Chapter 6
Correlation

Abstract This chapter introduces data frames, random sampling, and correlation. Readers learn how to perform permutation tests to assess the significance of derived correlations.

6.1 Introduction

It might be tempting to look at the graphs you have produced thus far and begin forming an argument about the relative importance of Ahab versus the whale in Melville's novel. Occurrences of *whale* certainly appear to occupy the central portion of the book, whereas *Ahab* is present at the beginning and at the end. It might also be tempting to begin thinking about the structure of the novel, and this data does provide some evidence for an argument about how the human dimensions of the narrative frame the more naturalistic. But is there, in fact, an inverse relationship?

6.2 Start Up Code

```r
rm(list = ls()) # Clear Workspace
text_v <- scan("data/text/melville.txt", what = "character", sep = "\n")
start_v <- which(text_v == "CHAPTER 1. Loomings.")
novel_lines_v <-  text_v[start_v:length(text_v)]
chap_positions_v <- grep("^CHAPTER \\d", novel_lines_v)
last_position_v <-  length(novel_lines_v)
chap_positions_v  <-  c(chap_positions_v , last_position_v)
```

M. L. Jockers, R. Thalken, *Text Analysis with R*, Quantitative Methods in the Humanities and Social Sciences,
https://doi.org/10.1007/978-3-030-39643-5_6

```r
chapter_raws_l <- list()
chapter_freqs_l <- list()
for(i in 1:length(chap_positions_v)){
  if(i != length(chap_positions_v)){
    chapter_title <- novel_lines_v[chap_positions_v[i]]
    start <- chap_positions_v[i] + 1
    end <- chap_positions_v[i + 1] - 1
    chapter_lines_v <- novel_lines_v[start:end]
    chapter_words_v <- tolower(paste(chapter_lines_v, collapse = " "))
    chapter_words_l <- strsplit(chapter_words_v, "\\W")
    chapter_word_v <- unlist(chapter_words_l)
    chapter_word_v <- chapter_word_v[which(chapter_word_v != "")]
    chapter_freqs_t <- table(chapter_word_v)
    chapter_raws_l[[chapter_title]] <-  chapter_freqs_t
    chapter_freqs_t_rel <- 100*(chapter_freqs_t/sum(chapter_freqs_t))
    chapter_freqs_l[[chapter_title]] <- chapter_freqs_t_rel
  }
}
whale_l <- lapply(chapter_freqs_l, '[', 'whale')
whales_m <- do.call(rbind, whale_l)
ahab_l <- lapply(chapter_freqs_l, '[', 'ahab')
ahabs_m <- do.call(rbind, ahab_l)
whales_v <- as.vector(whales_m[,1])
ahabs_v <- as.vector(ahabs_m[,1])
whales_ahabs_m <- cbind(whales_v, ahabs_v)
colnames(whales_ahabs_m) <- c("whale", "ahab")
```

6.3 Correlation Analysis

Using the frequency data you compiled for *ahab* and *whale*, you can run a correlation analysis to see if there is a statistically significant relationship between the two variables. A correlation analysis attempts to determine the extent to which there is a relationship, or linear dependence, between two sets of points. Thought of another way, correlation analysis attempts to assess the way that the occurrences of *whale* and *ahab* behave in unison or in opposition to each other over the course of the novel. You can use a correlation analysis to answer a question such as: to what extent does the usage of *whale* change (increase or decrease) in relation to the usage of *ahab*? R offers a simple function, `cor`, for calculating the strength of a possible correlation. But before you can employ the `cor` function on the `whales_ahabs_m` object, you need to deal with the fact that there are some cells in the matrix that contain the value `NA`. Not every chapter in *Moby Dick* had an occurrence of *whale* (or *ahab*), so in the previous practice exercise when you ran

```
whale_l  <-  lapply(chapter_freqs_l, "[", "whale")
```

R found no *hits* for *whale* in some chapters of the novel and recorded an NA, as in *not available* or *missing*. You may recall seeing this NA output when you viewed the contents of whales_ahabs_m matrix:

```
whales_ahabs_m[1:16, ]
##             whale      ahab
##  [1,] 0.13368984        NA
##  [2,] 0.06882312        NA
##  [3,] 0.10000000        NA
##  [4,]         NA        NA
##  [5,]         NA        NA
##  [6,] 0.24067389        NA
##  [7,] 0.21097046        NA
##  [8,]         NA        NA
##  [9,] 0.24711697        NA
## [10,]         NA        NA
## [11,]         NA        NA
## [12,]         NA        NA
## [13,] 0.17341040        NA
## [14,]         NA        NA
## [15,]         NA        NA
## [16,] 0.16037063 0.3385602
```

As you see here, there are no occurrences of *whale* in chapters 4 or 5 and no occurrences of *ahab* until chapter 16. Because cor is a mathematical function that requires numerical data, you need to replace the NA values before running the correlation analysis. Since the appearance of an NA in these cases is equivalent to *zero* (there are exactly *zero* occurrences of the keyword in the given chapter), you can safely replace all the occurrences of NA in the whales_ahabs_m matrix with zero. One way to do this is by embedding the conditional is.na function inside a call to the which function as in: which(is.na(whales_ahabs_m)). To set the values to 0, place the entire expression inside the brackets of whales_ahabs_m and assign a 0 to those items that meet the condition:

```
whales_ahabs_m[which(is.na(whales_ahabs_m))] <- 0
```

This is the short and easy way to achieve our objective, but for the sake of illustration we will break it down with comments added to explain what is going on:

```
# identify the position of NA values in the matrix
the_na_positions <- which(is.na(whales_ahabs_m))
# set the values held in the found positions to zero
whales_ahabs_m[the_na_positions] <- 0
```

With the `NAs` set to zero, the correlation can be run.

```
cor(whales_ahabs_m)
##              whale       ahab
## whale  1.0000000 -0.2411126
## ahab  -0.2411126  1.0000000
```

Because `whales_ahabs_m` is a matrix of two columns, the result of calling `cor` is a new matrix containing two rows and two columns. The row and column names are the same, and the values held in the cells are the correlation values. It is no surprise to see that *whale* is perfectly correlated with *whale* and *ahab* with *ahab*. The positive `1.0000000` in these cells is not very informative, which is to say that running `cor` over the entire matrix as we have done here results in a lot of extraneous information. That is because `cor` runs the correlation analysis for every possible combination of columns in the matrix. With a two column matrix such as this, it is really overkill. The results could be made a lot simpler by just giving `cor` the two vectors that you really want to correlate:

```
mycor <- cor(whales_ahabs_m[,"whale"], whales_ahabs_m[,"ahab"])
mycor
## [1] -0.2411126
```

The resulting number (-0.2411126) is a measure of the strength of linear dependence between the values in the *whale* column and the values in the *ahab* column. This result, called the *Pearson Product-moment correlation coefficient*, is expressed as a number between `-1` and `+1`. A negative one (`-1`) coefficient represents perfectly negative correlation; if the correlation between *ahab* and *whale* were `-1`, then we would know that as the usage of *whale* increases, the usage of *ahab* decreases proportionally. Positive one (`+1`) represents perfect positive correlation (as one variable goes up and down the other variable does so in an identical way). Zero (`0`) represents no correlation at all.

The further the coefficient is from zero, in either a positive or negative direction, the stronger the correlation; conversely the closer the result is to `0`, the less dependence there is between the two variables. Here, with *whale* and *ahab* a correlation coefficient of -0.2411126 is observed. This suggests that while there is a slight inverse relationship (i.e., negative correlation), it is not strongly correlated since the result is closer to `0` than to `-1`. Having said that, how one interprets the meaning, or significance, of the correlation indicated by this coefficient is largely dependent upon the context of the analysis and upon the number of observations or data points under consideration. Generally speaking a coefficient between `-0.3` and `-0.1` on the negative side of `0` and between `0.1` and `0.3` on the positive side of `0` is considered quite small. Strong correlation is usually seen as existing at levels less than `-0.5` or greater than `0.5`.

This correlation test does not lead us to any easy conclusions about the relationship between occurrences of *whale* and occurrences of *ahab*. These two data points, for *ahab* and *whale*, appear to show only a weak inverse relationship. Nevertheless, there is much more to be considered.

Consider, for example, what we explored in Chap. 4, and how the use of synonyms and pronouns complicates these results. When *Ahab* is not being referred to by name, he is undoubtedly appearing as either *he* or *him*. The same may be said for the *whale* and the various appellations of *whale* that Melville evokes: *monster, leviathan, etc.* Using the techniques described in Chap. 4, you could investigate all of these and more. But before leaving this seemingly weak correlation, it might be useful to run a few more experiments to see just how significant or insignificant the result really is.

As noted above, the number of samples can be a factor in how the importance of the correlation coefficient is judged, and in this case there are `135` observations for each variable: one observation for each chapter in the novel.

One way of contextualizing this coefficient is to calculate how likely it is that we would observe this coefficient by mere chance alone. In other words, assuming there is no relationship between the occurrences of *whale* and *ahab* in the novel, what are the chances of observing a correlation coefficient of -0.2411126? A fairly simple test can be constructed by randomizing the order of the values in either the *ahabs* or the *whales* column and then retesting the correlation of the data.

6.4 A Word About Data Frames

Before explaining the randomization test in detail, we want to return to something mentioned earlier about the R matrix object and its limitations and then introduce you to another important data object in R: the *data frame*.

Thus far we have barely used data frames, but as it happens, data frames are R's bread and butter data type, and they offer us some flexibility that we do not get with matrix objects. Like a matrix, a data frame can be thought of as similar to a table in a database or a sheet in an Excel file: a data frame has some number of rows and some number of columns, and each column contains a specific type of data. A major difference between a matrix and a data frame, however, is that in a data frame, one column may contain character values and another numerical values. To see how this works, enter the following code to create a simple matrix of three rows by three columns:

```
x <- matrix(1, 3, 3)
x
##      [,1] [,2] [,3]
## [1,]    1    1    1
## [2,]    1    1    1
## [3,]    1    1    1
```

If you ask R to return the data type (class) of any one of the values in this matrix, it will return the class *numeric*.

```
class(x[1,2]) # get class of cell in first row second column
## [1] "numeric"
```

Now change the value of one cell in this matrix so that it contains *character* data instead of a number.

```
x[1,2] <- "Sam I am"
x
##      [,1] [,2]       [,3]
## [1,] "1"  "Sam I am" "1"
## [2,] "1"  "1"        "1"
## [3,] "1"  "1"        "1"
```

You will notice right away that all of the values in the matrix are now shown inside quotation marks. This is because the entire matrix has been converted to character data. Those `1`'s are no longer numbers, they are the `1` *character*. Among other things, this means that you cannot perform mathematical operations on them anymore! If you check the class, R will report the change:

```
class(x[1,2]) # get class of cell in first row second column
## [1] "character"
class(x[1,3]) # get class of cell in first row third column
## [1] "character"
```

To see the difference between a matrix and a data frame, recreate the first matrix example and then convert it to a data frame, like this:

```
x <- matrix(1, 3, 3)
x_df <- as.data.frame(x)
x_df
##   V1 V2 V3
## 1  1  1  1
## 2  1  1  1
## 3  1  1  1
```

You can see immediately that a data frame displays differently. Instead of bracketed row and column numbers, you now see column headers (V1, V2, V3) and simple row numbers without the brackets. You can now repeat the experiment from above and assign some character data into one of the cells in this data frame.

```
x_df[1,2] <- "Sam I am"
class(x_df[1,2]) # get class of cell in first row second column
## [1] "character"
class(x_df[1,3]) # get class of cell in first row third column
## [1] "numeric"
x_df
##   V1       V2 V3
## 1  1 Sam I am  1
## 2  1        1  1
## 3  1        1  1
```

When using a matrix, the assignment of character data to any one cell resulted in all the cells in the matrix being converted into character data. Here, with a data frame, only the data in the column containing the target cell are converted to character data, not the entire table of data. The takeaway is that a data frame can have columns containing different types of data. This will be especially useful as your data get more complicated. You may, for example, want a way of storing character based metadata (such as *author gender*, or *chapter title*) alongside the numerical data associated with these metadata facets.

Another handy thing about data frames is that you can access columns of data using a bit of R shorthand. If you want to see all the values in the second column of the `x_df` variable, you can do so using bracketed index references, just as you have done previously with matrix objects. To see the entire second column, for example, you might do this:

```
x_df[,2]
## [1] "Sam I am" "1"        "1"
```

Alternatively, you can use the fact that the data frame has a header to get column information by referencing the column name, like this:

```
x_df[,"V2"]
## [1] "Sam I am" "1"        "1"
```

And, most alternatively, you can use the shorthand (`$`) to get column data like this:

```
x_df$V2
## [1] "Sam I am" "1"        "1"
```

That is a basic overview of data frames. Now we will return to correlating values in *Moby Dick*.

6.5 Testing Correlation with Randomization

In this section you will use your new knowledge of data frames. First convert the matrix object `whales_ahabs_m` into a data frame called `cor_data_df`:

```
cor_data_df <- as.data.frame(whales_ahabs_m)
```

As a gut check, you can use the `cor` function on the entire data frame, just as you did with the matrix object. The output should be the same.

```
cor(cor_data_df)
##             whale       ahab
## whale  1.0000000 -0.2411126
## ahab  -0.2411126  1.0000000
```

The goal now is to determine if that observed correlation coefficient of -0.2411126 could have been likely to occur by mere chance. To assess this you are going to take the values for one of the two columns in the data frame and *shuffle* them into a random order. You will then run a new correlation test with the randomized column. In this way, a chance distribution of the values that is independent of the actual structure of the chapters in the book can be simulated. If the correlation of the shuffled data is similar to the actual (as in *unshuffled*) data, then you will have to concede that the relationship between *whale* and *ahab* observed in the actual data is really no different from what might be observed if you threw all the occurrences of *whale* and *ahab* up in the air and then created `135` arbitrary piles.

The first step is to randomize the order of the values (the word frequency measurements) in one of the two columns of data in `cor_data_df`. Since the columns contain chapter-by-chapter measurements, this randomizing will have the effect of shuffling the chapter order for one set of measurements and leaving the other set in chronological order. R provides a function called `sample` for generating a random shuffling of data. At its most simple, the `sample` function requires a vector of values to shuffle. So, to get a random ordering of the values in the *whale* column of `cor_data_df` you can simply enter:

```
sample(cor_data_df$whale)
```

Go ahead and try entering this a few times and you will see that each time `sample` randomly shuffles the order of the values from the *whale* column.

With the ability to randomize the values, you now need to correlate these randomized values against the ordered values in the unshuffled *ahab* column. Using the dollar sign to reference the columns in the data frame, the expression can be written as simply as this:

```
cor(sample(cor_data_df$whale), cor_data_df$ahab)
```

In our first test of this code, R returned a correlation coefficient of −0.0094803. Your results will be different given the random sampling. We then copied and pasted this code ten more times and observed the following correlation coefficients for the various shuffles of the data.

```
## [1] 0.122331
## [1] 0.00818978
## [1] -0.01610114
## [1] -0.1289073
## [1] 0.05115036
## [1] 0.0443622
## [1] 0.08513762
## [1] -0.1019796
## [1] 0.07842781
## [1] 0.04410211
```

As you see, in this small sample of ten randomizations, the highest positive correlation coefficient was `0.122331` and the lowest negative correlation coefficient was `-0.1289073`. Remember that the actual correlation coefficient before we began shuffling anything was `-0.2411126`. In other words, the actual data seems to be quite a bit below (i.e., further from `0`) what is observed when shuffling the data and simulating a chance distribution of values. Still, `10` randomizations are not very many. Instead of copying and pasting the code over and over again, you can develop a more programmatic way of testing the correlation using a `for` loop and `10,000` iterations!

With a `for` loop you can repeat the randomization and correlation test process multiple times and at each iteration capture the result into a new vector. With this new vector of `10,000` correlation values, it will be easy to generate some statistics that describe the distribution of the random data and offer a better way of assessing the significance of the actual observed correlation coefficient in the unshuffled data.

The code required for this is simple. Begin by creating an empty container variable called `mycors_v`, and then create a `for` loop that iterates a set number of times (`10,000` in our example). Within the curly braces of that loop, you will add code for shuffling and then correlating the vectors. At each step in the loop, you will capture the correlation coefficient by adding it to the `mycors_v` vector using the `c` function. Here is how we wrote it:

```
mycors_v <- NULL
for(i in 1:10000){
  mycors_v <- c(
    mycors_v,
    cor(sample(cor_data_df$whale),
        cor_data_df$ahab)
    )
}
```

With this step completed, you can now use some basic R functions such as `min`, `max`, `range`, `mean`, and `sd` to get a general sense of the results.

Here is what our randomization tests returned; your results will be similar but not identical:

```
min(mycors_v)
## [1] -0.2992775
max(mycors_v)
## [1] 0.3484449
range(mycors_v)
## [1] -0.2992775  0.3484449
mean(mycors_v)
## [1] -0.0003656731
sd(mycors_v)
## [1] 0.08633026
```

What these descriptive statistics reveal is that our actual observed value is more typical of the extremes than the norm. A low `standard deviation` suggests that most of the values recorded are close to the `mean`, and here the `mean` is very close to zero ($-3.6567306 \times 10^{-4}$), which you will recall from above can be interpreted as meaning very little correlation. A high `standard deviation` would indicate that the values are spread out over a wide range of values. So even though the `min` value of `-0.2992775` is slightly less than our actual observed value of `-0.2411126`, that `-0.2992775` is very *atypical* of the randomized data. In fact, using a bit of additional code that we will not explain here, we can generate a plot showing the distribution of all the values in `mycors_v` (Fig. 6.1).

```
h <- hist(mycors_v, breaks = 100, col="grey",
          xlab = "Correlation Coefficient",
          main = "Histogram of Random Correlation Coefficients\n
          with Normal Curve",
          plot = T)
xfit <- seq(min(mycors_v), max(mycors_v), length = 1000)
yfit <- dnorm(xfit, mean = mean(mycors_v), sd = sd(mycors_v))
yfit  <-  yfit * diff(h$mids[1:2]) * length(mycors_v)
lines(xfit, yfit, col = "black", lwd = 2)
```

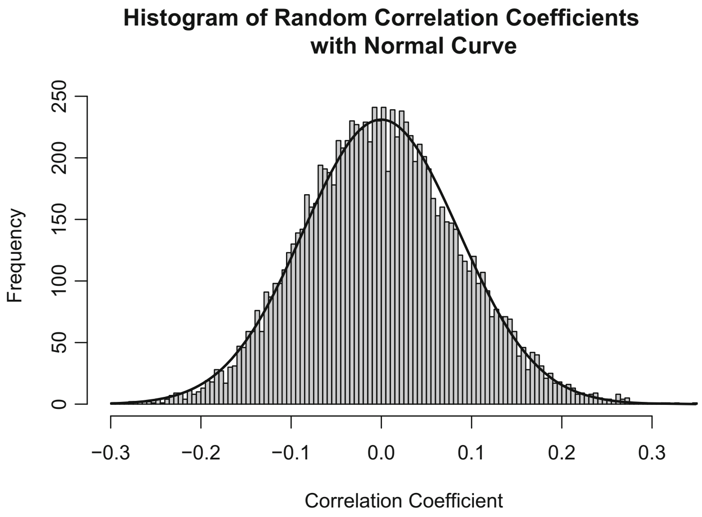

Fig. 6.1 Histogram plot of random correlation coefficients

The plot reveals, in dramatic fashion, just how much the data clusters around the `mean`, which as you recall from above is nearly `0`. It also dramatizes the outlier status of the actual value (−0.2411126) that was observed. In `10,000` random iterations, only 19 correlation coefficients were calculated to be less than the actual observed value and the actual observed value was nearly 3 (2.79) `standard deviations` away from the `mean`. In short, the probability of observing a random value as extreme as the actual value observed (−0.2411126) is just 0.48%.[1]

6.6 Practice

1. Add two more columns to the matrix with data for the words *i* and *my* and then rerun the `cor` function. Though we have only used `cor` for two columns so far, we can use it just as easily on a matrix with two or more columns. Do not forget to set the frequencies for any chapters where the word *does not* occur to zero. What does the result tell you about the usage of the words *i* and *my*?

2. Calculate the correlation coefficient for *i* and *my* and run a randomization test to evaluate whether the results are significant.

[1]Another way to test the significance of a correlation coefficient is to use the `cor.test` function. Use `?cor.test` to learn about this function and then run it using the `method = "pearson"` argument. To make more sense out of the results, consider consulting http://en.wikipedia.org/wiki/P-value on t-tests.