Mock Exam
Non-Parametric Statistics

January 2, 2021

Algorithmic Instructions

- All the numerical values required need to be put on an A4 sheet and uploaded, alongside the required plots.
- For all computations based on permutation/resampling, as well as split conformal, use $B = 1000$ replicates, and $seed = 100$
- For full conformal prediction, set the grid size equal to 100.
- Both for confidence and prediction intervals, as well as tests, set $\alpha = 0.1$
- When reporting confidence/prediction intervals, always provide upper and lower bounds, together with the point value.

Exercise 1

In the file `snowlevel.rda` you can find a dataframe with the measurements of snow level [m] in town and snow level [m] in the highest point of the municipality for 100 towns of Trentino-Alto Adige and Aosta Valley. Assume all data independent and data within each geographical region also identically distributed.

Now:

1. Check the two groups, using bagplots based on the Tukey's depth, for the presence of bivariate outliers and, if present, remove them from the following analysis. Report the row indices of the outliers and upload the two bagplots.
2. Compute, for every group, the sample median of the snow levels according to the Tukey depth and report them.
3. Build a permutation test for the equality of the Tukey medians of the two populations, using as the test statistic the maximum-norm of the difference between the two sample medians, and report and comment the p-value of the test.
4. For the Aosta Valley, estimate via naive-bootstrap the variance and the bias of the the two components of the sample Tukey median and report them.

5. For the Aosta Valley, compute the reverse percentile bootstrap confidence intervals for the two components of the Tukey median and report them.

Exercise 2

The file `uphillperformance.rda` contains a dataframe with information about the climbing performance of 1000 ski-mountaineering athletes: `u.speed` is the vertical climbing speed [km/h], `t.days` is the number of training days per year [days/yr], `w.ski` is the weight of the skis the athlete uses [kg], and `c.intake` is average amount of calories an athlete eats on training days [Kcal/day]. Modeling this data as i.i.d. realizations of a four-dimensional random variable:

1. Build three univariate smoothing splines of order 4 (select the lambda parameter via Leave-One-Out CV) to predict the vertical climbing speed using the other covariates one at a time. Report the three required lambda values, the corresponding equivalent degrees of freedom, and upload three plots of the estimated regression lines.
2. By using a full conformal approach using the absolute value of the regression residuals as non conformity scores, compute three prediction intervals for vertical climbing speed provided by the three models when the three regressors assume their corresponding sample mean values, for the prediction use the optimal lambda/DoF found at point 1. Report the three intervals.
3. Build an additive model, using cubic b-spline terms for the main effects, and no interactions. Report the adjusted R^2 and the p-values of the approximate test for smooth terms.
4. Report the qq-plot of the residuals of the previous model and the p-value of the Shapiro test. According to the result of the test choose the correct approach to perform model selection, and reduce the model to the significant components. Upload a plot of the regression surface.
5. After having written the expression of the reduced additive model, provide an approximate confidence interval for the mean vertical uphill speed for an athlete with regressor values equal to the sample mean.

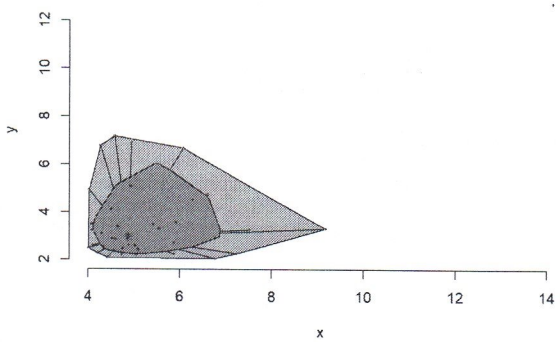
```
# ---------- Ex1 -------------------------------------------------------------------------
# Don't forget to set the wd
# Let's load the dataset
load("snowlevel.rda")
head(df_1)
```

```
##       town highest_point       region
## 1 4.118212      3.533241 Aosta_Valley
## 2 4.507993      4.136909 Aosta_Valley
## 3 5.038113      2.196594 Aosta_Valley
## 4 7.243578      2.258364 Aosta_Valley
## 5 6.077921      6.678795 Aosta_Valley
## 6 6.759198      2.050455 Aosta_Valley
```

```
# numerosity of the two groups
table(df_1$region)
```

```
##
## Aosta_Valley  Trentino_AA
##           40           60
```

```
# split the two groups
av_index = df_1$region=='Aosta_Valley'
data_av  = df_1[av_index,]
data_aa  = df_1[!av_index,]

# point 1
# we need to use a bagplot: remember that the bagplot is made by default using the Tukey distance.
av_obj = aplpack::bagplot(data_av[1:2])
plot(av_obj)
```

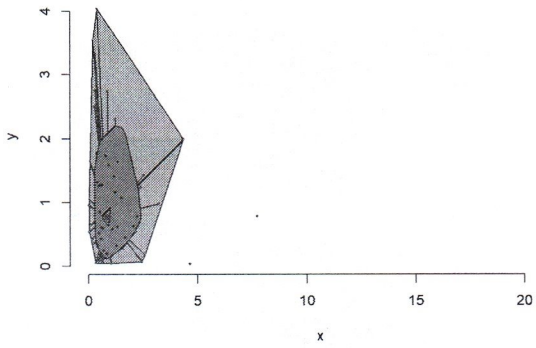



```
## [1] "bagplot plottet"
```

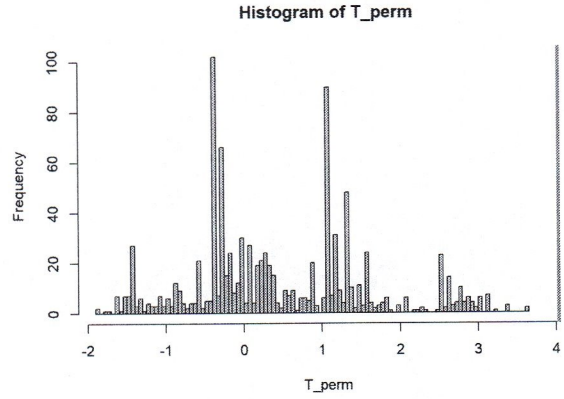
```
# 1 outlier
# where?
which(data_av$town>12)
```

```
## [1] 25
```

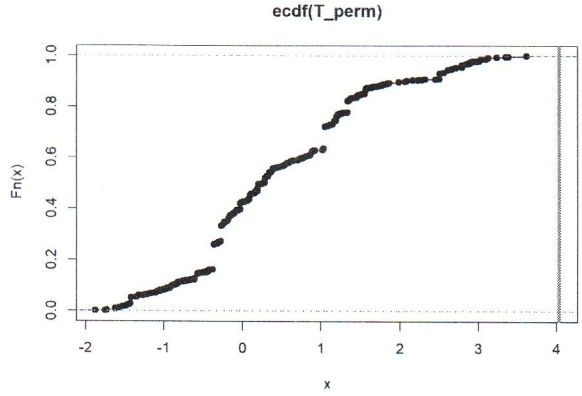
```
# row 25
# let's remove it
data_av_clean = data_av[-25,]

# now aa
aa_obj = aplpack::bagplot(data_aa[1:2])
# 3 outliers
# quick way is to "isolate them"
out_aa = which(data_aa[1]> 4.5 & data_aa[2]<1)
# 22, 23 and 40

data_aa_clean = data_aa[-out_aa,]

# point 2
# the "median" in this case is the depth median, so the point with minimal depth
library(DepthProc)
```




```
av_median = depthMedian(data_av_clean[1:2],depth_params = list(method='Tukey'))
aa_median = depthMedian(data_aa_clean[1:2],depth_params = list(method='Tukey'))

# point 3
# compute test statistic
T0 = max(av_median-aa_median)

# let's create a "pooled" dataset
df_1_clean = rbind(data_av_clean[1:2],data_aa_clean[1:2])

# new numerosities
n_av  = nrow(data_av_clean)
n_aa  = nrow(data_aa_clean)
n_tot = n_av + n_aa

# Estimating the permutational distribution under H0
library(pbapply)
B = 1000

wrapper = function(){
  perm           = sample(1:n_tot)
  data_perm      = df_1_clean[perm,]
  av_perm        = data_perm[1:n_av,]
  aa_perm        = data_perm[(n_av+1):n_tot,]
  av_median_perm = depthMedian(av_perm,depth_params = list(method='Tukey'))
  aa_median_perm = depthMedian(aa_perm,depth_params = list(method='Tukey'))
  max(av_median_perm-aa_median_perm)
}

set.seed(100)
T_perm = pbreplicate(B,wrapper(),simplify = 'vector')

# plotting the permutational distribution under H0
hist(T_perm,xlim=range(c(T0,T_perm)),breaks=100)
abline(v=T0,col=3,lwd=4)
```




```
plot(ecdf(T_perm),xlim=range(c(T0,T_perm)))
abline(v=T0,col=3,lwd=4)
```




```
# p-value
p_val = sum(T_perm>=T0)/B
p_val
```

```
## [1] 0
```

```
# p-val is zero, I cannot refuse the null hypothesis

# point 4 - 5
# it's enough to generate only once the resampled distribution
wrapper_bs = function(){
  perm           = sample(1:n_av,replace = T)
  data_perm      = data_av_clean[perm,]
  av_median_perm = depthMedian(data_perm[1:2],depth_params = list(method='Tukey'))
}

set.seed(100) # don't forget to do it every time you run a random number generation
bs_dist = pbreplicate(B,wrapper_bs())

# variance
apply(bs_dist,1,var)
```

```
##          town highest_point
##    0.07060550    0.07350193
```

```
# bias
rowMeans(bs_dist)-av_median
```

```
##          town highest_point
##    0.03526303    0.08906276
```

```
alpha = 0.1

# right quantile wrapper
right_wrap = function(data){quantile(data,1-alpha/2)}
left_wrap  = function(data){quantile(data,alpha/2)}
#Left quantile wrapper

right.quantile = apply(bs_dist,1,right_wrap)
left.quantile  = apply(bs_dist,1,left_wrap)

CI.RP = rbind(av_median - (right.quantile - av_median),
              av_median,
              av_median - (left.quantile - av_median))
CI.RP
```

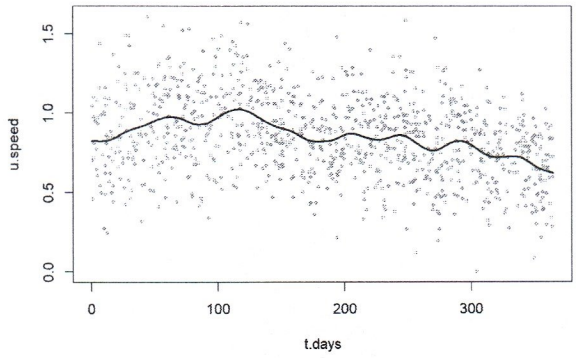
```
##               town highest_point
##           4.181124      2.340906
## av_median 4.861681      2.914937
##           5.129483      3.212666
```

```
rm(list=ls())
```

```
# ---------- Ex2 ----------------------------------------------------------------------
load("uphillperformance.rda")
attach(df_2)

# point 1
# the splines built by the smooth.spline command are of order 4 by default.
fit = smooth.spline(t.days,u.speed,cv=T)
```

```
## Warning in smooth.spline(t.days, u.speed, cv = T): cross-validation with non-
## unique 'x' values seems doubtful
```

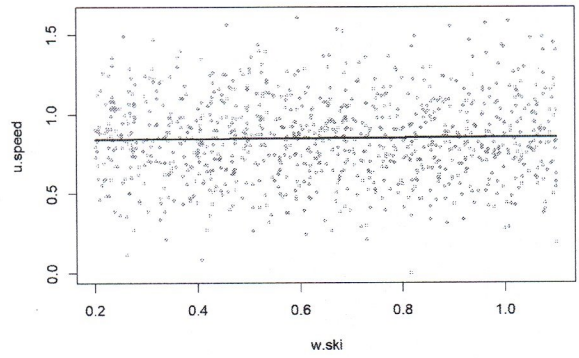
```
plot(t.days ,u.speed,cex =.5, col =" darkgrey ")
lines(fit,col="blue",lwd=2)
```




```
fit$lambda
```

```
## [1] 0.000199546
```

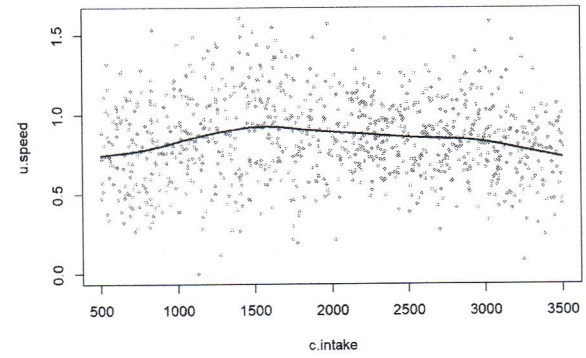
```
tdays.df = fit$df

fit = smooth.spline(w.ski,u.speed,cv=T)
plot(w.ski ,u.speed,cex =.5, col =" darkgrey ")
lines(fit,col="blue",lwd=2)
```




```
fit$lambda
```

```
## [1] 78.64727
```

```
wski.df = fit$df

fit = smooth.spline(c.intake,u.speed,cv=T)
plot(c.intake ,u.speed,cex =.5, col =" darkgrey ")
lines(fit,col="blue",lwd=2)
```




```
fit$lambda
```

```
## [1] 0.01619654
```

```r
cintake.df = fit$df

# point 2
# for the conformal part, let's use the conformalInference package
# (it uses by default the desired NCM)
library(conformalInference)

# we need to define custom train and predict functions.
# let's not forget the optimal value of df/lambda
train_ss = function(x,y,out=NULL){
  smooth.spline(x,y,df=tdays.df)
}

predict_ss = function(obj, new_x){
  predict(obj,new_x)$y
}

pred = conformal.pred(t.days,u.speed,mean(t.days),train.fun = train_ss,
                      predict.fun = predict_ss,alpha=.1)
c(pred$lo,pred$pred,pred$up)
```

```
## [1] 0.4665826 0.8239787 1.1968858
```

```r
# according to how I specified my train_ss, I need to redefine for every univariate spline.

train_ss = function(x,y,out=NULL){
  smooth.spline(x,y,df=wski.df)
}

pred = conformal.pred(w.ski,u.speed,mean(w.ski),train.fun = train_ss,
                      predict.fun = predict_ss,alpha=.1)
c(pred$lo,pred$pred,pred$up)
```

```
## [1] 0.4665826 0.8486303 1.2374582
```

```r
train_ss=function(x,y,out=NULL){
  smooth.spline(x,y,df=cintake.df)
}

pred=conformal.pred(c.intake,u.speed,mean(c.intake),train.fun = train_ss,
                    predict.fun = predict_ss,alpha=.1)
c(pred$lo,pred$pred,pred$up)
```

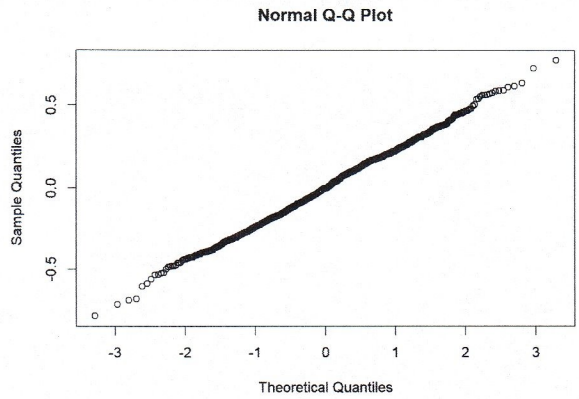
```
## [1] 0.5071550 0.8924194 1.2780306
```

```r
# point3
library(mgcv)
```

```r
# cubic splines are built using the bs='cr' argument
model_gam = gam(u.speed ~ s(t.days,bs='cr') + s(w.ski,bs='cr') + s(c.intake,bs='cr'))
summary(model_gam)
```

```
##
## Family: gaussian
## Link function: identity
##
## Formula:
## u.speed ~ s(t.days, bs = "cr") + s(w.ski, bs = "cr") + s(c.intake,
##     bs = "cr")
##
## Parametric coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 0.848826   0.007437   114.1   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##              edf Ref.df      F p-value
## s(t.days)   5.383  6.485 22.237  <2e-16 ***
## s(w.ski)    1.000  1.000  2.307   0.129
## s(c.intake) 3.808  4.703  9.131  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.161   Deviance explained = 16.9%
## GCV = 0.055934  Scale est. = 0.055308  n = 1000
```

```r
# point4
# if my residuals are normal, I can think about using the default tests,
# otherwise I need to implement a permutational/boostrap strategy
qqnorm(model_gam$residuals)
```




```
shapiro.test(model_gam$residuals)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  model_gam$residuals
## W = 0.99829, p-value = 0.4284
```

```
# luckly my residuals are normal, so I can use the default tests.

# point5
model_gam = gam(u.speed ~ s(t.days,bs='cr') + s(c.intake,bs='cr'))
summary(model_gam)
```

```
##
## Family: gaussian
## Link function: identity
##
## Formula:
## u.speed ~ s(t.days, bs = "cr") + s(c.intake, bs = "cr")
##
## Parametric coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 0.848826   0.007442   114.1   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##               edf Ref.df      F p-value
## s(t.days)   5.417  6.522 21.830  <2e-16 ***
## s(c.intake) 3.852  4.755  8.963  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =   0.16   Deviance explained = 16.7%
## GCV = 0.055951  Scale est. = 0.055377  n = 1000
```

```
# my regression surface is indeed a surface, so I can plot it by using rgl

days.grid   = seq(range(t.days)[1],range(t.days)[2],length.out = 100)
intake.grid = seq(range(c.intake)[1],range(c.intake)[2],length.out = 100)
grid        = expand.grid(days.grid,intake.grid)
names(grid) = c('t.days','c.intake')
pred        = predict(model_gam,newdata=grid)

library(rgl)
persp3d(days.grid,intake.grid,pred,col='grey30')
points3d(t.days,c.intake,u.speed,col='black',size=5)

newdata = data.frame(t.days=mean(t.days),c.intake=mean(c.intake))
pred    = predict(model_gam,newdata=newdata,se.fit=T)
alpha   = .1

lwr = pred$fit-pred$se.fit*qt(1-(alpha/2),nrow(df_2))
lvl = pred$fit
upr = pred$fit+pred$se.fit*qt(1-(alpha/2),nrow(df_2))

c(lwr,lvl,upr)
```

```
##         1         1         1
## 0.8629651 0.8974877 0.9320104
```

```
#-------------------------------------------------------------------------------
# Ex 1
#-------------------------------------------------------------------------------
B     = 1000
seed  = 100
alpha = 0.1

load('snowlevel.rda')
data = df_1
head(data)
```

```
##       town highest_point       region
## 1 4.118212      3.533241 Aosta_Valley
## 2 4.507993      4.136909 Aosta_Valley
## 3 5.038113      2.196594 Aosta_Valley
## 4 7.243578      2.258364 Aosta_Valley
## 5 6.077921      6.678795 Aosta_Valley
## 6 6.759198      2.050455 Aosta_Valley
```

```
summary(data)
```

```
##       town          highest_point              region
##  Min.   : 0.01193   Min.   : 0.03216   Aosta_Valley:40
##  1st Qu.: 0.58333   1st Qu.: 0.58805   Trentino_AA :60
##  Median : 2.19059   Median : 1.75300
##  Mean   : 3.04469   Mean   : 2.07203
##  3rd Qu.: 4.67113   3rd Qu.: 2.87399
##  Max.   :21.19029   Max.   :12.70787
```

```
#----#
# 1. #
#----#
library(rgl)
library(DepthProc)
```

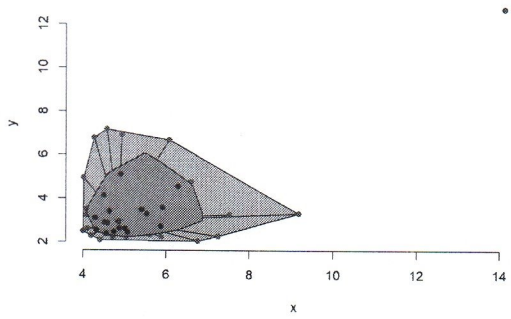
```
data_1 = data[which(data$region=='Aosta_Valley'), c(1,2)]
data_2 = data[which(data$region=='Trentino_AA'), c(1,2)]
dim(data_1)
```

```
## [1] 40  2
```

```
dim(data_2)
```

```
## [1] 60  2
```

```
method_name = 'Tukey'

bgplot_1 = aplpack::bagplot(data_1, cex=1)
```



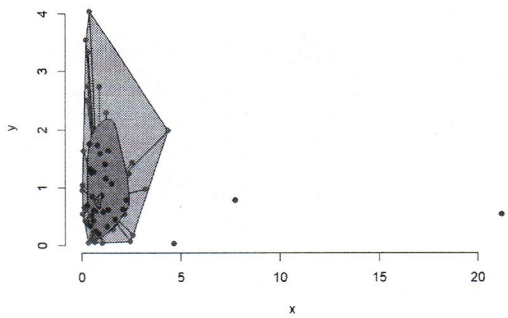

```
bgplot_1$pxy.outlier
```

```
##             x        y
## [1,] 14.12985 12.70787
```

```
ind1_out = which(data_1$town>10)
data_1   = data_1[-ind1_out,]
dim(data_1)
```

```
## [1] 39  2
```

```
bgplot_2 = aplpack::bagplot(data_2, cex=1)
```




```
bgplot_2$pxy.outlier
```

```
##              x          y
## [1,]  4.651733 0.03215715
## [2,] 21.190289 0.54348727
## [3,]  7.721833 0.78122826
```

```
ind2_out = which(data_2$town>4 & data_2$highest_point<1)
data_2 = data_2[-ind2_out,]
dim(data_2)
```

```
## [1] 57  2
```

```
#----#
# 2. #
#----#
data_1_med = depthMedian(data_1, depth_params=list(method=method_name))
data_1_med
```

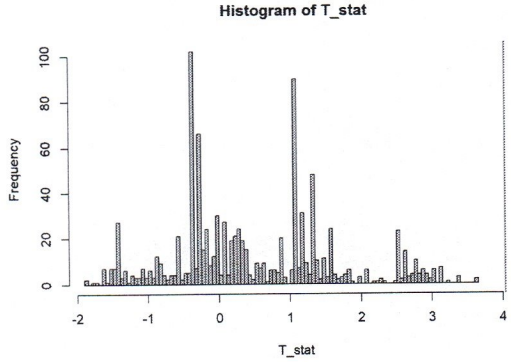
```
##          town highest_point
##      4.861681      2.914937
```

```
data_2_med = depthMedian(data_2, depth_params=list(method=method_name))
data_2_med
```

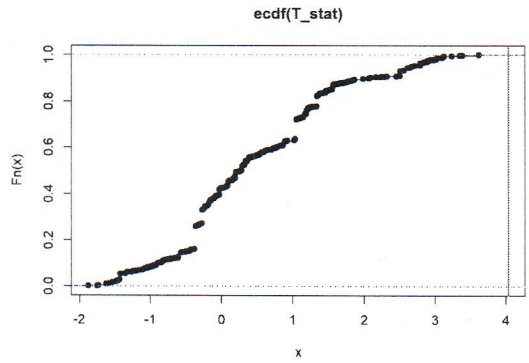
```
##          town highest_point
##     0.8290935     0.7265828
```

```
#----#
# 3. #
#----#
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)
T0     = max(data_1_med - data_2_med)
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_med = depthMedian(data_1_perm, depth_params=list(method=method_name))
  data_2_perm_med = depthMedian(data_2_perm, depth_params=list(method=method_name))
  T_stat[k]       = max(data_1_perm_med - data_2_perm_med)
  print(k/B)
}
```

```
# Permutational distribution of T
hist(T_stat, xlim=range(c(T_stat,T0)), breaks=100)
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
```

```
## [1] 0
```

```
# The p-value is a mathematical 0, we believe that the two medians are different

#----#
# 4. #
#----#
set.seed(seed)
Tb_0   = depthMedian(data_1, depth_params=list(method=method_name))
T_boot = cbind(numeric(B), numeric(B))
for(k in 1:B){
  ind         = sample(1:dim(data_1)[1], replace=T)
  data_1_boot = data_1[ind,]
  T_boot[k,]  = depthMedian(data_1_boot, depth_params=list(method=method_name))
  print(k/B)
}
```

```
# Variance
apply(T_boot, 2, var)
```

```
## [1] 0.06920829 0.07323471
```

```
# Bias
apply(T_boot, 2, mean) - Tb_0
```

```
##          town highest_point
##    0.03536604    0.08861120
```

```
#----#
# 5. #
#----#
# Reverse Percentile - 1
right_quantile = quantile(T_boot[,1], 1-alpha/2)
left_quantile  = quantile(T_boot[,1], alpha/2)
CI_1 = c(Tb_0[1]-(right_quantile-Tb_0[1]), Tb_0[1], Tb_0[1]-(left_quantile-Tb_0[1]))
CI_1
```

```
##     town     town     town
## 4.181124 4.861681 5.129483
```

```
# Reverse Percentile - 2
right_quantile = quantile(T_boot[,2], 1-alpha/2)
left_quantile  = quantile(T_boot[,2], alpha/2)
CI_2 = c(Tb_0[2]-(right_quantile-Tb_0[2]), Tb_0[2], Tb_0[2]-(left_quantile-Tb_0[2]))
CI_2
```

```
## highest_point highest_point highest_point
##      2.340906      2.914937      3.219675
```

```
#-------------------------------------------------------------------------------
# Ex 2
#-------------------------------------------------------------------------------
B     = 1000
seed  = 100
alpha = 0.1

load('uphillperformance.rda')
data = df_2
head(data)
```

```
##     u.speed    t.days     w.ski  c.intake
## 1 0.5651256 171.39566 0.3789990 2888.3395
## 2 0.7784163 174.99206 1.0529804 2092.3586
## 3 0.9008884 241.12479 0.4358677 1896.8733
## 4 1.2240380  62.47449 0.3156987 2218.1946
## 5 1.5335203  77.02170 0.6729155  835.5832
## 6 0.5691382 193.06364 0.7429789  582.1884
```

```
summary(data)
```

```
##     u.speed           t.days             w.ski           c.intake
##  Min.   :0.0000   Min.   :  0.2907   Min.   :0.2007   Min.   : 502.1
##  1st Qu.:0.6801   1st Qu.: 98.0771   1st Qu.:0.4265   1st Qu.:1260.7
##  Median :0.8493   Median :189.8703   Median :0.6380   Median :2038.9
##  Mean   :0.8488   Mean   :187.8606   Mean   :0.6448   Mean   :1999.9
##  3rd Qu.:1.0269   3rd Qu.:277.0679   3rd Qu.:0.8667   3rd Qu.:2729.8
##  Max.   :1.6067   Max.   :364.6957   Max.   :1.0999   Max.   :3499.3
```

```
#----#
# 1. #
#----#
y  = data$u.speed
x1 = data$t.days
x2 = data$w.ski
x3 = data$c.intake

fit_1 = smooth.spline(x1, y, cv=T)
```

```
## Warning in smooth.spline(x1, y, cv = T): cross-validation with non-unique 'x'
## values seems doubtful
```

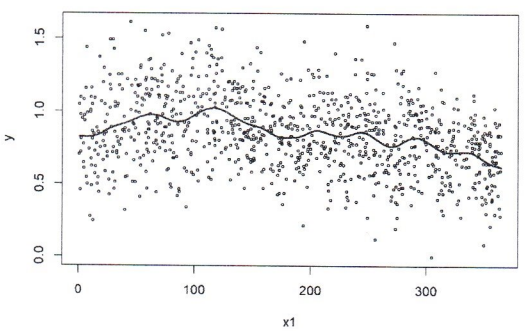
```
fit_1$df
```

```
## [1] 17.66694
```

```
fit_1$lambda
```

```
## [1] 0.000199546
```

```
plot(x1, y, cex=.5)
lines(fit_1, col='blue', lwd=2)
```



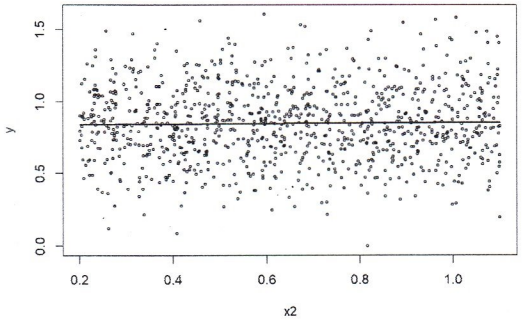

```
fit_2 = smooth.spline(x2, y, cv=T)
fit_2$df
```

```
## [1] 2.030227
```

```
fit_2$lambda
```

```
## [1] 78.64727
```

```
plot(x2, y, cex=.5)
lines(fit_2, col='blue', lwd=2)
```




```
fit_3 = smooth.spline(x3, y, cv=T)
fit_3$df
```

```
## [1] 6.567047
```

```
fit_3$lambda
```

```
## [1] 0.01619654
```

```
plot(x3, y, cex=.5)
lines(fit_3, col='blue', lwd=2)

#----#
# 2. #
#----#
opt_1 = fit_1$df
opt_2 = fit_2$df
opt_3 = fit_3$df

library(conformalInference)

# x1
train_ss   = function(x,y,out=NULL){smooth.spline(x,y,df=opt_1)}
predict_ss = function(obj, new_x){predict(obj,new_x)$y}
c_preds    = conformal.pred(x1, y, mean(x1), alpha=alpha, train.fun=train_ss, predict.fun=predict_ss)

cbind(c_preds$lo, c_preds$pred, c_preds$up)
```

```
##           [,1]      [,2]     [,3]
## [1,] 0.4665826 0.8239787 1.196886
```

```
# x2
train_ss   = function(x,y,out=NULL){smooth.spline(x,y,df=opt_2)}
predict_ss = function(obj, new_x){predict(obj,new_x)$y}
c_preds    = conformal.pred(x2, y, mean(x2), alpha=alpha, train.fun=train_ss, predict.fun=predict_ss)

cbind(c_preds$lo, c_preds$pred, c_preds$up)
```

```
##           [,1]      [,2]     [,3]
## [1,] 0.4665826 0.8486303 1.237458
```

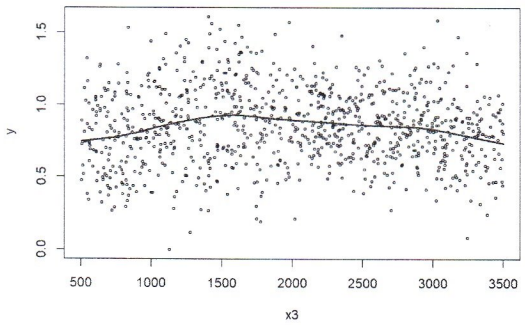
```
# x3
train_ss   = function(x,y,out=NULL){smooth.spline(x,y,df=opt_3)}
predict_ss = function(obj, new_x){predict(obj,new_x)$y}
c_preds    = conformal.pred(x3, y, mean(x3), alpha=alpha, train.fun=train_ss, predict.fun=predict_ss)

cbind(c_preds$lo, c_preds$pred, c_preds$up)
```

```
##           [,1]      [,2]     [,3]
## [1,] 0.507155 0.8924194 1.278031
```

```
#----#
# 3. #
#----#
library(mgcv)
```

```
## Loading required package: nlme
```

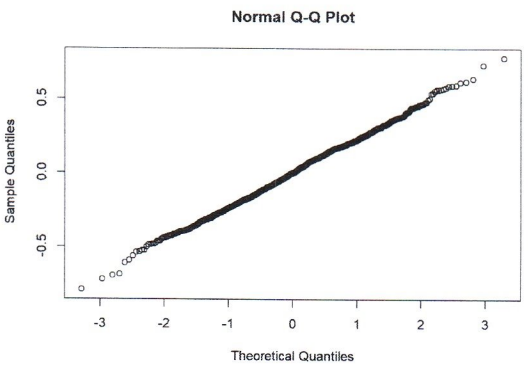
```
## This is mgcv 1.8-33. For overview type 'help("mgcv-package")'.
```




```
gam_m = gam(y ~ s(x1, bs='cr') + s(x2, bs='cr') + s(x3, bs='cr'))
summary(gam_m)
```

```
##
## Family: gaussian
## Link function: identity
##
## Formula:
## y ~ s(x1, bs = "cr") + s(x2, bs = "cr") + s(x3, bs = "cr")
##
## Parametric coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 0.848826   0.007437   114.1   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##         edf Ref.df      F p-value
## s(x1) 5.383  6.485 22.237  <2e-16 ***
## s(x2) 1.000  1.000  2.307   0.129
## s(x3) 3.808  4.703  9.131  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.161   Deviance explained = 16.9%
## GCV = 0.055934  Scale est. = 0.055308  n = 1000
```

```
#----#
# 4. #
#----#
qqnorm(gam_m$residuals)
```




```
shapiro.test(gam_m$residuals)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  gam_m$residuals
## W = 0.99829, p-value = 0.4284
```

```
# Normality assumptions hold
gam_red = gam(y ~ s(x1, bs='cr') + s(x3, bs='cr'))
summary(gam_red)
```

```
##
## Family: gaussian
## Link function: identity
##
## Formula:
## y ~ s(x1, bs = "cr") + s(x3, bs = "cr")
##
## Parametric coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 0.848826   0.007442   114.1   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##         edf Ref.df      F p-value
## s(x1) 5.417  6.522 21.830  <2e-16 ***
## s(x3) 3.852  4.755  8.963  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =   0.16   Deviance explained = 16.7%
## GCV = 0.055951  Scale est. = 0.055377  n = 1000
```

```
library(rgl)
x1_grid     = seq(range(x1)[1], range(x1)[2], length.out=100)
x3_grid     = seq(range(x3)[1], range(x3)[2], length.out=100)
grid        = expand.grid(x1_grid, x3_grid)
names(grid) = c('x1', 'x3')
pred        = predict(gam_red, newdata=grid)
persp3d(x1_grid, x3_grid, pred, col='yellow')
points3d(x1, x3, y, col='black', size=5)

#----#
# 5. #
#----#
# Prediction
newdata = data.frame(x1=mean(x1), x3=mean(x3))
pred    = predict(gam_red, 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)
```

```
##         lwr       lvl       upr
## 1 0.8629651 0.8974877 0.9320104
```