ACTEX

Exam PA Study Manual

Fall 2019 Edition, Component I

Ambrose Lo, FSA, Ph.D., CERA

ACTEX Learning | Learn Today. Lead Tomorrow.

ACTEX Learning

Learn Today. Lead Tomorrow.

Actuarial & Financial Risk Resource Materials

Since 1972

Printed in the United States of America.

ACTEX Study Manual for SOA Exam PA (Predictive Analytics)

Ambrose Lo, PhD, FSA, CERA

December 2019 Edition

Preface (Please Read Carefully!)

> *"The PA modules are so difficult to follow."*
>
> *"The PA modules make things unnecessarily complicated and are riddled with errors."*
>
> *"I feel that the modules don't cover enough material for me to handle the exam well. I have to supplement my learning with external sources."*
>
> *"I hate having to alternate among the PA modules, the R Markdown files, the required textbooks, and online readings."*

These are some of the most common "complaints" an overwhelming number of PA exam candidates have voiced when studying for the exam using the PA e-learning modules as the main learning resource. These comments, as well as the importance of passing this exam to earn the Associateship of the Society of Actuaries (ASA) designation under the post-July 2018 exam curriculum, have motivated us to develop from scratch a completely new study manual specifically for the PA exam that is in high demand, but in short supply.

This study manual fills the needs of numerous candidates by synthesizing, streamlining, and augmenting (if needed) the material in the PA e-learning modules in a coherent and easy-to-follow manner. With this manual, you will have in your possession a reliable learning resource that hosts all of the useful materials in a single place; there is no longer a need to alternate among the e-learning modules, the required textbooks, R markdown files, and additional online readings. Starting from the very basics and adopting a case study approach, we will learn step by step how to implement predictive models in R, understand what the computer output means, and write a report that is to the liking of PA exam graders. No prior knowledge in R and the SRM exam material is assumed.

What is Exam PA Like?

Exam PA (Predictive Analytics) is a five hour and fifteen minute computer-based exam (the extra 15 minutes are meant to allow for breaks, if desired) which was offered for the first time in December 2018 by the Society of Actuaries (SOA). Currently, it is offered two times a year, in June and in December, and each testing window lasts for two days. In December 2019, it will be delivered via computer-based testing (CBT) on December 12-13, 2019. The registration deadline is November 12, 2019.[i] More information about Exam PA can be found at the SOA's official web page:

`https://www.soa.org/education/exam-req/edu-exam-pa-detail/`.

Among the many logistical details there, you should know that:

[i]Since exam registration windows are typically open for six weeks, it is expected that registration for the December 2019 Exam PA will begin on October 1, 2019.

- The combined PA module and PA exam fee ($1,125) is paid at the time of initial registration and purchase of the PA e-learning modules. Even if you don't read the modules, the same $1,125 fee has to be paid!
- You will have 11 months from the date of the purchase of the PA modules to register for one exam without incurring additional cost. You can access the modules for 12 months from the date of purchase; no extensions are allowed. You should keep this timeline in mind when you decide to purchase the modules.
- There is a link to the exam registration page on Slide 22 of Module 9. For your convenience, here is the URL of that page: `https://www.soa.org/Education/Exam-Req/Registration/predictive-analytics.aspx`.

This new exam is the first of its kind in the history of actuarial exams and calls for a completely different learning approach. Instead of using a financial calculator to perform calculations based on formulas acquired from textbooks and/or study manuals and solving 30 to 35 multiple-choice questions within 3 to 3.5 hours as you did in other preliminary exams, in Exam PA you will be provided with a single, extended business problem[ii] which consists of a series of well-defined tasks. To tackle these tasks, you will be provided with the following:

- A computer equipped with Microsoft Word, Microsoft Excel, R (a powerful programming language), and RStudio (a convenient platform to implement R).[iii]
- A dataset in CSV format that accompanies the business problem.
- A project statement of 5 to 6 pages describing a series of tasks you are required to complete, using R if needed. These tasks include but are not limited to:
  - ▷ Exploration and visualization of data
  - ▷ Construction of predictive models
  - ▷ Application of model validation techniques
  - ▷ Creation and selection of features
  - ▷ Interpretation of the results of the predictive models you construct
  - ▷ Preparation of an executive summary written for a general, non-technical audience

  Specific guidance will be provided in each task to help you stay on track.
- An R markdown file with chunks of R code to help you with more involved programming tasks so that your focus is on analysis and communication.
- A report template produced in Word for you to put your written responses to the different tasks.

At the end of the exam, you will submit the following items electronically for grading:

- The completed report template containing all of your written responses.

[ii] The problem is not necessarily actuarial in focus. For example, the December 2018 PA exam deals with mine safety and uses data from the U.S. National Institute for Occupational Safety and Health Mining Program.

[iii] According to the PA exam syllabus, the Prometric computers will have the 2016 versions of Microsoft Word and Excel, version 3.5.0 of R, and version 1.1.463 of RStudio.

- An R Markdown file carrying the R code you use to solve the given tasks. It is expected that your code can be run from beginning to end and there is commentary explaining what your code does.
- Other files (e.g., Excel files) that you believe will support your submission.

You will be graded on both the technical accuracy of your work as well as the clarity of your thought process illustrated in the Word report and the R Markdown file. In all cases, you are expected to justify the decisions you make carefully. As you can see, the shift of focus from working out multiple-choice questions efficiently to crafting computer-aided written responses makes Exam PA a completely different (and hopefully enjoyable!) learning experience compared with all other preliminary exams you have taken.

SRM vs. PA

Exam PA requires Exam SRM (Statistics for Risk Modeling) as a formal prerequisite. Only candidates who have credit for Exam SRM can register[iv] for Exam PA. This credit may be earned either by passing Exam SRM or via transition credit. If you "passed" Exam SRM by transition credit, you should note that knowledge of the Exam SRM learning objectives will be assumed for Exam PA and you may need more time than other candidates to grasp the material for Exam PA. As a rough estimate:

- You need at least *two months* of intensive study to master the material in Exam PA, if you earn the credit for Exam SRM by passing the exam.
- You need at least *three months* of intensive study to master the material in Exam PA, if you earn the credit for Exam SRM by transition credit.

In essence, Exam PA and Exam SRM are about the same subject, but with a different focus. Exam SRM focuses on the underlying theory, covering the uses, motivations, mechanics, pros and cons, do's and don'ts of, and similarities and differences between different predictive analytic methods. Exam PA illustrates and applies the theory you learned in Exam SRM to real data by means of computer-based implementations using R. Some practical considerations are also presented. After taking Exam PA, you will gain a much more thorough understanding of how the predictive analytic methods taught in Exam SRM work.

NOTE

This study manual is self-contained in the sense that it does not assume that you have passed Exam SRM. All of the theory will be explained from first principles in precise and concise terms and you will be able to follow every bit even if you have not taken Exam SRM.

[iv] To purchase the PA modules, you must either have credit for Exam SRM or be registered for a future administration of the exam.

What is Special about This Study Manual?

We fully understand that you have an acutely limited amount of time for study and that Exam PA, as a relatively new exam, is difficult to prepare for. With this in mind, the overriding objective of this new study manual is to help you grasp the material in Exam PA as effectively and efficiently as possible, and pass it with considerable ease. To maximize your learning efficiency, we have adopted a three-component structure in this study manual.

- *Component 1:* The first component of the manual is a crash course in R covering the elements of R programming that are particularly germane to Exam PA. They include the basics of R programming and data visualization using the `ggplot2` package, covered respectively in Chapters 1 and 2 of the manual. At the completion of this component, you will be equipped with the fundamental R programming skills necessary for implementing predictive models and making some high-quality graphs in the rest of this manual.

- *Component 2:* Armed with R basics, you will gain hands-on experience with different predictive models in the second component, also the linchpin of this manual. Drawing upon a wide range of case studies based on real data in different disciplines for illustration purposes, this component delves into the ins and outs, pros and cons, and do's and don'ts of linear models, generalized linear models, decision trees, and principal components and cluster analyses, to be covered in Chapters 3 to 6, respectively. After going through this important component, you will be able to write R code (using only the R packages available at the exam) to fit these statistical models, evaluate their performance, select model features, and interpret the output produced. Mention is also made of what PA graders are looking for in your solution when you analyze these models.

  The expected release date for Component 2 is **mid-October** 2019.

NOTE

All R markdown files (organized by section) and datasets that accompany this manual will be uploaded to

`https://www.actuariallearning.com/Account/Login`

for you to download. A key code will be needed. You should run the R Markdown files as you work through this study manual, making sure your output agrees with what is shown in the manual. This is especially important if you have ordered a print copy of this study manual—run the code to see the colors! ☺

- *Component 3:* The third and last component concludes this manual with an analysis of the four released sample and exam projects [v] (Student Success, December 2018 PA exam, Hospital Readmissions, June 2019 PA exam) and original exam-oriented sample projects that are designed in the same format as the Hospital Readmissions sample project and the June 2019 PA exam. These full-length sample projects give you a holistic review of the entire exam syllabus and additional opportunities to practice. Detailed illustrative solutions are provided

[v] Due to copyright reasons, their project statements will not be reproduced in the manual.

and tips and advice offered on how to write your report in a way that appeals to PA graders (remember, much of this exam is about communication).

The expected release date for Component 3 is the end of October 2019.

This manual throughout is also characterized by the following features that make your learning as effective as possible:

- Each chapter starts by explicitly stating which learning objectives and outcomes of the PA exam syllabus we are going to cover, to assure you that we are on track and hitting the right target.
- Objects in R are shown in `typewriter` font and code chunks with output in gray boxes for easy identification. Functions and commands that are of importance are boxed to aid retention.
- In-text examples and end-of-chapter practice problems provide additional opportunities for you to practice writing R code.

Errata

While we go to great lengths to polish and proofread this manual, some mistakes will inevitably go unnoticed. We would like to apologize in advance for any errors, typographical or otherwise, (hopefully not as many as there are in the PA e-learning modules!) and would greatly appreciate it if you could bring them to the author's attention via email (`ambrose-lo@uiowa.edu`) so that they can be fixed in a future edition of the manual. Compliments and criticisms are also welcome. The author will try his best to respond to any inquiries as soon as possible and an ongoing errata list will be maintained online at `https://sites.google.com/site/ambroseloyp/publications/PA`. Students who report errors will be entered into a quarterly drawing for a $100 in-store credit.

Ambrose Lo
September 2019

About the Author

Professor Ambrose Lo, Ph.D., FSA, CERA, is currently Associate Professor of Actuarial Science with tenure at the Department of Statistics and Actuarial Science, The University of Iowa. He earned his B.S. in Actuarial Science (first class honors) and Ph.D. in Actuarial Science from The University of Hong Kong in 2010 and 2014 respectively. He joined The University of Iowa in August 2014 as Assistant Professor of Actuarial Science. His research interests lie in dependence structures, quantitative risk management as well as optimal (re)insurance. His research papers have been published in top-tier actuarial journals, such as *ASTIN Bulletin: The Journal of the International Actuarial Association*, *Insurance: Mathematics and Economics*, and *Scandinavian Actuarial Journal*.

Besides dedicating himself to actuarial research, Ambrose attaches equal importance to teaching, through which he nurtures the next generation of actuaries and serves the actuarial profession. He has taught courses on financial derivatives, mathematical finance, life contingencies, credibility theory, advanced probability theory, and regression and time series analysis. His emphasis in teaching is always placed on the development of a thorough understanding of the subject matter complemented by concrete problem-solving skills. Besides coauthoring the *ACTEX Study Manual for SOA Exam SRM* (Fall 2019 Edition), he is also the sole author of the *ACTEX Study Manual for CAS Exam MAS-I* (Fall 2019 Edition), *ACTEX Study Manual for SOA Exam PA* (December 2019 Edition), and the textbook *Derivative Pricing: A Problem-Based Primer* (2018) published by Chapman & Hall/CRC Press.

Part I

A Crash Course in R

Chapter 1

Basics of R Programming

DECEMBER 2019 EXAM PA LEARNING OBJECTIVES

1. Topic: Predictive Analytics Problems and Tools

Learning Objectives

The Candidate will be able to articulate the types of problems that can be addressed by predictive modeling and be able to work with RStudio to implement basic R packages and commands.

Learning Outcomes

b) Write and execute basic commands in R using RStudio.

(Note: We will defer a) to Part II of this study manual.)

4. Topic: Data Types and Exploration

Learning Outcomes

c) Understand basic methods of handling missing data.

Chapter overview: In Exam PA, you are expected to use R to manipulate data, create graphs for visualizing relationships between variables, and implement predictive models. Based on the computer output, you will develop promising predictive models for making predictions and interpret your findings. A basic understanding of R programming is therefore a prerequisite to doing well in the exam. In this chapter, we will familiarize ourselves with some basic R commands and functions that will be heavily used in the remainder of this study manual. We begin in Section 1.1, where we set up our R toolkit and learn about the three most commonly used types of data. Section 1.2 expands the discussion in Section 1.1 and presents four important data structures that R use to store information. The subtle differences between these data structures are highlighted. Section 1.3 applies what we know about data structures to solve a number of typical data management problems. Section 1.4 concludes this chapter with a discussion of `for` loops, which are important tools for doing simulation. After completing this chapter, you will become comfortable with the R environment and be ready to use R to construct predictive models in subsequent chapters.

It should be pointed out that Exam PA is not an exam on R programming *per se*; R is used mostly as a means to perform data exploration and run predictive models. As stated in the preface, sample code chunks are provided in the exam to help you accomplish more involved programming

tasks so that your focus is not on writing long code, but on analysis and interpretation[i] (in fact, I doubt if you will be asked to write a `for` loop from scratch). For this reason, this preparatory chapter, written specifically for students taking Exam PA, is necessarily limited in scope and focuses on a very small set (say, 5%) of tools in R that allow you to accomplish almost all (say, 99%) of the programming tasks you will likely encounter in the exam. Still, a fundamental understanding of R programming is useful for making sense of what the given code chunks do and figuring out how to modify the code (e.g., change the parameters, if needed) to suit our purpose.

1.1 Getting Started

Developed in the early 1990s, R is an extremely versatile open-source programming language for statistical computing, graphics, and, in recent years, document preparation. For Exam PA, the most conspicuous merits of R are three-fold:

- Unlike many commercial statistical software platforms, R is free to use. This point is important as it ensures that every PA candidate can access and practice using the software platform required for the exam for free. (Well, you still have to pay $1125 just to access the PA modules!)
- R has superior graphics capabilities. It allows you to create elegant, informative, and customizable graphs which would be difficult, if not impossible, to create in other programming languages.
- R outperforms many other software platforms due to the wide collection of add-on packages that can significantly enhance the functionality of R. More impressively, these packages can be downloaded from online repositories for free.

In this introductory section, we will walk through the installation of R and RStudio (a good companion to R), the basic features of RStudio, and the common data types in R.

1.1.1 Basic Infrastructure

First things first, let's install R and RStudio on your computer (if you have not already done so).

Installing R. R can be freely downloaded and installed from *The Comprehensive R Archive Network* (CRAN) at `https://cran.rstudio.com`. The top of the web page shows three links for downloading R:

- Download R for Linux
- Download R for (Mac) OS X
- Download R for Windows

Choose the link that works for the operating system of your computer. To install R on Windows, for example, click "Download R for Windows", then the "base" link, and finally "Download R 3.6.1 for Windows" (3.6.1 may be replaced by a more updated version of R). An installer program will be downloaded. Running the program and stepping through the installation wizard (the default

[i] See Slide 3 of PA Module 1.

- Put spaces around all binary operators such as `=`, `+`, `-`, `<-`.

  Good: `x < 1`

  Bad: `x<1`

- Do not put a space before a comma, but always put one after a comma.

  Good: `df[, 2]`

  Bad: `df[ , 2]` and `df[ ,2]`

- Do not place spaces around code in parentheses or square brackets, with the exception that a space is always placed after a comma.

  Good: `if (x <= 1)` and `df[1, ]`

  Bad: `if ( x <= 1 )` and `df[1,]`

- Short comments can be put after code preceded by two spaces, `#`, followed by one space, e.g.,

```
a <- 1 + 1  # to save the result of 1 + 1 in an object named a
```

- Function definitions should first list arguments without default values, followed by those with default values.

Note that graders for Exam PA will not be evaluating how closely you follow these programming style guidelines (ironically, many of the Rmd files provided by the SOA defy these guidelines!). However, writing code with these guidelines in mind will make your code more readable and help graders understand your work better.

1.2 Data Structures

There are a wealth of structures in R that we can use to hold data. These *data structures* (not to be confused with "data types" in the preceding section) are distinguished by the type of data they can hold, their structural flexibility, and the notation used to extract individual elements. In this section, we will cover a number of data structures that are particularly important in Exam PA. Here is an overview:

Name	Dimension	Individual elements/components of same data type?
Vectors	One-dimensional	Yes
Matrices	Two-dimensional	Yes
Data frames	Two-dimensional	Potentially different
Lists	One-dimensional	Potentially different

1.2.1 Vectors

Vector operations. A distinguishing property of R is that it is a *vectorized* programming language, meaning that operations are applied to a vector element-wise without the need for looping through the entire vector. In R, a *vector* is defined as an ordered (i.e., the order matters) collection of elements, all of which are of the same type (e.g., numeric, character, logical). The easiest way to create a vector is through the `c()` function (standing for "concatenate" or "combine"), which strings the comma-separated arguments inside the parentheses together to form a vector. For the purpose of illustration, let's build four vectors in CHUNK 1.

```
# CHUNK 1
a <- c(1, 2, 3, 4, 5)
b <- c(5, 4, 3, 2, 1)
c <- c("A", "B", "C")
d <- c(TRUE, FALSE, FALSE, TRUE, TRUE)
```

Here `a` and `b` are numeric vectors of the same length (which is 5), `c` is a character vector, and `d` is a logical vector. A shorthand for creating `a` is `1:5`, where the colon operator `:` generates a sequence of consecutive integers in either direction. Similarly, `b` can be created by the command `5:1`. More generally, the `seq(from, to, by)` function returns a sequence of numbers (not necessarily integers) starting from the number indicated by the `from` argument and ending with the number indicated by the `end` argument, with the `by` argument specifying the increment of the sequence. For instance:

```
# CHUNK 1 (Cont.)
seq(0, 1, 0.25)

## [1] 0.00 0.25 0.50 0.75 1.00
```

The length of a vector can be found by the `length()` function. Note that unlike in linear algebra, there is no such thing as a row vector or a column vector in R.

```
# CHUNK 1 (Cont.)
length(a)

## [1] 5

length(c)

## [1] 3
```

To add, multiply, or raise each element of a numeric vector by a common factor, it suffices to apply the operation directly to the vector itself. No loops are needed. CHUNK 2 contains a series of examples.

```
# CHUNK 2
a + 2   # add 2 to each element of a

## [1] 3 4 5 6 7
```

Thus $\hat{\beta}_0 = 5.2506$, $\hat{\beta}_1 = 0.8137$, and $\hat{\beta}_2 = 1.5166$.

□

Remark. In Chapter 3, we will learn how to use the `lm()` function to fit a linear model and the `coef()` function to extract the least squares estimates.

```
m <- lm(Y ~ X1 + X2)
coef(m)

## (Intercept)          X1          X2
##   5.2505543   0.8137472   1.5166297
```

1.2.3 Data Frames

Basic properties. A substantially more general version of matrices, *data frames* are central to the use of many R graphics and modeling functions and are justifiably the most important data structure in Exam PA. From a technical point of view, they can be thought of as an ordered collection of vectors combined column by column. These vectors must be of the same length, but, unlike matrices, can be of *different types*, i.e., some columns can be numeric while some can be character or logical. This allows for a wide variety of data types all stored within the same data structure for analysis. For example, in CHUNK 14 we use the `data.frame()` function to create a data frame with three columns named `x`, `y`, and `z`, the first two of which are numeric vectors and the last one is a character vector.

```
# CHUNK 14
x <- 6:10
y <- 11:15
z <- c("one", "two", "three", "four", "five")
df <- data.frame(x, y, z)  # construct a data frame with x, y, and z as columns
df

##    x  y     z
## 1  6 11   one
## 2  7 12   two
## 3  8 13 three
## 4  9 14  four
## 5 10 15  five
```

From a statistical point of view, each column of a data frame can be interpreted as observed values of the same *variable* (that's why each column must have only one mode of data) and each row represents the set of measurements taken or characteristics for the same *observation* corresponding to these variables, much like an Excel spreadsheet. The variables across different columns are allowed to be of different types and capture different kinds of information; see the pictorial illustration in Figure 1.2.2. For instance, if we use a data frame to construct a dataset for a book of life insurance

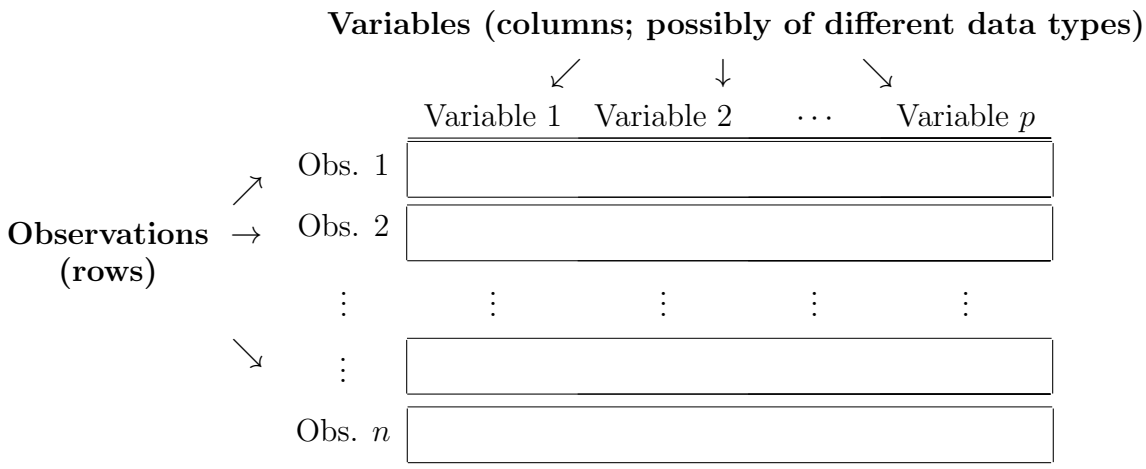

Figure 1.2.2: A pictorial illustration of the structure of a generic data frame.

policies, then its columns (variables) may represent information such as the names of policyholders, the amount insured, and the age of the policyholders, and each row (observation) may carry the information for a particular policy. This observation-by-variable arrangement parallels the usual rectangular way we think of a dataset and makes a data frame almost synonymous with a dataset. In the rest of this manual, the terms "dataset" and "data frame" will be used interchangeably.

The `data.frame()` function is the most primitive way to construct a data frame and is used mostly when the data frame in question is relatively small or we want to create one on the fly (some statistical functions such as `predict()` require a data frame as an argument). In Exam PA, almost always you will read in an external CSV file to create a data frame with thousands of observations. For such a large data frame, it is often useful to use the `head(<data frame>, n)` function to print out only the first few rows to get a sense of the data. By default, only the first six rows will be printed, but this can be changed by the `n` argument. The `tail(<data frame>, n)` function is dual to `head()` and prints out the last few rows. Let's try out the `head()` and `tail()` functions in CHUNK 15.

```
# CHUNK 15
head(df, n = 3)  # print out the first three rows

##   x  y     z
## 1 6 11   one
## 2 7 12   two
## 3 8 13 three

tail(df, n = 2)  # print out the last two rows

##    x  y    z
## 4  9 14 four
## 5 10 15 five
```

Attributes. A data frame has a number of attributes that can be extracted by functions.

```
if (<condition>) {
  <statement_1>
  <statement_2>
  ...
} else {
  <statement_1>
  <statement_2>
  ...
}
```

Here `<condition>` is a logical variable immediately following the `if` command. If `<condition>` equals `TRUE`, then all statements inside the following curly braces `{}` are executed (you may omit curly braces when there is only a single statement to execute). If `<condition>` fails, then all statements inside the curly braces that follow the `else` command are executed. In the first call of the `sumDiff.mod()` function, a list with two components, `sum` and `diff`, is returned. In the second call, we override the default and set `list = FALSE` so that the function outputs a data frame with `sum` and `diff` as its variables (columns).

To see an example of a function with numeric default values, let's revisit the `head()` function that we introduced when learning data frames. Consulting its help page (type `?head`), we see that this function returns by default the first six rows of a vector, matrix, or data frame, but the number of rows to display can be changed by the `n` argument.

```
# CHUNK 26
head(sumDiff.mod(1:8, 8:1, list = FALSE))         # show first six rows

##   sum diff
## 1   9   -7
## 2   9   -5
## 3   9   -3
## 4   9   -1
## 5   9    1
## 6   9    3

head(sumDiff.mod(1:8, 8:1, list = FALSE), n = 3)  # show first three rows

##   sum diff
## 1   9   -7
## 2   9   -5
## 3   9   -3
```

1.3 Basic Data Management

Now that we have a basic understanding of the data-holding structures in R, we now look at some elementary data management problems commonly encountered in Exam PA. Even though the dataset provided in the exam (in the form of an external CSV file) can be imported easily into R in the usual observation-by-variable (rectangular) format, there are often data management issues

that should be resolved before a predictive model can be applied, to avoid producing misleading or meaningless results. To illustrate these data management issues, we will focus on a small-scale dataset involving eight hypothetical actuaries in a certain insurance company set up in CHUNK 1 of this section.[x]

```
# CHUNK 1
x <- 1:8
name <- c("Embryo Luo", "", "Peter Smith", NA,
          "Angela Peterson", "Emily Johnston", "Barbara Scott", "Benjamin Eng")
gender <- c("M", "F", "M", "O", "F", "F", "F", "M")
age <- c(-1, 25, 22, 50, 30, 42, 29, 36)
exams <- c(10, 3, 0, 4, 6, 7, 5, 9)
Q1 <- c(10, NA, 4, 7, 8, 9, 8, 7)
Q2 <- c(9, 9, 5, 7, 8, 10, 9, 8)
Q3 <- c(9, 7, 5, 8, 10, 10, 7, 8)
actuary <- data.frame(x, name, gender, age, exams, Q1, Q2, Q3,
                      stringsAsFactors = FALSE)
actuary

##   x            name gender age exams Q1 Q2 Q3
## 1 1      Embryo Luo      M  -1    10 10  9  9
## 2 2                      F  25     3 NA  9  7
## 3 3     Peter Smith      M  22     0  4  5  5
## 4 4            <NA>      O  50     4  7  7  8
## 5 5 Angela Peterson      F  30     6  8  8 10
## 6 6  Emily Johnston      F  42     7  9 10 10
## 7 7   Barbara Scott      F  29     5  8  9  7
## 8 8    Benjamin Eng      M  36     9  7  8  8
```

Each row of the dataset has information about an actuary's name, gender, age, number of exams passed, and three evaluation ratings (on a scale from 1 to 10) assigned by their supervisors according to three performance measures. Although the dataset you will encounter in Exam PA is likely to have thousands of observations, for demonstration purposes we have deliberately chosen a toy dataset like `actuary` so that you can easily appreciate the changes you have made to the dataset. Nevertheless, the data management techniques we are going to illustrate are so efficient that they will apply equally well to large datasets.

Using this toy dataset, we will demonstrate how the following data management tasks can be accomplished in R:

1. *(Missing/abnormal values)* Some entries in the dataset are missing or do not make sense. How to deal with these missing or abnormal entries?

2. *(Deletion)* How to delete the variable `x`, which is simply a column of row numbers and not useful for most predictive purposes?

[x]By default, the `data.frame()` function converts character vectors to factors. The self-explanatory option `stringsAsFactors = FALSE` prevents this from happening. In any case, this is not a very important point for Exam PA.

3. *(Creation + sorting)* How to create a new numeric variable that averages the three evaluation ratings and provides an overall performance measure for each actuary? Based on the overall performance measure, which actuary has the best performance?

4. *(Identification)* How to identify Associates (those who have passed 7 to 9 exams) and Fellows (those who have passed 10 exams)? More generally, how to create a new categorical variable that represents whether an actuary is an Associate, a Fellow, or a pre-ASA student?

5. *(Merge)* How to add new observations and variables to an existing dataset?

6. *(Compound)* How to create a compound variable that represents both the gender and title of an actuary?

A PA project typically involves one or more of these logistical tasks. We will examine the six tasks in turn.

Task 1: Missing values. In data analysis, a *missing value* is said to arise when no value is available for a particular entry in a dataset for whatever reason. In R, missing values are denoted by the special symbol `NA`, which stands for "not available." We can see that the `actuary` dataset has two missing values:

- The second observation has `Q1` indicated as a missing value.
- The name of the fourth observation is missing.

Note that although the name of the second observation is left blank (indicated by the empty character string `""`), it is *not* treated technically in R as a missing value.

Missing values are problematic in data analysis because many functions in R will return a missing value if some of its arguments contain missing values (although most functions have the `na.rm` argument to handle missing values). How should we deal with missing values in the exam? Should we remove observations and variables that contain missing values altogether, or should we retain these missing values and replace them by some "educated" guesses? Slide 37 of PA Module 4 provides an authoritative answer:[xi]

EXAM NOTE

"[F]or the Predictive Analytics exam, it will always be sufficient for your analysis to deal with missing observations by either eliminating the records or variables that contain them. However, it would be appropriate to comment on more sophisticated approaches if they are deemed appropriate."

We will therefore focus on how to simply remove *observations* that contain missing values (we prefer to remove observations rather than columns because datasets in Exam PA tend to have far more observations than columns). Here are two common methods, both of which rely on logical subsetting:

[xi] More sophisticated methods for handling missing values include replacing these values with mean, median, or mode, depending on the type of the variable in question, and using a model based on the combinations of other variables to fill in the missing values (a process called *imputation*). See Slides 42 to 48 of PA Module 4 if you are interested.

1. *Using the* `is.na()` *function:* The `is.na()` function returns an object of the same size as its argument. The elements of this object are equal to `TRUE` if the corresponding elements of the original object are a missing value and equal to `FALSE` otherwise.

   Run CHUNK 2 to remove the second observation, which has a missing value for `Q1`.

```
# CHUNK 2
is.na(actuary$Q1)

## [1] FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE

actuary.1 <- actuary[!is.na(actuary$Q1), ]
actuary.1

##   x            name gender age exams Q1 Q2 Q3
## 1 1      Embryo Luo      M  -1    10 10  9  9
## 3 3     Peter Smith      M  22     0  4  5  5
## 4 4            <NA>      0  50     4  7  7  8
## 5 5 Angela Peterson      F  30     6  8  8 10
## 6 6  Emily Johnston      F  42     7  9 10 10
## 7 7   Barbara Scott      F  29     5  8  9  7
## 8 8    Benjamin Eng      M  36     9  7  8  8
```

   The code in CHUNK 2 is a perfect illustration of logical subsetting—using appropriate logical tests to identify elements that satisfy certain conditions. As the vector `is.na(actuary$Q1)` returns

   `c(FALSE, TRUE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE)`,

   with `TRUE` corresponding to the second row, its negation `!is.na(actuary$Q1)` is

   `c(TRUE, FALSE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE)`.

   When this logical vector serves as the row index vector for `actuary` and the column index is left blank, all rows except the second row are included. The resulting data frame, called `actuary.1`, has no more missing values for `Q1`.

   You may ask:

   > Why not use a simple command like `actuary.1 <- actuary[-2, ]` to get rid of the second observation?

   While this command works perfectly for the `actuary` dataset here because it is easy to spot which rows or columns contain missing values, it may not work well for a large dataset for which visually searching for missing values is practically impossible. The power of logical subsetting is that it allows us to extract and modify, via a user-specified logical test, the desired observations efficiently *without having to know in advance where these observations are located in the dataset*. As long as we know how to characterize the desired observations by logical tests, R will do the search for us automatically and quickly. In the remainder of this section, we will see repeated applications of logical subsetting.

1.4 `for` Loops

We end this introductory chapter on R programming with a brief discussion on `for` loops, which allow us to execute a series of statements repetitively with a minimal amount of code for a pre-specified number of times. Although `for` loops are not heavily used in Exam PA (so you may not want to spend too much time learning these devices), they play an important role in simulation studies and are good programming tools to know in general.

Basic structure. The general syntax of a `for` loop is as follows:

```
for (<var> in <vec>) {
  <statement_1>
  <statement_2>
  ...
}
```

where

`<var>` is the index variable to loop through,

`<vec>` is a numeric or character vector that carries the possible values of `var` over different iterations of the loop, and

`<statement_1>`, `<statement_2>`, `...` are statements that will be run for each value of `var` in the order listed in `vec`.

As a simple example, let's use a `for` loop to construct a function evaluating the partial sum

$$\sum_{i=1}^{n} i \qquad \left(= \frac{n(n+1)}{2}\right)$$

for every positive integer n. Run CHUNK 1 to create such a function named `psum` and calculate the partial sum for $n = 1, 3$, and 10.

```
# CHUNK 1
psum <- function(n) {
  s <- 0
  for (i in 1:n) {
    s <- s + i
  }
  return(s)  # don't miss the return value
}

psum(1)

## [1] 1

psum(3)

## [1] 6

psum(10)

## [1] 55
```

The `psum()` function computes the partial sums recursively as follows:

$$\begin{aligned}
s_0 &= 0 && \text{(starting value)}\\
s_1 &= s_0 + 1 = 0 + 1 = 1, && (i = 1)\\
s_2 &= s_1 + 2 = 1 + 2 = 3, && (i = 2)\\
s_3 &= s_2 + 3 = 3 + 3 = 6, && (i = 3)\\
&\vdots && \vdots\\
s_{10} &= s_9 + 10 = 45 + 10 = 55. && (i = 10)
\end{aligned}$$

Note that a `for` loop itself does not return any output. It is thus important to include the final value of the partial sum with the command `return(s)`, without which the `psum()` function will not output anything.

For this particular task, it turns out that simpler code based on vectorization can be used, as CHUNK 2 shows.

```
# CHUNK 2
psum.vec <- function(n) {
  x <- 1:n
  return(sum(x))
}

psum.vec(1)

## [1] 1

psum.vec(3)

## [1] 6

psum.vec(10)

## [1] 55
```

[HARDER!] Typical application of `for` loops: Simulation studies. We now see a statistical application for which the use of a `for` loop seems indispensable: To undertake simulation studies. As we have learned from Exam MFE/IFM, simulation entails generating repeated random samples and using these samples to estimate probabilistic quantities of interest. Here we will apply simulation to illustrate a fundamental concept we may have overlooked in our VEE Mathematical Statistics course: The genuine meaning of a *confidence interval* (CI).

Consider a random sample $\{X_1, \ldots, X_n\}$ from a normal distribution with unknown mean μ and known standard deviation σ. We know from our prior studies that a 95% two-sided CI for the population mean μ is

$$\bar{X} \pm 1.96 \times \frac{\sigma}{\sqrt{n}} := \left[\bar{X} - 1.96 \times \frac{\sigma}{\sqrt{n}}, \bar{X} + 1.96 \times \frac{\sigma}{\sqrt{n}}\right],$$

where $\bar{X} = \sum_{i=1}^{n} X_i/n$ is the sample mean and 1.96 is the 97.5% percentile of the standard normal distribution. After observing the realized values of $X_1, \ldots, X_n$, denoted by $x_1, \ldots, x_n$, we can

1.5 End-of-Chapter Practice Problems

NOTE

Tasks in Exam PA may not be phrased in exactly the same format as the following problems. However, the programming skills and concepts covered in these problems reinforce what we have learned in the main text of this chapter and may help you complete a task more effectively and efficiently in the exam.

Problem 1.5.1. (Splitting a dataset into a training set and a test set) You are given:

- `dat` is a given dataset (data frame) which you are trying to split into two sets, one set (the training set) for fitting the predictive model you are interested in and one set (the test set) for evaluating the predictive performance of the fitted model.
- `partition` is an integer vector whose elements are selected from `1:nrow(dat)` without replacement (i.e., all of the elements are distinct). You would like to use `partition` as the row index vector of `dat` to form the training set and put the rest of the observations in the test set.

Taking `dat` and `partition` as given inputs, write R code to construct the training set and test set, named as `train` and `test`, respectively.

(Note: All of the released PA sample and exam projects involve the split illustrated in this problem.)

Problem 1.5.2. (Based on December 2018 Exam PA: Writing a function to calculate loglikelihood) For $i = 1, \ldots, n$, let Y_i be a Poisson random variable with mean μ_i and fitted mean $\hat{\mu}_i$ based on a given predictive model (e.g., Poisson regression model, to be studied in Chapter 4) and the method of maximum likelihood estimation.

(a) Show that the maximized loglikelihood function is

$$l(\hat{\mu}_1, \ldots, \hat{\mu}_n) = \sum_{i=1}^{n} y_i \ln \hat{\mu}_i - \sum_{i=1}^{n} \hat{\mu}_i + \text{constants not involving } \hat{\mu}_i\text{'s},$$

provided that the $\ln \hat{\mu}_i$'s are well-defined.

(b) Write an R function called `LL` to compute the maximized loglikelihood function that takes the following as arguments:

- `observed` is a vector of observed values of the Y_i's
- `predicted` is a vector of fitted values of the Y_i's

It is not given *a priori* that the fitted values are non-negative. If $\hat{\mu}_i$ is a non-positive value and $\ln \hat{\mu}_i$ is encountered, your function should take care of this by changing $\hat{\mu}_i$ to a very small number, say 0.000001 (note that the log of zero is not defined).

(Hint: You will find the `ifelse()` construct useful.)

Problem 1.5.3. (Based on the new performance measure, how to create a new categorical variable that classifies each actuary as "Excellent", "Good", "Unsatisfactory"?) Consider the `actuary.n` dataset in Section 1.3. If needed, run the following code to get it set up again:

```
x <- 1:8
name <- c("Embryo Luo", "", "Peter Smith", NA,
          "Angela Peterson", "Emily Johnston", "Barbara Scott", "Benjamin Eng")
gender <- c("M", "F", "M", "O", "F", "F", "F", "M")
age <- c(-1, 25, 22, 50, 30, 42, 29, 36)
exams <- c(10, 3, 0, 4, 6, 7, 5, 9)
Q1 <- c(10, NA, 4, 7, 8, 9, 8, 7)
Q2 <- c(9, 9, 5, 7, 8, 10, 9, 8)
Q3 <- c(9, 7, 5, 8, 10, 10, 7, 8)
actuary <- data.frame(x, name, gender, age, exams, Q1, Q2, Q3,
                      stringsAsFactors = FALSE)
actuary.n <- na.omit(actuary)
actuary.n$S <- (actuary.n$Q1 + actuary.n$Q2 + actuary.n$Q3) / 3
```

Suppose that as the boss of the six actuaries, you would like to classify them according to the following scheme:

Criterion	Classification
`S <= 7`	Unsatisfactory
`7 < S < 9`	Good
`S >= 9`	Excellent

Write R commands to perform these classifications and save the classifications as a new variable called `classify`.

Problem 1.5.4. (Practice with the `paste()` or `paste0()` functions) Write R code in terms of the `paste()` or `paste0()` functions to produce the following vector of character strings.

```
##  [1] "1st"  "2nd"  "3rd"  "4th"  "5th"  "6th"  "7th"  "8th"  "9th"  "10th"
```

You should avoid typing the ten character strings one by one.

Problem 1.5.5. (Constructing the Fibonacci sequence) The sequence of Fibonacci numbers $F_1, F_2, \ldots$ is defined by $F_1 = F_2 = 1$ and

$$F_n = F_{n-1} + F_{n-2} \quad \text{for } n \geq 3.$$

(a) Write an R function called `Fib` to calculate F_n for any positive integer n.

(Note: There are different ways to define this function. Some definitions are more efficient, and some less so.)

(b) Use the function in part (a) to calculate F_2, F_3, F_5, F_{10}, and F_{50}.

Chapter 2

Data Visualization with the ggplot2 Package

DECEMBER 2019 EXAM PA LEARNING OBJECTIVES

3. Topic: Data Visualization

Learning Objectives

The Candidate will be able to create effective graphs in RStudio.

Learning Outcomes

a) Understand the key principles of constructing graphs.

b) Create a variety of graphs using the ggplot2 package.

4. Topic: Data Types and Exploration

e) Apply univariate and bivariate data exploration techniques.

Chapter overview: An integral part of any predictive analytic exercise is the use of graphical displays to visualize the relationships between variables of interest. In this regard, one of the key strengths of `R` as a programming language is that it offers versatile graphing capabilities, both in the base installation and with add-on packages. A wide variety of high-quality graphs can be produced with a minimal amount of code. In Exam PA, you are asked to take advantage of `R`'s graphing capabilities and supplement your written report with the aid of different types of graphical displays. Instead of using `R`'s base graphical platform, Exam PA specifically requires that you produce graphs using the `ggplot2` package,[i] which is probably new to you even if you have used R before. Compared to `R`'s base graphics system, `ggplot2` calls for a vastly different syntax based on the so-called "grammar of graphics" (in fact, "gg" stands for "grammar of graphics") and lends its way to producing sophisticated graphs that would be cumbersome to create using base R graphics.

Synthesizing the material in the first four chapters of the book *Data Visualization: A Practical Introduction*, this chapter presents some of the most important graphical functions in the `ggplot2` package for Exam PA. These functions can be used to construct a wide variety of graphs such as

[i]The `ggplot2` package is developed by Hadley Wickham, who is also a core developer of RStudio. Earlier the package was called `ggplot`, but later substantial changes were made, so the name of the package was upgraded to `ggplot2`.

scatterplots, histograms, and bar charts. In Section 2.1, we will learn the basic structure of a ggplot, make some simple but informative plots, and learn how to tweak the appearance of a ggplot, all in the context of a small-scale dataset of automobiles. Section 2.2 draws upon the data visualization techniques covered in Section 2.1 to perform data exploration. Using a real insurance dataset with more than 20,000 observations, we introduce additional geoms and illustrate the use of ggplots to uncover patterns and relationships and generate hypotheses which can be answered in a predictive model.

2.1 Fundamentals of a ggplot

2.1.1 Basic Features

Let's begin by installing and loading the ggplot2 package.

```
# CHUNK 1
#install.packages("ggplot2") # uncomment this line the first time you use ggplot2
library(ggplot2)
```

Skeleton. In ggplot2, a plot consists of two parts: The core ggplot() function (not ggplot2()!) and a chain of additional functions that define the type of plot pasted using the plus (+) sign.

1. ggplot() *function:* The ggplot() function initializes the plot, defines the source of data using the data argument (almost always a data frame in Exam PA; recall Subsection 1.2.3), and, most importantly, specifies what variables in the data are "mapped" to visual elements in the plot by the mapping argument. Mappings in a ggplot are specified using the aes() function, with aes standing for "aesthetics." They determine the role different variables play in the plot. The variables may, for instance, correspond to visual elements such as the x- or y-variables, color, size, and shape, specified by the x, y, color, size, and shape aesthetics, respectively.

2. *Geom functions:* Subsequent to the ggplot() function, we put in *geometric objects*, or *geoms* for short, which include points, lines, bars, histograms, box plots, and many other possibilities, by means of one or more *geom functions*. Placed layer by layer, these geoms determine what kind of plot is to be drawn and modify its visual characteristics, taking the data and aesthetic mappings specified in the ggplot() function as inputs.

Here is the generic structure of a ggplot:

```
ggplot(data = <DATA>, mapping = aes(<AESTHETIC_1> = <VARIABLE_1>,
                                    <AESTHETIC_2> = <VARIABLE_2>,
                                    ...) +
  geom_<TYPE>(<...>) +
  geom_<TYPE>(<...>) +
  <OTHER_FUNCTIONS>  +
  ...
```

Saving a ggplot. After the expenditure of so much effort, naturally you will want to save your ggplots so that you can incorporate them into your PA exam report. While `ggplot2` offers a way to save a graph for import into Microsoft Word (read Section 3.7 of *Data Visualization: A Practical Introduction* if you are interested), a much simpler solution that is allowed by the SOA is to just copy and paste. Simply right-click the ggplot created in R Markdown, select "`Copy Image`", and paste it on your report in Word. Simple and effective!

2.2 Data Exploration

Now that we have learned how to make some simple graphs by `ggplot2`, in this section we will apply our data visualization techniques to perform *exploratory data analysis* (EDA), which is often the warm-up part in a PA exam project. To give you some idea of how important EDA is, let's look at the very first task in the July 2019 PA exam:

> 1. *(5 points)* Explore the relationship of each variable to (the target variable)
>
> Use graphical displays and summary statistics to form preliminary conclusions regarding which variables are likely to have significant predictive power.

The first task in the Hospital Readmissions sample project is in a similar spirit:

> 1. *(6 points)* Perform univariate exploration of the four non-factor variables
>
> Use graphical displays and summary statistics to determine if any of these variables should be transformed and, if so, what transformation should be made. Do your recommended transformations, if any, and delete the original variables.

Data exploration is indispensable in any modeling exercise. Among its many purposes, it allows us to perform "sense checks" on the data (do the observations make sense?), identify potential data errors that may derail our analysis, appreciate the basic relationships between variables, determine the need for any transformations of variables, and, most importantly, decide on an appropriate predictive model that is likely to make practical sense. To accomplish EDA, we will draw upon a variety of numerical summary statistics and graphical tools that provide insights into the central tendency, dispersion, and other aspects of the variables of interest. After completing this section, you should be able to use R and `ggplot2` to generate useful summary statistics and plots for different types of variables that commonly arise in Exam PA.

2.2.1 Univariate Data Exploration

Let's begin with *univariate* data exploration—exploration that sheds light on the distribution of only one variable at a time. Such exploration usually takes two forms:

- *Statistical summaries:* To summarize the data by numerical statistics that capture different distributional properties of a variable. These summary statistics, such as the mean, variance, and frequencies, provide a broad overview of a variable's distribution and make it easy for us to compare different variables.

- *Graphical summaries:* To visualize the distribution of a variable by graphical displays such as histograms, box plots, and bar charts. Graphical summaries allow us to get a quick glimpse at the overall distribution of a variable and are often more informative than looking at a table of numerical statistics. Furthermore, they may reveal outliers that cannot be easily detected by numerical statistics alone.

Very often, summary statistics and graphical representations are used in combination with one another, but the specific statistical and graphical tools will depend on whether the variables you are analyzing are numeric or categorical.

Numeric variables.

- *Statistical summaries:* The central tendency of a numeric variable, whether it be continuous or discrete, is often quantified by its mean or median, which can be readily produced in R by applying the `summary()` function to the variable of interest. Common measures of dispersion or spread include variance, standard deviation, and interquartile range (defined as the difference between the 75% quantile and the 25% quantile of a variable).

- *Graphical summaries—histograms and box plots:* To visualize the distribution of numeric variables, histograms and box plots are good graphical aids. *Histograms* divide the observations into several equally spaced bins (or buckets) and provide a visual summary of the count or relative frequency in each bin. Looking at a histogram, we can learn about the overall shape of the distribution of a numeric variable.

  Box-and-whiskers plots, or simply *box plots*, visualize the distribution of a numeric variable by placing its 25% quantile, the median, the 75% quantile in a "box," with the rest of the data points constituting the whiskers. The amount of spacing between different parts of a box plot reflects the degree of dispersion and skewness of the variable's distribution. "Outliers," defined as data points that are above or below 1.5 times the interquartile range from either edge of the box, are shown as large dotted points. Although box plots do not directly show the actual shape of the variable's distribution, they offer a useful graphical summary of the key numeric statistics and allow for a visual comparison of the distributions of different numeric variables (in particular, the relative magnitude of the five summary statistics) or the distribution of the same numeric variable across different levels of another categorical variable.

Case study: Personal injury insurance claims. For the purpose of illustration, in this section we will look at the personal injury insurance dataset named `persinj.csv`. Downloaded online from

> `http://www.businessandeconomics.mq.edu.au/our_departments/Applied_Finance_and_Actuarial_Studies/acst_docs/glms_for_insurance_data/data/persinj.xls`[iv]

and pre-processed to suit our purpose, this dataset contains the information of $n = 22,036$ settled personal injury insurance claims. These claims were reported during the period from July 1989 to the end of 1999, with claims settled with zero payment not included. The variables in the dataset are described in Table 2.2. In Section 4.4, we will build a model to predict the size of personal injury insurance claims using other variables in the dataset. For now, we will perform data exploration of

[iv]This is part of the companion web page of the textbook *Generalized Linear Models for Insurance Data* (2008), by de Jong and Heller.

2.3 End-of-Chapter Practice Problems

Problem 2.3.1. (Small differences in code, large differences in output!) Consider the personal injury insurance dataset in Section 2.2 again and the following chunks of R commands (which look similar!):

```
persinj <- read.csv("persinj.csv")
persinj$inj <- as.factor(persinj$inj)
persinj$legrep <- as.factor(persinj$legrep)
library(ggplot2)

# CHUNK 1
ggplot(persinj, aes(x = inj, color = legrep)) + geom_bar()

# CHUNK 2
ggplot(persinj, aes(x = inj, fill = legrep)) + geom_bar()

# CHUNK 3
ggplot(persinj, aes(x = inj)) + geom_bar(fill = legrep)

# CHUNK 4
ggplot(persinj, aes(x = inj)) + geom_bar(aes(fill = legrep))
```

Make a guess of what each chunk of code does. Then run the code in R and see the output.

Problem 2.3.2. (Data exploration: Univariate and bivariate) The `ggplot2` package comes with a dataset named `diamonds` that contains the prices and other attributes of approximately 54,000 diamonds. To load the dataset, use the following commands:

```
library(ggplot2)
data(diamonds)
```

(a) Determine the number of observations and variables in the `diamonds` dataset.

With the aid of appropriate graphical displays and/or summary statistics, complete the following tasks.

(b) Perform univariate exploration of the price of diamonds (`price`) and the quality of the cut (`cut`). Determine if any of these two variables should be transformed and, if so, what transformation should be made. Do your recommended transformation(s), if any, and delete the original variable(s).

(c) Explore the relationship between the price of diamonds and the weight of diamonds (`caret`).

(d) Explore the relationship between the price of diamonds and the quality of the cut.

(e) Reconcile the apparent contradiction between what you get in parts (c) and (d).

Solutions

Solution to Problem 2.3.1. Although the four chunks of code look similar, they produce drastically different output.

- *CHUNK 1:* Here we are making a bar chart for injury code with the *boundary* (not the interior) of the vertical bars color-coded according to legal representation. As you can see, the colors are hardly perceptible.
- *CHUNK 2:* This is similar to CHUNK 1, except that this time it is the *interior* of the vertical bars that is color-coded according to legal representation. The output is the same as the left panel of Figure 2.2.10.
- *CHUNK 3:* This chunk of code does not work (try to run it in R and you will get an error!). The reason is that the `fill` argument (not `fill` aesthetic) of the `geom_bar()` function is mapped to a variable (`legrep` here) instead of a constant (e.g., `"blue"`). This is the opposite of the mistake discussed on page 68.
- *CHUNK 4:* This chunk of code generates the same output as CHUNK 2. Instead of putting the `fill` aesthetic in the `ggplot()` call, it is placed inside the `geom_bar()` function.

□

Solution to Problem 2.3.2. (a) The number of rows of the `diamonds` dataset can be obtained by the `nrow()` function:

```
nrow(diamonds)

## [1] 53940
```

To get the number of variables, we can first extract the column names of the dataset via the `colnames()` function, then apply the `length()` function to calculate its length:

```
length(colnames(diamonds))

## [1] 10
```

More efficiently, we can apply the `dim()` function to `diamonds` to get both of its row and column dimensions.

```
dim(diamonds)

## [1] 53940    10
```

Remark. You can also see the number of observations and variables of the `diamonds` dataset directly from the environment pane: