Statistics and Data Analysis in Physical Geography

Practical exercises 7

Multivariate Analysis

Geert Sterk, Niko Wanders, Philip Kraaijenbrink

Department of Physical Geography
Geosciences Faculty
Utrecht University

2023

7.1 Principal components

Principal component analysis (PCA) tries to find major direction of variability, or, after scaling the variables, major directions of linear correlation in the data. PCA can be seen as an exploratory technique that investigates multiple linear correlations among variables, without making the distinction between dependent and independent variables (unlike e.g. linear regression and discriminant analysis). In that sense the technique is sometimes called "unsupervised" – we don't predict one specific variable from others and can't measure its predictive capacity from looking at residuals from a single variable.

Principal components is not a statistical technique in the sense of inference, testing, confidence intervals etc; it is rather a way to see whether sets of variables contain redundant information and can be summarized by a smaller set of new variables, called principal components. In some sense you can explain PCA in terms of a projection: from *n* correlated variables, without loss of information the new set of *n* PC are new uncorrelated axes, that explain a decreasing amount of the total variability present. The projected values of the data are called *PC scores*, the rotation matrix to go from the data space to the PC space has coefficients called *loadings*.

7.1.1 Principal component analysis workflow practice

Load the file TABLE612.TXT from Davis book from the website in R, similarly as in the regression exercise 6.

```
# load the tidyverse packages
library(tidyverse)

# load the dataset
table612 = read_delim(
  file      = 'https://www.kgs.ku.edu/Mathgeo/Books/Stat/ASCII/TABLE612.TXT',
  delim     = "\t",
  )
```

To perform a PCA using matrix calculations by R, conduct the following steps. Check the intermediate outputs.

1. Make a scatter plot of the `x1` and `x2` data in table612 using `ggplot`
2. Create a covariance matrix of table612 (use function `cov()`),
3. Calculate the eigen values and vectors from the covariance matrix (use function `eigen()`) and store it as new variable `eigen_table`,

4. Project the data on the two principal components (see also lecture notes 7, Eq. 7.6). To do this in R:

    a. First change the class of table612 to matrix by using
    `X = as.matrix(table612)`

    b. `U = eigen_table$vectors`

    c. Use the function `%*%` to perform matrix multiplication in R (see https://r-coder.com/matrix-operations-r/ for more information on matrix calculations in R). Using a regular `*` would perform an item wise multiplication instead.
    `S = X %*% U`

5. Plot the first two columns of the matrix (`X[,1:2]`) and the transformed data (`S[,1:2]`) using the regular `plot` function of R.

6. Now project the data on the first principal component only using (Eq. 7.7):
```
U1 = U[,1]
S2 = X %*% U1
```

7. Plot the projected data:
`plot(S2, rep(0,25))`

8. Create a new dataset of x1 and x2 with transformed data (reduced variance) using Eq. 7.9 from the lecture notes. The function `t()` allows you to transform a matrix:
`Xnew = S2 %*% t(U1)`

9. Make again a plot of the x1 and x2 data
`plot(Xnew)`

7.1.2 Principal component analysis on a test data set

Satellite imagery is crucial for various applications, but its high dimensionality can sometimes be challenging. Principal Component Analysis (PCA) is therefore a powerful tool that reduces this complexity by extracting key information from satellite images. In this exercise, you'll explore how PCA can help in satellite image analysis for feature extraction, noise reduction and data reduction. In this exercise, you will work with an ASTER satellite image of the Almere area in the Netherlands and apply PCA analysis to the data.

Create a new folder on your computer for this analysis, create a subfolder "data", and download the data set **"`aster_flevo.tif`"** (on blackboard) into it. Next, install the terra package to deal with spatial grids using `install.packages()`. Create a new script and start by loading the tidyverse and terra packages.

Load the ASTER image in R by using:

```
aster = rast('data/aster_flevo.tif')
```

The ASTER image consists of 6 different bands, or layers, that all have the reflectance of a different wavelength:

band1 = 560 nm (Green)
band2 = 660 nm (Red)
band3 = 820 nm (Infrared)
band4 = 1650 nm (Infrared)
band5 = 2165 nm (Mid Infrared)
band6 = 2205 nm (Mid Infrared)

Some bands show more information than others. Using PCA, you will evaluate which bands are most relevant to make an image classification, and how you can reduce the total information of all bands and remove information that is redundant.

First, start by having a look at the data by running `print(aster)` in the console to see the image information.

A satellite image basically consists of a stack of matrices describing the pixel values where the pixels are associated with specific coordinates. The dimensions are indicated in the image information under "dimensions". Have a look at the statistics of each band or matrix using

```
summary(aster)
```

Next, have a look at a false color composite of the satellite image showing the near infrared, red and green bands by running the following command:

```
plotRGB(aster, r=3, g=2, b=1, stretch='lin')
```

Now, have a look at all bands separately in grayscale using

```
plot(aster, col=gray.colors(255))
```

Exercise 7.1.
Which band do you think is the best option to classify land use types and why?

In the next step look at the correlations between the different bands. To do this, the data first has to be transformed. It is now in a stack of matrices, but to calculate the correlation we need a matrix where the rows describe each observation (i.e. each image pixel) and the columns describe the variables (i.e. each image band). To transform the image to such a matrix use and get a matrix of the coordinates for each row, use:

```
aster_table = as_tibble(aster)
aster_coord = as.data.frame(aster, xy=T)[,1:2]
```

Have a look at the table using print(aster_table). Does this meet your expectations? How many observations are there?

Now apply the correlation function `cor` to the data.

Look at the correlations between the bands and create two scatter plots using `ggplot2`, one for the two bands with the lowest correlation and one for the two bands with the highest correlation. Since there are many observations, plotting all points will test your system so take a sample of the data instead, e.g. 10000 observations. You can use the tidyverse function `sample_n()` to sample a specific amount of rows.

Additionally, also make a plot between NIR and SWIR1 and inspect its output.

Exercise 7.2a
Looking at the variances in the NIR band and SWIR1 band, which one has the largest variance?

Exercise 7.2b
If we would only consider these two bands, how would you describe the first and the second principal component?

Principal components analysis (PCA) is a method in which original data is transformed into a new set of data which may better capture the essential information. Often some variables are highly correlated such that the information contained in one variable is largely a duplication of the information contained in another variable. Instead of throwing away the redundant data, PCA condenses the information in inter-correlated variables into a few variables, called principal components (PC's).

So, PCA is a decorrelation procedure that reorganizes by statistical means the reflectance values from as many of the spectral bands as we choose to include in the analysis. In this case we will include all six bands. Through PCA we will create a new orthogonal coordinate system (the principal components), which can be used to transform the original data into new, uncorrelated data. Usually only a few principle components will be sufficient to capture the essential information.

In this case, the six bands have all the same unit (i.e. reflectance) and so we can use the covariance matrix. If they would have different units, one chooses the correlation matrix instead. Have a look at these outputs:

```
cov(aster_table)
cov(scale(aster_table))
cor(aster_table)
all.equal(cor(aster_table), cov(scale(aster_table)))
```

The scale function subtracts the mean and divides by the standard deviation, so the result has mean zero and unit standard deviation (and variance). Note that the covariance of the scaled data equals the correlation of the data. This means that if we work with covariances of scaled data, we effectively work with correlations of the (unscaled) data.

Exercise 7.3
What do the values on the diagonal of the covariance matrix represent?

a. covariances between two different variables
b. variance of a single variable
c. standard deviation of a single variable
d. mean value of a single variable

Carry out a principal component analysis manually on all six variables. First calculate the eigen vectors (use command "eigen") from the covariance matrix. Create the matrix U which contains all six eigen vectors, similar as how you have done in the workflow above.

Exercise 7.4
How much of the total variance is explained by PC1? PC2? PC3? and PC4?

Also, similar to the workflow before, convert the table to a matrix and project the original data on all six PC's by running the commands:

```
X = as.matrix(aster_table)
S = X %*% U
```

Create a correlation matrix of S and look at the correlations between the projected data.

Use the projected data in combination with the coordinates stored earlier to convert the matrix with transformed observations back into a 2D image using the code below. Create a plot of the projected data on the first PC and compare the result with the "best" band you have selected under exercise 7.1.

```
# convert to raster image
pc_img = rast(setNames(bind_cols(aster_coord, S), c('x','y',paste0('PC',1:6))))

# make plot of PC1
# (note, we take negative of image for visual purposes here)
plot(-pc_img[[1]], col=gray.colors(255, 0, 1))
```

Exercise 7.5a
What do you conclude about the difference or similarity between these two images?

Exercise 7.5b
Look at the loadings of PC1 (in matrix U). Which band is the most important for PC1? Is this answer in agreement with your previous answer?

Now create plots of the projected data on all PCs using

```
plot(pc_img, col=gray.colors(255, 0, 1))
```

And look at the summary statistics of the PC matrix

```
summary(S)
```

Exercise 7.6
Which PC's contain useful information in your opinion?

Convert the projected data S back to the original values (by using the inverse of U, which is U^T and calculated by `t(U)`) and store it as `X_back`. Compare the converted data with the original data:

```
all.equal(X[,1:5], X_back[,1:5], check.attributes=F)
```

Is the result what you expect?

Now, project the original data on PC1 only (also see workflow) and call the result `S1`. Convert it to a raster image again by adapting the code for exercise 7.5a. Inspect the projected image and compare it with the previous projected data on PC1 created for exercise 7.5a.

Now transform the data back to the original values and call the result `X1`. Compare the values of the new matrix with the original matrix `X` and compare the summary statistics using the `summary()` function.

Finally, convert the matrix to a raster again, plot it as the same color image as you did in the beginning of this exercise.

```
# create image from the matrix and check results
x1_img = rast(setNames(bind_cols(aster_coord, X1), c('x','y',names(aster_table))))

# plot false color
plotRGB(x1_img, r=3, g=2, b=1, stretch='lin')
```

Also plot the bands separately again:

```
# plot all bands separately as grayscale
plot(x1_img, col=gray.colors(255, 0, 1))
```

Exercise 7.7
What can you say about the two matrices? Are they equal or different. If they are different, what is the reason for this?

Repeat all the same steps as for Exercise 7.7, but now use the first three PC's in the transformation instead of just the first one.

Exercise 7.8a
What has happened to the projected and then back-converted data when three PC's are used? Is there a loss of information?

Exercise 7.8b
What is the advantage in terms of data storage when three principle components are used?

You have calculated the matrix, eigenvectors and eigenvalues "by hand" in the exercises above, but R also has an automatic routine for PCA using the function `prcomp()`. Apply a PCA to the satellite data using prcomp

```
# perform PCA with dedicated function
pca_from_r = prcomp(aster_table, scale=F)
pca_from_r
```

Exercise 7.9
How do we calculate eigenvalues for principal components from the "standard deviation" of each component?

- taking the square root of the standard deviation
- eigenvalues are equal to the standard deviations
- by taking the square of the standard deviation

Note that the results of the prcomp are almost the same as when calculating the PC's manually, but the difference is that the `prcomp` command centres the projected data around the mean. It stores the centering (and scaling parameters when selected) as well as the transformed matrix as separate subitems, accessible using the dollar sign:

```
pca_from_r$center
pca_from_r$scale
pca_from_r$x
```

Depending on the purpose of the image classification, different combinations of PC's can be made to obtain maximum separation between different feature classes. For instance, we could combine PC1 and PC2 in one image, where we plot PC1 as green and PC2 as red, but other combinations could be tried as well. However, in actual image classification often the real frequency bands are used instead of the projected data on the PC's. Only in case the image contains a lot of noise there can be a real advantage when the classification is done on the PC transformed data.

An example of plotting a PC image on multiple color channels could be:

```
plotRGB(pc_img, r=1, g=2, b=3, stretch='lin')
```

VOLUNTARY BONUS EXERCISE
Discriminant Function Analysis (DFA)

Note this exercise uses base R functionality, and was not adapted for the tidyverse

DFA is a multivariate technique for describing a mathematical function that will distinguish among predefined groups of samples. As an eigenvalue-eigenvector method, DFA has a strong connection to multiple regression and principal components analysis. In addition, DFA is the counterpart to ANOVA: in DFA, continuous variables (measurements) are used to predict a categorical variable (group membership), whereas ANOVA uses a categorical variable to explain variation in (predict) one or more continuous variables. Two examples help to show how DFA can be useful.

For the first example, suppose you have a series of morphological measurements on several species and want to know how well those measurements allow those species to be distinguished. One might try to perform a series of t-tests or ANOVAs to test for differences among the species, but this would be tedious, especially if there were many variables. One might also try a principal components analysis (PCA) to see how the groups plot in multidimensional space, and this is often a good exploratory approach. DFA takes a similar approach to the PCA but seeks a function that will maximize the differences among the groups. The function will show how well the species can be distinguished, as well as where the classification is more robust and where it is more likely to fail.

In the second example, suppose you are studying ancient artifacts that are thought to have come from a set of mines. You have collected a set of rock samples from those mines and have made a consistent set of geochemical measurements on those samples. You have also made those measurements on the artifacts. DFA can be used to find a function that uses your geochemical measurements to separate your samples into the mines from which they came. That function can then be applied to the artifacts to predict which mine was the source of each artifact.

How it works

There are several types of discriminant function analysis, but this exercise will focus on classical (Fisherian) discriminant analysis, which is the one most widely used. In the simplest case, there will be two groups of data. If there are three variables in your data set (x, y, z), DFA will find an equation that maximizes the separation of the two groups using those variables. The discriminant function has the following form:

$$a(x - \bar{x}) + b(y - \bar{y}) + c(z - \bar{z})$$

where *a*, *b*, and *c* are the coefficients (slopes) of the discriminant function. Each sample or case will therefore have a single value called its score.

An example

Our example uses a data set of geochemical measurements on water samples from wells (data are in BRINELN.CSV and are from Davis (2002) “Statistics and Data Analysis in

Geology", they are available on blackboard). The data set is in standard form, with rows corresponding to samples and columns corresponding to variables. Each sample is assigned to one of three stratigraphic units, listed in the last column ("GROUP"). Because DFA assumes multivariate normality, the data had to be checked to make sure that there are no strong departures from normality before performing the analysis. In this case a log transformation was needed in order to assure near-normality of the data. Such log transformations are quite common for geochemical data. In the file BRINELN the transformation has already been done.

```
brine = read.table('BRINELN.CSV', header=TRUE, sep=',', row.names=1)
head(brine)
```

The **pairs()** function is useful for making cross plots of the data. The cross-plots should only compare the measurement variables in columns 1-6, because the last column is the group name:

```
pairs(brine[ ,1:6])
```

The cross plots show clearly how well the different geochemical variables are related to each other, with some variable having a high correlation (e.g. NA and CL) while others are poorly correlated (e.g. HCO3 and CL).

Discriminant Function Analysis

The discriminant function analysis is performed with the **lda()** function of the MASS packages, which needs to be installed first:

```
install.packages('MASS')
library(MASS)
brine.lda = lda(GROUP ~ HCO3 + SO4 + CL + CA + MG + NA, data=brine)
```

The format of this call is much like a linear regression or ANOVA in that we specify a formula. Here, the **GROUP** variable should be treated as the dependent variable, with the geochemical measurements as the independent variables. In this case, no interactions between the variables are being modelled. The name of the data frame must be supplied to the **data** parameter. After running the DFA, the first step is to view the results:

```
brine.lda
```

The first part of the output displays the formula that was fitted. This is followed by the prior probabilities of the groups, which reflects the proportion of each group within the dataset. In other words, if you had no measurements and the number of measured samples represented the actual relative abundances of the groups, the prior probabilities would describe the probability that any unknown sample would belong to each of the groups. Next comes the group means, which is a table of the average value of each of the variables for each of your groups. Scanning this table can help you to see if the groups are distinctive in terms of one or more of the variables. Next are the coefficients of the discriminant function (a, b, and c). Because there are three groups, there are 2 [3-1] linear discriminants (if you had only two groups, you would need only 1 [2-1] linear discriminants). For each linear

discriminant (LD1 and LD2), there is one coefficient corresponding, in order, to each of the variables. Last in the list is the proportion of the trace, which gives the variance explained by each discriminant function. Here, discriminant 1 explains nearly 70% of the variance, with the remainder explained by discriminant 2.

Use the DFA to classify the data

The **predict()** function, also part of the MASS package, uses the **lda()** results to assign your samples to the groups. In other words, since **lda()** derived a linear function that should classify the groups, **predict()** allows you to apply this function to the same data to see how successful the classification function is. Following the statistical convention that $\hat{x}$ is the prediction of x, .hat is added to the object name to make it clear that these are the predictions:

```
brine.hat = predict(brine.lda)
brine.hat
```

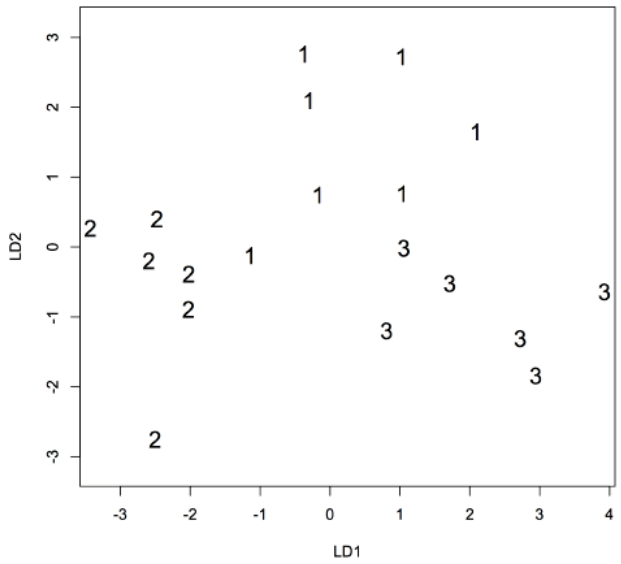
The output starts with the assigned classifications of each of our samples. Next it lists the posterior probabilities of each sample to each group, with the probabilities in each row (that is, for each sample) summing to 1.0. These posterior probabilities measure the strength of each classification. If one of these probabilities for a sample is much greater than all the others, that sample was assigned to one group with a high degree of certainty. It two or more of the probabilities are nearly equal, the assignment is much less certain. If there are many groups, the following command is a quick way to find the maximum probability for each sample:

```
apply(brine.hat$posterior, MARGIN=1, FUN=max)
```

For this data set, most of these maximum probabilities are large (>0.9), indicating that most samples are confidently assigned to one group.

If most of these probabilities are large, the overall classification is successful. The last part of the **predict()** output lists the scores of each sample for each discriminant function axis. These scores can be plotted to show graphically how your groups are distributed in the discriminant function, just as scores from a principal components analysis could be plotted.

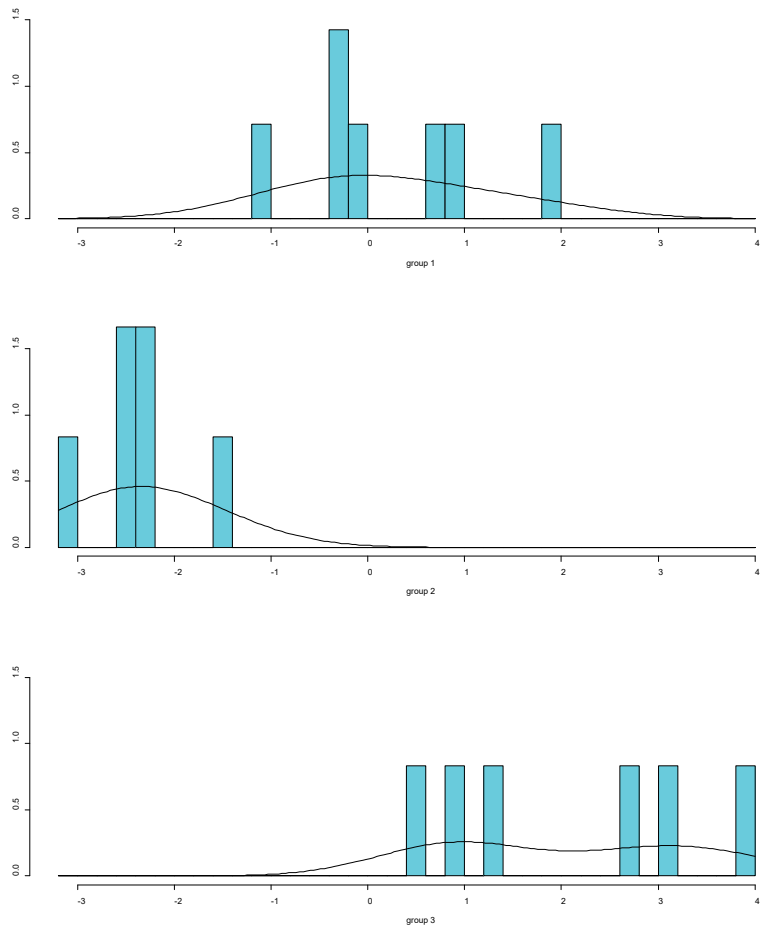
```
plot(brine.lda)
```



In this plot, the three groups occupy distinctly different and non-overlapping regions, with a single case of group 1 lying close to group 2. This is a successful discrimination.

A second type of plot shows how the data plot along the first discriminant function axis:

```
plot(brine.lda, dimen=1, type="both")
```



Again, note the good separation of the groups along discriminant function 1, particularly so for group 2.

Evaluating the DFA

The effectiveness of DFA in classifying the groups must be evaluated, and this is done by comparing the assignments made by **predict()** to the actual group assignments. The **table()** function is most useful for this. By convention, it is called with the actual assignments as the first argument and the fitted assignments as the second argument:

```
tab = table(brine$GROUP, brine.hat$class)
tab
```

The rows in the output correspond to the groups specified in the original data and the columns correspond to the classification made by the DFA. In a perfect classification, large values would lie along the diagonal, with zeroes off the diagonal, which would indicate that all samples that belong to group 1 were discriminated by the DFA as belonging to group 1, and so on. The form of this table can give you considerable insight into which groups are reliably discriminated. It can also show which groups are likely to be confused and which types of misclassification are more common than others.

This command will calculate the overall predictive accuracy, that is, the proportion of cases that lie along the diagonal:

```
sum(tab[row(tab) == col(tab)]) / sum(tab)
```

Here the predictive accuracy is almost 95%, quite a success. This approach measures what is called the re-substitution error, how well the samples are classified when all the samples are used to develop the discriminant function.

A second approach for evaluating a DFA is leave-one-out cross-validation (also called jackknifed validation), which excludes one observation, formulates a discriminant function using the remaining data, and uses that function to classify the excluded observation. The advantage here is that the classification is totally independent of the excluded sample. This cross-validation is done automatically for each sample in the data set. To do this, add `CV=TRUE` (think **C**ross-**V**alidation) to the `lda()` call:

```
brine2.lda = lda(GROUP ~ HCO3 + SO4 + Cl + Ca + Mg + Na,
                 data=brine, CV=TRUE)
```

The success of the discrimination can be measured similarly:

```
tab2 = table(brine$GROUP, brine2.lda$class)
tab2
sum(tab2[row(tab2) == col(tab2)]) / sum(tab2)
```

In this data set, the jackknifed validation is considerably less accurate (only 79% accurate), reflecting that re-substitution error always overestimates the performance of a DFA. Such a discrepancy is particularly common with small data sets such as this, and discriminant function analysis is often much more successful with large data sets.

Prediction of new cases

The **predict()** command can also be used for classification. For example, assume that the same variables have been measured on a new set of water samples in a data frame brine.new:

```
brine.new       = brine[1:2,]
brine.new$GROUP = NULL
brine.new[1,]   = c(1.0, 1.4, 2.1, 1.9, 1.7, 2.1)
brine.new[2,]   = c(0.8, 1.2, 2.0, 1.7, 1.6, 2.0)
brine.new
```

The following command will classify the new set according to the discriminant function:

```
new = predict(brine.lda, brine.new)
new
```

If left unspecified, the prior probabilities will be the same as those used in the preceding **lda()** command. The results of the classification are stored as separate items in a list (here named **new**).

```
new$class      # The groups into which the brine.new were classified
new$posterior  # posterior probabilities of group classifications
new$x          # yields scores of df1, df2,... for brine.new
```

Assumptions

Discriminant function analysis is a parametric method. Like its cousins ANOVA, regression, and principal components analysis, it makes several assumptions. The most important assumptions are:

- The cases (rows) must belong to one, and only one, of the groups. In other words, the groups must be mutually exclusive.

- The number of cases for each group must not be greatly different, and the cases must be independent.

- Discriminant function performs better as sample size increases. A good guideline is that there should be at least four times as many samples as there are independent variables. For example, the brine data set has 6 independent variables, but only 19 cases, so sample size is rather small. The minimum sample size is the number of independent variables plus 2.

- Discriminant function analysis is highly sensitive to outliers. Each group should have the same variance for any independent variable, although the variances can differ among the independent variables. For many types of data, a log transformation will make the data more homoscedastic (that is, have equal variances).

- The independent variables should be multivariate normal. That is, when all the other independent variables are held constant, the independent variable under examination should have a normal distribution.

References

Maindonald, J., and J. Braun, 2003. Data Analysis and Graphics Using R. Cambridge: Cambridge, 362 p.

Davis, J.C., 2002. Statistics and Data Analysis in Geology, Third Edition. Wiley: New York, 638 p.