Woltermann: Getting familiar with R

Completed by: Leon Woltermann

NB: The worksheet has beed developed and prepared by Lincoln Mullen. Source: Lincoln A. Mullen, *Computational Historical Thinking: With Applications in R (2018–)*: http://dh-r.lincolnmullen.com.

Aim of this worksheet

After completing this worksheet, you should feel comfortable typing commands into the R console and into an R Markdown document. In particular, you should know how to use values, variables, and functions, how to install and load packages, and how to use the built-in help for R and its packages.

Values

R lets you store several different kinds of *values*. These values are the information that we actually want to do something with.

One kind of value is a number. Notice that typing this number, either in an R Markdown document or at the console, produces an identical output

```
42
```

```
## [1] 42
```

(1) Create a numeric value that has a decimal point:

```
1.2
```

```
## [1] 1.2
```

Of course numbers can be added together (with `+`), subtracted (with `-`), multiplied (with `*`), and divided (with `/`), along with other arithmetical operations. Let's add two numbers, which will produce a new number.

```
2 + 2
```

```
## [1] 4
```

(2) Add two lines, one that multiplies two numbers, and another that subtracts two numbers.

```
1*1
```

```
## [1] 1
```

```
10/2
```

```
## [1] 5
```

Another important kind of value is a character vector. (Most other programming languages would call these strings.) These contain text. To create a string, include some characters in between quotation marks `""`. (Single quotation marks work too, but in general use double-quotation marks as a matter of style.) For instance:

```
"Hello, beginning R programmer"
```

```
## [1] "Hello, beginning R programmer"
```

(3) Create a string with your own message.

```
"hello world"
```

```
## [1] "hello world"
```

Character vectors can't be added together with `+`. But they can be joined together with the `paste()` function.

```
paste("Hello", "everybody")
```

```
## [1] "Hello everybody"
```

(4) Mimic the example above and paste three strings together.

```
paste("hello", "world", "everyone!")
```

```
## [1] "hello world everyone!"
```

(5) Now explain in a sentence what happened.

The function "paste()" merges strings.

Another very important kind of value are logical values. There are only two of them: `TRUE` and `FALSE`.

```
# This is true
TRUE
```

```
## [1] TRUE
```

```
# This is false
FALSE
```

```
## [1] FALSE
```

Notice that in the block above, the `#` character starts a *comment*. That means that from that point on, R will ignore whatever is on that line until a new line begins.

Logical values aren't very exciting, but they are useful when we compare other values to one another. For instance, we can compare two numbers to one another.

```
2 < 3
```

```
## [1] TRUE
```

```
2 > 3
```

```
## [1] FALSE
```

```
2 == 3
```

```
## [1] FALSE
```

(6) What do each of those comparison operators do? (Note the double equal sign: `==`.)

"<" returns true if a value is greater, false if lesser. ">" returs true if a value is lesser, false if greater. "==" returns true if values are equal, false if not.

(7) Create your own comparisons between numeric values. See if you can create a comparison between character vectors.

```
1 < 2
```

```
## [1] TRUE
```

```
1100 > 22334
```

```
## [1] FALSE
```

```
1.2 == 1.2
```

```
## [1] TRUE
```

R has a special kind of value: the missing value. This is represented by `NA`.

```
NA
```

```
## [1] NA
```

Try adding `2 + NA`.

```
2 + NA
```

```
## [1] NA
```

(8) Does that answer make sense? Why or why not?

NA is not a numeric values. It is not 0. It is a different data type. Therefore, the operation is not 2 + 0 but 2 + none which returns none.

Variables

We wouldn't be able to get very far if we only used values. We also need a place to store them, a way of writing them to the computer's memory. We can do that by *assignment* to a variable. Assignment has three parts: it has the name of a variable (which cannot contain spaces), an assignment operator, and a value that will be assigned. Most programming languages use a rinky-dink `=` for assignment, which works in R too. But R is awesome because the assignment operator is `<-`, a lovely little arrow which tells you that the value goes into the variable. For example:

```
number <- 42
```

Notice that nothing was printed as output when we did that. But now we can just type `number` and get the value which is stored in the variable.

```
number
```

```
## [1] 42
```

It works with character vectors too.

```
computer_name <- "HAL 9000"
```

No output, but this prints the value stored in the variable.

```
computer_name
```

```
## [1] "HAL 9000"
```

(9) In the assignment above, what is the name of the variable? What is the assignment operator? What is the value assigned to the variable?

The variable name is computer_name. The assignment operator is <-. The value is the string "HAl 9000"

Notice that we can use variables any place that we used to use values. For example:

```
x <- 2
y <- 5
x * y
```

```
## [1] 10
```

```
x + 9
```

```
## [1] 11
```

(10) Explain in your own words what just happened.

The numeric values 2 and 5 got assigned to the variables x and y. The variables now store the numeric values.

(11) Now create two assignments. Assign a number to a variable and a character vector to a different variable.

```
a = 5
b = "apples"
```

(12) Now create a third variable with a numeric value and, using the variable with a numeric value from earlier, add them together.

```
c = paste(a, b)
print(c)
```

```
## [1] "5 apples"
```

Can you predict what the result of running this code will be? (That is, what value is stored in `a`?)

```
a <- 10
b <- 20
a <- b
a
```

(13) Predict your answer, then run the code. What is the value stored in `a` by the end? Explain why you were right or wrong.

20

Vectors

Variables are better than just values, but we still need to be able to store multiple values. If we have to store each value in its own variable, then we are going to need a lot of variables. In R every value is actually a vector. That means it can store more more than one value.

Notice the funny output after running this line:

```
"Some words"
```

```
## [1] "Some words"
```

What does the `[1]` in the output mean? It means that the value has one item inside it. We can test that with the `length()` function

```
length("Some words")
```

```
## [1] 1
```

Sure enough, the length is `1`: R is counting the number of items, not the number of words or characters.

That would seem to imply that we can add multiple items (or values) inside a vector. R lets us do that with the `c()` (for "combine") function.

```
many <- c(1, 5, 2, 3, 7)
many
```

```
## [1] 1 5 2 3 7
```

(14) What is the length of the vector stored in `many`?

```
length(many)
```

```
## [1] 5
```

Let's try multiplying `many` by 2:

```
many * 2
```

```
## [1]  2 10  4  6 14
```

(15) What happened?

all items in the vector were multiplied by 2

(16) What happens when you add 2 to `many`?

```
many + 2
```

```
## [1] 3 7 4 5 9
```

(17) Can you create a variable containing several names as a character vectors?

```
names = c("peter", "anna", "daniel")
```

(18) Hard question: what is happening here? Why does R give you a warning message?

```
c(1, 2, 3, 4, 5) + c(10, 20)
```

```
## Warning in c(1, 2, 3, 4, 5) + c(10, 20): longer object length is not a multiple
## of shorter object length
```

```
## [1] 11 22 13 24 15
```

This operation adds the first value of the first vector to the first value of the second vector and so on. This operation cant be finished perfectly since the length of the longer vector has to be a multiply of the length of the shorter vector. In this example the last object of the first list is added to the first object of the second list, so the second object of the second list remains. The operation is calculated as follows: 1+10 2+20 3+10 4+20 5+10 'no value + 20' would be the last step for the operation to be complete but there is no value because 2 is not a multiply of 5. So r gives a warning message.

Built-in functions

Wouldn't it be nice to be able to do something with data? Let's take some made up data: the price of books that you or I have bought recently.

```
book_prices <- c(19.99, 25.78, 5.33, 45.00, 22.45, 21.23)
```

We can find the total amount that I spent using the sum function.

```
sum(book_prices)
```

```
## [1] 139.78
```

(19) Try finding the average price of the books (using `mean()`) and the median price of the books (using `median()`).

```
mean(book_prices)
```

```
## [1] 23.29667
```

```
median(book_prices)
```

```
## [1] 21.84
```

(20) Can you figure out how to find the most expensive book (hint: the book with the maximum price) and the least expensive book (hint: the book with the minimum price)?

```
sorted = sort(book_prices)
print(paste("price of most expensive book: ", sorted[length(sorted)]))
```

```
## [1] "price of most expensive book:  45"
```

```
print(paste("price of least expensive book: ", sorted[1]))
```

```
## [1] "price of least expensive book:  5.33"
```

(21) Hard question: what is happening here?

```
book_prices >= mean(book_prices)
```

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

This operation compares, if the mean of all book prices in the vector is less than or equal to each book price in the vector. 19.99 >= 23.29667 = False, 25.78 >= 23.29667 = True, 5.33 >= 23.29667 = False, 45.00 >= 23.29667 = True , 22.45 >= 23.29667 = False, 21.23 >= 23.29667 = False.

Using the documentation

Let's try a different set of book prices. This time, we have a vector of book prices, but there are some books for which we don't know how much we paid. Those are the `NA` values.

```
more_books <- c(19.99, NA, 25.78, 5.33, NA, 45.00, 22.45, NA, 21.23)
```

(22) How many books did we buy? (Hint: what is the length of the vector.)

```
length(more_books)
```

```
## [1] 9
```

Let's try finding the total using `sum()`.

```
sum(more_books)
```

```
## [1] NA
```

(23) That wasn't very helpful. Why did R give us an answer of `NA`?

NA is a none value. An operation between a numeric value and a none value always returns a none value.

We need to find a way to get the value of the books that we know about. This is an option to the `sum()` function. If you know the name of a function, you can search for it by typing a question mark followed without a space by the name of the function. For example, `?sum`. Look up the `sum()` function's documentation. Read at least the "Arguments" and the "Examples" section.

(24) How can you get the sum for the values which aren't missing?

```
sum(more_books, na.rm=TRUE)
```

```
## [1] 139.78
```

Look up the documentation for `?mean`, `?max`, `?min`.

(25) Use those functions on the vector with missing values.

```
mean(more_books, na.rm=TRUE)
```

```
## [1] 23.29667
```

```
max(more_books, na.rm=TRUE)
```

```
## [1] 45
```

```
min(more_books, na.rm=TRUE)
```

```
## [1] 5.33
```

Data frames and loading packages

We are historians, and we want to work with complex data. Another reason R is awesome is that it includes a kind of data structure called *data frames*. Think of a data frame as basically a spreadsheet. It is tabular data, and the columns can contain any kind of data available in R, such as character vectors, numeric vectors, or logical vectors. R has some sample data built in, but let's use some historical data from the historydata package.

You can load a package like this:

```
library(historydata)
```

(26) The dplyr package is very helpful. Try loading it as well.

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

You might get an error message if you don't have either package installed. If you need to install it, run `install.packages("historydata")` at the R console.

We don't know what is in the historydata package, so let's look at its help. Run this command: `help(package = "historydata")`.

(27) Let's use the data in the `paulist_missions` data frame. According to the package documentation, what is in this data frame?

Data on a roman catholic missionary order called Paulist Fathers

We can print it by using the name of the variable.

```
head(paulist_missions, 10)
```

```
## # A tibble: 10 x 11
##    mission_number church    city  state start_date end_date confessions converts
##             <int> <chr>     <chr> <chr> <chr>      <chr>          <int>    <int>
##  1              1 St. Jose~ New ~ NY    4/6/1851   4/20/18~        6000        0
##  2              2 St. Mich~ Lore~ PA    4/27/1851  5/11/18~        1700        0
##  3              3 St. Mary~ Holl~ PA    5/18/1851  5/28/18~        1000        0
##  4              4 Church o~ John~ PA    5/31/1851  6/8/1851        1000        0
##  5              5 St. Pete~ New ~ NY    9/28/1851  10/12/1~        4000        0
##  6              6 St. Patr~ New ~ NY    10/19/1851 11/3/18~        7000        0
##  7              7 St. Patr~ Erie  PA    11/17/1851 11/28/1~        1000        0
##  8              8 St. Phil~ Cuss~ PA    12/1/1851  12/8/18~         270        0
```

```
##  9              9 St. Vinc~ Youn~ PA    12/10/1851 12/19/1~        1000        0
## 10             10 St. Pete~ Sara~ NY    1/11/1852  1/22/18~         600        3
## # ... with 3 more variables: order <chr>, lat <dbl>, long <dbl>
```

The `head()` function just gives us the first number of items in a vector or data frame.

(28) That showed us some of the data but not all. The `str()` function is helpful. Look up the documentation for it, and then run it on `paulist_missions`.

```
str(paulist_missions)
```

```
## tibble [841 x 11] (S3: tbl_df/tbl/data.frame)
##  $ mission_number: int [1:841] 1 2 3 4 5 6 7 8 9 10 ...
##  $ church        : chr [1:841] "St. Joseph's Church" "St. Michael's Church" "St. Mary's Church" "Chu
##  $ city          : chr [1:841] "New York" "Loretto" "Hollidaysburg" "Johnstown" ...
##  $ state         : chr [1:841] "NY" "PA" "PA" "PA" ...
##  $ start_date    : chr [1:841] "4/6/1851" "4/27/1851" "5/18/1851" "5/31/1851" ...
##  $ end_date      : chr [1:841] "4/20/1851" "5/11/1851" "5/28/1851" "6/8/1851" ...
##  $ confessions   : int [1:841] 6000 1700 1000 1000 4000 7000 1000 270 1000 600 ...
##  $ converts      : int [1:841] 0 0 0 0 0 0 0 0 0 3 ...
##  $ order         : chr [1:841] "Redemptorist" "Redemptorist" "Redemptorist" "Redemptorist" ...
##  $ lat           : num [1:841] 40.7 40.5 40.4 40.3 40.7 ...
##  $ long          : num [1:841] -74 -78.6 -78.4 -78.9 -74 ...
```

(29) Also try the `glimpse()` function.

```
glimpse(paulist_missions)
```

```
## Rows: 841
## Columns: 11
## $ mission_number <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, ~
## $ church         <chr> "St. Joseph's Church", "St. Michael's Church", "St. Mar~
## $ city           <chr> "New York", "Loretto", "Hollidaysburg", "Johnstown", "N~
## $ state          <chr> "NY", "PA", "PA", "PA", "NY", "NY", "PA", "PA", "PA", "~
## $ start_date     <chr> "4/6/1851", "4/27/1851", "5/18/1851", "5/31/1851", "9/2~
## $ end_date       <chr> "4/20/1851", "5/11/1851", "5/28/1851", "6/8/1851", "10/~
## $ confessions    <int> 6000, 1700, 1000, 1000, 4000, 7000, 1000, 270, 1000, 60~
## $ converts       <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0, 0, 0, 0, 3, 0, 3~
## $ order          <chr> "Redemptorist", "Redemptorist", "Redemptorist", "Redemp~
## $ lat            <dbl> 40.71435, 40.50313, 40.42729, 40.32674, 40.71435, 40.71~
## $ long           <dbl> -74.00597, -78.63030, -78.38890, -78.92197, -74.00597, ~
```

(30) Bonus: which package does the `glimpse()` function come from?

With the help(glimpse) function we can find out that the glimpse() function comes from the dplyr package.

We will get into subsetting data in more detail later. But for now, notice that we can get just one of the colums using the `$` operator. For example:

```
head(paulist_missions$city, 20)
```

```
##  [1] "New York"      "Loretto"       "Hollidaysburg" "Johnstown"
##  [5] "New York"      "New York"      "Erie"          "Cussewago"
##  [9] "Youngstown"    "Saratoga"      "Troy"          "Albany"
## [13] "Detroit"       "Philadelphia"  "Philadelphia"  "Cohoes"
## [17] "Wheeling"      "Cincinnati"    "Louisville"    "Albany"
```

(31) Can you print the first 20 numbers of converts? of confessions?

```
head(paulist_missions$converts, 20)
```

```
##  [1] 0 0 0 0 0 0 0 0 0 3 0 0 0 0 0 0 0 3 0 3 3
```

```
head(paulist_missions$confessions, 20)
```

```
##  [1] 6000 1700 1000 1000 4000 7000 1000  270 1000  600 3100 4000 3000 3000 1700
## [16] 1400 1250 5000 2000 4500
```

(32) What was the mean number of converts? the maximum? How about for confessions?

```
mean(paulist_missions$converts)
```

```
## [1] 2.507729
```

```
max(paulist_missions$converts)
```

```
## [1] 50
```

```
mean(paulist_missions$confessions, na.rm=TRUE)
```

```
## [1] 1760.832
```

```
max(paulist_missions$confessions, na.rm=TRUE)
```

```
## [1] 11650
```

(33) Bonus: what was the ratio between confessions and conversions?

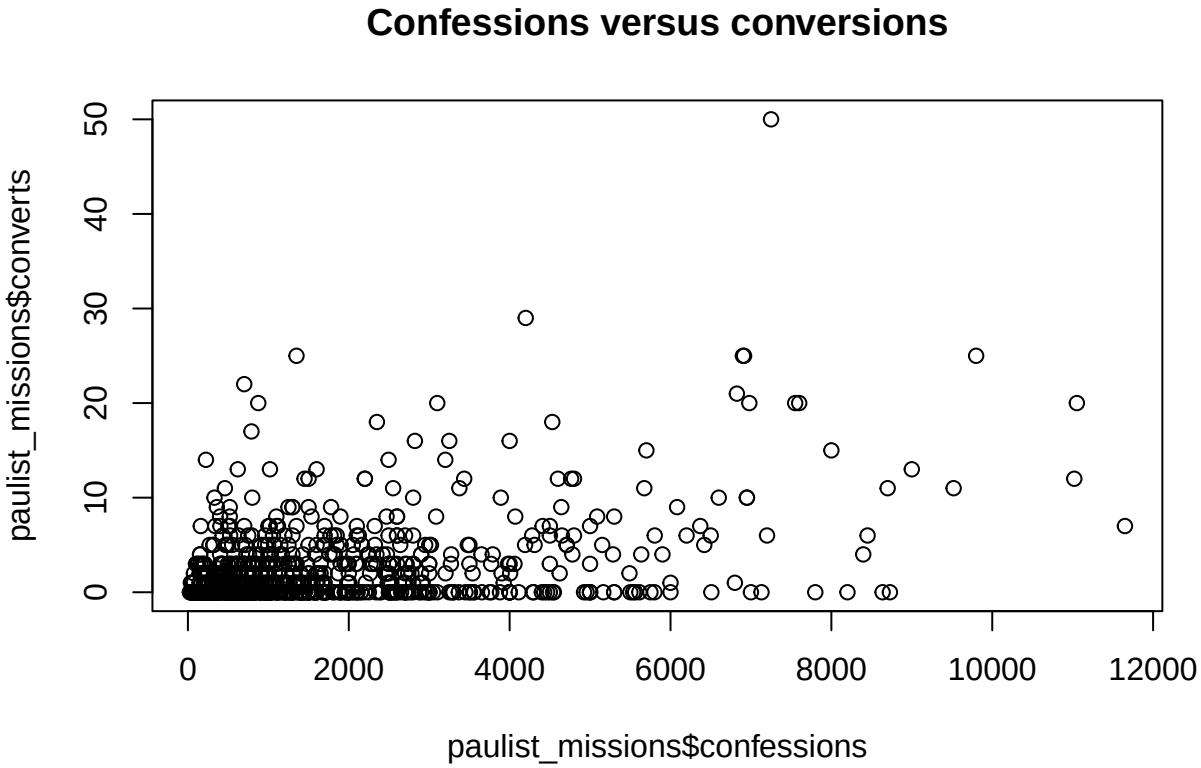
```
max(paulist_missions$converts)/max(paulist_missions$confessions, na.rm=TRUE)*100
```

```
## [1] 0.4291845
```

Plots

And for fun, let's make a scatter plot of the number of confessions versus the number of conversions.

```
plot(paulist_missions$confessions, paulist_missions$converts)
title("Confessions versus conversions")
```

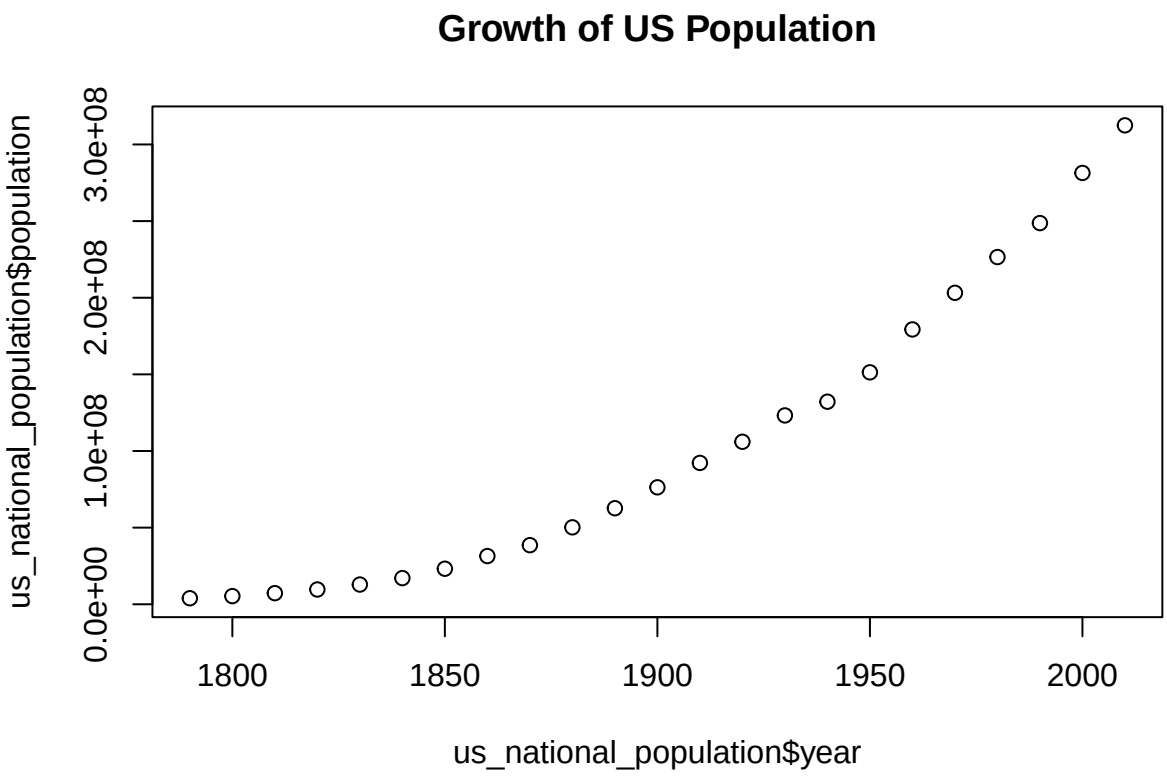


(34) What might you be able to learn from this plot?

I think the most interesting aspect of this plot are the irregularities i.e the dots which are more distant from the majority of dots. Some churches had a high number of confessions (rough indication of people attending the church) but no converts. Others had lesser confessions but converts above average. To be highly hypothetical, it could be an indication of the willingness of people in a certain area to convert or maybe even the openess of the community of a church towards people who are willing to convert. In this context, an interesting part is the line of dots which goes along the x axis. This line demonstrates that a large number of churches were not making converts at all.

(35) There are other datasets in historydata. Can you make a plot from one or more of them?

```
plot(us_national_population$year, us_national_population$population)
title("Growth of US Population")
```


Growth of US Population
us_national_population$population
0.0e+00
1.0e+08
2.0e+08
3.0e+08
1800
1850
1900
1950
2000
us_national_population$year