2051    -2.3685
Revise       0.2320  0.1941     1.1955
```

Figure 3-19: Robust regression tells the same story but different strength of relationship.

3.4. Challenge Taken

The challenge the book has taken on is how to explain data-driven asset management to a workable depth because a full depth explanation would require that I write, and you read approximately 2,500 pages—and that is not going to happen. The chapter has demonstrated how the challenge will be met throughout the book.

The obvious purpose of the chapter was to bring its readers to have a working critical-mass of skills with "R." If the asset management organization does not gain and migrate the skills of "R," beginning with a few role holders, it cannot move beyond system reports and recountive insight deliverables.

Although critical mass skills with the analytic software are a prerequisite to data-driven operations, notice that another purpose of the chapter is to present and explain three of the many insight deliverables for asset management that will arise with each chapter of the book. For this chapter, they are to know-thy-data and to evaluate the correlation and strength of relationship between variables. Skills in the software are accumulated in the context of conducting the methods.

Accordingly, the chapter is representative of how the book will take on the challenge of providing a working depth of explanation to data-driven asset management. Each chapter, as its objective, will introduce insight deliverables applicable to functional aspects of asset management. Thence, they will present an integrated explanation of the insight, the associated data analytic methods and their conduct with the triad of software. Along the way, the book will refer readers to published full-depth explanations.

For reasons of practicality, some of the examples are representative rather depicting an actual functional aspect of asset management. That has been the case for the chapter.

However, the insight methods of the chapter and are written to be universally applicable. Thus, they are direct go-byes to functional aspects of asset management. Consequently, readers can apply them directly to their own roles by merely swapping in their own data.

There is another reason for representative examples over asset-management-specific cases. When a topic insight deliverable entails analytics, some readers will want a full-depth explanation of the underlying principles. Accordingly, the critical-mass explanation of the insight can track a referenced full-depth explanation of their principles.

Bibliography

de Vries, Andries and Meys, Joris. R for Dummies. John Wiley & Son, Inc. 2015.

Field, Andy and Miles, Jeremy. Discovering Statistics Using R. Sage Publications, Inc. 2012.

Matloff, Norman. Art of Programing R. No Starch Press. 2011.