STT3030 Lab 2

2024

Régression linéaire et Classification linéaire

Options par défaut des blocs de code R

Cette option spécifie que le code source R sera affiché dans le document généré.

```
knitr::opts_chunk$set(echo = TRUE)
```

Répertoire de travail

```
getwd()
```

```
## [1] "/Users/agathefernandesmachado/Documents/PhD/STT3030/Labs"
```

```
# A modifier:
#setwd("/Users/agathefernandesmachado/stt3030/lab2")
```

Installation des packages

Une fois installés dans votre environnement, vous n'avez plus besoin d'utiliser ces commandes.

```
#install.packages("MASS")
#install.packages("ISLR")
#install.packages("nnet")
#install.packages("ggplot2")
```

Chargement des packages

```
library(MASS)
library(ISLR)
library(nnet)
library(ggplot2)
```

Visualisation d'un jeu de données

On commence par visualiser les premières lignes du jeu de données Boston, disponible dans la librairie "MASS". La variable sortie Y correspond à "medv" (median house value in Boston for 506 census tracts in Boston). On essaiera de prédire Y à l'aide de plusieurs variables explicatives: "rm" (average number of rooms per house), "age" (average age of houses), et "lstat" (percent of households with low socioeconomic status).

```
?Boston
head(Boston)
```

```
##      crim zn indus chas   nox    rm  age    dis rad tax ptratio  black lstat
## 1 0.00632 18  2.31    0 0.538 6.575 65.2 4.0900   1 296    15.3 396.90  4.98
## 2 0.02731  0  7.07    0 0.469 6.421 78.9 4.9671   2 242    17.8 396.90  9.14
```

```
## 3 0.02729  0  7.07    0 0.469 7.185 61.1 4.9671   2 242    17.8 392.83  4.03
## 4 0.03237  0  2.18    0 0.458 6.998 45.8 6.0622   3 222    18.7 394.63  2.94
## 5 0.06905  0  2.18    0 0.458 7.147 54.2 6.0622   3 222    18.7 396.90  5.33
## 6 0.02985  0  2.18    0 0.458 6.430 58.7 6.0622   3 222    18.7 394.12  5.21
##   medv
## 1 24.0
## 2 21.6
## 3 34.7
## 4 33.4
## 5 36.2
## 6 28.7
```

Type d'objet sous R du jeu de données Boston:

```
class(Boston)
```

```
## [1] "data.frame"
```

Sur R, les matrices "matrix" ou dataframes "data.frame", sont des structures de données bidimensionnelles, c'est-à-dire comportant un nombre de lignes et un nombre de colonnes. Tous les éléments d'une matrice doivent être du même type de données (par exemple, tous numériques, tous caractères, etc.). Au contraire, un objet de type "data.frame" peut contenir des colonnes de types de données différents.

Combien y a-t-il d'observations dans ce jeu de données, noté n ? Et, combien de variables explicatives disposons-nous pour prédire Y ?

```
n <- nrow(Boston)
p <- ncol(Boston)-1 # On supprime la colonne medv, correspondant à Y

cat("Nombre d'observations:", n, "\n")
```

```
## Nombre d'observations: 506
```

```
cat("Nombre de variables explicatives:", p, "\n")
```

```
## Nombre de variables explicatives: 13
```

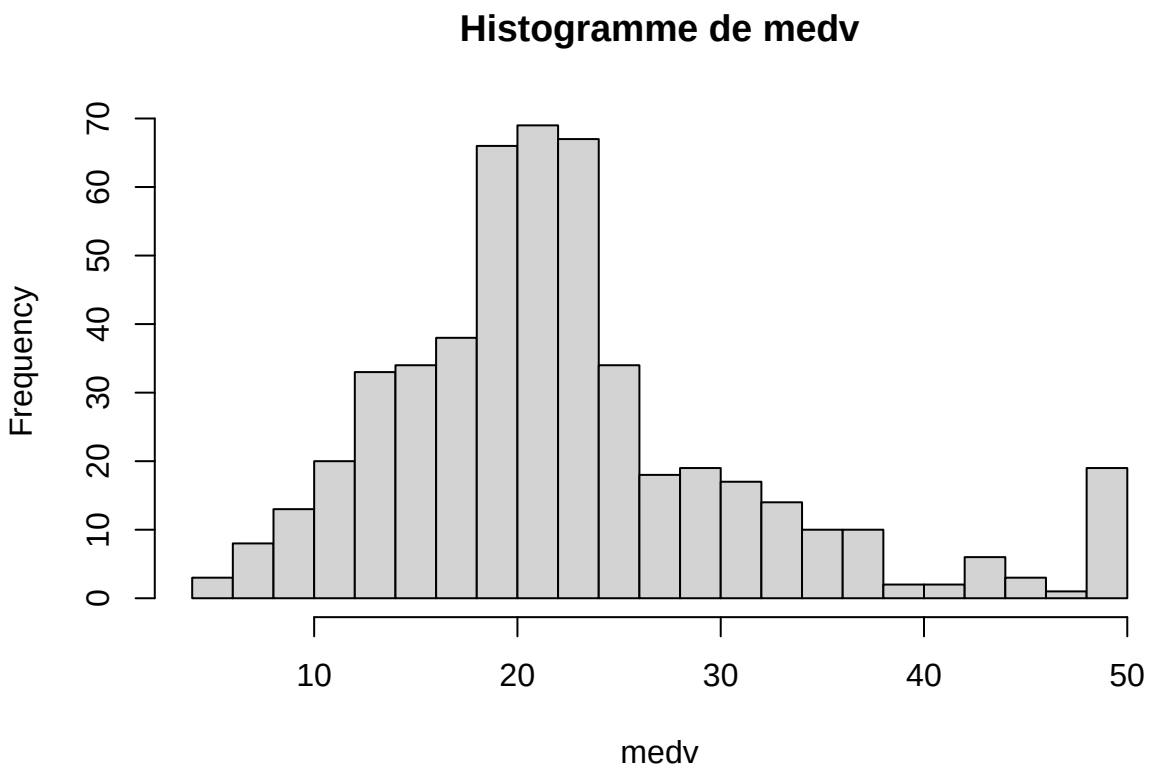
Données manquantes ?

```
na_values <- sum(is.na(Boston))
cat("Nombre d'observations avec des données manquantes:", na_values, "\n")
```

```
## Nombre d'observations avec des données manquantes: 0
```

Histogramme de la variable à prédire medv avec un nombre de "breaks" égal à 20:

```
hist(Boston$medv, breaks = 20, main = "Histogramme de medv", xlab = "medv")
```



A l'exception de medv, y a-t-il dans le jeu de données de Boston une variable quantitative ? Une variable qualitative ? Si oui, quel est son nombre de catégories ?

```
str(Boston) # me donne le type de chaque colonne
```

```
## 'data.frame':    506 obs. of  14 variables:
##  $ crim   : num  0.00632 0.02731 0.02729 0.03237 0.06905 ...
##  $ zn     : num  18 0 0 0 0 0 12.5 12.5 12.5 12.5 ...
##  $ indus  : num  2.31 7.07 7.07 2.18 2.18 2.18 7.87 7.87 7.87 7.87 ...
##  $ chas   : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ nox    : num  0.538 0.469 0.469 0.458 0.458 0.458 0.524 0.524 0.524 0.524 ...
##  $ rm     : num  6.58 6.42 7.18 7 7.15 ...
##  $ age    : num  65.2 78.9 61.1 45.8 54.2 58.7 66.6 96.1 100 85.9 ...
##  $ dis    : num  4.09 4.97 4.97 6.06 6.06 ...
##  $ rad    : int  1 2 2 3 3 3 5 5 5 5 ...
##  $ tax    : num  296 242 242 222 222 222 311 311 311 311 ...
##  $ ptratio: num  15.3 17.8 17.8 18.7 18.7 18.7 15.2 15.2 15.2 15.2 ...
##  $ black  : num  397 397 393 395 397 ...
##  $ lstat  : num  4.98 9.14 4.03 2.94 5.33 ...
##  $ medv   : num  24 21.6 34.7 33.4 36.2 28.7 22.9 27.1 16.5 18.9 ...
```

```
?Boston # me donne les informations sur les variables
```

Il n'y a que deux variables qualitatives: "chas" (variable binaire) et "rad" (variable multi-classes):

```
unique(Boston$chas)
```

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

```
k1 <- length(unique(Boston$chas))
cat("Nombre de catégories de la variable chas:", k1, "\n")
```

```
## Nombre de catégories de la variable chas: 2
```

```r
unique(Boston$rad)
```

```
## [1]  1  2  3  5  4  8  6  7 24
```

```r
k2 <- length(unique(Boston$rad))
cat("Nombre de catégories de la variable rad:", k2, "\n")
```

```
## Nombre de catégories de la variable rad: 9
```

Régression linéaire sur "medv"

Séparez aléatoirement le jeu de données Boston en un échantillon d'entraînement (80%) et un échantillon de test (20%) à l'aide de la fonction "sample". Pour cela dans un premier temps, on fixe le "seed" afin d'être en mesure de retrouver nos résultats si on relance le script R plusieurs fois.

```r
set.seed(2024)

id <- sample(nrow(Boston)) # mélange des numéros de lignes de Boston
prop <- 0.8
prop*nrow(Boston)
```

```
## [1] 404.8
```

```r
n_a <- round(prop*nrow(Boston)) # on utilise round pour avoir un nombre entier
n_a
```

```
## [1] 405
```

```r
Boston_a <- Boston[id[1:n_a],] # échantillon d'entraînement
Boston_t <- Boston[id[(n_a+1):nrow(Boston)],] # échantillon de test
```

Vérifiez que vous obtenez les bonnes proportions, 80%/20%, sur les échantillons d'entraînement et de test:

```r
cat("Proportion de données d'entraînement:", round(nrow(Boston_a)/nrow(Boston)*100), "%", "\n")
```

```
## Proportion de données d'entraînement: 80 %
```

```r
cat("Proportion de données de test:", round(nrow(Boston_t)/nrow(Boston)*100), "%", "\n")
```

```
## Proportion de données de test: 20 %
```

Définissez une fonction permettant de calculer l'EQM prenant en argument deux vecteurs de données: "y" pour les valeurs observées de Y et "preds" pour prédictions.

```r
calcul_EQM <- function(y, preds){
  eqm <- sum((preds - y)^2)/length(y)
  eqm
  # ou
  # return(eqm) # comme en Python
}
```

Par la suite, on calculera l'EQM sur les données de test et d'entraînement pour chaque modèle de régression.

Régression linéaire simple avec lsat

Entraînez un modèle de régression linéaire simple sur les données d'entraînement en prenant comme seule variable explicative "lstat" (variable numérique). Affichez le résumé du modèle.

```r
fit_1 <- lm(medv ~ lstat, data = Boston_a)
summary(fit_1)
```

```
##
## Call:
## lm(formula = medv ~ lstat, data = Boston_a)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -15.091  -4.024  -1.493   1.997  24.603
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 34.60321    0.63317   54.65   <2e-16 ***
## lstat       -0.96605    0.04378  -22.07   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.258 on 403 degrees of freedom
## Multiple R-squared:  0.5471, Adjusted R-squared:  0.546
## F-statistic: 486.9 on 1 and 403 DF,  p-value: < 2.2e-16
```

Affichez les deux coefficients du modèle linéaire ainsi qu'un intervalle de confiance au niveau de confiance 95% sur chacun des coefficients:

```
fit_1$coefficients
```

```
## (Intercept)       lstat
##  34.6032128  -0.9660521
```

```
confint(fit_1)
```

```
##                 2.5 %     97.5 %
## (Intercept) 33.358485 35.8479402
## lstat       -1.052122 -0.8799819
```

Prédire le modèle sur les données de test:

1) Calculez les prédictions à la main puis affichez les 5 premières prédictions:

```
b0 <- fit_1$coefficients[1]
b1 <- fit_1$coefficients[2]
preds_1 <- b0 + b1 * Boston_t$lstat
preds_1[1:5]
```

```
## [1] 31.57947 28.26591 18.86622 21.07848 28.11134
```

2) Calculez les prédictions à l'aide de la fonction "predict" et comparez les valeurs avec 1):

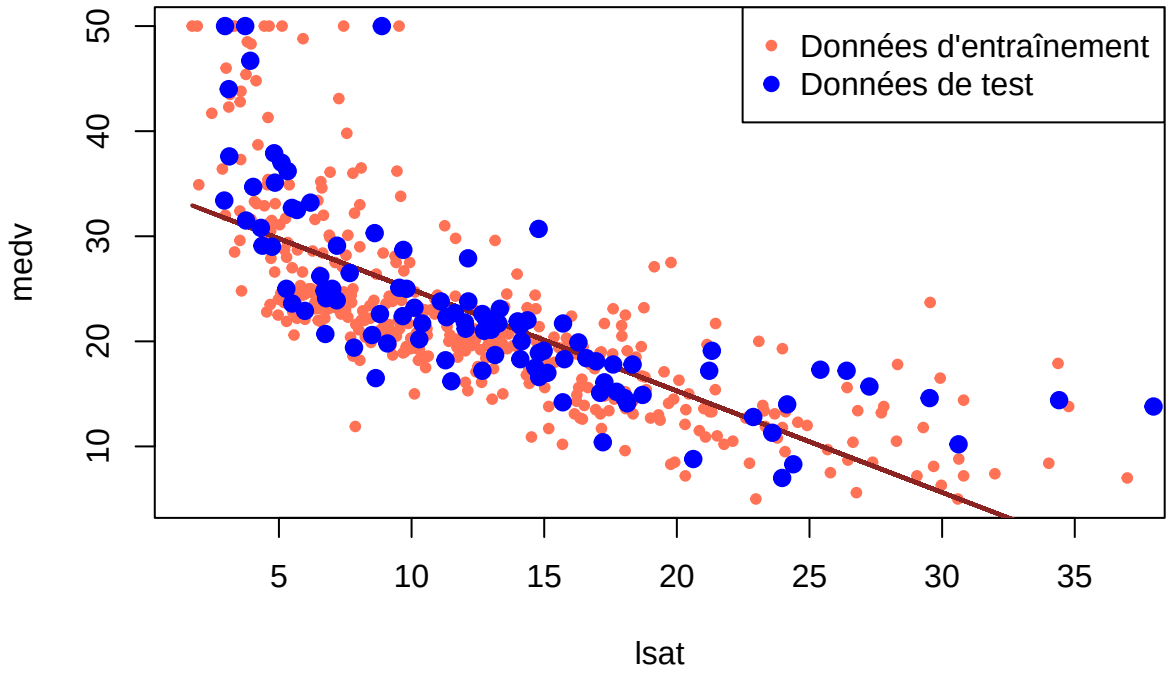
```
preds_1 <- predict(fit_1, newdata = Boston_t)
preds_1[1:5]
```

```
##      227      250      471      367       83
## 31.57947 28.26591 18.86622 21.07848 28.11134
```

On peut désormais afficher le graphe de la droite des moindres carrés ainsi que les coordonnées des points contenus dans les échantillons d'apprentissage et de test. Ajoutez une légende à ce graphique afin d'indiquer le nom des échantillons et les couleurs associés.

```
plot(Boston_a$lstat, Boston_a$medv, pch = 20, col = "coral1",
     xlab = "lsat", ylab = "medv")
lines(Boston$lstat, b0 + b1*Boston$lstat, col = "brown4", lwd = 2)
points(Boston_t$lstat, Boston_t$medv, col = "blue", pch = 19, lwd = 2)
```

```
legend("topright", legend = c("Données d'entraînement", "Données de test"),
       col = c("coral1", "blue"), pch = c(20, 19))
```



Calculez l'EQM sur les données d'entraînement et les données de test:

```
cat("EQM pour les données d'entraînement:",
    round(calcul_EQM(Boston_a$medv, fit_1$fitted.values), 2), "\n")
```

```
## EQM pour les données d'entraînement: 38.97
```

```
# ou
# cat("EQM pour les données d'entraînement:",
#    calcul_EQM(Boston_a$medv, predict(fit_1, newdata = Boston_a)), "\n")

cat("EQM pour les données de test:",
    round(calcul_EQM(Boston_t$medv, preds_1), 2), "\n")
```

```
## EQM pour les données de test: 36.7
```

On peut également calculer des intervalles de prédiction au niveau de confiance 95% pour les prédictions sur les données de l'échantillon de test. Affichez ces intervalles pour les 5 premières observations.

```
pred_int <- predict(fit_1, Boston_t, interval ="prediction")
pred_int[1:5,]
```

```
##          fit       lwr      upr
## 227 31.57947 19.234240 43.92470
## 250 28.26591 15.936648 40.59517
## 471 18.86622  6.543816 31.18863
## 367 21.07848  8.759584 33.39738
## 83  28.11134 15.782652 40.44003
```

Ajout de la variable qualitative "chas"

Entraînez un modèle de régression linéaire avec les variables "lstat" et "chas" sur les données d'entraînement, sans intéractions. Affichez le résumé du modèle.

On commence par indiquer à R que la variable "chas" est catégorielle:

```
Boston$chas <- as.factor(Boston$chas)
Boston_a$chas <- as.factor(Boston_a$chas)
Boston_t$chas <- as.factor(Boston_t$chas)
```

Quelles sont les deux modalités de cette variable ?

```
unique(Boston$chas)
```

```
## [1] 0 1
## Levels: 0 1
```

On peut désormais apprendre le modèle linéaire en spécifiant les variables à utiliser:

```
fit_2 <- lm(medv ~ lstat + chas, data = Boston_a)
summary(fit_2)
```

```
##
## Call:
## lm(formula = medv ~ lstat + chas, data = Boston_a)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -14.716  -3.817  -1.322   1.682  24.963
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 34.15646    0.63264   53.99  < 2e-16 ***
## lstat       -0.95697    0.04309  -22.21  < 2e-16 ***
## chas1        4.64080    1.18704    3.91 0.000109 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.15 on 402 degrees of freedom
## Multiple R-squared:  0.5637, Adjusted R-squared:  0.5615
## F-statistic: 259.7 on 2 and 402 DF,  p-value: < 2.2e-16
```

Quelle est la modalité de référence de la variable chas dans ce modèle ? Réponse: 0

La modalité 0 est la "baseline" pour la variable "chas" car il s'agit de la première valeur affichée comme "levels":

```
levels(Boston$chas)
```

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

Il est possible de changer cette modalité de référence en modifiant l'ordre de ces "levels". Cela changera ainsi la modalité de référence dans le modèle linéaire.

```
chas_modif <- factor(Boston$chas, levels = c("1", "0"))
levels(chas_modif)
```

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

Prédire le modèle sur les données de test:

1) Calculez les prédictions à la main puis affichez les 5 premières prédictions:

```
b0 <- fit_2$coefficients[1]
b1 <- fit_2$coefficients[2]
a1 <- fit_2$coefficients[3]
# Le modèle dépend désormais de la modalité de la variable chas
preds_2 <- b0 + b1 *Boston_t$lstat + a1 * as.numeric(Boston_t$chas == "1")
preds_2[1:5]
```

```
## [1] 31.16115 27.87876 18.56747 20.75892 27.72564
```

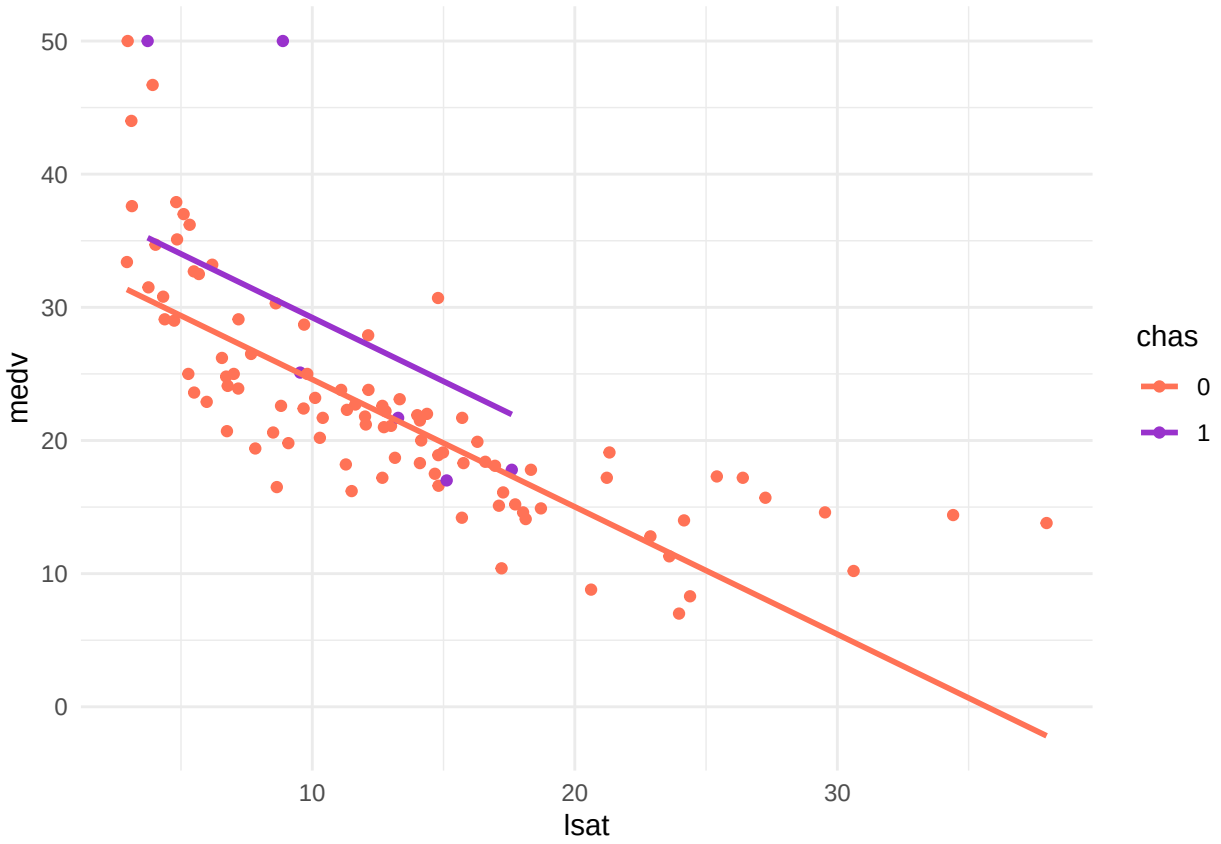
2) Calculez les prédictions à l'aide de la fonction "predict" et comparez les valeurs avec 1):

```
preds_2 <- predict(fit_2, newdata = Boston_t)
preds_2[1:5]
```

```
##      227      250      471      367       83
## 31.16115 27.87876 18.56747 20.75892 27.72564
```

On peut désormais afficher le modèle linéaire appris (i.e. les deux droites obtenues) ainsi que les données de test ("medv" en fonction de "lstat"), en différenciant par couleur sur "chas".

```
df <- data.frame(medv = Boston_t$medv, lsat = Boston_t$lstat,
                 chas = Boston_t$chas, medv_pred = preds_2)
ggplot(df, aes(x = lsat, y = medv, color = chas)) +
  geom_point() +
  geom_line(aes(y = medv_pred), linewidth = 1) +
  scale_color_manual(values = c("0" = "coral1", "1" = "darkorchid")) +
  labs(x = "lsat",
       y = "medv") +
  theme_minimal()
```



Calculez l'EQM sur les données d'entraînement et les données de test:

```
cat("EQM pour les données d'entraînement:",
    round(calcul_EQM(Boston_a$medv, fit_2$fitted.values), 2), "\n")
```

```
## EQM pour les données d'entraînement: 37.55
```

```
cat("EQM pour les données de test:",
    round(calcul_EQM(Boston_t$medv, preds_2), 2), "\n")
```

```
## EQM pour les données de test: 34.68
```

Ajout d'un terme d'intéraction entre "lsat" et "chas"

Entraînez un modèle de régression linéaire avec les variables "lstat" et "chas" sur les données d'entraînement, avec intéraction. Affichez le résumé du modèle.

```
fit_3 <- lm(medv ~ lstat * chas, data = Boston_a)
summary(fit_3)
```

```
##
## Call:
## lm(formula = medv ~ lstat * chas, data = Boston_a)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -14.619  -3.794  -1.345   1.595  25.027
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
```

```
## (Intercept) 33.90158    0.64825  52.297  < 2e-16 ***
## lstat       -0.93690    0.04454 -21.037  < 2e-16 ***
## chas1        7.95605    2.26104   3.519 0.000483 ***
## lstat:chas1 -0.29287    0.17016  -1.721 0.085991 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.135 on 401 degrees of freedom
## Multiple R-squared:  0.5669, Adjusted R-squared:  0.5637
## F-statistic:   175 on 3 and 401 DF,  p-value: < 2.2e-16
```

Prédire le modèle sur les données de test:

1) Calculez les prédictions à la main puis affichez les 5 premières prédictions:

```
b0 <- fit_3$coefficients[1]
b1 <- fit_3$coefficients[2]
a1 <- fit_3$coefficients[3]
gamma <- fit_3$coefficients[4]
# Le modèle dépend de la modalité de la variable chas
preds_3 <- b0 + b1 * Boston_t$lstat + a1 * as.numeric(Boston_t$chas == "1") +
  gamma * Boston_t$lstat * as.numeric(Boston_t$chas == "1")
preds_3[1:5]
```

```
## [1] 30.96907 27.75549 18.63942 20.78493 27.60559
```

2) Calculez les prédictions à l'aide de la fonction "predict" et comparez les valeurs avec 1):

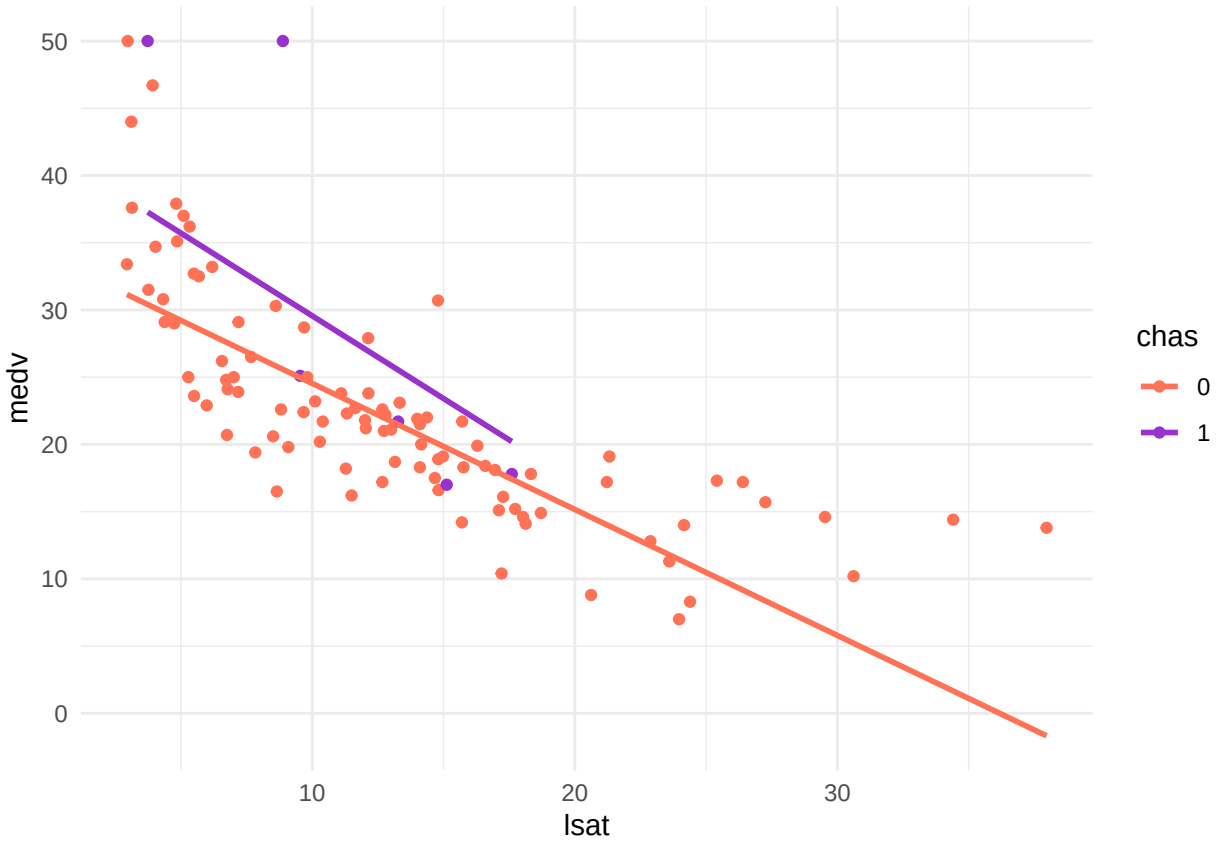
```
preds_3 <- predict(fit_3, newdata = Boston_t)
preds_3[1:5]
```

```
##      227      250      471      367       83
## 30.96907 27.75549 18.63942 20.78493 27.60559
```

On peut désormais afficher le modèle linéaire appris (i.e. les deux droites obtenues) ainsi que les données de test ("medv" en fonction de "lstat"), en différenciant par couleur sur "chas".

```
df <- data.frame(medv = Boston_t$medv, lsat = Boston_t$lstat,
                 chas = Boston_t$chas, medv_pred = preds_3)
ggplot(df, aes(x = lsat, y = medv, color = chas)) +
  geom_point() +
  geom_line(aes(y = medv_pred), size = 1) +
  scale_color_manual(values = c("0" = "coral1", "1" = "darkorchid")) +
  labs(x = "lsat",
       y = "medv") +
  theme_minimal()
```

```
## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
## i Please use `linewidth` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```



En ajoutant un terme d'intéraction, on modifie la pente de la droite des moindres carrés initiale dans le groupe "chas" = 1.

Calculez l'EQM sur les données d'entraînement et les données de test:

```
cat("EQM pour les données d'entraînement:",
    round(calcul_EQM(Boston_a$medv, fit_3$fitted.values), 2), "\n")
```

```
## EQM pour les données d'entraînement: 37.27
```

```
cat("EQM pour les données de test:",
    round(calcul_EQM(Boston_t$medv, preds_3), 2), "\n")
```

```
## EQM pour les données de test: 33.39
```

On peut se demander si l'ajout de cette intéraction est importante pour prédire "medv" à l'aide du test anova.

```
anova(fit_2, fit_3)
```

```
## Analysis of Variance Table
##
## Model 1: medv ~ lstat + chas
## Model 2: medv ~ lstat * chas
##   Res.Df   RSS Df Sum of Sq      F  Pr(>F)
## 1    402 15206
## 2    401 15095  1    111.51 2.9624 0.08599 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

L'hypothèse nulle est que les deux modèles (fit_2 et fit_3) s'ajustent aux données de manière équivalente, et l'hypothèse alternative est que le modèle complet (fit_3) est supérieur. Ici, la statistique F est de 2.96 et la

p-value associée est supérieure à 0.05. Ainsi, on ne peut pas rejeter l'hypothèse nulle. Il ne semble donc pas utile d'inclure le terme d'intéraction entre "lstat" et "chas" dans le modèle afin de prédire au mieux "medv".

Régression linéaire multiple

```
fit_4 <- lm(medv ~ ., data = Boston_a)
summary(fit_4)
```

```
##
## Call:
## lm(formula = medv ~ ., data = Boston_a)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -15.0569  -2.6464  -0.6459   1.9598  28.3416
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  32.234386   5.720415   5.635 3.35e-08 ***
## crim         -0.081478   0.045410  -1.794 0.073546 .
## zn            0.028543   0.015342   1.860 0.063581 .
## indus         0.029579   0.071611   0.413 0.679789
## chas1         1.710132   0.941614   1.816 0.070110 .
## nox         -16.042411   4.247027  -3.777 0.000183 ***
## rm            4.392700   0.457977   9.592  < 2e-16 ***
## age          -0.002386   0.014336  -0.166 0.867894
## dis          -1.244867   0.216666  -5.746 1.84e-08 ***
## rad           0.230352   0.076879   2.996 0.002907 **
## tax          -0.010357   0.004390  -2.360 0.018787 *
## ptratio      -1.040653   0.145206  -7.167 3.86e-12 ***
## black         0.009259   0.002954   3.134 0.001855 **
## lstat        -0.493835   0.056737  -8.704  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.68 on 391 degrees of freedom
## Multiple R-squared:  0.7543, Adjusted R-squared:  0.7462
## F-statistic: 92.35 on 13 and 391 DF,  p-value: < 2.2e-16
```

En vue des p-values, quelles variables vous semblent le plus significatives ?

```
names(summary(fit_4)$coefficients[which(summary(fit_4)$coefficients[,4] <= 0.05), 4])[-1]
```

```
## [1] "nox"     "rm"      "dis"     "rad"     "tax"     "ptratio" "black"
## [8] "lstat"
```

Prédire le modèle sur les données de test à l'aide de la fonction "predict":

```
preds_4 <- predict(fit_4, newdata = Boston_t)
```

Calculez l'EQM sur les données d'entraînement et les données de test:

```
cat("EQM pour les données d'entraînement:",
    round(calcul_EQM(Boston_a$medv, fit_4$fitted.values), 2), "\n")
```

```
## EQM pour les données d'entraînement: 21.14
```

```r
cat("EQM pour les données de test:",
    round(calcul_EQM(Boston_t$medv, preds_4), 2), "\n")
```

```
## EQM pour les données de test: 26.75
```

Régression linéaire multiple avec intéractions

```r
fit_5 <- lm(medv ~ .^2, data = Boston_a) # Pour ajouter les intéractions de degré 2
summary(fit_5)
```

```
##
## Call:
## lm(formula = medv ~ .^2, data = Boston_a)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -9.9274 -1.5808 -0.1131  1.3938 17.3508
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)   -1.775e+02  8.351e+01  -2.125 0.034338 *
## crim          -1.924e+01  9.301e+00  -2.068 0.039430 *
## zn             2.607e-02  5.949e-01   0.044 0.965081
## indus         -3.299e+00  2.010e+00  -1.641 0.101756
## chas1          2.881e+01  2.430e+01   1.186 0.236678
## nox            5.576e+01  9.209e+01   0.606 0.545241
## rm             2.347e+01  6.809e+00   3.448 0.000643 ***
## age            1.150e+00  3.270e-01   3.518 0.000500 ***
## dis           -3.161e+00  5.575e+00  -0.567 0.571135
## rad           -5.185e-02  3.093e+00  -0.017 0.986634
## tax            1.131e-01  1.780e-01   0.635 0.525805
## ptratio        2.968e+00  3.584e+00   0.828 0.408211
## black          1.634e-01  9.399e-02   1.739 0.083092 .
## lstat          1.770e+00  1.053e+00   1.681 0.093769 .
## crim:zn        2.955e-01  2.341e-01   1.262 0.207811
## crim:indus    -6.245e-01  7.461e-01  -0.837 0.403219
## crim:chas1     3.371e+00  1.090e+00   3.092 0.002167 **
## crim:nox      -2.459e+00  1.116e+00  -2.203 0.028325 *
## crim:rm        2.983e-01  8.704e-02   3.427 0.000691 ***
## crim:age      -7.530e-03  4.866e-03  -1.547 0.122780
## crim:dis      -2.765e-01  1.359e-01  -2.034 0.042777 *
## crim:rad      -1.274e+00  9.546e-01  -1.334 0.183061
## crim:tax       8.416e-02  7.086e-02   1.188 0.235868
## crim:ptratio   2.604e-01  5.046e-01   0.516 0.606192
## crim:black    -5.003e-06  2.758e-04  -0.018 0.985536
## crim:lstat     3.296e-02  7.357e-03   4.480 1.05e-05 ***
## zn:indus      -3.792e-04  5.433e-03  -0.070 0.944406
## zn:chas1      -1.064e-01  7.265e-02  -1.465 0.143963
## zn:nox         2.024e-01  5.892e-01   0.343 0.731508
## zn:rm          1.931e-03  3.089e-02   0.063 0.950177
## zn:age        -3.519e-04  1.055e-03  -0.334 0.738956
## zn:dis         1.172e-02  8.736e-03   1.342 0.180688
## zn:rad        -1.657e-03  8.919e-03  -0.186 0.852704
## zn:tax         3.889e-04  2.301e-04   1.690 0.091967 .
```

```
## zn:ptratio    -3.108e-03  8.299e-03  -0.374 0.708291
## zn:black      -4.610e-04  1.006e-03  -0.458 0.647004
## zn:lstat      -1.083e-02  6.114e-03  -1.772 0.077396 .
## indus:chas1   -5.323e-01  4.247e-01  -1.253 0.210967
## indus:nox      4.220e+00  1.868e+00   2.259 0.024556 *
## indus:rm       2.538e-01  1.664e-01   1.525 0.128240
## indus:age      3.418e-04  4.668e-03   0.073 0.941675
## indus:dis     -2.864e-02  7.517e-02  -0.381 0.703473
## indus:rad     -5.154e-03  6.290e-02  -0.082 0.934750
## indus:tax      5.869e-04  7.249e-04   0.810 0.418728
## indus:ptratio -5.156e-02  5.014e-02  -1.028 0.304628
## indus:black    1.816e-03  2.375e-03   0.765 0.444932
## indus:lstat   -1.401e-02  2.362e-02  -0.593 0.553371
## chas1:nox     -2.431e+01  1.520e+01  -1.600 0.110601
## chas1:rm      -3.898e+00  1.322e+00  -2.948 0.003435 **
## chas1:age      2.650e-02  6.421e-02   0.413 0.680152
## chas1:dis      2.530e+00  1.598e+00   1.583 0.114402
## chas1:rad     -1.046e+00  6.319e-01  -1.655 0.098874 .
## chas1:tax      4.530e-02  4.030e-02   1.124 0.261814
## chas1:ptratio -5.097e-01  7.651e-01  -0.666 0.505805
## chas1:black    1.516e-02  1.791e-02   0.847 0.397799
## chas1:lstat   -1.433e-01  2.064e-01  -0.695 0.487808
## nox:rm         2.100e+00  6.382e+00   0.329 0.742316
## nox:age       -7.136e-01  2.713e-01  -2.630 0.008949 **
## nox:dis        5.215e+00  4.485e+00   1.163 0.245886
## nox:rad        1.785e+00  2.509e+00   0.711 0.477319
## nox:tax       -1.507e-01  1.766e-01  -0.853 0.394279
## nox:ptratio   -2.962e+00  3.729e+00  -0.794 0.427732
## nox:black     -4.420e-02  4.598e-02  -0.961 0.337179
## nox:lstat      1.707e+00  7.845e-01   2.176 0.030327 *
## rm:age        -3.963e-02  2.670e-02  -1.484 0.138724
## rm:dis         2.955e-01  3.771e-01   0.784 0.433899
## rm:rad        -9.623e-02  1.811e-01  -0.532 0.595433
## rm:tax        -1.938e-02  1.197e-02  -1.620 0.106291
## rm:ptratio    -4.342e-01  2.490e-01  -1.744 0.082179 .
## rm:black      -1.304e-03  4.995e-03  -0.261 0.794198
## rm:lstat      -3.464e-01  5.866e-02  -5.906 9.15e-09 ***
## age:dis       -1.900e-02  1.089e-02  -1.745 0.081979 .
## age:rad        1.474e-02  5.391e-03   2.733 0.006625 **
## age:tax       -2.244e-04  2.686e-04  -0.835 0.404220
## age:ptratio   -1.078e-02  8.087e-03  -1.333 0.183657
## age:black     -7.125e-04  2.555e-04  -2.789 0.005612 **
## age:lstat     -6.455e-03  2.275e-03  -2.837 0.004845 **
## dis:rad        1.258e-02  8.864e-02   0.142 0.887221
## dis:tax       -4.256e-03  3.212e-03  -1.325 0.186123
## dis:ptratio   -5.239e-02  1.122e-01  -0.467 0.640843
## dis:black     -7.535e-04  7.390e-03  -0.102 0.918853
## dis:lstat      1.414e-01  5.974e-02   2.366 0.018569 *
## rad:tax       -7.963e-04  1.814e-03  -0.439 0.660952
## rad:ptratio   -2.357e-02  1.069e-01  -0.220 0.825669
## rad:black      1.007e-03  3.087e-03   0.326 0.744405
## rad:lstat     -2.730e-02  2.414e-02  -1.131 0.258908
## tax:ptratio    7.758e-03  3.276e-03   2.368 0.018474 *
## tax:black     -1.093e-04  2.407e-04  -0.454 0.650042
```

```
## tax:lstat     -1.299e-03  1.701e-03  -0.763 0.445760
## ptratio:black -1.291e-03  4.260e-03  -0.303 0.761978
## ptratio:lstat  1.094e-02  3.696e-02   0.296 0.767342
## black:lstat   -8.656e-04  5.190e-04  -1.668 0.096346 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.936 on 313 degrees of freedom
## Multiple R-squared:  0.9226, Adjusted R-squared:  0.9001
## F-statistic: 40.98 on 91 and 313 DF,  p-value: < 2.2e-16
```

Combien le modèle a-t-il appris de paramètres ?

```
length(fit_5$coefficients)
```

```
## [1] 92
```

Il pourrait être utile de faire de la sélection de variables ici.

Corrélation entre variables explicatives

On va se concentrer désormais sur le modèle contenant les variables explicatives “nox”, “age” et “ptratio”, afin de prédire “mdev”.

Calculez la corrélation de Pearson (linéaire) entre toutes les combinaisons de deux variables:

```
cor(Boston$age, Boston$nox, method = "pearson")
```

```
## [1] 0.7314701
```

```
cor(Boston$age, Boston$ptratio, method = "pearson")
```

```
## [1] 0.261515
```

```
cor(Boston$nox, Boston$ptratio, method = "pearson")
```

```
## [1] 0.1889327
```

Construisez le modèle linéaire avec ces trois variables et observez la significativité:

```
fit_6 <- lm(medv ~ ptratio + nox + age, data = Boston_a)
summary(fit_6)
```

```
##
## Call:
## lm(formula = medv ~ ptratio + nox + age, data = Boston_a)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -14.646  -4.442  -1.282   2.916  33.336
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  74.75336    3.46748  21.558  < 2e-16 ***
## ptratio      -2.02230    0.17115 -11.816  < 2e-16 ***
## nox         -26.58462    4.48430  -5.928  6.6e-09 ***
## age          -0.00464    0.01870  -0.248    0.804
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

```
## Residual standard error: 7.241 on 401 degrees of freedom
## Multiple R-squared:  0.3967, Adjusted R-squared:  0.3922
## F-statistic:  87.9 on 3 and 401 DF,  p-value: < 2.2e-16
```

La variable "age" ne semble pas significative dans ce modèle.

Supprimons la variable "nox" du modèle linéaire et observons une nouvelle fois la significativité:

```
fit_7 <- lm(medv ~ ptratio + age, data = Boston_a)
summary(fit_7)
```

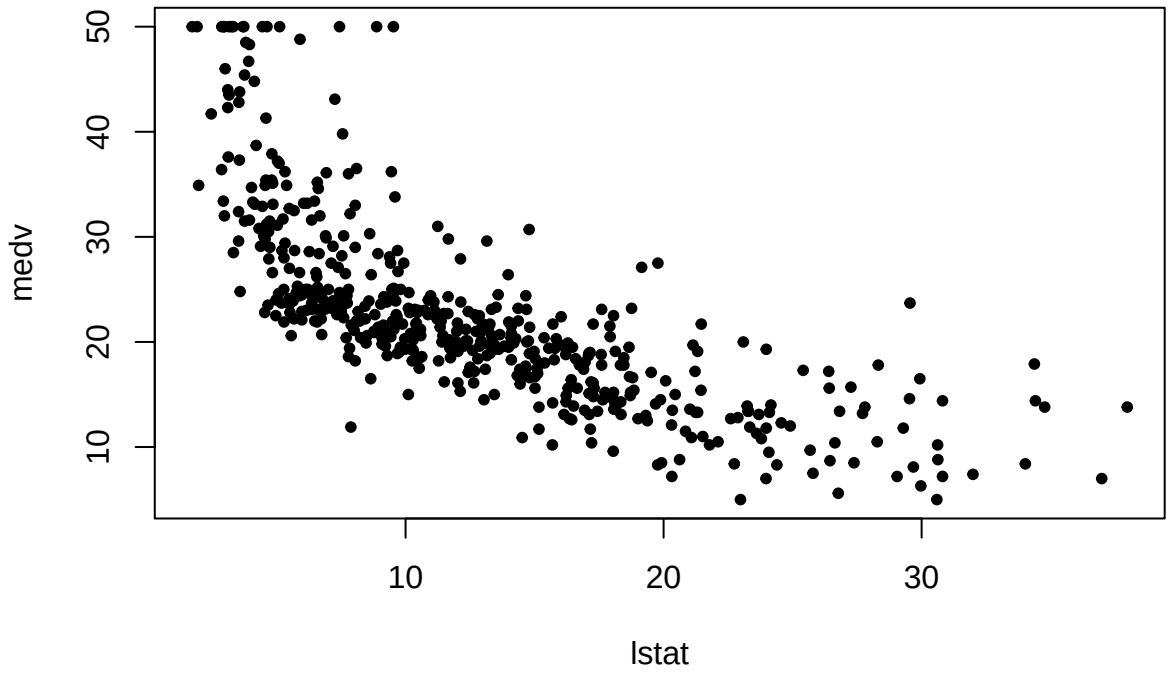
```
##
## Call:
## lm(formula = medv ~ ptratio + age, data = Boston_a)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -15.767  -4.103  -1.131   3.175  33.744
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 65.47829    3.22320  20.315  < 2e-16 ***
## ptratio     -2.02317    0.17827 -11.349  < 2e-16 ***
## age         -0.08354    0.01369  -6.104 2.44e-09 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 7.542 on 402 degrees of freedom
## Multiple R-squared:  0.3438, Adjusted R-squared:  0.3406
## F-statistic: 105.3 on 2 and 402 DF,  p-value: < 2.2e-16
```

Cette fois-ci, la variable "age" est significative. Ainsi, pour la prédiction, les variables inutiles/corrélées sont nuisibles. Dans ce cas, il peut être utile de choisir une seule variable explicative parmi les deux corrélées. La sélection de variables sera au coeur du prochain cours.

Régression polynomiale

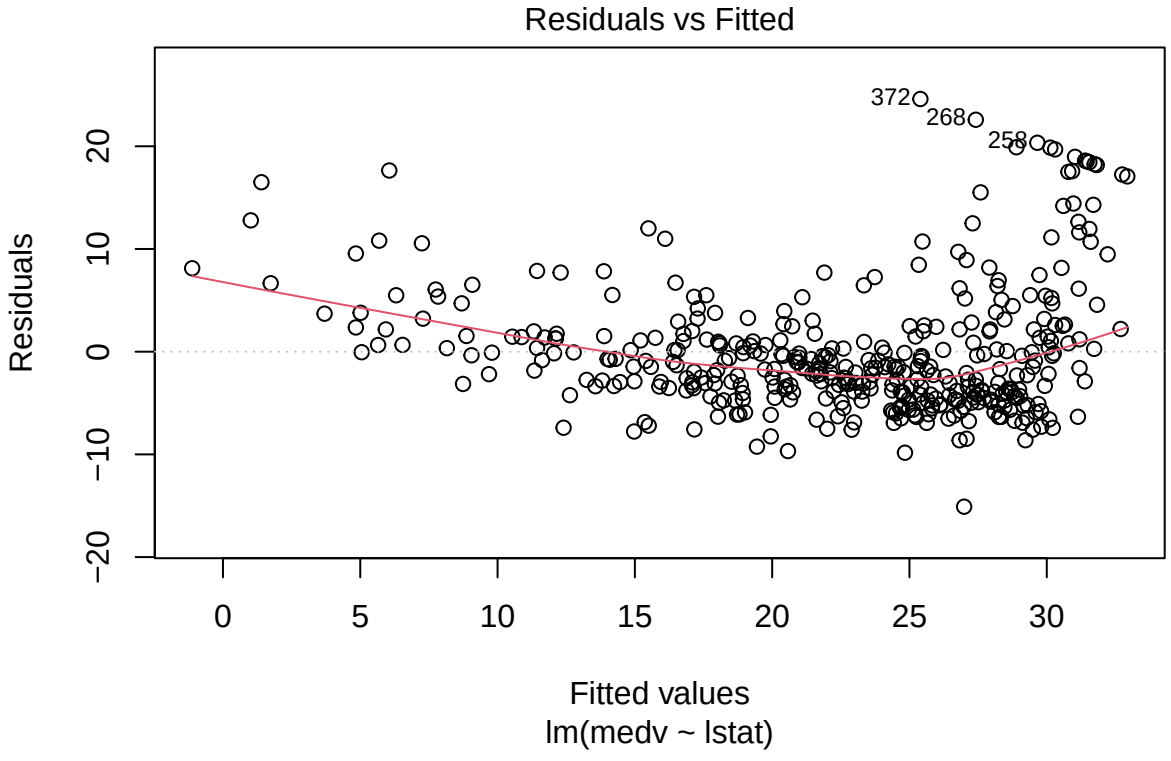
Etudions la relation entre les variables "lstat" et "medv". On commence par afficher le graphique représentant la variable "medv" en fonction de la variable "lstat" pour toutes les observations du jeu de données Boston.

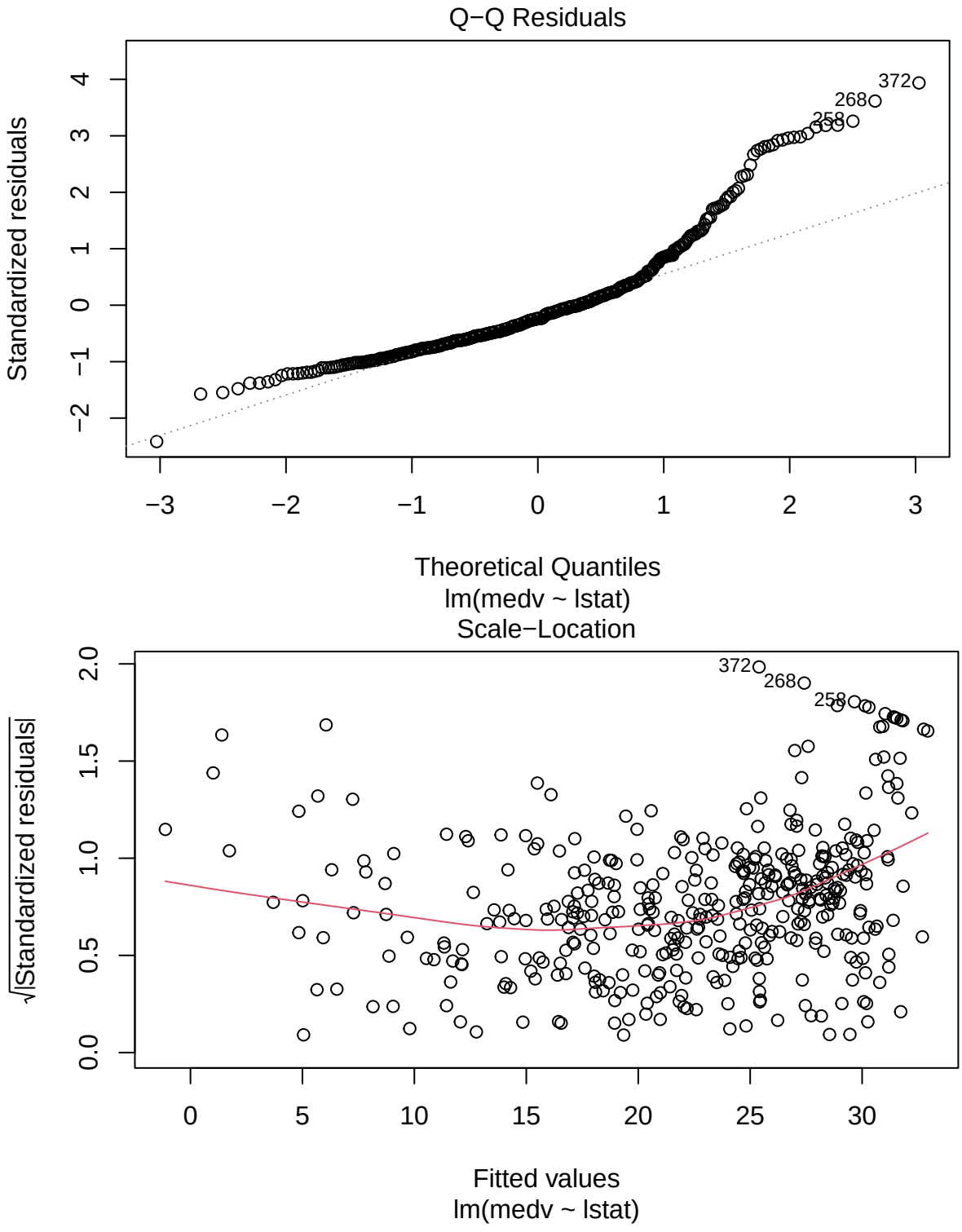
```
plot(Boston$lstat, Boston$medv, pch = 20, xlab = "lstat", ylab = "medv")
```

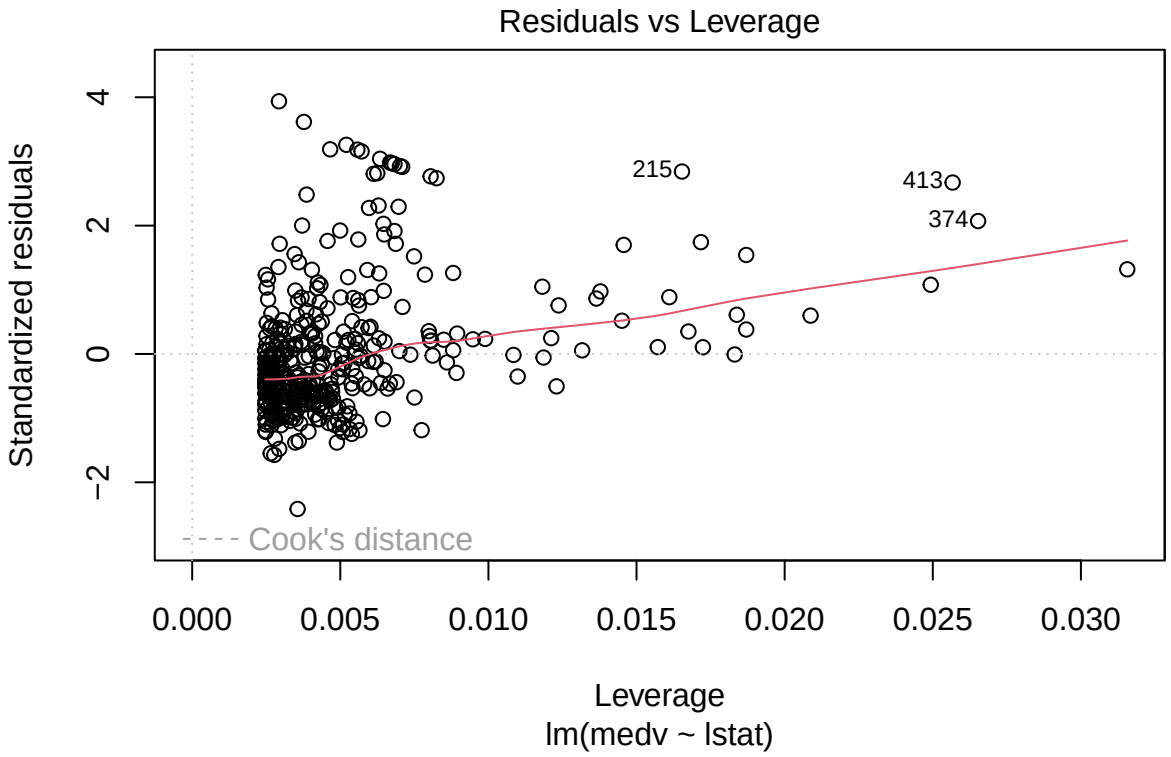


Il semble que la relation ne soit pas linéaire entre les valeurs 0 et 10 de la variable “lstat”. On peut également l’observer à l’aide du graphique “Residuals vs Fitted” du modèle linéaire entre “lstat” et “medv”:

```
plot(fit_1)
```




Q−Q Residuals
Standardized residuals
372
268
258
−3
−2
−1
0
1
2
3
4
Theoretical Quantiles
lm(medv ~ lstat)
Scale−Location
√|Standardized residuals|
372
268
258
0.0
0.5
1.0
1.5
2.0
0
5
10
15
20
25
30
Fitted values
lm(medv ~ lstat)



Ainsi, incluons un degré de plus pour la variable “lstat”:

```
fit_8 <- lm(medv ~ lstat+I(lstat^2), data = Boston_a) # Polynôme de degré 2 pour lstat
# ou
# fit_8 <- lm(medv ~ poly(lstat, 2, raw=TRUE), data = Boston_a)
summary(fit_8)
```

```
##
## Call:
## lm(formula = medv ~ lstat + I(lstat^2), data = Boston_a)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -15.1501  -3.9843  -0.6949   2.3874  25.5603
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 42.809252   1.004151  42.632   <2e-16 ***
## lstat       -2.345405   0.144597 -16.220   <2e-16 ***
## I(lstat^2)   0.043846   0.004423   9.912   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.617 on 402 degrees of freedom
## Multiple R-squared:  0.6361, Adjusted R-squared:  0.6343
## F-statistic: 351.3 on 2 and 402 DF,  p-value: < 2.2e-16
```

On peut une nouvelle fois réaliser un test anova afin de comparer ce modèle avec le modèle linéaire simple entre “medv” et “lstat”.

```
anova(fit_1, fit_8)
```

```
## Analysis of Variance Table
##
## Model 1: medv ~ lstat
## Model 2: medv ~ lstat + I(lstat^2)
##   Res.Df   RSS Df Sum of Sq      F    Pr(>F)
## 1    403 15784
## 2    402 12684  1    3100.2 98.254 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Ici, la statistique F est de 98.3 et la p-value associée est pratiquement nulle. Ainsi, on peut rejeter l'hypothèse nulle et considérer que le modèle contenant les prédicteurs "lstat" et "lstat"^2 est supérieur au modèle qui ne contient que le prédicteur "lstat".

On peut également introduire un degré plus élevé sur "lstat". Par exemple, considérons d = 5:

```
fit_9 <- lm(medv ~ poly(lstat, 5, raw = TRUE), data = Boston_a)
summary(fit_9)
```

```
##
## Call:
## lm(formula = medv ~ poly(lstat, 5, raw = TRUE), data = Boston_a)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -13.5315  -3.0343  -0.7429   1.9802  27.2258
##
## Coefficients:
##                                Estimate Std. Error t value Pr(>|t|)
## (Intercept)                   6.787e+01  4.113e+00  16.499  < 2e-16 ***
## poly(lstat, 5, raw = TRUE)1 -1.212e+01  1.772e+00  -6.839 3.01e-11 ***
## poly(lstat, 5, raw = TRUE)2  1.296e+00  2.646e-01   4.897 1.41e-06 ***
## poly(lstat, 5, raw = TRUE)3 -7.019e-02  1.740e-02  -4.034 6.57e-05 ***
## poly(lstat, 5, raw = TRUE)4  1.799e-03  5.144e-04   3.496 0.000525 ***
## poly(lstat, 5, raw = TRUE)5 -1.731e-05  5.577e-06  -3.104 0.002044 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.296 on 399 degrees of freedom
## Multiple R-squared:  0.6789, Adjusted R-squared:  0.6749
## F-statistic: 168.7 on 5 and 399 DF,  p-value: < 2.2e-16
```

Régression logistique

On s'intéresse désormais au jeu de données "Smarket" disponible dans la librairie "ISLR" sous R.

```
?Smarket # Pour avoir les informations sur le jeu de données
head(Smarket)
```

```
##   Year   Lag1   Lag2   Lag3   Lag4   Lag5 Volume  Today Direction
## 1 2001  0.381 -0.192 -2.624 -1.055  5.010 1.1913  0.959        Up
## 2 2001  0.959  0.381 -0.192 -2.624 -1.055 1.2965  1.032        Up
## 3 2001  1.032  0.959  0.381 -0.192 -2.624 1.4112 -0.623      Down
## 4 2001 -0.623  1.032  0.959  0.381 -0.192 1.2760  0.614        Up
## 5 2001  0.614 -0.623  1.032  0.959  0.381 1.2057  0.213        Up
## 6 2001  0.213  0.614 -0.623  1.032  0.959 1.3491  1.392        Up
```

Ici, on cherche à prédire la variable “Direction”.

Quel est le type de cette variable ?

```
class(Smarket$Direction)
```

```
## [1] "factor"
```

Quelles sont les différentes modalités de cette variable ? Et, selon R, quelle est la modalité de référence ? Réponse: “Down”

```
unique(Smarket$Direction)
```

```
## [1] Up   Down
## Levels: Down Up
```

Comme dans la partie précédente, on commence par diviser le dataset en deux échantillons: entraînement et test. Nous n’avons pas besoin de reconfigurer le “seed” ici puisque la cellule plus haut a déjà été exécuté.

```
prop <- 0.8
n_a <- round(prop * nrow(Smarket))
id_a <- sample(nrow(Smarket), size = n_a, replace = FALSE)
Smarket_a <- Smarket[id_a, ] # échantillon d'entraînement
Smarket_t <- Smarket[-id_a, ] # échantillon de test
```

Réaliser un modèle GLM de type régression logistique pour prédire la variable “Direction” en fonction des variables “Lag1”, “Lag”, “Lag3”, “Lag4”, “Lag5” et “Volume”.

```
fit_glm <- glm(Direction ~Lag1 + Lag2 + Lag3 + Lag4 + Lag5 + Volume,
               data = Smarket_a, family = binomial)
summary(fit_glm)
```

```
##
## Call:
## glm(formula = Direction ~ Lag1 + Lag2 + Lag3 + Lag4 + Lag5 +
##     Volume, family = binomial, data = Smarket_a)
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)  0.045225   0.267260   0.169    0.866
## Lag1        -0.016823   0.056564  -0.297    0.766
## Lag2        -0.076408   0.056226  -1.359    0.174
## Lag3         0.002212   0.056083   0.039    0.969
## Lag4         0.011214   0.056548   0.198    0.843
## Lag5        -0.038871   0.054841  -0.709    0.478
## Volume       0.056807   0.174590   0.325    0.745
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1382.2  on 999  degrees of freedom
## Residual deviance: 1379.6  on 993  degrees of freedom
## AIC: 1393.6
##
## Number of Fisher Scoring iterations: 3
```

Ici, on peut calculer l’EQM sur les données d’entraînement, qui se nomme “Brier score” dans le cas d’une tâche de classification. Il faut simplement convertir la variable factorielle “Direction” en variable numérique 0/1.

```r
# as.numeric transforme les niveaux factoriels "0" et "1" en 1 et 2 par défaut,
# donc on soustrait 1 pour obtenir 0 et 1.
numeric_var <- as.numeric(Smarket_a$Direction) - 1
cat("Brier score pour les données d'entraînement:",
    round(calcul_EQM(numeric_var, fit_glm$fitted.values), 2), "\n")
```

```
## Brier score pour les données d'entraînement: 0.25
```

On peut désormais prédire la régression logistique sur les données de test en utilisant la fonction “predict” et afficher les 5 premières valeurs.

```r
preds_glm <- predict(fit_glm, newdata = Smarket_t)
preds_glm[1:5]
```

```
##          4          9         12         16         17
## 0.06319493 0.12328672 0.12198624 0.09344182 0.15905860
```

Attention, ces valeurs ne correspondent pas aux scores prédits puisqu’elles ne sont pas dans l’intervalle [0,1]. Il s’agit des valeurs obtenues avant application de la fonction logistique.

```r
1 / (1 + exp(-preds_glm[1:5]))
```

```
##         4         9        12        16        17
## 0.5157935 0.5307827 0.5304588 0.5233435 0.5396810
```

On peut également obtenir ces valeurs en ajoutant une option dans la fonction “predict”:

```r
scores <- predict(fit_glm, newdata = Smarket_t, type = 'response')
scores[1:5]
```

```
##         4         9        12        16        17
## 0.5157935 0.5307827 0.5304588 0.5233435 0.5396810
```

Pour transformer ces scores en classes “Up” (1) /“Down” (0), on peut assigner la valeur “Up”/1 aux scores supérieurs à 0.5 et “Down”/0 sinon.

```r
preds_glm <- ifelse(scores > 0.5, "Up", "Down")
preds_glm[1:5]
```

```
##    4    9   12   16   17
## "Up" "Up" "Up" "Up" "Up"
```

On peut calculer l’erreur de prédiction sur les données de test en comptant le nombre de fois où le modèle se trompe:

```r
err_down <- sum(preds_glm == "Up" & Smarket_t$Direction == "Down")
cat("Nombre de fois où le modèle prédit 'Up' alors que la classe réelle est 'Down':",
    err_down, "\n")
```

```
## Nombre de fois où le modèle prédit 'Up' alors que la classe réelle est 'Down': 122
```

```r
err_up <- sum(preds_glm == "Down" & Smarket_t$Direction == "Up")
cat("Nombre de fois où le modèle prédit 'Down' alors que la classe réelle est 'Up':",
    err_up, "\n")
```

```
## Nombre de fois où le modèle prédit 'Down' alors que la classe réelle est 'Up': 6
```

```r
err_total <- err_down + err_up
cat("Nombre total d'erreurs de prédiction sur l'échantillon de test:",
    err_total, "\n")
```

```
## Nombre total d'erreurs de prédiction sur l'échantillon de test: 128
```

```r
err_taux <- (err_total/nrow(Smarket_t))*100
cat("Taux d'erreur de prédiction sur l'échantillon de test:",
    round(err_taux, 2), "%", "\n")
```

```
## Taux d'erreur de prédiction sur l'échantillon de test: 51.2 %
```

Régression multinomiale

On commence par télécharger les données iris disponibles ici: https://archive.ics.uci.edu/dataset/53/iris.

On importe le jeu de données grâce à la fonction read_csv.

```r
?read.csv
Iris <- read.csv("iris.data", header = FALSE)
colnames(Iris) <- c('sepal.len','sepal.wid','petal.len','petal.wid','species')
head(Iris)
```

```
##   sepal.len sepal.wid petal.len petal.wid     species
## 1       5.1       3.5       1.4       0.2 Iris-setosa
## 2       4.9       3.0       1.4       0.2 Iris-setosa
## 3       4.7       3.2       1.3       0.2 Iris-setosa
## 4       4.6       3.1       1.5       0.2 Iris-setosa
## 5       5.0       3.6       1.4       0.2 Iris-setosa
## 6       5.4       3.9       1.7       0.4 Iris-setosa
```

Ici, on cherche à prédire la variable “species”, qui correspond à une espèce d’iris. Les variables explicatives sont: “sepal.len” correspondant à la longueur du sépale de la fleur, “sepal.wid” à sa largeur, “petal.len” à la longueur du pétale et “petal.wid” à sa largeur.

Quel est le type de la variable “species” ? Transformez le type de la variable en facteur si son type est différent de “factor”:

```r
class(Iris$species)
```

```
## [1] "character"
```

```r
Iris$species <- as.factor(Iris$species)
class(Iris$species)
```

```
## [1] "factor"
```

Quelles sont les différentes modalités de cette variable ? Et, selon R, quelle est la modalité de référence ? Réponse: setosa

```r
unique(Iris$species)
```

```
## [1] Iris-setosa     Iris-versicolor Iris-virginica
## Levels: Iris-setosa Iris-versicolor Iris-virginica
```

Comme dans la partie précédente, on commence par diviser le dataset en deux échantillons: entraînement (80%) et test (20%).

```r
prop <- 0.8
n_a <- round(prop * nrow(Iris))
id_a <- sample(nrow(Iris), size = n_a, replace = FALSE)
Iris_a <- Iris[id_a, ] # échantillon d'entraînement
Iris_t <- Iris[-id_a, ] # échantillon de test
```

Pour réaliser un modèle de régression multinomiale en R, on ne peut pas utiliser la fonction “glm” comme pour la régression logistique. On utilise la fonction “multinom” de la librairie “nnet”.

```r
fit_multi <- multinom(species ~., data = Iris_a, family = multinomial)
```

```
## # weights:  18 (10 variable)
## initial  value 131.833475
## iter  10 value 19.068769
## iter  20 value 3.283711
## iter  30 value 2.393067
## iter  40 value 2.379347
## iter  50 value 2.342956
## iter  60 value 2.324265
## iter  70 value 2.275549
## iter  80 value 2.267370
## iter  90 value 2.245689
## iter 100 value 2.227965
## final  value 2.227965
## stopped after 100 iterations
```

```r
summary(fit_multi)
```

```
## Call:
## multinom(formula = species ~ ., data = Iris_a, family = multinomial)
##
## Coefficients:
##                  (Intercept) sepal.len  sepal.wid petal.len petal.wid
## Iris-versicolor    49.45097 -15.34434  -8.257048  25.54886 -14.67061
## Iris-virginica    -83.69343 -21.02290 -28.135012  48.76376  49.92485
##
## Std. Errors:
##                  (Intercept) sepal.len sepal.wid petal.len petal.wid
## Iris-versicolor    67.35271  82.95965  98.29765  115.1330  55.59942
## Iris-virginica     80.82289  84.26631 100.46054  120.0478  67.15035
##
## Residual Deviance: 4.455929
## AIC: 24.45593
```

On peut désormais prédire la régression multinomiale sur les données de test en utilisant la fonction “predict” et afficher les 5 premières valeurs.

```r
preds_multi <- predict(fit_multi, newdata = Iris_t)
preds_multi[1:5]
```

```
## [1] Iris-setosa Iris-setosa Iris-setosa Iris-setosa Iris-setosa
## Levels: Iris-setosa Iris-versicolor Iris-virginica
```

Cette fois-ci, la fonction “predict” par défaut nous renvoie directement les classes prédites (après avoir sélectionné la classe obtenant le score prédit le plus élevé parmi les trois classes d’espèces).

Pour récupérer les scores prédits pour chaque espèce et chaque observation de la base de test, on spécifie une option dans la fonction “predict”.

```r
scores_multi <- predict(fit_multi, newdata = Iris_t, type = "probs")
scores_multi[1:5, ]
```

```
##    Iris-setosa Iris-versicolor Iris-virginica
## 5    1.0000000    3.211111e-11   9.980820e-93
## 7    1.0000000    1.787729e-08   1.833172e-84
## 10   0.9999995    5.161265e-07   9.358233e-86
```

```
## 18   1.0000000    3.645201e-12   2.993986e-90
## 19   1.0000000    6.550372e-14   4.851288e-93
```

On peut calculer l'erreur de prédiction sur les données de test en comptant le nombre de fois où le modèle se trompe:

```
err_total <- sum(preds_multi != Iris_t$species)
cat("Nombre total d'erreurs de prédiction sur l'échantillon de test:",
    err_total, "\n")
```

```
## Nombre total d'erreurs de prédiction sur l'échantillon de test: 1
```

```
err_taux <- (err_total/nrow(Iris_t))*100
cat("Taux d'erreur de prédiction sur l'échantillon de test:",
    round(err_taux, 2), "%", "\n")
```

```
## Taux d'erreur de prédiction sur l'échantillon de test: 3.33 %
```