Basic Computing 2 – Packages, Functions, Documenting code

Stefano Allesina

Basic Computing 2

- **Goal:** Show how to install, load and use the many freely available `R` packages. Illustrate how to write user-defined functions and how to organize code. Showcase basic plotting functions. Introduce the package `knitr` for writing beautiful reports.
- **Audience:** Biologists with basic knowledge of `R`.
- **Installation:** To produce well-documented code, we need to instal the package `knitr`. We will also use the package `MASS` for statistics.

Packages

`R` is the most popular statistical computing software among biologists due to its highly specialized packages, often written from biologists for biologists. You can contribute a package too! The `RStudio` support (`goo.gl/harVqF`) provides guidance on how to start developing `R` packages and Hadley Wickham's free online book (`r-pkgs.had.co.nz`) will make you a pro.

You can find highly specialized packages to address your research questions. Here are some suggestions for finding an appropriate package. The Comprehensive R Archive Network (CRAN) offers several ways to find specific packages for your task. You can either browse packages (`goo.gl/7oVyKC`) and their short description or select a scientific field of interest (`goo.gl/0WdIcu`) to browse through a compilation of packages related to each discipline.

From within your `R` terminal or `RStudio` you can also call the function `RSiteSearch("KEYWORD")`, which submits a search query to the website `search.r-project.org`. The website `rseek.org` casts an even wider net, as it not only includes package names and their documentation but also blogs and mailing lists related to `R`. If your research interests relate to high-throughput genomic data, you should have a look the packages provided by Bioconductor (`goo.gl/7dwQlq`).

Installing a package

To install a package type

```
install.packages("name_of_package")
```

in the Console, or choose the panel **Packages** and then click on *Install* in `RStudio`.

Loading a package

To load a package type

```
library(name_of_package)
```

or call the command into your script. If you want your script to automatically install a package in case it's missing, use this boilerplate:

```
if (!require(needed_package, character.only = TRUE, quietly = TRUE)) {
    install.packages(needed_package)
    library(needed_package, character.only = TRUE)
}
```

Example

For example, say we want to access the dataset `bacteria`, which reports the incidence of *H. influenzae* in Australian children. The dataset is contained in the package `MASS`.

First, we need to load the package:

```
library(MASS)
```

Now we can load the data:

```
data(bacteria)
bacteria[1:3,]
```

```
##   y ap hilo week  ID     trt
## 1 y  p   hi    0 X01 placebo
## 2 y  p   hi    2 X01 placebo
## 3 y  p   hi    4 X01 placebo
```

Do shorter titles lead to more citations?

To keep learning about `R`, we study a simple problem: do papers with shorter titles have more citations? This is what claimed by Letchford *et al.*, who in 2015 analyzed 140,000 papers (dx.doi.org/10.1098/rsos.150266) finding that shorter titles correlated with a larger number of citations.

In the `data/citations` folder, you find information on all the articles published between 2004 and 2013 by three top disciplinary journals (*Nature Neuroscience*, *Nature Genetics*, and *Ecology Letters*), which we are going to use to explore the robustness of these findings.

We start by reading the data in. This is a simple `csv` file, so that we can use

```
papers <- read.csv("../data/citations/nature_neuroscience.csv")
```

to load the data. However, running `str(papers)` shows that all the columns containing text have been automatically converted to `Factor` (categorical values, which is good when performing regressions). Because we want to manipulate these columns (for example, count how many characters are in a title), we want to avoid this automatic behavior. We can accomplish this by calling the function `read.csv` with an extra argument:

```
papers <- read.csv("../data/citations/nature_neuroscience.csv", stringsAsFactors = FALSE)
```

Next, we want to take a peek at the data. How large is it?

```
dim(papers)
```

```
## [1] 2000    7
```

Let's see the first few rows:

```
head(papers, 3)
```

```
##             Authors                                   Title Year
## 1 Logothetis, N.K.               Francis crick 1916-2004. 2004
## 2       Narain, C. Object-specific unconscious processing. 2005
## 3       Narain, C.                       Going down BOLDly. 2006
##            Source.title Page.start Page.end Cited.by
## 1  Nature neuroscience       1027     1028        0
## 2 Nature neuroscience.       1288       NA        0
## 3  Nature neuroscience        474       NA        0
```

Now, we want to test whether papers with longer titles do accrue fewer (or more) citations than those with shorter titles. The first step is therefore to add another column to the data, containing the length of the title for each paper:

```
papers$TitleLength <- nchar(papers$Title)
```

Basic statistics in R

In the original paper, Letchford *et al.* used rank-correlation: rank all the papers according to their title length and the number of citations. If the Kendall's Tau (rank correlation) is positive, then longer titles are associated with more citations; if Tau is negative, longer titles are associated with fewer citations. In R you can compute rank correlation using:

```
kendall_cor <- cor(papers$TitleLength, papers$Cited.by, method = "kendall")
kendall_cor
```

```
## [1] 0.04528715
```

To perform a significance test, use

```
cor.test(papers$TitleLength, papers$Cited.by, method = "kendall")
```

```
##
##  Kendall's rank correlation tau
##
## data:  papers$TitleLength and papers$Cited.by
## z = 3.0023, p-value = 0.002679
## alternative hypothesis: true tau is not equal to 0
## sample estimates:
##        tau
## 0.04528715
```

showing that the correlation between the ranks is positive and significant. We have found the opposite effect than Letchford *et al.*—longer titles are associated with **more** citations!

Now we are going to examine the data in a different way, to test whether these results are robust.

Basic plotting in R

To plot the title length vs. number of citations, we need to learn about plotting in R. To produce a simple scatterplot using the base functions for plotting, simply type:

```
plot(papers$TitleLength, papers$Cited.by)
```

It is hard to detect any trend in this plot, as there are a few papers with many more citations than the rest. We can transform the data by plotting on the y-axis the log10 of citations + 1 (so that papers with zero citations do not cause errors):

```
plot(papers$TitleLength, log10(papers$Cited.by + 1))
```

Again, it's hard to see any trend in here. Maybe we should plot the best fitting line and overlay it on top of the graph. To do so, we first need to learn about regressions in R.

Regressions in R

R was born for statistics — the fact that it's very easy to fit a linear regression is not surprising! To build a linear model, simply write

```
# model y = a + bx + error
my_model <- lm(y ~ x)
```

Because it's more convenient to call the code in this way, let's add a new column to the data frame with the log of citations:

```
papers$LogCits <- log10(papers$Cited.by + 1)
```

And perform a linear regression:

```
model_cits <- lm(papers$LogCits ~ papers$TitleLength)
# This is the best fitting line
model_cits
```

```
##
## Call:
## lm(formula = papers$LogCits ~ papers$TitleLength)
##
## Coefficients:
##        (Intercept)  papers$TitleLength
##           1.358195            0.006193
```

```
# This is a summary of all the statistics
summary(model_cits)
```

```
##
## Call:
## lm(formula = papers$LogCits ~ papers$TitleLength)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.68022 -0.25261  0.02803  0.29188  1.82838
##
## Coefficients:
##                     Estimate Std. Error t value Pr(>|t|)
## (Intercept)        1.3581953  0.0445997   30.45   <2e-16 ***
## papers$TitleLength 0.0061927  0.0005339   11.60   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4522 on 1998 degrees of freedom
## Multiple R-squared:  0.06309,    Adjusted R-squared:  0.06263
## F-statistic: 134.6 on 1 and 1998 DF,  p-value: < 2.2e-16
```

And plotting

```
# plot the points
plot(papers$TitleLength, log10(papers$Cited.by + 1))
# add the best fitting line
abline(model_cits, col = "red")
```

Again, we find a positive trend. One thing to consider, is that in the database we have papers spanning a decade. Naturally, older papers have had more time to accrue citations. In our models, we would like to control for this effect. First, let's plot the distribution of citations for a few years. To produce an histogram in R, use

```
hist(papers$LogCits)
# increase the number of breaks
hist(papers$LogCits, breaks = 15)
```

Alternatively, estimate the density using

```
plot(density(papers$LogCits))
```

Let's plot the density for years 2004, 2009, 2013:

```
# plot the density for the older papers:
plot(density(papers$LogCits[papers$Year == 2004]))
lines(density(papers$LogCits[papers$Year == 2009]), col = "red")
lines(density(papers$LogCits[papers$Year == 2013]), col = "blue")
```

As expected, younger papers have fewer citations. We can account for this fact in our regression model:

```
model_year_length <- lm(papers$LogCits ~ as.factor(papers$Year) + papers$TitleLength)
summary(model_year_length)
```

```
##
## Call:
## lm(formula = papers$LogCits ~ as.factor(papers$Year) + papers$TitleLength)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.88880 -0.21178  0.01232  0.25186  1.51362
##
## Coefficients:
##                               Estimate Std. Error t value Pr(>|t|)
## (Intercept)                  1.6953101  0.0499837  33.917  < 2e-16 ***
## as.factor(papers$Year)2005 -0.0277435  0.0424764  -0.653 0.513735
## as.factor(papers$Year)2006 -0.1137399  0.0431189  -2.638 0.008409 **
## as.factor(papers$Year)2007 -0.1603601  0.0435023  -3.686 0.000234 ***
## as.factor(papers$Year)2008 -0.1630806  0.0442859  -3.682 0.000237 ***
## as.factor(papers$Year)2009 -0.2373747  0.0430316  -5.516 3.91e-08 ***
## as.factor(papers$Year)2010 -0.2824792  0.0433906  -6.510 9.48e-11 ***
## as.factor(papers$Year)2011 -0.4927523  0.0425956 -11.568  < 2e-16 ***
## as.factor(papers$Year)2012 -0.6278381  0.0423156 -14.837  < 2e-16 ***
## as.factor(papers$Year)2013 -0.6539580  0.0419966 -15.572  < 2e-16 ***
## papers$TitleLength           0.0056728  0.0004645  12.213  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3912 on 1989 degrees of freedom
## Multiple R-squared:  0.302,  Adjusted R-squared:  0.2985
## F-statistic: 86.06 on 10 and 1989 DF,  p-value: < 2.2e-16
```

Using `as.factor(papers$Year)` we have fitted a model in which each year has a different baseline, and the title length influences this baseline. Again, we find that longer titles are associated with more citations.

Random numbers

As a reminder, the Kendall's τ takes as input two rankings x and y, both of length n. It calculates the number of "concordant pairs", in which if $x_i > x_j$ then $y_i > y_j$ and "discordant pairs". Then,

$$\tau = \frac{\text{num. concordant} - \text{num. discordant}}{\frac{n(n-1)}{2}}$$

If x and y were completely independent, we would expect τ to be distributed with a mean of 0. The variance of the null distribution of τ (and hence the p-value calculated above) depends on the data, and is typically approximated as a normal distribution. If you want to have a stronger result, you can use randomizations to approximate the p-value. Simply, compute τ for the actual data, and for many "fake" datasets obtained randomizing the data. Your p-value is well approximated by the proportion of τ values for the randomized sets that exceed the τ value for the actual data.

To perform this randomization, or any simulation, we typically need to draw random numbers. `R` has functions to sample random numbers from very many different statistical distributions. For example:

```r
runif(5) # sample 5 numbers from the uniform distribution between 0 and 1
```

```
## [1] 0.49624208 0.81103292 0.51494268 0.09401232 0.29883759
```

```r
runif(5, min = 1, max = 9) # set the limits of the uniform distribution
```

```
## [1] 7.164843 2.158266 2.507597 4.905866 7.580374
```

```r
rnorm(3) # three values from standard normal
```

```
## [1] 0.3353088 0.8581134 0.7317063
```

```r
rnorm(3, mean = 5, sd = 4) # specify mean and standard deviation
```

```
## [1] 9.590280 4.767659 2.568968
```

To sample from a set of values, use `sample`:

```r
v <- c("a", "b", "c", "d")
sample(v, 2) # without replacement
```

```
## [1] "d" "a"
```

```r
sample(v, 6, replace = TRUE) # with replacement
```

```
## [1] "b" "b" "c" "d" "b" "b"
```

```r
sample(v) # simply shuffle the elements
```

```
## [1] "c" "a" "d" "b"
```

Let's try to write a randomization to calculate p-value associated with the τ observed for year 2006.

```r
# first, we subset the data
papers_year <- papers[papers$Year == 2006, ] # get all rows matching the year
# compute original tau
tau_original <- cor(papers_year$TitleLength, papers_year$Cited.by, method = "kendall")
tau_original
```

```
## [1] 0.1140872
```

Now we want to calculate it on the “fake” data sets. To have confidence in the first two decimal digits, we should perform about ten thousand randomizations. This and similar randomization techniques are known as “bootstrapping”.

```
num_randomizations <- 10000
pvalue <- 0 # initialize at 0
for (i in 1:num_randomizations){
  # calculate cor on shuffled data
  tau_shuffle <- cor(papers_year$TitleLength,
                     sample(papers_year$Cited.by), # scramble the citations at random
                     method = "kendall")
  if (tau_shuffle >= tau_original){
    pvalue <- pvalue + 1
  }
}
# calculate proportion
pvalue <- pvalue / num_randomizations
pvalue
```

```
## [1] 0.0084
```

Note that the p-value is different (and in fact smaller) than that calculated using the normal approximation:

```
cor.test(papers_year$TitleLength, papers_year$Cited.by, method = "kendall")
```

```
##
##  Kendall's rank correlation tau
##
## data:  papers_year$TitleLength and papers_year$Cited.by
## z = 2.3902, p-value = 0.01684
## alternative hypothesis: true tau is not equal to 0
## sample estimates:
##       tau
## 0.1140872
```

Whenever possible, use randomizations, rather than relying on classical tests. They are more computationally expensive, but they allow you to avoid making assumptions about your data.

Writing functions

We have written code that analyzes one year of data. If we wanted to repeat the analysis on a different year, we would have to modify the code slightly. Instead of doing that, we can write a function that allows us to select a given year, and randomize the data. To do so, we need to learn about functions.

The R community provides about 7,000 packages. Still, sometimes there isn’t an already made function capable of doing what you need. In these cases, you can write your own functions. In fact, it is in general a good idea to always divide your analysis into functions, and then write a small “master” program that calls the functions and performs the analysis. In this way, the code will be much more legible, and you will be able to recycle the functions for your other projects.

A function in R has this form:

```
my_function_name <- function(arguments of the function){
  # Body of the function
  # ...
  #
  return(return_value) # this is optional
}
```

A few examples:

```
sum_two_numbers <- function(a, b){
  apb <- a + b
  return(apb)
}
sum_two_numbers(5, 7.2)
```

```
## [1] 12.2
```

You can set a default value for some of the arguments: if not specified by the user, the function will use these defaults:

```
sum_two_numbers <- function(a = 1, b = 2){
  apb <- a + b
  return(apb)
}
sum_two_numbers()
```

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

```
sum_two_numbers(3)
```

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

```
sum_two_numbers(b = 9)
```

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

The return value is optional:

```
my_factorial <- function(a = 6){
  if ( (as.integer(a) != a) | (a < 0) ) {
    print("Please enter a positive integer!")
  } else {
    tmp <- 1
    for (i in 1:a){
      tmp <- tmp * i
    }
    print(paste(a, "! = ", tmp, sep = ""))
  }
}
my_factorial()
```

```
## [1] "6! = 720"
```

```
my_factorial(10)
```

```
## [1] "10! = 3628800"
```

You can return **only one** object. If you need to return multiple values, organize them into a vector/matrix/list and return that.

```
order_two_numbers <- function(a, b){
  if (a > b) return(c(a, b))
  return(c(b,a))
}
order_two_numbers(runif(1), runif(1))
```

```
## [1] 0.57797384 0.03461971
```

Having learned a little about functions, we want to write one that takes as input a vector of citations, a vector of title lengths, and a number of randomizations to perform. The function returns the value of τ as well as the associated p-value. In R, we can write:

```
tau_citations_titlelength <- function(citations, titlelength, num_randomizations = 1000){
  tau_original <- cor(titlelength, citations, method = "kendall")
  pvalue <- 0 # initialize at 0
  for (i in 1:num_randomizations){
    # calculate cor on shuffled data
    tau_shuffle <- cor(titlelength,
                       sample(citations), # scramble the citations at random
                       method = "kendall")
    if (tau_shuffle >= tau_original){
      pvalue <- pvalue + 1
    }
  }
  # calculate proportion
  pvalue <- pvalue / num_randomizations
  # return a list
  return(list(tau = tau_original,
              pvalue = pvalue))
}
```

We can write a loop that calls in turn the function for each year separately:

```
all_years <- sort(unique(papers$Year))
for (my_year in all_years){
  tmp <- tau_citations_titlelength(papers$Cited.by[papers$Year == my_year],
                                   papers$TitleLength[papers$Year == my_year],
                                   1000)
  print(paste(my_year, "-> Tau:", round(tmp$tau, 3), "pvalue:", tmp$pvalue))
}
```

```
## [1] "2004 -> Tau: 0.006 pvalue: 0.45"
## [1] "2005 -> Tau: 0.054 pvalue: 0.136"
## [1] "2006 -> Tau: 0.114 pvalue: 0.016"
## [1] "2007 -> Tau: 0.01 pvalue: 0.383"
## [1] "2008 -> Tau: 0.065 pvalue: 0.107"
## [1] "2009 -> Tau: 0.012 pvalue: 0.408"
## [1] "2010 -> Tau: -0.052 pvalue: 0.846"
## [1] "2011 -> Tau: 0.145 pvalue: 0.001"
## [1] "2012 -> Tau: 0.114 pvalue: 0.006"
## [1] "2013 -> Tau: 0.045 pvalue: 0.167"
```

Organizing and running code

Now we would like to be able to automate the analysis, such that we can repeat it for each journal. This is a good place to pause and introduce how to go about writing programs that are well-organized, easy to write, and easy to debug.

1. Take the problem, and divide it into its basic building blocks

2. Write the code for each building block separately, and test it thoroughly.
3. Extensively document the code, so that you can understand what you did, how you did it, and why.
4. Combine the building blocks into a master program.

For example, let's say we want to write a program that takes as input the name of a file containing the data on titles, years and citations for a given journal. The program should first run the linear model:

`log(citations + 1) ~ Year (categorical) + TitleLength`

And output the coefficient associated with `TitleLength` as well as its p-value.

Then, the program should run the Kendall's test for each year separately, again outputting τ and the p-value obtained with the normal approximation for each year.

Dividing it into blocks, we need to wite:

- code to load the data, calculate title lengths and log citations
- a function to perform the linear model
- a function to perform the Kendall's test
- a master code putting it all together

Our first function:

```
load_data <- function(filename){
  papers <- read.csv(filename, stringsAsFactors = FALSE)
  papers$TitleLength <- nchar(papers$Title)
  papers$LogCits <- log10(papers$Cited.by + 1)
  return(papers)
}
```

Make sure that everything is well by testing our function on the data:

```
for (my_file in list.files("../data/citations", full.names = TRUE)){
  print(my_file)
  print(basename(my_file))
  papers <- load_data(my_file)
}
```

```
## [1] "../data/citations/ecology_letters.csv"
## [1] "ecology_letters.csv"
## [1] "../data/citations/nature_genetics.csv"
## [1] "nature_genetics.csv"
## [1] "../data/citations/nature_neuroscience.csv"
## [1] "nature_neuroscience.csv"
```

Now the function to fit the linear model:

```
linear_model_year_length <- function(papers){
  my_model <- summary(lm(LogCits ~ as.factor(Year) + TitleLength, data = papers))
  # Extract the coefficient and the pvalue
  estimate <- my_model$coefficients["TitleLength","Estimate"]
  pvalue <- my_model$coefficients["TitleLength","Pr(>|t|)"]
  return(list(estimate = estimate,
              pvalue = pvalue))
}
```

Let's run this on all files:

```
for (my_file in list.files("../data/citations", full.names = TRUE)){
  print(basename(my_file))
  papers <- load_data(my_file)
```

```r
  linear_model <- linear_model_year_length(papers)
  print(paste("Linear model -> coefficient",
        round(linear_model$estimate, 5),
        "pvalue", round(linear_model$pvalue, 5)))
}
```

```
## [1] "ecology_letters.csv"
## [1] "Linear model -> coefficient -3e-04 pvalue 0.45814"
## [1] "nature_genetics.csv"
## [1] "Linear model -> coefficient 0.00276 pvalue 0"
## [1] "nature_neuroscience.csv"
## [1] "Linear model -> coefficient 0.00567 pvalue 0"
```

Now the function that calls the Kendall's test for each year: we write two functions. One subsets the data, and the other simply runs the test.

```r
Kendall_test <- function(a, b){
  my_test <- cor.test(a, b, method = "kendall")
  return(list(estimate = as.numeric(my_test$estimate),
              pvalue = my_test$p.value))
}

call_Kendall_by_year <- function(papers){
  all_years <- sort(unique(papers$Year))
  for (yr in all_years){
    my_test <- Kendall_test(papers$TitleLength[papers$Year == yr],
                 papers$Cited.by[papers$Year == yr])
    print(paste("Year", yr, "-> estimate", my_test$estimate, "pvalue", my_test$pvalue))
  }
}
```

Now a master function to test that the program is working:

```r
analyze_journal <- function(my_file){
  # First, the linear model
  print(basename(my_file))
  papers <- load_data(my_file)
  linear_model <- linear_model_year_length(papers)
  print(paste("Linear model -> coefficient",
        round(linear_model$estimate, 5),
        "pvalue", round(linear_model$pvalue, 5)))
  # Then, Kendall year by year
  call_Kendall_by_year(papers)
}
analyze_journal("../data/citations/nature_genetics.csv")
```

```
## [1] "nature_genetics.csv"
## [1] "Linear model -> coefficient 0.00276 pvalue 0"
## [1] "Year 2004 -> estimate 0.045610559310653 pvalue 0.370327813824674"
## [1] "Year 2005 -> estimate -0.0550279205871904 pvalue 0.267581272430449"
## [1] "Year 2006 -> estimate 0.0596670158288517 pvalue 0.213567208478809"
## [1] "Year 2007 -> estimate -0.0390688441353209 pvalue 0.418061571895653"
## [1] "Year 2008 -> estimate 0.0674234973583218 pvalue 0.149431264169447"
## [1] "Year 2009 -> estimate 0.0297023932900025 pvalue 0.524960049608212"
## [1] "Year 2010 -> estimate -0.158944978390644 pvalue 0.00176578355758891"
## [1] "Year 2011 -> estimate 0.130352296718688 pvalue 0.00961339158713347"
## [1] "Year 2012 -> estimate 0.168763320248576 pvalue 0.000111079233389202"
## [1] "Year 2013 -> estimate 0.0792087074318355 pvalue 0.0880890674881281"
```

Finally, let's analyze all the journals!

```
for (my_file in list.files("../data/citations", full.names = TRUE)){
  analyze_journal(my_file)
}
```

```
## [1] "ecology_letters.csv"
## [1] "Linear model -> coefficient -3e-04 pvalue 0.45814"
## [1] "Year 2004 -> estimate -0.0330650126212166 pvalue 0.613150449110409"
## [1] "Year 2005 -> estimate 0.027544573550034 pvalue 0.671742234063288"
## [1] "Year 2006 -> estimate -0.0639728739951933 pvalue 0.334992423839896"
## [1] "Year 2007 -> estimate 0.130805775658087 pvalue 0.0792684296315683"
## [1] "Year 2008 -> estimate -0.0946065886996697 pvalue 0.185851799603582"
## [1] "Year 2009 -> estimate -0.00528034747952401 pvalue 0.932130040345834"
## [1] "Year 2010 -> estimate 0.0478717663229855 pvalue 0.429380872167305"
## [1] "Year 2011 -> estimate -0.103776346604215 pvalue 0.0989632669799205"
## [1] "Year 2012 -> estimate 0.0708774786316527 pvalue 0.205434298051716"
## [1] "Year 2013 -> estimate -0.0532355687009939 pvalue 0.326025326942473"
## [1] "nature_genetics.csv"
## [1] "Linear model -> coefficient 0.00276 pvalue 0"
## [1] "Year 2004 -> estimate 0.045610559310653 pvalue 0.370327813824674"
## [1] "Year 2005 -> estimate -0.0550279205871904 pvalue 0.267581272430449"
## [1] "Year 2006 -> estimate 0.0596670158288517 pvalue 0.213567208478809"
## [1] "Year 2007 -> estimate -0.0390688441353209 pvalue 0.418061571895653"
## [1] "Year 2008 -> estimate 0.0674234973583218 pvalue 0.149431264169447"
## [1] "Year 2009 -> estimate 0.0297023932900025 pvalue 0.524960049608212"
## [1] "Year 2010 -> estimate -0.158944978390644 pvalue 0.00176578355758891"
## [1] "Year 2011 -> estimate 0.130352296718688 pvalue 0.00961339158713347"
## [1] "Year 2012 -> estimate 0.168763320248576 pvalue 0.000111079233389202"
## [1] "Year 2013 -> estimate 0.0792087074318355 pvalue 0.0880890674881281"
## [1] "nature_neuroscience.csv"
## [1] "Linear model -> coefficient 0.00567 pvalue 0"
## [1] "Year 2004 -> estimate 0.00589142291037872 pvalue 0.918745470907139"
## [1] "Year 2005 -> estimate 0.0535867457573801 pvalue 0.243552261640925"
## [1] "Year 2006 -> estimate 0.114087173434659 pvalue 0.0168412989934458"
## [1] "Year 2007 -> estimate 0.00966171293776095 pvalue 0.843003041215137"
## [1] "Year 2008 -> estimate 0.0652257952013621 pvalue 0.200992557539214"
## [1] "Year 2009 -> estimate 0.012124492386758 pvalue 0.79906576984121"
## [1] "Year 2010 -> estimate -0.0518768931100408 pvalue 0.28616709714268"
## [1] "Year 2011 -> estimate 0.145346619264126 pvalue 0.00174156563082107"
## [1] "Year 2012 -> estimate 0.114380879075143 pvalue 0.0125050217924557"
## [1] "Year 2013 -> estimate 0.0446967410995934 pvalue 0.318724166924561"
```

Discussion: How many significant results we should expect, when citations and title lengths are completely independent?

Documenting the code using `knitr`

> *Let us change our traditional attitude to the construction of programs: Instead of imagining that our main task is to instruct a computer what to do, let us concentrate rather on explaining to humans what we want the computer to do.*

> Donald E. Knuth, Literate Programming, 1984

When doing experiments, we typically keep track of everything we do in a laboratory notebook, so that when writing the manuscript, or responding to queries, we can go back to our documentation to find exactly what we did, how we did it, and possibly why we did it. The same should be true for computational work.

`RStudio` makes it very easy to build a computational laboratory notebook. First, create a new `R Markdown` file (choose `File -> New File -> R Markdown` from the menu).

The gist of it is that you write a text file (`.Rmd`). The file is then read by an iterpreter that transforms it into an `.html` or `.pdf` file, or even into a Word document. You can use special syntax to render the text in different ways. For example,

```
***********

*Test* **Test2**

# Very large header

## Large header

### Smaller header

## Unordered lists

* First
* Second
    + Second 1
    + Second 2

1. This is
2. A numbered list

You can insert `inline code`

-----------
```

The code above yields:

Test **Test2**

Very large header

Large header

Smaller header

Unordered lists

- First
- Second
    - Second 1
    - Second 2

1. This is
2. A numbered list

You can insert `inline code`

The most important feature of `R Markdown`, however, is that you can include blocks of code, and they will be interpreted and executed by `R`. You can therefore combine effectively the code itself with the description of what you are doing.

For example, including

~~~
```r
print("hello world!")
```
~~~

will become

```r
print("hello world!")
```

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

If you don't want to run the `R` code, but just display it, use `{r, eval = FALSE}`; if you want to show the output but not the code, use `{r, echo = FALSE}`.

You can include plots, tables, and even render equations using LaTeX. In summary, when exploring your data or writing the methods of your paper, give `R Markdown` a try!

You can find inspiration in the notes for the Boot Camp: both the notes for Basic and Advanced Computing are written in `R MarkDown`.

Programming Challenge

Instructions

You will work with your own group to solve the following exercise. When you have found the solution, go to `https://stefanoallesina.github.io/BSD-QBio4/` and follow the link `Submit solution to challenge` 2 to submit your answer (alternatively, you can go directly to `goo.gl/forms/QJhKmdGqRCIuGNPa2`. At the end of the boot camp, the group with the highest number of correct solutions will be declared the winner. If you need extra time, you can work with your group during free time or in the breaks in the schedule.

Google Flu Trends

Google Flu started strong, with a paper in Nature (Ginsberg *et al.*, 2008) showing that using data on web queries, one could predict the number of physician visits for influenza-like symptoms. Over time, the quality of predictions degraded considerably, requiring many adjustments to the model. Now defunct, Google Flu Trends has been proposed as a poster child case of the *Big Data hubris* (Lanzer *et al.* Science 2014). In the folder `data/GoogleFlu` you can find the data used by Preis & Moat (2014, `dx.doi.org/10.1098/rsos.140095`) to show that, once accounted for some additional historical data, Google Flu Trends are correlated with outpatient visits due to influenza-like illness.

1. Read the data and plot number of visits vs. `GoogleFluTrends`
2. Calculate the (Pearson's) correlation using `cor`
3. The data spans 2010-2013. In Aug 2013 Google Flu changed their algorithm. Did this lead to improvements? Compare the data from Aug to Dec 2013 with the same months in 2010, 2011, and 2012. For each, calculate the correlation, and see whether the correlation is higher for 2013.

Hints You will need to extract the year from a string for each row. To do so, you can use `substr(google$WeekCommencing, 1,4)`.