Getting to do statistics in **R**

Alexandre Courtiol

Leibniz Institute of Zoo and Wildlife Research

June 2018

Leibniz Institute for Zoo
and Wildlife Research
IN THE FORSCHUNGSVERBUND BERLIN E.V.

Physalia
Courses

Getting started with **R**

1. Some basic tests

2. Principal Component Analysis

3. Linear Models

R provides many statistical tests out of the box

E.g. the usual correlation tests:

Just the correlation coefficient:

```
cor(x = iris$Sepal.Length, y = iris$Sepal.Width, method = "pearson")
## [1] -0.1175698
```

R provides many statistical tests out of the box

E.g. the usual correlation tests:

Just the correlation coefficient:

```r
cor(x = iris$Sepal.Length, y = iris$Sepal.Width, method = "pearson")
## [1] -0.1175698
```

Or the actual test:

```r
cor.test(x = iris$Sepal.Length, y = iris$Sepal.Width, method = "pearson")
##
##  Pearson's product-moment correlation
##
## data:  iris$Sepal.Length and iris$Sepal.Width
## t = -1.4403, df = 148, p-value = 0.1519
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  -0.27269325  0.04351158
## sample estimates:
##        cor
## -0.1175698
```

R provides many statistical tests out of the box

E.g. the usual correlation tests:

Just the correlation coefficient:

```
cor(x = iris$Sepal.Length, y = iris$Sepal.Width, method = "pearson")
## [1] -0.1175698
```

Or the actual test:

```
cor.test(x = iris$Sepal.Length, y = iris$Sepal.Width, method = "pearson")
##
##  Pearson's product-moment correlation
##
## data:  iris$Sepal.Length and iris$Sepal.Width
## t = -1.4403, df = 148, p-value = 0.1519
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  -0.27269325  0.04351158
## sample estimates:
##        cor
## -0.1175698
```

Note: two other `methods` are available: “`spearman`” & “‘`kendall`”.

R provides many statistical tests out of the box

Note: many (not all) tests allow for the use of both standard and formula-based syntax:

E.g.

```
cor.test(formula = ~ Sepal.Length + Sepal.Width, data = iris)

##
##  Pearson's product-moment correlation
##
## data:  Sepal.Length and Sepal.Width
## t = -1.4403, df = 148, p-value = 0.1519
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  -0.27269325  0.04351158
## sample estimates:
##        cor
## -0.1175698
```

is synonymous to:

```
cor.test(x = iris$Sepal.Length, y = iris$Sepal.Width)

##
##  Pearson's product-moment correlation
##
## data:  iris$Sepal.Length and iris$Sepal.Width
## t = -1.4403, df = 148, p-value = 0.1519
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  -0.27269325  0.04351158
## sample estimates:
##        cor
## -0.1175698
```

R provides many statistical tests out of the box

E.g. test(s) for comparing two unpaired groups:

The *t*-test (parametric):

```
t.test(x = iris$Sepal.Length[iris$Species == "versicolor"],
       y = iris$Sepal.Length[iris$Species == "setosa"])
```

```
##
##  Welch Two Sample t-test
##
## data:  iris$Sepal.Length[iris$Species == "versicolor"] and iris$Sepal.Length[iris$Species == "setosa"]
## t = 10.521, df = 86.538, p-value < 2.2e-16
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  0.7542926 1.1057074
## sample estimates:
## mean of x mean of y
##     5.936     5.006
```

R provides many statistical tests out of the box

E.g. test(s) for comparing two unpaired groups:

The *t*-test (parametric):

```
t.test(x = iris$Sepal.Length[iris$Species == "versicolor"],
       y = iris$Sepal.Length[iris$Species == "setosa"])
```

```
##
##  Welch Two Sample t-test
##
## data:  iris$Sepal.Length[iris$Species == "versicolor"] and iris$Sepal.Length[iris$Species == "setosa"]
## t = 10.521, df = 86.538, p-value < 2.2e-16
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  0.7542926 1.1057074
## sample estimates:
## mean of x mean of y
##     5.936     5.006
```

The Mann-Whitney U test (non-parametric):

```
wilcox.test(x = iris$Sepal.Length[iris$Species == "versicolor"],
            y = iris$Sepal.Length[iris$Species == "setosa"])
```

```
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  iris$Sepal.Length[iris$Species == "versicolor"] and iris$Sepal.Length[iris$Species == "setosa"]
## W = 2331.5, p-value = 8.346e-14
## alternative hypothesis: true location shift is not equal to 0
```

R provides many statistical tests out of the box

E.g. test(s) for comparing two paired groups:

The paired *t*-test (parametric):

```
t.test(x = iris$Sepal.Length[iris$Species == "versicolor"],
       y = iris$Petal.Length[iris$Species == "versicolor"], paired = TRUE)
```

```
##
##  Paired t-test
##
## data:  iris$Sepal.Length[iris$Species == "versicolor"] and iris$Petal.Length[iris$Species == "versicolor"]
## t = 34.006, df = 49, p-value < 2.2e-16
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  1.576957 1.775043
## sample estimates:
## mean of the differences
##                   1.676
```

R provides many statistical tests out of the box

E.g. test(s) for comparing two paired groups:

The paired *t*-test (parametric):

```
t.test(x = iris$Sepal.Length[iris$Species == "versicolor"],
       y = iris$Petal.Length[iris$Species == "versicolor"], paired = TRUE)
```

```
##
##  Paired t-test
##
## data:  iris$Sepal.Length[iris$Species == "versicolor"] and iris$Petal.Length[iris$Species == "versicolor"]
## t = 34.006, df = 49, p-value < 2.2e-16
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  1.576957 1.775043
## sample estimates:
## mean of the differences
##                   1.676
```

The Wilcox-signed-rank test (non-parametric):

```
wilcox.test(x = iris$Sepal.Length[iris$Species == "versicolor"],
            y = iris$Petal.Length[iris$Species == "versicolor"], paired = TRUE)
```

```
##
##  Wilcoxon signed rank test with continuity correction
##
## data:  iris$Sepal.Length[iris$Species == "versicolor"] and iris$Petal.Length[iris$Species == "versicolor"]
## V = 1275, p-value = 7.454e-10
## alternative hypothesis: true location shift is not equal to 0
```

R provides many statistical tests out of the box

E.g. test(s) for comparing two paired groups:

The paired *t*-test (parametric):

```
t.test(x = iris$Sepal.Length[iris$Species == "versicolor"],
       y = iris$Petal.Length[iris$Species == "versicolor"], paired = TRUE)

##
##  Paired t-test
##
## data:  iris$Sepal.Length[iris$Species == "versicolor"] and iris$Petal.Length[iris$Species == "versicolor"]
## t = 34.006, df = 49, p-value < 2.2e-16
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  1.576957 1.775043
## sample estimates:
## mean of the differences
##                   1.676
```

The Wilcox-signed-rank test (non-parametric):

```
wilcox.test(x = iris$Sepal.Length[iris$Species == "versicolor"],
            y = iris$Petal.Length[iris$Species == "versicolor"], paired = TRUE)

##
##  Wilcoxon signed rank test with continuity correction
##
## data:  iris$Sepal.Length[iris$Species == "versicolor"] and iris$Petal.Length[iris$Species == "versicolor"]
## V = 1275, p-value = 7.454e-10
## alternative hypothesis: true location shift is not equal to 0
```

Note: if you forget to specify that the data are `paired`, it won't run the right test!

R provides many statistical tests out of the box

E.g. test(s) for comparing more than 2 unpaired groups:

The Kurskal-Wallis test (non-parametric):

```
kruskal.test(formula = Petal.Length ~ Species, data = iris)

##
##  Kruskal-Wallis rank sum test
##
## data:  Petal.Length by Species
## Kruskal-Wallis chi-squared = 130.41, df = 2, p-value < 2.2e-16
```

R provides many statistical tests out of the box

E.g. test(s) for comparing more than 2 unpaired groups:

The Kurskal-Wallis test (non-parametric):

```
kruskal.test(formula = Petal.Length ~ Species, data = iris)
```

```
##
##  Kruskal-Wallis rank sum test
##
## data:  Petal.Length by Species
## Kruskal-Wallis chi-squared = 130.41, df = 2, p-value < 2.2e-16
```

The "*test for equal means in a one-way layout*" (parametric):

```
oneway.test(formula = Petal.Length ~ Species, data = iris)
```

```
##
##  One-way analysis of means (not assuming equal variances)
##
## data:  Petal.Length and Species
## F = 1828.1, num df = 2.000, denom df = 78.073, p-value < 2.2e-16
```

R provides many statistical tests out of the box

E.g. test(s) for comparing more than 2 unpaired groups:

The Kurskal-Wallis test (non-parametric):

```
kruskal.test(formula = Petal.Length ~ Species, data = iris)
```

```
##
##  Kruskal-Wallis rank sum test
##
## data:  Petal.Length by Species
## Kruskal-Wallis chi-squared = 130.41, df = 2, p-value < 2.2e-16
```

The "*test for equal means in a one-way layout*" (parametric):

```
oneway.test(formula = Petal.Length ~ Species, data = iris)
```

```
##
##  One-way analysis of means (not assuming equal variances)
##
## data:  Petal.Length and Species
## F = 1828.1, num df = 2.000, denom df = 78.073, p-value < 2.2e-16
```

Note: linear models allow for more sofisticated parametric alternatives (see later).

R provides many statistical tests out of the box

E.g. test(s) for comparing more than 2 paired groups:

The Quade test (non-parametric):

```
quade.test(y = as.matrix(iris[, c("Petal.Length", "Sepal.Length", "Petal.Width")]))
```

```
##
##  Quade test
##
## data:  as.matrix(iris[, c("Petal.Length", "Sepal.Length", "Petal.Width")])
## Quade F = 454.3, num df = 2, denom df = 298, p-value < 2.2e-16
```

R provides many statistical tests out of the box

E.g. test(s) for comparing more than 2 paired groups:

The Quade test (non-parametric):

```
quade.test(y = as.matrix(iris[, c("Petal.Length", "Sepal.Length", "Petal.Width")]))
##
##  Quade test
##
## data:  as.matrix(iris[, c("Petal.Length", "Sepal.Length", "Petal.Width")])
## Quade F = 454.3, num df = 2, denom df = 298, p-value < 2.2e-16
```

The Friedman test (non-parametric):

```
friedman.test(y = as.matrix(iris[, c("Petal.Length", "Sepal.Length", "Petal.Width")]))
##
##  Friedman rank sum test
##
## data:  as.matrix(iris[, c("Petal.Length", "Sepal.Length", "Petal.Width")])
## Friedman chi-squared = 300, df = 2, p-value < 2.2e-16
```

R provides many statistical tests out of the box

E.g. test(s) for comparing more than 2 paired groups:

The Quade test (non-parametric):

```
quade.test(y = as.matrix(iris[, c("Petal.Length", "Sepal.Length", "Petal.Width")]))
##
##  Quade test
##
## data:  as.matrix(iris[, c("Petal.Length", "Sepal.Length", "Petal.Width")])
## Quade F = 454.3, num df = 2, denom df = 298, p-value < 2.2e-16
```

The Friedman test (non-parametric):

```
friedman.test(y = as.matrix(iris[, c("Petal.Length", "Sepal.Length", "Petal.Width")]))
##
##  Friedman rank sum test
##
## data:  as.matrix(iris[, c("Petal.Length", "Sepal.Length", "Petal.Width")])
## Friedman chi-squared = 300, df = 2, p-value < 2.2e-16
```

Note: linear mixed-effects models allow for more sofisticated parametric alternatives.

R provides many statistical tests out of the box

E.g. test(s) for comparing variances between groups:

The F-test (parametric):

```
var.test(x = iris$Sepal.Length, y = iris$Petal.Length) ## max 2 groups, must be normaly distributed
##
##  F test to compare two variances
##
## data:  iris$Sepal.Length and iris$Petal.Length
## F = 0.22004, num df = 149, denom df = 149, p-value < 2.2e-16
## alternative hypothesis: true ratio of variances is not equal to 1
## 95 percent confidence interval:
##  0.1594015 0.3037352
## sample estimates:
## ratio of variances
##          0.2200361
```

R provides many statistical tests out of the box

E.g. test(s) for comparing variances between groups:

The F-test (parametric):

```
var.test(x = iris$Sepal.Length, y = iris$Petal.Length) ## max 2 groups, must be normaly distributed
##
##  F test to compare two variances
##
## data:  iris$Sepal.Length and iris$Petal.Length
## F = 0.22004, num df = 149, denom df = 149, p-value < 2.2e-16
## alternative hypothesis: true ratio of variances is not equal to 1
## 95 percent confidence interval:
##  0.1594015 0.3037352
## sample estimates:
## ratio of variances
##          0.2200361
```

The Bartlett test (parametric):

```
bartlett.test(formula = Sepal.Length ~ Species, data = iris)
##
##  Bartlett test of homogeneity of variances
##
## data:  Sepal.Length by Species
## Bartlett's K-squared = 16.006, df = 2, p-value = 0.0003345
```

R provides many statistical tests out of the box

E.g. test(s) for comparing variances between groups (continues):

The Fligner test (non-parametric):

```
fligner.test(formula = Sepal.Length ~ Species, data = iris)
```

```
##
##  Fligner-Killeen test of homogeneity of variances
##
## data:  Sepal.Length by Species
## Fligner-Killeen:med chi-squared = 11.618, df = 2, p-value = 0.003
```

R provides many statistical tests out of the box

E.g. test(s) for comparing variances between groups (continues):

The Fligner test (non-parametric):

```
fligner.test(formula = Sepal.Length ~ Species, data = iris)
##
##  Fligner-Killeen test of homogeneity of variances
##
## data:  Sepal.Length by Species
## Fligner-Killeen:med chi-squared = 11.618, df = 2, p-value = 0.003
```

Note: also `ansari.test()` and `mood.test()` for rank-based two-sample test for a difference in scale parameters.

R provides many statistical tests out of the box

E.g. test(s) for comparing 2 distributions:

The Kolmogorov-Smirnov test (non-parametric):

```
ks.test(x = iris$Sepal.Length, y = iris$Petal.Length)
## Warning in ks.test(x = iris$Sepal.Length, y = iris$Petal.Length):  p-value will be approximate in the presence of ties
##
##  Two-sample Kolmogorov-Smirnov test
##
## data:  iris$Sepal.Length and iris$Petal.Length
## D = 0.56, p-value < 2.2e-16
## alternative hypothesis: two-sided
```

R provides many statistical tests out of the box

E.g. test(s) for comparing 2 distributions:

The Kolmogorov-Smirnov test (non-parametric):

```r
ks.test(x = iris$Sepal.Length, y = iris$Petal.Length)
## Warning in ks.test(x = iris$Sepal.Length, y = iris$Petal.Length):  p-value will be approximate in the presence of ties
##
##  Two-sample Kolmogorov-Smirnov test
##
## data:  iris$Sepal.Length and iris$Petal.Length
## D = 0.56, p-value < 2.2e-16
## alternative hypothesis: two-sided
```

The Shapiro-Wilk Normality test (non-parametric):

```r
set.seed(2L)
shapiro.test(x = rnorm(100))
##
##  Shapiro-Wilk normality test
##
## data:  rnorm(100)
## W = 0.97498, p-value = 0.05375
```

R provides many statistical tests out of the box

E.g. test(s) for comparing a binomial outcome to a probability (exact):

The exact binomial test:

```
binom.test(x = 8, n = 10, p = 0.5) ## 8 heads out of 10 coin throws -> is the coin biased?

##
##  Exact binomial test
##
## data:  8 and 10
## number of successes = 8, number of trials = 10, p-value = 0.1094
## alternative hypothesis: true probability of success is not equal to 0.5
## 95 percent confidence interval:
##  0.4439045 0.9747893
## sample estimates:
## probability of success
##                    0.8
```

R provides many statistical tests out of the box

E.g. test(s) for comparing two independent proportions:

The "*test of equal proportions*":

```
prop.test(x = cbind(success = c(8, 4), failure = c(2, 6))) ## 8 heads out of 10 for one coin, 4 out of 10 for the other, do they differ?
## Warning in prop.test(x = cbind(success = c(8, 4), failure = c(2, 6))):  Chi-squared approximation may be incorrect
##
##  2-sample test for equality of proportions with continuity correction
##
## data:  cbind(success = c(8, 4), failure = c(2, 6))
## X-squared = 1.875, df = 1, p-value = 0.1709
## alternative hypothesis: two.sided
## 95 percent confidence interval:
##  -0.0919928  0.8919928
## sample estimates:
## prop 1 prop 2
##    0.8    0.4
```

R provides many statistical tests out of the box

E.g. test(s) of independence:

The Fisher exact test:

```
## check WorldPhones before running the code!
fisher.test(WorldPhones, simulate.p.value = TRUE, B = 100) ## simulation needed as too large for exact test!
##
##  Fisher's Exact Test for Count Data with simulated p-value (based on 100 replicates)
##
## data:  WorldPhones
## p-value = 0.009901
## alternative hypothesis: two.sided
```

R provides many statistical tests out of the box

E.g. test(s) of independence:

The Fisher exact test:

```
## check WorldPhones before running the code!
fisher.test(WorldPhones, simulate.p.value = TRUE, B = 100) ## simulation needed as too large for exact test!

##
##  Fisher's Exact Test for Count Data with simulated p-value (based on 100 replicates)
##
## data:  WorldPhones
## p-value = 0.009901
## alternative hypothesis: two.sided
```

The Chi-squared test for independence:

```
chisq.test(WorldPhones)

##
##  Pearson's Chi-squared test
##
## data:  WorldPhones
## X-squared = 2194.4, df = 36, p-value < 2.2e-16
```

R provides many statistical tests out of the box

E.g. test(s) of independence:

The Fisher exact test:

```r
## check WorldPhones before running the code!
fisher.test(WorldPhones, simulate.p.value = TRUE, B = 100) ## simulation needed as too large for exact test!
```

```
##
##  Fisher's Exact Test for Count Data with simulated p-value (based on 100 replicates)
##
## data:  WorldPhones
## p-value = 0.009901
## alternative hypothesis: two.sided
```

The Chi-squared test for independence:

```r
chisq.test(WorldPhones)
```

```
##
##  Pearson's Chi-squared test
##
## data:  WorldPhones
## X-squared = 2194.4, df = 36, p-value < 2.2e-16
```

Note: the McNemar test is also available when the same subjects are measured in two conditions (see `?mcnemar.test`).

Many more simple statistical tests are available in **R** packages

Some examples:

- `coin` provides permutation implementations of many tests.
- `nsm3` provides tons of non-parametric tests.
- `PMCMR` provides post-hoc tests for non-parametric tests.
- `nortest` provides several tests for normality.

Note: this list is only a very small subset!!

A note before we continue

R's original primary goal was to perform statistical analyses. So among the many thousands of packages many focus on statistical tools and by no means I will try to cover or even summarise this diversity.

I have chosen to only illustrate some of the tools I know and that I have used to show you how to do some statistics in **R**.

Since time is limited, I will not for example illustrate any Bayesian methods, nor machine learning methods, although some good packages exist for that too!

Getting started with **R**

1. Some basic tests
2. Principal Component Analysis
3. Linear Models

PCA without packages

PCA is a traditional method for dimentionality reduction:

```
head(USArrests)  ## original coordinates
```

```
##            Murder Assault UrbanPop Rape
## Alabama      13.2     236       58 21.2
## Alaska       10.0     263       48 44.5
## Arizona       8.1     294       80 31.0
## Arkansas      8.8     190       50 19.5
## California    9.0     276       91 40.6
## Colorado      7.9     204       78 38.7
```

PCA without packages

PCA is a traditional method for dimentionality reduction:

```
head(USArrests)  ## original coordinates
```

```
##            Murder Assault UrbanPop Rape
## Alabama      13.2     236       58 21.2
## Alaska       10.0     263       48 44.5
## Arizona       8.1     294       80 31.0
## Arkansas      8.8     190       50 19.5
## California    9.0     276       91 40.6
## Colorado      7.9     204       78 38.7
```

```
pca_US <- prcomp(~ Murder + Assault + Rape, data = USArrests, scale. = TRUE) ## scaling is not the default (but should be)
```

PCA without packages

PCA is a traditional method for dimentionality reduction:

```
head(USArrests)  ## original coordinates
```

```
##            Murder Assault UrbanPop Rape
## Alabama      13.2     236       58 21.2
## Alaska       10.0     263       48 44.5
## Arizona       8.1     294       80 31.0
## Arkansas      8.8     190       50 19.5
## California    9.0     276       91 40.6
## Colorado      7.9     204       78 38.7
```

```
pca_US <- prcomp(~ Murder + Assault + Rape, data = USArrests, scale. = TRUE) ## scaling is not the default (but should be)
```

```
head(pca_US$x)  ## new coordinates
```

```
##                   PC1        PC2         PC3
## Alabama    -1.1980278  0.8338118 -0.16217848
## Alaska     -2.3087473 -1.5239622  0.03833574
## Arizona    -1.5033307 -0.4983038  0.87822311
## Arkansas   -0.1759894  0.3247326  0.07111174
## California -2.0452358 -1.2725770  0.38153933
## Colorado   -1.2634133 -1.4264063 -0.08369314
```

PCA without packages

PCA is a traditional method for dimentionality reduction:

```
head(USArrests)  ## original coordinates

##            Murder Assault UrbanPop Rape
## Alabama      13.2     236       58 21.2
## Alaska       10.0     263       48 44.5
## Arizona       8.1     294       80 31.0
## Arkansas      8.8     190       50 19.5
## California    9.0     276       91 40.6
## Colorado      7.9     204       78 38.7
```

```
pca_US <- prcomp(~ Murder + Assault + Rape, data = USArrests, scale. = TRUE) ## scaling is not the default (but should be)
```

```
head(pca_US$x)  ## new coordinates

##                   PC1        PC2         PC3
## Alabama    -1.1980278  0.8338118 -0.16217848
## Alaska     -2.3087473 -1.5239622  0.03833574
## Arizona    -1.5033307 -0.4983038  0.87822311
## Arkansas   -0.1759894  0.3247326  0.07111174
## California -2.0452358 -1.2725770  0.38153933
## Colorado   -1.2634133 -1.4264063 -0.08369314
```

```
pca_US$rotation ## coefficients applied to original coordinate (after z-scoring them) to obtain the new coordinates by linear combination

##                PC1        PC2        PC3
## Murder  -0.5826006  0.5339532 -0.6127565
## Assault -0.6079818  0.2140236  0.7645600
## Rape    -0.5393836 -0.8179779 -0.1999436
```

PCA without packages

The first axis alone captures more than 78% of the total variation in the data:

```
summary(pca_US)
```

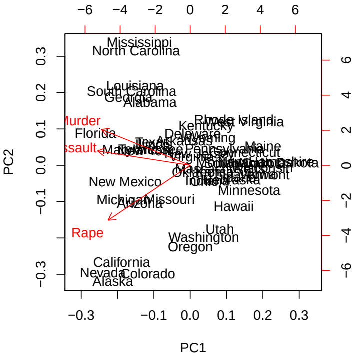
```
## Importance of components:
##                           PC1    PC2     PC3
## Standard deviation     1.5358 0.6768 0.42822
## Proportion of Variance 0.7862 0.1527 0.06112
## Cumulative Proportion  0.7862 0.9389 1.00000
```

PCA without packages

The first axis alone captures more than 78% of the total variation in the data:

```
summary(pca_US)

## Importance of components:
##                           PC1    PC2     PC3
## Standard deviation     1.5358 0.6768 0.42822
## Proportion of Variance 0.7862 0.1527 0.06112
## Cumulative Proportion  0.7862 0.9389 1.00000
```

```
biplot(x = pca_US)
```



PCA with ade4

ade4 is a package with several multivariate tools, including the PCA:

```
library(ade4)
pca_US_ade4 <- dudi.pca(df = USArrests[, -3], scale = TRUE, scannf = FALSE)

summary(pca_US_ade4)
```

```
## Class: pca dudi
## Call: dudi.pca(df = USArrests[, -3], scale = TRUE, scannf = FALSE)
##
## Total inertia: 3
##
## Eigenvalues:
##     Ax1     Ax2     Ax3
##  2.3586  0.4581  0.1834
##
## Projected inertia (%):
##     Ax1     Ax2     Ax3
##  78.619  15.268   6.112
##
## Cumulative projected inertia (%):
##     Ax1   Ax1:2   Ax1:3
##   78.62   93.89  100.00
```

PCA with ade4

ade4 is a package with several multivariate tools, including the PCA:

```
library(ade4)
pca_US_ade4 <- dudi.pca(df = USArrests[, -3], scale = TRUE, scannf = FALSE)

summary(pca_US_ade4)
```

```
## Class: pca dudi
## Call: dudi.pca(df = USArrests[, -3], scale = TRUE, scannf = FALSE)
##
## Total inertia: 3
##
## Eigenvalues:
##     Ax1     Ax2     Ax3
##  2.3586  0.4581  0.1834
##
## Projected inertia (%):
##     Ax1     Ax2     Ax3
##  78.619  15.268   6.112
##
## Cumulative projected inertia (%):
##     Ax1   Ax1:2   Ax1:3
##   78.62   93.89  100.00
```

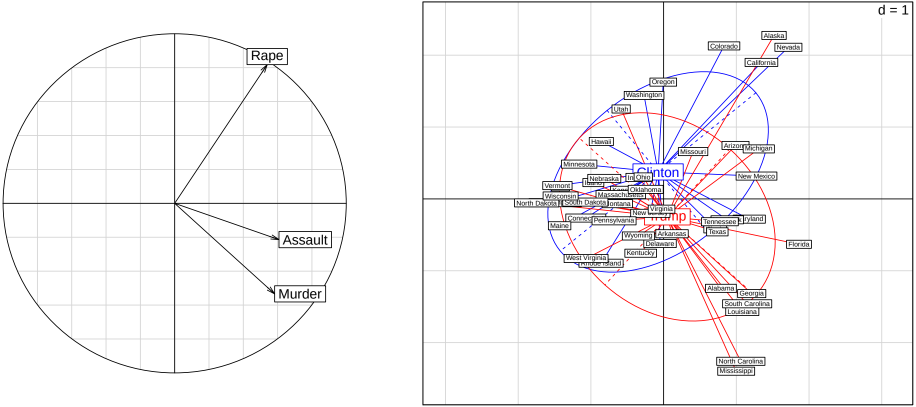
Let us add voting data to this dataset:

```
USArrests$Vote <- rep("Trump", times = 50)
Clinton_state <- c("California", "Colorado", "Connecticut", "Delaware", "Hawaii", "Illinois", "Maine", "Maryland", "Massachusetts",
                   "Minnesota", "Nevada", "New Hampshire", "New Jersey","New Mexico", "New York", "Oregon", "Rhode Island", "Vermont",
                   "Virginia", "Washington")
USArrests[Clinton_state, "Vote"] <- "Clinton"
USArrests$Vote <- factor(USArrests$Vote)
```

PCA with ade4

The package allows a different kind of plot that is quite interesting:

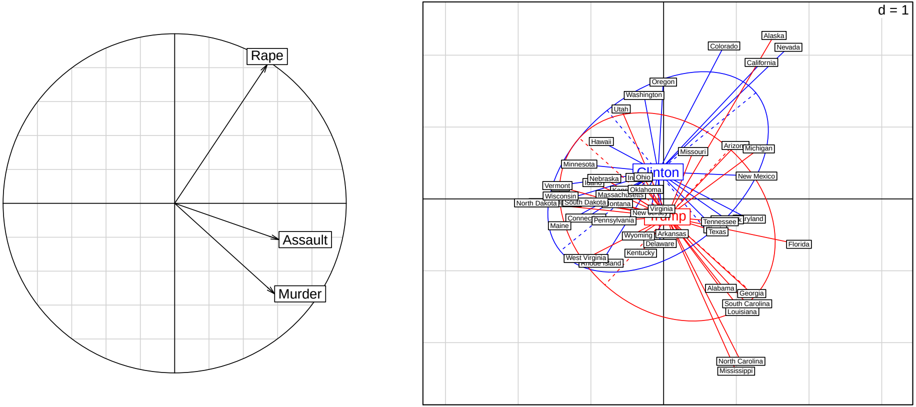
```r
par(mfrow = c(1, 2))
s.corcircle(dfxy = pca_US_ade4$c1)
s.class(dfxy = pca_US_ade4$l1, fac = USArrests$Vote, col = c("blue", "red"))
s.label(dfxy = pca_US_ade4$l1, label = rownames(USArrests), add.plot = TRUE, clabel = 0.5)
```



PCA with ade4

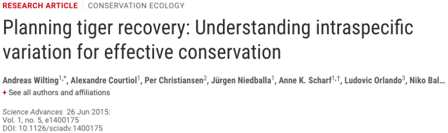
The package allows a different kind of plot that is quite interesting:

```r
par(mfrow = c(1, 2))
s.corcircle(dfxy = pca_US_ade4$c1)
s.class(dfxy = pca_US_ade4$l1, fac = USArrests$Vote, col = c("blue", "red"))
s.label(dfxy = pca_US_ade4$l1, label = rownames(USArrests), add.plot = TRUE, clabel = 0.5)
```



Note: using `co` & `li` instead of `c1` & `l1` would respect the relative contribution of each principal component according to the variance they capture.

Example of application of PCA: a revision of tiger taxonomy



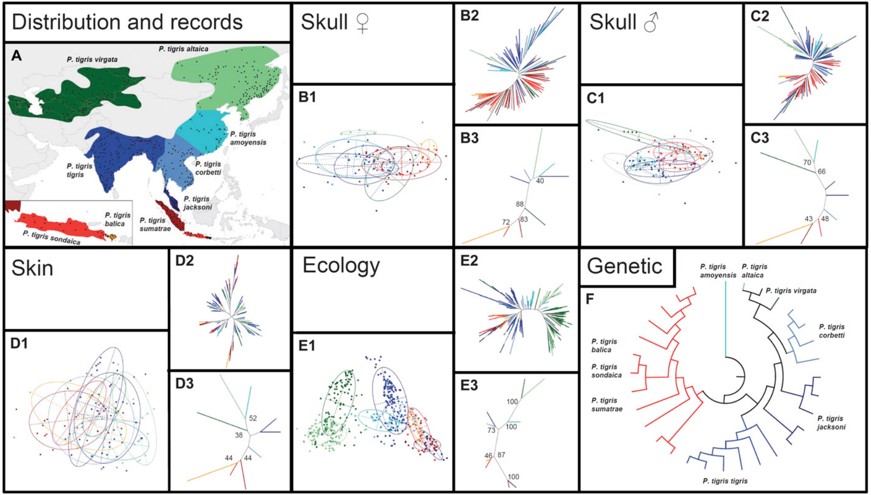
RESEARCH ARTICLE | CONSERVATION ECOLOGY

Planning tiger recovery: Understanding intraspecific variation for effective conservation

Andreas Wilting[1,*], Alexandre Courtiol[1], Per Christiansen[2], Jürgen Niedballa[1], Anne K. Scharf[1,†], Ludovic Orlando[3], Niko Bal...

+ See all authors and affiliations

Science Advances 26 Jun 2015:
Vol. 1, no. 5, e1400175
DOI: 10.1126/sciadv.1400175



Getting started with **R**

1. Some basic tests

2. Principal Component Analysis

3. Linear Models

Getting started with **R**

1. Some basic tests

2. Principal Component Analysis

3. Linear Models
   - introduction
   - traditional linear model (LM)
   - generalised linear models (GLM)
   - other linear models

What is a linear model?

A statistical model represents, often in considerably idealized form, the data-generating process (`https://en.wikipedia.org/wiki/Statistical_model`).

What is a linear model?

A statistical model represents, often in considerably idealized form, the data-generating process (`https://en.wikipedia.org/wiki/Statistical_model`).

In a linear model, the data-generating process is assumed to be a linear function: it is constructed from a set of terms by multiplying each term by a constant (a model parameter) and adding the results.

What is a linear model?

A statistical model represents, often in considerably idealized form, the data-generating process (`https://en.wikipedia.org/wiki/Statistical_model`).

In a linear model, the data-generating process is assumed to be a linear function: it is constructed from a set of terms by multiplying each term by a constant (a model parameter) and adding the results.

R allows to fit efficiently and easily all main kinds of linear models:

- classical linear models (t-test, correlation, linear regression, ANOVA, ANCOVA): LM
- generalized linear models (logistic regression, Poisson regression. . .): GLM
- linear mixed-effects models: LMM
- generalized linear mixed-effects models: GLMM
- general additive models & general additive mixed models: GAM & GAMM

What is a linear model?

A statistical model represents, often in considerably idealized form, the data-generating process (`https://en.wikipedia.org/wiki/Statistical_model`).

In a linear model, the data-generating process is assumed to be a linear function: it is constructed from a set of terms by multiplying each term by a constant (a model parameter) and adding the results.

R allows to fit efficiently and easily all main kinds of linear models:

- classical linear models (t-test, correlation, linear regression, ANOVA, ANCOVA): LM
- generalized linear models (logistic regression, Poisson regression. . .): GLM
- linear mixed-effects models: LMM
- generalized linear mixed-effects models: GLMM
- general additive models & general additive mixed models: GAM & GAMM

Note: I have a 100 hours course on the topic (`https://github.com/courtiol/LM2GLMM`) but it may be a bit terse without the bla bla. . .

Linear models in **R**

R is very rich in terms of capabilities to fit linear models due to an increasing number of dedicated packages!

For now, no other software seems to be remotely as good (prognostic: only `Julia` or `Python` may change that within a decade but I find it unlikely).

Linear models in **R**

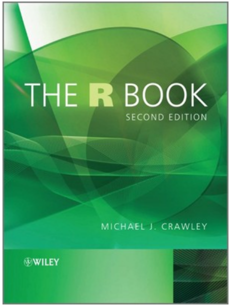
R is very rich in terms of capabilities to fit linear models due to an increasing number of dedicated packages!

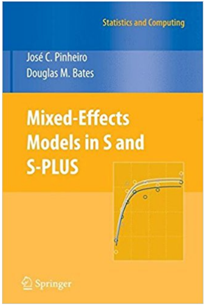
For now, no other software seems to be remotely as good (prognostic: only `Julia` or `Python` may change that within a decade but I find it unlikely).

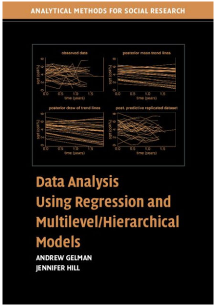
Models	Packages for fitting	Helper packages
LM	`none`; `spaMM`	`car`; `lmtest`; `visreg`
GLM	`none`; `spaMM`	`car`; `DHARMa`; `visreg`
LMM	`lme4`; `spaMM`; `glmmTMB`	`DHARMa`; `pbkrtest`; `visreg`
GLMM	`lme4`; `spaMM`; `glmmTMB`	`DHARMa`; `pbkrtest`; `visreg`
GAM	`mgcv`	`DHARMa`; `visreg`
GAMM	`mgcv`	

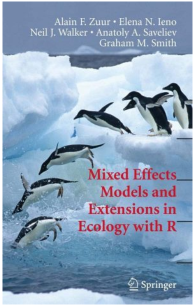
Note: those are my personal favorite ones, but they are plenty more out there.

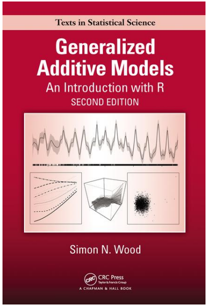
Good books dealing with linear models in **R**






ANALYTICAL METHODS FOR SOCIAL RESEARCH
Data Analysis Using Regression and Multilevel/Hierarchical Models
ANDREW GELMAN
JENNIFER HILL


Alain F. Zuur • Elena N. Ieno
Neil J. Walker • Anatoly A. Saveliev
Graham M. Smith
Mixed Effects Models and Extensions in Ecology with R
Springer



Note: it is also useful to look at books focussed on statistics and not **R**!

Preparing data for (G)LM(M) & GA(M)

To maximize the chances of success prepare your data as follow:

- one row = one observation (if repeated measures, use several rows!)
- qualitative variables of class `factor` (check the levels, drop unused ones, set the reference properly)
- no `NA` (models can somewhat deal with them but it is a major source of headackes)
- data frames (i.e. object of class `data.frame`) and not tibbles (`tbl`)

Preparing data for (G)LM(M) & GA(M)

To maximize the chances of success prepare your data as follow:

- one row = one observation (if repeated measures, use several rows!)
- qualitative variables of class `factor` (check the levels, drop unused ones, set the reference properly)
- no `NA` (models can somewhat deal with them but it is a major source of headackes)
- data frames (i.e. object of class `data.frame`) and not tibbles (`tbl`)

Example of a good dataset:

```
head(iris)

##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa

str(iris)

## 'data.frame': 150 obs. of  5 variables:
##  $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##  $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
##  $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
##  $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
##  $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...

any(is.na(iris))

## [1] FALSE
```

Getting started with **R**

1. Some basic tests

2. Principal Component Analysis

3. Linear Models
   - introduction
   - traditional linear model (LM)
   - generalised linear models (GLM)
   - other linear models

LM: notation

Simple notation:

$$y_i = \hat{\beta}_0 + \hat{\beta}_1 \times x_{1,i} + \hat{\beta}_2 \times x_{2,i} + \cdots + \hat{\beta}_p \times x_{p,i} + \varepsilon_i$$

LM: notation

Simple notation:

$$y_i = \hat{\beta}_0 + \hat{\beta}_1 \times x_{1,i} + \hat{\beta}_2 \times x_{2,i} + \cdots + \hat{\beta}_p \times x_{p,i} + \varepsilon_i$$

$$y_i = \hat{y}_i + \varepsilon_i$$

LM: notation

Simple notation:

$$y_i = \hat{\beta}_0 + \hat{\beta}_1 \times x_{1,i} + \hat{\beta}_2 \times x_{2,i} + \cdots + \hat{\beta}_p \times x_{p,i} + \varepsilon_i$$

$$y_i = \hat{y}_i + \varepsilon_i$$

- y_i = the observations to explain / response variable / dependent variable
- $\hat{y}_i$ = the fitted values
- $x_{j,i}$ = constants derived from the predictors / explanatory variables / independent variables
- $\hat{\beta}_j$ = the (model parameter / regression coefficient) estimates
- ε_i = the residuals (i.e. the estimates for the error which is Gaussian with constant variance)

LM: notation

Matrix notation:

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \dots \\ y_n \end{bmatrix} = \begin{bmatrix} 1 & x_{1,1} & x_{1,2} & \dots & x_{1,p} \\ 1 & x_{2,1} & x_{2,2} & \dots & x_{2,p} \\ 1 & x_{3,1} & x_{3,2} & \dots & x_{3,p} \\ \dots & \dots & \dots & \dots & \dots \\ 1 & x_{n,1} & x_{n,2} & \dots & x_{n,p} \end{bmatrix} \begin{bmatrix} \hat{\beta}_0 \\ \hat{\beta}_1 \\ \hat{\beta}_2 \\ \dots \\ \hat{\beta}_p \end{bmatrix} + \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \varepsilon_3 \\ \dots \\ \varepsilon_n \end{bmatrix}$$

LM: notation

Matrix notation:

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \dots \\ y_n \end{bmatrix} = \begin{bmatrix} 1 & x_{1,1} & x_{1,2} & \dots & x_{1,p} \\ 1 & x_{2,1} & x_{2,2} & \dots & x_{2,p} \\ 1 & x_{3,1} & x_{3,2} & \dots & x_{3,p} \\ \dots & \dots & \dots & \dots & \dots \\ 1 & x_{n,1} & x_{n,2} & \dots & x_{n,p} \end{bmatrix} \begin{bmatrix} \hat{\beta}_0 \\ \hat{\beta}_1 \\ \hat{\beta}_2 \\ \dots \\ \hat{\beta}_p \end{bmatrix} + \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \varepsilon_3 \\ \dots \\ \varepsilon_n \end{bmatrix}$$

$Y = X\widehat{\beta} + \varepsilon = \widehat{Y} + \varepsilon$

$\varepsilon = Y - \widehat{Y}$

LM: notation

Matrix notation:

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \dots \\ y_n \end{bmatrix} = \begin{bmatrix} 1 & x_{1,1} & x_{1,2} & \dots & x_{1,p} \\ 1 & x_{2,1} & x_{2,2} & \dots & x_{2,p} \\ 1 & x_{3,1} & x_{3,2} & \dots & x_{3,p} \\ \dots & \dots & \dots & \dots & \dots \\ 1 & x_{n,1} & x_{n,2} & \dots & x_{n,p} \end{bmatrix} \begin{bmatrix} \hat{\beta}_0 \\ \hat{\beta}_1 \\ \hat{\beta}_2 \\ \dots \\ \hat{\beta}_p \end{bmatrix} + \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \varepsilon_3 \\ \dots \\ \varepsilon_n \end{bmatrix}$$

$$Y = X\widehat{\beta} + \varepsilon = \widehat{Y} + \varepsilon$$

$$\varepsilon = Y - \widehat{Y}$$

- Y = the vector of observations
- Y = the vector of fitted values
- X = a matrix called the design matrix (or the model matrix)
- ε = the vector of residuals

LM: specifications

R formula notation:

```r
mod <- lm(Petal.Length ~ Petal.Width, data = iris)
```

LM: specifications

R formula notation:

```
mod <- lm(Petal.Length ~ Petal.Width, data = iris)
```

```
formula(mod)  ## the formula

## Petal.Length ~ Petal.Width
```

LM: specifications

R formula notation:

```
mod <- lm(Petal.Length ~ Petal.Width, data = iris)
```

```
formula(mod)  ## the formula

## Petal.Length ~ Petal.Width
```

```
head(model_frame <- model.frame(mod))  ## the data used for the fit

##   Petal.Length Petal.Width
## 1          1.4         0.2
## 2          1.4         0.2
## 3          1.3         0.2
## 4          1.5         0.2
## 5          1.4         0.2
## 6          1.7         0.4
```

LM: specifications

R formula notation:

```
mod <- lm(Petal.Length ~ Petal.Width, data = iris)
```

```
formula(mod)  ## the formula

## Petal.Length ~ Petal.Width
```

```
head(model_frame <- model.frame(mod))  ## the data used for the fit

##   Petal.Length Petal.Width
## 1          1.4         0.2
## 2          1.4         0.2
## 3          1.3         0.2
## 4          1.5         0.2
## 5          1.4         0.2
## 6          1.7         0.4
```

```
head(model.response(model_frame))  ## the response variable

##   1   2   3   4   5   6
## 1.4 1.4 1.3 1.5 1.4 1.7
```

LM: specifications

R formula notation:

```
mod <- lm(Petal.Length ~ Petal.Width, data = iris)
```

```
formula(mod)  ## the formula

## Petal.Length ~ Petal.Width
```

```
head(model_frame <- model.frame(mod))  ## the data used for the fit

##   Petal.Length Petal.Width
## 1          1.4         0.2
## 2          1.4         0.2
## 3          1.3         0.2
## 4          1.5         0.2
## 5          1.4         0.2
## 6          1.7         0.4
```

```
head(model.response(model_frame))  ## the response variable

##   1   2   3   4   5   6
## 1.4 1.4 1.3 1.5 1.4 1.7
```

```
head(model.matrix(mod))  ## the model matrix

##   (Intercept) Petal.Width
## 1           1         0.2
## 2           1         0.2
## 3           1         0.2
## 4           1         0.2
## 5           1         0.2
## 6           1         0.4
```

LM: specifications

R formula notation:

```
mod <- lm(Petal.Length ~  Species, data = iris)
```

```
formula(mod)  ## the formula
```

```
## Petal.Length ~ Species
```

```
head(model_frame <- model.frame(mod))  ## the data used for the fit
```

```
##   Petal.Length Species
## 1          1.4  setosa
## 2          1.4  setosa
## 3          1.3  setosa
## 4          1.5  setosa
## 5          1.4  setosa
## 6          1.7  setosa
```

```
head(model.response(model_frame))  ## the response variable
```

```
##   1   2   3   4   5   6
## 1.4 1.4 1.3 1.5 1.4 1.7
```

```
head(model.matrix(mod))  ## the model matrix
```

```
##   (Intercept) Speciesversicolor Speciesvirginica
## 1           1                 0                0
## 2           1                 0                0
## 3           1                 0                0
## 4           1                 0                0
## 5           1                 0                0
## 6           1                 0                0
```

LM: specifications

R formula notation:

```
mod <- lm(Petal.Length ~  Petal.Width + Species, data = iris)
```

```
formula(mod)  ## the formula

## Petal.Length ~ Petal.Width + Species
```

```
head(model_frame <- model.frame(mod))  ## the data used for the fit

##   Petal.Length Petal.Width Species
## 1          1.4         0.2  setosa
## 2          1.4         0.2  setosa
## 3          1.3         0.2  setosa
## 4          1.5         0.2  setosa
## 5          1.4         0.2  setosa
## 6          1.7         0.4  setosa
```

```
head(model.response(model_frame))  ## the response variable

##   1   2   3   4   5   6
## 1.4 1.4 1.3 1.5 1.4 1.7
```

```
head(model.matrix(mod))  ## the model matrix

##   (Intercept) Petal.Width Speciesversicolor Speciesvirginica
## 1           1         0.2                 0                0
## 2           1         0.2                 0                0
## 3           1         0.2                 0                0
## 4           1         0.2                 0                0
## 5           1         0.2                 0                0
## 6           1         0.4                 0                0
```

LM: specifications

R formula notation:

```
mod <- lm(Petal.Length ~  Petal.Width + Species + Petal.Width:Species, data = iris)
```

```
formula(mod)  ## the formula
## Petal.Length ~ Petal.Width + Species + Petal.Width:Species
```

```
head(model_frame <- model.frame(mod))  ## the data used for the fit
##   Petal.Length Petal.Width Species
## 1          1.4         0.2  setosa
## 2          1.4         0.2  setosa
## 3          1.3         0.2  setosa
## 4          1.5         0.2  setosa
## 5          1.4         0.2  setosa
## 6          1.7         0.4  setosa
```

```
head(model.response(model_frame))  ## the response variable
##   1   2   3   4   5   6
## 1.4 1.4 1.3 1.5 1.4 1.7
```

```
head(model.matrix(mod))  ## the model matrix
##   (Intercept) Petal.Width Speciesversicolor Speciesvirginica Petal.Width:Speciesversicolor Petal.Width:Speciesvirginica
## 1           1         0.2                 0                0                             0                            0
## 2           1         0.2                 0                0                             0                            0
## 3           1         0.2                 0                0                             0                            0
## 4           1         0.2                 0                0                             0                            0
## 5           1         0.2                 0                0                             0                            0
## 6           1         0.4                 0                0                             0                            0
```

LM: specifications

R formula notation:

```
mod <- lm(Petal.Length ~  Petal.Width*Species, data = iris)
```

```
formula(mod)  ## the formula

## Petal.Length ~ Petal.Width * Species
```

```
head(model_frame <- model.frame(mod))  ## the data used for the fit

##   Petal.Length Petal.Width Species
## 1          1.4         0.2  setosa
## 2          1.4         0.2  setosa
## 3          1.3         0.2  setosa
## 4          1.5         0.2  setosa
## 5          1.4         0.2  setosa
## 6          1.7         0.4  setosa
```

```
head(model.response(model_frame))  ## the response variable

##   1   2   3   4   5   6
## 1.4 1.4 1.3 1.5 1.4 1.7
```

```
head(model.matrix(mod))  ## the model matrix

##   (Intercept) Petal.Width Speciesversicolor Speciesvirginica Petal.Width:Speciesversicolor Petal.Width:Speciesvirginica
## 1           1         0.2                 0                0                             0                            0
## 2           1         0.2                 0                0                             0                            0
## 3           1         0.2                 0                0                             0                            0
## 4           1         0.2                 0                0                             0                            0
## 5           1         0.2                 0                0                             0                            0
## 6           1         0.4                 0                0                             0                            0
```

LM: specifications

R formula notation:

```
mod <- lm(Petal.Length ~  Petal.Width/Species, data = iris)  ## dangerous
```

```
formula(mod)  ## the formula

## Petal.Length ~ Petal.Width/Species
```

```
head(model_frame <- model.frame(mod))  ## the data used for the fit

##   Petal.Length Petal.Width Species
## 1          1.4         0.2  setosa
## 2          1.4         0.2  setosa
## 3          1.3         0.2  setosa
## 4          1.5         0.2  setosa
## 5          1.4         0.2  setosa
## 6          1.7         0.4  setosa
```

```
head(model.response(model_frame))  ## the response variable

##   1   2   3   4   5   6
## 1.4 1.4 1.3 1.5 1.4 1.7
```

```
head(model.matrix(mod))  ## the model matrix

##   (Intercept) Petal.Width Petal.Width:Speciesversicolor Petal.Width:Speciesvirginica
## 1           1         0.2                             0                            0
## 2           1         0.2                             0                            0
## 3           1         0.2                             0                            0
## 4           1         0.2                             0                            0
## 5           1         0.2                             0                            0
## 6           1         0.4                             0                            0
```

LM: specifications

R formula notation:

```
mod <- lm(Petal.Length ~  1, data = iris)
```

```
formula(mod)  ## the formula

## Petal.Length ~ 1
```

```
head(model_frame <- model.frame(mod))  ## the data used for the fit

##   Petal.Length
## 1          1.4
## 2          1.4
## 3          1.3
## 4          1.5
## 5          1.4
## 6          1.7
```

```
head(model.response(model_frame))  ## the response variable

##   1   2   3   4   5   6
## 1.4 1.4 1.3 1.5 1.4 1.7
```

```
head(model.matrix(mod))  ## the model matrix

##   (Intercept)
## 1           1
## 2           1
## 3           1
## 4           1
## 5           1
## 6           1
```

LM: specifications

R formula notation:

```
mod <- lm(Petal.Length ~  ., data = iris)
```

```
formula(mod)  ## the formula
```

```
## Petal.Length ~ Sepal.Length + Sepal.Width + Petal.Width + Species
```

```
head(model_frame <- model.frame(mod))  ## the data used for the fit
```

```
##   Petal.Length Sepal.Length Sepal.Width Petal.Width Species
## 1          1.4          5.1         3.5         0.2  setosa
## 2          1.4          4.9         3.0         0.2  setosa
## 3          1.3          4.7         3.2         0.2  setosa
## 4          1.5          4.6         3.1         0.2  setosa
## 5          1.4          5.0         3.6         0.2  setosa
## 6          1.7          5.4         3.9         0.4  setosa
```

```
head(model.response(model_frame))  ## the response variable
```

```
##   1   2   3   4   5   6
## 1.4 1.4 1.3 1.5 1.4 1.7
```

```
head(model.matrix(mod))  ## the model matrix
```

```
##   (Intercept) Sepal.Length Sepal.Width Petal.Width Speciesversicolor Speciesvirginica
## 1           1          5.1         3.5         0.2                 0                0
## 2           1          4.9         3.0         0.2                 0                0
## 3           1          4.7         3.2         0.2                 0                0
## 4           1          4.6         3.1         0.2                 0                0
## 5           1          5.0         3.6         0.2                 0                0
## 6           1          5.4         3.9         0.4                 0                0
```

Understanding the design matrix

It is most important to understand the design matrix because the interpretation of the parameters depend on it!

The design matrix depends on:

- your formula
- your data
- the contrasts (for qualitative variables)

Understanding the design matrix

It is most important to understand the design matrix because the interpretation of the parameters depend on it!

The design matrix depends on:

- your formula
- your data
- the contrasts (for qualitative variables)

By default, each category is compared to the first one:

```
mod <- lm(Petal.Length ~  Species, data = iris)
model.matrix(mod)[c(1, 51, 101), ]
```

```
##     (Intercept) Speciesversicolor Speciesvirginica
## 1             1                 0                0
## 51            1                 1                0
## 101           1                 0                1
```

Understanding the design matrix

It is most important to understand the design matrix because the interpretation of the parameters depend on it!

The design matrix depends on:

- your formula
- your data
- the contrasts (for qualitative variables)

By default, each category is compared to the first one:

```
mod <- lm(Petal.Length ~  Species, data = iris)
model.matrix(mod)[c(1, 51, 101), ]
```

```
##     (Intercept) Speciesversicolor Speciesvirginica
## 1             1                 0                0
## 51            1                 1                0
## 101           1                 0                1
```

But other several alternative exist; e.g.:

```
mod2 <- lm(Petal.Length ~  Species, data = iris, contrasts = list(Species = "contr.sum"))
model.matrix(mod2)[c(1, 51, 101), ]
```

```
##     (Intercept) Species1 Species2
## 1             1        1        0
## 51            1        0        1
## 101           1       -1       -1
```

Understanding the design matrix

It is most important to understand the design matrix because the interpretation of the parameters depend on it!

The design matrix depends on:

- your formula
- your data
- the contrasts (for qualitative variables)

By default, each category is compared to the first one:

```
mod <- lm(Petal.Length ~  Species, data = iris)
model.matrix(mod)[c(1, 51, 101), ]
```

```
##     (Intercept) Speciesversicolor Speciesvirginica
## 1             1                 0                0
## 51            1                 1                0
## 101           1                 0                1
```

But other several alternative exist; e.g.:

```
mod2 <- lm(Petal.Length ~  Species, data = iris, contrasts = list(Species = "contr.sum"))
model.matrix(mod2)[c(1, 51, 101), ]
```

```
##     (Intercept) Species1 Species2
## 1             1        1        0
## 51            1        0        1
## 101           1       -1       -1
```

Note 1: default contrats (`"contr.treatment"`) are easy to interpret!

Understanding the design matrix

It is most important to understand the design matrix because the interpretation of the parameters depend on it!

The design matrix depends on:

- your formula
- your data
- the contrasts (for qualitative variables)

By default, each category is compared to the first one:

```
mod <- lm(Petal.Length ~  Species, data = iris)
model.matrix(mod)[c(1, 51, 101), ]
```

```
##     (Intercept) Speciesversicolor Speciesvirginica
## 1             1                 0                0
## 51            1                 1                0
## 101           1                 0                1
```

But other several alternative exist; e.g.:

```
mod2 <- lm(Petal.Length ~  Species, data = iris, contrasts = list(Species = "contr.sum"))
model.matrix(mod2)[c(1, 51, 101), ]
```

```
##     (Intercept) Species1 Species2
## 1             1        1        0
## 51            1        0        1
## 101           1       -1       -1
```

Note 1: default contrats (`"contr.treatment"`) are easy to interpret!

Note 2: contrasts do not alter predicted values and thus likelihood, AIC. . .

Understanding the design matrix

Challenge: find out whether these different representations of gender are equivalent or not?

- `"boy"` vs `"girl"`
- `"male"` vs `"female"`
- 0 vs 1
- 1 vs 2
- `TRUE` vs `FALSE`

Note: no need to fit a model, use the function `model.matrix()` with a formula!

LM outputs: parameter estimates

```
mod <- lm(Petal.Length ~  Petal.Width + Species, data = iris)
```

Simply printing the object provides you with the parameter estimates:

```
mod

##
## Call:
## lm(formula = Petal.Length ~ Petal.Width + Species, data = iris)
##
## Coefficients:
##       (Intercept)        Petal.Width  Speciesversicolor   Speciesvirginica
##             1.211              1.019              1.698              2.277
```

LM outputs: parameter estimates

```
mod <- lm(Petal.Length ~  Petal.Width + Species, data = iris)
```

Simply printing the object provides you with the parameter estimates:

```
mod

##
## Call:
## lm(formula = Petal.Length ~ Petal.Width + Species, data = iris)
##
## Coefficients:
##       (Intercept)        Petal.Width  Speciesversicolor   Speciesvirginica
##             1.211              1.019              1.698              2.277
```

If you need to work with them, use the specific extractor instead:

```
coefficients(mod) ## or coef(mod)

##       (Intercept)       Petal.Width Speciesversicolor  Speciesvirginica
##          1.211397          1.018712          1.697791          2.276693
```

LM outputs: parameter estimates

```
mod <- lm(Petal.Length ~  Petal.Width + Species, data = iris)
```

You can also easily extract the covariance matrix of the estimates:

```
vcov(mod)
```

```
##                    (Intercept)  Petal.Width Speciesversicolor Speciesvirginica
## (Intercept)        0.004256508 -0.005701674       0.003303912      0.007295083
## Petal.Width       -0.005701674  0.023177537      -0.025031740     -0.041256016
## Speciesversicolor  0.003303912 -0.025031740       0.032742072      0.047410394
## Speciesvirginica   0.007295083 -0.041256016       0.047410394      0.079143501
```

LM outputs: parameter estimates

```
mod <- lm(Petal.Length ~  Petal.Width + Species, data = iris)
```

You can also easily extract the covariance matrix of the estimates:

```
vcov(mod)

##                    (Intercept)  Petal.Width Speciesversicolor Speciesvirginica
## (Intercept)        0.004256508 -0.005701674       0.003303912      0.007295083
## Petal.Width       -0.005701674  0.023177537      -0.025031740     -0.041256016
## Speciesversicolor  0.003303912 -0.025031740       0.032742072      0.047410394
## Speciesvirginica   0.007295083 -0.041256016       0.047410394      0.079143501
```

And thus the standard errors:

```
sqrt(diag(vcov(mod)))

##       (Intercept)       Petal.Width Speciesversicolor  Speciesvirginica
##        0.06524192        0.15224171        0.18094771        0.28132455
```

LM outputs: parameter estimates

```
mod <- lm(Petal.Length ~  Petal.Width + Species, data = iris)
```

You can also easily extract the covariance matrix of the estimates:

```
vcov(mod)
```

```
##                    (Intercept)  Petal.Width Speciesversicolor Speciesvirginica
## (Intercept)        0.004256508 -0.005701674       0.003303912      0.007295083
## Petal.Width       -0.005701674  0.023177537      -0.025031740     -0.041256016
## Speciesversicolor  0.003303912 -0.025031740       0.032742072      0.047410394
## Speciesvirginica   0.007295083 -0.041256016       0.047410394      0.079143501
```

And thus the standard errors:

```
sqrt(diag(vcov(mod)))
```

```
##       (Intercept)       Petal.Width Speciesversicolor  Speciesvirginica
##        0.06524192        0.15224171        0.18094771        0.28132455
```

You can also get confidence intervals:

```
confint(mod)
```

```
##                       2.5 %   97.5 %
## (Intercept)       1.0824564 1.340338
## Petal.Width       0.7178294 1.319594
## Speciesversicolor 1.3401762 2.055407
## Speciesvirginica  1.7206988 2.832688
```

LM outputs: parameter estimates

```
mod <- lm(Petal.Length ~  Petal.Width + Species, data = iris)
```

You can also easily extract the covariance matrix of the estimates:

```
vcov(mod)

##                    (Intercept)  Petal.Width Speciesversicolor Speciesvirginica
## (Intercept)        0.004256508 -0.005701674       0.003303912      0.007295083
## Petal.Width       -0.005701674  0.023177537      -0.025031740     -0.041256016
## Speciesversicolor  0.003303912 -0.025031740       0.032742072      0.047410394
## Speciesvirginica   0.007295083 -0.041256016       0.047410394      0.079143501
```

And thus the standard errors:

```
sqrt(diag(vcov(mod)))

##       (Intercept)       Petal.Width Speciesversicolor  Speciesvirginica
##        0.06524192        0.15224171        0.18094771        0.28132455
```

You can also get confidence intervals:

```
confint(mod)

##                       2.5 %   97.5 %
## (Intercept)       1.0824564 1.340338
## Petal.Width       0.7178294 1.319594
## Speciesversicolor 1.3401762 2.055407
## Speciesvirginica  1.7206988 2.832688
```

Note: that reveals that there are much more information in the object `mod` than it is being printed!

LM outputs: the model object

The fitted model object is in fact a big list of class "lm":

```
class(mod)

## [1] "lm"

typeof(mod)

## [1] "list"

names(mod)

##  [1] "coefficients"  "residuals"     "effects"       "rank"          "fitted.values" "assign"        "qr"
##  [8] "df.residual"   "contrasts"     "xlevels"       "call"          "terms"         "model"
```

So you can extract information from it; e.g.:

```
mod$df.residual

## [1] 146
```

but it is safer to use extractors if they are available!

LM: example of other outputs

There are quite a few extractors out there:

```
logLik(mod)

## 'log Lik.' -64.7851 (df=5)

AIC(mod)

## [1] 139.5702
```

LM: example of other outputs

There are quite a few extractors out there:

```
logLik(mod)

## 'log Lik.' -64.7851 (df=5)

AIC(mod)

## [1] 139.5702
```

Here is how you can get the list of S3 methods for the class "lm":

```
methods(class = "lm")

##  [1] add1           alias          anova          case.names     coerce         confint        cooks.distance
##  [8] deviance       dfbeta         dfbetas        drop1          dummy.coef     effects        extractAIC
## [15] family         formula        hatvalues      influence      initialize     kappa          labels
## [22] logLik         model.frame    model.matrix   nobs           plot           predict        print
## [29] proj           qr             residuals      rstandard      rstudent       show           simulate
## [36] slotsFromS3    summary        variable.names vcov
## see '?methods' for accessing help and source code
```

Note: the list will change depending on the packages that are attached to the **R** session!

LM tests: coefficients

For LM, simply use `summary()`:

```
summary(mod)
```

```
##
## Call:
## lm(formula = Petal.Length ~ Petal.Width + Species, data = iris)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.02977 -0.22241 -0.01514  0.18180  1.17449
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)        1.21140    0.06524  18.568  < 2e-16 ***
## Petal.Width        1.01871    0.15224   6.691 4.41e-10 ***
## Speciesversicolor  1.69779    0.18095   9.383  < 2e-16 ***
## Speciesvirginica   2.27669    0.28132   8.093 2.08e-13 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3777 on 146 degrees of freedom
## Multiple R-squared:  0.9551,Adjusted R-squared:  0.9542
## F-statistic:  1036 on 3 and 146 DF,  p-value: < 2.2e-16
```

LM tests: predictors

Don't use the default `anova()` function which perform type-I analysis-of-variance:

```
anova(mod)
```

```
## Analysis of Variance Table
##
## Response: Petal.Length
##               Df Sum Sq Mean Sq  F value    Pr(>F)
## Petal.Width    1 430.48  430.48 3016.792 < 2.2e-16 ***
## Species        2  13.01    6.51   45.591 4.137e-16 ***
## Residuals    146  20.83    0.14
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Instead use the better function `Anova()` from the package `car` which perfroms type-II analysis-of-variance:

```
library(car)
Anova(mod)
```

```
## Anova Table (Type II tests)
##
## Response: Petal.Length
##               Sum Sq  Df F value    Pr(>F)
## Petal.Width   6.3892   1  44.775 4.409e-10 ***
## Species      13.0113   2  45.591 4.137e-16 ***
## Residuals    20.8334 146
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

LM tests: predictors

Don't use the default `anova()` function which perform type-I analysis-of-variance:

```
anova(mod)
```

```
## Analysis of Variance Table
##
## Response: Petal.Length
##               Df Sum Sq Mean Sq  F value    Pr(>F)
## Petal.Width    1 430.48  430.48 3016.792 < 2.2e-16 ***
## Species        2  13.01    6.51   45.591 4.137e-16 ***
## Residuals    146  20.83    0.14
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Instead use the better function `Anova()` from the package `car` which perfroms type-II analysis-of-variance:

```
library(car)
Anova(mod)
```

```
## Anova Table (Type II tests)
##
## Response: Petal.Length
##               Sum Sq  Df F value    Pr(>F)
## Petal.Width   6.3892   1  44.775 4.409e-10 ***
## Species      13.0113   2  45.591 4.137e-16 ***
## Residuals    20.8334 146
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Note: p-values are the same no matter the order of the predictors in the formula for type-II (but not for type-I!).

LM tests: the overall model

Before looking at significance for estimates or predictor, always start by checking that your model fits the data better than a null model:

```
mod <- lm(Petal.Length ~  Petal.Width + Species, data = iris)
mod_null <-  lm(Petal.Length ~  1, data = iris)
anova(mod, mod_null)
```

```
## Analysis of Variance Table
##
## Model 1: Petal.Length ~ Petal.Width + Species
## Model 2: Petal.Length ~ 1
##   Res.Df    RSS Df Sum of Sq    F    Pr(>F)
## 1    146  20.83
## 2    149 464.33 -3   -443.49 1036 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Note 1: this was also given at the bottom of the summary table!

Note 2: here using `anova()` is perfectly fine!

LM predictions: fitted values

You can easily obtain the prediction for your observation (i.e. fitted values):

```
fitted(mod)[1:39]
```

```
##        1        2        3        4        5        6        7        8        9       10       11       12       13
## 1.415139 1.415139 1.415139 1.415139 1.415139 1.618882 1.517010 1.415139 1.415139 1.313268 1.415139 1.415139 1.313268
##       14       15       16       17       18       19       20       21       22       23       24       25       26
## 1.313268 1.415139 1.618882 1.618882 1.517010 1.517010 1.517010 1.415139 1.618882 1.415139 1.720753 1.415139 1.415139
##       27       28       29       30       31       32       33       34       35       36       37       38       39
## 1.618882 1.415139 1.415139 1.415139 1.415139 1.618882 1.313268 1.415139 1.415139 1.415139 1.415139 1.313268 1.415139
```

LM predictions: fitted values

You can easily obtain the prediction for your observation (i.e. fitted values):

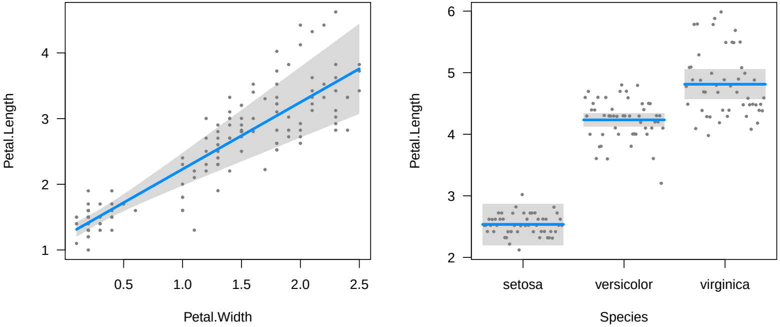
```
fitted(mod)[1:39]

##        1        2        3        4        5        6        7        8        9       10       11       12       13
## 1.415139 1.415139 1.415139 1.415139 1.415139 1.618882 1.517010 1.415139 1.415139 1.313268 1.415139 1.415139 1.313268
##       14       15       16       17       18       19       20       21       22       23       24       25       26
## 1.313268 1.415139 1.618882 1.618882 1.517010 1.517010 1.517010 1.415139 1.618882 1.415139 1.720753 1.415139 1.415139
##       27       28       29       30       31       32       33       34       35       36       37       38       39
## 1.618882 1.415139 1.415139 1.415139 1.415139 1.618882 1.313268 1.415139 1.415139 1.415139 1.415139 1.313268 1.415139
```

As expected, observations are equal to the fitted values + residuals:

```
head(cbind("response" = model.response(model.frame(mod)),
           "fitted" = fitted(mod),
           "resid" = residuals(mod),
           "fitted + resid" = fitted(mod) + residuals(mod)))

##   response   fitted       resid fitted + resid
## 1      1.4 1.415139 -0.01513927            1.4
## 2      1.4 1.415139 -0.01513927            1.4
## 3      1.3 1.415139 -0.11513927            1.3
## 4      1.5 1.415139  0.08486073            1.5
## 5      1.4 1.415139 -0.01513927            1.4
## 6      1.7 1.618882  0.08111841            1.7
```

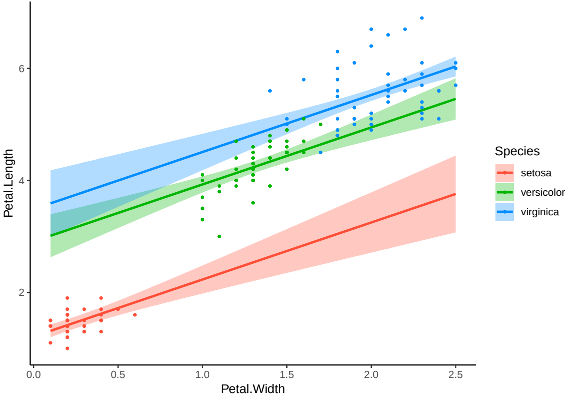
LM predictions: fast and dirty plot

```
library(visreg)
par(mfrow = c(1, 2))
visreg(mod)
```



LM predictions: fast & less dirty plot

```r
library(visreg)
library(ggplot2)
visreg(fit = mod, xvar = "Petal.Width", by = "Species", overlay = TRUE, gg = TRUE) +
  theme_classic()
```



Note: if you have different quantitative predictors you can specify the value for the non focal predictor using the argument "cond".

LM predictions: by “hand”

The most difficult step is to create the data frame defining the predictor values:

```
library(dplyr)
data_for_predictions <- iris %>%
                            group_by(Species) %>%
                            do(data.frame(Petal.Width = seq(min(.$Petal.Width), max(.$Petal.Width), length.out = 30))) %>%
                            data.frame()
```

```
head(data_for_predictions)
```

```
##   Species Petal.Width
## 1  setosa   0.1000000
## 2  setosa   0.1172414
## 3  setosa   0.1344828
## 4  setosa   0.1517241
## 5  setosa   0.1689655
## 6  setosa   0.1862069
```

```
tail(data_for_predictions)
```

```
##       Species Petal.Width
## 85 virginica    2.310345
## 86 virginica    2.348276
## 87 virginica    2.386207
## 88 virginica    2.424138
## 89 virginica    2.462069
## 90 virginica    2.500000
```

LM predictions: by “hand”

The most difficult step is to create the data frame defining the predictor values:

```r
library(dplyr)
data_for_predictions <- iris %>%
                            group_by(Species) %>%
                            do(data.frame(Petal.Width = seq(min(.$Petal.Width), max(.$Petal.Width), length.out = 30))) %>%
                            data.frame()
```

```r
head(data_for_predictions)
```

```
##   Species Petal.Width
## 1  setosa   0.1000000
## 2  setosa   0.1172414
## 3  setosa   0.1344828
## 4  setosa   0.1517241
## 5  setosa   0.1689655
## 6  setosa   0.1862069
```

```r
tail(data_for_predictions)
```

```
##      Species Petal.Width
## 85 virginica    2.310345
## 86 virginica    2.348276
## 87 virginica    2.386207
## 88 virginica    2.424138
## 89 virginica    2.462069
## 90 virginica    2.500000
```

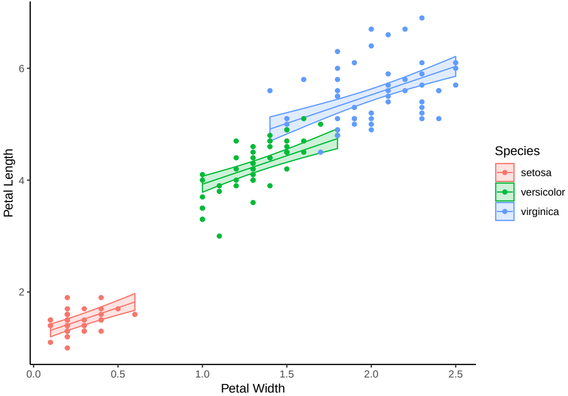
Then, it is easy:

```r
pred_mod <- predict(object = mod, newdata = data_for_predictions, interval = "confidence") ## prediction intervals are also possible!
head(pred_mod)
```

```
##        fit      lwr      upr
## 1 1.313268 1.198914 1.427622
## 2 1.330832 1.218369 1.443296
## 3 1.348396 1.237613 1.459180
## 4 1.365960 1.256636 1.475284
## 5 1.383524 1.275430 1.491618
## 6 1.401088 1.293986 1.508190
```

LM predictions: by “hand”

```
data_for_plot <- cbind(pred_mod, data_for_predictions)
ggplot(data = data_for_plot, mapping = aes(x = Petal.Width, y = fit, colour = Species)) +
  geom_line() +
  geom_ribbon(mapping = aes(ymin = lwr, ymax = upr, fill = Species), alpha = 0.2) +
  geom_point(data = iris, mapping = aes(y = Petal.Length, x = Petal.Width, colour = Species)) +
  labs(x = "Petal Width", y = "Petal Length") +
  theme_classic()
```



LM assumptions: generalities

Model structure:

- linearity
- lack of perfect multicollinearity (design matrix of full rank)
- predictor variables have fixed values

LM assumptions: generalities

Model structure:

- linearity
- lack of perfect multicollinearity (design matrix of full rank)
- predictor variables have fixed values

Errors:

- independence (no serial autocorrelation)
- constant variance (homoscedasticity)
- normality

LM assumptions: linearity

Departure from linearity can originate from a multitude of reasons and can create all kinds of problems.

LM assumptions: linearity

Departure from linearity can originate from a multitude of reasons and can create all kinds of problems.

Diagnostics:

- thinking
- other assumptions violated

LM assumptions: linearity

Departure from linearity can originate from a multitude of reasons and can create all kinds of problems.

Diagnostics:

- thinking
- other assumptions violated

Solutions:

- different model structure → change the formula
- transform one or several predictors (e.g. polynomials) → function `poly()`
- transform the response (e.g. log and power transformation) → function `powerTransform()` in `car` (see later)

LM assumptions: linearity

Departure from linearity can originate from a multitude of reasons and can create all kinds of problems.

Diagnostics:

- thinking
- other assumptions violated

Solutions:

- different model structure → change the formula
- transform one or several predictors (e.g. polynomials) → function `poly()`
- transform the response (e.g. log and power transformation) → function `powerTransform()` in `car` (see later)

Alternatives:

- non-linear models → function `nls` or dedicated package (e.g. `nlme`)
- general additive models → package `mgcv`

LM assumptions: linearity

Departure from linearity can originate from a multitude of reasons and can create all kinds of problems.

Diagnostics:

- thinking
- other assumptions violated

Solutions:

- different model structure → change the formula
- transform one or several predictors (e.g. polynomials) → function `poly()`
- transform the response (e.g. log and power transformation) → function `powerTransform()` in `car` (see later)

Alternatives:

- non-linear models → function `nls` or dedicated package (e.g. `nlme`)
- general additive models → package `mgcv`

Quiz: can you express the following models as LM?

- $y_i = \hat{\alpha} + \varepsilon_i$
- $y_i = x_i^{\hat{\beta}} + \varepsilon_i$
- $y_i = \hat{\alpha} + \hat{\beta}_1 x_i + \hat{\beta}_2 x_i^2 + \hat{\beta}_3 x_i^3 + \varepsilon_i$

LM assumptions: lack of perfect multicollinearity

The number of parameters to be estimated must be equal to the rank of the design matrix.

Caused by having less data than parameters or when there is linear dependence between the column vectors of the design matrix. In such case, some parameters cannot be computed.

LM assumptions: lack of perfect multicollinearity

The number of parameters to be estimated must be equal to the rank of the design matrix.

Caused by having less data than parameters or when there is linear dependence between the column vectors of the design matrix. In such case, some parameters cannot be computed.

Diagnostics:

- plot the predictors against each other → function `pairs()`
- `findLinearCombos` from the package `caret`

LM assumptions: lack of perfect multicollinearity

The number of parameters to be estimated must be equal to the rank of the design matrix.

Caused by having less data than parameters or when there is linear dependence between the column vectors of the design matrix. In such case, some parameters cannot be computed.

Diagnostics:

- plot the predictors against each other → function `pairs()`
- `findLinearCombos` from the package `caret`

Solutions:

- change design matrix (change parameterization or drop redundant effects) → argument `formula`
- change the experimental design
- collect more data

LM assumptions: lack of perfect multicollinearity

The number of parameters to be estimated must be equal to the rank of the design matrix.

Caused by having less data than parameters or when there is linear dependence between the column vectors of the design matrix. In such case, some parameters cannot be computed.

Diagnostics:

- plot the predictors against each other → function `pairs()`
- `findLinearCombos` from the package `caret`

Solutions:

- change design matrix (change parameterization or drop redundant effects) → argument `formula`
- change the experimental design
- collect more data

Alternatives:

- none

LM assumptions: lack of perfect multicollinearity

The number of parameters to be estimated must be equal to the rank of the design matrix.

Caused by having less data than parameters or when there is linear dependence between the column vectors of the design matrix. In such case, some parameters cannot be computed.

Diagnostics:

- plot the predictors against each other → function `pairs()`
- `findLinearCombos` from the package `caret`

Solutions:

- change design matrix (change parameterization or drop redundant effects) → argument `formula`
- change the experimental design
- collect more data

Alternatives:

- none

Note: strong albeit imperfect collinearity is not great either; possible check correlation between estimates (→ `cov2cor(vcov(mod))`) and variance inflation factors (→ `vif(mod)`).

LM assumptions: predictor variables have fixed values

The dependent variable are represented by fixed values.

The presence of measurement errors is the main cause of violation. Violation can trigger both estimates and tests to be biased.

LM assumptions: predictor variables have fixed values

The dependent variable are represented by fixed values.

The presence of measurement errors is the main cause of violation. Violation can trigger both estimates and tests to be biased.

Diagnostics:

- thinking & replication

LM assumptions: predictor variables have fixed values

The dependent variable are represented by fixed values.

The presence of measurement errors is the main cause of violation. Violation can trigger both estimates and tests to be biased.

Diagnostics:

- thinking & replication

Solutions:

- often ignored in practice
- better measurements

LM assumptions: predictor variables have fixed values

The dependent variable are represented by fixed values.

The presence of measurement errors is the main cause of violation. Violation can trigger both estimates and tests to be biased.

Diagnostics:

- thinking & replication

Solutions:

- often ignored in practice
- better measurements

Alternatives:

- multipurpose numerical approaches → function `optim()` or dedicated packages (e.g. `nloptr`, `rjags`, `nimble`, `rstan`)
- errors-in-variables models → not much directly but any procedure allowing for latent variables can handle that; packages (e.g. `sem`, `lavaan`, `OpenMX`)
- reduced major axis regression → dedicated packages (e.g. `lmodel2`)

LM assumptions: independence (no serial autocorrelation)

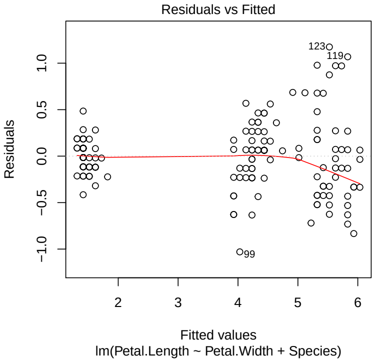
A lack of independence (serial autocorrelation) in the residuals can appear if there is a departure from linearity, if data have been sampled non-randomly (e.g. spatial or temporal series), or if there is an overarching structure (e.g. repeated measures within individuals, families, species, ...). Lack of independence increases the risk of false positive (sometimes a lot).

LM assumptions: independence (no serial autocorrelation)

A lack of independence (serial autocorrelation) in the residuals can appear if there is a departure from linearity, if data have been sampled non-randomly (e.g. spatial or temporal series), or if there is an overarching structure (e.g. repeated measures within individuals, families, species, ...). Lack of independence increases the risk of false positive (sometimes a lot).

Diagnostic by eye:

```r
plot(mod, which = 1)
```



LM assumptions: independence (no serial autocorrelation)

A lack of independence (serial autocorrelation) in the residuals can appear if there is a departure from linearity, if data have been sampled non-randomly (e.g. spatial or temporal series), or if there is an overarching structure (e.g. repeated measures within individuals, families, species, ...). Lack of independence increases the risk of false positive (sometimes a lot).

Diagnostic by Durbin-Watson test:

```
durbinWatsonTest(mod) ## from package car (DW varies between 0 & 4, 2 is best, you wish for non-significant p-value)

##  lag Autocorrelation D-W Statistic p-value
##    1       0.1313867      1.734855   0.054
##  Alternative hypothesis: rho != 0
```

Note: the alternative from the package `lmtest` offer to rank the residuals according to a variable.

LM assumptions: independence (no serial autocorrelation)

A lack of independence (serial autocorrelation) in the residuals can appear if there is a departure from linearity, if data have been sampled non-randomly (e.g. spatial or temporal series), or if there is an overarching structure (e.g. repeated measures within individuals, families, species, ...). Lack of independence increases the risk of false positive (sometimes a lot).

Diagnostic by Durbin-Watson test:

```
durbinWatsonTest(mod) ## from package car (DW varies between 0 & 4, 2 is best, you wish for non-significant p-value)
```

```
##  lag Autocorrelation D-W Statistic p-value
##    1       0.1313867      1.734855   0.054
##  Alternative hypothesis: rho != 0
```

Note: the alternative from the package `lmtest` offer to rank the residuals according to a variable.

Solutions:

- transformation or different model structure (see linearity)
- aggregation or sub-sampling

LM assumptions: independence (no serial autocorrelation)

A lack of independence (serial autocorrelation) in the residuals can appear if there is a departure from linearity, if data have been sampled non-randomly (e.g. spatial or temporal series), or if there is an overarching structure (e.g. repeated measures within individuals, families, species, ...). Lack of independence increases the risk of false positive (sometimes a lot).

Diagnostic by Durbin-Watson test:

```
durbinWatsonTest(mod) ## from package car (DW varies between 0 & 4, 2 is best, you wish for non-significant p-value)
```

```
##  lag Autocorrelation D-W Statistic p-value
##    1       0.1313867      1.734855   0.054
##  Alternative hypothesis: rho != 0
```

Note: the alternative from the package `lmtest` offer to rank the residuals according to a variable.

Solutions:

- transformation or different model structure (see linearity)
- aggregation or sub-sampling

Alternatives:

- general additive models (GAM and GAMM) → dedicated package `mgcv`
- mixed models (LMM and GLMM) → dedicated packages (e.g. `spaMM`, `lme4`)

LM assumptions: constant variance (homoscedasticity)

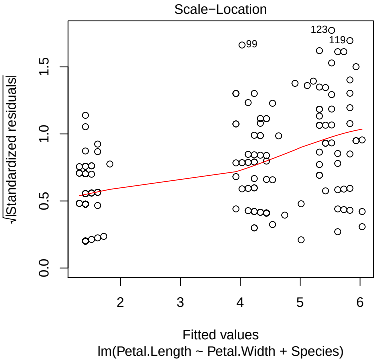
Heteros(c/k)edasticity can emerge when there is a mean - variance relationship, when there is non independence between observations, when reaction norm changes acording to the treatement. It can create both false positives and false negative.

LM assumptions: constant variance (homoscedasticity)

Heteros(c/k)edasticity can emerge when there is a mean - variance relationship, when there is non independence between observations, when reaction norm changes acording to the treatement. It can create both false positives and false negative.

Diagnostic by eye:

```r
plot(mod, which = 3)
```



LM assumptions: constant variance (homoscedasticity)

Heteros(c/k)edasticity can emerge when there is a mean - variance relationship, when there is non independence between observations, when reaction norm changes acording to the treatement. It can create both false positives and false negative.

Diagnostic by Breusch-Pagan test:

```
library(lmtest)
bptest(mod)  ## BP = df is best, you wish for non-significant p-value
```

```
##
##  studentized Breusch-Pagan test
##
## data:  mod
## BP = 28.571, df = 3, p-value = 2.755e-06
```

LM assumptions: constant variance (homoscedasticity)

Heteros(c/k)edasticity can emerge when there is a mean - variance relationship, when there is non independence between observations, when reaction norm changes acording to the treatement. It can create both false positives and false negative.

Diagnostic by Breusch-Pagan test:

```
library(lmtest)
bptest(mod)  ## BP = df is best, you wish for non-significant p-value
```

```
##
##  studentized Breusch-Pagan test
##
## data:  mod
## BP = 28.571, df = 3, p-value = 2.755e-06
```

Solutions: modeling the heteroscedasticity

```
library(spaMM)
mod_heter_spaMM <- fitme(Petal.Length ~  Petal.Width + Species,
                         resid.model = ~ Species,
                         data = iris)
AIC(mod)
```

```
## [1] 139.5702
```

```
print(AIC(mod_heter_spaMM)) ## much better fit!
```

```
##        marginal AIC:
##            87.84896
```

LM assumptions: constant variance (homoscedasticity)

Heteros(c/k)edasticity can emerge when there is a mean - variance relationship, when there is non independence between observations, when reaction norm changes acording to the treatement. It can create both false positives and false negative.

Diagnostic by Breusch-Pagan test:

```
library(lmtest)
bptest(mod)  ## BP = df is best, you wish for non-significant p-value

##
##  studentized Breusch-Pagan test
##
## data:  mod
## BP = 28.571, df = 3, p-value = 2.755e-06
```

Solutions: modeling the heteroscedasticity

```
library(spaMM)
mod_heter_spaMM <- fitme(Petal.Length ~  Petal.Width + Species,
                         resid.model = ~ Species,
                         data = iris)
AIC(mod)

## [1] 139.5702

print(AIC(mod_heter_spaMM)) ## much better fit!

##       marginal AIC:
##            87.84896
```

Alternatives:

- GLM (if stemming from an expected relationship between mean and variance) → function `glm`

LM assumptions: normality

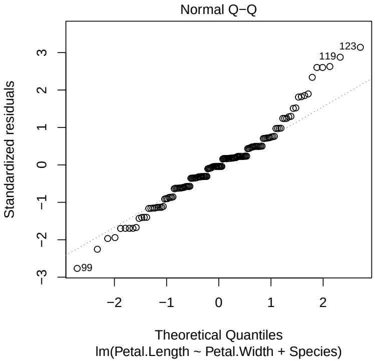
The distribution of residuals can be skewed, this is often caused by the presence of outliers, and/or when the process generating the data is very different from normal (e.g. Poisson, Binomial...).

LM assumptions: normality

The distribution of residuals can be skewed, this is often caused by the presence of outliers, and/or when the process generating the data is very different from normal (e.g. Poisson, Binomial...).

Diagnostic by eye:

```
plot(mod, which = 2)
```



LM assumptions: normality

The distribution of residuals can be skewed, this is often caused by the presence of outliers, and/or when the process generating the data is very different from normal (e.g. Poisson, Binomial...).

Diagnostic by test (many test are possible):

```
shapiro.test(mod$residuals)  ## stat = 1 when normal, you wish for non-significant p-value
```

```
##
##  Shapiro-Wilk normality test
##
## data:  mod$residuals
## W = 0.96925, p-value = 0.001924
```

LM assumptions: normality

The distribution of residuals can be skewed, this is often caused by the presence of outliers, and/or when the process generating the data is very different from normal (e.g. Poisson, Binomial...).

Diagnostic by test (many test are possible):

```
shapiro.test(mod$residuals)  ## stat = 1 when normal, you wish for non-significant p-value
```

```
##
##  Shapiro-Wilk normality test
##
## data:  mod$residuals
## W = 0.96925, p-value = 0.001924
```

Solutions:

- transformation or different model structure (see linearity)
- taking outliers out (mindfully!)

LM assumptions: normality

The distribution of residuals can be skewed, this is often caused by the presence of outliers, and/or when the process generating the data is very different from normal (e.g. Poisson, Binomial...).

Diagnostic by test (many test are possible):

```
shapiro.test(mod$residuals)  ## stat = 1 when normal, you wish for non-significant p-value
##
##  Shapiro-Wilk normality test
##
## data:  mod$residuals
## W = 0.96925, p-value = 0.001924
```

Solutions:

- transformation or different model structure (see linearity)
- taking outliers out (mindfully!)

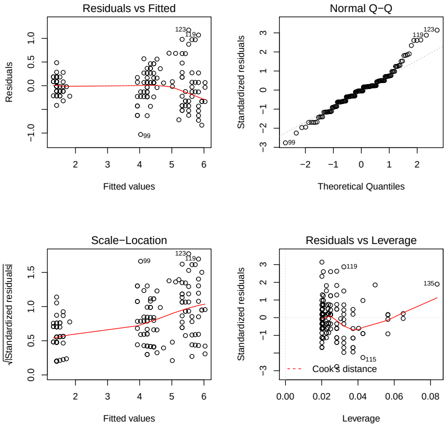
Alternatives:

- GLM (if stemming from the data generating process) → function `glm`

LM assumptions: simple glimpse

You can check all assumptions about the erros at once:

```r
par(mfrow = c(2, 2))
plot(mod)
```



LM assumptions: outliers

There is a powerful function in **R**:

```
influence.measures(mod)
```

```
## Influence measures of
##   lm(formula = Petal.Length ~ Petal.Width + Species, data = iris) :
##
##         dfb.1_  dfb.Pt.W dfb.Spcsvrs dfb.Spcsvrg    dffit cov.r   cook.d    hat inf
## 1   -0.005155  0.000756     0.00102    0.000367 -0.00582 1.049 8.51e-06 0.0203
## 2   -0.005155  0.000756     0.00102    0.000367 -0.00582 1.049 8.51e-06 0.0203
## 3   -0.039218  0.005750     0.00773    0.002791 -0.04424 1.046 4.92e-04 0.0203
## 4    0.028900 -0.004237    -0.00569   -0.002056  0.03260 1.048 2.67e-04 0.0203
## 5   -0.005155  0.000756     0.00102    0.000367 -0.00582 1.049 8.51e-06 0.0203
## 6    0.017579  0.013609    -0.02152   -0.018998  0.03386 1.052 2.89e-04 0.0239
## 7   -0.032568 -0.006861     0.01940    0.015075 -0.04511 1.047 5.12e-04 0.0205
## 8    0.028900 -0.004237    -0.00569   -0.002056  0.03260 1.048 2.67e-04 0.0203
## 9   -0.005155  0.000756     0.00102    0.000367 -0.00582 1.049 8.51e-06 0.0203
## 10   0.075521 -0.029709     0.00591    0.015059  0.07734 1.045 1.50e-03 0.0235
## 11   0.028900 -0.004237    -0.00569   -0.002056  0.03260 1.048 2.67e-04 0.0203
## 12   0.062998 -0.009237    -0.01241   -0.004483  0.07106 1.042 1.27e-03 0.0203
## 13   0.035054 -0.013790     0.00275    0.006990  0.03590 1.051 3.24e-04 0.0235
## 14  -0.086276  0.033940    -0.00676   -0.017203 -0.08835 1.043 1.96e-03 0.0235
## 15  -0.073339  0.010753     0.01445    0.005218 -0.08273 1.040 1.72e-03 0.0203
## 16  -0.025768 -0.019948     0.03155    0.027847 -0.04964 1.050 6.20e-04 0.0239
## 17  -0.069267 -0.053624     0.08480    0.074856 -0.13343 1.032 4.46e-03 0.0239
## 18  -0.032568 -0.006861     0.01940    0.015075 -0.04511 1.047 5.12e-04 0.0205
## 19   0.050956  0.010735    -0.03035   -0.023587  0.07058 1.042 1.25e-03 0.0205
## 20  -0.004733 -0.000997     0.00282    0.002191 -0.00656 1.049 1.08e-05 0.0205
## 21   0.097189 -0.014250    -0.01914   -0.006916  0.10963 1.033 3.01e-03 0.0203
## 22  -0.025768 -0.019948     0.03155    0.027847 -0.04964 1.050 6.20e-04 0.0239
## 23  -0.141957  0.020814     0.02796    0.010101 -0.16013 1.014 6.40e-03 0.0203
## 24  -0.003221 -0.005781     0.00761    0.007085 -0.00986 1.060 2.45e-05 0.0305
## 25   0.166055 -0.024347    -0.03271   -0.011816  0.18732 1.002 8.73e-03 0.0203
## 26   0.062998 -0.009237    -0.01241   -0.004483  0.07106 1.042 1.27e-03 0.0203
## 27   0.004091  0.003167     0.00501    0.004421  0.00788 1.053 1.56e-05 0.0239
```

LM assumptions: outliers

Interpretation of the output from `influence.measures(mod)`:

- `dfb.1_` → extent to which the intercept changes if a given observation is dropped
- `dfb.Pt.W` → extent to which the slope for `Petal.Width` changes if a given observation is dropped
- `dfb.Spcsvrs` → extent to which the estimate for `versicolor` changes if a given observation is dropped
- `dfb.Spcsvrg` → extent to which the estimate for `virginica` changes if a given observation is dropped

Note: for the df-betas, the name would change for another model as they use the abbreviated name of the estimates:

```
abbreviate(stats:::variable.names.lm(mod))

##       (Intercept)       Petal.Width Speciesversicolor  Speciesvirginica
##            "(In)"            "Pt.W"         "Spcsvrs"         "Spcsvrg"
```

LM assumptions: outliers

Interpretation of the output from `influence.measures(mod)`:

- `dfb.1_` → extent to which the intercept changes if a given observation is dropped
- `dfb.Pt.W` → extent to which the slope for `Petal.Width` changes if a given observation is dropped
- `dfb.Spcsvrs` → extent to which the estimate for `versicolor` changes if a given observation is dropped
- `dfb.Spcsvrg` → extent to which the estimate for `virginica` changes if a given observation is dropped
- `dffit` → extent to which the predicted y-values changes if a given observation is dropped (scaled by the standard deviation of the fit at the point)

Note: for the df-betas, the name would change for another model as they use the abbreviated name of the estimates:

```
abbreviate(stats:::variable.names.lm(mod))

##       (Intercept)       Petal.Width Speciesversicolor  Speciesvirginica
##            "(In)"            "Pt.W"         "Spcsvrs"         "Spcsvrg"
```

LM assumptions: outliers

Interpretation of the output from `influence.measures(mod)`:

- `dfb.1_` → extent to which the intercept changes if a given observation is dropped
- `dfb.Pt.W` → extent to which the slope for `Petal.Width` changes if a given observation is dropped
- `dfb.Spcsvrs` → extent to which the estimate for `versicolor` changes if a given observation is dropped
- `dfb.Spcsvrg` → extent to which the estimate for `virginica` changes if a given observation is dropped
- `dffit` → extent to which the predicted y-values changes if a given observation is dropped (scaled by the standard deviation of the fit at the point)
- `cov.r` → extent to which the covariance matrix of parameter estimates changes if a given observation is dropped

Note: for the df-betas, the name would change for another model as they use the abbreviated name of the estimates:

```
abbreviate(stats:::variable.names.lm(mod))

##       (Intercept)       Petal.Width Speciesversicolor  Speciesvirginica
##            "(In)"            "Pt.W"         "Spcsvrs"         "Spcsvrg"
```

LM assumptions: outliers

Interpretation of the output from `influence.measures(mod)`:

- `dfb.1_` → extent to which the intercept changes if a given observation is dropped
- `dfb.Pt.W` → extent to which the slope for `Petal.Width` changes if a given observation is dropped
- `dfb.Spcsvrs` → extent to which the estimate for `versicolor` changes if a given observation is dropped
- `dfb.Spcsvrg` → extent to which the estimate for `virginica` changes if a given observation is dropped
- `dffit` → extent to which the predicted y-values changes if a given observation is dropped (scaled by the standard deviation of the fit at the point)
- `cov.r` → extent to which the covariance matrix of parameter estimates changes if a given observation is dropped
- `cook.d` → F statistics comparing simultaneously the changes in all estimates when the observation is dropped or not

Note: for the df-betas, the name would change for another model as they use the abbreviated name of the estimates:

```
abbreviate(stats:::variable.names.lm(mod))

##       (Intercept)       Petal.Width Speciesversicolor  Speciesvirginica
##            "(In)"            "Pt.W"         "Spcsvrs"         "Spcsvrg"
```

LM assumptions: outliers

Interpretation of the output from `influence.measures(mod)`:

- `dfb.1_` → extent to which the intercept changes if a given observation is dropped
- `dfb.Pt.W` → extent to which the slope for `Petal.Width` changes if a given observation is dropped
- `dfb.Spcsvrs` → extent to which the estimate for `versicolor` changes if a given observation is dropped
- `dfb.Spcsvrg` → extent to which the estimate for `virginica` changes if a given observation is dropped
- `dffit` → extent to which the predicted y-values changes if a given observation is dropped (scaled by the standard deviation of the fit at the point)
- `cov.r` → extent to which the covariance matrix of parameter estimates changes if a given observation is dropped
- `cook.d` → F statistics comparing simultaneously the changes in all estimates when the observation is dropped or not
- `hat` → diagonal element of the hat matrix (the hat values); extent to which an observation is unusual in terms of X values (leverage)

Note: for the df-betas, the name would change for another model as they use the abbreviated name of the estimates:

```
abbreviate(stats:::variable.names.lm(mod))

##       (Intercept)       Petal.Width Speciesversicolor  Speciesvirginica
##            "(In)"            "Pt.W"         "Spcsvrs"         "Spcsvrg"
```

LM assumptions: outliers

Interpretation of the output from `influence.measures(mod)`:

- `dfb.1_` → extent to which the intercept changes if a given observation is dropped
- `dfb.Pt.W` → extent to which the slope for `Petal.Width` changes if a given observation is dropped
- `dfb.Spcsvrs` → extent to which the estimate for `versicolor` changes if a given observation is dropped
- `dfb.Spcsvrg` → extent to which the estimate for `virginica` changes if a given observation is dropped
- `dffit` → extent to which the predicted y-values changes if a given observation is dropped (scaled by the standard deviation of the fit at the point)
- `cov.r` → extent to which the covariance matrix of parameter estimates changes if a given observation is dropped
- `cook.d` → *F* statistics comparing simultaneously the changes in all estimates when the observation is dropped or not
- `hat` → diagonal element of the hat matrix (the hat values); extent to which an observation is unusual in terms of X values (leverage)
- `inf` → some overal add hoc receipe to spot influential observation (not to be taken too seriously)

Note: for the df-betas, the name would change for another model as they use the abbreviated name of the estimates:

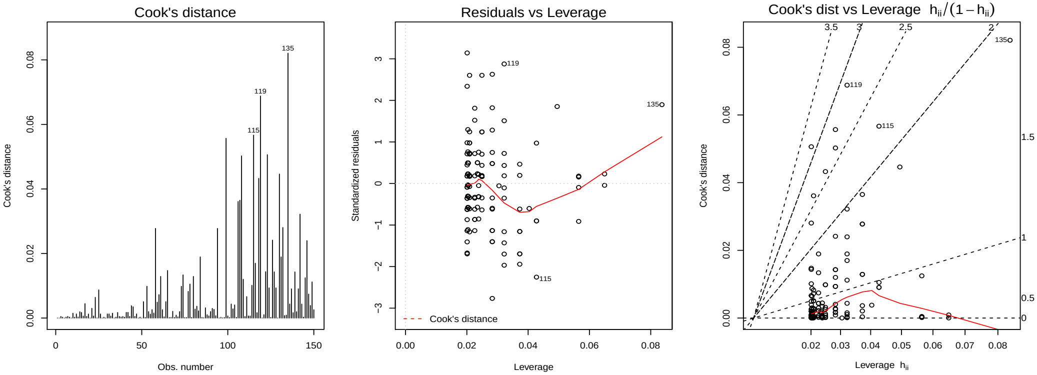
```r
abbreviate(stats:::variable.names.lm(mod))
```

```
##       (Intercept)       Petal.Width Speciesversicolor  Speciesvirginica
##            "(In)"            "Pt.W"         "Spcsvrs"         "Spcsvrg"
```

LM assumptions: outliers

There are also plotting possibilitites:

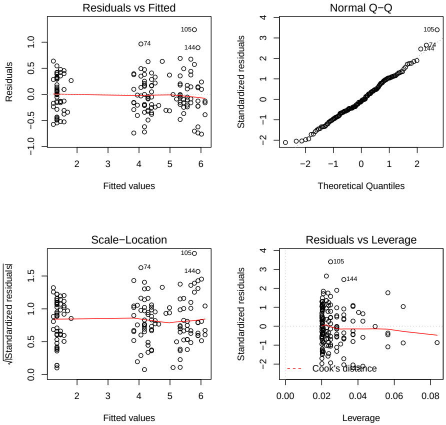
```r
par(mfrow = c(1, 3))
plot(mod, which = 4:6)
```



LM assumptions: simple glimpse at residuals

What would it look like if it was perfect?

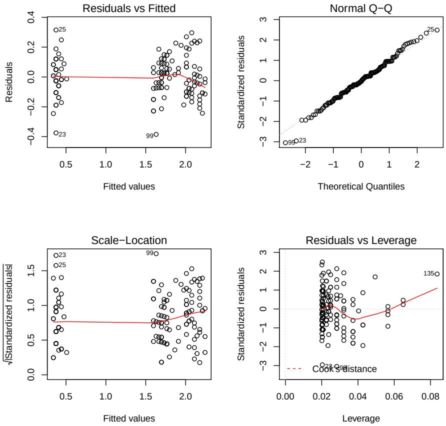
```
iris$Fake.Petal.Length <- simulate(object = mod)[, 1]  ## redo it, it will change each time!
mod_perfect <- lm(Fake.Petal.Length ~  Petal.Width + Species, data = iris)
par(mfrow = c(2, 2))
plot(mod_perfect)
```



LM assumptions: fixing iris?

Fixing attempt:

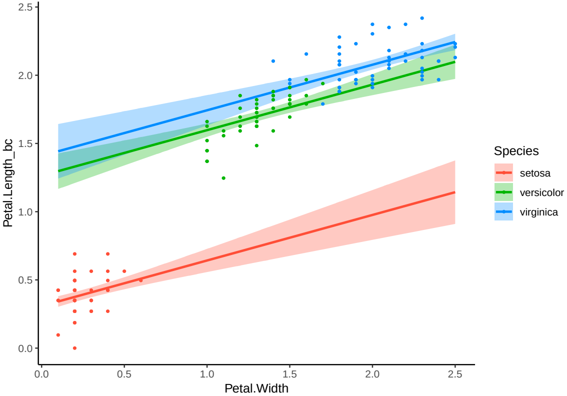
```
bc <- powerTransform(mod)
iris$Petal.Length_bc <- bcPower(iris$Petal.Length, lambda = bc$lambda)
mod_bc <- lm(Petal.Length_bc ~ Petal.Width + Species, data = iris)
par(mfrow = c(2, 2))
plot(mod_bc)
```



LM assumptions: fixing iris?

Plotting predictions:

```r
visreg(fit = mod_bc, xvar = "Petal.Width", by = "Species", overlay = TRUE, gg = TRUE) +
  theme_classic()
```

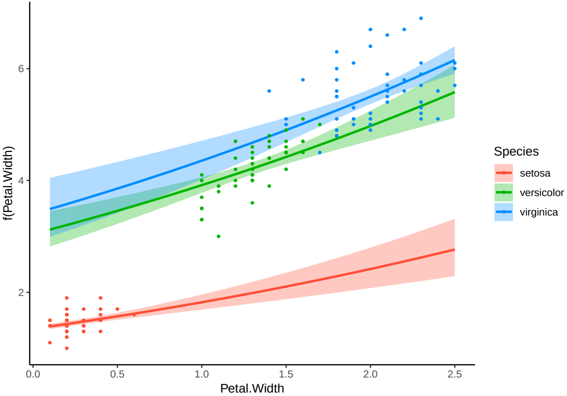


Note: that is not very useful because it is on the BoxCoxed scale!

LM assumptions: fixing iris?

Plotting predictions:

```
visreg(fit = mod_bc, xvar = "Petal.Width", by = "Species", overlay = TRUE, gg = TRUE,
       trans = function(x) bcnPowerInverse(x, lambda = bc$lambda, gamma = 0), partial = TRUE) +
  theme_classic()
```



Getting started with **R**

1. Some basic tests

2. Principal Component Analysis

3. Linear Models
   - introduction
   - traditional linear model (LM)
   - generalised linear models (GLM)
   - other linear models

The generalised linear model: what for?

GLM are used for fitting data generating processes for which a relationship between mean and variance is expected.

The generalised linear model: what for?

GLM are used for fitting data generating processes for which a relationship between mean and variance is expected.

That includes the analysis of:

- binary events (probabilities)
- binomial events (probabilities)
- Poisson processes (counts)
- negative binomail processes (counts)
- variances (positive continuous)

The generalised linear model: specifications

Definition:

$$Y = g^{-1}(\widehat{\eta}) + \varepsilon = g^{-1}(X\widehat{\beta}) + \varepsilon$$

with:

- $\hat{\eta}_i = \hat{\beta}_0 + \hat{\beta}_1 \times x_{1,i} + \hat{\beta}_2 \times x_{2,i} + \cdots + \hat{\beta}_p \times x_{p,i}$
- $E(Y) = \mu = g^{-1}(\eta)$
- $Var(Y) = \phi V(\mu)$

Notation:

- η the linear predictor
- g the link function (g^{-1} is sometimes called the mean function)
- V the variance function
- ϕ is the dispersion parameter

The generalised linear model: specifications

Definition:

$$Y = g^{-1}(\widehat{\eta}) + \varepsilon = g^{-1}(X\widehat{\beta}) + \varepsilon$$

with:

- $\hat{\eta}_i = \hat{\beta}_0 + \hat{\beta}_1 \times x_{1,i} + \hat{\beta}_2 \times x_{2,i} + \cdots + \hat{\beta}_p \times x_{p,i}$
- $E(Y) = \mu = g^{-1}(\eta)$
- $Var(Y) = \phi V(\mu)$

Notation:

- η the linear predictor
- g the link function (g^{-1} is sometimes called the mean function)
- V the variance function
- ϕ is the dispersion parameter

This is identical to the LM if:

- $\mu = g^{-1}(\eta) = \eta$, thus if g is the identity function
- $\phi = \sigma^2$, thus if the dispersion parameter equals the error variance
- $V(\mu) = 1$, thus if the variance function is constant

The generalised linear model: the Challenger dataset

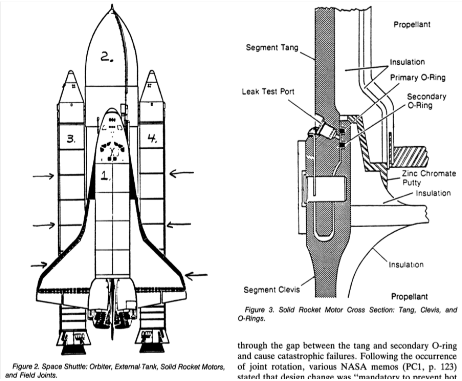


Figure 2. Space Shuttle: Orbiter, External Tank, Solid Rocket Motors, and Field Joints.

Figure 3. Solid Rocket Motor Cross Section: Tang, Clevis, and O-Rings.

through the gap between the tang and secondary O-ring and cause catastrophic failures. Following the occurrence of joint rotation, various NASA memos (PC1, p. 123) stated that design change was "mandatory to prevent hot

```r
head(Challenger, n = 3L)
```

```
##   oring_tot oring_dt temp psi flight
## 1         6        0   66  50      1
## 2         6        1   70  50      2
## 3         6        0   69  50      3
```

Note: we will study both the probability that one oring fails (binary event) or that at least one oring fails (binomial event) as a function of the temperature.

The generalised linear model: the VonBort dataset

```
head(VonBort, n = 3L)
```

```
##   deaths year corps fisher
## 1      0 1875     G     no
## 2      0 1875     I     no
## 3      0 1875    II    yes
```

Note: we will compare the number of deaths caused by horse (or mule) kicks between the 14 corps of the Prussian army.

The generalised linear model: specifications

In **R** notation:

```r
Challenger$issue <- Challenger$oring_dt > 0

mod_challenger_binar <- glm(issue ~ temp, family = binomial(link = "logit"), data = Challenger)
```

The generalised linear model: specifications

In **R** notation:

```
Challenger$issue <- Challenger$oring_dt > 0

mod_challenger_binar <- glm(issue ~ temp, family = binomial(link = "logit"), data = Challenger)
```

```
Challenger$oring_ok <- Challenger$oring_tot - Challenger$oring_dt

mod_challenger_binom <- glm(cbind(oring_dt, oring_ok) ~ temp, family = binomial(link = "logit"), data = Challenger)
```

The generalised linear model: specifications

In **R** notation:

```
Challenger$issue <- Challenger$oring_dt > 0

mod_challenger_binar <- glm(issue ~ temp, family = binomial(link = "logit"), data = Challenger)
```

```
Challenger$oring_ok <- Challenger$oring_tot - Challenger$oring_dt

mod_challenger_binom <- glm(cbind(oring_dt, oring_ok) ~ temp, family = binomial(link = "logit"), data = Challenger)
```

```
mod_horsekick <- glm(deaths ~ corps, family = poisson(link = "log"), data = VonBort)
```

The generalised linear model: specifications

The `family` object contains all kinds of useful information required for the fitting procedure:

```
names(binomial(link = "logit"))

##  [1] "family"     "link"         "linkfun"      "linkinv"      "variance"     "dev.resids" "aic"          "mu.eta"
##  [9] "initialize" "validmu"      "valideta"     "simulate"
```

The generalised linear model: specifications

The `family` object contains all kinds of useful information required for the fitting procedure:

```
names(binomial(link = "logit"))
```

```
##  [1] "family"     "link"       "linkfun"    "linkinv"    "variance"   "dev.resids" "aic"        "mu.eta"
##  [9] "initialize" "validmu"    "valideta"   "simulate"
```

The link function:

```
probs <- seq(0.1, 0.9, by = 0.1)
logits <- binomial(link = "logit")$linkfun(mu = probs)
logits
```

```
## [1] -2.1972246 -1.3862944 -0.8472979 -0.4054651  0.0000000  0.4054651  0.8472979  1.3862944  2.1972246
```

The generalised linear model: specifications

The `family` object contains all kinds of useful information required for the fitting procedure:

```
names(binomial(link = "logit"))
```

```
##  [1] "family"     "link"       "linkfun"    "linkinv"    "variance"   "dev.resids" "aic"        "mu.eta"
##  [9] "initialize" "validmu"    "valideta"   "simulate"
```

The link function:

```
probs <- seq(0.1, 0.9, by = 0.1)
logits <- binomial(link = "logit")$linkfun(mu = probs)
logits
```

```
## [1] -2.1972246 -1.3862944 -0.8472979 -0.4054651  0.0000000  0.4054651  0.8472979  1.3862944  2.1972246
```

The inverse link function:

```
binomial(link = "logit")$linkinv(eta = logits)
```

```
## [1] 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9
```

The generalised linear model: specifications

The `family` object contains all kinds of useful information required for the fitting procedure:

```
names(binomial(link = "logit"))
```

```
##  [1] "family"     "link"       "linkfun"    "linkinv"    "variance"   "dev.resids" "aic"        "mu.eta"
##  [9] "initialize" "validmu"    "valideta"   "simulate"
```

The link function:

```
probs <- seq(0.1, 0.9, by = 0.1)
logits <- binomial(link = "logit")$linkfun(mu = probs)
logits
```

```
## [1] -2.1972246 -1.3862944 -0.8472979 -0.4054651  0.0000000  0.4054651  0.8472979  1.3862944  2.1972246
```

The inverse link function:

```
binomial(link = "logit")$linkinv(eta = logits)
```

```
## [1] 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9
```

The variance function:

```
binomial(link = "logit")$variance(mu = probs)
```

```
## [1] 0.09 0.16 0.21 0.24 0.25 0.24 0.21 0.16 0.09
```

The generalised linear model: specifications

The `family` object contains all kinds of useful information required for the fitting procedure:

```
names(binomial(link = "logit"))
```

```
##  [1] "family"     "link"       "linkfun"    "linkinv"    "variance"   "dev.resids" "aic"        "mu.eta"
##  [9] "initialize" "validmu"    "valideta"   "simulate"
```

The link function:

```
probs <- seq(0.1, 0.9, by = 0.1)
logits <- binomial(link = "logit")$linkfun(mu = probs)
logits
```

```
## [1] -2.1972246 -1.3862944 -0.8472979 -0.4054651  0.0000000  0.4054651  0.8472979  1.3862944  2.1972246
```

The inverse link function:

```
binomial(link = "logit")$linkinv(eta = logits)
```

```
## [1] 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9
```

The variance function:

```
binomial(link = "logit")$variance(mu = probs)
```

```
## [1] 0.09 0.16 0.21 0.24 0.25 0.24 0.21 0.16 0.09
```

Note: you can use these functions to better understand GLM or when you need them to process some outputs.

The outputs are similar to those from `lm` fits!

```
mod_challenger_binar
```

```
##
## Call:  glm(formula = issue ~ temp, family = binomial(link = "logit"),
##     data = Challenger)
##
## Coefficients:
## (Intercept)         temp
##     23.7750      -0.3667
##
## Degrees of Freedom: 22 Total (i.e. Null);  21 Residual
## Null Deviance:     26.4
## Residual Deviance: 14.43  AIC: 18.43
```

```
confint(mod_challenger_binar)
```

```
## Waiting for profiling to be done...
##                  2.5 %     97.5 %
## (Intercept)  7.2430347 58.1947978
## temp        -0.8772585 -0.1217173
```

The outputs are similar to those from `lm` fits!

```
mod_challenger_binom
##
## Call:  glm(formula = cbind(oring_dt, oring_ok) ~ temp, family = binomial(link = "logit"),
##     data = Challenger)
##
## Coefficients:
## (Intercept)         temp
##      8.8169      -0.1795
##
## Degrees of Freedom: 22 Total (i.e. Null);  21 Residual
## Null Deviance:     20.71
## Residual Deviance: 9.527  AIC: 24.87
confint(mod_challenger_binom)
## Waiting for profiling to be done...
##                 2.5 %      97.5 %
## (Intercept)  1.9549041 16.49138135
## temp        -0.3073739 -0.07257416
```

The outputs are similar to those from `lm` fits!

```r
mod_horsekick
```

```
##
## Call:  glm(formula = deaths ~ corps, family = poisson(link = "log"),
##     data = VonBort)
##
## Coefficients:
## (Intercept)       corpsI      corpsII     corpsIII      corpsIV       corpsV      corpsVI     corpsVII    corpsVIII
##  -2.231e-01    4.072e-09   -2.877e-01   -2.877e-01   -6.931e-01   -3.747e-01    6.062e-02   -2.877e-01   -8.267e-01
##     corpsIX       corpsX      corpsXI     corpsXIV      corpsXV
##  -2.076e-01   -6.454e-02    4.463e-01    4.055e-01   -6.931e-01
##
## Degrees of Freedom: 279 Total (i.e. Null);  266 Residual
## Null Deviance:     323.2
## Residual Deviance: 297.1  AIC: 630.2
```

```r
head(confint(mod_horsekick))
```

```
## Waiting for profiling to be done...
##                  2.5 %    97.5 %
## (Intercept) -0.7566949 0.2298300
## corpsI      -0.6999361 0.6999361
## corpsII     -1.0585453 0.4561131
## corpsIII    -1.0585453 0.4561131
## corpsIV     -1.5958865 0.1280845
## corpsV      -1.1704167 0.3841036
```

The outputs are similar to those from `lm` fits!

```
mod_horsekick
```

```
##
## Call:  glm(formula = deaths ~ corps, family = poisson(link = "log"),
##     data = VonBort)
##
## Coefficients:
## (Intercept)       corpsI      corpsII     corpsIII      corpsIV       corpsV      corpsVI     corpsVII    corpsVIII
##  -2.231e-01    4.072e-09   -2.877e-01   -2.877e-01   -6.931e-01   -3.747e-01    6.062e-02   -2.877e-01   -8.267e-01
##     corpsIX       corpsX      corpsXI     corpsXIV      corpsXV
##  -2.076e-01   -6.454e-02    4.463e-01    4.055e-01   -6.931e-01
##
## Degrees of Freedom: 279 Total (i.e. Null);  266 Residual
## Null Deviance:      323.2
## Residual Deviance: 297.1  AIC: 630.2
```

```
head(confint(mod_horsekick))
```

```
## Waiting for profiling to be done...
##                  2.5 %    97.5 %
## (Intercept) -0.7566949 0.2298300
## corpsI      -0.6999361 0.6999361
## corpsII     -1.0585453 0.4561131
## corpsIII    -1.0585453 0.4561131
## corpsIV     -1.5958865 0.1280845
## corpsV      -1.1704167 0.3841036
```

Note: but the interpretation of the parameters is very different since they are expressed on the scale of the linear predictor!!

Interpreting estimates

There is no general receipe, it all depends on the link function used. . .

Interpreting estimates

There is no general receipe, it all depends on the link function used...

- For logistic regressions (`link = "logit"`; not for all binomial models), use odd-ratios:

```
exp(coef(mod_challenger_binar)["temp"])

##      temp
## 0.6930169

1/exp(coef(mod_challenger_binar)["temp"])

##     temp
## 1.442966
```

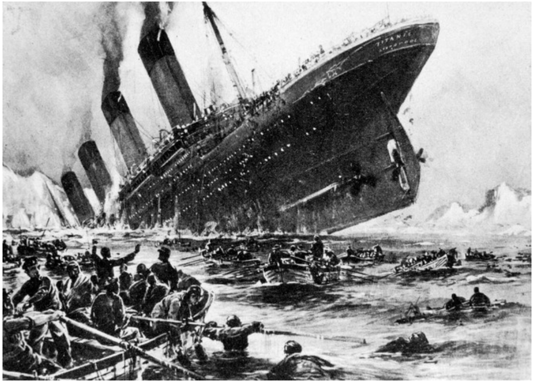
Every decrease by one degree increases the odd of failure for at least one oring by 1.4 time!

```
exp(coef(mod_challenger_binom)["temp"])

##      temp
## 0.8356945

1/exp(coef(mod_challenger_binom)["temp"])

##    temp
## 1.19661
```

Every decrease by one degree increases the odd of failure for exactly one oring by 1.2 time!

The generalised linear model: challenge

Try to understand what influenced survival (i.e. access to lifeboats) during the Titanic disaster.



```
head(TitanicSurvival)
```

```
##                                 survived    sex     age passengerClass
## Allen, Miss. Elisabeth Walton        yes female 29.0000            1st
## Allison, Master. Hudson Trevor       yes   male  0.9167            1st
## Allison, Miss. Helen Loraine          no female  2.0000            1st
## Allison, Mr. Hudson Joshua Crei       no   male 30.0000            1st
## Allison, Mrs. Hudson J C (Bessi       no female 25.0000            1st
## Anderson, Mr. Harry                  yes   male 48.0000            1st
```

Getting started with **R**

1. Some basic tests

2. Principal Component Analysis

3. Linear Models
   - introduction
   - traditional linear model (LM)
   - generalised linear models (GLM)
   - other linear models