R tutorial, session 3

H. Seitz (IGH)
(herve.seitz@igh.cnrs.fr)

October 13, 2014

Contents

1 Assessing normality with the Shapiro-Wilk test

Let's define two datasets, called `a` and `b`:

Figure 1: **Defining two vectors of values.**

Let's run the Shapiro-Wilk test on these datasets, using the `shapiro.test()` command:

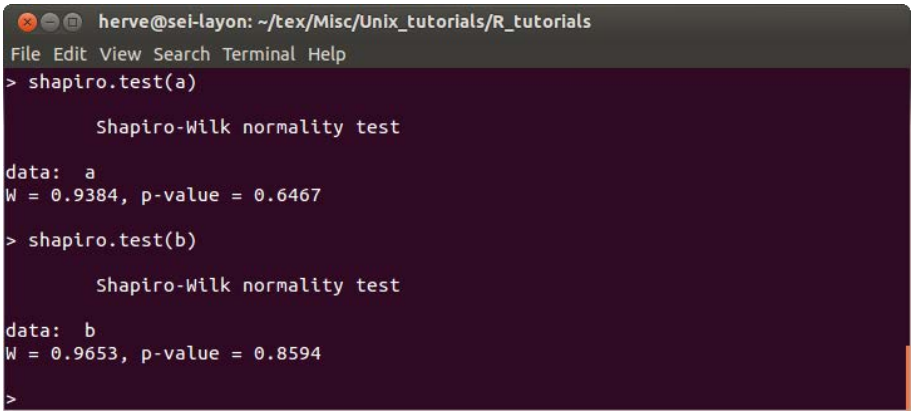

Figure 2: **Testing normality using the Shapiro-Wilk test.**

As both p-values are high (0.6467 for `a` and 0.8594 for `b`), we can consider the two datasets follow a normal distribution, and compare them using the t-test. The output of the `shapiro.test()` command gives the name of the test that was used ("Shapiro-Wilk normality test"), the name of the data vector analyzed (`a` in the first case, `b` in the second one), the value of the `W` statistics (that the test uses to calculate the p-value), and the p-value. If for some reason you would like to extract just the p-value,you can type:

Figure 3: **Extracting Shapiro-Wilk test p-value.** Note that the precision is better than in the default test output (7 significant digits here).

Just like for data frames, whose column vectors can be extracted using the "$" sign followed by the name of the column (see document for session 1), here we extract a value named "p-value" from the output of the test. *N.B.*: the name of values that can be extracted from the output of a test can be found in section "Value" of the help page displayed by `?shapiro.test`.

2 Assessing variance homogeneity with the Levene test

The Levene test is not provided by the base R program, but it is part of an additional package named `car`. Let's install that package (as seen in session 1) using the `install.packages()` command:

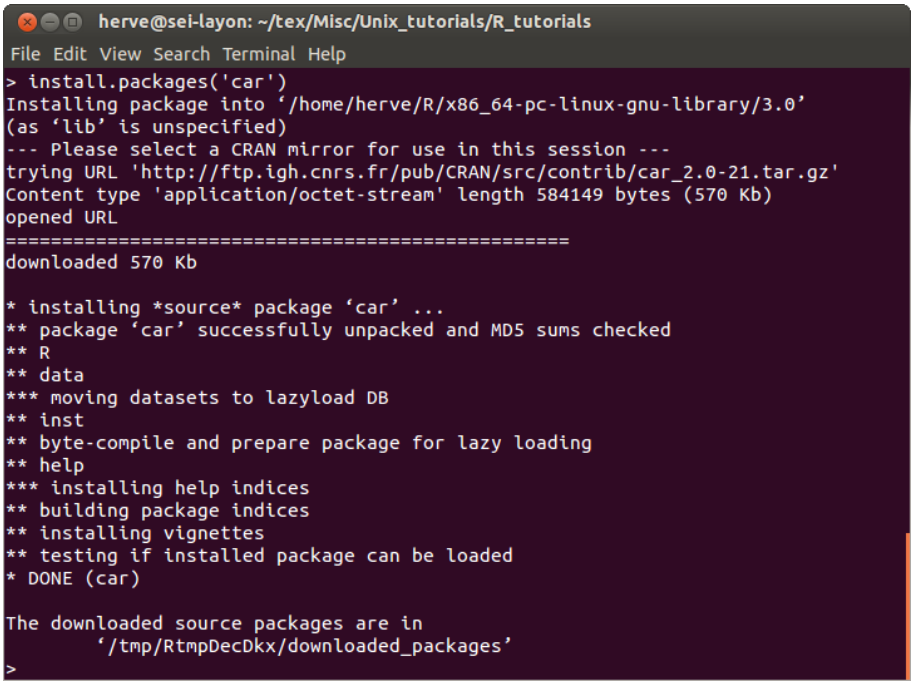

Figure 4: **Installing the car package.**

You only need to install a package once per computer. But then, if you want to use it during any given R session, you have to tell R to load that package, because you will be needing it. This is done using the `library()` command:

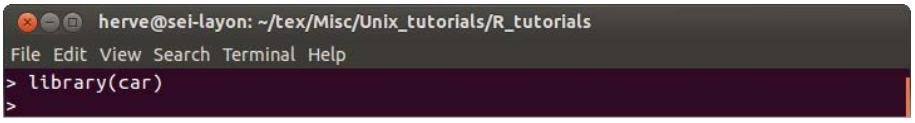

Figure 5: **Loading the (already installed) car package.**

Now we can use the commands provided by this package, in particular the `leveneTest` command. Evaluating the homogeneity of variances of two vectors is a little bit tricky with that command: you have to provide two arguments, the first one being a concatenation of the values of the two vectors (for example, `c(a,b)`, which is a vector concatenating values from `a` and values from `b`) and the second one being a vector of "factors" flagging each value in the concatenated vector with the name of its category:

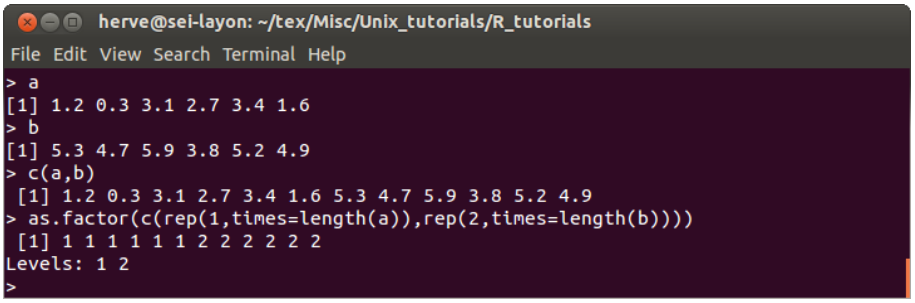

Figure 6: **Concatenating two vectors of values and defining groups in a variable of type "factor".**

So you can run the Levene test with these two inputs:

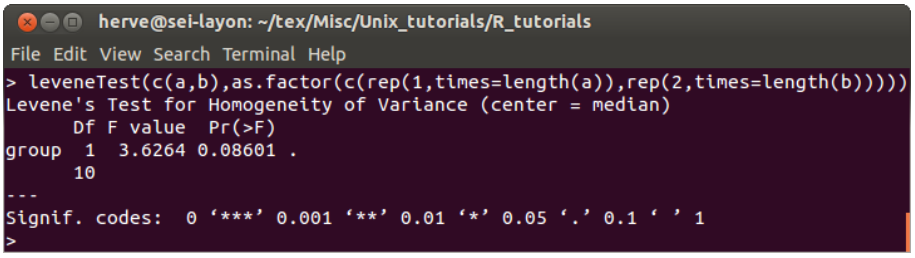

Figure 7: **Testing variance homogeneity using the Levene test.**

Here the p-value is reasonably large (0.8601): we can consider variances to be homogeneous, and use the t-test with equal variances.

3 Comparing datasets with the t-test

The command for the t-test is, very simply, `t.test()`. It requires two mandatory arguments (the two data vectors to be compared):

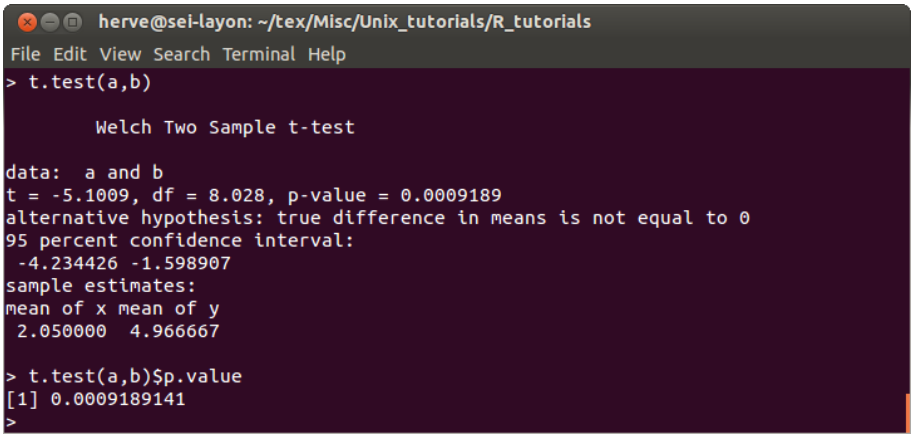

Figure 8: **Running the t.test() command with default settings.** It is possible to extract the p-value from the test's output using the "$p.value" extractor (see last two lines).

By default, the `t.test()` runs Welch's t-test (that does not assume homogeneous variances). Here we saw that vectors `a` and `b` had homogeneous variances, so we should rather use Student's t-test. This is achieved by providing a third, optional argument named `var.equal`):

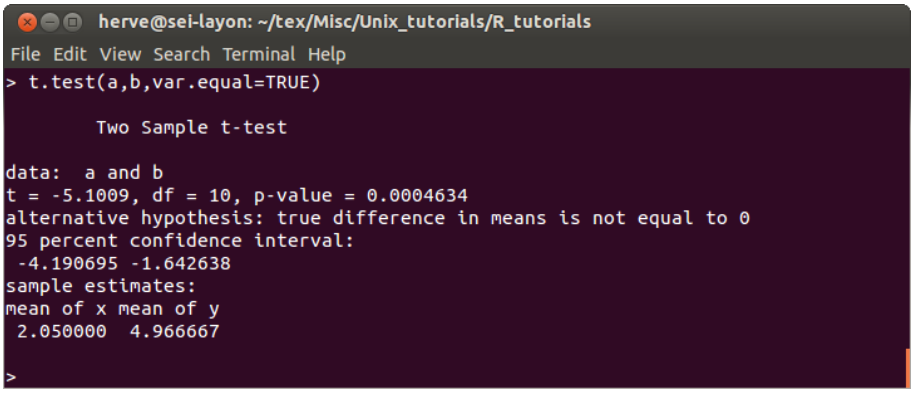

Figure 9: **Running Student's t-test.**

You can also set `var.equal` to "FALSE", then it will run Welch's t-test (but as this is the default option, you don't actually need to explicitly type `var.equal=FALSE`: see figure 8).

Additional options allow you to run a one-sided or two-sided t-test (option `alternative='greater'` or `alternative='less'`, two-sided t-test being the default) and a paired t-test (option `paired=TRUE`, with FALSE being the default). Of course, a paired t-test can only be applied to vectors having the same number of elements (which is the case in our example: `a` and `b` both have 6 elements). If really the data is paired, but somehow some replicates are missing in one of the two vectors (because of an experimental problem for example), you can fill the blanks in the shorter vector with `NA` ("NA" stands for "not available"):

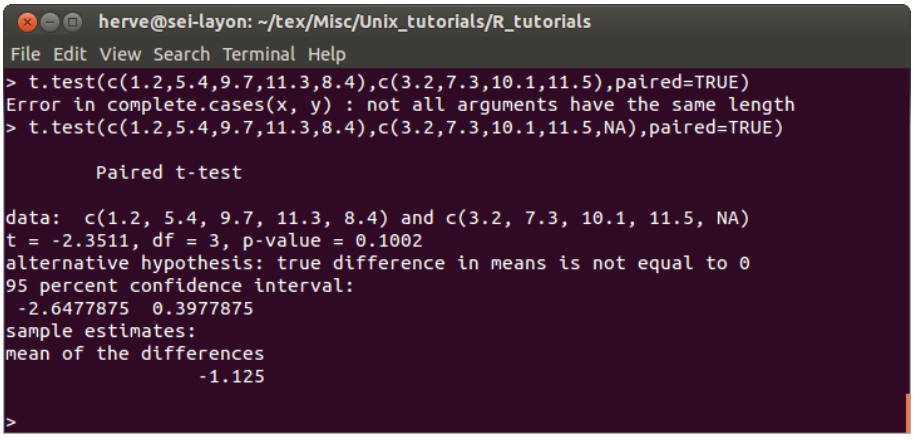

Figure 10: **Paired t-test, with a missing value in one dataset.** Top two lines: attempting to run a paired t-test with vectors or unequal lengths results in an error.

In a paired t-test, R considers that the pairs are constituted of the elements having the same rank in each vector (here: 1.2 of the first dataset is paired with 3.2 of the second dataset, 5.4 of the first dataset is paired with 7.3 of the second dataset, ...).

4 log-transformation and Wilcoxon test

Now if we consider a new data vector, called `c`, and we want to compare it to `a`:

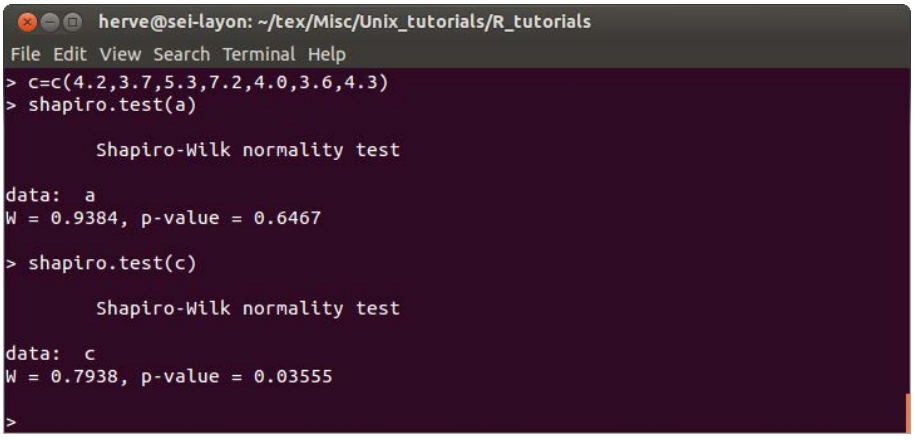

Figure 11: **Two datasets, one of them being non-normally distributed.**

We can have a look at the normality of the logarithm of their values:

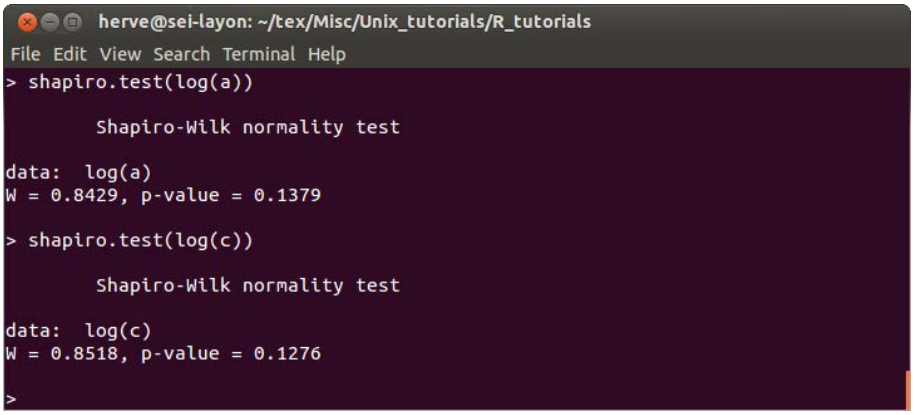

Figure 12: **Assessing the normality of log(data).**

Conveniently, the logarithm of these two datasets appears to be normally distributed, allowing us to use the t-test even in this situation:

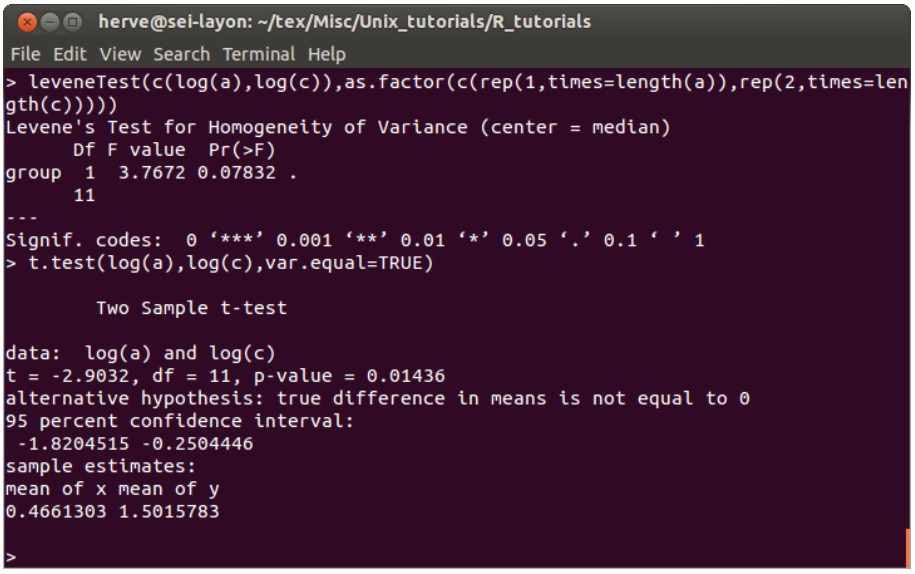

Figure 13: **Running the t-test on log-transformed data.** We first compared their variances using the Levene test.

But from time to time, some datasets will be so weirdly distributed that even their log won't be suitable:

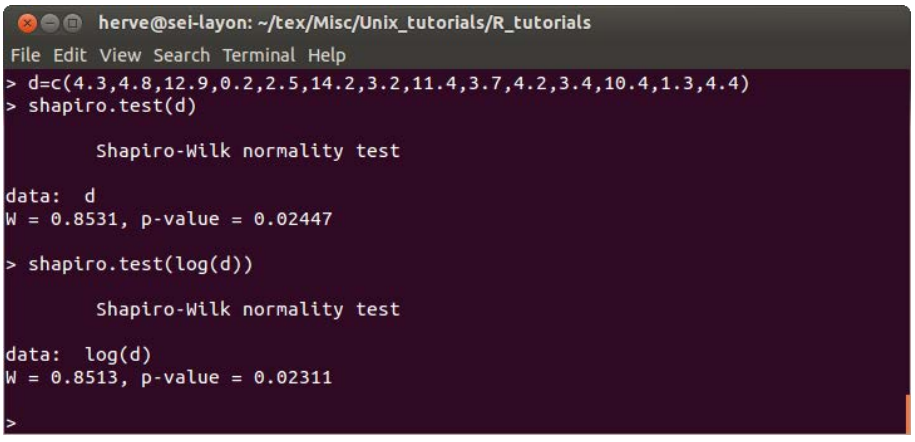

Figure 14: **A non-normally distributed dataset, whose log is non-normally distributed.**

Here, we have no better choice than using the Wilcoxon test to compare it to dataset `a`:

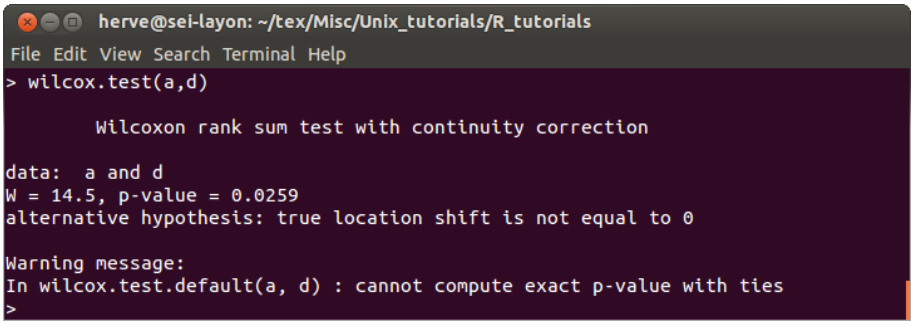

Figure 15: **Running the Wilcoxon test.** Note that, when the concatenation of the two datasets contains repeats of the same values, the Wilcoxon test is less precise: it displays a warning message (here: value "3.4" is shared between vectors a and d).

5 Correction for multiple hypothesis testing

When performing multiple statistical tests, you will end up with a list of p-values (for example, you want to know what mRNAs are misregulated in a series of mutant replicates, relatively to a series of wild-type replicates; you will get one t-test p-value per mRNA: that's going to be more than 20,000 p-values if you analyzed every murine mRNA for example). With a p-value cutoff of 0.05, 5% of these p-values will be lower than the cutoff, hence the difference will be flagged "significant". But this will just be a meaningless consequence of the fact that, just by chance, the wild-type and mutant replicates fell in the two opposite tails of the Gaussian curve (if you did some more replicates of the experiment, the very same mRNAs would come out with a high p-value).

Hence, when testing for multiple hypotheses, it is necessary to use a more stringent cutoff, adapted to the number of tested hypotheses. The most simple p-value adjustment procedure is called the "Bonferroni correction": if N hypotheses are to be tested, you should replace your usual significance threshold on the p-value (0.05 for most people) by: $\frac{usual\ \ threshold}{N}$.

In fact, instead of dividing the threshold by N, people generally prefer to multiply p-values by N (which is formally equivalent), except that, when the multiplied p-value ends up being larger than 1, it is set to 1. These corrected p-values are then to be compared to the usual significance threshold (*e.g.*, 0.05).

Such p-value adjustment scheme proves too conservative when the number of tested hypotheses is really large (like the ones we currently get with high-throughput analyses in molecular biology): even significant changes are often attributed a large corrected p-value, resulting in a high rate of false negatives.

A softer p-value adjustment procedure is the "Benjamini-Hochberg" correction (also called "False discovery rate", FDR, control): raw p-values are first ranked by decreasing order, then each p-value is multiplied by its rank (setting it to 1 when the multiplied p-value ends up being larger than 1). This way, only the last one (*i.e.*, the smallest one) is multiplied by N. Empirically, this correction scheme appears to be better than the Bonferroni correction for the identification of differentially regulated genes in high-throughput experiments.

p-values can be adjusted using the `p.adjust()` command:

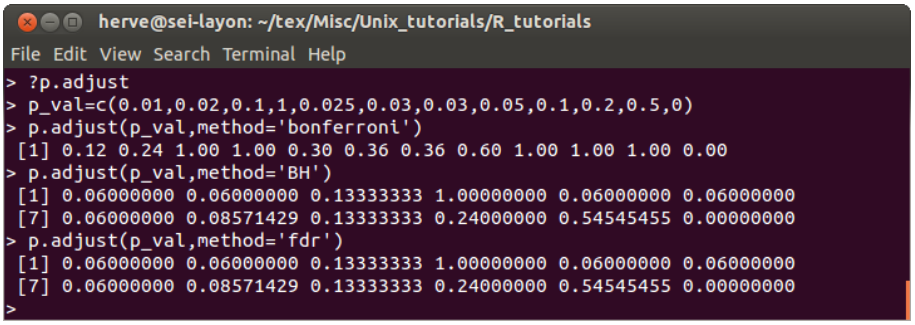

Figure 16: **Adjusting p-values.** Note that methods 'BH' (Benjamini-Hochberg) and 'fdr' give the same result (these are synonyms).

You can see the list of available correction methods (and some background about each of them) by typing `?p.adjust`.

6 Correlation test

Given two datasets (*e.g.*, 3´ UTR length and ORF length, for every annotated human mRNA), you may want to know whether these two variables are correlated, meaning that an increase in one variable is usually associated with an increase in the other variable (or inversely: an increase in one variable is usually associated with a decrease in the other variable).

Data file 'Length_sample.dat' contains 3´ UTR and ORF length data (for a sample of 20,000 annotated human mRNAs in Ensembl).

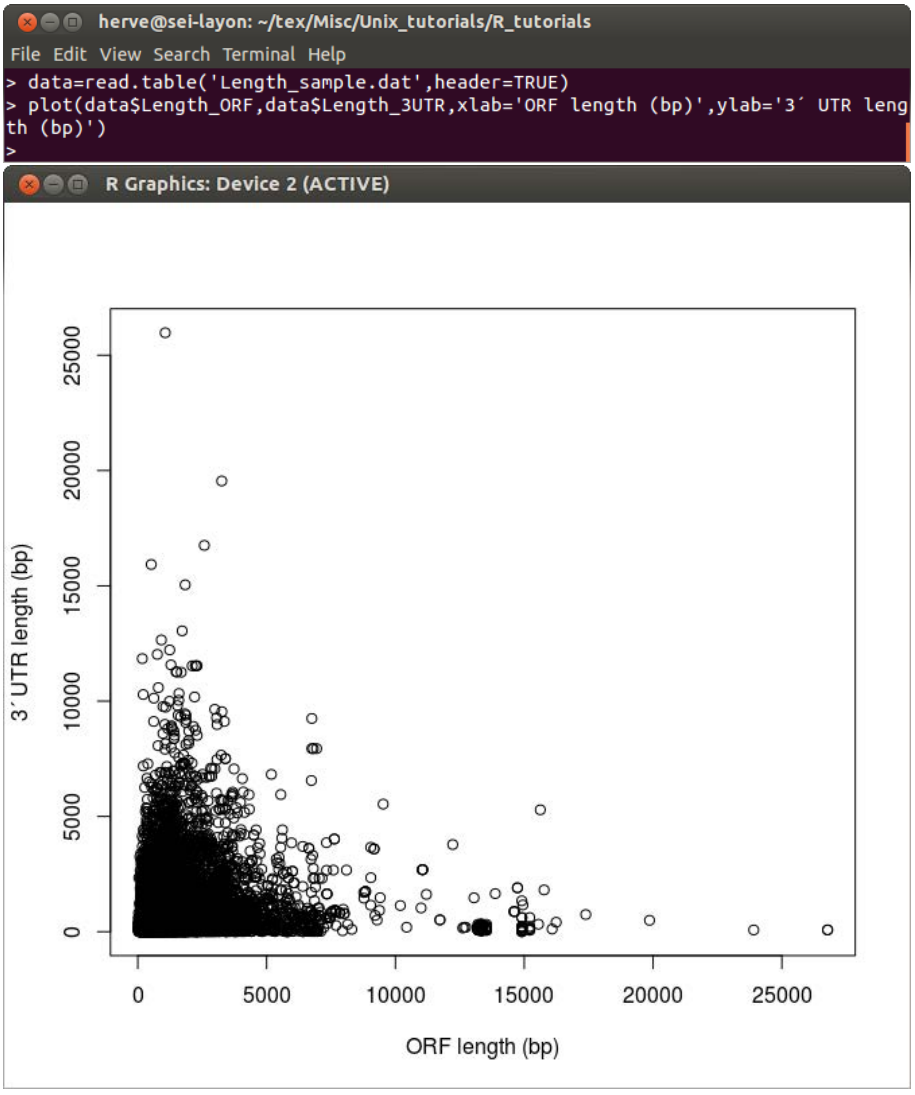

Figure 17: **Loading 3´ UTR and ORF length data, and plotting.** The bottom panel is the resulting scatter plot.

Visual inspection of the scatter plot does not reveal any clear trend: some genes have very short ORFs and very long 3´ UTRs, while some genes have very long ORFs and very short 3´ UTRs (which suggests that there could be a negative correlation between these two variables), but there is also a huge cloud of points in the bottom left corner of the graph: any trend in that highly populated part could affect the overall correlation, and it is hard to tell whether that trend is positive or negative: everything is black there, we really can't tell how

most points distribute.

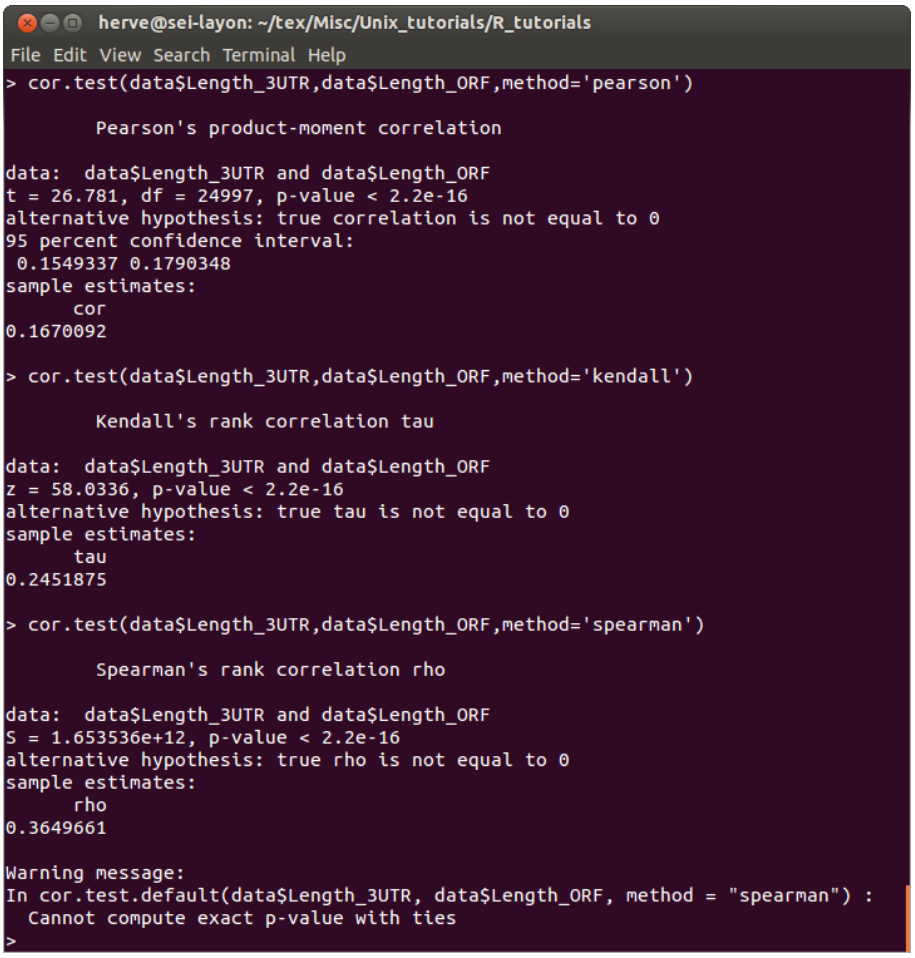

Figure 18: **Testing for correlation.**

It is sometimes hard to choose the most relevant method among the three proposed coefficient correlation methods ("Pearson's product-moment correlation", "Kendall's τ" and "Spearman's ρ"). Each of these three correlation coefficients ranges from -1 to +1 (with a coefficient = 0 meaning a total absence of correlation, +1 meaning a perfect positive correlation, -1 meaning a perfectly negative correlation, and intermediary values meaning a weaker positive or negative correlation). It is also important to pay attention to the associated p-value, which tells you how confident you can be about the measured correlation coefficient (the p-value measures the probability that the data is sampled from an infinite dataset whose correlation coefficient is 0). Here are a few tips that can help you pick the best suited method:

- Pearson's product-moment correlation: by construction, that method tests for linear correlation. Its correlation coefficient can only be -1 or +1 when the data points are perfectly aligned. It is possible that the two variables are strongly correlated, but without being proportional to each other (for example, y can be proportional to x^2 rather than to x): in that case, the calculated correlation coefficient won't be ± 1, despite a perfect correlation.

- Kendall's τ tests whether y is a monotonous function of y (*i.e.*: $\tau = +1$ means that every increase in x is accompanied by an increase in y, even though the increases are not

proportional; $\tau = -1$ means that every increase in x is accompanied by a decrease in y). This correlation coefficient has a very intuitive definition: τ is the number of concordant pairs of points, minus the number of discordant pairs, and that difference is divided by the total number of possible pairs of points.

- Spearman's ρ also tests whether y is a monotonous function of x. Its definition is less intuitive, but Spearman's ρ has the advantage of being more robust to noise (see figure 19).

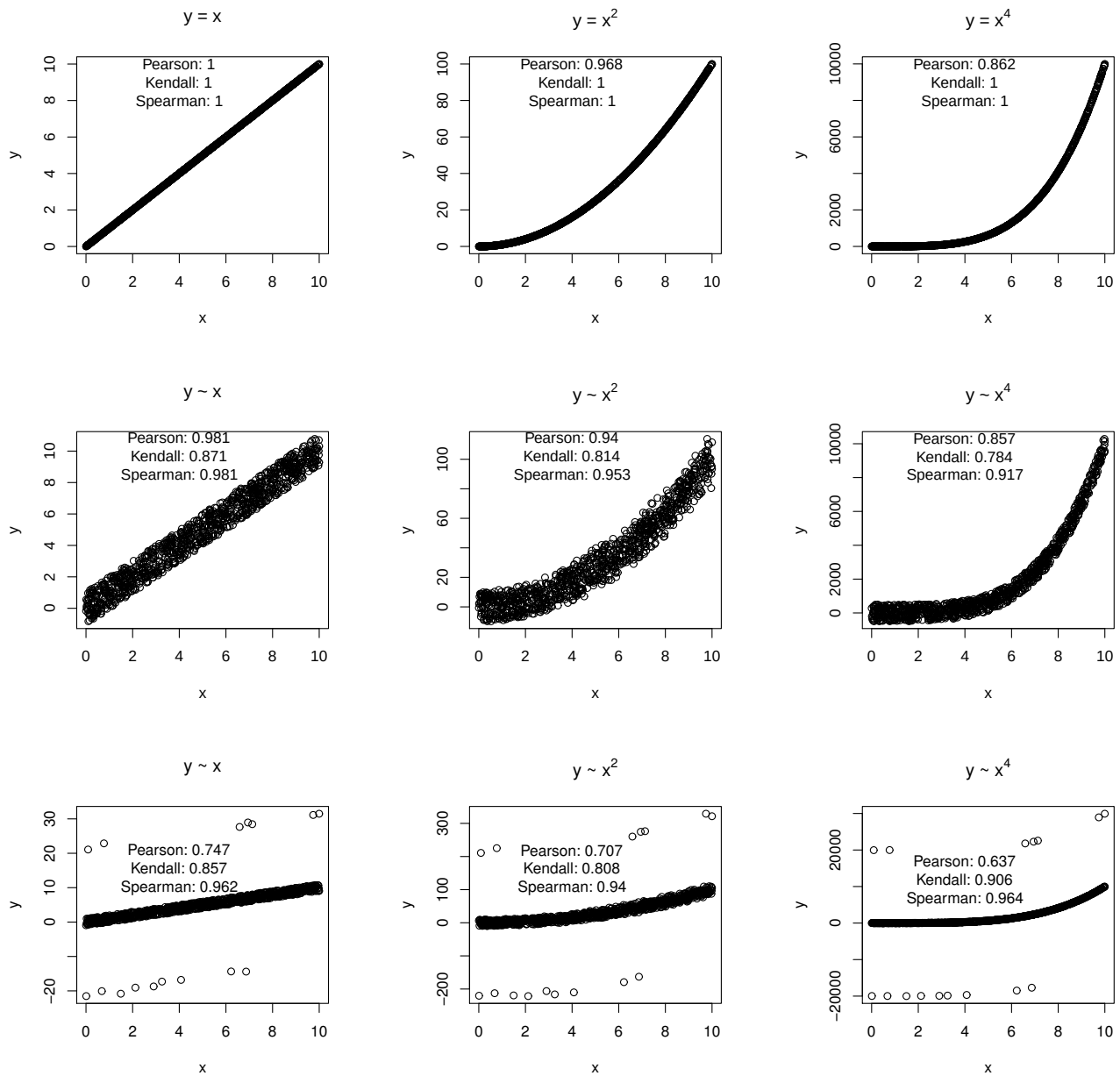

Figure 19: **Comparison of correlation methods on modeled datasets.** Top row: computer-generated datasets were generated, with $y = x$ (left panel), $y = x^2$ (middle panel) and $y = x^4$ (right panel). Correlation coefficients (according to the Pearson, Kendall and Spearman methods respectively) are displayed in each panel. Middle row: same thing, but some random noise was added to the data. Bottom row: same thing, but outliers (data points falling very far away from the rest of the population) were added. The R code used to generate these 9 panels is given in section 7.

7 Annex: generation of graphs for the comparison of correlation methods.

R commands used to generate figure 19:

```
x=runif(1000,0,10)
y=x
postscript('Correlation_illustration_1.ps',horizontal=FALSE,paper='special',width=4,
height=4)
plot(x,y,main=expression('y = x'))
pearson=cor.test(x,y,method='pearson')$estimate
kendall=cor.test(x,y,method='kendall')$estimate
spearman=cor.test(x,y,method='spearman')$estimate
text(4,0.9*max(y),paste('Pearson: ',signif(pearson,digits=3),'\nKendall: ',
signif(kendall,digits=3),'\nSpearman: ',signif(spearman,digits=3),sep=''))
dev.off()
y=x^2
postscript('Correlation_illustration_2.ps',horizontal=FALSE,paper='special',width=4,
height=4)
plot(x,y,main=expression(paste('y = ',x^2,sep='')))
pearson=cor.test(x,y,method='pearson')$estimate
kendall=cor.test(x,y,method='kendall')$estimate
spearman=cor.test(x,y,method='spearman')$estimate
text(4,0.9*max(y),paste('Pearson: ',signif(pearson,digits=3),'\nKendall: ',
signif(kendall,digits=3),'\nSpearman: ',signif(spearman,digits=3),sep=''))
dev.off()
y=x^4
postscript('Correlation_illustration_3.ps',horizontal=FALSE,paper='special',width=4,
height=4)
plot(x,y,main=expression(paste('y = ',x^4,sep='')))
pearson=cor.test(x,y,method='pearson')$estimate
kendall=cor.test(x,y,method='kendall')$estimate
spearman=cor.test(x,y,method='spearman')$estimate
text(4,0.9*max(y),paste('Pearson: ',signif(pearson,digits=3),'\nKendall: ',
signif(kendall,digits=3),'\nSpearman: ',signif(spearman,digits=3),sep=''))
dev.off()
y=x+runif(1000,-1,1)
postscript('Correlation_illustration_4.ps',horizontal=FALSE,paper='special',width=4,
height=4)
plot(x,y,main=expression('y ~ x'))
pearson=cor.test(x,y,method='pearson')$estimate
kendall=cor.test(x,y,method='kendall')$estimate
spearman=cor.test(x,y,method='spearman')$estimate
text(4,0.9*max(y),paste('Pearson: ',signif(pearson,digits=3),'\nKendall: ',
signif(kendall,digits=3),'\nSpearman: ',signif(spearman,digits=3),sep=''))
dev.off()
y=(x+runif(1000,-0.5,0.5))^2+runif(1000,-10,10)
```

```
postscript('Correlation_illustration_5.ps',horizontal=FALSE,paper='special',width=4,
height=4)
plot(x,y,main=expression(paste('y ~ ',x^2,sep='')))
pearson=cor.test(x,y,method='pearson')$estimate
kendall=cor.test(x,y,method='kendall')$estimate
spearman=cor.test(x,y,method='spearman')$estimate
text(4,0.9*max(y),paste('Pearson: ',signif(pearson,digits=3),'\nKendall: ',
signif(kendall,digits=3),'\nSpearman: ',signif(spearman,digits=3),sep=''))
dev.off()
y=x^4+runif(1000,-500,500)
postscript('Correlation_illustration_6.ps',horizontal=FALSE,paper='special',width=4,
height=4)
plot(x,y,main=expression(paste('y ~ ',x^4,sep='')))
pearson=cor.test(x,y,method='pearson')$estimate
kendall=cor.test(x,y,method='kendall')$estimate
spearman=cor.test(x,y,method='spearman')$estimate
text(4,0.9*max(y),paste('Pearson: ',signif(pearson,digits=3),'\nKendall: ',
signif(kendall,digits=3),'\nSpearman: ',signif(spearman,digits=3),sep=''))
dev.off()
outliers=sample(c(-1,0,1),1000,replace=TRUE,prob=c(1,98,1))
y=x+runif(1000,-1,1)
y=y+2*max(y)*outliers
postscript('Correlation_illustration_7.ps',horizontal=FALSE,paper='special',width=4,
height=4)
plot(x,y,main=expression('y ~ x'))
pearson=cor.test(x,y,method='pearson')$estimate
kendall=cor.test(x,y,method='kendall')$estimate
spearman=cor.test(x,y,method='spearman')$estimate
text(4,0.5*max(y),paste('Pearson: ',signif(pearson,digits=3),'\nKendall: ',
signif(kendall,digits=3),'\nSpearman: ',signif(spearman,digits=3),sep=''))
dev.off()
y=(x+runif(1000,-0.5,0.5))^2+runif(1000,-10,10)
y=y+2*max(y)*outliers
postscript('Correlation_illustration_8.ps',horizontal=FALSE,paper='special',width=4,
height=4)
plot(x,y,main=expression(paste('y ~ ',x^2,sep='')))
pearson=cor.test(x,y,method='pearson')$estimate
kendall=cor.test(x,y,method='kendall')$estimate
spearman=cor.test(x,y,method='spearman')$estimate
text(4,0.5*max(y),paste('Pearson: ',signif(pearson,digits=3),'\nKendall: ',
signif(kendall,digits=3),'\nSpearman: ',signif(spearman,digits=3),sep=''))
dev.off()
y=x^4+runif(1000,-50,50)
y=y+2*max(y)*outliers
postscript('Correlation_illustration_9.ps',horizontal=FALSE,paper='special',width=4,
height=4)
plot(x,y,main=expression(paste('y ~ ',x^4,sep='')))
```

```
pearson=cor.test(x,y,method='pearson')$estimate
kendall=cor.test(x,y,method='kendall')$estimate
spearman=cor.test(x,y,method='spearman')$estimate
text(4,0.4*max(y),paste('Pearson: ',signif(pearson,digits=3),'\nKendall: ',
signif(kendall,digits=3),'\nSpearman: ',signif(spearman,digits=3),sep=''))
dev.off()
```