Getting familiar with R

Bernhard Koschicek

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

The best way to learn R or computational history is to practice. These worksheets contain a series of questions designed to teach you about R or different computational methods. The worksheets are R Markdown documents that include text and code together. The places where you are expected to answer questions are marked like this.

```
(@) Can you make a plot from this dataset?
```

Beneath each question is a space to either create a code block or write an answer.

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:

```
42.4
```

```
## [1] 42.4
```

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.

```r
3+5
```

```
## [1] 8
```

```r
9-31
```

```
## [1] -22
```

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:

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

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

(3) Create a string with your own message.

```r
"Hello World, this is a String"
```

```
## [1] "Hello World, this is a String"
```

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

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

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

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

```r
paste("Hello","World", "Testing")
```

```
## [1] "Hello World Testing"
```

(5) Now explain in a sentence what happened

The strings are pasted together and the paste() function insert a space between the spaces

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

```r
# 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: `==`.)

"x < y" = x smaller then y; "x > y" x greater than y; "x == y" if x the same as y

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

```
5 == 5
```

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

```
"value" > "Values"
```

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

```
"value" == "Value"
```

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

```
414212.2 == 414212.2
```

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

```
2 != 2
```

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

```
c("test", "welt") == c("Test", "welt")
```

```
## [1] FALSE  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?

Yes because a numeric value with a NA value gives you nothing, like 34*0 gives you 0

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?

variable: computer_name and number; <- assignment operator (can be also =); 42 or "HAL 900"

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.

x is asigned 2 and y is asigned 5. The first output is 10 because 5*2 is 10 and the second output is 2 +9 therefore 11. it is because in line 3 we didn't asign the output to a value

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

```
x = 2
c = c("Hello")
```

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

```
y = 40
y+x*y/x-y+3*x-4
```

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

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; it is 20 because the value of be is asigned to a therefore overwrites the first value of a.

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`?

```
5
```

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

```
length(many)
```

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

Let's try multiplying `many` by `2`:

```
many * 2
```

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

(15) What happened?

Because it is a vector, every value is multipled by 2 separatly

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

```
many + 2
```

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

every value gets +2

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

```
chars = c("Hallo", "Welt", "this", "is", "boring", ";)")
```

(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): Länge des längeren Objektes
##       ist kein Vielfaches der Länge des kürzeren Objektes
```

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

because one vector is smaller then the other, therefore the values of the second vetor is applied again and again

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)?

```
max(book_prices)
```

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

```
min(book_prices)
```

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

(21) Hard question: what is happening here?

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

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

> This checks which book is above or equal the mean value and gives TRUE for each which is above or equal

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`?

> because as above, NA destroys numeric values :D the function sum sums up all values and because NA is a missing value, but nonetheless is interpretat as value there can be no numeric sum up

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)
```

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

```
max(more_books)
```

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

```
min(more_books)
```

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

```
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)
```

```
## Warning: package 'historydata' was built under R version 3.5.3
```

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

```
library(dplyr)
```

```
## Warning: package 'dplyr' was built under R version 3.5.3
```

```
##
## 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?

> This package provides sample datasets of interest to historians, analogous to the datasets package for R generally and the histdata package for datasets from the history of statistics.

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
##             <int> <chr>  <chr> <chr> <chr>      <chr>          <int>
##  1              1 St. J~ New ~ NY    4/6/1851   4/20/18~        6000
##  2              2 St. M~ Lore~ PA    4/27/1851  5/11/18~        1700
##  3              3 St. M~ Holl~ PA    5/18/1851  5/28/18~        1000
##  4              4 Churc~ John~ PA    5/31/1851  6/8/1851        1000
##  5              5 St. P~ New ~ NY    9/28/1851  10/12/1~        4000
##  6              6 St. P~ New ~ NY    10/19/1851 11/3/18~        7000
##  7              7 St. P~ Erie  PA    11/17/1851 11/28/1~        1000
##  8              8 St. P~ Cuss~ PA    12/1/1851  12/8/18~         270
##  9              9 St. V~ Youn~ PA    12/10/1851 12/19/1~        1000
## 10             10 St. P~ Sara~ NY    1/11/1852  1/22/18~         600
## # ... with 4 more variables: converts <int>, 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)
```

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

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

```
glimpse(paulist_missions
        )
```

```
## Observations: 841
## Variables: 11
## $ mission_number <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, ...
## $ church         <chr> "St. Joseph's Church", "St. Michael's Church", ...
## $ city           <chr> "New York", "Loretto", "Hollidaysburg", "Johnst...
## $ state          <chr> "NY", "PA", "PA", "PA", "NY", "NY", "PA", "PA",...
## $ start_date     <chr> "4/6/1851", "4/27/1851", "5/18/1851", "5/31/185...
## $ end_date       <chr> "4/20/1851", "5/11/1851", "5/28/1851", "6/8/185...
## $ confessions    <int> 6000, 1700, 1000, 1000, 4000, 7000, 1000, 270, ...
## $ converts       <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0, 0, 0, 0,...
## $ order          <chr> "Redemptorist", "Redemptorist", "Redemptorist",...
## $ lat            <dbl> 40.71435, 40.50313, 40.42729, 40.32674, 40.7143...
## $ long           <dbl> -74.00597, -78.63030, -78.38890, -78.92197, -74...
```

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

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?

```r
conv = head(paulist_missions$converts, 20)
conf = head(paulist_missions$confessions, 20)
```

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

```r
mean(conv)
```

```
## [1] 0.6
```

```r
max(conv)
```

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

```r
mean(conf)
```

```
## [1] 2626
```

```r
max(conf)
```

```
## [1] 7000
```

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

```r
paulist_missions$converts / paulist_missions$confessions
```

```
##   [1] 0.0000000000 0.0000000000 0.0000000000 0.0000000000 0.0000000000
##   [6] 0.0000000000 0.0000000000 0.0000000000 0.0000000000 0.0050000000
##  [11] 0.0000000000 0.0000000000 0.0000000000 0.0000000000 0.0000000000
##  [16] 0.0000000000 0.0024000000 0.0000000000 0.0015000000 0.0006666667
##  [21] 0.0048387097 0.0013043478 0.0011764706 0.0000000000 0.0000000000
##  [26] 0.0000000000 0.0000000000 0.0000000000 0.0000000000 0.0140000000
##  [31] 0.0303030303 0.0040000000 0.0010000000 0.0023076923 0.0015384615
##  [36] 0.0000000000 0.0011764706 0.0013333333 0.0000000000 0.0000000000
##  [41] 0.0000000000 0.0000000000 0.0012000000 0.0000000000 0.0010714286
##  [46] 0.0030000000 0.0000000000 0.0050000000 0.0057142857 0.0000000000
##  [51] 0.0026315789 0.0010000000 0.0000000000 0.0000000000 0.0000000000
##  [56] 0.0003571429 0.0022222222 0.0000000000 0.0173076923 0.0000000000
##  [61] 0.0000000000 0.0100000000 0.0000000000 0.0011111111 0.0012000000
##  [66] 0.0157894737 0.0142857143 0.0058823529 0.0000000000 0.0000000000
##  [71] 0.0064516129 0.0000000000 0.0000000000 0.0000000000 0.0025000000
##  [76] 0.0000000000 0.0000000000 0.0010869565 0.0024390244 0.0014285714
##  [81] 0.0009090909 0.0000000000 0.0000000000 0.0029761905 0.0000000000
##  [86] 0.0037500000 0.0046598322 0.0026315789 0.0000000000 0.0000000000
##  [91] 0.0000000000 0.0046511628 0.0017391304 0.0022222222 0.0054945055
##  [96] 0.0019417476 0.0017543860 0.0017621145 0.0068965517 0.0023076923
## [101] 0.0007100592 0.0015384615 0.0014285714 0.0000000000 0.0000000000
## [106] 0.0024000000 0.0010714286 0.0013215859 0.0000000000 0.0000000000
## [111] 0.0000000000 0.0018939394 0.0026315789 0.0000000000 0.0010000000
## [116] 0.0009090909 0.0014388489 0.0000000000 0.0039920160 0.0000000000
```

```
## [121] 0.0000000000 0.0015151515 0.0027027027 0.0010526316 0.0000000000
## [126] 0.0027027027 0.0000000000 0.0008695652 0.0020000000 0.0001470588
## [131] 0.0000000000 0.0000000000 0.0008000000 0.0022222222 0.0000000000
## [136] 0.0007097232 0.0000000000 0.0033333333 0.0000000000 0.0000000000
## [141] 0.0011538462 0.0005000000 0.0014084507 0.0030769231 0.0012000000
## [146] 0.0000000000 0.0013333333 0.0007142857 0.0000000000 0.0035714286
## [151] 0.0009090909 0.0000000000 0.0018518519 0.0013333333 0.0054794521
## [156] 0.0037500000 0.0008333333 0.0042857143 0.0014285714 0.0043120345
## [161] 0.0000000000 0.0006250000 0.0008333333 0.0081250000 0.0000000000
## [166] 0.0000000000 0.0000000000 0.0000000000 0.0000000000 0.0000000000
## [171] 0.0000000000 0.0000000000 0.0060000000 0.0013333333 0.0000000000
## [176] 0.0012500000 0.0015151515 0.0007194245 0.0006666667 0.0000000000
## [181] 0.0000000000 0.0014285714 0.0000000000 0.0000000000 0.0015094340
## [186] 0.0014285714 0.0000000000 0.0314285714 0.0015555556 0.0034188034
## [191] 0.0100000000 0.0076595745 0.0035087719 0.0007142857 0.0082758621
## [196] 0.0200000000 0.0012121212 0.0042857143 0.0014018692 0.0133333333
## [201] 0.0000000000 0.0012500000 0.0080000000 0.0023076923 0.0022222222
## [206] 0.0030769231 0.0012500000 0.0032432432 0.0021428571 0.0051851852
## [211] 0.0005882353 0.0014444444 0.0012500000 0.0250000000 0.0031250000
## [216] 0.0000000000 0.0030769231 0.0090909091 0.0057142857 0.0025000000
## [221] 0.0015151515 0.0133333333 0.0270270270 0.0008333333 0.0028571429
## [226] 0.0050000000 0.0054545455 0.0046153846 0.0125000000 0.0069047619
## [231] 0.0025000000 0.0043750000 0.0007500000 0.0014000000 0.0000000000
## [236] 0.0215189873 0.0068181818 0.0209677419 0.0062500000 0.0153846154
## [241] 0.0052049447 0.0031948882 0.0239130435 0.0042857143 0.0072289157
## [246] 0.0038834951 0.0038461538 0.0027272727 0.0009523810 0.0000000000
## [251] 0.0045146727 0.0021276596 0.0027027027 0.0075000000 0.0026666667
## [256] 0.0093023256 0.0080000000 0.0156250000 0.0062500000 0.0008571429
## [261] 0.0033333333 0.0000000000 0.0018750000 0.0036231884 0.0015789474
## [266] 0.0000000000 0.0106837607 0.0041176471 0.0000000000 0.0025923526
## [271] 0.0000000000 0.0010554090 0.0022222222 0.0266666667 0.0000000000
## [276] 0.0205882353 0.0043290043 0.0158730159 0.0020000000 0.0009677419
## [281] 0.0056710775 0.0000000000 0.0019386676 0.0007558579 0.0036363636
## [286] 0.0016339869 0.0072727273 0.0015503876 0.0011834320 0.0030146425
## [291] 0.0070754717 0.0019607843 0.0000000000 0.0017241379 0.0026315789
## [296] 0.0000000000 0.0000000000 0.0000000000 0.0000000000 0.0011627907
## [301] 0.0022338049 0.0000000000 0.0000000000 0.0043516101 0.0000000000
## [306] 0.0000000000 0.0000000000 0.0000000000 0.0042194093 0.0023116043
## [311] 0.0000000000 0.0021905805 0.0000000000 0.0030487805 0.0000000000
## [316] 0.0000000000 0.0000000000 0.0000000000 0.0000000000 0.0017064846
## [321] 0.0000000000 0.0027027027 0.0000000000 0.0000000000 0.0088888889
## [326] 0.0437500000 0.0000000000 0.0050561798 0.0030000000 0.0000000000
## [331] 0.0133843212 0.0075566751 0.0000000000 0.0039735099 0.0021505376
## [336] 0.0185185185 0.0051282051 0.0024390244 0.0102564103 0.0007547170
## [341] 0.0022271715 0.0006666667 0.0000000000 0.0000000000 0.0000000000
## [346] 0.0266666667 0.0000000000 0.0056818182 0.0022222222 0.0000000000
## [351] 0.0000000000 0.0000000000 0.0000000000 0.0028436019 0.0000000000
## [356] 0.0000000000 0.0185185185 0.0023255814 0.0029498525 0.0000000000
## [361] 0.0002547122 0.0000000000 0.0000000000 0.0000000000 0.0026178010
## [366] 0.0053333333 0.0000000000 0.0004000000 0.0006451613 0.0013888889
## [371] 0.0016129032 0.0026086957 0.0042735043 0.0173267327 0.0000000000
## [376] 0.0000000000 0.0000000000 0.0000000000 0.0000000000 0.0000000000
## [381] 0.0004327131 0.0024390244 0.0011600928 0.0010989011 0.0009174312
## [386] 0.0000000000 0.0049230769 0.0060362173 0.0045248869 0.0000000000
```

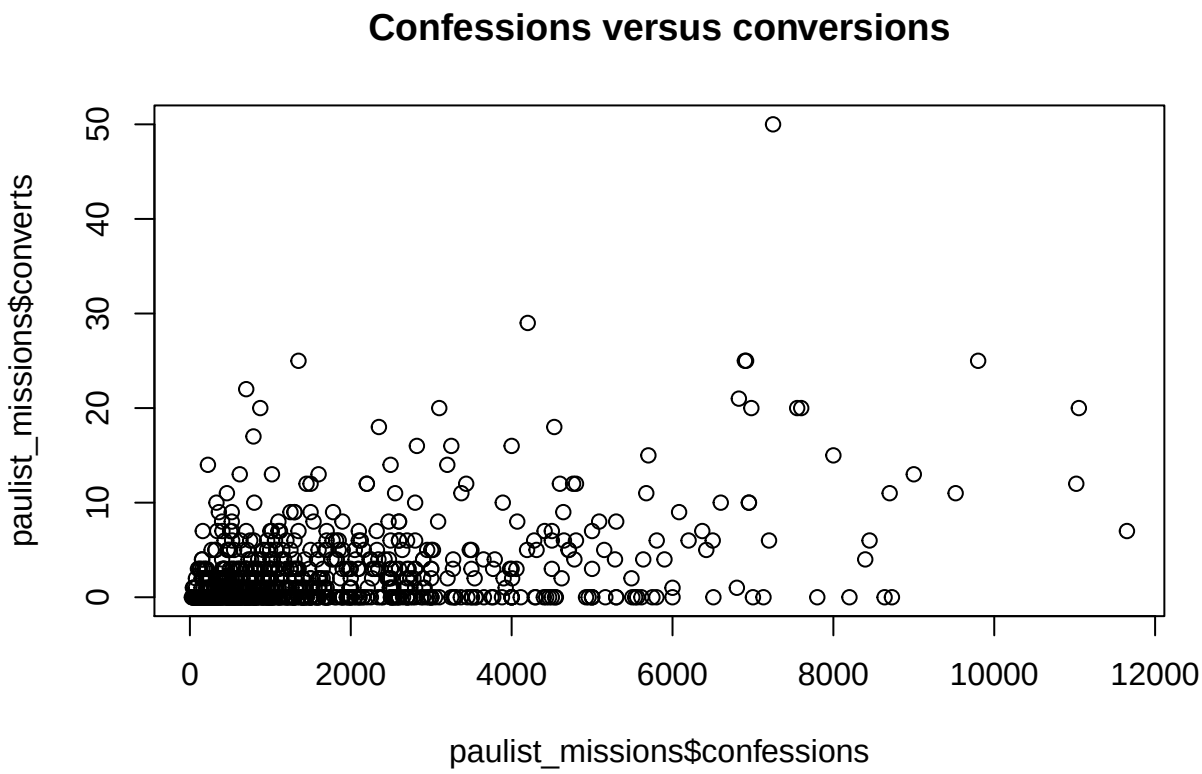
```
## [391] 0.0000000000 0.0033333333 0.0000000000 0.0000000000 0.0007407407
## [396] 0.0006000000 0.0013793103 0.0200000000 0.0000000000 0.0029411765
## [401] 0.0300000000           NA 0.0000000000 0.0000000000           NA
## [406] 0.0138888889 0.0004524887 0.0007299270 0.0024096386 0.0003427005
## [411] 0.0000000000 0.0003703704 0.0030769231 0.0003642987 0.0014797764
## [416] 0.0010309278 0.0000000000 0.0000000000 0.0009852217 0.0000000000
## [421] 0.0117647059 0.0042553191 0.0000000000 0.0000000000 0.0000000000
## [426] 0.0000000000 0.0000000000 0.0000000000 0.0000000000 0.0004000000
## [431] 0.0006250000 0.0000000000 0.0000000000 0.0000000000 0.0000000000
## [436] 0.0000000000 0.0007342144 0.0000000000 0.0053285968 0.0049261084
## [441] 0.0025188917 0.0015873016 0.0000000000 0.0008064516 0.0000000000
## [446] 0.0019388195 0.0010622477 0.0035294118 0.0000000000 0.0047058824
## [451] 0.0000000000 0.0000000000 0.0031250000 0.0032000000 0.0027027027
## [456] 0.0000000000 0.0074626866 0.0261437908 0.0000000000 0.0080000000
## [461] 0.0026315789 0.0010000000 0.0000000000 0.0000000000 0.0000000000
## [466] 0.0016161616 0.0000000000 0.0000000000 0.0000000000 0.0010889292
## [471] 0.0000000000 0.0000000000 0.0000000000 0.0014388489 0.0000000000
## [476] 0.0095238095 0.0000000000 0.0063636364 0.0050150451 0.0000000000
## [481] 0.0005966587 0.0004024145 0.0000000000 0.0000000000 0.0000000000
## [486] 0.0000000000 0.0000000000 0.0000000000 0.0000000000 0.0000000000
## [491] 0.0017897092 0.0000000000 0.0000000000 0.0000000000 0.0000000000
## [496] 0.0005128205 0.0000000000 0.0000000000 0.0070000000 0.0000000000
## [501] 0.0012643678 0.0033333333 0.0018096272 0.0033821871 0.0000000000
## [506] 0.0000000000 0.0006779661 0.0044444444 0.0000000000 0.0000000000
## [511] 0.0000000000 0.0000000000 0.0022727273 0.0000000000 0.0000000000
## [516] 0.0000000000 0.0000000000 0.0000000000 0.0000000000 0.0000000000
## [521] 0.0008830022 0.0000000000 0.0007407407 0.0006244146 0.0035087719
## [526] 0.0021739130 0.0000000000 0.0000000000 0.0018170806 0.0100000000
## [531] 0.0000000000 0.0000000000 0.0000000000 0.0024038462 0.0000000000
## [536] 0.0000000000 0.0040404040 0.0000000000 0.0033003300 0.0228571429
## [541] 0.0000000000 0.0068965517 0.0000000000 0.0056697378 0.0079051383
## [546] 0.0250000000 0.0023255814 0.0028169014 0.0000000000 0.0036153290
## [551]           NA 0.0016528926 0.0010695187 0.0000000000 0.0000000000
## [556] 0.0000000000 0.0012048193 0.0000000000 0.0232558140 0.0000000000
## [561] 0.0000000000 0.0000000000           NA 0.0000000000 0.0027272727
## [566] 0.0006250000 0.0000000000 0.0018181818 0.0022471910 0.0023255814
## [571] 0.0000000000 0.0000000000 0.0026143791 0.0009345794 0.0000000000
## [576] 0.0020242915 0.0000000000 0.0036363636 0.0007610350 0.0039024390
## [581] 0.0000000000 0.0033333333 0.0000000000 0.0000000000 0.0022641509
## [586] 0.0016545334 0.0000000000 0.0000000000 0.0000000000 0.0004764173
## [591] 0.0022727273 0.0034934498 0.0003734130 0.0000000000 0.0000000000
## [596] 0.0000000000 0.0046511628 0.0000000000 0.0000000000 0.0000000000
## [601] 0.0000000000 0.0005037783 0.0000000000 0.0014577259 0.0000000000
## [606] 0.0000000000 0.0000000000 0.0166666667 0.0000000000 0.0000000000
## [611] 0.0127450980 0.0008116883 0.0000000000 0.0020920502 0.0000000000
## [616] 0.0000000000 0.0000000000 0.0032388664 0.0069930070 0.0034883721
## [621] 0.0020408163 0.0071428571 0.0025706941 0.0000000000 0.0024038462
## [626] 0.0234375000 0.0121212121 0.0000000000 0.0019656020 0.0000000000
## [631] 0.0010611205 0.0000000000 0.0010337698 0.0003724395 0.0024000000
## [636] 0.0000000000 0.0012221204 0.0000000000 0.0000000000 0.0013793103
## [641] 0.0000000000 0.0000000000 0.0105263158 0.0000000000 0.0000000000
## [646] 0.0046189376 0.0028653295 0.0023980815 0.0000000000           NA
## [651] 0.0001666667 0.0000000000 0.0000000000 0.0000000000 0.0000000000
## [656] 0.0004992511 0.0000000000 0.0000000000 0.0000000000 0.0000000000
```

```
## [661] 0.0000000000 0.0004990020 0.0006006006 0.0000000000 0.0000000000
## [666] 0.0028235294 0.0000000000 0.0114285714 0.0000000000 0.0000000000
## [671] 0.0040000000 0.0011029412 0.0000000000 0.0000000000 0.0011927481
## [676] 0.0056074766 0.0008097166 0.0000000000 0.0032258065 0.0006896552
## [681] 0.0012804097 0.0011554622 0.0000000000 0.0000000000 0.0000000000
## [686] 0.0007788162 0.0000000000 0.0000000000 0.0000000000 0.0009706853
## [691] 0.0000000000 0.0015873016 0.0007077141 0.0000000000 0.0000000000
## [696] 0.0000000000 0.0000000000 0.0000000000 0.0000000000 0.0000000000
## [701] 0.0000000000 0.0017605634 0.0025510204 0.0000000000 0.0005649718
## [706] 0.0072000000 0.0000000000 0.0000000000 0.0008368201 0.0006944444
## [711] 0.0000000000 0.0000000000 0.0000000000 0.0000000000 0.0056112224
## [716] 0.0024048096 0.0006008584 0.0015723270 0.0000000000 0.0008097166
## [721] 0.0000000000 0.0000000000 0.0000000000 0.0003442341 0.0010958904
## [726] 0.0008153282 0.0000000000 0.0030769231 0.0000000000 0.0000000000
## [731] 0.0000000000 0.0012330456 0.0014285714 0.0000000000 0.0000000000
## [736] 0.0000000000 0.0000000000 0.0000000000 0.0037735849 0.0000000000
## [741] 0.0000000000 0.0000000000 0.0037735849           NA 0.0000000000
## [746] 0.0000000000 0.0000000000 0.0000000000 0.0000000000 0.0000000000
## [751] 0.0014367816 0.0000000000 0.0000000000 0.0000000000 0.0040000000
## [756] 0.0000000000 0.0000000000 0.0007957560 0.0068762279 0.0046875000
## [761] 0.0000000000 0.0069230769 0.0032592593 0.0000000000 0.0009230769
## [766] 0.0007537688 0.0000000000 0.0000000000 0.0052083333 0.0027272727
## [771] 0.0000000000 0.0008333333 0.0012903226 0.0000000000 0.0048780488
## [776] 0.0000000000 0.0027972028 0.0000000000 0.0000000000 0.0000000000
## [781] 0.0000000000 0.0193548387 0.0009302326 0.0025000000 0.0025316456
## [786] 0.0000000000 0.0000000000 0.0016666667 0.0054545455 0.0000000000
## [791] 0.0000000000 0.0000000000 0.0625000000 0.0000000000 0.0000000000
## [796] 0.0000000000 0.0000000000 0.0000000000 0.0000000000 0.0006211180
## [801] 0.0000000000 0.0000000000 0.0003952569 0.0000000000 0.0000000000
## [806] 0.0000000000 0.0069230769 0.0000000000 0.0000000000 0.0000000000
## [811] 0.0019860973 0.0000000000 0.0000000000 0.0023076923 0.0000000000
## [816] 0.0016949153 0.0000000000 0.0007568590 0.0016666667 0.0026490066
## [821] 0.0000000000 0.0000000000 0.0025000000 0.0014374701 0.0052631579
## [826] 0.0011111111 0.0042857143 0.0007389163 0.0017777778 0.0000000000
## [831] 0.0000000000 0.0000000000 0.0000000000 0.0000000000 0.0000000000
## [836] 0.0029673591 0.0000000000 0.0000000000 0.0000000000 0.0046082949
## [841] 0.0136986301
```

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")
```

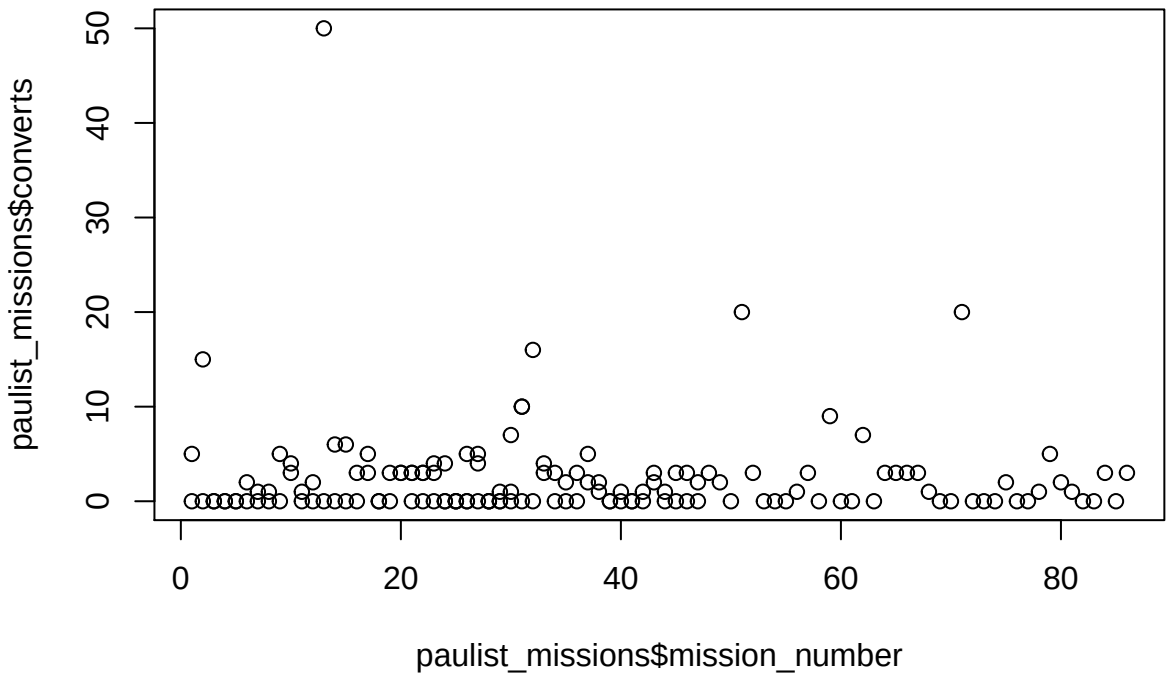


(34) What might you be able to learn from this plot?

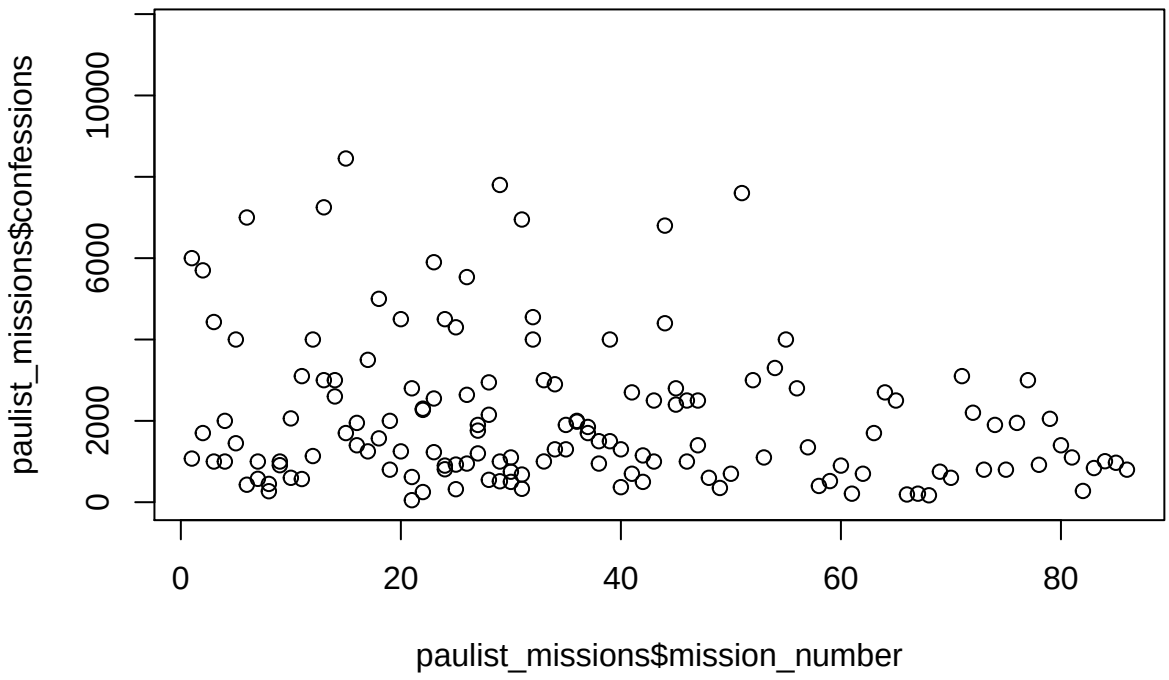
That there were not so many confestions and converts

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

```
plot(paulist_missions$mission_number, paulist_missions$converts)
```




```
plot(paulist_missions$mission_number, paulist_missions$confessions)
```




```
#plot(paulist_missions$order, paulist_missions$converts) <= doesn't work cause "Error in plot.window(..
```