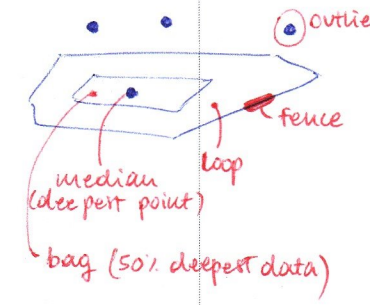
```
################################################################################################
###                                          DEPTH                                           ###
################################################################################################
library(MASS)
library(rgl)
library(DepthProc)
library(hexbin)
#library(depth) # if "Simplicial/Liu" or "Oja" depth

data = cars
head(data)
plot(data, pch=19, cex=.5)
plot(hexbin(data, xbins=10)) # Hexagonal binning to have an idea of the density of the data

### ------------------------------------------------------------------------------------------
### Tukey (or: Mahalanobis, Euclidean)
### ------------------------------------------------------------------------------------------
### Depths
method_name = 'Tukey'
deep        = depth(data, method=method_name)
hist(deep)

### Depth of a point relative to the data
depth(c(0,0), data, method=method_name)

### Median (deepest point)
depthMedian(data, depth_params=list(method=method_name))

### Plot of depths
depthContour(data, depth_params=list(method=method_name), graph_params=list(main='Contours'))
depthContour(data, depth_params=list(method=method_name), points=T)
#col = colorRampPalette(c('white', 'navy'))
#depthContour(data, depth_params=list(method=method_name), points=T, colors=col)
#depthContour(data, depth_params=list(method=method_name), levels=20)
#depthContour(data, depth_params=list(method=method_name), pmean=F, pdmedian=F)

depthPersp(data, depth_params=list(method=method_name))
#depthPersp(data, depth_params=list(method=method_name), plot_method='rgl')

### Plots with probabilistic meaning
#source('depthPredictiveContour.R')
#depthPredictiveContour(data)

### Depth of the mean vs. median
data_mean = colMeans(data)
data_med  = depthMedian(data, depth_params=list(method=method_name))
depth(data_mean, data, method=method_name)
depth(data_med,  data, method=method_name)
plot(data, pch=19, cex=0.5)
points(data_mean[1], data_mean[2], pch=19, col='blue')
points(data_med[1],  data_med[2],  pch=19, col='red')

### ------------------------------------------------------------------------------------------
### Outliers detection
### ------------------------------------------------------------------------------------------
# Bagplot
aplpack::bagplot(data, cex=1)
#aplpack::bagplot(data, cex=1, show.whiskers=F)
#aplpack::bagplot(data, cex=1, show.loophull=F)
#aplpack::bagplot.pairs(data, cex=1) # if dim(data)[2] > 2
bgplot = aplpack::bagplot(data, cex=1)
bgplot$pxy.outlier

### ------------------------------------------------------------------------------------------
### DD-Plot
### ------------------------------------------------------------------------------------------
data_1 = cars
data_2 = cars+2
ddPlot(data_1, data_2, depth_params=list(method=method_name))

### ------------------------------------------------------------------------------------------
### Comments
### ------------------------------------------------------------------------------------------
### ---- Depth ----
### Remember that "depthContour" does *not* have any forecasting or probabilistic meaning,
### they're not densities, they're regions determined as union of points for which the depth
### is above a certain trashold.
### Do the isolines underly the correct distribution of points? (Are shapes coherent?)
### ---- Bagplot ----
### What is the shape of the bagplot of a given distribution? Sample from distributions:
###       Beta(alpha,beta)      rbeta
###       Bernoulli(p)          rbinom
###       Binomial(n,p)         rbinom
###       Cauchy(--)            rcauchy
###       Chi-Square(--)        rchisq
###       Exponential(lambda)   rexp
###       F(--)                 rf
###       Gamma(alpha,beta)     rgamma
###       Geometric(--)         rgeom
###       Hypergeometric(--)    rhyper
###       LogNormal             rlnorm
###       Negative Binomial     rnbinom
###       Normal(mu,sd^2)       rnorm
###       Poisson(lambda)       rpois
###       t-student(--)         rt
###       Uniform(a,b)          runif
###       Weibull(lambda,k)     rweibull
```

: Mahalanobis is not robust since it's based on measures of mean and variance, which are not robust (moreover it's not good for asymmetric data)

: check if 2 distributions are similar

no probabilistic meaning

From bagplot:
- location (depth median)
- spread (size of the bag)
- correlation (orientation of the bag)
- skewness (shape of the bag and the loop)
- tails (points near the boundary of the loop and outliers)



robust to presence of outliers (however it doesn't distinguish longer from smaller deviations)

distribution independent

```
##############################################################################################
###                                      SIGN TEST                                         ###
##############################################################################################

### ------------------------------------------------------------------------------------------
### One sample
### ------------------------------------------------------------------------------------------
data  = cars$speed
boxplot(data)
q     = 0.5
c     = 10
n     = length(data)
signs = sign(data>c)
W     = sum(signs==1)

### Two-sided test ---------------------------------------------------------------------------
###    H0: P(X>c)  = q    ==>    H0: W ~ Binom(n,q)         test: binom.test(W,n,q,"two.sided")
###    H1: P(X>c) != q           H1: W ~ Binom(n,p), p!=q
plot(0:n, dbinom(0:n, n, q))
abline(v=c(W,n-W), col='red')
points(0:n, dbinom(0:n,n,q), col=(0:n>=max(W,n-W)|0:n<=min(W,n-W))+1, pch=19)
p_value = 2*(1-pbinom(max(W,n-W)-1, n, q))
p_value

automatic_test = binom.test(W,n,q,"two.sided")
automatic_test$p.value

### One-side test ----------------------------------------------------------------------------
###    H0: P(X>c) = q    ==>    H0: W ~ Binom(n,q)          test: binom.test(W,n,q,"greater")
###    H1: P(X>c) > q           H1: W ~ Binom(n,p), p>q
plot(0:n, dbinom(0:n, n, q))
abline(v=W, col='red')
points(0:n, dbinom(0:n,n,q), col=(0:n>=W)+1, pch=19)
p_value = 1-pbinom(W-1, n, q)
p_value

automatic_test = binom.test(W,n,q,"greater")
automatic_test$p.value

### One-side test ----------------------------------------------------------------------------
###    H0: P(X>c) = q    ==>    H0: W ~ Binom(n,q)           test: binom.test(W,n,q,"less")
###    H1: P(X>c) < q           H1: W ~ Binom(n,p), p<q
plot(0:n, dbinom(0:n, n, q))
abline(v=(n-W), col='red')
points(0:n, dbinom(0:n,n,q), col=(0:n>=(n-W))+1, pch=19)
p_value = 1-pbinom((n-W)-1, n, q)
p_value

automatic_test = binom.test(W,n,q,"less")
automatic_test$p.value

### ------------------------------------------------------------------------------------------
### Two samples-paired   <-- we're not testing the equality in distribution but the Hp. that in a random pair X has the same probability of being longer or smaller than Y
### ------------------------------------------------------------------------------------------
data  = cars                    # = [X,Y]
diff  = data[,1] - data[,2]     # = X - Y
boxplot(diff)
q     = 0.5
n     = length(diff)
signs = sign(diff)
W     = sum(signs==1)

### Two-sided test ---------------------------------------------------------------------------
###    H0: P(X>Y)  = q    ==>    H0: P(Z>0)  = q    ==>    H0: W ~ Binom(n,q)
###    H1: P(X>Y) != q           H1: P(Z>0) != q           H1: W ~ Binom(n,p), p!=q
###                                                       test: binom.test(W,n,q,"two.sided")
plot(0:n, dbinom(0:n, n, q))
abline(v=c(W,n-W), col='red')
points(0:n, dbinom(0:n,n,q), col=(0:n>=max(W,n-W)|0:n<=min(W,n-W))+1, pch=19)
p_value = 2*(1-pbinom(max(W,n-W)-1, n, q))
p_value

automatic_test = binom.test(W,n,q,"two.sided")
automatic_test$p.value

### One-side test ----------------------------------------------------------------------------
###    H0: P(X>Y) = q    ==>    H0: P(Z>0) = q    ==>    H0: W ~ Binom(n,q)
###    H1: P(X>Y) > q           H1: P(Z>0) > q           H1: W ~ Binom(n,p), p>q
###                                                      test: binom.test(W,n,q,"greater")
plot(0:n, dbinom(0:n, n, q))
abline(v=W, col='red')
points(0:n, dbinom(0:n,n,q), col=(0:n>=W)+1, pch=19)
p_value = 1-pbinom(W-1, n, q)
p_value

automatic_test = binom.test(W,n,q,"greater")
automatic_test$p.value

### One-side test ----------------------------------------------------------------------------
###    H0: P(X>Y) = q    ==>    H0: P(Z>0) = q    ==>    H0: W ~ Binom(n,q)
###    H1: P(X>Y) < q           H1: P(Z>0) < q           H1: W ~ Binom(n,p), p<q
###                                                      test: binom.test(W,n,q,"less")
plot(0:n, dbinom(0:n, n, q))
abline(v=(n-W), col='red')
points(0:n, dbinom(0:n,n,q), col=(0:n>=(n-W))+1, pch=19)
p_value = 1-pbinom((n-W)-1, n, q)
p_value

automatic_test = binom.test(W,n,q,"less")
automatic_test$p.value
```

Takes into account the magnitude of the deviations (through a team contest)

```
##############################################################################################
###                                                                                        ###
###                                        RANK TEST                                       ###
###                                     Mann-Whitney U test                                ###
###                                                                                        ###
##############################################################################################
seed = 100
B    = 1000

### ------------------------------------------------------------------------------------------
### Two samples
### ------------------------------------------------------------------------------------------
### Two-sided test
###    H0: P(data_1 > data_2)  = 0.5
###    H1: P(data_1 > data_2) != 0.5
data_1    = cars$speed
data_2    = cars$dist
data_pool = c(data_1, data_2)
n1        = length(data_1)
n2        = length(data_2)
n         = n1 + n2
ranks     = rank(data_pool)

R1 = sum(ranks[1:length(data_1)])
R2 = sum(ranks[(length(data_1)+1):length(ranks)])
U1 = R1 - n1*(n1+1)/2
U2 = R2 - n2*(n2+1)/2

# MC computation of the p-value
set.seed(seed)
U1.sim = numeric(B)
U2.sim = numeric(B)
for (k in 1:B){
  ranks.temp = sample(1:n)
  R1.temp    = sum(ranks.temp[1:n1])
  R2.temp    = sum(ranks.temp[(n1+1):(n1+n2)])
  U1.temp    = R1.temp - n1*(n1+1)/2
  U2.temp    = R2.temp - n2*(n2+1)/2
  U1.sim[k]  = U1.temp
  U2.sim[k]  = U2.temp
}

par(mfrow=c(2,1))
hist(U1.sim, xlim=c(0,n1*n2))
abline(v=c(U1,U2), col='red')
abline(v=n1*n2/2, lwd=3)

hist(U2.sim, xlim=c(0,n1*n2))
abline(v=c(U1,U2), col='red')
abline(v=n1*n2/2, lwd=3)

U.star  = max(U1, U2)
p_value = 2 * sum(U1.sim >= U.star)/B
p_value
```

It weights different magnitudes of deviations

```
#############################################################################################
###                                  SIGNED RANK TEST                                     ###
###                              Wilcoxon Signed Rank W test                              ###
#############################################################################################
seed = 100
B    = 1000

### -----------------------------------------------------------------------------------------
### One sample
### -----------------------------------------------------------------------------------------
data    = cars$speed
q       = 0.5
c       = 10
n       = length(data)
ranks   = rank(abs(data-c))
W.plus  = sum(ranks[data>c])
W.minus = sum(ranks[data<c])   # "W.plus + W.minus" should be equal to "n*(n+1)/2
W       = W.plus - W.minus

# MC computation of the p-value
set.seed(seed)
W.sim = numeric(B)
for(k in 1:B){
  ranks.temp = sample(1:n)
  signs.temp = 2*rbinom(n, 1, q) - 1
  W.temp     = sum(signs.temp*ranks.temp)
  W.sim[k]   = W.temp
}

hist(W.sim, xlim=c(-n*(n+1)/2, n*(n+1)/2))
abline(v=W, col='red', lwd=3)
abline(v=0, lwd=3)

### Two-sided test ------------------------------------------------------------------------
###    H0: P(X>c)  = q
###    H1: P(X>c) != q
p_value = 2*sum(W.sim>= abs(W))/B
p_value

### One-side test -------------------------------------------------------------------------
###    H0: P(X>c) = q
###    H1: P(X>c) > q
p_value = sum(W.sim>=abs(W))/B
p_value

### One-side test -------------------------------------------------------------------------
###    H0: P(X>c) = q
###    H1: P(X>c) < q
p_value = sum(W.sim<=abs(W))/B
p_value

### -----------------------------------------------------------------------------------------
### Two samples-paired
### -----------------------------------------------------------------------------------------
data    = cars                      # = [X,Y]
diff    = data[,1] - data[,2]       # = X - Y
boxplot(diff)
q       = 0.5
n       = length(diff)
ranks   = rank(abs(diff))
W.plus  = sum(ranks[diff>0])
W.minus = sum(ranks[diff<0])   # "W.plus + W.minus" should be equal to "n*(n+1)/2
W       = W.plus - W.minus

# MC computation of the p-value
set.seed(seed)
W.sim = numeric(B)
for(k in 1:B){
  ranks.temp = sample(1:n)
  signs.temp = 2*rbinom(n, 1, q) - 1
  W.temp     = sum(signs.temp*ranks.temp)
  W.sim[k]   = W.temp
}

hist(W.sim, xlim=c(-n*(n+1)/2, n*(n+1)/2))
abline(v=W, col='red', lwd=3)
abline(v=0, lwd=3)

### Two-sided test ------------------------------------------------------------------------
###    H0: P(X>Y)  = q    ==>    H0: P(Z>0)  = q
###    H1: P(X>Y) != q           H1: P(Z>0) != q
p_value = 2*sum(W.sim>= abs(W))/B
p_value

### One-side test -------------------------------------------------------------------------
###    H0: P(X>Y) = q    ==>    H0: P(Z>0) = q
###    H1: P(X>Y) > q           H1: P(Z>0) > q
p_value = sum(W.sim>=abs(W))/B
p_value

### One-side test -------------------------------------------------------------------------
###    H0: P(X>Y) = q    ==>    H0: P(Z>0) = q
###    H1: P(X>Y) < q           H1: P(Z>0) < q
p_value = sum(W.sim<=abs(W))/B
p_value
```

```
##############################################################################################
###                                                                                        ###
###                                    PERMUTATION TEST                                    ###
###                                Two independent samples (1-dim)                         ###
##############################################################################################
seed = 100
B    = 1000

### Two-sided test ---------------------------------------------------------------------------
###    H0: X1  =^d X2
###    H1: X1 !=^d X2
### Test statistic: abs(mean(data_1)-mean(data_2)) // or: abs(median(data_1)-median(data_2))
data_1 = cars$speed
data_2 = cars$dist
n1     = length(data_1)
n2     = length(data_2)
n      = n1+n2

set.seed(seed)
data_pool      = c(data_1, data_2)
perm           = sample(1:n)
data_pool_perm = data_pool[perm]
data_1_perm    = data_pool_perm[1:n1]
data_2_perm    = data_pool_perm[(n1+1):n]

par(mfrow=c(1,2))
boxplot(data_1, data_2, main='Original')
boxplot(data_1_perm, data_2_perm, main='Permuted')

# CMC to estimate the p-value
T0     = abs(mean(data_1)-mean(data_2))
T_stat = numeric(B)
for(k in 1:B){
  perm           = sample(1:n)
  data_pool_perm = data_pool[perm]
  data_1_perm    = data_pool_perm[1:n1]
  data_2_perm    = data_pool_perm[(n1+1):n]
  T_stat[k]      = abs(mean(data_1_perm)-mean(data_2_perm))
}

# Permutational distribution of T
hist(T_stat, xlim=range(c(T_stat,T0)), breaks=30)
abline(v=T0, col=3, lwd=2)
plot(ecdf(T_stat), xlim=range(c(T_stat,T0)))
abline(v=T0, col=3, lwd=2)

# p-value
p_value = sum(T_stat>=T0)/B
p_value

### One-side test ----------------------------------------------------------------------------
###    H0: var(data_1) = var(data_2)                      (we think data_1 has higher var)
###    H1: var(data_1) > var(data_2)
### Test statistic: (s^2)_1 / (s^2)_2
data_1    = cars$speed
data_2    = cars$dist
n1        = length(data_1)
n2        = length(data_2)
n         = n1+n2
data_pool = c(data_1, data_2)

# CMC to estimate the p-value
set.seed(seed)
T0     = var(data_1)/var(data_2)
T_stat = numeric(B)
for(k in 1:B){
  perm           = sample(1:n)
  data_pool_perm = data_pool[perm]
  data_1_perm    = data_pool_perm[1:n1]
  data_2_perm    = data_pool_perm[(n1+1):n]
  T_stat[k]      = var(data_1_perm)/var(data_2_perm)
}

# Permutational distribution of T
hist(T_stat, xlim=range(c(T_stat,T0)), breaks=30)
abline(v=T0, col=3, lwd=2)
plot(ecdf(T_stat), xlim=range(c(T_stat,T0)))
abline(v=T0, col=3, lwd=2)

# p-value
p_value = sum(T_stat>=T0)/B
p_value

### ------------------------------------------------------------------------------------------
### More informations
### ------------------------------------------------------------------------------------------
factorial(n)                                  # cardinality of the permutational space
factorial(n)/(2*factorial(n1)*factorial(n2))  # number of distinct values of T
1/(factorial(n))/(2*factorial(n1)*factorial(n2))  # minimum achievable p-value
```

If we want to test the equality in distribution we should choose the appropriate test:

- if we think that the two distr. may be shifted:
  $T = |\bar{X}_1 - \bar{X}_2|$
- if we think that the two distr. may have diff. variance:
  $T = S_1^2 / S_2^2$

```
#############################################################################################
###                                    PERMUTATION TEST                                   ###
###                                Two independent samples (n-dim)                        ###
#############################################################################################
seed = 100
B    = 1000

### Two-sided test
###    H0: X1  =^d X2
###    H1: X1 !=^d X2
### Test statistic: squared distance between the two sample mean/median vectors
data_1    = cars
data_2    = cars+1
n1        = dim(data_1)[1]
n2        = dim(data_2)[1]
n         = n1+n2
data_pool = rbind(data_1, data_2)

# CMC to estimate the p-value
set.seed(seed)
data_1_mean = colMeans(data_1)    # or: colMedianians(data_1)
data_2_mean = colMeans(data_2)
T0          = as.numeric((data_1_mean-data_2_mean) %*% (data_1_mean-data_2_mean))
T_stat      = numeric(B)
for(k in 1:B){
  perm             = sample(1:n)
  data_pool_perm   = data_pool[perm,]
  data_1_perm      = data_pool_perm[1:n1,]
  data_2_perm      = data_pool_perm[(n1+1):n,]
  data_1_perm_mean = colMeans(data_1_perm)
  data_2_perm_mean = colMeans(data_2_perm)
  T_stat[k]        = as.numeric((data_1_perm_mean-data_2_perm_mean) %*%
                                (data_1_perm_mean-data_2_perm_mean))
}

# Permutational distribution of T
hist(T_stat, xlim=range(c(T_stat,T0)), breaks=30)
abline(v=T0, col=3, lwd=2)
plot(ecdf(T_stat), xlim=range(c(T_stat,T0)))
abline(v=T0, col=3, lwd=2)

# p-value
p_value = sum(T_stat>=T0)/B
p_value
```

```r
##############################################################################################
###                                    PERMUTATION TEST                                    ###
###                                  Center of simmetry                                    ###
##############################################################################################
seed = 100
B    = 1000

### ------------------------------------------------------------------------------------------
### One dimensional
### ------------------------------------------------------------------------------------------
### Two-sided test
###    H0: "center" is the center of simmetry
###    H1: "center" is *not* the center of simmetry
### Test statistic: distance between the sample mean/median and the center
data   = cars$dist
n      = length(data)
center = c(42)
boxplot(data)
plot(data, cex=0.5, pch=19)
points(center, pch=19, col='red')

# CMC to estimate the p-value
set.seed(seed)
data_mean = mean(data)    # or: colMedianians(data)
T0        = abs(data_mean - center)
T_stat    = numeric(B)
for(k in 1:B){
  signs.perm     = rbinom(n, 1, 0.5)*2-1
  data_perm      = center + (data-center)*signs.perm
  data_perm_mean = mean(data_perm)
  T_stat[k]      = abs(data_perm_mean-center)
}

# Permutational distribution of T
hist(T_stat, xlim=range(c(T_stat,T0)), breaks=30)
abline(v=T0, col=3, lwd=2)
plot(ecdf(T_stat), xlim=range(c(T_stat,T0)))
abline(v=T0, col=3, lwd=2)

# p-value
p_value = sum(T_stat>=T0)/B
p_value

### ------------------------------------------------------------------------------------------
### Multi dimensional
### ------------------------------------------------------------------------------------------
### Two-sided test
###    H0: "center" is the center of simmetry
###    H1: "center" is *not* the center of simmetry
### Test statistic: squared distance between the sample mean/median and the center (other on notes)
data   = cars
n      = dim(data)[1]
p      = dim(data)[2]
center = c(15,1)
boxplot(data)
plot(data, cex=0.5, pch=19)
points(center[1], center[2], pch=19, col='red')

# CMC to estimate the p-value
set.seed(seed)
data_mean = colMeans(data)    # or: colMedianians(data)
T0        = as.numeric((data_mean-center) %*% (data_mean-center))
T_stat    = numeric(B)
for(k in 1:B){
  signs.perm     = rbinom(n, 1, 0.5)*2-1
  data_perm      = center + (data-center)*matrix(signs.perm, nrow=n, ncol=p, byrow=FALSE)
  data_perm_mean = colMeans(data_perm)
  T_stat[k]      = as.numeric((data_perm_mean-center) %*% (data_perm_mean-center))
}

# Permutational distribution of T
hist(T_stat, xlim=range(c(T_stat,T0)), breaks=30)
abline(v=T0, col=3, lwd=2)
plot(ecdf(T_stat), xlim=range(c(T_stat,T0)))
abline(v=T0, col=3, lwd=2)

# p-value
p_value = sum(T_stat>=T0)/B
p_value
```

```
#############################################################################################
###                                    PERMUTATION TEST                                   ###
###                                      Regression                                       ###
#############################################################################################
seed = 100
B    = 1000

### Model: Y = b0 + b1*X1 + b2*X2 + b3*X3 + epsilon
data = iris[,1:4]    # data = [x1, x2, x3, y]
x1   = data[,1]
x2   = data[,2]
x3   = data[,3]
Y    = data[,4]
n    = dim(data)[1]

### -----------------------------------------------------------------------------------------
### Parametric
### -----------------------------------------------------------------------------------------
param = lm(Y ~ x1 + x2 + x3)
summary(param)
qqnorm(param$residuals)
shapiro.test(param$residuals)

### -----------------------------------------------------------------------------------------
### Non Parametric
### -----------------------------------------------------------------------------------------
### |-----------------------------------|
### | Global test (H0: b1 = b2 = b3 = 0) |
### |-----------------------------------|
fit_glob = lm(Y ~ x1 + x2 + x3)
T0_glob  = summary(fit_glob)$f[1]

# CMC to estimate the p-value
set.seed(seed)
T_glob_stat = numeric(B)
for(k in 1:B){
  perm           = sample(1:n)
  Y_perm         = Y[perm]
  T_glob_stat[k] = summary(lm(Y_perm ~ x1 + x2 + x3))$f[1]
}

# Permutational distribution of T
par(mfrow=c(1,2))
hist(T_glob_stat, xlim=range(c(T_glob_stat,T0_glob)), breaks=30)
abline(v=T0_glob, col=3, lwd=2)
plot(ecdf(T_glob_stat), xlim=range(c(T_glob_stat,T0_glob)))
abline(v=T0_glob, col=3, lwd=2)

# p-value
p_value = sum(T_glob_stat>=T0_glob)/B
p_value

### |-------------------------|
### | Test on x1 (H0: b1 = 0) |
### |-------------------------|
fit_glob = lm(Y ~ x1 + x2 + x3)
T0_1     = abs(summary(fit_glob)$coefficients[2,3])
fit_1_H0 = lm(Y ~ x2 + x3)
res_1    = fit_1_H0$residuals

# CMC to estimate the p-value
set.seed(seed)
T_1_stat = numeric(B)
for(k in 1:B){
  perm        = sample(1:n)
  res_1_perm  = res_1[perm]
  Y_perm      = fit_1_H0$fitted.values + res_1_perm
  T_1_stat[k] = abs(summary(lm(Y_perm ~ x1 + x2 +x3))$coefficients[2,3])
}

# Permutational distribution of T
par(mfrow=c(1,2))
hist(T_1_stat, xlim=range(c(T_1_stat,T0_1)), breaks=30)
abline(v=T0_1, col=3, lwd=2)
plot(ecdf(T_1_stat), xlim=range(c(T_1_stat,T0_1)))
abline(v=T0_1, col=3, lwd=2)

# p-value
p_value = sum(T_1_stat>=T0_1)/B
p_value

### |-------------------------|
### | Test on x2 (H0: b2 = 0) |
### |-------------------------|
fit_glob = lm(Y ~ x1 + x2 + x3)
T0_2     = abs(summary(fit_glob)$coefficients[3,3])
fit_2_H0 = lm(Y ~ x1 + x3)
res_2    = fit_2_H0$residuals

# CMC to estimate the p-value
set.seed(seed)
T_2_stat = numeric(B)
for(k in 1:B){
  perm        = sample(1:n)
  res_2_perm  = res_2[perm]
  Y_perm      = fit_2_H0$fitted.values + res_2_perm
  T_2_stat[k] = abs(summary(lm(Y_perm ~ x1 + x2 +x3))$coefficients[3,3])
}
```

```r
# Permutational distribution of T
par(mfrow=c(1,2))
hist(T_2_stat, xlim=range(c(T_2_stat,T0_2)), breaks=30)
abline(v=T0_2, col=3, lwd=2)
plot(ecdf(T_2_stat), xlim=range(c(T_2_stat,T0_2)))
abline(v=T0_2, col=3, lwd=2)

# p-value
p_value = sum(T_2_stat>=T0_2)/B
p_value

### |-------------------------|
### | Test on x3 (H0: b3 = 0) |
### |-------------------------|
fit_glob = lm(Y ~ x1 + x2 + x3)
T0_3     = abs(summary(fit_glob)$coefficients[4,3])
fit_3_H0 = lm(Y ~ x1 + x2)
res_3    = fit_3_H0$residuals

# CMC to estimate the p-value
set.seed(seed)
T_3_stat = numeric(B)
for(k in 1:B){
  perm        = sample(1:n)
  res_3_perm  = res_3[perm]
  Y_perm      = fit_3_H0$fitted.values + res_3_perm
  T_3_stat[k] = abs(summary(lm(Y_perm ~ x1 + x2 +x3))$coefficients[4,3])
}

# Permutational distribution of T
par(mfrow=c(1,2))
hist(T_3_stat, xlim=range(c(T_3_stat,T0_3)), breaks=30)
abline(v=T0_3, col=3, lwd=2)
plot(ecdf(T_3_stat), xlim=range(c(T_3_stat,T0_3)))
abline(v=T0_3, col=3, lwd=2)

# p-value
p_value = sum(T_3_stat>=T0_3)/B
p_value

### |-------------------------|
### | Test on x3 (H0: b3 = c) |
### |-------------------------|
c        = 0.55
fit_glob = lm(Y - c*x3 ~ x1 + x2 + x3)
T0_3     = abs(summary(fit_glob)$coefficients[4,3])

# Permutations of the residuals of the reduced model
# Under H0: Y = b0 + b1*X1 + b2*X2 + c*X3
fit_3_H0 = lm(Y - c*x3 ~ x1 + x2)
res_3    = fit_3_H0$residuals

# CMC to estimate the p-value
set.seed(seed)
T_3_stat = numeric(B)
for(k in 1:B){
  perm        = sample(1:n)
  res_3_perm  = res_3[perm]
  Y_perm      = fit_3_H0$fitted.values + res_3_perm + c*x3
  T_3_stat[k] = abs(summary(lm(Y_perm - c*x3 ~ x1 + x2 +x3))$coefficients[4,3])
}

# Permutational distribution of T
par(mfrow=c(1,2))
hist(T_3_stat, xlim=range(c(T_3_stat,T0_3)), breaks=30)
abline(v=T0_3, col=3, lwd=2)
plot(ecdf(T_3_stat), xlim=range(c(T_3_stat,T0_3)))
abline(v=T0_3, col=3, lwd=2)

# p-value
p_value = sum(T_3_stat>=T0_3)/B
p_value
```

```
#############################################################################################
###                                    PERMUTATION TEST                                   ###
###                                        ANOVA                                          ###
#############################################################################################
seed = 100
B    = 1000

# We can always start from the parametric ANOVA/MANOVA. The assumptions hold either:
#  * each group is normally distributed and the variance is equal among groups
#  * the residuals are normally distributed

### -----------------------------------------------------------------------------------------
### One-way ANOVA (p=1, g=6)
### -----------------------------------------------------------------------------------------
###    H0: X1 =^d .. =^d XG == all the observations belong to the same population
###    H1: exists i,j: Xi !=^d Xj
### Test statistic: from the parametric case
data     = chickwts                  # data = [values, levels]  (check the levels to be factors)
data[,2] = as.factor(data[,2])
n        = dim(data)[1]
g        = nlevels(data[,2])
plot(data[,2], data[,1])

x      = data[,1]
levels = data[,2]

fit = aov(x ~ levels)
summary(fit)
T0  = summary.aov(fit)[[1]][1,4]
T0

# CMC to estimate the p-value
set.seed(seed)
T_stat = numeric(B)
for(k in 1:B){
  perm      = sample(1:n)
  x_perm    = x[perm]
  fit_perm  = aov(x_perm ~ levels)
  T_stat[k] = summary.aov(fit_perm)[[1]][1,4]
}

# Permutational distribution of T
par(mfrow=c(1,2))
hist(T_stat, xlim=range(c(T_stat,T0)), breaks=30)
abline(v=T0, col=3, lwd=2)
plot(ecdf(T_stat), xlim=range(c(T_stat,T0)))
abline(v=T0, col=3, lwd=2)

# p-value
p_value = sum(T_stat>=T0)/B
p_value

### -----------------------------------------------------------------------------------------
### Two-way ANOVA (p=1, g=2, b=2)
### -----------------------------------------------------------------------------------------
###    H0: X1 =^d .. =^d XG == all the observations belong to the same population
###    H1: exists i,j: Xi !=^d Xj (on at least one of the two factors)
### Test statistic: from the parametric case
data     = ToothGrowth               # data = [values, level1, level2]  (check the levels to be factors)
data[,2] = as.factor(data[,2])
data[,3] = as.factor(data[,3])
n        = dim(data)[1]
g        = nlevels(data[,2])
b        = nlevels(data[,3])

par(mfrow=c(1,2))
plot(data[,2], data[,1])
plot(data[,3], data[,1])
table(data[,2], data[,3])

x   = data[,1]
lv1 = data[,2]
lv2 = data[,3]

### |--------------------------------|
### | Test 1: test on the interaction |
### |--------------------------------|
fit_1 = aov(x ~ lv1 + lv2 + lv1:lv2)
summary.aov(fit_1)
T0_1  = summary.aov(fit_1)[[1]][3,4]
T0_1

# CMC to estimate the p-value
# The idea is to permute the residuals under H0
set.seed(seed)
fit_1_H0 = aov(x ~ lv1 + lv2)
res_1    = fit_1_H0$residuals
T_stat_1 = numeric(B)
for(k in 1:B){
  perm        = sample(1:n)
  res_1_perm  = res_1[perm]
  x_1_perm    = fit_1_H0$fitted.values + res_1_perm
  fit_1_perm  = aov(x_1_perm ~ lv1 + lv2 + lv1:lv2)
  T_stat_1[k] = summary(fit_1_perm)[[1]][3,4]
}

# Permutational distribution of T
par(mfrow=c(1,2))
hist(T_stat_1, xlim=range(c(T_stat_1,T0_1)), breaks=30)
abline(v=T0_1, col=3, lwd=2)
```

shapiro.test(fit$residuals)
qqnorm(fit$residuals)

```r
plot(ecdf(T_stat_1), xlim=range(c(T_stat_1,T0_1)))
abline(v=T0_1, col=3, lwd=2)

# p-value
p_value = sum(T_stat_1>=T0_1)/B
p_value

### |------------------------------------------------------------|
### | Test 2: test on lv2 without interaction (considering lv1) |
### |------------------------------------------------------------|
fit_2 = aov(x ~ lv1 + lv2)
summary.aov(fit_2)
T0_2  = summary.aov(fit_2)[[1]][2,4]
T0_2

# CMC to estimate the p-value
# The idea is to permute the residuals under H0
set.seed(seed)
fit_2_H0 = aov(x ~ lv1)
res_2    = fit_2_H0$residuals
T_stat_2 = numeric(B)
for(k in 1:B){
  perm        = sample(1:n)
  res_2_perm  = res_2[perm]
  x_2_perm    = fit_2_H0$fitted.values + res_2_perm
  fit_2_perm  = aov(x_2_perm ~ lv1 + lv2)
  T_stat_2[k] = summary(fit_2_perm)[[1]][2,4]
}

# Permutational distribution of T
par(mfrow=c(1,2))
hist(T_stat_2, xlim=range(c(T_stat_2,T0_2)), breaks=30)
abline(v=T0_2, col=3, lwd=2)
plot(ecdf(T_stat_2), xlim=range(c(T_stat_2,T0_2)))
abline(v=T0_2, col=3, lwd=2)

# p-value
p_value = sum(T_stat_2>=T0_2)/B
p_value

### |------------------------------------------------------------|
### | Test 3: test on lv1 without interaction (considering lv2) |
### |------------------------------------------------------------|
fit_3 = aov(x ~ lv1 + lv2)
summary.aov(fit_3)
T0_3  = summary.aov(fit_3)[[1]][1,4]
T0_3

# CMC to estimate the p-value
# The idea is to permute the residuals under H0
set.seed(seed)
fit_3_H0 = aov(x ~ lv2)
res_3    = fit_3_H0$residuals
T_stat_3 = numeric(B)
for(k in 1:B){
  perm        = sample(1:n)
  res_3_perm  = res_3[perm]
  x_3_perm    = fit_3_H0$fitted.values + res_3_perm
  fit_3_perm  = aov(x_3_perm ~ lv1 + lv2)
  T_stat_3[k] = summary(fit_3_perm)[[1]][1,4]
}

# Permutational distribution of T
par(mfrow=c(1,2))
hist(T_stat_3, xlim=range(c(T_stat_3,T0_3)), breaks=30)
abline(v=T0_3, col=3, lwd=2)
plot(ecdf(T_stat_3), xlim=range(c(T_stat_3,T0_3)))
abline(v=T0_3, col=3, lwd=2)

# p-value
p_value = sum(T_stat_3>=T0_3)/B
p_value

### Other tests: with the same policy we can also perform
### Test 4: test on lv1 *and* lv2 (H0: both *not* relevant)
### Test 5: test on lv1/lv2 without the other level (H0: lv* *not* relevant)

### ------------------------------------------------------------------------------------------
### One-way MANOVA (p=4, g=6)
### ------------------------------------------------------------------------------------------
###    H0: X1 =^d .. =^d XG == all the observations belong to the same population
###    H1: exists i,j: Xi !=^d Xj
### Test statistic: from the parametric case
data   = iris              # data = [values_1, .., values_p, levels]  (check the levels to be factors)
x      = data[,1:4]
levels = data[,5]
n      = dim(data)[1]
p      = dim(x)[2]
g      = nlevels(levels)

fit = manova(as.matrix(x) ~ levels)
summary.manova(fit, test="Wilks")
T0  = -summary.manova(fit, test="Wilks")$stats[1,2]
T0

# CMC to estimate the p-value
set.seed(seed)
T_stat = numeric(B)
for(k in 1:B){
  perm        = sample(1:n)
  levels_perm = levels[perm]
```

```
  fit_perm    = manova(as.matrix(x) ~ levels_perm)
  T_stat[k]   = -summary.manova(fit_perm, test="Wilks")$stat[1,2]
}

# Permutational distribution of T
par(mfrow=c(1,2))
hist(T_stat, xlim=range(c(T_stat,T0)), breaks=30)
abline(v=T0, col=3, lwd=2)
plot(ecdf(T_stat), xlim=range(c(T_stat,T0)))
abline(v=T0, col=3, lwd=2)

# p-value
p_value = sum(T_stat>=T0)/B
p_value
```

```
#############################################################################################
###                                    PERMUTATION TEST                                   ###
###                                   Confidence Intervals                                ###
#############################################################################################
seed  = 100
B     = 1000
alpha = 0.1

### Univariate CI for t-test (mean)
data = data[,1]    # make sure it to be a vector
uni_t_perm = function(data, mu0, B=1000){
  data_trans = data-mu0
  T0         = abs(mean(data_trans))
  T_perm     = numeric(B)
  n          = length(data)
  for(k in 1:B){
    refl      = rbinom(n, 1, 0.5)*2 -1
    T_perm[k] = abs(mean(data_trans*refl))
  }
  return (sum(T_perm>=T0)/B)
}

library(pbapply)
library(parallel)
# Remember to always adjust the grid!
grid = seq(0, 2.5, length.out =100)
cl   = makeCluster(2)
clusterExport(cl, varlist=list("data", "uni_t_perm"))

perm_wrap     = function(grid_point){uni_t_perm(data, grid_point, B=1000)}
pval_function = pbsapply(grid, perm_wrap, cl=cl)
plot(grid, pval_function, type='l')
abline(h=alpha, col='red')
abline(v=range(grid[pval_function>alpha]), col='red')
range(grid[pval_function>alpha])
```

```
#############################################################################################
###                                      BOOTSTRAP                                        ###
###                                  Confidence Intervals                                 ###
#############################################################################################
seed  = 100
B     = 1000
alpha = 0.1

### ---------------------------------------------------------------------------------------
### Naive
### ---------------------------------------------------------------------------------------
grades = read.table('parziali.txt', header=T)
data   = grades[,'PI']
boxplot(data)

# Estimator 1: 0.95 quantile    (remember that 0.5 = median)
# Estimator 2: prob(data>=18)
# Estimator 3: mean
est1 = quantile(data, 0.95)
est2 = sum(data>=18)/length(data)
est3 = mean(data)

# Bootstrap distribution of the estimators
set.seed(seed)
T_boot_est1 = numeric(B)
T_boot_est2 = numeric(B)
T_boot_est3 = numeric(B)
for(k in 1:B){
  data_boot      = sample(data, replace=T)
  T_boot_est1[k] = quantile(data_boot, 0.95)
  T_boot_est2[k] = sum(data_boot>=18)/length(data_boot)
  T_boot_est3[k] = mean(data_boot)
}

par(mfrow=c(1,3))
plot(ecdf(T_boot_est1), main="Quantile")
abline(v=est1, lty=2, col='red')
plot(ecdf(T_boot_est2), main="Probability")
abline(v=est2, lty=2, col='red')
plot(ecdf(T_boot_est3), main="Mean")
abline(v=est3, lty=2, col='red')

### |---------------------------|
### | CI Estimator 1 - quantile |
### |---------------------------|
### Reverse Percentile
par(mfrow=c(1,1))
plot(ecdf(T_boot_est1), main="Estimator 1")
abline(v=est1, lty=2, col='red')

right_quantile = quantile(T_boot_est1, 1-alpha/2)
left_quantile  = quantile(T_boot_est1, alpha/2)
CI_est1 = c(est1-(right_quantile-est1), est1, est1-(left_quantile-est1))
CI_est1
abline(v=CI_est1[c(1,3)], col='red')

### |-----------------------------|
### | CI Estimator 2 - probabiity |
### |-----------------------------|
### Reverse Percentile
par(mfrow=c(1,1))
plot(ecdf(T_boot_est2), main="Estimator 2")
abline(v=est2, lty=2, col='red')

right_quantile = quantile(T_boot_est2, 1-alpha/2)
left_quantile  = quantile(T_boot_est2, alpha/2)
CI_est2 = c(est2-(right_quantile-est2), est2, est2-(left_quantile-est2))
CI_est2
abline(v=CI_est2[c(1,3)], col='red')

### t-intervals (we use a standard deviation formula)
par(mfrow=c(1,1))
plot(ecdf(T_boot_est2), main="Estimator 2")
abline(v=est2, lty=2, col='red')

prob = sum(data>=18)/length(data)

set.seed(seed)
T_boot_est2 = numeric(B)
for(k in 1:B){
  data_boot      = sample(data, replace=T)
  prob_perm      = sum(data_boot>=18)/length(data_boot)
  T_boot_est2[k] = (prob_perm - prob) / sqrt(prob_perm*(1-prob_perm)/length(data))
}

right_t_quantile = quantile(T_boot_est2, 1-alpha/2)
left_t_quantile  = quantile(T_boot_est2, alpha/2)
CI_t_est2 = c(prob-right_t_quantile * sqrt(prob*(1-prob)/length(data)), prob,
              prob-left_t_quantile *  sqrt(prob*(1-prob)/length(data)))
CI_t_est2
abline(v=CI_t_est2[c(1,3)], col='red')

### |-----------------------|
### | CI Estimator 3 - mean |
### |-----------------------|
### Reverse Percentile
par(mfrow=c(1,1))
plot(ecdf(T_boot_est3), main="Estimator 3")
abline(v=est3, lty=2, col='red')
```

```
right_quantile = quantile(T_boot_est3, 1-alpha/2)
left_quantile  = quantile(T_boot_est3, alpha/2)
CI_est3 = c(est3-(right_quantile-est3), est3, est3-(left_quantile-est3))
CI_est3
abline(v=CI_est3[c(1,3)], col='red')

### ------------------------------------------------------------------------------------------
### Parametric (Gaussian)
### ------------------------------------------------------------------------------------------
grades = read.table('parziali.txt', header=T)
data   = grades[,'PI']
boxplot(data)
shapiro.test(data)
#shapiro.test(log(data))  # -> if ok, then data are log-normal

# Estimator 1: 0.95 quantile    (remember that 0.5 = median)
# Estimator 2: prob(data>=18)
# Estimator 3: mean
est1 = quantile(data, 0.95)
est2 = sum(data>=18)/length(data)
est3 = mean(data)

# Bootstrap distribution of the estimators
set.seed(seed)
T_boot_est1 = numeric(B)
T_boot_est2 = numeric(B)
T_boot_est3 = numeric(B)
for(k in 1:B){
  data_boot      = rnorm(length(data), mean(data), sd(data))
  #data_boot      = exp(rnorm(length(data), mean(log(data)), sqrt(var(log(data)))))
  T_boot_est1[k] = quantile(data_boot, 0.95)
  T_boot_est2[k] = sum(data_boot>=18)/length(data_boot)
  T_boot_est3[k] = mean(data_boot)
}

par(mfrow=c(1,3))
plot(ecdf(T_boot_est1), main="Quantile")
abline(v=est1, lty=2, col='red')
plot(ecdf(T_boot_est2), main="Probability")
abline(v=est2, lty=2, col='red')
plot(ecdf(T_boot_est3), main="Mean")
abline(v=est3, lty=2, col='red')

### |---------------------------|
### | CI Estimator 1 - quantile |
### |---------------------------|
### Reverse Percentile
par(mfrow=c(1,1))
plot(ecdf(T_boot_est1), main="Estimator 1")
abline(v=est1, lty=2, col='red')

right_quantile = quantile(T_boot_est1, 1-alpha/2)
left_quantile  = quantile(T_boot_est1, alpha/2)
CI_est1 = c(est1-(right_quantile-est1), est1, est1-(left_quantile-est1))
CI_est1
abline(v=CI_est1[c(1,3)], col='red')

### |-----------------------------|
### | CI Estimator 2 - probabiity |
### |-----------------------------|
### Reverse Percentile
par(mfrow=c(1,1))
plot(ecdf(T_boot_est2), main="Estimator 2")
abline(v=est2, lty=2, col='red')

right_quantile = quantile(T_boot_est2, 1-alpha/2)
left_quantile  = quantile(T_boot_est2, alpha/2)
CI_est2 = c(est2-(right_quantile-est2), est2, est2-(left_quantile-est2))
CI_est2
abline(v=CI_est2[c(1,3)], col='red')

### t-intervals (we use a standard deviation formula)
par(mfrow=c(1,1))
plot(ecdf(T_boot_est2), main="Estimator 2")
abline(v=est2, lty=2, col='red')

prob = sum(data>=18)/length(data)

set.seed(seed)
T_boot_est2 = numeric(B)
for(k in 1:B){
  data_boot      = sample(data, replace=T)
  prob_perm      = sum(data_boot>=18)/length(data_boot)
  T_boot_est2[k] = (prob_perm - prob) / sqrt(prob_perm*(1-prob_perm)/length(data))
}

right_t_quantile = quantile(T_boot_est2, 1-alpha/2)
left_t_quantile  = quantile(T_boot_est2, alpha/2)
CI_t_est2 = c(prob-right_t_quantile * sqrt(prob*(1-prob)/length(data)), prob,
              prob-left_t_quantile *  sqrt(prob*(1-prob)/length(data)))
CI_t_est2
abline(v=CI_t_est2[c(1,3)], col='red')

### |-----------------------|
### | CI Estimator 3 - mean |
### |-----------------------|
### Reverse Percentile
par(mfrow=c(1,1))
plot(ecdf(T_boot_est3), main="Estimator 3")
abline(v=est3, lty=2, col='red')
```

```
right_quantile = quantile(T_boot_est3, 1-alpha/2)
left_quantile  = quantile(T_boot_est3, alpha/2)
CI_est3 = c(est3-(right_quantile-est3), est3, est3-(left_quantile-est3))
CI_est3
abline(v=CI_est3[c(1,3)], col='red')

### ---------------------------------------------------------------------------------------
### Comparison: parametric vs. non parametric
### ---------------------------------------------------------------------------------------
sd   = sd(T_boot_est3)                                  # standard deviation
var  = var(T_boot_est3)                                 # variance
bias = mean(T_boot_est3) - est3                         # bias
MSE  = sd(T_boot_est3)^2+(mean(T_boot_est3)-est3)^2     # MSE

#data.frame("Non Param"=c(var,bias,MSE), "Param"=c(var,bias,MSE))
```

```
##############################################################################################
###                                        BOOTSTRAP                                       ###
###                                        Regression                                      ###
##############################################################################################
seed  = 100
B     = 1000
alpha = 0.1

### ------------------------------------------------------------------------------------------
### Linear Regression
### ------------------------------------------------------------------------------------------
# data = [X, Y]
grades = read.table('parziali.txt', header=T)
data   = grades
x      = data[,1]
y      = data[,2]
fm     = lm(y ~ x)

plot(x,y,asp=1)
abline(coefficients(fm), col='red')
points(x, fitted(fm), col='red', pch=16)
summary(fm)
shapiro.test(fm$residuals)

### Bootstrap distribution for b0 (intercept) and b1 (slope)
fitted_obs = fitted(fm)
res_obs    = residuals(fm)
b0_obs     = coefficients(fm)[1]
b1_obs     = coefficients(fm)[2]

set.seed(seed)
T_boot_b0 = numeric(B)
T_boot_b1 = numeric(B)
for(k in 1:B){
  y_boot       = fitted_obs + sample(res_obs, replace=T)
  fm_boot      = lm(y_boot ~ x)
  T_boot_b0[k] = coefficients(fm_boot)[1]
  T_boot_b1[k] = coefficients(fm_boot)[2]
}

par(mfrow=c(1,2))
plot(ecdf(T_boot_b0), main='Intercept')
abline(v=b0_obs, lty=2, col='red')
plot(ecdf(T_boot_b1), main='Slope')
abline(v=b1_obs, lty=2, col='red')

### Bootstrap standard deviation and covariance
sd(T_boot_b0)
sd(T_boot_b1)
cov(T_boot_b0, T_boot_b1)

### CI (Reverse Percentile)
right_quantile_b0 = quantile(T_boot_b0, 1-alpha/2)
right_quantile_b1 = quantile(T_boot_b1, 1-alpha/2)
left_quantile_b0  = quantile(T_boot_b0, alpha/2)
left_quantile_b1  = quantile(T_boot_b1, alpha/2)

CI_b0 = c(b0_obs-(right_quantile_b0-b0_obs), b0_obs, b0_obs-(left_quantile_b0-b0_obs))
CI_b0
plot(ecdf(T_boot_b0), main='Intercept')
abline(v=b0_obs, lty=2, col='red')
abline(v=CI_b0[c(1,3)], col='red')

CI_b1 = c(b1_obs-(right_quantile_b1-b1_obs), b1_obs, b1_obs-(left_quantile_b1-b1_obs))
CI_b1
plot(ecdf(T_boot_b1), main='Slope')
abline(v=b1_obs, lty=2, col='red')
abline(v=CI_b1[c(1,3)], col='red')

### CI at a given x0
x0                    = 24
mean_x0_obs           = b0_obs + b1_obs * x0
T_boot_mean_x0        = T_boot_b0 + T_boot_b1*x0
right_quantile_mean_x0 = quantile(T_boot_mean_x0, 1-alpha/2)
left_quantile_mean_x0  = quantile(T_boot_mean_x0, alpha/2)
CI_mean_x0            =  c(mean_x0_obs - (right_quantile_mean_x0-mean_x0_obs),
                           mean_x0_obs,
                           mean_x0_obs - (left_quantile_mean_x0-mean_x0_obs))
plot(ecdf(T_boot_mean_x0), main='Conditional mean at x0')
abline(v=mean_x0_obs, lty=2, col='green')
abline(v=CI_mean_x0[c(1,3)], col='green')

### ------------------------------------------------------------------------------------------
### Non Linear Regression
### ------------------------------------------------------------------------------------------
# data = [X, Y]
data = subset(Orange, Orange$Tree==3)
data = data[,c(2,3)]
x    = data[,1]
y    = data[,2]
plot(x,y,pch=19)

logistic = function(t,L,k,midpoint){L/(1+exp((midpoint-t)/k))}
model    = nls(y ~ logistic(x,L,k,midpoint), start=list(L=150, k=500, midpoint=700))
summary(model)
x_grid   = seq(range(x)[1], range(x)[2], length.out=100)
#x_grid   = seq(0, 2500, length.out=100)
y_pred   = predict(model, list(x=x_grid))
plot(x,y,pch=19)
lines(x_grid,y_pred,col='red')
```

$$f(x) = \frac{L}{1+e^{-k(x-x_0)}}$$

```
### Bootstrap distribution for L
fitted_obs = fitted(model)
res_obs    = residuals(model)
L_obs      = coefficients(model)[1]

set.seed(seed)
T_boot_L     = numeric(B)
formula_boot = y_boot ~ logistic(x,L,k,midpoint)
for(k in 1:B){
  y_boot      = fitted_obs + sample(res_obs, replace=T)
  fm_boot     = nls(formula_boot, start=list(L=150, k=500, midpoint=700))
  T_boot_L[k] = coefficients(fm_boot)[1]
}

plot(ecdf(T_boot_L), main='L')
abline(v=L_obs, lty=2, col='red')

### CI (Reverse Percentile)
right_quantile_L = quantile(T_boot_L, 1-alpha/2)
left_quantile_L  = quantile(T_boot_L, alpha/2)
CI_L             = c(L_obs-(right_quantile_L-L_obs), L_obs, L_obs-(left_quantile_L-L_obs))
CI_L
abline(v=CI_L[c(1,3)], col='red')

### In this case L is the asymptot, we can see its effect on the curve:
plot(x,y,pch=19,main='Fitted curve vs. Data', xlim=c(0,2500), ylim=c(20,200))
lines(x_grid,y_pred,col='red')
abline(h=CI_L, col='red')
```

```
#############################################################################################
###                                      BOOTSTRAP                                        ###
###                                Two independent samples (1-dim)                        ###
#############################################################################################
seed  = 100
B     = 1000
alpha = 0.1

grades = read.table('parziali.txt', header=T)
gender = read.table('sesso.txt', header=T)
data_1 = grades[gender==1,'PI']
data_2 = grades[gender==0,'PI']

par(mfrow=c(1,2))
boxplot(data_1, ylim=range(c(data_1,data_2)))
boxplot(data_2, ylim=range(c(data_1,data_2)))

# Bootstrap for medians
T0 = quantile(data_1, 0.5) - quantile(data_2, 0.5)

set.seed(seed)
T_boot = numeric(B)
for(k in 1:B){
  data_1_boot = sample(data_1, replace=T)
  data_2_boot = sample(data_2, replace=T)
  T_boot[k]   = quantile(data_1_boot, 0.5) - quantile(data_2_boot, 0.5)
}

par(mfrow=c(1,1))
plot(ecdf(T_boot))
abline(v=T0, lty=2)

right_quantile = quantile(T_boot, 1-alpha/2)
left_quantile  = quantile(T_boot, alpha/2)
CI_med = c(T0-(right_quantile-T0), T0, T0-(left_quantile-T0))
CI_med
abline(v=CI_med[c(1,3)], col='red')
```

```
#############################################################################################
###                                      BOOTSTRAP                                        ###
###                                   One sample (n-dim)                                  ###
#############################################################################################
seed  = 100
B     = 1000
alpha = 0.1

load('snowlevel.rda')
data = df_1[which(df_1$region=='Aosta_Valley' & df_1$town<10), c(1,2)]  # data = [feature_1, feature_2]

### |---------------------------------|
### | Bootstrap: bidimensional median |
### |---------------------------------|
library(DepthProc)
method_name = 'Tukey'

set.seed(seed)
T0     = depthMedian(data, depth_params=list(method=method_name))
T_boot = cbind(numeric(B), numeric(B))
for(k in 1:B){
  ind        = sample(1:dim(data)[1], replace=T)
  data_boot  = data[ind,]
  T_boot[k,] = depthMedian(data_boot, depth_params=list(method=method_name))
}

### |-------------------|
### | Bias and Variance |
### |-------------------|
apply(T_boot, 2, var)           # Variance
apply(T_boot, 2, mean) - T0     # Bias

### |----------------------|
### | Confidence Intervals |
### |----------------------|
# Reverse Percentile - 1
right_quantile = quantile(T_boot[,1], 1-alpha/2)
left_quantile  = quantile(T_boot[,1], alpha/2)
CI_1 = c(T0[1]-(right_quantile-T0[1]), T0[1], T0[1]-(left_quantile-T0[1]))
CI_1

# Reverse Percentile - 2
right_quantile = quantile(T_boot[,2], 1-alpha/2)
left_quantile  = quantile(T_boot[,2], alpha/2)
CI_2 = c(T0[2]-(right_quantile-T0[2]), T0[2], T0[2]-(left_quantile-T0[2]))
CI_2
```

```
#############################################################################################
###                                      BOOTSTRAP                                        ###
###                                   Test and p-values                                   ###
#############################################################################################
seed  = 100
B     = 1000
alpha = 0.1

### One dimensional
###    H0: median(data)=0 vs. H1: median(data)!=0
data = stabledist::rstable(1000,1.8,0)
hist(data)
boxplot(data)

library(pbapply)
library(parallel)
set.seed(seed)
T0      = median(data)
T_boot  = numeric(B)
cl      = makeCluster(2)
wrapper = function(dummy){T_boot = median(sample(data, replace=T))}
clusterExport(cl=cl, list('data'))
T_boot  = pbsapply(T_boot, wrapper, cl=cl)

# Confidence Interval
right_quantile = quantile(T_boot, 1-alpha/2)
left_quantile  = quantile(T_boot, alpha/2)
CI = c(T0-(right_quantile-T0), T0, T0-(left_quantile-T0))
CI
plot(ecdf(T_boot))
abline(v=CI, col='red')

# P-value
alpha_grid = seq(0.001, 0.5, length.out=100)
CI_calc = function(alpha){
  right_quantile = quantile(T_boot, 1-alpha/2)
  left_quantile  = quantile(T_boot, alpha/2)
  CI = c(T0-(right_quantile-T0), T0-(left_quantile-T0))
  names(CI) = c('lwr','upr')
  return(CI)
}

CI_list = pblapply(alpha_grid, CI_calc)
CI_mat  = dplyr::bind_rows(CI_list)
check   = CI_mat[,1]>0 | CI_mat[,2]<0
(alpha_grid[check])[1]   # <- p-value
```

we're checking H_0: median = 0
(we're checking if $0 \in CI_\alpha$)

```
#############################################################################################
###                                NON PARAMETRIC REGRESSION                              ###
#############################################################################################
load('nlr_data.rda')
data = data.frame("age"=Wage$age,"wage"=Wage$wage)    # data = [X, Y]
x    = data[,1]
y    = data[,2]

### ---------------------------------------------------------------------------------------
### Linear model
### ---------------------------------------------------------------------------------------
m_linear = lm(y ~ x)
summary(m_linear)
x_grid   = seq(range(x)[1], range(x)[2], length.out=100)
y_pred   = predict(m_linear, list(x=x_grid), se=T)
se.bands = cbind(y_pred$fit+2*y_pred$se.fit, y_pred$fit-2*y_pred$se.fit)
plot(x, y, xlim=range(x_grid), cex=.5)
lines(x_grid, y_pred$fit, lwd=2, col='blue')
matlines(x_grid, se.bands, lwd=1, col='blue', lty=3)
par(mfrow=c(2,2))
plot(m_linear)
dev.off()

### ---------------------------------------------------------------------------------------
### Polynomial model (orthogonal polynomial)
### ---------------------------------------------------------------------------------------
m_list = lapply(1:10, function(degree){lm(y ~ poly(x, degree=degree))})
do.call(anova, m_list)
m_poly = m_list[[4]]
summary(m_poly)
x_grid   = seq(range(x)[1], range(x)[2], length.out=100)
y_pred   = predict(m_poly, list(x=x_grid), se=T)
se.bands = cbind(y_pred$fit+2*y_pred$se.fit, y_pred$fit-2*y_pred$se.fit)
plot(x, y, xlim=range(x_grid), cex=.5)
lines(x_grid, y_pred$fit, lwd=2, col='blue')
matlines(x_grid, se.bands, lwd=1, col='blue', lty=3)
par(mfrow=c(2,2))
plot(m_poly)
dev.off()

# Alternative: we can obtain non-orthogonal polynomial with:
# m_list = lapply(1:10, function(degree){lm(y ~ poly(x, degree=degree, raw=T))})

### ---------------------------------------------------------------------------------------
### Polynomial model for a probability (dummy_var = I_{y>val})
### ---------------------------------------------------------------------------------------
val = 250
m_list = lapply(1:5, function(degree){glm(I(y>val) ~ poly(x, degree=degree), family='binomial')})
do.call(anova, m_list)
m_poly = m_list[[4]]
summary(m_poly)
x_grid   = seq(range(x)[1], range(x)[2], length.out=100)
y_pred_0 = predict(m_poly, list(x=x_grid), se=T)
y_pred   = exp(y_pred_0$fit)/(1+exp(y_pred_0$fit))
se.bands = cbind(y_pred_0$fit+2*y_pred_0$se.fit, y_pred_0$fit-2*y_pred_0$se.fit)
se.bands = exp(se.bands)/(1+exp(se.bands))
plot(x, I(y>val), xlim=range(x_grid), cex=.5)
lines(x_grid, y_pred, lwd=2, col='blue')
matlines(x_grid, se.bands, lwd=1, col='blue', lty=3)

### ---------------------------------------------------------------------------------------
### Local Regression: CUT
### ---------------------------------------------------------------------------------------
n_cut = 4
table(cut(x,n_cut))
m_cut    = lm(y ~ cut(x,n_cut))
x_grid   = seq(range(x)[1], range(x)[2], length.out=100)
y_pred   = predict(m_cut, list(x=x_grid), se=T)
se.bands = cbind(y_pred$fit+2*y_pred$se.fit, y_pred$fit-2*y_pred$se.fit)
plot(x, y, xlim=range(x_grid), cex=.5)
lines(x_grid, y_pred$fit, lwd=2, col='blue')
matlines(x_grid, se.bands, lwd=1, col='blue', lty=3)

### ---------------------------------------------------------------------------------------
### Local Regression: CUT - probability (dummy_var = I_{y>val})
### ---------------------------------------------------------------------------------------
n_cut = 4
val   = 250
table(cut(x,n_cut))
m_logit = glm(I(y>val) ~ cut(x,n_cut), family='binomial')
x_grid   = seq(range(x)[1], range(x)[2], length.out=100)
y_pred_0 = predict(m_logit, list(x=x_grid), se=T)
y_pred   = exp(y_pred_0$fit)/(1+exp(y_pred_0$fit))
se.bands = cbind(y_pred_0$fit+2*y_pred_0$se.fit, y_pred_0$fit-2*y_pred_0$se.fit)
se.bands = exp(se.bands)/(1+exp(se.bands))
plot(x, I(y>val), xlim=range(x_grid), cex=.5)
lines(x_grid, y_pred, lwd=2, col='blue')
matlines(x_grid, se.bands, lwd=1, col='blue', lty=3)

### ---------------------------------------------------------------------------------------
### Local Regression: CUT in estabilished points
### ---------------------------------------------------------------------------------------
br       = c(min(x), mean(x), max(x))
# We can produce un-even bins (we may want smaller bins where we have a lot of data)
#br       = c(seq(min(x), mean(x), length.out=10), seq(mean(x)+1, max(x), length.out=20))
m_cut    = lm(y ~ cut(x, breaks=br))
x_grid   = seq(range(x)[1], range(x)[2], length.out=100)
y_pred   = predict(m_cut, list(x=x_grid), se=T)
se.bands = cbind(y_pred$fit+2*y_pred$se.fit, y_pred$fit-2*y_pred$se.fit)
plot(x, y, xlim=range(x_grid), cex=.5)
```

NONPARAMETRIC Hp.

- $Y = f(X) + \varepsilon$
- $\mathbb{E}[\varepsilon] = 0$
- the samples are iid $(y_1, x_1), \dots, (y_n, x_n)$ and they have the same joint distribution
- f has no parametric assumptions
- errors ε are $\perp\!\!\!\perp$ from the input ($\Rightarrow$ errors are iid)

$y_i = \beta_0 + \sum_{j=1}^{d} \beta_j x_i^j + \varepsilon_i$

disadvantage: all the point influence all

$y_i = \beta_0 + \beta_1 c_1(x) + \dots + \beta_K c_K(x) + \varepsilon_i$

β_j is the mean response for $x \in [c_j, c_{j+1}]$

```
lines(x_grid, y_pred$fit, lwd=2, col='blue')
matlines(x_grid, se.bands, lwd=1, col='blue', lty=3)

### ----------------------------------------------------------------------------------------
### Local Regression: LOCAL AVERAGING
### ----------------------------------------------------------------------------------------
library(np)
band_w   = 10     # higher -> more points considered to do the average
m_loc    = npreg(x, y, ckertype='uniform', bws=band_w)
x_grid   = seq(range(x)[1], range(x)[2], length.out=100)
y_pred   = predict(m_loc, list(x=x_grid), se=T)
se.bands = cbind(y_pred$fit+2*y_pred$se.fit, y_pred$fit-2*y_pred$se.fit)
plot(x, y, xlim=range(x_grid), cex=.5)
lines(x_grid, y_pred$fit, lwd=2, col='blue')
matlines(x_grid, se.bands, lwd=1, col='blue', lty=3)

### ----------------------------------------------------------------------------------------
### Local Regression: KERNEL AVERAGING
### ----------------------------------------------------------------------------------------
library(np)
band_w   = 10     # higher -> more points considered to do the average
m_ker    = npreg(x, y, ckertype='gauss', bws=band_w)
x_grid   = seq(range(x)[1], range(x)[2], length.out=100)
y_pred   = predict(m_ker, list(x=x_grid), se=T)
se.bands = cbind(y_pred$fit+2*y_pred$se.fit, y_pred$fit-2*y_pred$se.fit)
plot(x, y, xlim=range(x_grid), cex=.5)
lines(x_grid, y_pred$fit, lwd=2, col='blue')
matlines(x_grid, se.bands, lwd=1, col='blue', lty=3)

### ----------------------------------------------------------------------------------------
### Local Regression: ADAPTIVE KERNEL
### ----------------------------------------------------------------------------------------
library(np)
band_w   = 10     # higher -> more points considered to do the average
m_ker    = npreg(x, y, ckertype='gauss', bws=band_w, bwscaling=T)
x_grid   = seq(range(x)[1], range(x)[2], length.out=100)
y_pred   = predict(m_ker, list(x=x_grid), se=T)
se.bands = cbind(y_pred$fit+2*y_pred$se.fit, y_pred$fit-2*y_pred$se.fit)
plot(x, y, xlim=range(x_grid), cex=.5)
lines(x_grid, y_pred$fit, lwd=2, col='blue')
matlines(x_grid, se.bands, lwd=1, col='blue', lty=3)

### ----------------------------------------------------------------------------------------
### Piecewise linear
### ----------------------------------------------------------------------------------------
cutoff      = mean(x)
x_cut       = x>cutoff
x_cut_model = (x-cutoff)*x_cut
model_cut   = lm(y ~ x + x_cut_model)
x_grid   = seq(range(x)[1], range(x)[2], length.out=100)
y_pred   = predict(model_cut, list(x=x_grid, x_cut_model=(x_grid-cutoff)*(x_grid>cutoff)), se=T)
se.bands = cbind(y_pred$fit+2*y_pred$se.fit, y_pred$fit-2*y_pred$se.fit)
plot(x, y, xlim=range(x_grid), cex=.5)
lines(x_grid, y_pred$fit, lwd=2, col='blue')
matlines(x_grid, se.bands, lwd=1, col='blue', lty=3)
summary(model_cut) # to see the change of slope

### ----------------------------------------------------------------------------------------
### Piecewise linear allowing discontinuities
### ----------------------------------------------------------------------------------------
cutoff      = mean(x)
x_cut       = x>cutoff
x_cut_model = (x-cutoff)*x_cut
model_cut   = lm(y ~ x + x_cut_model + I(x>cutoff))
x_grid   = seq(range(x)[1], range(x)[2], length.out=100)
y_pred   = predict(model_cut, list(x=x_grid, x_cut_model=(x_grid-cutoff)*(x_grid>cutoff)), se=T)
se.bands = cbind(y_pred$fit+2*y_pred$se.fit, y_pred$fit-2*y_pred$se.fit)
plot(x, y, xlim=range(x_grid), cex=.5)
lines(x_grid, y_pred$fit, lwd=2, col='blue')
matlines(x_grid, se.bands, lwd=1, col='blue', lty=3)
summary(model_cut) # to see the change of slope

### ----------------------------------------------------------------------------------------
### Piecewise allowing discontinuities and different functional forms
### ----------------------------------------------------------------------------------------
cutoff      = mean(x)
x_cut       = x>cutoff
x_cut_model = (x-cutoff)*x_cut
model_cut   = lm(y ~ poly(x, degree=3) + poly(x_cut_model, degree=3) + I(x>cutoff))
x_grid   = seq(range(x)[1], range(x)[2], length.out=100)
y_pred   = predict(model_cut, list(x=x_grid, x_cut_model=(x_grid-cutoff)*(x_grid>cutoff)), se=T)
se.bands = cbind(y_pred$fit+2*y_pred$se.fit, y_pred$fit-2*y_pred$se.fit)
plot(x, y, xlim=range(x_grid), cex=.5)
lines(x_grid, y_pred$fit, lwd=2, col='blue')
matlines(x_grid, se.bands, lwd=1, col='blue', lty=3)
summary(model_cut) # to see the change of slope
```

we don't have to choose the same degree of polynomial in every portion

piecewise polynomial functions which are constrained to join smoothly at knots

```
################################################################################
###                              SPLINES                                     ###
################################################################################
library(splines)
load('nlr_data.rda')
data = data.frame("age"=Wage$age,"wage"=Wage$wage)    # data = [X, Y]
x    = data[,1]
y    = data[,2]

### ------------------------------------------------------------------------------
### General Splines
### ------------------------------------------------------------------------------
degree   = 15    # if = 3 -> cubic splines
model_sp = lm(y ~ bs(x, degree=degree))
x_grid   = seq(range(x)[1], range(x)[2], length.out=100)
y_pred   = predict(model_sp, list(x=x_grid), se=T)
se.bands = cbind(y_pred$fit+2*y_pred$se.fit, y_pred$fit-2*y_pred$se.fit)
plot(x, y, xlim=range(x_grid), cex=.5)
lines(x_grid, y_pred$fit, lwd=2, col='blue')
matlines(x_grid, se.bands, lwd=1, col='blue', lty=3)

#-----
# We can ad some break points (and plot them)
# br        = c(seq(min(x), mean(x), length=2), seq(mean(x)+1, max(x), 1))
# br        = c(seq(min(x), max(x), 100)
# model_cut = lm(y ~ bs(x, knots=br, degree=degree))
# ..
# knot_pred = predict(model_cut, list(x=br))
# points(br, knot_pred)
#-----

#-----
# We can either add the number of knots (where we break) or we can specify the degrees
# of freddom knowing that: degree_of_freedom = #knots + degree. If we know where to put
# the knots then we can define the break points, otherwise we specify the number of points.
# If we want to have n knots and d degree (df = n+d) we write:
# n_knot = ..
# degree = ..
# model_cut = lm(y ~ bs(x, degree=degree, df=n_knot+degree))
# ..
# knots = attr(bs(x, degree=degree, df=n+d), 'knots')
# knots
# knot_pred = predict(model_cut, data.frame(x=knots))
# points(br, knots_pred)
#-----

#-----
# We can specify the quantiles of the data on which we want a break instead of the knots:
# br = c(quantile(x, probs=c(0.2, 0.4, 0.6, 0.8)))
#-----

#-----
# We can specify the quantiles of the data and in addition another value
# val = ..
# br  = c(quantile(x, probs=c(0.2, 0.4, 0.6, 0.8)), val)
#-----

### ------------------------------------------------------------------------------
### Natural Splines
### ------------------------------------------------------------------------------
knots     = quantile(x, probs=c(0.1, 0.5, 0.9))
bd_knots  = quantile(x, probs=c(0.05, 0.95))
model_cut = lm(y ~ ns(x, knots=knots, Boundary.knots=bd_knots))
x_grid    = seq(range(x)[1], range(x)[2], length.out=100)
y_pred    = predict(model_cut, list(x=x_grid), se=T)
se.bands  = cbind(y_pred$fit+2*y_pred$se.fit, y_pred$fit-2*y_pred$se.fit)
plot(x, y, xlim=range(x_grid), cex=.5)
lines(x_grid, y_pred$fit, lwd=2, col='blue')
matlines(x_grid, se.bands, lwd=1, col='blue', lty=3)
knot_pred = predict(model_cut, list(x=knots))
points(knots, knot_pred, pch=19, col='red', lwd=9)
bd_knots_pred = predict(model_cut, list(x=bd_knots))
points(bd_knots, bd_knots_pred, pch=19, col='green', lwd=9)

# Note: knots and bd_knots can be values and not quantiles

### ------------------------------------------------------------------------------
### Smoothing Splines
### ------------------------------------------------------------------------------
# We can specify the degree of freedom (df) or the penalty (lambda)
fit = smooth.spline(x, y, df=61)
fit = smooth.spline(x, y, lambda=1e-10)   # higher the lambda, smoother the curve
plot(x, y, cex=.5)
lines(fit, col='blue', lwd=2)

# We can let lambda/df be choosen in an automatic way (minimizing the CV/GCV error)
fit = smooth.spline(x, y, cv=T)  # CV
fit = smooth.spline(x, y, cv=F)  # GCV
plot(x, y, cex=.5)
lines(fit, col='blue', lwd=2)

fit$lambda
fit$df

# Prediction
val = c(50.5, 60.5, 90.5)
predict(fit, val)
```

high variance at boundaries

CUBIC SPLINES:
3rd order polynomial in each bin, with 1st and 2nd derivatives continuous at knots

order = k
degree = k-1

Solution

] Extraction of the knots

NATURAL SPLINES:
add knots at boundary and impose linear fit beyond

← natural cubic splines:

linear cubic linear

penalized regression over the natural splines basis (the higher the λ (smoother param) the smoother the result)

cubic smoothing splines (natural cubic splines)

```
### ---------------------------------------------------------------------------
### Monotone case
### ---------------------------------------------------------------------------
data = subset(Orange, Orange$Tree==3)
data = data[,c(2,3)]
x    = data[,1]
y    = data[,2]

library(fda)
basis    = create.bspline.basis(range(x), breaks=x, norder=4)
cvec0    = matrix(0, basis$nbasis, 1)
Wfd0     = fd(cvec0, basis)
start_fd = fdPar(Wfd0, Lfdobj=2, lambda=1e4)
res      = smooth.monotone(x, y, start_fd)
Wfd      = res$Wfdobj
beta     = res$beta
x_grid   = seq(range(x)[1], range(x)[2], length.out=100)
y_pred   = beta[1] + beta[2]*eval.monfd(x_grid, Wfd, 0)
plot(x, y, cex=.5)
lines(x_grid, y_pred)
```

```
##############################################################################################
###                                          GAMs                                          ###
##############################################################################################
seed  = 100
B     = 1000
alpha = 0.1

### ------------------------------------------------------------------------------------------
### GAMs given orders                                                  [not the best library]
### ------------------------------------------------------------------------------------------
library(gam)
# data = [x1, .., xp, y]
load('nlr_data.rda')
data  = data.frame("year"=Wage$year, "age"=Wage$age, "wage"=Wage$wage)
x1    = data[,1]
x2    = data[,2]
y     = data[,3]
gam_m = gam(y ~ s(x1,1) + s(x2,1))
# we can modify the degree of smoothness for each term
summary(gam_m)
par(mfrow=c(1,2))
plot(gam_m)

x1_grid     = seq(range(x1)[1], range(x1)[2], length.out=100)
x2_grid     = seq(range(x2)[1], range(x2)[2], length.out=100)
grid        = expand.grid(x1_grid, x2_grid)
names(grid) = c('x1', 'x2')
pred        = predict(gam_m, newdata=grid)

library(plot3D)
persp3D(z=pred, theta=120)

library(rgl)
persp3d(x1_grid, x2_grid, pred, col='yellow')
points3d(x1, x2, y, col='black', size=5)

### ------------------------------------------------------------------------------------------
### Cubic Regression Splines
### ------------------------------------------------------------------------------------------
library(mgcv)
# data = [x1, .., xp, y]
load('nlr_data.rda')
data = as.data.frame(cbind("education"=Prestige$education,
                           "income"   =Prestige$income,
                           "prestige" =Prestige$prestige))
x1    = data[,1]
x2    = data[,2]
y     = data[,3]
gam_m = gam(y ~ s(x1, bs='cr') + s(x2, bs='cr'))
#gam_m = gam(y ~ s(x1, bs='cr') + s(x2, bs='cr',k=7))  # we can also decide #knots (k)
summary(gam_m)
par(mfrow=c(1,2))
plot(gam_m)
hist(gam_m$residuals)
qqnorm(gam_m$residuals)
shapiro.test(gam_m$residuals)

library(rgl)
x1_grid     = seq(range(x1)[1], range(x1)[2], length.out=100)
x2_grid     = seq(range(x2)[1], range(x2)[2], length.out=100)
grid        = expand.grid(x1_grid, x2_grid)
names(grid) = c('x1', 'x2')
pred        = predict(gam_m, newdata=grid)
persp3d(x1_grid, x2_grid, pred, col='yellow')
points3d(x1, x2, y, col='black', size=5)

### ------------------------------------------------------------------------------------------
### Cubic Regression Splines with interaction
### ------------------------------------------------------------------------------------------
### |---------------|
### | Cubic Splines |
### |---------------|
inter = x1*x2
gam_m = gam(y ~ s(x1, bs='cr') + s(x2, bs='cr') + s(I(inter), bs='cr'))
summary(gam_m)
par(mfrow=c(1,2))
plot(gam_m)
hist(gam_m$residuals)
qqnorm(gam_m$residuals)
shapiro.test(gam_m$residuals)

library(rgl)
x1_grid     = seq(range(x1)[1], range(x1)[2], length.out=100)
x2_grid     = seq(range(x2)[1], range(x2)[2], length.out=100)
grid        = expand.grid(x1_grid, x2_grid)
names(grid) = c('x1', 'x2')
pred        = predict(gam_m, newdata=data.frame(grid, inter=grid$x1*grid$x1))
persp3d(x1_grid, x2_grid, pred, col='yellow')
points3d(x1, x2, y, col='black', size=5)

### |--------------------|
### | Thin Plate Splines |
### |--------------------|
gam_m = gam(y ~ s(x1, bs='cr') + s(x2, bs='cr') + s(x1, x2, bs='tp'))
summary(gam_m)
par(mfrow=c(1,2))
plot(gam_m)
hist(gam_m$residuals)
qqnorm(gam_m$residuals)
shapiro.test(gam_m$residuals)
```

Instead of considering:
$f(x) = f(x_1, .., x_p)$
we consider:
$f(x) = f(x_1) + .. + f(x_p)$
we're dealing with one covariate at time: we decide the best fitting separately (≠ from considering p different regressions because, even if fixed, we're considering the other covariates)

```r
library(rgl)
x1_grid     = seq(range(x1)[1], range(x1)[2], length.out=100)
x2_grid     = seq(range(x2)[1], range(x2)[2], length.out=100)
grid        = expand.grid(x1_grid, x2_grid)
names(grid) = c('x1', 'x2')
pred        = predict(gam_m, newdata=data.frame(grid))
persp3d(x1_grid, x2_grid, pred, col='yellow')
points3d(x1, x2, y, col='black', size=5)

### ------------------------------------------------------------------------------------------
### Normal residuals - CI & Prediction
### ------------------------------------------------------------------------------------------
# Prediction
newdata = data.frame(x1=15, x2=15000)
pred    = predict(gam_m, newdata=newdata, type='response', se.fit=T)

# Confidence Intervals
lwr = pred$fit-pred$se.fit*qt(1-(alpha/2), nrow(data))
lvl = pred$fit
upr = pred$fit+pred$se.fit*qt(1-(alpha/2), nrow(data))
cbind(lwr, lvl, upr)

### ------------------------------------------------------------------------------------------
### *NOT* normal residuals - Bootstrap CI & Prediction
### ------------------------------------------------------------------------------------------
set.seed(seed)
newdata    = data.frame(x1=15, x2=15000)
fitted_obs = gam_m$fitted.values
res_obs    = gam_m$residuals
pred_obs   = predict(gam_m, newdata=newdata)

pred_boot = numeric(B)
for(k in 1:B){
  y_boot       = fitted_obs + sample(res_obs, replace=T)
  gam_m_boot   = gam(y_boot ~ s(x1, bs='cr') + s(x2, bs='cr') + s(x1, x2, bs='tp'))
  pred_boot[k] = predict(gam_m_boot, newdata=newdata)
}

right_quantile = quantile(pred_boot, 1-alpha/2)
left_quantile  = quantile(pred_boot, alpha/2)
CI_boot = c(pred_obs-(right_quantile-pred_obs), pred_obs, pred_obs-(left_quantile-pred_obs))
CI_boot

### ------------------------------------------------------------------------------------------
### *NOT* normal residuals - Permutation test for coefficient
### ------------------------------------------------------------------------------------------
# Test for the interaction
T0 = abs(summary(gam_m)$s.table[3,3])
gam_H0 = gam(y ~ s(x1, bs='cr') + s(x2, bs='cr'))
res_H0 = gam_H0$residuals
n      = nrow(data)

set.seed(seed)
T_stat = numeric(B)
for(k in 1:B){
  perm        = sample(nrow(data))
  res_H0_perm = res_H0[perm]
  y_perm      = gam_H0$fitted.values + res_H0_perm
  gam_perm    = gam(y_perm ~ s(x1, bs='cr') + s(x2, bs='cr') + s(x1, x2, bs='tp'))
  T_stat[k]   = abs(summary(gam_perm)$s.table[3,3])
}

par(mfrow=c(1,2))
hist(T_stat)
abline(v=T0, lty=2, col='red')
plot(ecdf(T_stat))
abline(v=T0, lty=2, col='red')

p_value = sum(T_stat>=T0)/B
p_value

### ------------------------------------------------------------------------------------------
### Cubic Splines for a probability model (dummy_var = I_{y>val})
### ------------------------------------------------------------------------------------------
val       = 20
gam_logit = gam(I(y>val) ~ s(x1,bs='cr') + s(x2,bs='cr'), family='binomial')

library(rgl)
x1_grid     = seq(range(x1)[1], range(x1)[2], length.out=100)
x2_grid     = seq(range(x2)[1], range(x2)[2], length.out=100)
grid        = expand.grid(x1_grid, x2_grid)
names(grid) = c('x1', 'x2')
preds       = predict(gam_logit, newdata=grid)
pfit        = exp(preds)/(1+exp(preds))
persp3d(x1_grid, x2_grid, pfit, col='yellow')
points3d(x1, x2, I(y>val), col='black', size=5)
```

Hp. $X_1, \dots, X_n$ iid F
moreover:
$\{X_1, \dots, X_n\} \perp\!\!\!\perp X_{n+1}$

```
################################################################################################
###                                FULL CONFORMAL PREDICTION                                 ###
################################################################################################
seed  = 100
B     = 1000
alpha = 0.1

### ------------------------------------------------------------------------------------------
### Univariate                                  X1,..,Xn ⟶ PI(Xn+1)?
### ------------------------------------------------------------------------------------------
parz    = read.table('parziali.txt')
x       = parz$PI
x_grid  = seq(min(x)-0.25*diff(range(x)), max(x)+0.25*diff(range(x)), length.out=100)
p_value = numeric(length(x_grid))

### |-------------------|
### | Discrepancy-based |
### |-------------------|
NC = function(z_aug, i){
  abs(z_aug[i] - mean(z_aug[-i]))
  #abs(z_aug[i] - median(z.aug[-i]))  # more robust
}

for(k in 1:length(x_grid)){
  x_aug  = c(x, x_grid[k])
  scores = numeric(length(x_aug))
  for(i in 1:length(scores)){
    scores[i] = NC(x_aug, i)
  }
  p_value[k] = sum(scores>=scores[length(x_aug)])/(length(x_aug))
}

# Plot of the p-values
plot(x_grid, p_value, type='l', ylim=c(0,1))
abline(h=c(0,1))
abline(h=alpha, col='red')
points(x, numeric(length(x)), pch=3)

# Prediction Interval
PI_grid = x_grid[which(p_value>=alpha)]
PI      = c(min(PI_grid), max(PI_grid))
PI
abline(v=PI, col='red')
points(PI_grid, numeric(length(PI_grid)), pch=16, col='red')

### |-----|
### | KNN |
### |-----|
K  = 5
NC = function(z_aug, i){
  distances2 = (as.matrix(dist(z_aug))[i,-i])^2
  mean(sort(distances2)[1:K])   # average linkage
  #min(sort(distances2)[1:K])   # single linkage
  #max(sort(distances2)[1:K])   # complete linkage
}

for(k in 1:length(x_grid)){
  x_aug  = c(x, x_grid[k])
  scores = numeric(length(x_aug))
  for(i in 1:length(scores)){
    scores[i] = NC(x_aug, i)
  }
  p_value[k] = sum(scores>=scores[length(x_aug)])/(length(x_aug))
}

# Plot of the p-values
plot(x_grid, p_value, type='l', ylim=c(0,1))
abline(h=c(0,1))
abline(h=alpha, col='red')
points(x, numeric(length(x)), pch=3)

# Prediction Interval
PI_grid = x_grid[which(p_value>=alpha)]
PI      = c(min(PI_grid), max(PI_grid))
PI
abline(v=PI, col='red')
points(PI_grid, numeric(length(PI_grid)), pch=16, col='red')

### ------------------------------------------------------------------------------------------
### Multivariate
### ------------------------------------------------------------------------------------------
data = read.table('parziali.txt')
x    = data[,1]
y    = data[,2]
plot(x,y,asp=1)

x_grid  = seq(min(x)-0.5*diff(range(x)), max(x)+0.5*diff(range(x)), length.out=10)
y_grid  = seq(min(y)-0.5*diff(range(y)), max(y)+0.5*diff(range(y)), length.out=10)
p_value = matrix(nrow=length(x_grid), ncol=length(y_grid))

### |--------------------------|
### | Predict a new (x,y) point |
### |--------------------------|
NC = function(z_aug, i){
  sum((z_aug[i,]-colMeans(z_aug[-i,]))^2)   # Euclidean distance
  #as.numeric(as.matrix(z_aug[i,]-colMeans(z_aug[-i,]))%*% solve(cov(z_aug[-i,])) %*%
  #             as.matrix(t(z_aug[i,]-colMeans(z_aug[-i,]))))   # Mahalanobis
}
```

given the dataset, we generate a random set that will capture the new prediction with prob. higher than $1-\alpha$

non-conformity score: measures the non-conformity of i to the set `z_aug[-i]`

```
for(k in 1:length(x_grid)){
  for(h in 1:length(y_grid)){
    data_aug = rbind(data, c(x_grid[k], y_grid[h]))
    scores   = numeric(dim(data_aug)[1])
    for(i in 1:length(scores)){
      scores[i] = NC(data_aug, i)
    }
    p_value[k,h] = sum(scores>=scores[dim(data_aug)[1]])/(dim(data_aug)[1])
  }
}

# Plot of the p-value
image(x_grid, y_grid, p_value, zlim=c(0,1), asp=1)
points(data, pch=16)
contour(x_grid, y_grid, p_value, levels=alpha, add=T)

### |------------|
### | Regression |
### |------------|
NC = function(z_aug, i){
  abs(z_aug[i,2]-sum(coefficients(lm(z_aug[-i,2] ~ z_aug[-i,1]))*c(1, z_aug[i,1])))
  # we can use anything, not only linear regression
}

for(k in 1:length(x_grid)){
  for(h in 1:length(y_grid)){
    data_aug = rbind(data, c(x_grid[k], y_grid[h]))
    scores   = numeric(dim(data_aug)[1])
    for(i in 1:length(scores)){
      scores[i] = NC(data_aug, i)
    }
    p_value[k,h] = sum(scores>=scores[dim(data_aug)[1]])/(dim(data_aug)[1])
  }
}

# Plot of the p-value
image(x_grid, y_grid, p_value, zlim=c(0,1), asp=1)
points(data, pch=16)
contour(x_grid, y_grid, p_value, levels=alpha, add=T)

### |-----|
### | KNN |
### |-----|
K  = 5
NC = function(z_aug, i){
  distances2 = (as.matrix(dist(z_aug))[i,-i])^2
  mean(sort(distances2)[1:K]) # average linkage
  #min(sort(distances2)[1:K]) # single linkage
  #max(sort(distances2)[1:K]) # complete linkage
}

for(k in 1:length(x_grid)){
  for(h in 1:length(y_grid)){
    data_aug = rbind(data, c(x_grid[k], y_grid[h]))
    scores   = numeric(dim(data_aug)[1])
    for(i in 1:length(scores)){
      scores[i] = NC(data_aug, i)
    }
    p_value[k,h] = sum(scores>=scores[dim(data_aug)[1]])/(dim(data_aug)[1])
  }
}

# Plot of the p-value
image(x_grid, y_grid, p_value, zlim=c(0,1), asp=1)
points(data, pch=16)
contour(x_grid, y_grid, p_value, levels=alpha, add=T)
```

$(X_1,Y_1),\dots,(X_n,Y_n),X_{n+1} \longrightarrow PI(Y_{n+1})?$

The non-conformity measure is between an observation i of the augmented calibration data and the prediction of the training data

```
#############################################################################################
###                          SPLIT CONFORMAL PREDICTION                                   ###
#############################################################################################
seed  = 100
B     = 1000
alpha = 0.1

### -----------------------------------------------------------------------------------------
### Univariate
### -----------------------------------------------------------------------------------------
parz       = read.table('parziali.txt')
data       = parz$PI
n          = length(data)
train_prop = 0.5

### |------------------|
### | Discrepancy-based |
### |------------------|
set.seed(seed)
train_id   = sample(1:n, ceiling(n*train_prop), replace=F)
train_set  = data[train_id]
x          = data[-train_id]
x_grid     = seq(min(x)-0.5*diff(range(x)), max(x)+0.5*diff(range(x)), length.out=100)
p_value    = numeric(length(x_grid))
train_mean = mean(train_set)   # or median

NC = function(z_aug, i){
  abs(z_aug[i] - train_mean)
}

for(k in 1:length(x_grid)){
  x_aug  = c(x, x_grid[k])
  scores = numeric(length(x_aug))
  for(i in 1:length(scores)){
    scores[i] = NC(x_aug, i)
  }
  p_value[k] = sum(scores>=scores[length(x_aug)])/(length(x_aug))
}

# Plot of the p-values
plot(x_grid, p_value, type='l', ylim=c(0,1))
abline(h=c(0,1))
abline(h=alpha, col='red')
points(x, numeric(length(x)), pch=3)

# Prediction Interval
PI_grid = x_grid[which(p_value>=alpha)]
PI      = c(min(PI_grid), max(PI_grid))
PI
abline(v=PI, col='red')
points(PI_grid, numeric(length(PI_grid)), pch=16, col='red')

### -----------------------------------------------------------------------------------------
### Multivariate
### -----------------------------------------------------------------------------------------
data       = read.table('parziali.txt')
n          = dim(data)[1]
train_prop = 0.5

set.seed(seed)
train_id   = sample(1:n, ceiling(n*train_prop), replace=F)
train_set  = data[train_id,]
calib_set  = data[-train_id,]
x          = calib_set[,1]
y          = calib_set[,2]
x_grid     = seq(min(x)-0.5*diff(range(x)), max(x)+0.5*diff(range(x)), length.out=20)
y_grid     = seq(min(y)-0.5*diff(range(y)), max(y)+0.5*diff(range(y)), length.out=20)
p_value    = matrix(nrow=length(x_grid), ncol=length(y_grid))

### |--------------------------|
### | Predict a new (x,y) point |
### |--------------------------|
train_mean = colMeans(train_set)

NC = function(z_aug, i){
  sum((z_aug[i,]-train_mean)^2) # Euclidean distance
}

for(k in 1:length(x_grid)){
  for(h in 1:length(y_grid)){
    data_aug = rbind(calib_set, c(x_grid[k], y_grid[h]))
    scores   = numeric(dim(data_aug)[1])
    for(i in 1:length(scores)){
      scores[i] = NC(data_aug, i)
    }
    p_value[k,h] = sum(scores>=scores[dim(data_aug)[1]])/(dim(data_aug)[1])
  }
}

# Plot of the p-value
image(x_grid, y_grid, p_value, zlim=c(0,1), asp=1)
points(calib_set, pch=16)
contour(x_grid, y_grid, p_value, levels=alpha, add=T)
points(train_mean[1], train_mean[2], pch=4, cex=4)
```

```
### |------------|
### | Regression |
### |------------|
# Model on training (can be any kind of model, check how to extract the prediction!)
fm = smooth.spline(train_set[,1], train_set[,2], lambda = lambda)

NC = function(z_aug, i){
  abs(z_aug[i,2] - predict(fm, x = z_aug[i,1])$y)
}

for(k in 1:length(x_grid)){
  for(h in 1:length(y_grid)){
    data_aug = rbind(calib_set, c(x_grid[k], y_grid[h]))
    scores   = numeric(dim(data_aug)[1])
    for(i in 1:length(scores)){
      scores[i] = NC(data_aug, i)
    }
    p_value[k,h] = sum(scores>=scores[dim(data_aug)[1]])/(dim(data_aug)[1])
  }
}

# Plot of the p-value
image(x_grid, y_grid, p_value, zlim=c(0,1))
points(data, pch=16)
contour(x_grid, y_grid, p_value, levels=alpha, add=T)
```

```
##############################################################################################
###                                CONFORMAL PREDICTION INTERVALS                          ###
###                                  library(conformalInference)                           ###
##############################################################################################
library(conformalInference)
seed  = 100
B     = 1000
alpha = 0.1

# The following are full conformal. For split: conformal.pred.split()

### ------------------------------------------------------------------------------------------
### Univariate
### ------------------------------------------------------------------------------------------
load('uphillperformance.rda')
data   = df_2
y      = data$u.speed
x      = data$t.days
x_grid = seq(range(x)[1], range(x)[2], length.out =50)

### |--------|
### | Linear |
### |--------|
fit   = lm(y ~ x)
preds = predict(fit, list(x=x_grid), se=T)
plot(x, y, xlim=range(x_grid), cex=.5, pch=19)
lines(x_grid, preds$fit, lwd=2, col='blue')

# Conformal Interval evaluated in new_val
new_val    = mean(x)
lm_train   = lm.funs(intercept = T)$train.fun
lm_predict = lm.funs(intercept = T)$predict.fun
c_preds    = conformal.pred(x, y, new_val, alpha=alpha, train.fun=lm_train, predict.fun=lm_predict)
cbind(c_preds$lo, c_preds$pred, c_preds$up)

# Conformal Intervals plotted
c_pr_plot = conformal.pred(x, y, x_grid, alpha=alpha, train.fun=lm_train, predict.fun=lm_predict)
lines(x_grid, c_pr_plot$pred, lwd=3, col='red', lty=3)
matlines(x_grid, cbind(c_pr_plot$up, c_pr_plot$lo), lwd=3, col='red', lty=3)

### |------------|
### | Polynomial |
### |------------|
fit   = lm(y ~ poly(x, degree=2))
preds = predict(fit, list(x=x_grid), se=T)
plot(x, y, xlim=range(x_grid), cex=.5, pch=19)
lines(x_grid, preds$fit, lwd=2, col='blue')

# Conformal Interval evaluated in new_val
design_mx  = matrix(poly(x, degree=2), ncol=2)
point_0    = mean(x)
new_val    = matrix(poly(point_0, degree=2, coefs=attr(poly(x, degree=2), 'coefs')), ncol=2)
lm_train   = lm.funs(intercept = T)$train.fun
lm_predict = lm.funs(intercept = T)$predict.fun
c_preds    = conformal.pred(design_mx, y, new_val, alpha=alpha, train.fun=lm_train, predict.fun=lm_predict)
cbind(c_preds$lo, c_preds$pred, c_preds$up)

# Conformal Intervals plotted
x_pred    = matrix(poly(x_grid, degree=2, coefs=attr(poly(x, degree=2), 'coefs')), ncol=2)
c_pr_plot = conformal.pred(design_mx, y, x_pred, alpha=alpha, train.fun=lm_train, predict.fun=lm_predict)
lines(x_grid, c_pr_plot$pred, lwd=3, col='red', lty=3)
matlines(x_grid, cbind(c_pr_plot$up, c_pr_plot$lo), lwd=3, col='red', lty=3)

### |---------|
### | Splines |
### |---------|
library(splines)
br    = c(quantile(x, probs=c(0.25, 0.5, 0.75)))
fit   = lm(y ~ bs(x, degree=3, knots=br))
preds = predict(fit, list(x=x_grid), se=T)
plot(x, y, xlim=range(x_grid), cex=.5, pch=19)
lines(x_grid, preds$fit, lwd=2, col='blue')

# Conformal Interval evaluated in new_val
design_mx  = bs(x, degree=3, knots=br)
point_0    = mean(x)
new_val    = matrix(bs(point_0, degree=3, knots=br), nrow=1)
lm_train   = lm.funs(intercept = T)$train.fun
lm_predict = lm.funs(intercept = T)$predict.fun
c_preds    = conformal.pred(design_mx, y, new_val, alpha=alpha, train.fun=lm_train, predict.fun=lm_predict)
cbind(c_preds$lo, c_preds$pred, c_preds$up)

# Conformal Intervals plotted
x_pred    = matrix(bs(x_grid, degree=3, knots=br), nrow=length(x_grid))
c_pr_plot = conformal.pred(design_mx, y, x_pred, alpha=alpha, train.fun=lm_train, predict.fun=lm_predict)
lines(x_grid, c_pr_plot$pred, lwd=3, col='red', lty=3)
matlines(x_grid, cbind(c_pr_plot$up, c_pr_plot$lo), lwd=3, col='red', lty=3)

### |------------------|
### | Smoothing Splines |
### |------------------|
fit = smooth.spline(x, y, cv=T)
opt = fit$df
plot(x, y, cex=.5, pch=19)
lines(fit, col='blue', lwd=2)

# Conformal Interval evaluated in new_val
new_val    = mean(x)
train_ss   = function(x,y,out=NULL){smooth.spline(x,y,df=opt)}
predict_ss = function(obj, new_x){predict(obj,new_x)$y}
```

```
c_preds    = conformal.pred(x, y, new_val, alpha=alpha, train.fun=train_ss, predict.fun=predict_ss)
cbind(c_preds$lo, c_preds$pred, c_preds$up)

# Conformal Intervals plotted
c_pr_plot = conformal.pred(x, y, x_grid, alpha=alpha, train.fun=train_ss, predict.fun=predict_ss)
lines(x_grid, c_pr_plot$pred, lwd=3, col='red', lty=3)
matlines(x_grid, cbind(c_pr_plot$up, c_pr_plot$lo), lwd=3, col='red', lty=3)

### ------------------------------------------------------------------------------------
### Multivariate (GAM)
### ------------------------------------------------------------------------------------
library(mgcv)
library(rgl)
load('uphillperformance.rda')
data        = df_2
y           = data$u.speed
x1          = data$t.days
x2          = data$w.ski
x1_grid     = seq(range(x1)[1], range(x1)[2], length.out =50)
x2_grid     = seq(range(x2)[1], range(x2)[2], length.out =50)
grid        = expand.grid(x1_grid, x2_grid)
names(grid) = c('x1', 'x2')

fit  = gam(y ~ s(x1, bs='cr') + s(x2, bs='cr'))
pred = predict(fit, newdata=grid)
persp3d(x1_grid, x2_grid, pred, col='yellow')
points3d(x1, x2, y, col='black', size=5)

train_gam = function(x, y, out=NULL){
  colnames(x) = c('x1', 'x2')
  train_data = data.frame(y,x)
  fit = gam(y ~ s(x1, bs='cr') + s(x2, bs='cr'), data=train_data)
}

predict_gam = function(obj, new_x){
  new_x = data.frame(new_x)
  colnames(new_x) = c('x1', 'x2')
  predict.gam(obj, new_x)
}

c_preds = conformal.pred(cbind(x1,x2), y, c(mean(x1),mean(x2)), alpha=alpha, train.fun=train_gam, predict.fun=predict_gam)
cbind(c_preds$lo, c_preds$pred, c_preds$up)
```

```
#############################################################################################
###                                  SURVIVAL ANALYSIS                                    ###
#############################################################################################
library(survival)
library(survminer)
library(dplyr)
# Surv()    : creates a survival object
# survfit() : fits a survival curve using a formula
# survdiff(): log-rank test for differences in survival between two or more groups

data('lung')
data = lung

# time  : survival time
# status: censoring status 1=censored, 2=dead

### ---------------------------------------------------------------------------------------
### Plot (reduced)
### ---------------------------------------------------------------------------------------
data_red    = head(lung)
data_red$ID = factor(seq(1:nrow(data_red)))
ggplot(data = data_red, aes(x=ID, y=time)) +
  geom_bar(stat='identity', width=0.2) +
  geom_point(aes(color=status, shape=as.factor(status)), size=6) +
  coord_flip()

### ---------------------------------------------------------------------------------------
### Survival Object
### ---------------------------------------------------------------------------------------
# The function Surv(time, event) create a survival object
Surv(data$time, data$status)

### ---------------------------------------------------------------------------------------
### KAPLAN-MEIER
### ---------------------------------------------------------------------------------------
### |-----------------------------------------|
### | Kaplan-Meier estimator for survival curve |
### |-----------------------------------------|
fit = survfit(Surv(time, status) ~ 1, data=data)
summary(fit)

# n           : total number of subjects
# time        : the event time points on the curve (t=t*_j)
# n.risk      : the number of subjects at risk at time t
# n.event     : the number of events that occurred at time t
# n.censor    : the number of censored subjects, who exit the risk set at time t
# surv        : the kaplan-meier estimator for survival S(t)
# std.err     : the standard error for S(t)
# lower, upper: lower and upper confidence limits for the survival curve S(t), respectively
# cumhaz      : the cumulative hazard curve H(t) = - log(S(t))
# std.err     : the standard error for H(t)

### |----------------------|
### | Median Survival time |
### |----------------------|
# Time at which the survival probability (S(t)) is 0.5
median_St = fit$time[fit$surv<=0.5][1]
median_St
# or
surv_median(fit)
# or
print(fit)

### |-------------------------|
### | Kaplan-Meier curve plot |
### |-------------------------|
plot(fit, conf.int=T, xlab='Time', ylab='Survival Probability', main='Kaplan-Meier curve')
# or
ggsurvplot(fit, data = data,
           risk.table = TRUE,
           risk.table.col = 'strata',
           surv.median.line='hv',
           ggtheme = theme_bw(),
           break.time.by = 90,
           title = 'Kaplan-Meier curve')

### |-------------------------------------|
### | Cumulative incidence function /     |
### | Cumulative failure probability (CFP) |
### |-------------------------------------|
# It shows the cumulative probabilities of experiencing the event of interest and
# it is computed as CFP(t) = P(T<t), so can be estimated as 1-S(t)
cumulative_incidence = 1 - fit$surv
ggsurvplot(fit, data = data,
           fun='event',
           risk.table = TRUE,
           risk.table.col = 'strata',
           surv.median.line='hv',
           ggtheme = theme_bw(),
           break.time.by = 90,
           title = 'Cumulative Incidence curve')

### |---------------------------------|
### | Cumulative hazard function (H(t)) |
### |---------------------------------|
# It can be interpreted as the cumulative force of mortality.
H = fit$cumhaz
ggsurvplot(fit, data = data,
           fun='cumhaz',
           risk.table = TRUE,
```

```
          risk.table.col = 'strata',
          ggtheme = theme_bw(),
          break.time.by = 90,
          title = 'Cumulative Hazard curve')

### |-------|
### | Recap |
### |-------|
curves = data.frame('time'          = fit$time,
                    'Survival'      = fit$surv,
                    'Cum_incidence' = 1-fit$surv,
                    'Cum_hazard'    = fit$cumhaz)
head(curves)

### |-----------------------------------|
### | Survival probability at given time |
### |-----------------------------------|
t_0 = 180
summary(fit, times=t_0)

# Survival probability every six months
t_0 = seq(0, 365*3, 182.5)
summary(fit, times=t_0)

### ---------------------------------------------------------------------------------------
### KAPLAN-MEIER by a feature
### ---------------------------------------------------------------------------------------
### |-----------------------------------------|
### | Kaplan-Meier estimator for survival curve |
### |-----------------------------------------|
feature     = data$sex
fit.feature = survfit(Surv(time, status) ~ feature, data=data)
print(fit.feature)
summary(fit.feature)$table
summary(fit.feature)

### |--------------------------------------|
### | Kaplan-Meier curves plot by a feature |
### |--------------------------------------|
plot(fit.feature, conf.int=T, xlab='Time', ylab='Survival Probability',
     col=c('blue', 'red'), main='Kaplan-Meier curve')
legend('topright', legend=c('Feat_1', 'Feat_2'), lty=c(1,1), col=c('blue', 'red'))
# or
ggsurvplot(fit.feature, data = data,
           conf.int = T,
           risk.table = TRUE,
           risk.table.col = "strata",
           surv.median.line = "hv",
           ggtheme = theme_bw(),
           break.time.by = 90,
           legend.labs = c("Feat_1", "Feat_2"),
           legend.title = "Feature",
           palette = c("blue", "red"),
           title = 'Kaplan-Meier curves by a feature')

### |-------------------------------------------------|
### | Cumulative incidence function /                 |
### | Cumulative failure probability (CFP) by a feature |
### |-------------------------------------------------|
ggsurvplot(fit.feature, data = data,
           fun = 'event',
           conf.int = T,
           risk.table = TRUE,
           risk.table.col = "strata",
           surv.median.line = "hv",
           ggtheme = theme_bw(),
           break.time.by = 90,
           legend.labs = c("Feat_1", "Feat_2"),
           legend.title = "Feature",
           palette = c("blue", "red"),
           title = 'Cumulative Incidence curves by a feature')

### |-----------------------------------------------|
### | Cumulative hazard function (H(t)) by a feature |
### |-----------------------------------------------|
ggsurvplot(fit.feature, data = data,
           fun = 'cumhaz',
           conf.int = T,
           risk.table = TRUE,
           risk.table.col = "strata",
           ggtheme = theme_bw(),
           break.time.by = 90,
           legend.labs = c("Feat_1", "Feat_2"),
           legend.title = "Feature",
           palette = c("blue", "red"),
           title = 'Cumulative Hazard curves by a feature')

### |---------------------------|
### | Log-rank test for a feature |
### |---------------------------|
# Comparison of two or more survival curves.
# The null hypothesis is that there is no difference in survival between the two groups.
# The log rank test is a non-parametric test, which makes no assumptions about the survival
# distributions. The test compares the observed number of events in each group to what
# would be expected if the null hypothesis were true (the survival curves were identical).
# The log rank statistic is approximately distributed as a chi-square test statistic.
log_rank = survdiff(Surv(time, status) ~ feature, data=data)
log_rank

# small p-value => the curves are different
# high p-value  => the curves are the same
```

```r
ggsurvplot(fit.feature, data = data,
           conf.int = T,
           risk.table = TRUE,
           risk.table.col = "strata",
           surv.median.line = "hv",
           ggtheme = theme_bw(),
           break.time.by = 90,
           legend.labs = c("Feat_1", "Feat_2"),
           legend.title = "Feature",
           palette = c("blue", "red"),
           title = 'Kaplan-Meier curves by a feature',
           pval = T)

### |--------------|
### | Hazard Ratio |
### |--------------|
# To quantify the difference in the survivals we can compute the hazard ratio, i.e.
# the ratio between the death hazard of the first group vs the other one.
# The risk of death in the first group is hazard_ratio * the risk of death in the second.
observed     = log_rank$obs
expected     = log_rank$exp
hazard_ratio = (observed[1]/expected[1])/(observed[2]/expected[2])
hazard_ratio

# = 1: no effect
# > 1: decrease in the hazard (increase of the survival) -> the numerator is a protective factor
# < 1: increase in the hazard (decrease of the survival) -> the numerator is a risk factor

### |-----------------------------------|
### | Survival probability at given time |
### |-----------------------------------|
t_0 = 180
summary(fit.feature, times=t_0)

# Survival probability every six months
t_0 = seq(0, 365*3, 182.5)
summary(fit.feature, times=t_0)

### ---------------------------------------------------------------------------------------
### COX PH MODEL - Univariate
### ---------------------------------------------------------------------------------------
# Compute Cox proportional-hazards regression models: coxph(formula, data, method)
# formula: linear model with a survival object as response variable
# data   : data frame containing the variables
# method : specifies how to handle ties
x1    = data$age
cox_m = coxph(Surv(time, status) ~ x1, data=data)
cox_m
summary(cox_m)

# 1) STATISTICAL SIGNIFICANCE
#    The column marked "z" gives the Wald statistic value. It corresponds to the
#    ratio of each regression coefficient to its standard error (z = coef/se(coef)).
#    The wald statistic evaluates, whether the beta coefficient of a given
#    variable is statistically significantly different from 0.
#
# 2) THE REGRESSION COEFFICIENTS
#    The second feature to note in the Cox model results is the the sign of the regression
#    coefficients (coef). A positive sign means that the hazard (risk of death) is higher,
#    and thus the prognosis is worse, for subjects with higher values of that variable.
#
# 3) HAZARD RATIO & CONFIDENCE INTERVAL
#    The exponentiated coefficients (exp(coef) = exp(0.0187) = 1.019), also known as hazard
#    ratios, give the effect size of covariates. For example, the increase of 1 unit in the
#    covariate increase the hazard of 1.9%. The summary output also gives upper and lower
#    95% confidence intervals for the hazard ratio (exp(coef)).
#
# 4) GLOBAL STATISTICAL SIGNIFICANCE OF THE MODEL
#    Finally, the output gives p-values for three alternative tests for overall significance
#    of the model: The likelihood-ratio test, Wald test, and score logrank statistics.
#    These three methods are asymptotically equivalent. For large enough N, they will give
#    similar results. For small N, they may differ somewhat. The Likelihood ratio test has
#    better behavior for small sample sizes, so it is generally preferred.

### |------------|
### | Curve plot |
### |------------|
plot(survfit(cox_m, data=data), lwd=2, lty=1, xlab='Time', ylab='Survival Probability',
     main='Baseline estimated survival probability')
grid()

### |---------------------------------------|
### | How the covariate influences the curve |
### |---------------------------------------|
# We take some values in the range of the covariate x1, for example:
x1_new = data.frame(x1 = c(50, 65, 80))
x1_new

fit_m = survfit(cox_m, newdata=x1_new)
fit_m

plot(fit_m, conf.int=T, col=c('red', 'green', 'blue'), lwd=2, lty=1,
     xlab='Time', ylab='Survival Probablity', main='Adjusted Survival Probability')
grid()
legend('topright', c('x1=50', 'x1=65', 'x1=80'), lty=c(1,1,1), lwd=c(2,2,2), col=c('red', 'green', 'blue'))

### ---------------------------------------------------------------------------------------
### COX PH MODEL - Multivariate
### ---------------------------------------------------------------------------------------
x1 = data$age
```

```r
x2 = as.factor(data$sex)
x3 = data$ph.karno
x4 = data$wt.loss

cox_mult = coxph(Surv(time, status) ~ x1 + x2 + x3 + x4, data=data)
cox_mult
summary(cox_mult)

# 1. The last three p-values indicate the significance of the model. These tests evaluate
#    the null hypothesis that all of the betas are 0 (low p-val => the model is valid).
# 2. We can look at the p-values of the singular covariates to see which are significant.
# 3. We can look at the HR for the covariates (exp(coef)) to see if they're >/< 1.
#    3.1. Consider the categorical variable, say x2, with exp(coef) = exp(0.514) = 1.67.
#         This means that being the (displayed) group of x2 increases the hazard by a factor
#         of 1.67, which means 67%. In this case being of that group is a bad prognostic.
#    3.2. Consider the continuous variable, say x3, with exp(coef) = exp(-0.013) = 0.987.
#         This means that, holding the other variables constant, a higher value fo x3 is
#         associated with good survival.

### |------------------------|
### | Plot for Hazard Ratios |
### |------------------------|
# It doesn't read the coxph with x1, .., xk
data$sex = as.factor(data$sex)
cox_plot = coxph(Surv(time, status) ~ age + sex + ph.karno + wt.loss, data=data)
ggforest(cox_plot, data=data)

### |------------|
### | Curve plot |
### |------------|
plot(survfit(cox_mult, data=data), lwd=2, lty=1, xlab='Time', ylab='Survival Probability',
     main='Baseline estimated survival probability')
grid()

### |---------------------------|
### | Cox Model Goodness of fit |
### |---------------------------|
### Martingale residuals
# We want the martingale residuals to have 0 mean along time
ggcoxdiagnostics(cox_mult, type='martingale')

### Deviance residuals
# These are a normalized transform of the martingale residuals.
# They should be symmetrically distributed about 0 with standard deviation of 1.
#   * Positive values correspond to individuals that died too soon compare to the expected survival times
#   * Negative values correspond to individuals that lieved too long
ggcoxdiagnostics(cox_mult, type='deviance')

### Shoenfeld residuals
# These residuals represent the difference between the observed covariate and the expected
# given the risk set at that time. They should be flat, centered about 0.
# A plot showing a non-random pattern against time is evidence of violation of the assumptions.
ggcoxdiagnostics(cox_mult, type='schoenfeld')

### Log(-log(KM)) - only for categorical variables
# We plot log(-log(KM(t))) vs. t or log(t) and look for parallelism.
# If we find parallelism then the assumptions are satisfied.
x_cat = as.factor(data$sex)
km    = survfit(Surv(time, status) ~ x_cat, data=data)
plot(km, fun='cloglog')

### |--------------|
### | COX.ZPH test |
### |--------------|
# The function cox.zph() is used to test the proportional hazards assumption for each
# covariate included in a Cox refression model fit. For each covariate, cox.zph() correlates
# the corresponding set of scaled Schoenfeld residuals with time, to test for independence
# between residuals and time. Also, it performs a global test for the model as a whole.
#
# The proportional hazard assumption is supported by a non-significant relationship between
# residuals and time, and refuted by a significant relationship.
#
# Test for PH using scaled Schoenfeld test for PH:
#    H0: Hazards are proportional
#    H1: Hazards are NOT proportional
# cox.zph() return tests for each X and for the global model
test.ph = cox.zph(cox_mult)
test.ph
```