Statistics

Basel R Bootcamp

www.therbootcamp.com

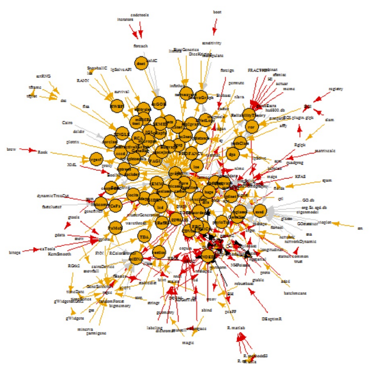
@therbootcamp

July 2018

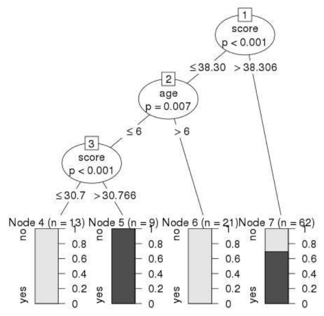
Stats? There is a package for that!

Package	Models
`stats`	Generalized linear model
`afex`	Anovas
`lme4`	Mixed effects regression
`rpart`	Decision Trees
`BayesFactor`	Bayesian statistics
`igraph`	Network analysis
`neuralnet`	Neural networks
`MatchIt`	Matching and causal inference
`survival`	Longitudinal survival analysis
…	Anything you can ever want!

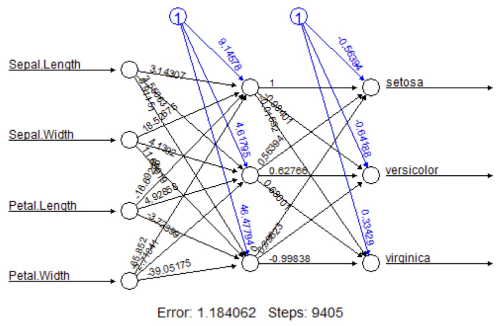
Networks / Cluster analyses



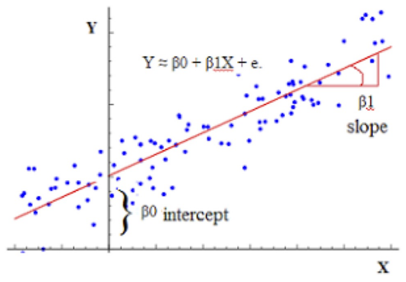
Decision Trees



Neutral Networks



Regression



In this session...

1 - Basic structure and arguments of most **statistical functions**

- `formula`: Specify your **variables**
- `data`: A **data frame** containing variables.

2 - Simple `htest` objects

- `t.test()`, `correlation`

3 - (Generalized) **linear model**

- `lm()`, `glm()`, `aov()`

4 - Explore statistical objects

- `MODEL$NAME`, `print()`, `summary()`, `names()`, `predict()`, `plot()`

5 - Conduct simulations

```
# t-test comparing height based on gender
t.test(formula = height ~ sex,
         data = baselers)

# Regression model
inc_glm <- glm(formula = income ~ .,
               data = baselers %>% select(-id))

# Summary information
summary(inc_glm)

# Dissect
inc_glm$coefficients # Acess coefficients
inc_glm$residuals # Access residuals

### Generate random data
x1 <- rnorm(n = 100, mean = 10, sd = 5)
x2 <- rnorm(n = 100, mean = 5, sd = 1)
noise <- rnorm(n= 100, mean = 0, sd = 10)

# Create y as a function of x1, x2, and noise
y <- x1 + x2 + noise

df <- data.frame(x1, x2, y)

# Regress y on x1 and x2
lm(formula = y ~ .,
   data = df)
```

Basic structure of statistical functions

Statistical functions always require a **data frame** called `data`, e.g.,...

sex	age	height	weight	income
male	44	174.3	113.4	6300
male	65	180.3	75.2	10900
female	31	168.3	55.5	5100

They also require a **formula** that specifies a **dependent** variable (y) as a function of one or more **independent** variables (x1, x2, ...) in the form:

formula = y ~ x1 + x2 +...

How to create a statistical object:

```
# Example: Create regression object (my_glm)
my_glm <- glm(formula = income ~ age + height,
              data = baselers)
```

Name of the statistical function

Dependent variable

All independent variable(s)

Model_Object <- **STAT_FUN**(**formula** = y ~ x1 + x2 + …,
data = mydata,
…)

Name of new object

Additional arguments specific to STAT_FUN

A dataframe containing all variables in formula (y, x1, x2, …)

Look for optional arguments

Statistical functions usually have many optional arguments.

Each of these have **default** values. To customise a test, **look at the help menu** and specify arguments explicitly.

Default vs. customised `glm()` (Generalized linear model)

```
# Default
glm(formula = income ~ age + education,
    data = baselers)

# Customised
glm(formula = eyecor ~ age + education,
    data = baselers,
    family = "binomial")  # Logistic regression
```

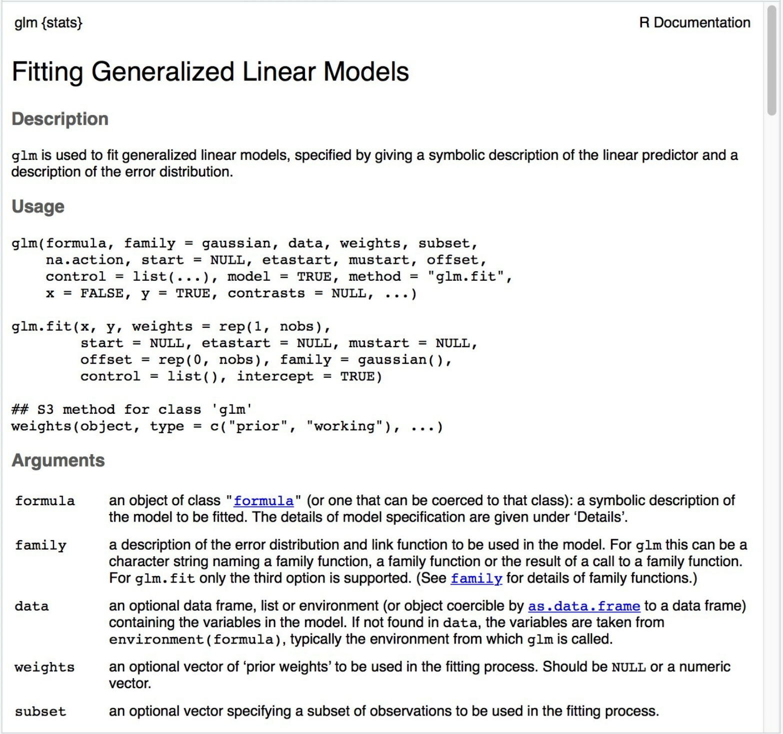
Default vs. customised `t-test`

```
# Default
t.test(formula = age ~ sex,
       data = baselers)

# Customised
t.test(formula = age ~ sex,
       data = baselers,
       alternative = "less", # One sided test
       var.equal = TRUE)     # Assume equal variance
```

Look for optional arguments

```
?glm
```



glm {stats} R Documentation

Fitting Generalized Linear Models

Description

`glm` is used to fit generalized linear models, specified by giving a symbolic description of the linear predictor and a description of the error distribution.

Usage

```
glm(formula, family = gaussian, data, weights, subset,
    na.action, start = NULL, etastart, mustart, offset,
    control = list(...), model = TRUE, method = "glm.fit",
    x = FALSE, y = TRUE, contrasts = NULL, ...)

glm.fit(x, y, weights = rep(1, nobs),
        start = NULL, etastart = NULL, mustart = NULL,
        offset = rep(0, nobs), family = gaussian(),
        control = list(), intercept = TRUE)

## S3 method for class 'glm'
weights(object, type = c("prior", "working"), ...)
```

Arguments

`formula`	an object of class "`formula`" (or one that can be coerced to that class): a symbolic description of the model to be fitted. The details of model specification are given under 'Details'.
`family`	a description of the error distribution and link function to be used in the model. For `glm` this can be a character string naming a family function, a family function or the result of a call to a family function. For `glm.fit` only the third option is supported. (See `family` for details of family functions.)
`data`	an optional data frame, list or environment (or object coercible by `as.data.frame` to a data frame) containing the variables in the model. If not found in `data`, the variables are taken from `environment(formula)`, typically the environment from which `glm` is called.
`weights`	an optional vector of 'prior weights' to be used in the fitting process. Should be `NULL` or a numeric vector.
`subset`	an optional vector specifying a subset of observations to be used in the fitting process.

Default vs. customised `glm()` (Generalized linear model)

```
# Default
glm(formula = income ~ age + education,
    data = baselers)

# Customised
glm(formula = eyecor ~ age + education,
    data = baselers,
    family = "binomial")  # Logistic regression
```

Default vs. customised `t-test`

```
# Default
t.test(formula = age ~ sex,
       data = baselers)

# Customised
t.test(formula = age ~ sex,
       data = baselers,
       alternative = "less", # One sided test
       var.equal = TRUE)     # Assume equal variance
```

Simple hypothesis tests

All of the basic **one and two sample hypothesis tests** are included in the `stats` package.

These tests take either a **formula** for the argument `formula`, or **individual vectors** for the arguments `x`, and `y`

Hypothesis Test	R Function
t-test	`t.test()`
Correlation Test	`cor.test()`
Chi-Square Test	`chisq.test()`

t-test with `t.test()`

```
# 1-sample t-test
t.test(x = baselers$age,
       mu = 40)  # Mean under H0

# 2-sample t-test (Output hidden)
t.test(formula = income ~ sex,
       data = baselers)
```

```
## 
##     One Sample t-test
## 
## data:  baselers$age
## t = 28, df = 10000, p-value <2e-16
## alternative hypothesis: true mean is not equal to 40
## 95 percent confidence interval:
##  44.29 44.93
## sample estimates:
## mean of x 
##     44.61
```

Simple hypothesis tests

All of the basic **one and two sample hypothesis tests** are included in the `stats` package.

These tests take either a **formula** for the argument `formula`, or **individual vectors** for the arguments `x`, and `y`

Hypothesis Test	R Function
t-test	`t.test()`
Correlation Test	`cor.test()`
Chi-Square Test	`chisq.test()`

Correlation test with `cor.test()`

```
# Correlation test
cor.test(x = baselers$age,
         y = baselers$income)

# Version using formula (same result as above)
cor.test(formula = ~ age + income,
         data = baselers)
```

```
##
##     Pearson's product-moment correlation
##
## data:  baselers$age and baselers$income
## t = 180, df = 8500, p-value <2e-16
## alternative hypothesis: true correlation is not equal to
## 95 percent confidence interval:
##  0.8882 0.8968
## sample estimates:
##    cor
## 0.8926
```

Regression with `glm()`, `lm()`

How to **create a regression model** predicting, e.g., how much money people spend on `food` as a function of `income`?

Part of the `baselers` dataframe:

food	income	happiness
610	6300	5
1550	10900	7
720	5100	7
680	4200	7
260	4000	5

Generalized regression with `glm()`

```
# food (y) on income (x1) and happiness (x2)
food_glm <- glm(formula = food ~ income + happiness,
                data = baselers)

# Print food_glm
food_glm
```

```
##
## Call:  glm(formula = food ~ income + happiness, data = baselers)
##
## Coefficients:
## (Intercept)       income    happiness
##    -302.089        0.101       52.205
##
## Degrees of Freedom: 8509 Total (i.e. Null);  8507 Residual
##   (1490 observations deleted due to missingness)
## Null Deviance:        1.27e+09
## Residual Deviance: 6.06e+08     AIC: 119000
```

Customising formulas

Include additional independent variables to formulas by "adding" them with `+`

```
# Include multiple terms with +
my_glm <- glm(formula = income ~ food + alcohol + happiness + hiking,
              data = baselers)
```

To **include all variables** in a dataframe, use the catch-all notation `formula = y ~ .`

```
# Use  y ~ . to include ALL variables
my_glm <- glm(formula = income ~ .,
              data = baselers)
```

To include **interaction terms** use `x1 : x2` or `x1 * x2` (also includes main effects) instead of `x1 + x2`

```
# Include an interaction term between food and alcohol
my_glm <- glm(formula = income ~ food * alcohol,
              data = baselers)
```

Exploring statistical objects

Explore statistical objects using **generic** functions such as `print()`, `summary()`, `predict()` and `plot()`.

Generic functions different things depending on the **class label** of the object.

```
# Create statistical object
obj <- STAT_FUN(formula = ...,
                data = ...)

names(obj)        # Elements
print(obj)        # Print
summary(obj)      # Summary
plot(obj)         # Plotting
predict(obj, ..) # Predict
```

```
# Create a glm object
my_glm <- glm(formula = income ~ happiness + age,
              data = baselers)

# print the my_glm object
print(my_glm)
```

```
##
## Call:  glm(formula = income ~ happiness + age, data = baselers)
##
## Coefficients:
## (Intercept)    happiness          age
##        1575         -100          149
##
## Degrees of Freedom: 8509 Total (i.e. Null);  8507 Residual
##   (1490 observations deleted due to missingness)
## Null Deviance:       6.33e+10
## Residual Deviance: 1.28e+10     AIC: 145000
```

Exploring statistical objects

Explore statistical objects using **generic** functions such as `print()`, `summary()`, `predict()` and `plot()`.

Generic functions different things depending on the **class label** of the object.

```
# Create statistical object
obj <- STAT_FUN(formula = ...,
                data = ...)

names(obj)        # Elements
print(obj)        # Print
summary(obj)      # Summary
plot(obj)         # Plotting
predict(obj, ..) # Predict
```

```
# Create a glm object
my_glm <- glm(formula = income ~ happiness + age,
              data = baselers)

# Show summary of the my_glm object
summary(my_glm)
```

```
##
## Call:
## glm(formula = income ~ happiness + age, data = baselers)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
##  -4045    -835       3     814    4899
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 1575.497     94.363   16.70  < 2e-16 ***
## happiness   -100.431     12.520   -8.02  1.2e-15 ***
## age          149.312      0.815  183.31  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Exploring statistical objects

Explore statistical objects using **generic** functions such as `print()`, `summary()`, `predict()` and `plot()`.

Generic functions different things depending on the **class label** of the object.

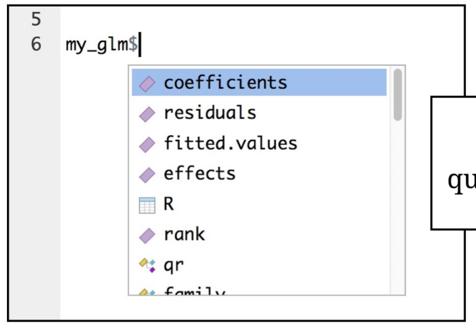
```
# Create statistical object
obj <- STAT_FUN(formula = ...,
                data = ...)

names(obj)        # Elements
print(obj)        # Print
summary(obj)      # Summary
plot(obj)         # Plotting
predict(obj, ..) # Predict
```

Many **statistical objects are lists**. Show elements with `names()`, access them with **$**.

```
# What are the named elements
names(my_glm)
```

```
##  [1] "coefficients"      "residuals"          "fitted.values"      "e
##  [6] "rank"              "qr"                 "family"             "l
```



RStudio Tip!
Hit ‘tab’ after $ to quickly see the named elements of a list

Exploring statistical objects

Explore statistical objects using **generic** functions such as `print()`, `summary()`, `predict()` and `plot()`.

Generic functions different things depending on the **class label** of the object.

```
# Create statistical object
obj <- STAT_FUN(formula = ...,
                data = ...)

names(obj)        # Elements
print(obj)        # Print
summary(obj)      # Summary
plot(obj)         # Plotting
predict(obj, ..) # Predict
```

```
# Look at coefficients
my_glm$coefficients
```

```
## (Intercept)   happiness         age
##      1575.5      -100.4       149.3
```

```
# First 5 fitted values
my_glm$fitted.values
```

```
##     1     2     3     4     5     6     7     8
##  7643 10578  5501  4904  4657 10279 11373  6994
```

```
# First 5 residuals
my_glm$residuals
```

```
##       1       2       3       4       5       6
## -1343.1   322.2  -401.2  -703.9  -656.8  1120.8
```

predict

`predict(model, newdata)` allows you to use your `model` to **predict outcomes** for `newdata`.

```
last_year
```

id	age	fitness	tattoos	income
1	44	7	6	6300
2	65	8	5	10900

```
this_year
```

id	age	fitness	tattoos	income
101	21	3	4	NA
102	23	6	8	NA

Fit `model` based on `leastyear`

```
# Create regression model predicting income
model <- lm(formula = income ~ age + tattoos,
            data = lastyear)

model$coefficients
```

```
## (Intercept)         age     tattoos
##      1418.3       145.7      -175.5
```

Now use `model` to `predict` values for `thisyear`

```
# Predict the income of people in thisyear
predict(object = model,
        newdata = thisyear)
```

```
##    1    2
## 3776 3366
```

tidy

The `tidy()` function from the `broom` package **converts** the most important results of many statistical object like "glm" to a **data frame**.

```r
# install and load broom
install.packages('broom')
library(broom)
```

```r
# Original printout
my_glm
```

```
##
## Call:  glm(formula = income ~ happiness + age, data = baselers
##
## Coefficients:
## (Intercept)    happiness          age
##        1575         -100          149
##
## Degrees of Freedom: 8509 Total (i.e. Null);  8507 Residual
##   (1490 observations deleted due to missingness)
## Null Deviance:        6.33e+10
## Residual Deviance: 1.28e+10     AIC: 145000
```

```r
# Tidy printout
tidy(my_glm)
```

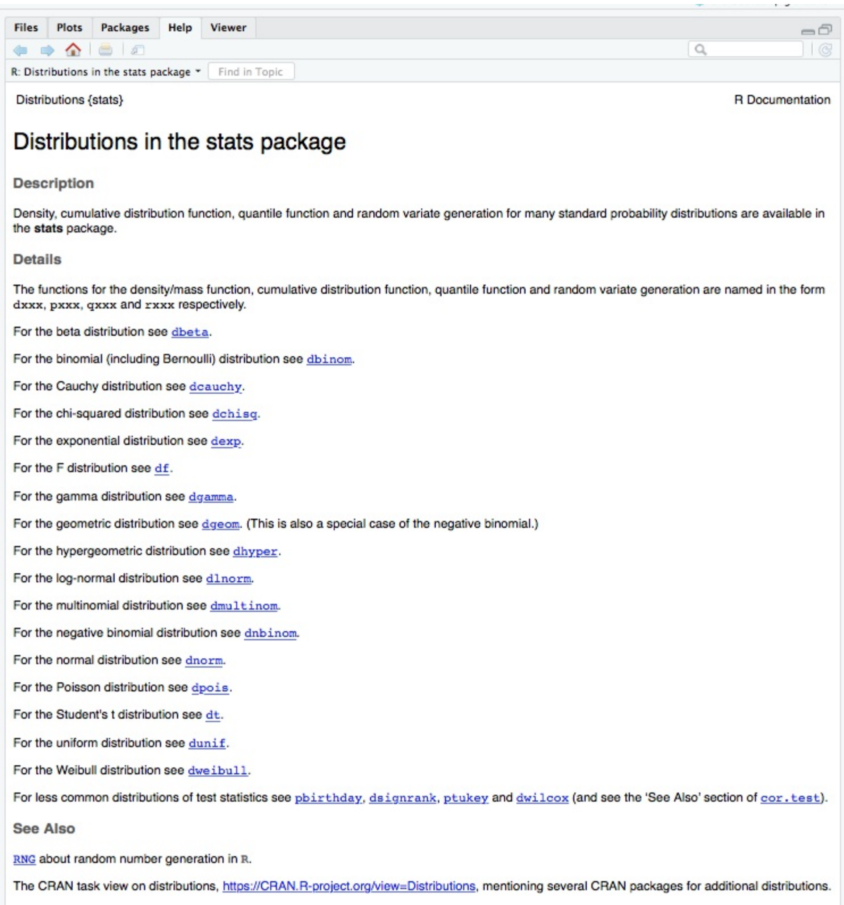
```
## # A tibble: 3 x 5
##   term        estimate std.error statistic  p.value
##   <chr>          <dbl>     <dbl>     <dbl>    <dbl>
## 1 (Intercept)    1575.    94.4       16.7  1.33e-61
## 2 happiness      -100.    12.5       -8.02 1.18e-15
## 3 age             149.     0.815    183.   0.
```

Sampling Functions

R gives you a host of functions for sampling data from common **statistical distributions** (see `?distributions`).

Use these to easily **simulate data**:

Distribution	R Function
Normal	`rnorm()`
Uniform	`runif()`
Beta	`rbeta()`
Binomial	`rbinom()`



sample()

Use `sample()` to **draw a random sample** from a vector.

```
# Simulate 8 flips of a fair coin
coin_flips <- sample(x = c("H", "T"),
                     size = 8,
                     prob = c(.5, .5),
                     replace = TRUE)

coin_flips
```

```
<<<<<<< HEAD
## [1] "T" "T" "H" "H" "T" "H" "T" "T"
=======
## [1] "H" "T" "H" "H" "T" "T" "T" "H"
>>>>>>> 57343262f063e289e06827d36f156b22e7edc7dd
```

```
# Table of counts
table(coin_flips)
```

```
## coin_flips
## H T
```

The Birthday problem

```
# Create an empty room
birthdays <- c()

# While none of the birthdays are the same,
#  keep adding new ones
while(all(!duplicated(birthdays))) {

  # Get new birthday
  new_day <- sample(x = 1:365, size = 1)

  # Add new_day to birthdays
  birthdays <- c(birthdays, new_day)

}

# Done! How many are in the room??
length(birthdays)
```

```
<<<<<<< HEAD
## [1] 13
=======
```

`rnorm`, `runif()`, `...`

Use the `rnorm()`, `runif()`, `...` functions to draw random **samples from probability distributions**.

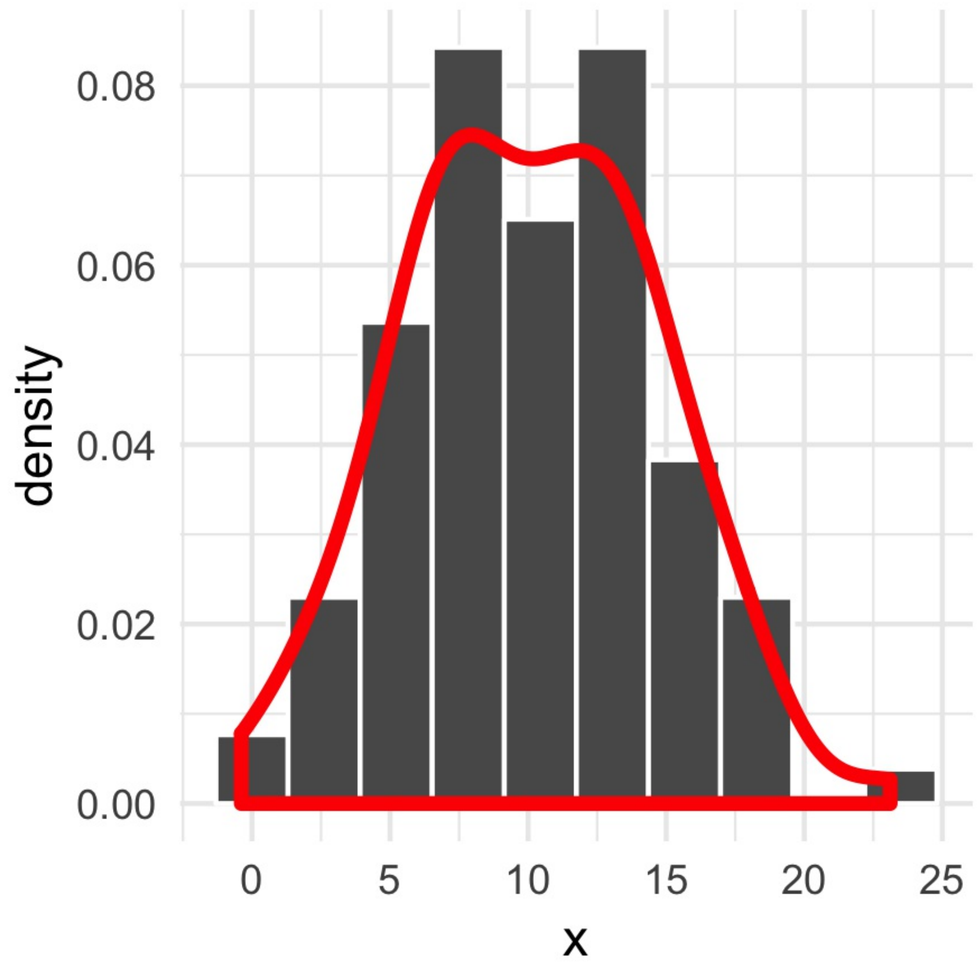
```
# Random sample from Normal distribution
mysamp <- rnorm(n = 100,    # Number of samples
                mean = 10, # Mean of pop
                sd = 5)    # SD of pop ...

mysamp[1:5]   # First 5 values
```

```
<<<<<<< HEAD
## [1]  8.567 14.250 15.241  3.318  7.258
=======
## [1] 13.317  8.822 18.485 10.627 19.035
>>>>>>> 57343262f063e289e06827d36f156b22e7edc7dd
```

```
mean(mysamp) # Mean
```

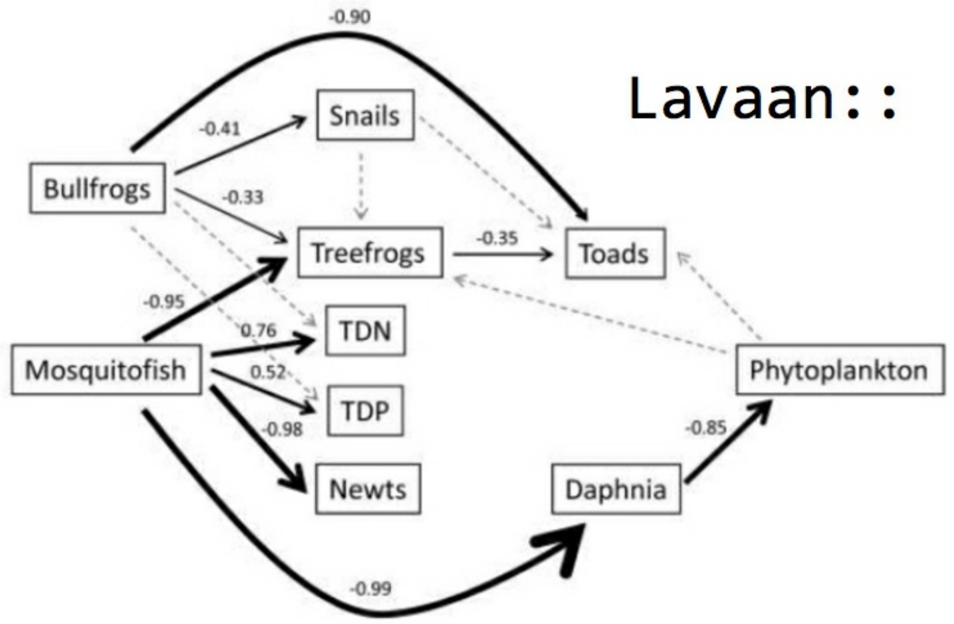
```
<<<<<<< HEAD
## [1] 10.13
=======
```



Other great statistics packages

BayesFactor
An R package for Bayesian data analysis

package	Description
afex	Factorial experiments
lme4	Mixed effects models
rstanarm	Bayesian mixed effects models
BayesFactor	Bayesian Models
forecast	Time series
lavaan	Latent variable and structural equation modelling



Summary

1 - There are **packages for every statistical procedure** you can imagine in R.

2 - Most have **formula** and **data** arguments (among many others).

3 - Use **help files** to understand the arguments of functions!

4 - Once you've created a statistical object, use **generic functions** to explore it: `print()`, `names()`, `summary()`, etc.

5 - Use **random sampling** functions to run simulations.

```
?t.test
```

t.test {stats} R Documentation

Student's t-Test

Description

Performs one and two sample t-tests on vectors of data.

Usage

```
t.test(x, ...)

## Default S3 method:
t.test(x, y = NULL,
       alternative = c("two.sided", "less", "greater"),
       mu = 0, paired = FALSE, var.equal = FALSE,
       conf.level = 0.95, ...)

## S3 method for class 'formula'
t.test(formula, data, subset, na.action, ...)
```

Arguments

`x`	a (non-empty) numeric vector of data values.
`y`	an optional (non-empty) numeric vector of data values.
`alternative`	a character string specifying the alternative hypothesis, must be one of `"two.sided"` (default), `"greater"` or `"less"`.

Practical

Link to practical