TP Statistique

Gabriel Canaple

19 Janvier 2024

(décrire la régression, l'état des résidus, le test de normalité, est-elle centrée, variance est constante ? indépendance ?) test de shapiro, student, chi2 régression linéaire = chap 6 du cours y = ax + b + epsilon y = variable expliquée a, b = inconnues x = variable explicative epsilon = résidus test de pertinence : a=0? test de biais : b=0?

on peut aussi avoir plusieurs variables explicatives : y = a1$x1$ + $a2$x2 + ... + an*xn + epsilon

PENSER A définir le risque alpha pour chacun des tests (si on sait pas quoi prendre prendre 5%)

Voici le plan de ce qui sera fait dans le TP.

0. Visualisation de chemins

Lecture du fichier des villes :

```
set.seed(35)
villes <- read.csv('DonneesGPSvilles.csv',header=TRUE,dec='.',sep=';',quote="\"")
str(villes)
```

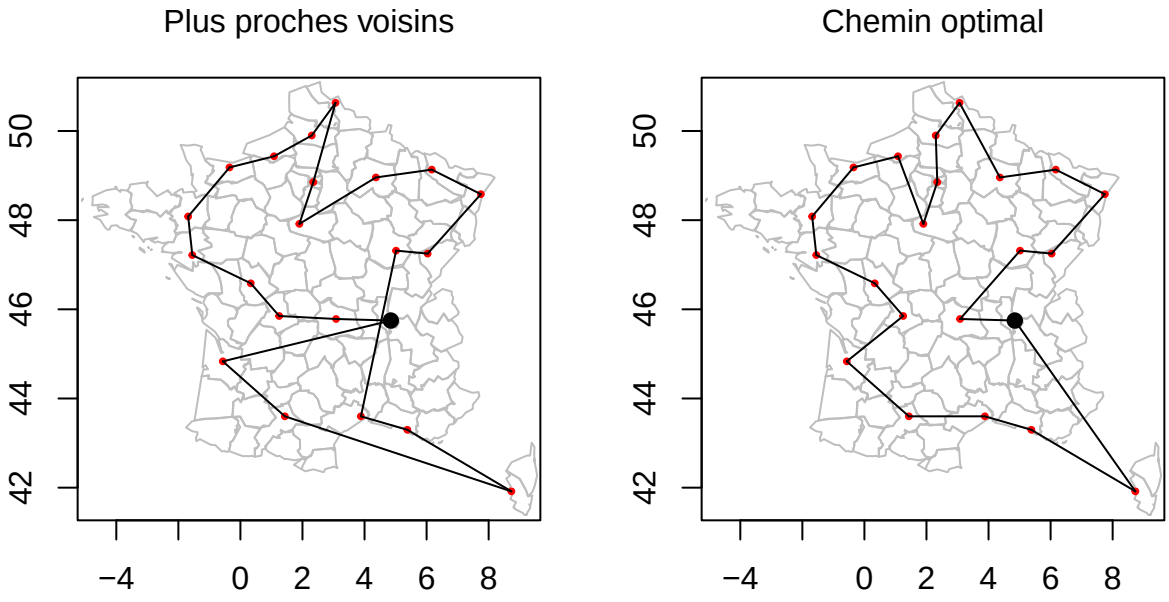
```
## 'data.frame':    22 obs. of  5 variables:
##  $ EU_circo : chr  "Sud-Est" "Sud-Est" "Nord-Ouest" "Est" ...
##  $ region   : chr  "Rhône-Alpes" "Corse" "Picardie" "Franche-Comté" ...
##  $ ville    : chr  "Lyon" "Ajaccio" "Amiens" "Besançon" ...
##  $ latitude : num  45.7 41.9 49.9 47.2 44.8 ...
##  $ longitude: num  4.847 8.733 2.3 6.033 -0.567 ...
```

Représentation des chemins par plus proches voisins et du chemin optimal :

```
set.seed(35)
coord <- cbind(villes$longitude,villes$latitude)
dist <- distanceGPS(coord)
voisins <- TSPnearest(dist)

pathOpt <- c(1,8,9,4,21,13,7,10,3,17,16,20,6,19,15,18,11,5,22,14,12,2)

par(mfrow=c(1,2),mar=c(1,1,2,1))
plotTrace(coord[voisins$chemin,], title='Plus proches voisins')
plotTrace(coord[pathOpt,], title='Chemin optimal')
```



Les longueurs des trajets (à vol d'oiseau) valent respectivement, pour la méthode des plus proches voisins :

```
## [1] 4303.568
```

et pour la méthode optimale :

```
## [1] 3793.06
```

Ceci illustre bien l'intérêt d'un algorithme de voyageur de commerce. Nous allons dans la suite étudier les performances de cet algorithme.

1. Comparaison d'algorithmes

Nombre de sommets fixes et graphes "identiques".

```
set.seed(35)
n <- 10 #nombre de noeud

#example de lancement unitaire
sommets <- data.frame(x = runif(n), y = runif(n))
couts <- distance(sommets)
TSPsolve(dist,'nearest')
```

```
## [1] 4303.568
```

```r
set.seed(35)

#calcul de plusieurs simulation de graphes qui seront analysées par les 5 méthodes
nsimu <- 50 #nombre de simu
methods <- c('arbitrary_insertion', 'repetitive_nn','two_opt','nearest','branch')
res <- array(0,dim=c(nsimu,length(methods)))
for(i in 1:nsimu){
  points <- data.frame(x = runif(n), y = runif(n))
  dist <- distance(points)
  res[i,] <- (sapply(methods, function(m){TSPsolve(dist,m)}))
}
colnames(res) <- c('insertion','repet_nn','two_opt','nearest','branch')
res
```

```
##       insertion repet_nn  two_opt  nearest   branch
##  [1,]  2.633229 2.633229 2.794858 3.115361 2.633229
##  [2,]  2.825005 2.825005 2.825005 2.825005 2.825005
##  [3,]  2.770953 2.772088 2.686786 2.772088 2.686786
##  [4,]  2.190799 2.278822 2.383056 2.340315 2.190799
##  [5,]  2.912380 2.912380 3.273722 3.190969 2.912380
##  [6,]  2.595074 2.595074 2.595074 2.655550 2.595074
##  [7,]  2.646389 2.645636 2.645636 2.646389 2.645636
##  [8,]  2.745412 2.858674 2.858674 2.858674 2.745412
##  [9,]  2.835688 2.923733 3.376079 3.859938 2.835688
## [10,]  3.373221 3.207543 3.862590 3.207543 3.176170
## [11,]  3.124444 3.124444 3.616421 3.124444 3.124444
## [12,]  2.890743 2.773278 2.773278 2.773278 2.773278
## [13,]  2.848830 2.844378 3.119311 2.844378 2.844378
## [14,]  2.519599 2.519599 2.728469 2.704847 2.519599
## [15,]  3.184302 3.461219 3.184302 3.677381 3.052222
## [16,]  2.884755 2.518767 2.518767 3.205048 2.518767
## [17,]  2.850912 3.068675 3.022136 3.633584 2.850912
## [18,]  2.354301 2.561375 2.452921 2.952219 2.354301
## [19,]  3.061982 3.047227 3.047227 3.047227 3.047227
## [20,]  2.740308 2.926457 2.954929 3.176949 2.721728
## [21,]  3.564941 3.653052 4.192473 3.653052 3.372445
## [22,]  3.781262 3.482438 3.631811 3.669414 3.429176
## [23,]  2.657181 2.556435 3.224014 2.974170 2.556435
## [24,]  2.461914 2.461914 2.589318 2.830472 1.577505
## [25,]  2.744475 2.744475 2.955204 2.744475 2.744475
## [26,]  2.803303 2.844387 3.003194 2.861528 2.803303
## [27,]  3.364041 3.364041 4.145997 3.469081 3.364041
## [28,]  2.979277 3.002270 3.472056 3.002270 2.979277
## [29,]  3.262933 3.175538 3.314849 3.428114 3.175538
## [30,]  2.591442 2.689641 2.696248 2.782370 2.579881
## [31,]  3.230837 3.183502 3.195253 3.545229 3.080201
## [32,]  2.709654 2.874452 2.730658 3.313752 2.709654
## [33,]  2.680623 2.680623 2.680623 2.680623 2.680623
## [34,]  3.326839 3.326839 3.898505 3.337159 3.326839
## [35,]  2.580709 2.668735 2.855464 2.857004 2.580709
## [36,]  2.930667 3.103744 2.930667 3.103744 2.930667
## [37,]  2.647981 2.647981 3.472386 2.647981 2.647981
## [38,]  3.132980 3.151552 3.151552 3.568476 3.064892
```

```
## [39,]  3.265676 3.048137 3.048137 4.035719 3.048137
## [40,]  3.287300 3.287300 3.287300 3.341229 3.188474
## [41,]  3.117638 3.216568 3.266287 3.261300 3.090471
## [42,]  2.831934 2.817945 2.817945 3.141993 2.817945
## [43,]  3.547758 3.196025 3.196025 3.413817 3.196025
## [44,]  2.894102 2.956698 2.956201 3.376576 2.894102
## [45,]  2.491742 2.658104 2.620072 2.841973 2.478914
## [46,]  2.359488 2.533346 3.180524 2.533346 2.359488
## [47,]  3.553958 3.374373 3.749130 3.611851 3.374373
## [48,]  2.390917 2.540526 2.975510 2.691639 2.390917
## [49,]  3.110281 3.334878 3.334878 3.800788 3.110281
## [50,]  2.575929 2.746920 2.575929 2.821483 1.960579
```

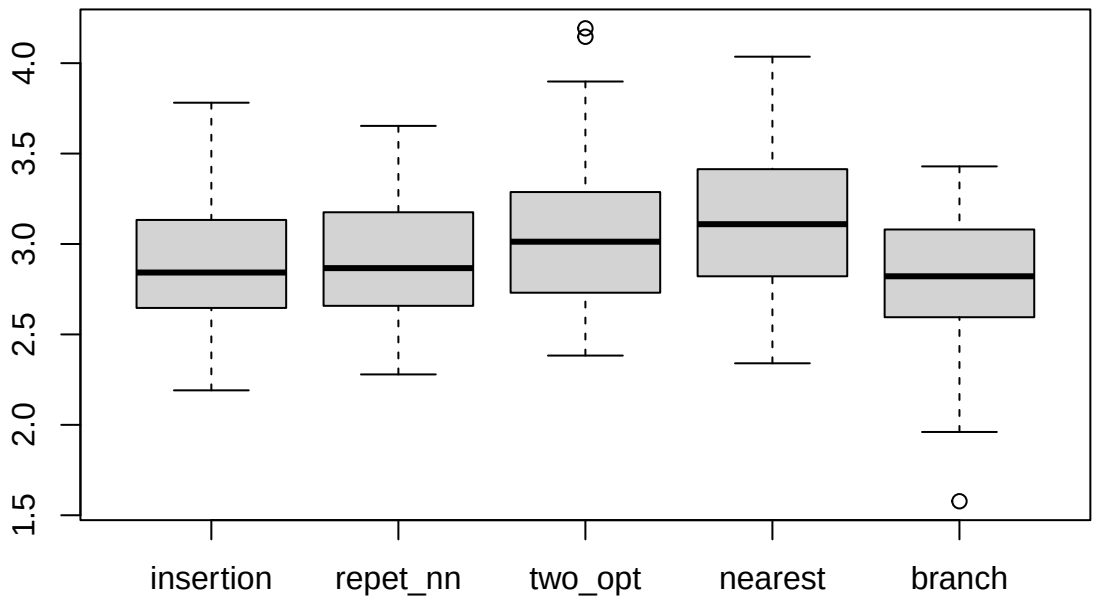
1.1. Longueur des chemins

Comparaison des longueurs de différentes méthodes :

- boxplots
- test entre 'nearest' et 'branch'
- tests 2 à 2

```
set.seed(35)
res2 <- as.vector(res)
meth_names <- c('insertion','repetitive_nn','two_opt','nearest','branch')
methods2 <- rep(meth_names,each=nsimu)

boxplot(res)
```




```
shapiro.test(res[,1])
```

```
##
##  Shapiro-Wilk normality test
##
## data:  res[, 1]
## W = 0.97675, p-value = 0.4246
```

```
shapiro.test(res[,2])
```

```
##
##  Shapiro-Wilk normality test
##
## data:  res[, 2]
## W = 0.97601, p-value = 0.3983
```

```
shapiro.test(res[,3])
```

```
##
##  Shapiro-Wilk normality test
##
## data:  res[, 3]
## W = 0.95152, p-value = 0.03939
```

```r
shapiro.test(res[,4])
```

```
##
##  Shapiro-Wilk normality test
##
## data:  res[, 4]
## W = 0.96797, p-value = 0.1911
```

```r
shapiro.test(res[,5])
```

```
##
##  Shapiro-Wilk normality test
##
## data:  res[, 5]
## W = 0.95604, p-value = 0.06067
```

```r
nearest_branch <- res[,4] - res[,5]
shapiro.test(nearest_branch) #ne suit pas une loi normale
```

```
##
##  Shapiro-Wilk normality test
##
## data:  nearest_branch
## W = 0.87519, p-value = 8.02e-05
```

```r
t.test(res[,4], res[,5], alternative = "greater")
```

```
##
##  Welch Two Sample t-test
##
## data:  res[, 4] and res[, 5]
## t = 4.0462, df = 97.621, p-value = 5.211e-05
## alternative hypothesis: true difference in means is greater than 0
## 95 percent confidence interval:
##  0.1814195       Inf
## sample estimates:
## mean of x mean of y
##  3.119036  2.811328
```

```r
result <- pairwise.t.test(res2,methods2, p.adjust.method = "bonferroni")
```

Analyse des boxplots

On observe qu'il n'y a pas de différence significative entre les algorithmes. Ils ont une moyenne proche, même si on remarque two_opt et nearest ont des valeurs globalement plus élevées que les trois autres, qui sont, eux, plus proches de 3.0 en valeur. two_opt se distingue aussi par une disperion plus faible que celle des quatre autres.

Test de normalité

Avec un seuil de 5%, on ne rejette pas l'hypothèse nulle pour les tests de Shapiro-Wilk (qui est que la distribution suit une loi normale), donc les distributions pour nearest et branch suivent une loi normale.

Comparaison de nearest et de branch and bound

Nous effectuons une soustraction afin de limiter l'incertitude. Nous pouvons le faire car nos deux algorithmes ont été exécutés sur des échantillons appariés. Nous prenons comme risque α le seuil à 5%. La p-valeur obtenue pour l'hypothèse que la différence est inférieure à 0 est égale à 0.008%, nous rejetons donc l'hypothèse H0, et nous pouvons donc en conclure que la différence est significativement supérieure à 0. La p-valeur étant très inférieure à notre risque, nous pouvons être assez confiants dans notre conclusion.

Test deux à deux

Nous prenons un risque α égal à 5%. La seule différence notable est entre nearest et branch, avec une p-valeur inférieure à 5%. Nous avons choisi de répartir les algorithmes entre 4 groupes : - Le premier groupe est composé d'insertion, et de repetitive-nn. - Le deuxième groupe se constitue de nearest. - Le troisième groupe contient two-opt. - Le quatrième groupe contient branch and bound La philosophie de cette répartition est de rassembler les algorithmes similaires entre eux, et ayant des différences avec les même algorithmes. A compléter.

1.2. Temps de calcul

Comparaison des temps à l'aide du package microbenchmark.

Exemple d'application de microbenchmark :

```
set.seed(35)
microbenchmark(sqrt(x),x^0.5, times=100, setup={x <- runif(1)})
```

```
## Unit: nanoseconds
##     expr min    lq   mean median    uq  max neval cld
##  sqrt(x)  75  94.0 121.00   98.0 111.0 1687   100  a
##    x^0.5 125 152.5 210.59  165.5 176.5 2751   100   b
```

Exemple d'application de la fonction TSPsolve :

```
set.seed(35)

microbenchmark::microbenchmark(TSPsolve(jeuDeDonnees, method=methods[1]),TSPsolve(jeuDeDonnees, method=m
```

```
## Unit: microseconds
##                                       expr      min        lq      mean
##  TSPsolve(jeuDeDonnees, method = methods[1])  212.748  234.9600  259.8685
##  TSPsolve(jeuDeDonnees, method = methods[2]) 1993.659 2044.4010 2317.2512
##  TSPsolve(jeuDeDonnees, method = methods[3])  207.229  236.5795  257.0792
##  TSPsolve(jeuDeDonnees, method = methods[4])    4.174    5.7935    7.2277
##  TSPsolve(jeuDeDonnees, method = methods[5])  843.035 1338.6720 2233.1786
##     median        uq      max neval cld
##   255.4725  287.5015  307.594    20  a
```

```
##  2206.2185 2588.6390 2765.891    20   b
##   247.6400  276.0350  322.450    20  a
##     6.9540    7.7870   12.776    20  a
##  1977.7395 2621.0605 6101.307    20   b
```

2. Etude de la complexité de l'algorithme Branch and Bound

2.1. Comportement par rapport au nombre de sommets : premier modèle

Récupération du temps sur 10 graphes pour différentes valeurs de n.

Ajustement du modèle linéaire de $\log(temps)^2$ en fonction de n.

Analyse de la validité du modèle :

- pertinence des coefficients et du modèle,
- étude des hypothèses sur les résidus.

Dans un premier temps, nous considérons les graphes générés auparavant, c'est-à-dire,

```
set.seed(35)
sommets <- data.frame(x = runif(n), y = runif(n))
couts <- distance(sommets)
```

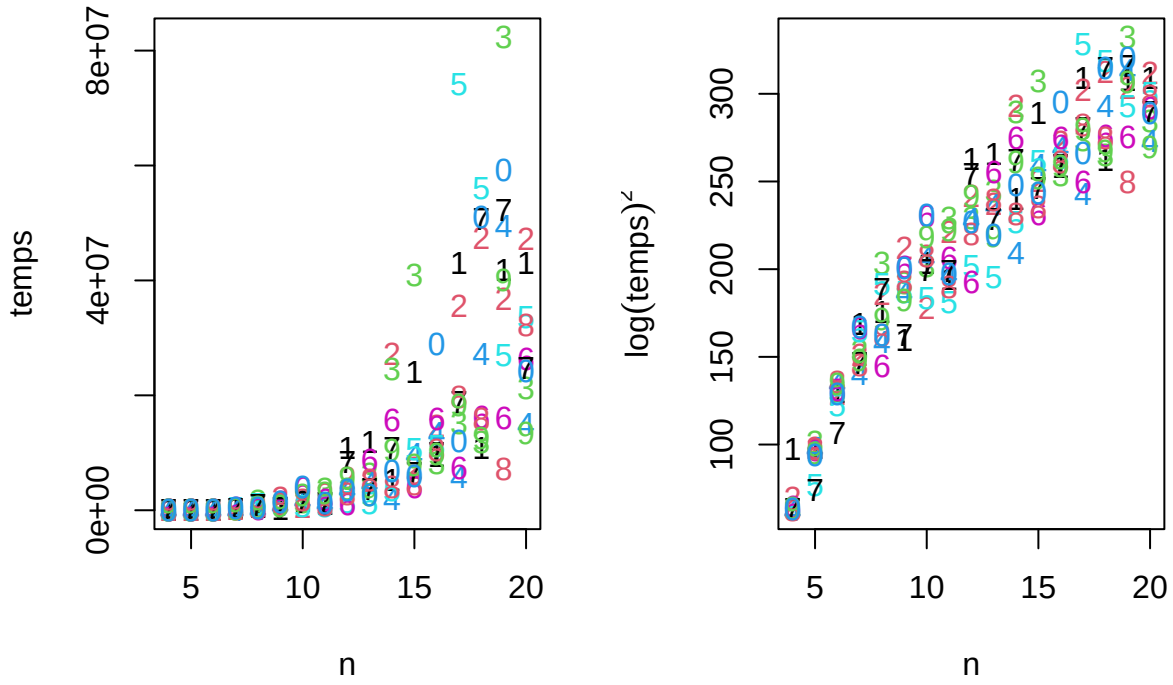
Nous construisons un modèle de régression linéaire simple du temps d'exécution de Branch&Bound en fonction du nombre de sommets n. Introduisons

```
set.seed(35)
seqn <- seq(4,20,1)
```

- Construire la matrice temps telle que la ième ligne soit obtenue par :

```
set.seed(35)
temps  <- matrix(nrow = length(seqn), ncol=10)
for (i in 1:17) {
  temps[i,] =
  microbenchmark(TSPsolve(couts, method = 'branch'),
  times = 10,
  setup = { n <- seqn[i]
  couts <- distance(cbind(x = runif(n), y = runif(n))) }
  )$time
}
```

```
set.seed(35)
par(mfrow=c(1,2)) # 2 graphiques sur 1 ligne
matplot(seqn, temps, xlab='n', ylab='temps')
matplot(seqn, log(temps)^2, xlab='n', ylab=expression(log(temps)^2))
```




```
set.seed(35)
vect_temps <- log(as.vector(temps))^2
vect_dim <- rep(seqn, times=10)
temps.lm <- lm(vect_temps ~ vect_dim)
summary(temps.lm)
```

```
##
## Call:
## lm(formula = vect_temps ~ vect_dim)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -59.08 -16.04  -0.56  17.94  53.13
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  49.5158     4.8961   10.11   <2e-16 ***
## vect_dim     13.6327     0.3777   36.09   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 24.13 on 168 degrees of freedom
## Multiple R-squared:  0.8858, Adjusted R-squared:  0.8851
## F-statistic:  1302 on 1 and 168 DF,  p-value: < 2.2e-16
```

Test de pertinence

Ici, la p-valeur de l'hypothèse a=0 (ligne vect_dim) est extrêmement inférieure à 5%, ainsi, l'hypothèse que a=0 est rejetée avec une grande confiance.

Etude du biais

Ici, la p-valeur de l'hypothèse b=0 (ligne vect_dim) est extrêmement inférieure à 5%, ainsi, l'hypothèse est rejetée avec une grande confiance.

Etude des résidus

Test de normalité

Etude graphique

Tests

Test d'espérance

Etude graphique

Tests

Test de variance

Etude graphique

Tests

Test d'indépendance

Etude graphique

Tests

2.2. Comportement par rapport au nombre de sommets : étude du comportement moyen

Récupération du temps moyen.

Ajustement du modèle linéaire de $\log(temps.moy)^2$ en fonction de n.

Analyse de la validité du modèle :

- pertinence des coefficients et du modèle,
- étude des hypothèses sur les résidus.

2.3. Comportement par rapport à la structure du graphe

Lecture du fichier 'DonneesTSP.csv'.

```
donnees <- read.csv(file='DonneesTSP.csv',header=TRUE)
str(donnees)
```

```
## 'data.frame':    70 obs. of  8 variables:
##  $ tps      : num  53692 144081 997803 2553322 6333009 ...
##  $ dim      : int  4 6 8 10 12 14 16 18 20 4 ...
##  $ mean.long: num  0.391 0.442 0.334 0.276 0.254 ...
##  $ mean.dist: num  0.665 0.592 0.537 0.506 0.502 ...
##  $ sd.dist  : num  0.276 0.259 0.246 0.238 0.227 ...
##  $ mean.deg : num  3 5 7 9 11 13 15 17 19 3 ...
##  $ sd.deg   : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ diameter : num  1 1 1 1 1 1 1 1 1 1 ...
```

Ajustement du modèle linéaire de $\log(temps.moy)^2$ en fonction de toutes les variables présentes. Modèle sans constante.

```
donnees$log.tps <- log(donnees$tps)#log(donnees$tps)^2
donnees$sqrt.dim <- sqrt(donnees$dim)
donnees$tps <- c() #on retire les variables tps et dim devenues inutiles
donnees$dim <- c()
str(donnees)
```

```
## 'data.frame':    70 obs. of  8 variables:
##  $ mean.long: num  0.391 0.442 0.334 0.276 0.254 ...
##  $ mean.dist: num  0.665 0.592 0.537 0.506 0.502 ...
##  $ sd.dist  : num  0.276 0.259 0.246 0.238 0.227 ...
##  $ mean.deg : num  3 5 7 9 11 13 15 17 19 3 ...
##  $ sd.deg   : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ diameter : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ log.tps  : num  10.9 11.9 13.8 14.8 15.7 ...
##  $ sqrt.dim : num  2 2.45 2.83 3.16 3.46 ...
```

Mise en œuvre d'une sélection de variables pour ne garder que les variables pertinentes.

Analyse de la validité du modèle :

- pertinence des coefficients et du modèle,
- étude des hypothèses sur les résidus.