Rapport TP1

Léos Coutrot - Colin Coërchon

09-10-2023

Contents

IV. Application: study your own data using a linear model with transformed data

Pour cet exercice, le jeu de données choisies était plutôt libre. Nous avons alors collecté sur internet un jeu de données indiquant la valeur boursière d'Apple depuis 1981 à différents moments dans le temps. Nous avons affiné le jeu de données pour qu'il garde uniquement la première valeur de chaque année. Le jeu de données complet est disponible ici : lien.

```
data<-read.table("appledata.csv",header=TRUE,sep = "\t")
print(data)
```

```
##     Adj.Close ANNEE
## 1    0.407747  1980
## 2    0.312015  1982
## 3    0.363427  1984
## 4    0.315561  1986
## 5    1.275682  1988
## 6    1.081430  1990
## 7    1.765242  1992
## 8    0.905105  1994
## 9    0.998285  1996
## 10   0.504969  1998
## 11   3.478462  2000
## 12   1.448097  2002
## 13   1.322554  2004
## 14   9.291433  2006
## 15  24.218641  2008
## 16  26.601469  2010
## 17  51.115936  2012
## 18  71.107201  2014
## 19  98.446655  2016
## 20 167.199890  2018
```

Notre colonne cible est donc `Adj.Close` qui correspond à "Adjusted close", la valeur de clôture ajustée d'une action ou d'un autre titre financier en bourse.

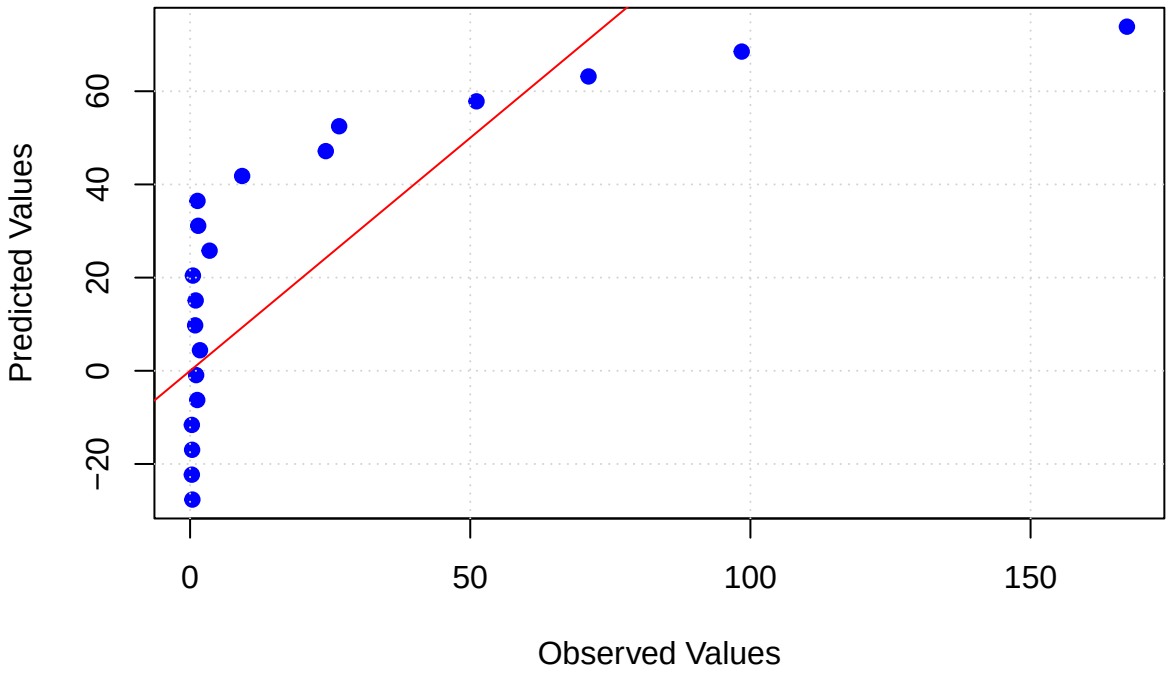
Nous avons choisi cette indicateur pour retracer l'évolution d'Apple au cours des 40 dernières années. En effet, après avoir fait quelques recherches, cela serait un des indicateurs les plus précis pour analyser la performance historique d'une action. Nous allons donc effectuer une régression linéaire sur notre tout nouveau jeu de données.

```
Y<-data$Adj.Close
X <- data$ANNEE
initial_fit <- lm(Adj.Close ~ ANNEE, data = data)
summary(initial_fit)
```

```
##
## Call:
## lm(formula = Adj.Close ~ ANNEE, data = data)
##
## Residuals:
##    Min     1Q Median     3Q    Max
```

```
## -35.14 -22.46  -4.68  13.28  93.34
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -5316.186   1193.433  -4.455 0.000306 ***
## ANNEE           2.671      0.597   4.474 0.000293 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 30.79 on 18 degrees of freedom
## Multiple R-squared:  0.5265, Adjusted R-squared:  0.5002
## F-statistic: 20.02 on 1 and 18 DF,  p-value: 0.0002934
```

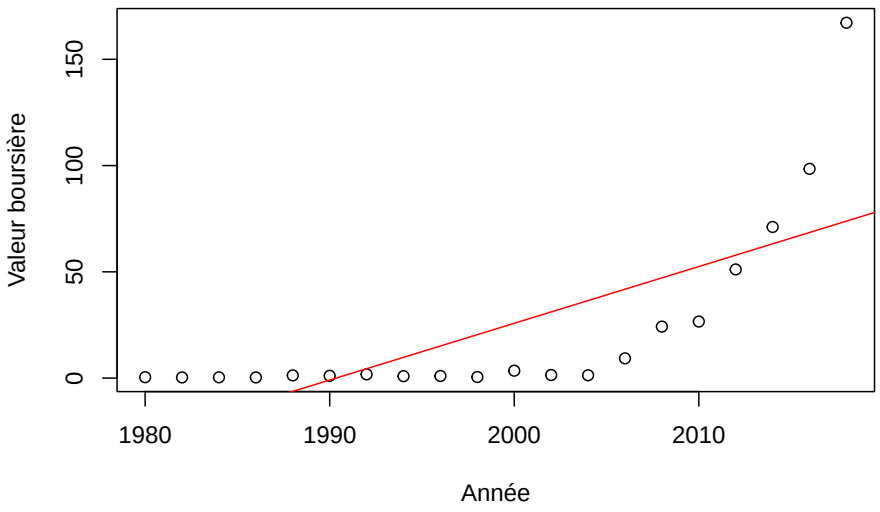
```r
plot(data$Adj.Close, fitted(initial_fit), xlab="Observed Values", ylab="Predicted Values",
     pch=19, col="blue")
grid()
abline(a=0, b=1, col="red")
```



On observe sur le graphique que le modèle proposé ne correspond visiblement pas. De plus, notre coefficient de détermination R^2 nous indique que notre modèle n'est potentiellement pas bon (cette valeur est fiable car nous travaillons ici dans un modèle avec peu de dimensions).

On se propose donc de visualiser le jeu de données avec le modèle proposé. Cela va nous permettre de visualiser, et donc mieux comprendre d'où viendrait l'erreur.

```r
plot(data$ANNEE,Y,xlab="Année",ylab="Valeur boursière")
abline(initial_fit,col="red")
```

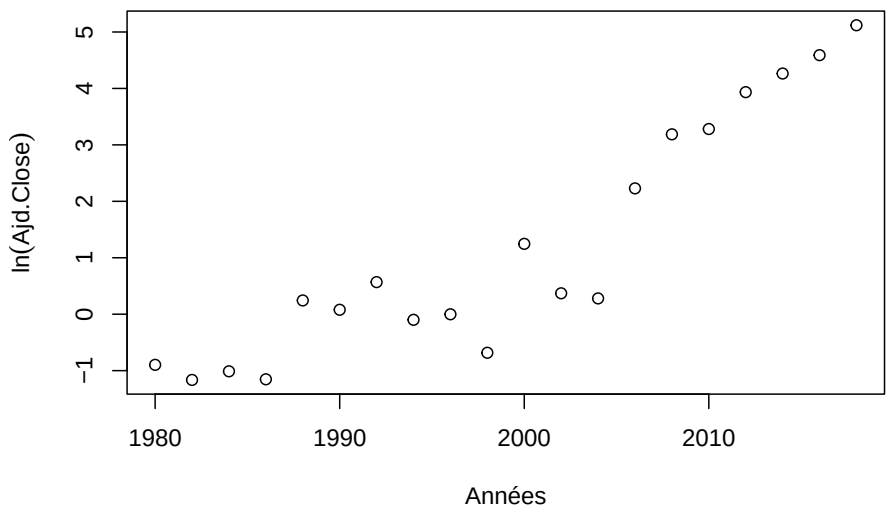


Sur le graphique, on observe ainsi que les données n'ont pas l'air de suivre un modèle linéaire. Un **modèle exponentiel** semblerait beaucoup plus adapté.

Pour ce faire, on peut appliquer à notre colonne cible (ici Y), la transformation suivante : $\widetilde{Y} = \ln(Y)$. Ainsi, on transforme les données de façon à ce qu'elles soient plus adaptés à un modèle linéaire.

Voici donc le nouveau jeu de données :

```r
data$Log_Close <- log(Y)
plot(data$ANNEE, data$Log_Close, xlab="Années", ylab=expression(ln(Ajd.Close)))
```



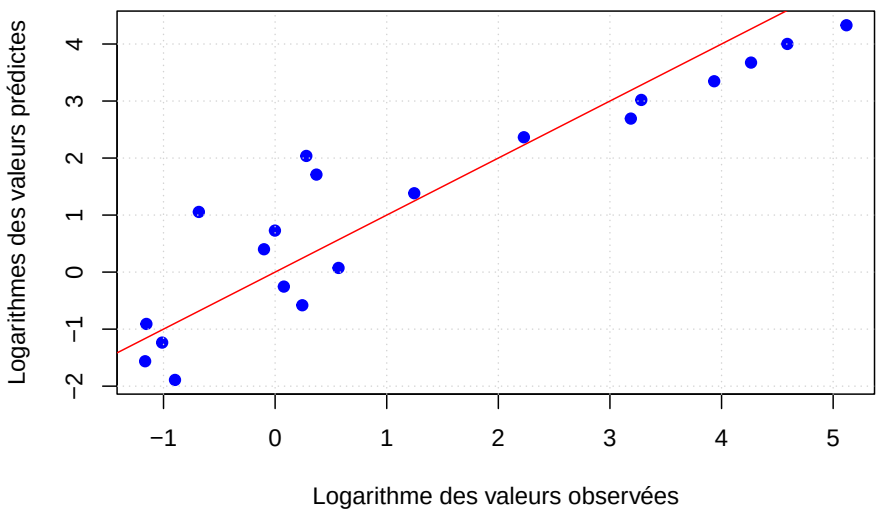
Le jeu semble désormais mieux se prêter à une régression linéaire.

```
log_fit <- lm(Log_Close ~ ANNEE, data = data)
summary(log_fit)
```

```
##
## Call:
## lm(formula = Log_Close ~ ANNEE, data = data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.7577 -0.3085  0.2970  0.5876  0.9938
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -325.96626   32.93524  -9.897 1.05e-08 ***
## ANNEE          0.16367    0.01648   9.934 9.88e-09 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.8497 on 18 degrees of freedom
## Multiple R-squared:  0.8457, Adjusted R-squared:  0.8372
## F-statistic: 98.69 on 1 and 18 DF,  p-value: 9.883e-09
```

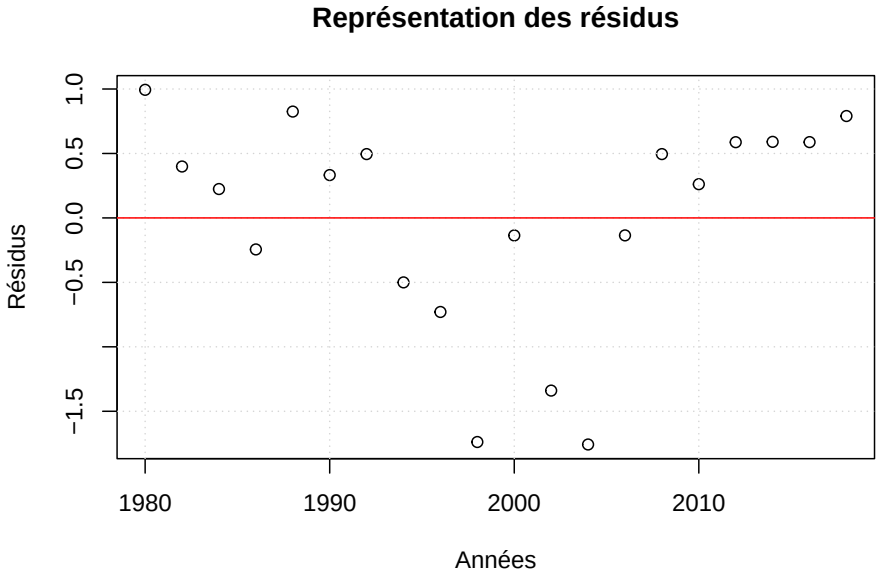
Et visuellement, ça donne ceci :

```
plot(data$Log_Close, fitted(log_fit), xlab="Logarithme des valeurs observées",
     ylab="Logarithmes des valeurs prédictes", pch=19, col="blue")
grid()
abline(a=0, b=1, col="red")
```



Et pour ce qui est des résidus ε de notre modèle :

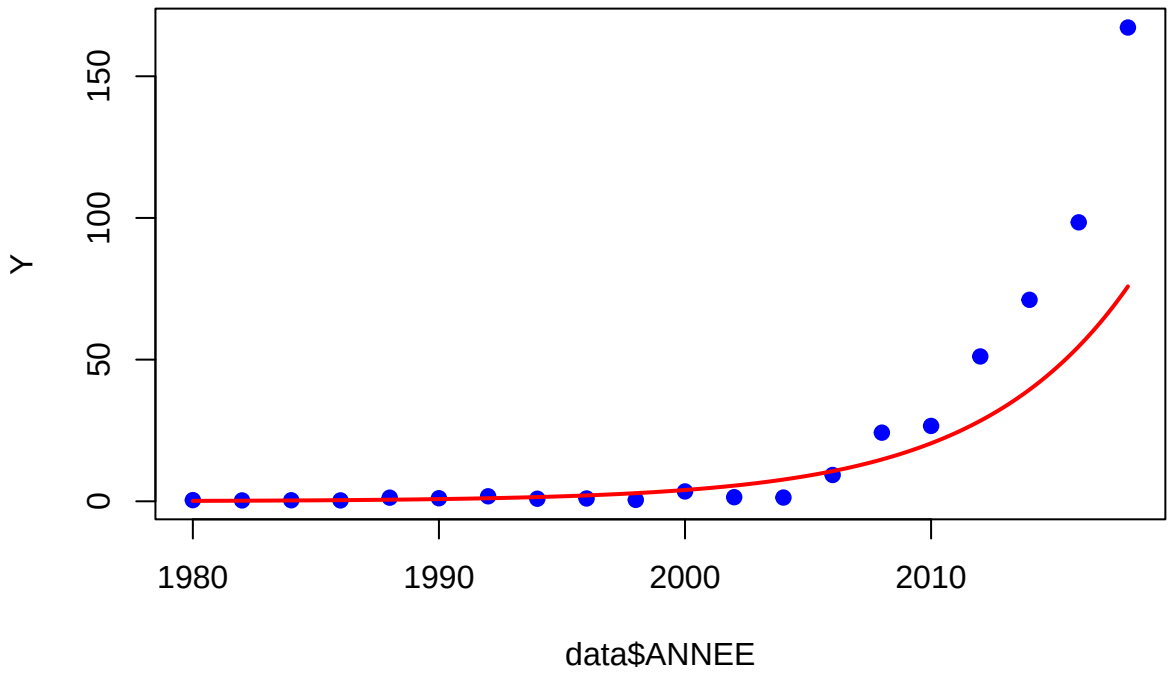
```
plot(data$ANNEE, residuals(log_fit), xlab="Années", ylab="Résidus",
     main="Représentation des résidus")
grid()
abline(h=0, col="red")
```



Le modèle proposé semble donc correspondre beaucoup mieux dans le cadre d'une régression linéaire. Notre coefficient de détermination R^2 est bien meilleur tout comme notre RSE (Residual squared error).

Si l'on souhaite revenir à nos anciennes valeurs tout en gardant la solution proposée par ce modèle (donc $\ln(Y) = at + b$), il nous suffit de visualiser $Y = e^{at+b}$.

```r
x_seq <- seq(min(data$ANNEE), max(data$ANNEE), by = 0.1)
a <- coef(log_fit)["ANNEE"]
b <- coef(log_fit)["(Intercept)"]
y_pred <- exp(a * x_seq + b)
plot(data$ANNEE, Y, pch=19, col="blue")
lines(x_seq, y_pred, col="red", lwd=2)
```



On remarque que le modèle proposé est bien meilleur que le précédent, cependant je pense personnellement qu'il pourrait être encore meilleur. Le début de l'explosion de l'exponentielle me semble être légèrement décalé par rapport à la réalité. Je pense que cela pourrait être dû au grand nombre de valeur entre 1980 et 2009 (lorsque la valeur boursière d'Apple était plus faible).

Pour améliorer le modèle je pense qu'il pourrait être pertinent d'affiner le jeu de données en ajoutant plus de données récentes (de 2019 à 2023). Malheureusement ces dernières n'existaient pas dans le jeu de données fourni.

V. Cookies Study

On nous donne dans cette partie un fichier `Cookies.csv` :

```
data_cookies<-read.table(file = "cookies.csv", header=TRUE,sep=",")
Y<-data_cookies$fat

head(data_cookies[, 1:8])
```

```
##     fat    X1100    X1102    X1104    X1106    X1108    X1110    X1112
## 1 12.57 0.259748 0.259482 0.259535 0.259270 0.259376 0.259270 0.259642
## 2 15.13 0.267625 0.267320 0.267117 0.266864 0.266965 0.267016 0.267422
## 3 12.63 0.251753 0.251654 0.251851 0.251900 0.252047 0.252342 0.252735
## 4 13.85 0.278077 0.277877 0.277827 0.278077 0.277777 0.278027 0.278177
## 5 15.25 0.288900 0.288484 0.288328 0.288328 0.288328 0.288796 0.289316
## 6 13.66 0.285423 0.284891 0.284625 0.284625 0.284678 0.284731 0.284785
```

Le fichier `Cookies.csv` comprend exactement 700 variables allant de "X1100" à "X2498". Et nous avons seulement 32 observations (cookies) dans ce jeu de données. L'objectif est d'ici d'effectuer une régression linéaire de la variable cible "Y" (`fat`) de notre jeu de données.

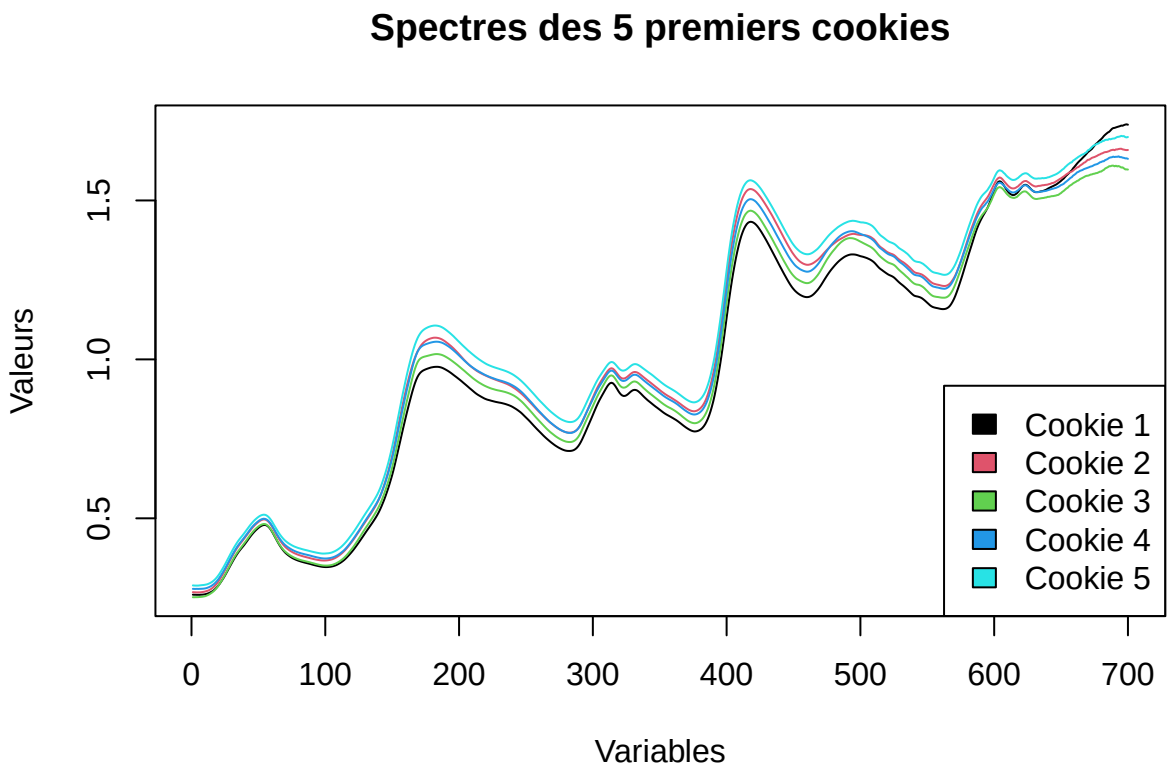
Et voilà en qualité d'exemple le spectre des valeurs pour les 5 premiers cookies pour les 700 variables de notre problème.

```
# On sélectionne les 5 premiers cookies
spectres <- as.matrix(data_cookies[1:5, -1])

# Le premier spectre
plot(spectres[1,], type='l', ylim=range(spectres), main="Spectres des 5 premiers cookies",
     xlab="Variables", ylab="Valeurs", col=1)

# Le tracé des 4 autres spectres
for (i in 2:5) {
    lines(spectres[i,], col=i)
}

legend("bottomright", legend=paste("Cookie", 1:5), fill=1:5)
```



Cette régression linéaire pourrait prendre en compte les 700 variables du problèmes et prendrait alors la forme :

$$Y = \beta_1 + \sum_{j=2}^{p+1} \beta_j X_j + \varepsilon$$

Avec :

- Y la valeur cible.
- $X = (X_1, \ldots, X^{p+1})$ les valeurs des différentes variables (avec $p = 700$ ici)
- Le "Data Set" : $D_n = \{(x_i, y_i) \,|\, i \in [\![1, n]\!],\, y_i \in \mathbb{R}, x_i \in \mathbb{R}^p\}$ (avec $n = 32$ ici).

Cela nous amènerait alors dans la partie du cours "*Towards parsimonious model*" car nous avons effectivement $p \gg n$. Quand le nombre de variables prédictives p est beaucoup plus grand que le nombre d'observations n, cela peut entraîner un surajustement du modèle, où le modèle peut parfaitement s'adapter aux données d'entraînement mais se comporte mal sur de nouvelles données. Il y a donc différentes méthodes connues pour pallier à se problème dont nous avons abordé quelques unes dans le cours n°2.

En bref, cette direction semble très intéressante, mais ce n'est pas l'objectif ici.

Dans notre situation, il nous est conseillé de transformer les données de 700 variables en 5 caractéristiques qui sont **la moyenne, l'écart-type, la "pente" (*détaillé plus tard*), le minimum et le maximum**. Et ceci pour chaque donnée de Data Set.

On se contentera donc du modèle suivant :

$$\boxed{Y = \beta_1 + \sum_{j=2}^{6} \beta_j X_j + \varepsilon}$$

Avec :

- $X_1 = (1, \ldots, 1)$.
- X_2 la moyenne.
- X_3 l'écart-type.
- X_4 la "pente" des valeurs.
- X_5 le minimum.
- X_6 le maximum.

En réduisant le nombre de caractéristiques à seulement 5, nous espérons construire un modèle plus simple qui capture l'essence **des informations spectrales** tout en évitant le surajustement dû à un trop grand nombre de variables p.

Voici la manière dont nous y sommes pris en R pour obtenir ce nouveau "Data Set" :

```r
spectra_data <- data_cookies[, -1]  # On enlève la "target value" Y (fat)

moyennes <- rowMeans(spectra_data)

ecart_types <- apply(spectra_data, 1, sd)

# Fonction pour calculer la pente d'une série de données
calculate_slope <- function(data) {
  time_sequence <- 1:length(data)
  linear_model <- lm(data ~ time_sequence)
  return(coef(linear_model)[2])
}

pentes <- apply(spectra_data, 1, calculate_slope)

minimums <- apply(spectra_data, 1, min)

maximums <- apply(spectra_data, 1, max)


features_data <- data.frame(
  Fat = data_cookies$fat,
  Moyenne = moyennes,
  EcartType = ecart_types,
  Pente = pentes,
  Min = minimums,
  Max = maximums
)

head(features_data)
```

```
##     Fat   Moyenne EcartType       Pente      Min     Max
## 1 12.57 0.9851499 0.4111868 0.001914311 0.259270 1.73946
## 2 15.13 1.0355417 0.4123933 0.001898164 0.266864 1.66273
## 3 12.63 1.0010620 0.4025158 0.001860203 0.251654 1.60960
## 4 13.85 1.0280481 0.4040351 0.001861782 0.277777 1.63881
## 5 15.25 1.0655011 0.4158252 0.001910926 0.288328 1.70320
## 6 13.66 1.0840236 0.4262425 0.001967228 0.284625 1.74356
```

À noter que la "pente", pour chaque ligne de donnée (chaque cookie), est calculée en effectuant une régression linéaire simple des valeurs spectrales par rapport à leur ordre séquentiel. Autrement dit, on trace les données spectrales et on calcule la pente de la ligne qui s'ajuste le mieux à ces points.

À présent, il est enfin temps de tenter une régression linéaire (c'est tout de même notre notre objectif initial dans cet exercice). Pour ça, on utilise comme préconisé dansle cours la fonction `lm` de R pour effectuer notre régression linéaire avec comme "Target Value" la colonne "Fat".

Voici le résultat sur RStudio :

```r
linear_model <- lm(Fat ~ ., data = features_data)

model_summary <- summary(linear_model)
print(model_summary)
```

```
##
## Call:
## lm(formula = Fat ~ ., data = features_data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.37145 -0.52481  0.03787  0.53121  1.22305
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -2.728e+00  5.739e+00  -0.475  0.63851
## Moyenne      9.563e+00  1.623e+01   0.589  0.56092
## EcartType    3.358e+02  1.404e+02   2.392  0.02430 *
## Pente       -6.614e+04  2.340e+04  -2.826  0.00893 **
## Min          5.956e-01  3.321e+01   0.018  0.98583
## Max         -3.338e+00  5.166e+00  -0.646  0.52381
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.7837 on 26 degrees of freedom
## Multiple R-squared:  0.7167, Adjusted R-squared:  0.6622
## F-statistic: 13.15 on 5 and 26 DF,  p-value: 1.939e-06
```

La régression linéaire est établie. Regardons d'un peu plus près les résultats, histoire de voir ce que nous pouvons modifier dans le modèle.

Commençons donc par afficher les coefficients β :

```r
print(model_summary$coefficients)
```

```
##                  Estimate   Std. Error    t value    Pr(>|t|)
```

```
## (Intercept) -2.727978e+00     5.738858 -0.4753520 0.638505397
## Moyenne      9.562675e+00    16.234519  0.5890335 0.560922304
## EcartType    3.358469e+02   140.427069  2.3916107 0.024299099
## Pente       -6.613780e+04 23399.193829 -2.8264992 0.008929608
## Min          5.955977e-01    33.213117  0.0179326 0.985829526
## Max         -3.338194e+00     5.165900 -0.6461980 0.523814483
```

Les coefficients donnent une indication de l'effet de chaque caractéristique sur la variable cible, "Fat". Un coefficient positif suggère une relation positive, c'est-à-dire que lorsque la caractéristique augmente, "Fat" tend aussi à augmenter, et vice versa.

On va ainsi déterminer à l'aide de la *p-value* les coefficients significatifs dans notre modèle linéaire. On prend par choix arbitraire $\alpha = 0.05$ pour classifier une *p-value* comme "petite".

```
significant_vars <- which(abs(model_summary$coefficients[, 4]) < 0.05)
if (length(significant_vars) > 0) {
  cat("Les variables suivantes sont statistiquement significatives:\n")
  print(names(significant_vars))
} else {
  cat("Aucune variable n'est statistiquement significative au seuil de 0.05.\n")
}
```

```
## Les variables suivantes sont statistiquement significatives:
## [1] "EcartType" "Pente"
```

La fonction `lm` de R nous donne aussi des informations essentielles sur le fameux coefficient R^2 :

```
##
## Ajustement du modèle:
```

```
## ----------------------
```

```
## R-squared (R²):  0.7166928
```

```
## Adjusted R-squared (R² ajusté):  0.6622106
```

Le R^2 donne une idée de la proportion de la variabilité de la variable cible qui est expliquée par les caractéristiques du modèle. Un R^2 plus proche de 1 indique un bon ajustement du modèle. Ici, on trouve un R^2 qui vaut 0.72 environ, le modèle n'est pas parfait, mais il reste quand même pertinent dans notre étude.

Intéressons-nous aux résidus maintenant (les ε) :

```
##
## Résidus du modèle:
```

```
## ----------------------
```

```
## Min:  -1.371454
```

```
## 1er Quartile:  -0.5248055
```

```
## Médiane:  0.03787396
```

```
## Moyenne:  1.539567e-17
```

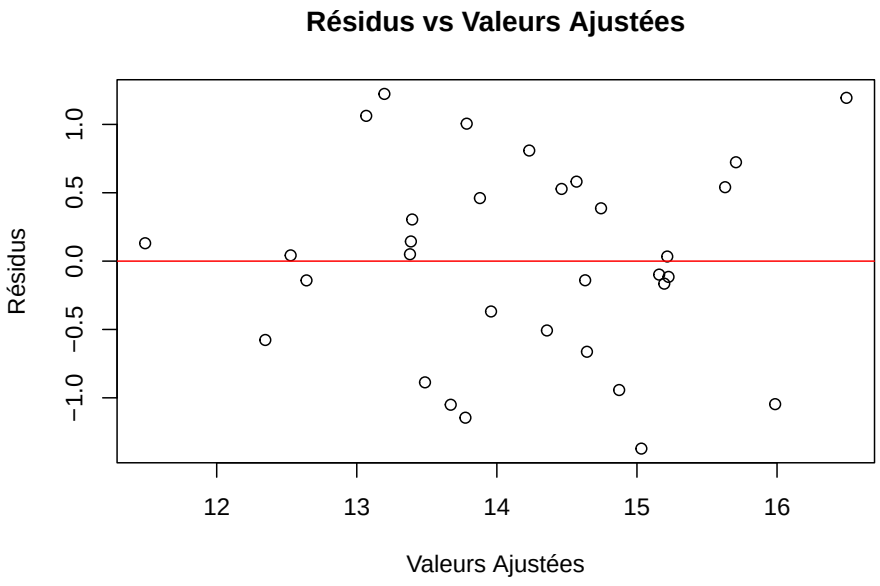
```
## 3ème Quartile:  0.5312058
```

```
## Max:  1.223046
```

Les résidus donnent une idée de la différence entre les valeurs prédites par le modèle et les valeurs réelles. Idéalement, les résidus devraient être centrés autour de zéro. Ici, on peut noter que leur moyenne est exactement à 0, et que la médiane s'en rapproche beaucoup. Les minimums et maximums ne sont pas eux-aussi pas adhérents, bien qu'ils s'écartent considérablement des valeurs réelles.

Ce graphique donne une bonne visualisation de la répartition de ces résidus ε de notre modèle.

```
plot(linear_model$fitted.values, linear_model$residuals,
     xlab = "Valeurs Ajustées", ylab = "Résidus",
     main = "Résidus vs Valeurs Ajustées")
abline(h = 0, col = "red")
```



Dans notre étude du jeu de données "Cookies.csv", nous avons tenté d'expliquer la teneur en matière grasse en utilisant cinq caractéristiques spectrales : **moyenne, écart-type, pente, minimum et maximum**. Notre modèle linéaire a un R^2 de 0.7167, indiquant qu'il explique 71,67% de la variabilité du gras, avec un R^2 ajusté de 0.6622. Seules les caractéristiques écart-type et pente sont statistiquement significatives pour cette prédiction.

Les résidus du modèle ont une moyenne et une médiane proche de zéro, indiquant un bon ajustement général. Cependant, certaines prédictions s'écartent notablement des valeurs réelles, suggérant des améliorations possibles pour le modèle (comme évoqué au début).