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.[xi]

```
# 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", "0", "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>      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
```

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:

[xi] 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.

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?

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:[xii]

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."

[xii] 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.

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:

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.

The resulting data frame `actuary.1` still has a missing value in the `name` column. Since names are unlikely to be used as a predictor or target variable, there is no harm in keeping that missing value.

2. *Using the* `complete.cases()` *or* `na.omit()` *functions:* These two functions are more global than `is.na()` and remove all of the missing values in a data frame. When the `complete.cases()` function is applied to a data frame, it returns a logical vector of the same length as the number of rows of the data frame with elements equal to `TRUE` if the corresponding rows contain "complete cases" (i.e., no missing values) and equal to `FALSE` otherwise. When this logical vector is used as the row index vector, only rows without missing values are retained. Run CHUNK 3 to use the `complete.cases()` function to wipe out all of the three missing values in two rows of the `actuary` dataset.

```
# CHUNK 3
cc <- complete.cases(actuary)
actuary.2 <- actuary[cc, ]
actuary.2
```

```
##   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
## 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
```

Observations 2 and 4 are removed as they contain missing values in either `name` or `Q1`.

Using the `na.omit()` function is even more straightforward. When the function is applied to a data frame, any rows with missing data are "omitted." Run CHUNK 4 to use the `na.omit()` function to remove missing values.

```
# CHUNK 4
actuary.3 <- na.omit(actuary)
actuary.3
```

```
##   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
## 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 resulting data frame is identical to that obtained in CHUNK 3.[xiii]

EXAM NOTE

As you eliminate observations with missing values in Exam PA, you can briefly mention in your write-up that doing so is justifiable as long as these observations are few enough (almost always the case in the exam) and there is no evidence that the missing values are systematically generated, so their removal is unlikely to introduce bias or impair the effectiveness of a predictive model.

Finally, we rename `actuary.3` as `actuary.n` ("n" for "new"), which we will continue to work with in the rest of this section. We also calculate the number of rows with missing values by the commands in CHUNK 5. As expected, two rows containing missing values are removed.

```
# CHUNK 5
actuary.n <- actuary.3  # cleaned dataset
actuary.n

##   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
## 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

nrow(actuary) - nrow(actuary.n)

## [1] 2
```

Note that the row numbers in the new data frame correspond to those in the original data frame.

While we have got rid of missing values, some entries in the new dataset are clearly amiss. First, the age of the first actuary, Embryo Luo, is `-1`, which is not possible and is likely the result of a data entry error (the actuary is really an "embryo"!?). Second, the actuary called Peter Smith (third observation in the original dataset) has no exams passed. While this may not be an error for sure, it is highly questionable—in this day and age, it is difficult to land a full-time actuarial job with no exams passed! In general, abnormal values cannot be easily detected by built-in R functions and are often identified with the aid of human input.

How should we deal with these two weird observations? Peter Smith is not as problematic since we are not completely sure whether his `exams` entry is an error or not. It is judicious to retain the entry, but *make a remark about its potential abnormality in your write-up if this was a real PA exam project.* The `age` entry of Embryo Luo is more troublesome. For now, we will keep it and proceed. However, if you are going to fit a predictive model using `age` as a predictor, then it makes sense to exclude Embryo Luo from your analysis.

[xiii] There are some very minor differences between the two data frames `actuary.2` and `actuary.3`. These differences are beyond the scope of this manual.

Task 2: Removing unimportant variables. In the course of a data analysis, we may want to remove variables which turn out to be redundant or add little value to our analysis. In the `actuary.n` data, the `x` variable is a vector of row numbers that do not provide any useful information for understanding the six actuaries. To delete a variable in a data frame, we can assign it to the special symbol `NULL` (not to be confused with `NA`). Run CHUNK 6 to delete the `x` variable in the `actuary.n` dataset.

```
# CHUNK 6
actuary.n$x <- NULL
actuary.n

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

Note that it is very important to append the name of the dataset, which is `actuary.n` in this case; otherwise, the `x` variable in the dataset would remain intact.

Task 3 (a): Adding new variables. The converse to removing an existing variable is to create new variables of interest, which may serve as predictors in a predictive model. To add a new variable to a data frame, we can use the dollar sign `$` notation to both name and define the new variable. The syntax is

```
<data_frame>$<new_var> <- <definition>
```

Even if the new variable do not exist in the original data frame, R will automatically expand the data frame to accommodate the new variable.

As an illustration, run CHUNK 7 to create a new variable that measures the average of the three evaluation ratings for each actuary.

```
# CHUNK 7
actuary.n$S <- (actuary.n$Q1 + actuary.n$Q2 + actuary.n$Q3) / 3
actuary.n

##              name gender age exams Q1 Q2 Q3        S
## 1      Embryo Luo      M  -1    10 10  9  9 9.333333
## 3     Peter Smith      M  22     0  4  5  5 4.666667
## 5 Angela Peterson      F  30     6  8  8 10 8.666667
## 6  Emily Johnston      F  42     7  9 10 10 9.666667
## 7   Barbara Scott      F  29     5  8  9  7 8.000000
## 8    Benjamin Eng      M  36     9  7  8  8 7.666667
```

As in Task 2, it is crucial that we specify the name of the dataset that should host the new variable. If you drop `"actuary.n"` on the left and use the command

```
S <- (actuary.n$Q1 + actuary.n$Q2 + actuary.n$Q3)/3,
```

then the new variable S will exist in the current workspace, but not as a new variable in the actuary.n data frame.

In CHUNK 7, we manually average the three evaluation ratings for each actuary. If there were ten or more ratings, doing such averaging manually will be cumbersome. In this regard, the apply() function can make our work a lot easier. This function allows us to apply an arbitrary function to any dimension of a matrix and a data frame, provided that its columns are of the same data structure. Its general syntax is

```
apply(<data_frame>, MARGIN, <function>),
```

where MARGIN = 1 indicates that the function will act on the rows of the data frame while MARGIN = 2 indicates that the function will act on its columns. Now run CHUNK 8 to calculate the row means for Q1, Q2, and Q3. The output is the same as that in CHUNK 7. (To avoid typing the column index vector c("Q1", "Q2", "Q3") by hand, use the paste0() function; see Example 1.3.3.)

```
# CHUNK 8
actuary.n$S <- apply(actuary.n[, c("Q1", "Q2", "Q3")], 1, mean)
actuary.n

##              name gender age exams Q1 Q2 Q3        S
## 1      Embryo Luo      M  -1    10 10  9  9 9.333333
## 3     Peter Smith      M  22     0  4  5  5 4.666667
## 5 Angela Peterson      F  30     6  8  8 10 8.666667
## 6  Emily Johnston      F  42     7  9 10 10 9.666667
## 7   Barbara Scott      F  29     5  8  9  7 8.000000
## 8    Benjamin Eng      M  36     9  7  8  8 7.666667
```

In the code above, we have extracted only the three columns, Q1, Q2, and Q3, whose row means are of interest.

Example 1.3.1. (Calculating column means) Consider the actuary.n dataset. For each of Q1, Q2, and Q3, write R commands to calculate its mean across the six actuaries.

Solution. We can adapt the commands in CHUNK 8 and change MARGINS from 1 to 2 to calculate the column means.

```
apply(actuary.n[, c("Q1", "Q2", "Q3")], 2, mean)

##       Q1       Q2       Q3
## 7.666667 8.166667 8.166667
```

Alternatively but more tediously, we can apply the mean() function to calculate the mean of each of Q1, Q2, and Q3, then concatenate the three means as a vector.

```
c(mean(actuary.n$Q1), mean(actuary.n$Q2), mean(actuary.n$Q3))

## [1] 7.666667 8.166667 8.166667
```

□

Task 3 (b): Sorting the actuaries with respect to average ratings. Now that we have created the average rating variable, we may want to identify the actuary with the highest rating and, more generally, sort the actuaries according to average ratings. Identifying the best-performing actuary can be easily done by applying the `which.max()` function to the vector of average ratings. As its name suggests, this function returns the index at which the average rating is "maximized." Putting this index to the row subscript of the `actuary.n` dataset while setting the column subscript to `"Name"`, we can identify the best-performing actuary. As you can expect, the `which.min()` function plays the opposite role and returns the index that gives rise to the smallest value of the vector in the argument. Now let's run CHUNK 9 to pick out the best- and worst-performing actuaries (so that we know who to reward and who to fire!)

```
# CHUNK 9
actuary.n[which.max(actuary.n$S), "name"]  # best actuary

## [1] "Emily Johnston"

actuary.n[which.min(actuary.n$S), "name"]  # worst actuary

## [1] "Peter Smith"
```

These results agree with what we observed in CHUNK 8.

To sort a data frame in R, we can use the `order()` function, which returns the permutation of indices when the elements of the vector supplied as an argument is sorted from smallest to largest. This vector of indices can then act as the row index vector to produce the sorted dataset. Let's run CHUNK 10 to see how this works.

```
# CHUNK 10
# Sorted by S in ascending order
actuary.n1 <- actuary.n[order(actuary.n$S), ]
actuary.n1

##              name gender age exams Q1 Q2 Q3        S
## 3     Peter Smith      M  22     0  4  5  5 4.666667
## 8    Benjamin Eng      M  36     9  7  8  8 7.666667
## 7   Barbara Scott      F  29     5  8  9  7 8.000000
## 5 Angela Peterson      F  30     6  8  8 10 8.666667
## 1       Embryo Luo     M  -1    10 10  9  9 9.333333
## 6  Emily Johnston      F  42     7  9 10 10 9.666667

# Sorted by S in descending order
actuary.n2 <- actuary.n[order(actuary.n$S, decreasing = TRUE), ]
actuary.n2

##              name gender age exams Q1 Q2 Q3        S
## 6  Emily Johnston      F  42     7  9 10 10 9.666667
## 1       Embryo Luo     M  -1    10 10  9  9 9.333333
## 5 Angela Peterson      F  30     6  8  8 10 8.666667
## 7   Barbara Scott      F  29     5  8  9  7 8.000000
## 8    Benjamin Eng      M  36     9  7  8  8 7.666667
## 3     Peter Smith      M  22     0  4  5  5 4.666667
```

```
# Sorted by gender, then by S, both in descending order
actuary.n3 <- actuary.n[order(actuary.n$gender, actuary.n$S, decreasing = TRUE), ]
actuary.n3

##              name gender age exams Q1 Q2 Q3        S
## 1      Embryo Luo      M  -1    10 10  9  9 9.333333
## 8    Benjamin Eng      M  36     9  7  8  8 7.666667
## 3     Peter Smith      M  22     0  4  5  5 4.666667
## 6  Emily Johnston      F  42     7  9 10 10 9.666667
## 5 Angela Peterson      F  30     6  8  8 10 8.666667
## 7   Barbara Scott      F  29     5  8  9  7 8.000000
```

In the first example, the six actuaries are sorted by `S` in ascending order, which is the default order. This can be overridden by setting the option `decreasing = TRUE`, meaning sorting in descending order, as in the second example. Sorting can also be done for more than one variables. In the last example, we sort the actuaries first by `gender`, then by `S` within each `gender`, both in descending order. Again, we can see that Emily Johnston has the highest average rating among the six actuaries.

Task 4 (a): Using logical tests to identify observations with certain characteristics. Under the post-July 2018 SOA exam curriculum, an Associate is an individual having passed seven to nine exams and a Fellow should have passed ten exams.[xiv] To single out Associates and Fellows, we can again apply logical subsetting with carefully formulated logical tests. For a small dataset like `actuary.n`, it is easy to spot which actuaries are an Associate or a Fellow. However, the subsetting techniques we introduce here will generalize well to large datasets like those in Exam PA; they call for a more systematic way to search through.

Fellows are easier to identify; they have passed exactly ten exams. Using this criterion to form a logical index vector for the row subscript, we can "extract" Fellows and save their information in a new data frame called `actuary.FSA` as in CHUNK 11.

```
# CHUNK 11
actuary.FSA <- actuary.n[actuary.n$exams == 10, ]
actuary.FSA

##          name gender age exams Q1 Q2 Q3        S
## 1 Embryo Luo      M  -1    10 10  9  9 9.333333
```

Here we are using the logical operator `==` (not to be confused with the equality sign `=`) to test if each element of the vector `actuary.n$exams` is equal to 10. This produces a logical vector with elements equal to `TRUE` if the logical test is true and equal to `FALSE` otherwise (the recycling rule is used implicitly to expand 10 to a 6-element vector where each element equals 10, then element-wise comparisons are performed), i.e., `actuary.n$exams == 10` equals

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

Passing this logical vector as the index vector for the row subscript returns the one and only one actuary, Embryo Luo, who has passed 10 exams, or equivalently, who is a Fellow. A list of commonly used logical operators in Exam PA are given in Table 1.1.

[xiv] For simplicity, we are ignoring e-learning modules and professional seminars like the APC and FAC here.

Logical Operator	Meaning
`>`	Greater than
`>=`	Greater than or equal to
`<`	Less than
`<=`	Less than or equal to
`==`	Exactly equal to
`!=`	Not equal to
`!x`	Negation of a logical vector `x`
`cond1 & cond2`	Are both `cond1` and `cond2` true?
`cond1 | cond2`	Is at least one of `cond1` and `cond2` true?
`a %in% c(a, b, c)`	Is `a` an element of the vector `c(a, b, c)`?

Table 1.1: Commonly used logical operators in Exam PA.

Identifying Associates is more complex. They have passed seven to nine exams. This defining criterion requires multiple logical tests to represent, as the rest of CHUNK 11 shows.

```
# CHUNK 11 (Cont.)
actuary.ASA <- actuary.n[actuary.n$exams >= 7 & actuary.n$exams <= 9, ]
actuary.ASA

##             name gender age exams Q1 Q2 Q3        S
## 6 Emily Johnston      F  42     7  9 10 10 9.666667
## 8   Benjamin Eng      M  36     9  7  8  8 7.666667
```

Here we are combining the results of two logical tests into a single logical vector through the `&` operator.[xv] This is how the code works:

- The logical test `actuary.n$exams >= 7` produces the logical vector

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

- The logical test `actuary.n$exams <= 9` produces the logical vector

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

- The overall logical vector `actuary.n$exams >= 7 & actuary.n$exams <= 9` has elements equal to `TRUE` if and only if the corresponding elements in the two component logical vectors are both `TRUE`. It is given by

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

  It follows that only the fourth and sixth actuaries, Emily Johnston and Benjamin Eng, are extracted as Associates.

[xv] Some call operators like `&` and `|` *Boolean* operators rather than logical operators.

Note that it is necessary to use two complete logical tests, `actuary.n$exams >= 7` and `actuary.n$exams <= 9`, rather than the condensed version `7 <= actuary.n$exams <= 9`, which is closer to what we say in English and Mathematics but unfortunately not accepted by R.

If you are interested in the number of Associates and Fellows, apply the `nrow()` function to the two new sub-data frames.

```
# CHUNK 11 (Cont.)
nrow(actuary.FSA)

## [1] 1

nrow(actuary.ASA)

## [1] 2
```

Example 1.3.2. (Further illustration of logical subsetting) For the `actuary.n` dataset, write R code to produce a data frame named `actuary.reduced` that includes all actuaries who are male *or* are Associates. Sort the observations by the number of exams passed in descending order.

Solution. The relevant logical test corresponding to all actuaries who are male *or* are Associates is:

```
test <- actuary.n$gender == "M" | (actuary.n$exams >= 7 & actuary.n$exams <= 9)
```

Putting `test` as the index vector for the row subscript, we get the desired sub-data frame.

```
actuary.reduced <- actuary.n[test, ]
actuary.reduced

##              name gender age exams Q1 Q2 Q3        S
## 1      Embryo Luo      M  -1    10 10  9  9 9.333333
## 3     Peter Smith      M  22     0  4  5  5 4.666667
## 6 Emily Johnston      F  42     7  9 10 10 9.666667
## 8    Benjamin Eng      M  36     9  7  8  8 7.666667
```

Finally, we sort the sub-data frame with respect to `exams` in descending order.

```
actuary.reduced <- actuary.reduced[order(actuary.reduced$exams,
                                         decreasing = TRUE), ]
actuary.reduced

##              name gender age exams Q1 Q2 Q3        S
## 1      Embryo Luo      M  -1    10 10  9  9 9.333333
## 8    Benjamin Eng      M  36     9  7  8  8 7.666667
## 6 Emily Johnston      F  42     7  9 10 10 9.666667
## 3     Peter Smith      M  22     0  4  5  5 4.666667
```

The commands above are collected in CHUNK 12. □

Task 4 (b): Creating a new categorical variable representing designations. Along the same line, we now turn to the creation of a new categorical variable that indicates whether each actuary is a Fellow (`"FSA"`), an Associate (`"ASA"`), or a pre-ASA student (`"student"`). Such a designation indicator can be developed by the following binary classification rules:

- If the number of exams passed is 10, then the actuary is a Fellow.
- Otherwise, if the number of exams passed is between 7 and 9, then the actuary is an Associate.
- Otherwise, the actuary is a student. (We still call him/her an actuary anyway)

These classification rules can be incorporated into R by the `ifelse()` construct, which can be considered a vectorized version of `if-else` on page 36 (recall that the logical test in `if-else` must return a logical *scalar*, not a vector). Its general syntax is

```
ifelse(<cond>, <result_1>, <result_2>)
```

where

`<cond>` is the logical test to be conducted on a vector,

`<result_1>` is the return value if the corresponding element of the logical test is true, and

`<result_2>` is the return value if the corresponding element of the logical test is false.

The result of `ifelse()` is a vector that is of the same length as the vector in the logical test in the first argument and is defined by the results in the second and third arguments. Here is a simple example:

```
x <- 1:3
# only first element of logical test is TRUE
ifelse(x == 1, "Yes", "No")

## [1] "Yes" "No"  "No"
```

Run CHUNK 13 to use `ifelse()` to define a new variable called `title` that represents the designation held by each actuary.

```
# CHUNK 13
actuary.n$title <- ifelse(actuary.n$exams == 10, "FSA",
                          ifelse(actuary.n$exams >= 7 & actuary.n$exams <= 9,
                                 "ASA", "student"))
actuary.n

##              name gender age exams Q1 Q2 Q3        S   title
## 1      Embryo Luo      M  -1    10 10  9  9 9.333333     FSA
## 3     Peter Smith      M  22     0  4  5  5 4.666667 student
## 5 Angela Peterson      F  30     6  8  8 10 8.666667 student
## 6  Emily Johnston      F  42     7  9 10 10 9.666667     ASA
## 7   Barbara Scott      F  29     5  8  9  7 8.000000 student
## 8    Benjamin Eng      M  36     9  7  8  8 7.666667     ASA
```

Notice that there are two `ifelse()`, one nested within another, due to the three-level nature of `title`.

Task 5: Horizontal and vertical concatenations. In the course of a data analysis, we may have access to an external dataset with the same columns or the same number of observations as the existing dataset.

- If the external dataset refers to the same observations as the current dataset, then we can do a horizontal concatenation and add new variables. This is achieved by the `cbind()` function, with the letter "c" standing for "columns." (Note that the `c()` function is for vectors and cannot be used to concatenate data frames.)
- If the external dataset has the same variables as the current dataset, then we can do a vertical concatenation and add new observations. This is achieved by the `rbind()` function, with the letter "r" standing for "rows."

In Exam PA, `cbind()` is more commonly used as there are functions generating a new data frame or new variables which we want to attach to an existing data frame. Let's suppose that we have an external source providing the smoking status (`"Y"` for smoking and `"N"` for not smoking) and the current salaries of the original eight actuaries. Run CHUNK 14 to set up this new data frame and combine it with the `actuary.n` dataset.

```
# CHUNK 14
new <- data.frame(smoke = c("N", "N", "Y", "Y", "N", "N", "N", "Y"),
                  salary = c(180000, 80000, 70000, 200000, 105000,
                             120000, 105000, 140000))

actuary.n <- cbind(actuary.n, new[cc, ])
actuary.n

##              name gender age exams Q1 Q2 Q3        S   title smoke salary
## 1      Embryo Luo      M  -1    10 10  9  9 9.333333     FSA     N 180000
## 3     Peter Smith      M  22     0  4  5  5 4.666667 student     Y  70000
## 5 Angela Peterson      F  30     6  8  8 10 8.666667 student     N 105000
## 6  Emily Johnston      F  42     7  9 10 10 9.666667     ASA     N 120000
## 7   Barbara Scott      F  29     5  8  9  7 8.000000 student     N 105000
## 8    Benjamin Eng      M  36     9  7  8  8 7.666667     ASA     Y 140000
```

Notice that we have used the `cc` vector defined in CHUNK 3 to exclude the second and fourth rows of the `new` data frame as these two rows contain missing values in the original `actuary` dataset. When using the `cbind()` function, we have to ensure that the data frames to be combined have the same number of rows.

Task 6: How to create a compound variable? Our last data management task is to define a *compound variable* that represents both the gender and smoking status of an actuary in a single variable. This amounts to pairing the level of gender (`"M"` or `"F"`) with the level of smoking status (`"Y"` or `"N"`) for each actuary. Such a compound variable can be useful when analyzing interaction effects between gender and smoking status.

There are a number of R functions that are designed to combine vectors of numeric or character variables to form new character strings. This can be useful for defining a formula in R (see Chapter 3) and interweaving existing categorical variables. The most important function for manipulating

character strings in Exam PA is the `paste()` function, which allows us to "paste" its arguments as character strings. The function is usually used when one or more of its arguments are vectors, in which case a vector of character strings will be produced (the recycling rule will be used if the vectors in the arguments are of unequal length). Let's run CHUNK 15 to see some examples.

```
# CHUNK 15
paste("Hello", "There!")

## [1] "Hello There!"

paste(c("a", "b", "c"), c("x", "y", "z"), sep = "")  # 3 strings

## [1] "ax" "by" "cz"

paste("The answer is", c("YES", "NO"))  # recycling rule

## [1] "The answer is YES" "The answer is NO"

paste0("Yr", 1:10)  # recycling rule

##  [1] "Yr1"  "Yr2"  "Yr3"  "Yr4"  "Yr5"  "Yr6"  "Yr7"  "Yr8"  "Yr9"  "Yr10"

paste("The answer is", c("YES", "NO"), sep = "...", collapse = " OR ")

## [1] "The answer is...YES OR The answer is...NO"
```

The `paste()` function has two optional arguments:

- The `sep` argument supplies a delimiter to separate the individual vectors. The default of the `sep` argument is one space (`" "`). In the second example above, we have changed the argument to no space (`""`), that is, no separator is used. In fact, the `paste0()` function used in the fourth example is a wrapper for `paste()` with the `sep` argument set *a priori* to "zero" space (`""`).
- When a vector of character strings will be produced by the `paste()` function but you want to "collapse" them into a single character string, the `collapse` argument can serve our purpose. The multiple character strings will then be concatenated using the delimiter indicated by the `collapse` argument.

Back to the `actuary.n` dataset, the compound variable we aim to create is a combination of gender and smoking status, with possible levels being `MY`, `MN`, `FY`, and `FN`. We do not want any space to go between the level of gender and the level of smoking status, so the `paste0()` function will serve our purpose well. Run CHUNK 16 to set up the compound variable named `genderSmoke`.

```
# CHUNK 16
actuary.n$genderSmoke <- paste0(actuary.n$gender, actuary.n$smoke)
actuary.n

##              name gender age exams Q1 Q2 Q3        S   title smoke salary
```

```
## 1      Embryo Luo      M  -1    10 10  9  9 9.333333      FSA      N 180000
## 3     Peter Smith      M  22     0  4  5  5 4.666667  student      Y  70000
## 5 Angela Peterson      F  30     6  8  8 10 8.666667  student      N 105000
## 6  Emily Johnston      F  42     7  9 10 10 9.666667      ASA      N 120000
## 7   Barbara Scott      F  29     5  8  9  7 8.000000  student      N 105000
## 8    Benjamin Eng      M  36     9  7  8  8 7.666667      ASA      Y 140000
##   genderSmoke
## 1          MN
## 3          MY
## 5          FN
## 6          FN
## 7          FN
## 8          MY
```

Example 1.3.3. (Further practice with the `paste0()` function) Use `paste()` or `paste0()` to rewrite the code in CHUNK 8 without typing `"Q1"`, `"Q2"`, and `"Q3"` one by one.

Solution. To produce the vector of character strings `c("Q1", "Q2", "Q3")`, we can apply the `paste0()` function to the string `"Q"` and the numeric vector `1:3` (no delimiter is needed):

```
text <- paste0("Q", 1:3)
text
```

```
## [1] "Q1" "Q2" "Q3"
```

Putting the `text` vector to the column subscript of the `actuary.n` dataset, we get the desired result:

```
actuary.n$S.new <- apply(actuary.n[, text], 1, mean)
actuary.n
```

```
##              name gender age exams Q1 Q2 Q3        S   title smoke salary
## 1      Embryo Luo      M  -1    10 10  9  9 9.333333     FSA     N 180000
## 3     Peter Smith      M  22     0  4  5  5 4.666667 student     Y  70000
## 5 Angela Peterson      F  30     6  8  8 10 8.666667 student     N 105000
## 6  Emily Johnston      F  42     7  9 10 10 9.666667     ASA     N 120000
## 7   Barbara Scott      F  29     5  8  9  7 8.000000 student     N 105000
## 8    Benjamin Eng      M  36     9  7  8  8 7.666667     ASA     Y 140000
##   genderSmoke    S.new
## 1          MN 9.333333
## 3          MY 4.666667
## 5          FN 8.666667
## 6          FN 9.666667
## 7          FN 8.000000
## 8          MY 7.666667
```

As you can see, the `S.new` variable is identical to `S`. □

Remark. The commands in this example are collected in CHUNK 17.