Nineth R Practice exercise: using a row of a data frame, the unlist and unname functions, composing a function

Alan E. Berger Feb 8, 2020; a couple edits Feb 9; edits on statistical significance Feb 11

available at https://github.com/AlanBerger/Practice-programming-exercises-for-R

Introduction

This is the nineth in a sequence of programming exercises in "composing" an R function to carry out a particular task. Several of these "exercise files" likely
will take several sessions to master the content. The material below practices composing a logical sequence of steps to program a function that will accomplish a specified task, and preparing a corresponding data frame.

The idea of this set of exercises is to practice correct use of R constructs and built in functions (functions that "come with" the basic R installation), while learning how to "put together" a correct sequence of blocks of commands that will obtain the desired result.
Note these exercises are quite cumulative - one should do them in order.

In these exercises, there will be a statement of what your function should do (what are the input variables and what the function should return) and a sequence of "hints". To get the most out of these exercises, try to write your function using as few hints as possible.
Note there are often several ways to write a function that will obtain the correct result. For these exercises the directions and hints may point toward a particular approach intended to practice particular constructs in R and a particular line of reasoning, even if there is a more efficent way to obtain the same result.
There may also be an existing R function or package that will do what is stated for a given practice exercise, but here the point is to practice formulating a logical sequence of steps, with each step a section of code, to obtain a working function, not to find an existing solution or a quick solution using a more powerful R construct that is better addressed later on.

Motivation for this exercise

In some cases, such as with a gene expression data set, one will want to extract and use subsets of a given **row** of a data frame. While an individual column (or a subset of an individual column) of a data frame can readily be extracted in the form of a vector, extracting a row or part of a row of a data frame results in a 1 row data frame, not a vector. We can use the **unlist** function to get the corresponding vector (recall that a data frame is a special type of list). For example:

```
# construct a simple data frame
m <- matrix(1:20, nrow = 4, ncol = 5)
df <- data.frame(m)  # convert the matrix m to a data frame
df
  X1 X2 X3 X4 X5
1  1  5  9 13 17
2  2  6 10 14 18
3  3  7 11 15 19
4  4  8 12 16 20

# one can readily get a column and a subset of a column of df as a vector:
df.X3 <- df[["X3"]]
df.X3
[1]  9 10 11 12
# and also:  df.X3 <- df[[3]]  or  df.X3 <- df$X3  or  df.X3 <- df[, 3]
#            or  df.X3 <- df[, "X3"]

df.colX3.subset <- df[["X3"]][c(2,4)]
df.colX3.subset
[1] 10 12
is.vector(df.colX3.subset)
[1] TRUE

# now let’s see what we get if we extract a row of df
df.row2 <- df[2, ]
class(df.row2)
[1] "data.frame"
df.row2
  X1 X2 X3 X4 X5
2  2  6 10 14 18

# note the row name given for df.row2 is the row name of
# the row of the data frame that df.row2 was extracted from

df.row2.subset <- df[2, c(2,4,5)]
class(df.row2.subset)
[1] "data.frame"
> df.row2.subset
  X2 X4 X5
2  6 14 18

mean(df.row2)
[1] NA
Warning message:
In mean.default(df.row2) : argument is not numeric or logical: returning NA

# The problem is that the mean function "demands" that its argument be a vector

# Note the sum function is not so demanding
sum(df.row2.subset )
[1] 38
# but in general, to avoid problems,
# if one wants something treated as a vector
# one should convert it to a vector

as.vector(df.row2)
  X1 X2 X3 X4 X5
2  2  6 10 14 18
class(as.vector(df.row2))
[1] "data.frame"
# that didn’t work, so how do we get df.row2 in the form of a vector?

unlist.df.row2 <- unlist(df.row2)  ################  unlist gets a vector
is.vector(unlist.df.row2)
[1] TRUE

str(unlist.df.row2)
 Named int [1:5] 2 6 10 14 18
 - attr(*, "names")= chr [1:5] "X1" "X2" "X3" "X4" ...
```

```
# so unlist gives us a vector whose entries have names
unlist.df.row2
X1 X2 X3 X4 X5
 2  6 10 14 18

mean(unlist.df.row2)  # and the R mean function will cheerfully calculate its mean
[1] 10

# if we don’t want the names of the entries we can "discard" them

unname(unlist.df.row2)   # remove the names from the vector entries
[1]  2  6 10 14 18

# note unlist doesn’t do what one might think if there is more than 1 row

df.rows2and3 <- df[c(2,3), ]
df.rows2and3
  X1 X2 X3 X4 X5
2  2  6 10 14 18
3  3  7 11 15 19

udf.23 <- unlist(df.rows2and3)  # this gets a vector, not a matrix
udf.23
X11 X12 X21 X22 X31 X32 X41 X42 X51 X52
  2   3   6   7  10  11  14  15  18  19

# and note  the ordering of the entries is "by column" of the matrix

class(udf.23)
[1] "integer"
> str(udf.23)
 Named int [1:10] 2 3 6 7 10 11 14 15 18 19
 - attr(*, "names")= chr [1:10] "X11" "X12" "X21" "X22" ...

# as matrix will get the corresponding matrix
m.df.rows2and3 <- as.matrix(df.rows2and3)
m.df.rows2and3
  X1 X2 X3 X4 X5
2  2  6 10 14 18
3  3  7 11 15 19

class(m.df.rows2and3)
[1] "matrix"
str(m.df.rows2and3)
 int [1:2, 1:5] 2 3 6 7 10 11 14 15 18 19
 - attr(*, "dimnames")=List of 2
  ..$ : chr [1:2] "2" "3"
  ..$ : chr [1:5] "X1" "X2" "X3" "X4" ...
```

A “real” example where one wants to extract subsets of rows of a data frame

We are going to use a very small subset of a “real” gene expression data set. While the following description is not necessary as far the R constructs being addressed, I think it is satisfying to have some understanding of the the data being used - and in a “real” analysis, that can be essential in selecting proper analysis procedures and in detecting any “problems” with the data. To very briefly, conceptually, summarize what gene expression is: DNA in a cell encodes information that enables cellular biochemical processes to synthesize specific proteins (and other biomolecules). This is subject to intricate regulatory control. For the purposes of this example, a *gene* will be regarded as a section of DNA that encodes for a protein. To synthesize a protein, the cellular “machinery” first *transcribes* the gene into copies of the corresponding **messenger RNA** (**mRNA**), (a “working copy” of that section of DNA), which is then used in the *translation* process of making copies of the protein. The amount of mRNA in a cell for each gene at a given time is used as a measure of the “activity” of the genes. This can be measured by *microarry* and now more commonly by *RNA Sequencing* (RNA-Seq). With microarray technology, for a given sample of tissue or a given sample of a subset of blood cells, measured levels of fluorescence corresponding to one or more *probes* for each gene is in principle proportional to the amount of mRNA for each gene in the sample. (We are not going to address *splice variants* here.) With RNA-Seq one obtains **count data** which necessitates more complex statistical modeling. In molecular biology and clinical research, measures of gene expression, i.e., mRNA levels, are used in research to understand the molecular biology of diseases, with an aim to develop better treatments, better diagnostic procedures, and better prognostic indicators. Gene expression levels can be used in classifying disease subtypes, for example tumor subtypes, which can be used in selecting better treatment regimens.

The following data frame is a tiny subset of an Illumina microarray data set comparing gene expression levels in 41 Wegener’s granulomatosis (**WG**) patients vs. 23 Normal Controls (**NC**). The microarray platform used has 22185 probes measuring the levels of mRNA for 18196 unique genes (as of the latest annotation for this platform - Jan 18, 2013); note there can be more than 1 probe measuring the amount of mRNA for a given gene. The full data set (and a link to the corresponding paper, and to the annotation for the microarray platform (GPL6104)) is available in the NCBI Gene Expression Omnibus (GEO) repository: https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE18885

```
# A copy of this data file (a small tab delimited text file) has been placed in
# GitHub, at https://github.com/AlanBerger/Practice-programming-exercises-for-R
# The file name there is
# tiny-subset-of-GSE18885-gene-expression-data-9-genes-WG-5-samples-Normal-
# Control-4-samples.tab.txt

# the url for reading this data file into an R data frame using read.delim
# (for reading in tab delimited text files) is given in the next 3 lines
url.for.data.file <- "https://raw.githubusercontent.com/AlanBerger/
Practice-programming-exercises-for-R/master/tiny-subset-of-GSE18885-
gene-expression-data-9-genes-WG-5-samples-Normal-Control-4-samples.tab.txt"

# Note this is NOT the same URL as
# https://github.com/AlanBerger/Practice-programming-exercises-for-R/blob/master/
#       tiny-subset-of-GSE18885-gene-expression-data-9-genes-WG-5-samples-Normal-
#       Control-4-samples.tab.txt
#
# and was obtained by clicking on the
# URL https://github.com/AlanBerger/Practice-programming-exercises-for-R
# and then clicking on the file name that starts with:  tiny-subset-of-GSE18885
# and then by, in the resulting web page, clicking on the "raw" box
# If one right clicks (in Windows) on the "raw" box, and then clicks on
# "Save link as..." one can
# download a copy of this file into your computer
# read in the data as a data frame
ma <- read.delim(url.for.data.file, nrows = 9, check.names = FALSE,
                 stringsAsFactors = FALSE)

# Note the use of nrows = 9 since there is information on the source of this data in
# later rows that should not be read in as data.
# The choice check.names = FALSE "tells" R to leave the column headers as is
# (otherwise R would replace spaces in column names with a period which is not visually
# appealing, and here we don't need to have "standard for R" column names).
ma
  Illumina PROBE_ID    gene   NC 1   NC 2    NC 3   NC 4    WG 1   WG 2    WG 3    WG 4   WG 5
1      ILMN_1730867    AZU1 7.5719 7.8004  9.2853 7.6631 10.1137 7.7436  9.2764 13.2686 7.5619
2      ILMN_1766736     BPI 7.6313 8.0540  8.5862 8.3428 10.2506 8.7722 10.3036 14.3789 8.2839
3      ILMN_1688580    CAMP 8.7349 8.4446  9.9021 8.6375 12.1782 9.1157 13.2256 14.9976 9.5919
4      ILMN_1806056 CEACAM8 8.3485 8.2501 10.2744 8.1821 11.8043 8.0849 12.3573 15.3588 8.3634
5      ILMN_1753347   DEFA4 7.9815 8.5601 10.7857 7.9922 12.5999 8.4461 11.2124 15.4463 8.6771
6      ILMN_1706635    ELA2 8.8103 8.8859 11.1437 8.5417 11.5893 8.8538 10.5037 15.0403 8.2517
7      ILMN_1796316    MMP9 7.5506 7.5359  7.6544 7.6399  9.8731 9.2065 12.5744 14.1953 8.0277
8      ILMN_1705183     MPO 8.4673 8.4729  9.8118 8.5687 10.6068 9.0168  9.2412 14.5349 8.2586
9      ILMN_1802867  RNASE3 7.8771 8.6170  9.2255 7.9219 11.4129 8.5187 10.3054 14.2301 8.6456
```

Here we have data for 9 Illumina probes, giving gene expression levels for 9 genes chosen for this example from genes relevant to Wegener's granulomatosis (**WG**), a rare systemic inflammatory disease (now "officially" called granulomatosis with polyangiitis (GPA)). The 4 Normal Control (NC) samples and the 5 Wegener's granulomatosis (WG) samples are the first four NC and the first five WG samples as listed in the GEO repository for this data set. The data is measured gene expression levels in the peripheral blood mononuclear cell (PBMC) fraction of blood cells. The measured expression levels for each sample were *normalized* by *scaling* so that the median of the expression values for all the probes for each sample was 256 (2ˆ8); (all the values for each sample were multiplied by a constant k specific to that sample so that the median of the values for each sample was 256). The choice of 256 does not matter for the analysis, it is just a convenient value.

Normalization (here done by scaling) adjusts values to account for, in particular, different total amounts of mRNA used for individual samples, and different *gain* settings in the fluorescence scanner; scaling is a sufficient normalization procedure for this type of Illumina microarray. The normalized expression values are then log base 2 transformed, i.e., the logarithm (base 2) of each scaled value was used in the analysis procedures described below. Note then the median of the scaled log base 2 transformed values for each sample is 8. This is a little above the "background noise" level for this technology for this data set. The **ma** data frame above contains a small illustrative subset of the scaled log base 2 transformed expression level values.

The log transformed values are more suitable for use with the t-test to get p-values for examining **statistical significance**
of differences in gene expression between the WG and NC groups. Statistical significance is a major topic in the Statistical Inference course (Course 6) in the Johns Hopkins University Data Science Specialization on Coursera; one doesn't need to be familiar with this for the purpose of this practice exercise. Here we are going to calculate, for each gene, the *p-value* and the WG/NC *fold change* (the ratio of the expression level of the gene in the WG samples divided by the expression level of the gene in the NC samples) and construct a data frame containing this information.

Side note: statistical significance (the *p-value*) is referring to how unusual it would be to have obtained, just by random chance, a test statistic as large or larger in magnitude than the one calculated for the actual data, when there was no difference between the two populations the data was sampled from (that is, no difference in terms of the value being measured, e.g., the expression level of a given gene), assuming the requirements for the statistical test to be valid were satisfied. As a simple example, if one flipped a coin 7 times and got all heads, how unlikely would that be if one assumed the coin was a “fair coin” - not weighted to favor either heads or tails.

It is important to be aware that **statistical significance** does not necessarily imply “practical significance”. For example, a medication under study to reduce blood pressure might result in a *very small* lowering of blood pressure, that could in theory be demonstrated in a sufficiently large clinical trial (i.e., with enough subjects in the clinical trial, one would expect to obtain a small p-value with a suitable statistical test on the blood pressure difference between treatment and no treatment). The small lowering of blood pressure would then be *statistically significant*, but not medically important (so NOT “significant” in the usual way the word significant is used). One thus needs to also consider the **effect size**, that is, how large some measure of the difference is, and is that of practical importance in the particular setting that the data is referring to. For example the level of change in gene expression that has an important effect for a protein that initiates an amplifying cascade of regulatory signaling in a cell might be of no importance for a protein whose effect is not similarly amplified. Interpretation of statistical analysis will often depend on subject area knowledge by the analyst and/or on an appropriate collaboration. The *fold change* as defined above is a measure of effect size commonly used in gene expression analysis.

Note for a full gene expression data set, which often has data for well over 10,000 genes it is customary to have the rows be for genes and the columns be for samples. This is opposite the usual layout of data frames in R where the observations (samples) are placed row by row, but having the genes in the rows is more amenable for use in Excel which was initially often used for analysis of this type of data, and Excel is a convenient way to get a “first look” at such data. So in this setting, we will want to get subsets of each row of the data frame in the form of vectors.

Programming Exercise: Calculate the p-value and fold change for comparing the WG vs. NC expression levels for each gene and place the results in a data frame

Approach: In a for loop running over the 9 rows of the **ma** data frame above; first obtain **as vectors**, the 4 expression values for the NC samples for the gene of the the row, and the 5 expression values for the WG samples for the gene of the row. Call these vectors, for example, NCvec and WGvec respectively. One can calculate the **two-sided p-value** for the difference in expression levels between the WG and NC groups for the gene via `pval <- t.test(WGvec, NCvec)$p.value` The WG/NC expression level **fold change** is given by `WG.over.NC.fold.change <- 2^(mean(WGvec) - mean(NCvec))` The form of this is due to the fact that the data in the ma data frame consists of log base 2 values. Side note: `2^(mean(WGvec) - mean(NCvec))` is equal to `2^mean(WGvec) / 2^mean(NCvec)` and the numerator and denominator in this expression are the *geometric mean* of the normalized (but not log 2 transformed) expression values for the WG samples and for the NC samples, respectively. Place the p-values in a vector and the fold change values in a vector and construct a data frame consisting of the names of the probe identifiers (column 1 of the ma data frame), the genes, and the p-values and the fold changes.

A working version of R code which does this is given below.

```r
# the url for reading the little gene expression data file into an R data frame using
# read.delim  (for reading in tab delimited text files) is given in the next 3 lines
url.for.data.file <- "https://raw.githubusercontent.com/AlanBerger/
Practice-programming-exercises-for-R/master/tiny-subset-of-GSE18885-
gene-expression-data-9-genes-WG-5-samples-Normal-Control-4-samples.tab.txt"

# read in the data as a data frame
ma <- read.delim(url.for.data.file, nrows = 9, check.names = FALSE,
                 stringsAsFactors = FALSE)

# display ma
ma
```

```
##   Illumina PROBE_ID    gene   NC 1   NC 2    NC 3   NC 4    WG 1   WG 2    WG 3
## 1      ILMN_1730867    AZU1 7.5719 7.8004  9.2853 7.6631 10.1137 7.7436  9.2764
## 2      ILMN_1766736     BPI 7.6313 8.0540  8.5862 8.3428 10.2506 8.7722 10.3036
## 3      ILMN_1688580    CAMP 8.7349 8.4446  9.9021 8.6375 12.1782 9.1157 13.2256
## 4      ILMN_1806056 CEACAM8 8.3485 8.2501 10.2744 8.1821 11.8043 8.0849 12.3573
## 5      ILMN_1753347   DEFA4 7.9815 8.5601 10.7857 7.9922 12.5999 8.4461 11.2124
## 6      ILMN_1706635    ELA2 8.8103 8.8859 11.1437 8.5417 11.5893 8.8538 10.5037
## 7      ILMN_1796316    MMP9 7.5506 7.5359  7.6544 7.6399  9.8731 9.2065 12.5744
## 8      ILMN_1705183     MPO 8.4673 8.4729  9.8118 8.5687 10.6068 9.0168  9.2412
## 9      ILMN_1802867  RNASE3 7.8771 8.6170  9.2255 7.9219 11.4129 8.5187 10.3054
##      WG 4   WG 5
## 1 13.2686 7.5619
## 2 14.3789 8.2839
## 3 14.9976 9.5919
## 4 15.3588 8.3634
## 5 15.4463 8.6771
## 6 15.0403 8.2517
## 7 14.1953 8.0277
## 8 14.5349 8.2586
## 9 14.2301 8.6456
```

```r
# now, in a for loop, get the p-values and fold changes
# for each gene as described above
num.genes <- nrow(ma)  # the number of genes in this data frame

gene <- ma$gene  # the column of gene names
probe.vec <- ma[[1]]
# get vectors to hold the p-value and fold change values
p.value <- numeric(num.genes)
fold.change <- numeric(num.genes)

for (i in 1:num.genes) {
# get the vector for the WG expression values and the vector
# for the NC expression values for the ith gene
   NCvec <- unlist(ma[i, 3:6])
   WGvec <- unlist(ma[i, 7:11])

# calculate the p-value and fold change
   pval <- t.test(NCvec, WGvec)$p.value  # two-sided unequal variance (Welch) t-test
   p.value[i] <- pval
   WG.over.NC.fold.change <- 2^(mean(WGvec) - mean(NCvec))
   fold.change[i] <- WG.over.NC.fold.change
}

# Construct the desired data frame. Include the PROBE_ID column since
# it is used as the identifier in the microarray platform annotation file, and
# the probe identifiers are all distinct. Also, several probes can "query" the
# same gene, and in fact one gene can code for variants of a protein using
# combinations of different sub-sections (exons) of the gene, so the information
# on which sub-section of the gene the probe is "matching" can be important.

analysis.results <- data.frame(probe.vec, gene, p.value, fold.change,
                               stringsAsFactors = FALSE, check.names = FALSE)
colnames(analysis.results) <- c("Illumina PROBE_ID", "gene", "two-sided p-value",
                                          "WG/NC fold change")
analysis.results
```

```
##   Illumina PROBE_ID    gene two-sided p-value WG/NC fold change
## 1      ILMN_1730867    AZU1        0.22993719          2.853366
## 2      ILMN_1766736     BPI        0.10421720          4.737957
## 3      ILMN_1688580    CAMP        0.05731959          7.423116
## 4      ILMN_1806056 CEACAM8        0.15314950          5.388804
## 5      ILMN_1753347   DEFA4        0.14680018          5.450864
## 6      ILMN_1706635    ELA2        0.30872616          2.833058
## 7      ILMN_1796316    MMP9        0.04866715          9.064328
## 8      ILMN_1705183     MPO        0.25730723          2.831340
## 9      ILMN_1802867  RNASE3        0.10343345          4.633701
```

In a "full" gene expression analysis, where one examines on the order of ten or twenty thousand genes, one needs to account for the large number of statistical tests being done, a consequence of which is having a considerable number of low p-values
occur just by random chance. This issue is often addressed by calculating the *Benjamini-Hochberg false discovery rate* (FDR) for each gene; the FDR is covered in the Statistical Inference course (Video 12 01 Multiple Comparisons). Note we used a small subset of the samples, and hence there is less *statistical power* and the p-values above are much larger than obtained when using the data from all the samples.

Hope this was informative and good practice. The next exercise will contain further practice in using data frames, utilizing this gene expression data set.

= =

Note the reader should not infer any endorsement or recommendation or approval for the material in this article from any of the sources or persons cited above or any other entities mentioned in this article.