Playing Data in R

PokMan HO

Contents

1 Motivation, Aims and Introduction

There are a lot of useful statistical materials scattered in the internet. This short note is aimed at summarizing the most important information and provide extensive links to these useful materials for readers. All links and quoted materials in this note are all open access (no "free sign-up" or information behind paywall) to the general public. For scientific materials, please consider access through sci-hub (domain se or tw, depending on your IP location), which is scripted by a Russian scientist. The author share the same view on scientific knowledge access right with this scientist.

The author attempt to minimize the use of external packages, demonstrating the effectiveness and power of the base R[1] environment (ver 3.6.0). By reading this note, the author assumes readers:

1. have basic understanding on R commands and syntax (e.g. know R's extensive use of brackets and commas, indentations are for human readability only...)
2. know that R is effective at doing columnize / rowwise operations
3. understand csv data structure (e.g. columns and rows indication; differences between data.frame and matrix...)

For readers just started, some materials can be found here

In this note R codes would be encapsulated in the following format:

```
sample code
# comment (description for the above code, if necessary)
# if the code does not have library(pkg) part, it is from the core R
```

2 Choosing statistical test(s)

This step should ideally be planned before collecting any data, with details of how to choose such test(s) explained here or here. Basically considering aspects are summarized in a few points (each variable has one of this decision):

- data type: categorical (ordered or unordered) or numerical?
- x-y pairing: which data type(s) are combining in your hypothesis?
- how many variables (independent and dependent are grouped separately) are considered in one test?
- data value distribution (decide after data collection): normally-distributed (parametric) or skewed (non-parametric OR data transformation then parametric)?

Remember: regardless how absurd the data looks, there is always an explanation to the pattern. Hence there is always no need to remove any data (unless the data is justified as low-quality data), although data removal is usually an easy path to go.

Below is a list of frequently used tests with parametric equivalent in round brackets and conditions as description:

- Kruskal (ANOVA / two-way ANOVA): numeric dependent, categorical independent with internal group(s)
- Wilcoxon (paired-t): numeric dependent, two numeric independents
- Spearman (Chi-Sq): numeric dependent, numeric independent

Since many biological data are found log-normal, log data transformation is justified. Other than that, it is not recommended doing data transformation due to the loss of data structure after such an operation.

3 Administrative issues

3.1 R environment

It is crucial to clean the environment just to make sure none of the previous operations have any chance to mess up your current code, making your analyses non-reproducible.

```
rm(list=ls(all=T))
#rm all

rm(list=ls(pattern="temp"))
#rm data names contain pattern "temp"
```

3.2 Sample data in this note

The author chose to use arbitrary data for demonstration purposes. Codes for generating such data sets were showed and explained below:

One numeric dependent + two categorical unordered independent + one numeric independent

```
set.seed(998)
# fix random number generation algorithm, for code reproducibility

a<-as.data.frame(matrix(nrow = 1e3, ncol = 4))
# initialize empty dataframe

colnames(a)=c("dep", "indep_1", "indep_2", "indep_3")

a$dep<-runif(nrow(a))
# sample number of values meet number of rows of dataframe a from uniform distribution between
0 and 1

a$indep_1<-c("a","b")
# repeating "a", "b" pattern until fill up the dataframe (length difference must be in multiples)

a$indep_2<-sample(c("a","c","d","e"),nrow(a), replace = T)
# sample number of values meet number of rows of dataframe a from designated pool with element
replacement after each element selection

a$indep_3<-rnorm(nrow(a))
# sample number of values meet number of rows of dataframe a from normal distribution with mean
0, sd 1
```

3.3 Colours

Colours used in this note were colour-blinded friendly, collected from various sources in the internet:

```
cbp <- c("#000000", "#E69F00", "#56B4E9", "#009E73", "#0072B2", "#D55E00",
"#CC79A7", "#e79f00", "#9ad0f3", "#F0E442", "#999999", "#cccccc", "#6633ff", "#00FFCC",
"#0066cc")
```

4 Data subsets

This is an important issue when you are given a huge data contaminated by useless data. Apart from the usual and straight-forward “subset" command, a filter command “which" (in “base" package[1]) is much more powerful. The code is:

```
a.0<-a[which(a$indep_2=="a" & a$indep_1!="a"),]
a_1<-a[which(a$indep_2=="a" | a$indep_1!="a"),]
# select whole rows in data.frame a which its columns indep_2 is "a" and/or indep_1 is not "a"
```

Chaining up these filter operators make data subset operation faster, quicker and more efficient. However, it is generally recommended not to mix “&" and “|" unless you’re very sure about what is going on with these filters. Separating these two filter operators generally make your code more readable and hence, potentially increase your work efficiency in the long run.

5 Non-parametric tests

Non-parametric tests are powerful because they can fit any data distribution types. However due to their intrinsic data handling method (ranks are used instead of actual data), only the data order can be preserved. So test result cannot be used to predict anything out of the reach of the data coverage.

Remember to confirm (in case it’s necessary) your independent variables as text string data type (the most flexible data type) before doing anything to them, as usually strings, by default, identified as “factor" data type by R (which is stored as a number).

```
a$indep_3<-as.character(a$indep_3)
# column conversion from any data to text string data type
```

5.1 Kruskal test

This test is very flexible, and is primarily for categorical independent variables. It is testing the rank differences (i.e. “variances" in parametric terms) between groups. There are two ways to use this code (in “stats" package[1]) - one dependent test against one or multiple independent variables. Since the code strictly accepts one independent variable, we fit its syntax limitation by combining variables together to make a single combined variable.

For one independent variable,

```
kruskal.test(a$dep ~ a$indep_1)
# y against x
```

For multiple independent variables,

```
kruskal.test(a$dep ~ interaction(a$indep_1, a$indep_2, a$indep_3))
# y against combined independent variables x1, x2, x3...
```

If kruskal.test result showed significance, that means there were at least one pairwise rank difference having significant difference. To find out which exact pair(s) are standing out from the crowd, we need a post-hoc test. I used Nemenyi pairwise comparison (in "PMCMR" package[2]), which does not require p-value correction[2].

For one dependent and one independent variables,

```
library(PMCMR)

posthoc.kruskal.nemenyi.test(a$dep ~ as.factor(a$indep_1))
# meet code requirement, convert independent variable to factor data type (if necessary)
```

For one dependent and multiple independent variables,

```
library(PMCMR)

posthoc.kruskal.nemenyi.test(a$dep ~ interaction(a$indep_1, a$indep_2, a$indep_3))
# "interaction" code already converted its content to factor data type, so there's no need to convert
as we've done for the above step
```

5.2 Wilcoxon test

This test can only test the rank mean difference between two groups, and the code (in "stats" package[1]) is

```
wilcox.test(a$dep ~ a$indep_1)
# y against x, with x only have two levels
```

5.3 Spearman correlation

This test is a test for correlation, which is only useful on one dependent and one independent variable. Both must be numbers ("numeric" or "integer" data type).

Data type conversion (if necessary)

```
a$indep_3<-as.numeric(a$indep_3)
# column conversion from any data to numeric data type
```

Spearman correlation code (in "stats" package[1])

```
cor.test(a$indep_3, a$dep, method="spearman")
# x, y, method of testing
```

6 Parametric tests

Parametric tests are less powerful than non-parametric tests in terms of data flexibility (as all of them assumes normal-distributed data). However within its limitation, all of them are powerful than their non-parametric equivalent. It is due to the linear model property, which a straight line can be extended to infinity on x-axis and still give a meaningful corresponding y-value.

The universal match of this type of test is the use of code (in "stats" package[1])

```
summary(lm(a$dep ~ a$indep_1 + a$indep_2 * a$indep_3^2))
# a basic summary of all parametric methods introduced in this note - they are all linear models
# this code in a summary of a linear model with a formula of y against independent x1 and interactive x2 and x3-squared
# resemble linear formula: y = k1 · x1 + k2 · x2 · x3^2
```

6.1 ANOVA

The full name is ANalysis Of VAriance, which has five assumptions. The most important ones are "the data is normally-distributed" and "variance between groups share the same value".

The code (in "stats" package[1]) for one-way ANOVA is

```
summary(aov(a$dep ~ a$indep_2))
# get a summary form of the y against x ANOVA test
```

The same code can also use for n-way ANOVA (here I demonstrate n=3)

```
summary(aov(a$dep ~ a$indep_1 + a$indep_2 * a$indep_3^2))
# same formula as the above linear model, just using the aov approach
```

6.2 Paired-t test

This test is for testing mean difference between two normally-distributed samples. The code (in "stats" package[1]) is

```
t.test(a$dep ~ a$indep_1, paired=T)
# y against x, which x only have one or two levels
```

6.3 Chi-Sq correlation

This test is for correlation on normally-distributed numeric dependent and independent variables. The code (in "stats" package[1]) is

```
cor.test(a$indep_3, a$dep, method="pearson")
# same with spearman above, only switched the method parameter
```

7 Data Normality

If you want to use parametric tests on your numeric data, this part is super-important and should be the first test to be carried out before going onto the statistical test itself. There are three ways to test for data normality and your data have to pass at least two (just a convention, the majority rules) to gain a "normal-distributed" data status.

7.1 Shapiro test

This is the most convenient and objective way to test for numeric variable normality. However in R there is a limitation: the number of data can only be between 3 and 5000. This maybe one of the few occasions you'd like to see non-significant p-value.

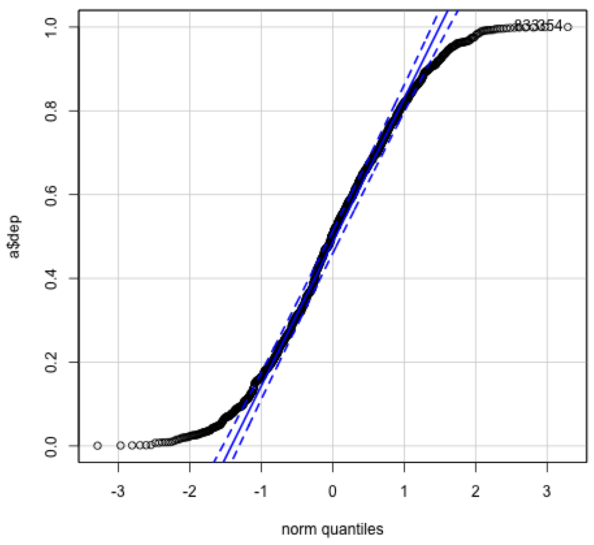
The code (in "stats" package[1]) is

```
shapiro.test(a$dep)
# testing hypothesis (p < 0.05): the data is skewed
# Hence null hypothesis (p-val > 0.05): the data is not skewed (i.e. normally-distributed).
```

7.2 qq-plot

```
library(car)
qqPlot(a$dep)
# indicated mean with 95% interval
```

There is also a base R version of qq-plot. However the base R version requires more lines of code and lack a 95% confidence interval for judging normality, hence it is not effective and efficient to spend time on coding it. Below is a sample of the qqPlot from "car" package on normality of the dependent variable of our sample dependent variable data:



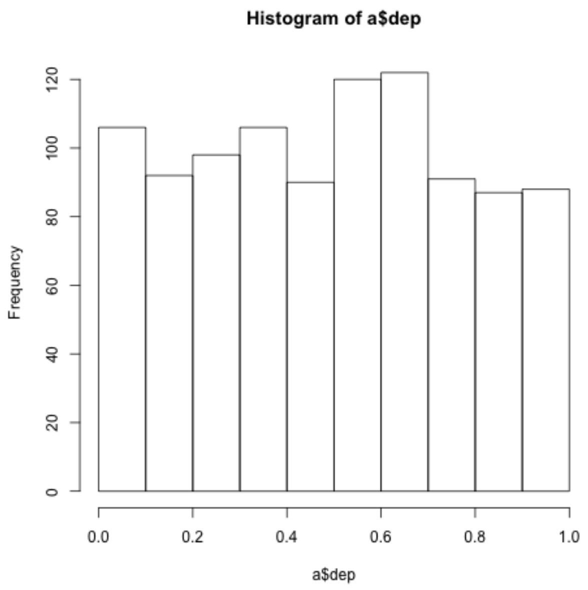
In the above graph, the solid line is the mean and dashed lines indicate the 95% confidence interval of that mean based on the input data. Data is called "normally-distributed" only when all plotted points are within the 95% confidence interval. Hence our dependent variable in the sample data is not normally-distributed judged by the qqPlot.

7.3 Histogram

This is the easiest method. However it requires you to interpret and judge the normality of data (subjectivity & biases??). So generally it is not recommended by the author unless one or more methods listed above cannot function as it should be, or they disagree with each other. As we all should have known, histogram are plotting frequencies on continuous numeric values. Thus the variable must be numeric data type [use as.numeric(variable) command if needed]

The code (in “graphics" package[1]) is

```
hist(a$dep)
```



In the above graph, if it does not look like normally-distributed, the data is considered as “not normally-distributed".

8 Plots

R is built by statisticians for data, so plotting in R is easy, flexible and powerful (as long as you know how to tweak parameters with correct grammar, which make the previous statement =trash). So below are some probably handy commands from the base R environment, hopefully assisting you to make R work for you.

8.1 Plot by group, data in wide format

For plotting number-number plots (plotting y∼x, z∼x, ... together), this code from “graphics" package[1] is particularly useful.

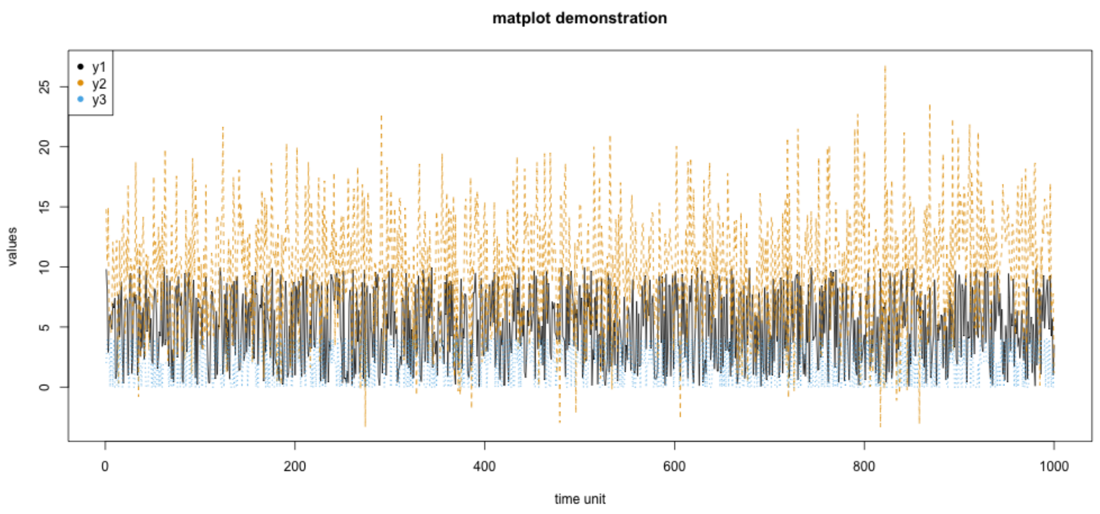
To demonstrate it, we need a suitable wide table format numeric data as below:

```
b<-seq(1e3)
# get a number sequence starting from 1, having 1e3 number of elements

b<-data.frame("time"=b, "y1"=runif(length(b))*10, "y2"=rnorm(length(b), mean = 10, sd = 5),
"y3"=sample(c(0,2,4), length(b), replace = T))
# define a data.frame with content in one go
# incorporate the sequence into data frame ("time" col)
# generate random numbers with elements same with length of the sequence ("y1" col)
# same number of elements, from a normal distribution ("y2" col)
# sample that number of elements from the given pool with element replacement ("y3" col)
```

Then plot with

```
matplot(x=b$time, y=b[,-1], type="l", col = cbp, xlab = "time unit", ylab = "values", main="mat-
plot demonstration")
# x-axis
# y-elements, columnwise excluding first column
# line-type plot
# line/point colour according to pre-defined colour-palette
# manual set x-label
# manual set y-label
# manual set chart title

legend("topleft", legend = c("y1", "y2", "y3"), pch = rep(16,3), col = cbp)
# use a pre-defined legend position
# group labels
# dot style indicating group in the graph
# colour of dots indicating for the groups
```

And the plot look like this:

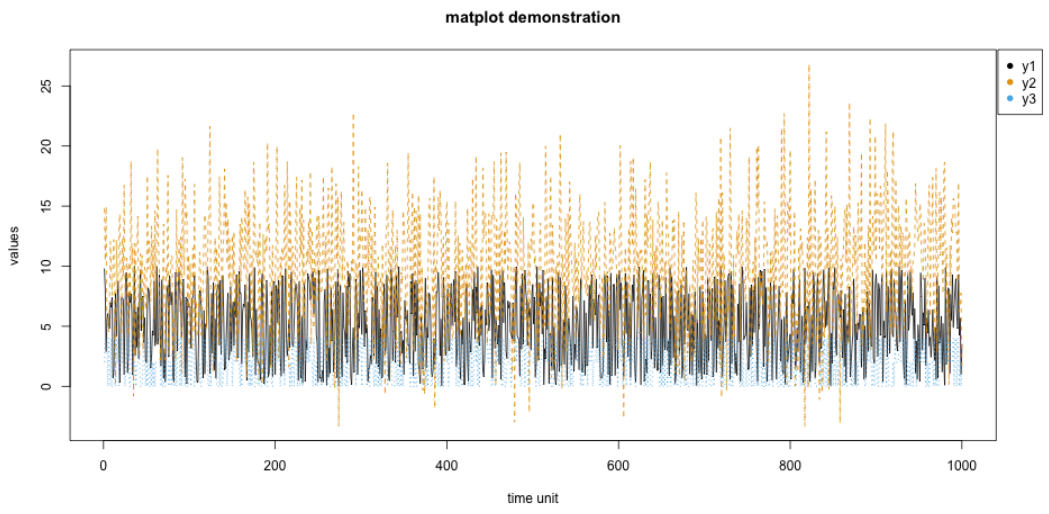


8.2 Legend position out of plot area

Using the same matplot, we can tweak a bit and make the legend outside of the plot area

```
par(mar=c(5.1, 4.1, 4.1, 8.1), xpd=TRUE)

matplot(x=b$time, y=b[,-1], type="l", col = cbp, xlab = "time unit", ylab = "values", main="matplot demonstration")

legend("topright", inset=c(-.05,0), legend = c("y1", "y2", "y3"), pch = rep(16,3), col = cbp)
# inset: x-shift, y-shift (positive as right and up)
```

And the plot look like this:



9 Non-Linear Least Square (NLLS) Method

When you data is not having linearity anywhere (e.g. growth, stress performance, functional response), you cannot use linear models (neither non-parametric nor parametric). Hence if you want to explain your data with a published model, you can only try to tweak model parameters and make the model fit onto your data. There are three ways (NLLS, maximum likelihood [ML] and Bayesian Inference [BI]). This section will be only including methods taught in the “Computational Methods in Ecology and Evolution" course at Silwood Park, original notes here.

The logic is simple:

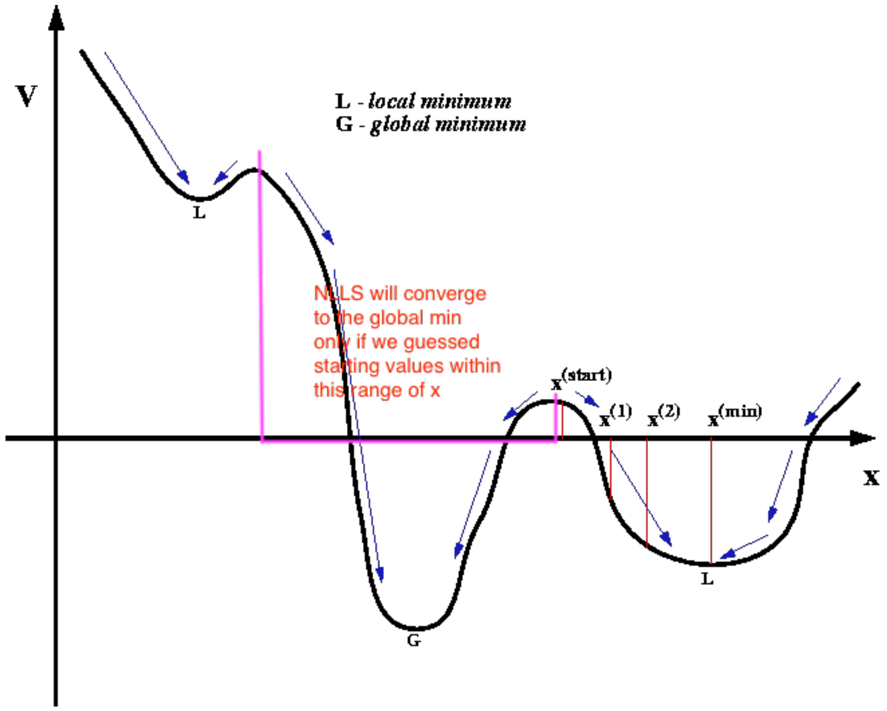
1. Identify models of interest & parameters you need to fit
2. Try your best to estimate a value for each parameter (use recursive linear model and R^2 values to judge for you if needed)
3. Use a distribution (either normal or uniform) to sample values around your guesstimated one
4. Try to fit those sets of sampled parameter values (i.e. starting values) in your model on your data (run R function)
5. Use AIC to extract the “best guess" group among all the sampled parameter values

This approach has some assumptions (of course, as always)

- You try your best at guesstimating the values for sampling
- You did a good job in guesstimating values that the computer can tweak bit of the starting values, resulting to the overall best fit (i.e. global minimum of model fitting)

- if you use a normal distribution, it even mean that your guess is very close to the “true value"

As you can see, these assumptions are based on “you", because only human commit errors. However computers are stubborn. Once they have trials that converge to the same stable parameter value (i.e. local minimum), it assumes that convergence = best value. As we know that local minimums can have values far from the global minimum, we have to be sure that our guess can let the computer end its job at the global minimum (modified visualization from original source below).


V
L - local minimum
G - global minimum
L
NLLS will converge to the global min only if we guessed starting values within this range of x
$x^{(start)}$
$x^{(1)}$
$x^{(2)}$
$x^{(min)}$
x
G
L

References

1. R Core Team. *R: A Language and Environment for Statistical Computing* R Foundation for Statistical Computing (Vienna, Austria, 2019). https://www.R-project.org/.
2. Pohlert, T. *The Pairwise Multiple Comparison of Mean Ranks Package (PMCMR)* R package (2014). https://CRAN.R-project.org/package=PMCMR.
3. Fox, J. & Weisberg, S. *An R Companion to Applied Regression* Second. http://socserv.socsci.mcmaster.ca/jfox/Books/Companion (Sage, Thousand Oaks CA, 2011).