République Tunisienne Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Rapport de Projet Régression Linéaire

Présenté par :

Salha MEDINI

Iheb MISSAOUI

Ameni JOMAA

Inscrits en 4éme année Cycle Ingénieur en Science des Données

---

Régression Linéaire

Supervisé par :

Dr Salah KHARDANI

Année universitaire 2022-2023

TABLE DES MATIÈRES

TABLE DES FIGURES

INTRODUCTION GÉNÉRALE

La régression est une technique utilisée pour décrire la relation entre une variable cible ou dépendante et une ou plusieurs variables indépendantes ou explicatives.
Elle permet de modéliser cette relation sous forme d'une fonction mathématique, généralement une fonction linéaire ou une fonction non linéaire.

La régression linéaire est la forme la plus simple de régression. Elle consiste à trouver la droite qui se rapproche le plus des données, c'est-à-dire la droite qui minimise l'erreur entre les valeurs prédites par le modèle et les valeurs réelles. Elle est souvent utilisée pour modéliser des relations linéaires simples entre les variables.
Il existe aussi des régressions non-linéaires, qui sont utilisées pour modéliser des relations plus complexes entre les variables, telque la régression logistique.
La régression logistique est basée sur la fonction logistique, qui transforme une variable linéaire en une probabilité qui est utilisée pour prédire la variable cible. Ce type de modèles peuvent utiliser des fonctions mathématiques plus avancées pour représenter la relation entre les variables.

Dans les deux cas, la régression permet de prévoir les valeurs de la variable cible à partir des valeurs des variables indépendantes. Il est utilisé dans de nombreux domaines tels que la finance, la médecine, la psychologie, la météorologie, l'analyse de données, la statistique, entre autres.

CHAPITRE 1

CHAPITRE 1 : RÉGRESSION LINÉAIRE SIMPLE

Introduction

Dans ce chapitre nous allons présenter la régression linéaire simple avec deux modèles, le premier avec Python et le deuxiéme avec R.

1.1 Défintion de la régression linéaire simple

La régression linéaire simple est un modèle statistique utilisé pour décrire la relation entre une variable à expliquer et une variable explicative.
Le but de la régression linéaire simple est de construire un modèle mathématique qui permet de prédire la variable cible en fonction de la variable expliquative.
La droite de régression linéaire simple est définie par une équation de la forme :
Y = b0 + b1 * X
où Y est la variable à expliquer, X est la variable expliquative, b0 est l'intercept, et b1 est le coefficient de régression. Ce coefficient de régression est utilisé pour décrire la pente de la droite de régression.

La régression linéaire simple est utilisée pour étudier la corrélation entre deux variables ou pour prédire une variable en fonction d'une autre. Il est utilisé dans de nombreux domaines tels que les sciences économiques, les sciences sociales, les sciences de la santé..

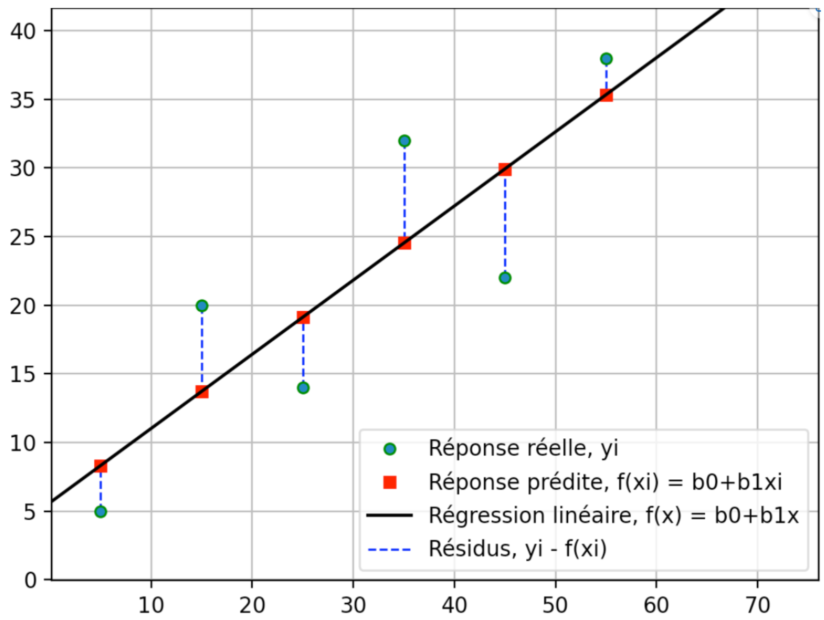


FIGURE 1.1 – Droite de régression linéaire simple

La prise d'écran illustrée par la figure 1.1 montre un exemple de droite de régression linéaire avec la nuage des points, anisi que les résidus.

1.2 La régression linéaire simple avec Python

1.2.1 Le but du modèle

Le but c'est de dévepper un model pour prédire le prix des maisons dans la région de Boston en fonction de la situation socio-économique des individus dans la région.

1.2.2 Dataset utilisée

La dataset Boston Housing est un jeu de données classique contenant des informations sur les logements dans la ville de Boston, y compris les caractéristiques des maisons et les prix de l'immobilier.

1.2.3 Les bibliothèques utilisées

Matplotlib est une bibliothèque Python pour la création de graphiques et de visualisations de données.
Pandas est utilisé pour manipuler des données sous forme de tableaux sous la forme DataFrame.
Sklearn est un module Python,il fournit des outils pour l'analyse de données, classification, régression..
Numpy est utilisé pour des calculs mathématiques.

1.2.4 Réalisation

```
[51] #Importer les bibliothèques nécessaires
     import pandas as pd
     import matplotlib.pyplot as plt
     from sklearn.model_selection import train_test_split
     from sklearn.linear_model import LinearRegression
     from sklearn import metrics
     from numpy import sqrt
     import numpy as np

[52] #Charger les données du dataset
     data=pd.read_csv('Boston.csv')
```

FIGURE 1.2 – Importation des données

[53]
```
#Afficher les cinq premières lignes du dataset
data.head()
```

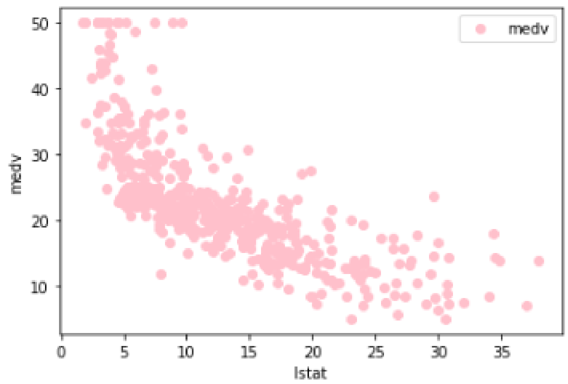
	Unnamed: 0	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	black	lstat	medv
0	1	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	24.0
1	2	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	21.6
2	3	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7
3	4	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4
4	5	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33	36.2

[54]
```
#Garder les données utils
data=data.loc[:,['lstat','medv']]

#Afficher les  premières lignes de la nouvelle dataset
data.head(5)
```

	lstat	medv
0	4.98	24.0
1	9.14	21.6
2	4.03	34.7
3	2.94	33.4

FIGURE 1.3 – Garder les données utiles

[55]
```
#Ploter la nuage des points
data.plot(x='lstat',y='medv',style='o', color='pink')
plt.xlabel('lstat')
plt.ylabel('medv')
plt.show()
```



[56]
```
#Préciser la variable expliquative et la variable à expliquer
#'LSTAT' est la variables explicative ; 'MEDV' est la variable expliquée
X=pd.DataFrame(data['lstat'])
y=pd.DataFrame(data['medv'])
```

FIGURE 1.4 – Nuage des points

```
[56] #Préciser la variable expliquative et la variable à expliquer
     #'LSTAT' est la variables explicative ; 'MEDV' est la variable expliquée
     X=pd.DataFrame(data['lstat'])
     y=pd.DataFrame(data['medv'])
```

FIGURE 1.5 – Préciser la variable expliquative et la variable à expliquer

La figure 1.6 montre que nous avons gardé que la variable expliquative "lstat" ou la situation socio-économique des individus dans la région, et la variable à expliquer "medv" le prix des maisons.

Calcule de la droite de régresion manuellement

```
[57] # calculer la moyenne de x et y
     x_bar = data['lstat'].mean()
     y_bar = data['medv'].mean()
```

FIGURE 1.6 – Calcule de x bar et y bar

```
# calculer les différences entre les valeurs de x et la moyenne de x
# et entre les valeurs de y et la moyenne de y
#Une nouvelle colonne (xi-x_bar)
data['x_diff'] = data['lstat'] - x_bar

#Une nouvelle colonne (yi-y_bar)
data['y_diff'] = data['medv'] - y_bar

#Calculer les produits des différences x et y
data['xy_diff'] = data['x_diff'] * data['y_diff']


# calculer la somme des différences x et y
#Somme  de [(xi-x_bar)]
x_diff_sum = data['x_diff'].sum()


#Somme  de [(yi-y_bar)]
y_diff_sum = data['y_diff'].sum()


#Somme de [(xi-x_bar)*(yi-y_bar)]
xy_diff_sum = data['xy_diff'].sum()


# calculer la somme des différences x au carré
x_diff_squared_sum = (data['x_diff']**2).sum()

# calculer les coefficients de régression
b1 = xy_diff_sum / x_diff_squared_sum
b0 = y_bar - (b1 * x_bar)
```

FIGURE 1.7 – Calcule des coefficients de régression

La figure 1.7 nous montre l'étape de calcule des coefficients de régression. On a b0= y bar - (b1* x bar) et b1 = La somme de (xi- x bar)*(yi-y bar) divisé par la somme de (xi-x bar)²

```
[59] # Tracer les données et la droite de régression manuellement calculés
     plt.scatter(X, y)
     plt.plot(X, (b0 + b1 * X), '-r')
     plt.show()
```

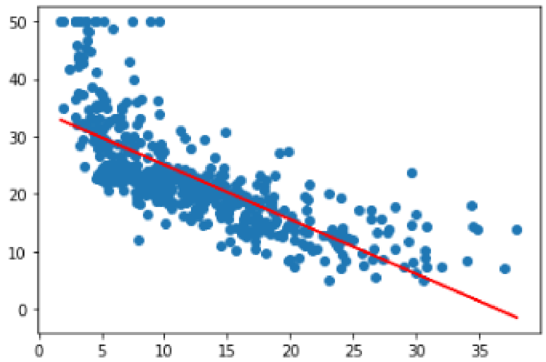


FIGURE 1.8 – Affichage graphique de la droite de régression linéaire simple

Evaluer le modele manuellement

```
[60] # Predire les valeurs de y en utilisant les coefficients de régression
     data['y_pred'] = b0 + b1 * data['lstat']
```

```
[61] # Calculer la somme des carrés des erreurs (SSE)
     sse = ((data['medv'] - data['y_pred'])**2).sum()
```

```
[62] # Calculer la somme des carrés totaux (SCT)
     y_mean = data['medv'].mean()
     sct = ((data['medv'] - y_mean)**2).sum()
```

```
[63] # Calculer le coefficient de détermination (R²)
     r2 = 1 - (sse / sct)
```

```
[64] # Calculer l'erreur quadratique moyenne (MSE)
     mse = sse / len(data)
```

```
[65] # Calculer la racine carrée de l'erreur quadratique moyenne (RMSE)
     rmse = np.sqrt(mse)
```

FIGURE 1.9 – Evaluer le modèle sans les fonctions prédéfinies

```
[66] print("SSE:", sse)
     print("SCT:", sct)
     print("R-squared:", r2)
     print("MSE:", mse)
     print("RMSE:", rmse)

     SSE: 19472.38141832644
     SCT: 42716.29541501977
     R-squared: 0.5441462975864797
     MSE: 38.48296722989415
     RMSE: 6.20346413142642
```

FIGURE 1.10 – Affichage des résultats d'évalution

Calcule de la droite de régression 2éme méthode

```
[68] #Diviser les données en 2 ensembles: ensemble d'entrainement et ensemble de test
     X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=1)

[69] #Afficher la taille de chaque ensemble
     print(X_train.shape + X_test.shape + y_train.shape + y_test.shape)

     (404, 1, 102, 1, 404, 1, 102, 1)

[70] #Créer le model de regression et l'entrainer avec les ensembles de test
     reg = LinearRegression()
     reg.fit (X_train,y_train)

     LinearRegression()
```

FIGURE 1.11 – Calcule de la droite de régression linéaire avec les fonctions prédéfinies

La deuxieme méthode consiste à utiliser les fonctions prédéfinies de python, pour la création du modèle ainsi que l'évaluation.

```
[71] #Prédire les valeurs pour X_test et les formater en un tableau
     y_pred=reg.predict(X_test)
     y_pred=pd.DataFrame(y_pred,columns=['Predicted'])
     y_pred
```

	Predicted
0	27.374117
1	27.697663
2	16.955936
3	26.847199
4	24.915168
...	...
97	26.791734
98	30.507891
99	22.317555
100	19.830873
101	16.909715

102 rows × 1 columns

FIGURE 1.12 – Prédiction pour les données de test avec le modele cré

Evaluer le model à travers les fonctions prédéfinies

```
[72] #Evaluer les performances du model cré
     print('MEAN ABSOLUTE ERROR', metrics.mean_absolute_error(y_test,y_pred))
     print('MEAN SQUARED ERROR', metrics.mean_squared_error(y_test,y_pred))
     print('ROOT ABSOLUTE ERROR', sqrt(metrics.mean_squared_error(y_test,y_pred)))
```

```
MEAN ABSOLUTE ERROR 5.078127727696937
MEAN SQUARED ERROR 46.994820919547124
ROOT ABSOLUTE ERROR 6.855276866731724
```

```
[73] #Ploter la nuage des points Et la droite de régression en superposition
     plt.scatter(X_train, y_train,color='pink')
     plt.plot(X_test, y_pred, color='blue')
     plt.show()
```

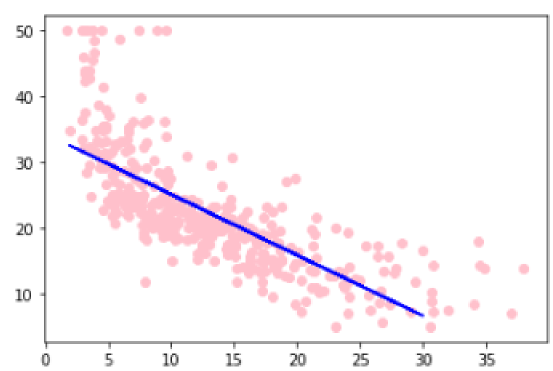


FIGURE 1.13 – Evaluation du modele cré

Les prises d'écran ci-dessus, montre les étapes suivies pour la création et l'évaluation du modèle de régression simple en Python.

1.3 La régression linéaire simple avec R

Dans la section suivant nous allons refaire le meme travail, avec le language R.

1.3.1 Le but du modèle

Le but cette section est de construire un modèle capable de prédire la taille ou le poids d'un humain.

1.3.2 Dataset utilisée

Heights and Weights Dataset : c'est un jeu de données simple, contenant la taille et le poids de 25 000 humains différents âgés de 18 ans.

1.3.3 Réalisation

Les figures illustées ci-joint montrent les étapes suivies pour aboutir à un modèle de régression linéaire avec R.

###Préparation des données ###

Data=read.csv("weight-height.csv")

print(Data)

```
    Gender   Height   Weight
1     Male 73.84702 241.8936
2     Male 68.78190 162.3105
3     Male 74.11011 212.7409
4     Male 71.73098 220.0425
5     Male 69.88180 206.3498
6     Male 67.25302 152.2122
7     Male 68.78508 183.9279
8     Male 68.34852 167.9711
9     Male 67.01895 175.9294
10    Male 63.45649 156.3997
```

#=> Ce sont des jeux de données des patients"

dim(Data)

```
> dim(Data)
[1] 10000     3
```

#Il y a 1000 patients et 3 colonnes

FIGURE 1.14 – Importation des données

attach(Data)

#=> Attacher le jeu de données pour appeler directement une variable du data

pour que toutes ses variables soient reconnues par R

FIGURE 1.15 – Créer le modèle

#Partie 1: Faire la régression

?lm

lm {stats} R Documentation

Fitting Linear Models

Description

`lm` is used to fit linear models, including multivariate ones. It can be used to carry out regression, single stratum analysis of variance and analysis of covariance (although aov may provide a more convenient interface for these).

Usage

```
lm(formula, data, subset, weights, na.action,
   method = "qr", model = TRUE, x = FALSE, y = FALSE, qr
   singular.ok = TRUE, contrasts = NULL, offset, ...)
```

FIGURE 1.16 – Prédiction pour les données de test avec le modele cré

(?lm)=> permet de donner une description du modèle linéaire

Model = lm(formula=Weight~Height,data=Data)

```
> summary(Model)

Call:
lm(formula = Weight ~ Height, data = Data)

Residuals:
    Min      1Q  Median      3Q     Max
-51.934  -8.236  -0.119   8.260  46.844

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept) -350.73719    2.11149  -166.1   <2e-16 ***
Height         7.71729    0.03176   243.0   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 12.22 on 9998 degrees of freedom
Multiple R-squared:  0.8552,	Adjusted R-squared:  0.8552
F-statistic: 5.904e+04 on 1 and 9998 DF,  p-value: < 2.2e-16
```

#=>Faire le modèle linéaire

#=>Résumer des résultats du modèle

summary(Model)

FIGURE 1.17 – Décrire le modèle cré

#=>Call nous rappelé du code saisis

#=>Résiduels permet de nous donner les écarts entre les valeurs prédites y_chapeau et y_réel(original) en précisant les mesures de dispersions

#=>L'analyse des résidus nous permet de savoir la qualité du modèle dans les parties suivantes

#Pour notre model

#beta_0 =-350,73719

#beta_1=7.71729

#beta_1>0 =>donc il'ya une dépendance positive entre le poids et la taille

#pr(>|t|) test de la pente

on a 3 étoiles donc une relation linéaire forte entre le poids et la taille

confint.default(Model)

FIGURE 1.18 – Analyse des résultats

```
> confint.default(Model)
                  2.5 %      97.5 %
(Intercept) -354.875628 -346.598756
Height         7.655036    7.779539
```

#=> Intervalle de confiance des coefficients

#=>Ici on a l'intervalle de confiance de beta_1 qui ne contient pas la valeur 0 donc on confirme qu'il a une dépendance entre poids et taille

attributes(Model)

```
> attributes(Model)
$names
 [1] "coefficients"  "residuals"     "effects"       "rank"
 [5] "fitted.values" "assign"        "qr"            "df.residual"
 [9] "xlevels"       "call"          "terms"         "model"

$class
[1] "lm"
```

#=>Cette fonction permet de nous donner toutes ce que contient notre modèle

FIGURE 1.19 – Afficher les attributs du modèle

Model$coefficients

```
> Model$coefficients
(Intercept)      Height
-350.737192    7.717288
```

S#=>Sélectionner seulement les coefficients

FIGURE 1.20 – Les coefficients du modèle cré

Model$residuals

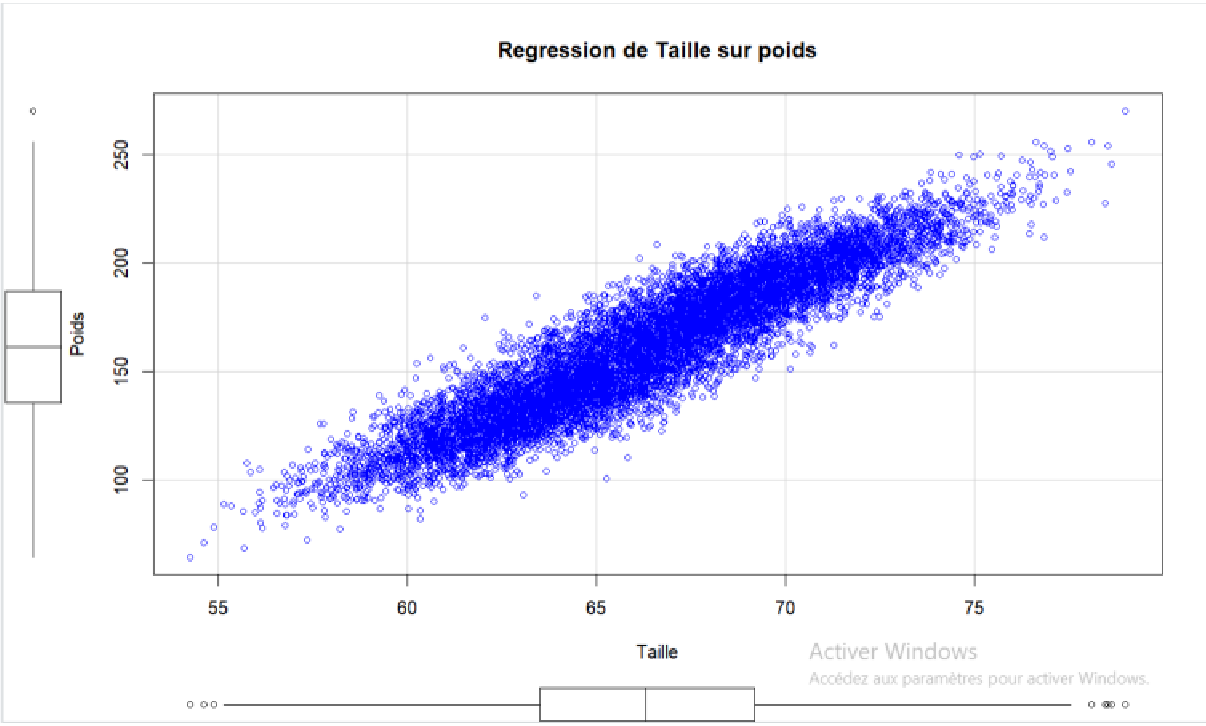
```
> Model$residuals
           1            2            3            4            5            6
22.732083254 -17.762073670 -8.450953029 17.211069022 17.789072921 -16.061519204
           7            8            9           10           11           12
 3.830823002 -8.756851722  9.462120276 17.424851492 -12.093126401 11.605661379
          13           14           15           16           17           18
18.044261525  5.505992626  2.797560894  0.883417003 -12.107281722 29.912388806
          19           20           21           22           23           24
-0.711224183  8.882149201  0.733143409 12.584482350 -10.581120710 -0.887158259
          25           26           27           28           29           30
-11.207481337 17.194066284  1.987807488 -4.259668676  5.641994424 -23.526708934
          31           32           33           34           35           36
 8.698338485 -7.692771421 -6.234382533 22.037610912 17.954033401  0.336488476
          37           38           39           40           41           42
15.144752387 15.996418274  6.735032914 17.591294493 -1.704715638 -12.323968040
          43           44           45           46           47           48
15.644907194  1.893715366 -5.599100783 -1.359305563 11.078690150  2.830499141
          49           50           51           52           53           54
 8.870667031 -0.281841112 16.402991040 10.182467779 -4.407070378 12.533482096
          55           56           57           58           59           60
 3.486514400  0.850565559 12.947202487 -3.324819109 -0.658111642 19.950412551
          61           62           63           64           65           66
-1.541911193 12.063886156 -23.651060402 -1.353720426 -20.555750009  2.208475804
          67           68           69           70           71           72
 8.365009012  9.863943113  8.507345041 15.492805842  0.605721237 14.022021327
          73           74           75           76           77           78
```

FIGURE 1.21 – Les résidus du modèle cré

###Partie 2 Faire le nuage et la droite de régression

```
install.packages("car")

#Ce package contient un outil pour visualiser les données

library(carData)

library(car)

?scatterplot

#=>Description textuelle du scatter plot

scatterplot(Weight~Height,data = Data,xlab="Taille",ylab="Poids",main="Régression de Taille sur poids",regLine=FALSE,ellipse=FALSE,smooth=FALSE,grid=TRUE)

#Si l'attribut ellipse = false on voit pas la concentration des poids

#Si l'attribut regLine = FALSE on voit pas la droite de régression

#Si 'attribut smooth = false on voit pas les lignes qui entoure la droite de régression
```

FIGURE 1.22 – Analyse des résultats



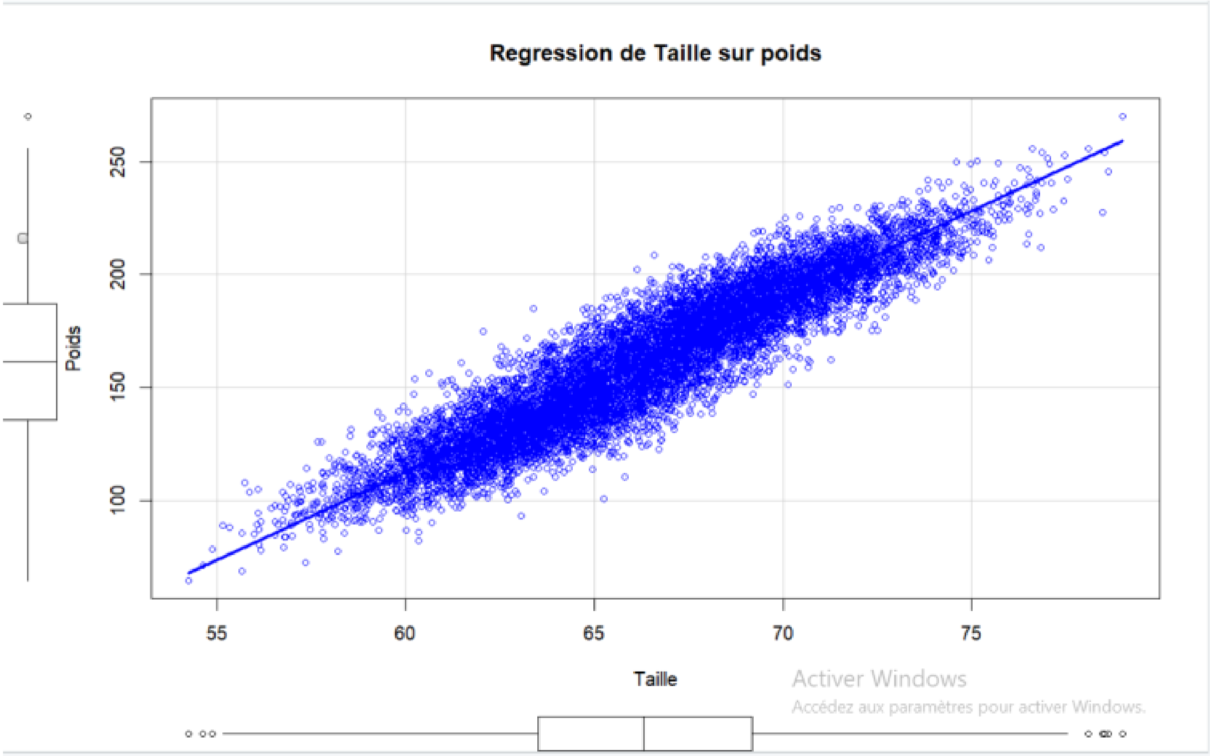
#On constate que les points sont concentrés en suivant une forme d’une ligne

#=> L'avantage c'est que scatterplot fait une boite à moustache pour chaque axe

FIGURE 1.23 – Régression de taille sur poids

#=>Maintenant on trace la droite de régression

scatterplot(Weight~Height,data = Data,xlab="Taille",ylab="Poids",main="Régression de Taille sur poids",regLine=TRUE,ellipse=FALSE,smooth=FALSE,grid=TRUE)

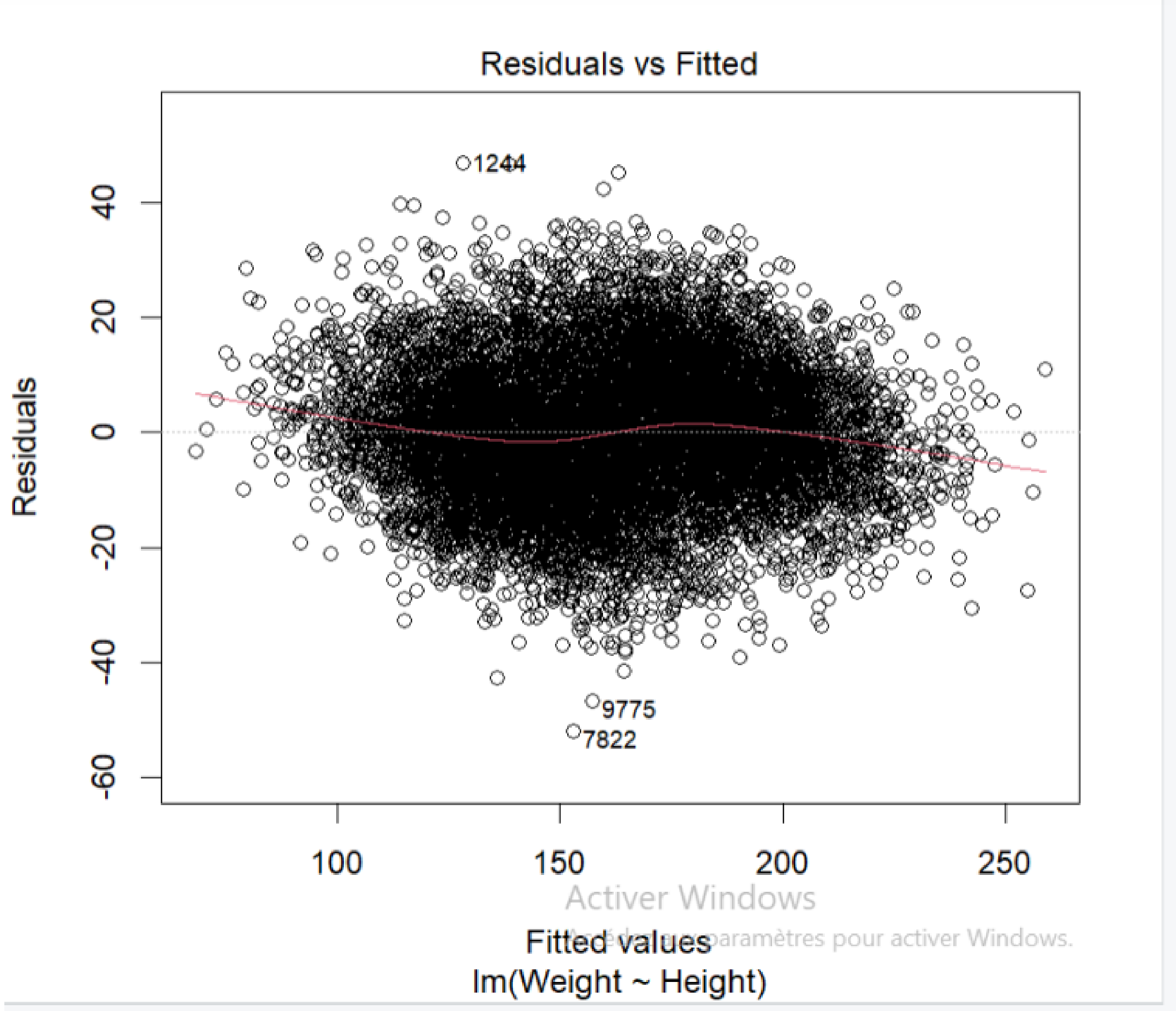


#Le coefficient de détermination r_carrée=0.8552

#=> Donc on 85% que la variation expliqué par le poids peut être expliqué par la variation de la taille

#visuellement on constate une concentration des points autour de la droite de régression

FIGURE 1.24 – Régression de taille sur poids avec la droite de régression



#=> On constate que la somme de résiduels est 0 puisque les points sont situés entre 40 et -40

#=>Les points sont homogènes (sont très proches)

FIGURE 1.25 – Les résidus et les valeurs ajustées

Passant maintenant à la régression simple sans fonctions prédéfinis.

Les prises d'écran ci-dessous montre les étapes pour la création d'un modèle de régression simple.

```
# Chargement des données
data <- read.csv("weight-height.csv")

# Sélection des variables indépendante et dépendante
x <- data$Height
y <- data$Weight

# Calcul des paramètres du modèle
beta <- sum((x-mean(x))*(y-mean(y)))/sum((x-mean(x))^2)
alpha <- mean(y) - beta*mean(x)

# Calcul de l'erreur standard
residuals <- y - (beta*x + alpha)
sigma <- sqrt(sum(residuals^2) / (nrow(data) - 2))

# Calcul de la t-value
t_value <- beta / (sigma / sqrt(sum(x^2)))

# Calcul de la p-value
p_value <- 2 * (1 - pt(abs(t_value), df = nrow(data) - 2))
```

FIGURE 1.26 – Chargement des données et calcule des parametres du modèle

```
# Test des hypothèses

if(p_value < 0.05) {

  print("La variable indépendante est statistiquement significative")

} else {

  print("La variable indépendante n'est pas statistiquement significative")

}
```

```
[1] "La variable indépendante est statistiquement significative"
>
```

```
# Calcul de l'intervalle de confiance

beta_se <- sigma / sqrt(sum(x^2))
```

FIGURE 1.27 – Les tests d'hypothèses

```
conf_int <- cbind(beta - qt(0.025, df = nrow(data) - 2) * beta_se, beta + qt(0.025, df = nrow(data) - 2)
* beta_se)

#Affichage des résultats

print(paste0("alpha = ", alpha))

print(paste0("beta = ", beta))

print(paste0("Intervalle de confiance = [", conf_int[1], ",", conf_int[2], "]"))
```

```
> #Affichage des résultats
> print(paste0("alpha = ", alpha))
[1] "alpha = -350.737191812137"
> print(paste0("beta = ", beta))
[1] "beta = 7.71728764078539"
> print(paste0("Intervalle de confiance = [", conf_int[1], ",", conf_int[2], "]"))
[1] "Intervalle de confiance = [7.72089077278991,7.71368450878087]"
> #Tracer les données et la droite de régression
> plot(x, y, xlab = "ma_variable_independante", ylab = "ma_variable_dependante", col = "blue")
> abline(a = alpha, b = beta, col = "red")
>
```

FIGURE 1.28 – Intervalle de confiance

```
#Tracer les données et la droite de régression

plot(x, y, xlab = "ma_variable_independante", ylab = "ma_variable_dependante", col = "blue")

abline(a = alpha, b = beta, col = "red")
```

FIGURE 1.29 – Traçage des données et la droite de régression

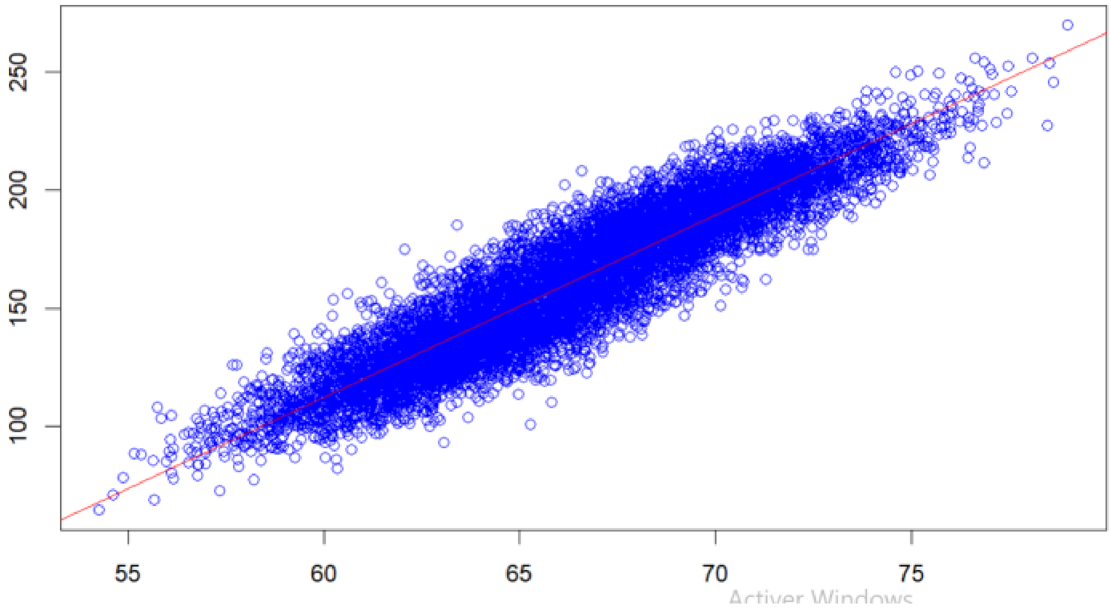


FIGURE 1.30 – Figure de la droite de régression

1.4 Conclusion

En conclusion, l'analyse de régression linéaire simple est un outil puissant pour comprendre la relation entre une seule variable indépendante et une variable dépendante. Nos résultats indiquent que la situation socio-économique des individus dans la région est un prédicteur significatif de prix des maisons dans cette région.

CHAPITRE 2

CHAPITRE : RÉGRESSION LINÉAIRE MULTIPLE

Introduction

Dans ce chapitre nous allons présenter la régression linéaire multiple avec deux modèles, le premier avec Python et le deuxième avec R.

2.1 Définition de la régression linéaire multiple

La régression linéaire multiple est un type de modèle statistique qui permet de prédire une variable cible en fonction de plusieurs variables explicatives.
Il s'agit d'une extension de la régression linéaire simple, dans laquelle il y a une seule variable expliquative.

La régression linéaire multiple est basée sur une relation linéaire entre les variables cible et explicatives. Elle est généralement représentée par une équation de la forme :

Y = b0 + b1X1 + b2X2 + ... + bnXn
où Y est la variable cible, X1, X2, ..., Xn sont les variables explicatives et b0, b1, b2, ..., bn sont des coefficients qui sont estimés à partir des données.
L'objectif de la régression linéaire multiple est de trouver les coefficients qui minimisent la somme des résidus au carré entre les valeurs prédites et les valeurs réelles de la variable cible.

2.2 Régression linéaire multiple avec Python

2.2.1 Le but du modèle

Le but du modèle que nous allons créer est d'analyser la relation entre la « publicité télévisée » , « publicité par radio », « publicité à travers les journaux » et les « ventes ».

2.2.2 Dataset utilisée

Advertising dataset un ensemble de données qui contient des informations sur les campagnes publicitaires, par télévision, par radio, par les journaux ainsi que les ventes de ce produits.

La variable cible/à expliquer est les ventes et les variables explicatives sont « publicité télévisée » , « publicité par radio » et « publicité à travers les journaux »

2.2.3 Les bibliothèques utilisées

Nous avons utilsés les memes modules du régression linéaire simple avec
Stats : ce module fournit des fonctions de calcul de statistiques mathématiques de données numériques (à valeur réelle). [1]

2.2.4 Réalisation

Les prises d'écran ci-dessous présentent les étapes de création d'un modéle de régression linéaire multiple en Python, avec les régles saisies manuellement, puis avec les fonctions prédéfinies.

```
#Importer les libreries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics
from sklearn.metrics import mean_squared_error , r2_score
from scipy import stats
```

FIGURE 2.1 – Importation des données

```
[69] # Charger les données
     #Le but est d'analyser la relation entre la « publicité télévisée » , « publicité par radio », « publicité à t
     #et les « ventes » à l'aide d'un modèle de régression linéaire multiple.
     dataset = pd.read_csv("Advertising.csv")
```

```
[70] #Afficher les cinq premières lignes du dataset
     dataset.head()
```

	Unnamed: 0	TV	Radio	Newspaper	Sales
0	1	230.1	37.8	69.2	22.1
1	2	44.5	39.3	45.1	10.4
2	3	17.2	45.9	69.3	9.3
3	4	151.5	41.3	58.5	18.5
4	5	180.8	10.8	58.4	12.9

FIGURE 2.2 – Chargement des données

```
[71] # Sélectionner les variables explicatives et la variable cible
     #La variable cible/à expliquer est les ventes
     #Les variables explicatives sont « publicité télévisée » , « publicité par radio » et  « publicité à travers
     x = dataset[['TV', 'Radio', 'Newspaper']]
     y = dataset['Sales']
```

```
[72] #Les coefficients de régression sont calculés en utilisant la formule de la régression linéaire multiple (X'X
```

```
[73] # Ajout de la constante pour la régression
     X = np.column_stack((np.ones(x.shape[0]), x))

     # Calcul des coefficients de la régression linéaire multiple manuellement
     beta = np.linalg.inv(X.T @ X) @ X.T @ y

     # Affichage des coefficients calculés manuellement
     print(beta)

     [ 2.93888937e+00  4.57646455e-02  1.88530017e-01 -1.03749304e-03]
```

FIGURE 2.3 – Calcule des coefficients Beta

La figure 2.3 montre le calcule des coefficients de régression multiple Beta en utilisant la régle : Beta = (XtX)-1*(XtY)

```
[74] # Calcul de la matrice de variance-covariance des coefficients
     X_bar = np.mean(X, axis=0)
     sigma_carre = np.sum((y - X @ beta)**2) / (X.shape[0] - X.shape[1])
     var_cov = sigma_carre * np.linalg.inv(X.T @ X)

     # Boucle pour calculer les limites inférieures et supérieures de l'intervalle de confiance
     alpha = 0.05
     for i in range(X.shape[1]):
         se = np.sqrt(var_cov[i, i])
         t = stats.t.ppf(1 - alpha/2, X.shape[0] - X.shape[1])
         Inf = beta[i] - t * se
         Sup = beta[i] + t * se
         print("Coefficient {}: {:.4f}  ({:.4f}, {:.4f})".format(i, beta[i], Inf, Sup))

     Coefficient 0: 2.9389  (2.3238, 3.5540)
     Coefficient 1: 0.0458  (0.0430, 0.0485)
     Coefficient 2: 0.1885  (0.1715, 0.2055)
     Coefficient 3: -0.0010  (-0.0126, 0.0105)
```

FIGURE 2.4 – Calcule de la matrice de covarience de beta et les intervalles de confiance

```
[75] #Diviser les données en des sets (test et entrainement)
     x_train, x_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 100)
```

```
[76] # Prédiction des ventes sur les données de test
     y_pred = x_test @ beta
```

```
[77] # Évaluation du modèle en utilisant la moyenne des erreurs au carré et le coefficient de détermination R²
     mse = mean_squared_error(y_test, y_pred)

     # Calcul du coefficient de détermination R²
     y_bar = np.mean(y_test)
     SCT = np.sum((y_test - y_bar)**2)
     Erreur = np.sum((y_test - y_pred)**2)
     r_carre = 1 - (Erreur / SCT)
     r_carre

     0.9161103358879178
```

FIGURE 2.5 – Evaluation du modèle

```
[78] #Créer le model
     model = LinearRegression()
     #Entrainer le model en lui passant les données d'entrainement
     model.fit(x_train, y_train)

     LinearRegression()
```

```
[79] #Comparer les valeur obtenus manuellement de l'intercept et les coefficients de régression
     # et les valeurs obetnus par les fonctions prédéfinies
```

```
[80] print("Intercept: ", model.intercept_)
     print("Coefficients:")
     list(zip(x, model.coef_))

     Intercept:  2.652789668879498
     Coefficients:
     [('TV', 0.0),
      ('Radio', 0.045425596023997955),
      ('Newspaper', 0.18975772766893614)]
```

FIGURE 2.6 – Développer un modèle avec les fonction prédéfinies

```python
#Prédire les ventes sur les données de test
y_pred= model.predict(x_test)
#Affichage des valeurs prédites
print("Prediction for our test set: {}".format(y_pred))
```

```
Prediction for our test set: [10.62160072 20.00625302 16.91850882 19.17040746 20.94974131 13.12284284
 11.80740696 12.32019766 20.57806782 20.95662688 10.79096475 19.54868702
  6.42403866 15.23133391  8.97226257  7.89897862 16.23599497 12.02636477
 17.09702178 11.26080277 16.97826292  9.75655721 20.82389762 17.20916742
 15.13816239 21.97290698 19.20181841 10.07501899 19.39017185 14.8673761
 14.36798893  7.55604543  9.96742165 14.76342565  7.20995576 13.60003295
  7.49088656 11.70865932 13.46091883 15.2229793  17.18088277 13.56738329
 14.30942267 13.72909849 11.88559349  8.77039705 12.1244102  19.20252289
  9.08376601  5.15367352 16.22852749 18.14111213 12.94835466 16.86274503
 17.86462435 12.33930625  4.3575739  11.25904494 16.11560622 13.56602169]
```

FIGURE 2.7 – Calcule des prédictions pour les données de test

```python
[83] #Afficher les valeurs rélles et la valeurs prédites
model_diff = pd.DataFrame({'Actual value': y_test, 'Predicted value': y_pred})
model_diff.head()
```

	Actual value	Predicted value
126	6.6	10.621601
104	20.7	20.006253
99	17.2	16.918509
92	19.4	19.170407
111	21.8	20.949741

FIGURE 2.8 – Comparer les valeurs prédites et les valeurs réelles

```python
[84] #Evalution du model:
from sklearn import metrics

meanAbErr = metrics.mean_absolute_error(y_test, y_pred)
meanSqErr = metrics.mean_squared_error(y_test, y_pred)
rootMeanSqErr = np.sqrt(metrics.mean_squared_error(y_test, y_pred))


print('Mean Absolute Error:', meanAbErr)
print('Mean Square Error:', meanSqErr)
print('Root Mean Square Error:', rootMeanSqErr)
```

```
Mean Absolute Error: 1.0638483124072013
Mean Square Error: 1.8506819941636936
Root Mean Square Error: 1.3603977338130542
```

FIGURE 2.9 – Affichage des résultats d'évalution

2.3 Régression linéaire multiple avec R

Dans cette section nous allons créer un modèle de régression linéaire multiple en utilisant le langage R, avec la meme dataset.

2.3.1 Réalisation

Les prises d'écran montrées par les figures ci-dessous explique le démarche utilisé pour arriver à notre modèle.

```
# Chargement des données

data <- read.csv("Advertising.csv")


# Sélection des variables indépendante et dépendante

X <- data.matrix(data[,  c("TV", "Radio", "Newspaper")])

Y <- data$Sales
```

FIGURE 2.10 – Importation des données et sélection des variable

print(X)

```
> print(X)
          TV Radio Newspaper
  [1,] 230.1  37.8      69.2
  [2,]  44.5  39.3      45.1
  [3,]  17.2  45.9      69.3
  [4,] 151.5  41.3      58.5
  [5,] 180.8  10.8      58.4
  [6,]   8.7  48.9      75.0
  [7,]  57.5  32.8      23.5
  [8,] 120.2  19.6      11.6
  [9,]   8.6   2.1       1.0
 [10,] 199.8   2.6      21.2
 [11,]  66.1   5.8      24.2
 [12,] 214.7  24.0       4.0
 [13,]  23.8  35.1      65.9
 [14,]  97.5   7.6       7.2
 [15,] 204.1  32.9      46.0
 [16,] 195.4  47.7      52.9
 [17,]  67.8  36.6     114.0
 [18,] 281.4  39.6      55.8
```

print(Y)

```
> print(Y)
  [1] 22.1 10.4  9.3 18.5 12.9  7.2 11.8 13.2  4.8 10.6  8.6 17.4  9.2  9.7 19.0 22.4 12.5 24.4
 [19] 11.3 14.6 18.0 12.5  5.6 15.5  9.7 12.0 15.0 15.9 18.9 10.5 21.4 11.9  9.6 17.4  9.5 12.8
 [37] 25.4 14.7 10.1 21.5 16.6 17.1 20.7 12.9  8.5 14.9 10.6 23.2 14.8  9.7 11.4 10.7 22.6 21.2
 [55] 20.2 23.7  5.5 13.2 23.8 18.4  8.1 24.2 15.7 14.0 18.0  9.3  9.5 13.4 18.9 22.3 18.3 12.4
 [73]  8.8 11.0 17.0  8.7  6.9 14.2  5.3 11.0 11.8 12.3 11.3 13.6 21.7 15.2 12.0 16.0 12.9 16.7
 [91] 11.2  7.3 19.4 22.2 11.5 16.9 11.7 15.5 25.4 17.2 11.7 23.8 14.8 14.7 20.7 19.2  7.2  8.7
```

FIGURE 2.11 – Affichage des variables expliquatives et la variable cible

```
# Fonction pour calculer les coefficients de la régression linéaire multiple
multiple_linear_regression <- function(X, y) {
  # Ajout d'une colonne de 1 à X pour gérer le terme d'interception
  X <- cbind(1, X)
  # Calcul de la transposée de X
  X_transpose <- t(X)
```

FIGURE 2.12 – Calcule des coefficients de régression

```
# Calcul de (X^T * X)^-1
X_transpose_X_inverse <- solve(X_transpose %*% X)
# Calcul de (X^T * y)
X_transpose_y <- X_transpose %*% y
# Calcul des coefficients de régression
beta <- X_transpose_X_inverse %*% X_transpose_y
```

FIGURE 2.13 – Résultat de Beta

```
# Prédiction pour de nouvelles valeurs de X
predict <- function(new_X) {
  new_X <- cbind(1, new_X)
  y_pred <- new_X %*% beta
  return(y_pred)
}
```

FIGURE 2.14 – Prédiction pour des nouvelles valeurs

```
# Test d'hypothèses pour vérifier si les coefficients de régression sont significatifs
t_test <- function(beta, X, y) {
  n <- nrow(X)
  p <- ncol(X)
  y_pred <- predict(X)
  residuals <- y - y_pred
  sse <- sum(residuals^2)
  sigma_hat <- sqrt(sse/(n-p-1))
  se <- matrix(sigma_hat*sqrt(diag(X_transpose_X_inverse)), ncol = 1)
  t_values <- beta/se
  p_values <- 2*pt(-abs(t_values), n-p-1)
  return(data.frame(beta, se, t_values, p_values))
}
```

FIGURE 2.15 – Tests d'hypothèses

```
# Intervalle de confiance pour les coefficients de régression
conf_interval <- function(beta, X, y, level = 0.95) {
  n <- nrow(X)
  p <- ncol(X)
  y_pred <- predict(X)
  residuals <- y - y_pred
```

FIGURE 2.16 – Intervalle de confiance

```
    sse <- sum(residuals^2)
    sigma_hat <- sqrt(sse/(n-p-1))
    se <- matrix(sigma_hat*sqrt(diag(X_transpose_X_inverse)), ncol = 1)
    t_value <- qt(1-((1-level)/2), n-p-1)
    lower <- beta - t_value*se
    upper <- beta + t_value*se
    return(data.frame(beta, lower, upper))
  }
  return(list(beta = beta, predict = predict, t_test = t_test, conf_interval = conf_interval))
}
```

FIGURE 2.17 – Calcule intervalle de confiance

#Voici comment utiliser les fonctions ajoutées pour la prédiction, le test d'hypothèses et l'intervalle de confiance dans la fonction de régression linéaire multiple:

Utilisation de la fonction

fit <- multiple_linear_regression(X, Y)

Prédiction pour de nouvelles valeurs de X

new_X <- cbind(TV = c(5, 6), Radio = c(4, 5),Newspaper=c(45,12))

y_pred <- fit$predict(new_X)

print(y_pred)

```
> new_X <- cbind(TV = c(5, 6), Radio = c(4, 5),Newspaper=c(45,12))
> print(y_pred)
         [,1]
[1,] 3.875145
[2,] 4.143677
>
```

Test d'hypothèses pour vérifier si les coefficients de régression sont significatifs

t_test_result <- fit$t_test(fit$beta, X, Y)

print(t_test_result)

FIGURE 2.18 – Evalution du modèle

La fonction predict(new X) permet de prédire les valeurs de y pour de nouvelles valeurs de X en utilisant les coefficients de régression calculés. La fonction ttest(beta, X, y) permet de tester si les coefficients de régression sont significatifs, elle retourne les valeurs beta, les erreurs standard, les valeurs t et les p-values pour chaque coefficient. La fonction confinterval(beta, X, y, level = 0.95) permet de calculer l'intervalle de confiance pour les coefficients de régression à un certain niveau de confiance (par défaut à 95). Elle retourne les valeurs beta, les bornes inférieures et supérieures pour chaque coefficient

```
> print(t_test_result)
                  beta          se   t_values     p_values
           2.938889369 0.311908236  9.4222884 1.267295e-17
TV         0.045764645 0.001394897 32.8086244 1.509960e-81
Radio      0.188530017 0.008611234 21.8934961 1.505339e-54
Newspaper -0.001037493 0.005871010 -0.1767146 8.599151e-01
>
```

Intervalle de confiance pour les coefficients de régression

conf_interval_result <- fit$conf_interval(fit$beta, X, Y, level = 0.95)

print(conf_interval_result)

```
Newspaper -0.001037493 0.005871010 -0.1767146 8.59
> print(conf_interval_result)
                  beta       lower      upper
           2.938889369  2.32376228 3.55401646
TV         0.045764645  0.04301371 0.04851558
Radio      0.188530017  0.17154745 0.20551259
Newspaper -0.001037493 -0.01261595 0.01054097
>
```

FIGURE 2.19 – Affichage des résultats d'évalution

Conclusion

Dans ce deuxième chapitre, nous avons présenté deux modèles de régression multiples, qui ont montré une forte dépendances entre la variable cible et les variables expliquatives.

CHAPITRE 3

CHAPITRE : RÉGRESSION LOGISTIQUE

Introduction

Dans ce dernier chapitre nous présenter la régression logistique avec deux modéles, avec Python et R.

3.1 Définition de la régression logistique

La régression logistique est un type de modèle statistique utilisé pour prédire une variable cible qui est binaire (0/1, vrai/faux, oui/non) en fonction d'autres variables explicatives. Il est utilisé lorsque on veut prédire à quelle catégorie appartient un individu ou un objet en fonction de ses caractéristiques.
La régression logistique est basée sur la fonction logistique, qui produit des résultats de probabilité compris entre 0 et 1, permettant ainsi de prédire la probabilité d'appartenir à chaque catégorie.
Les paramètres du modèle sont ajustés en utilisant une technique d'optimisation comme la descente de gradient pour maximiser la précision de la prédiction.

3.2 La régression logistique avec Python

3.2.1 Le but du modèle

Le but est de prédire si un passion est atteint par le cancer de sein ou pas, à partir d'un ensemble de données.

3.2.2 La dataset utilisée

Une collection de données qui est utilisée pour former et tester des modèles pour le diagnostic et le traitement du cancer du sein.
L'ensemble de données fournit les informations sur les patients. Il comprend plus de 4 000 enregistrements et 15 attributs.
Il comprend généralement des informations telles que les données démographiques du patient, les antécédents médicaux et les résultats d'imagerie, ainsi que les résultats du diagnostic et du traitement.

3.2.3 Réalisation

Les étapes de préparation d'un modèle de régression logistique sont illustrées par les captures d'écran ci-dessous.

```
#import les biliothèques necessaires
from sklearn.datasets import load_breast_cancer
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from matplotlib import pyplot as plt
import seaborn as sns
```

```
[2] #Chargement des données
    data = load_breast_cancer()
    X = data.data
    y = data.target
```

FIGURE 3.1 – Importation des biliothèques et des données

```
[3] # Séparation des données d'entraînement et de test
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
[4] # Entraînement du modèle de régression logistique
    clf = LogisticRegression(random_state=0)
    clf.fit(X_train, y_train)
```

FIGURE 3.2 – Dévision des données et création du modèle

```
[5] # Prédiction sur les données de test
    y_pred = clf.predict(X_test)
```

```
# Evaluation de la performance du modèle
acc = accuracy_score(y_test, y_pred)
print("précision: {:.2f}%".format(acc * 100))

précision: 96.49%
```

FIGURE 3.3 – Prédiction pour les données de test

```python
[7] #Ploter la matrice de confusion

    cm=confusion_matrix(y_test,y_pred)
    plt.figure(figsize=(12,6))
    plt.title("Matrice de confusion")
    sns.heatmap(cm, annot=True,fmt='d', cmap='Blues')
    plt.ylabel("Les valeurs rélles")
    plt.xlabel("Les valeurs prédites")
    plt.savefig('matrice.png')
```

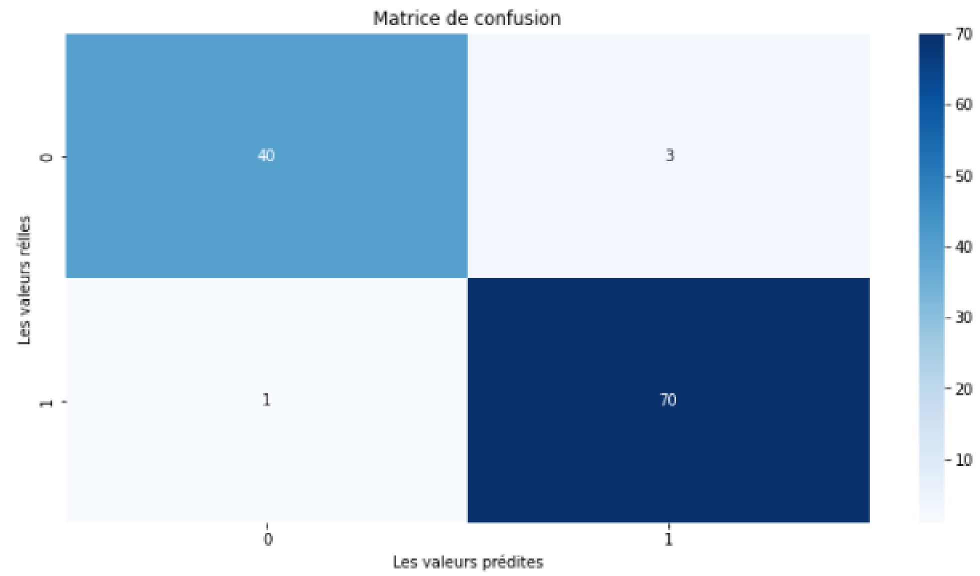


FIGURE 3.4 – Affichage graphique de la matrice de confusion

3.2.3.1 La matrice de confusion

La matrice de confusion est un résumé des résultats de prédiction pour un problème particulier de régression ou de classification.
Elle compare les données réelles pour une variable cible à celles prédites par un modèle.
Les prédictions justes et fausses sont révélées et réparties par classe, ce qui permet de les comparer avec des valeurs définies.

3.3 La régression logistique avec R

Dans cette section, nous allons refaire le meme travail déja fait mais cette fois avec le langage R, et avec une autre dataset.

3.3.1 Dataset utilisée

Framingham dataset :c'est un ensemble de données sur les maladies cardiaques « Framingham » comprend plus de 4 240 enregistrements, 16 colonnes et 15 attributs.

3.3.2 Objectif du modèle

L'objectif est créer un modèle à partir de l'ensemble de données pour de prédire si le patient présente un risque de maladie coronarienne future (CHD) sur 10 ans.

3.3.3 Réalisation

Les captures d'écran ci-dessus montre les étapes suivies pour obtenir le mdèle de prédiction. Après le chargement du bibliothèque stats et la dataset nous avons suivis le démarche suivant.

```
x <- data.matrix(data[, c("currentSmoker", "age", "totChol")])

y <- data$TenYearCHD

print(x)
```

FIGURE 3.5 – Choisir les données cibles

```
> print(x)
      currentSmoker age totChol
 [1,]             0  39     195
 [2,]             0  46     250
 [3,]             1  48     245
 [4,]             1  61     225
 [5,]             1  46     285
 [6,]             0  43     228
 [7,]             0  63     205
 [8,]             1  45     313
 [9,]             0  52     260
[10,]             1  43     225
[11,]             0  50     254
[12,]             0  43     247
[13,]             1  46     294
[14,]             0  41     332
```

print(y)

```
> print(y)
  [1] 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 1 0 0 1 0 0 0 1 1 0 0 0 0 0 0 0 0 1 0 1 0
 [47] 0 0 1 0 1 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 1 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 [93] 0 0 1 0 0 1 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
[139] 1 1 1 0 0 1 0 1 0 0 1 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0
[185] 0 0 0 1 0 0 1 1 0 1 1 0 1 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0
[231] 1 0 0 0 1 0 0 0 1 0 0 1 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0
[277] 0 1 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
[323] 0 0 0 1 0 0 1 0 0 0 0 0 0 0 1 0 0 0 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 0 0 1
[369] 0 1 0 1 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[415] 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
[461] 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0
[507] 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 1 1 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 1
[553] 0 1 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0
[599] 0 0 0 0 1 1 1 0 1 0 0 0 1 0 0 0 0 0 0 0 0 1 1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

FIGURE 3.6 – Affichage des données

```
# Construire le modèle de régression logistique
logit_model <- glm(y ~ x, family = binomial())

# Afficher les résultats du modèle
summary(logit_model)
```

FIGURE 3.7 – Construction du modèle

```
> # Construire le modèle de régression logistique
> logit_model <- glm(y ~ x, family = binomial())
> # Afficher les résultats du modèle
> summary(logit_model)

Call:
glm(formula = y ~ x, family = binomial())

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.1898  -0.6151  -0.4594  -0.3405   2.5187

Coefficients:
                 Estimate Std. Error z value Pr(>|z|)
(Intercept)    -6.6339494  0.3717370 -17.846  < 2e-16 ***
xcurrentSmoker  0.4573326  0.0924041   4.949 7.45e-07 ***
xage            0.0816158  0.0056594  14.421  < 2e-16 ***
xtotChol        0.0019876  0.0009956   1.996   0.0459 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 3564.1  on 4187  degrees of freedom
Residual deviance: 3311.5  on 4184  degrees of freedom
  (50 observations effacées parce que manquantes)
AIC: 3319.5

Number of Fisher Scoring iterations: 5
```

FIGURE 3.8 – Description du modèle

```
# Prédire les valeurs en utilisant le modèle

predictions <- predict(logit_model, newdata = data, type = "response")


# Afficher les prédictions

head(predictions)
```

```
> head(predictions)
         1          2          3          4          5          6
0.04464430 0.08449914 0.14528092 0.32063608 0.13518959 0.06468803
>
```

FIGURE 3.9 – Prédiction

Conclusion

En conclusion, ce dernier chapitre a démontré l'utilisation de la régression logistique pour modéliser la relation entre des variables indépendantes et une variable dépendante binaire. Les résultats montrent que le modèle est capable de prédire avec précision le résultat avec un niveau de signification élevé.

CONCLUSION GÉNÉRALE

En terme de conclusion, la régression est utilisée dans de nombreux domaines tels que la finance, la médecine, la psychologie, la météorologie, l'analyse de données, la statistique, entre autres.
Elle permet de prévoir les valeurs de la variable cible à partir des valeurs des variables indépendantes et de comprendre les relations entre les variables.
Il est important de choisir judicieusement les variables indépendantes et de disposer de données propres et organisées pour obtenir des résultats fiables.

NETOGRAPHIE

[1] Disponible sur le site officiel de PYTHON. ://docs.python.org/3/library/statistics.html. Consulté le 01/01/2023.