Séance 2: Quelques éléments d'analyses descriptives

J Gaudart, R Giorgi,JC Thalabard, D Thiam, S Whegang

Septembre 2010

Objectifs

- Manipulation d'un jeu de données
- Tabulations croisées
- Quelques représentations graphiques

Manipulation de fichiers

```
> setwd("/media/JCT31012009/Enseignement/Stafav/Seminaire_R_Marseille/Septembre_2010/Fichiers_Travail/")
```

Utilisation des packages *Epi* et *epicalc*

```
> library(Epi)
> library(epicalc)
```

Lecture du fichier Nutrition.csv

```
> Nutrition = read.table("Nutrition.csv",header=TRUE,sep=";")
> des(Nutrition)
```

```
 No. of observations =  337
   Variable      Class           Description
1  id            integer
2  doe           factor
3  dox           factor
4  dob           factor
5  fail          integer
6  job           factor
7  energy        numeric
8  height        numeric
9  weight        numeric
10 fat           numeric
11 fibre         numeric
```

```
> head(Nutrition)
```

```
   id      doe      dox      dob fail    job energy height weight   fat fibre
1 102 17/01/76 02/12/86 02/03/39    0 Farmer  22.86 181.61  88.18  9.17  1.40
2  59 16/07/73 05/07/82 05/07/12    0 Farmer  23.88 165.99  58.74  9.65  0.94
3 126 17/03/70 20/03/84 24/12/19   13 Worker  24.95 152.40  49.90 11.25  1.25
4  16 16/05/69 31/12/69 17/09/06    3 Farmer  22.24 171.20  89.40  7.58  1.56
5 247 16/03/68 25/06/79 10/07/18   13  Clerk  18.54 177.80  97.07  9.15  0.99
6 272 16/03/69 13/12/73 06/03/20    3  Clerk  20.31 175.26  61.01  8.54  0.77
```

Manipulation de variables dates

On souhaite déclarer les variables dob, doe et dox comme des variables dates et pouvoir les manipuler comme telles

- Fonction *as.Date*()

```
> Nutrition$dob= as.Date(Nutrition$dob,format="%d/%m/%Y")
> Nutrition$doe= as.Date(Nutrition$doe,format="%d/%m/%Y")
> Nutrition$dox= as.Date(Nutrition$dox,format="%d/%m/%Y")
```

On vérifie que les premières dates ont été converties correctement

```
> Nutrition[1:8,1:4]
```

```
   id      doe      dox      dob
1 102 76-01-17 86-12-02 39-03-02
2  59 73-07-16 82-07-05 12-07-05
3 126 70-03-17 84-03-20 19-12-24
4  16 69-05-16 69-12-31  6-09-17
5 247 68-03-16 79-06-25 18-07-10
6 272 69-03-16 73-12-13 20-03-06
7 268 69-02-16 86-12-02 19-06-24
8 206 67-01-17 86-12-02 17-09-26
```

- Calcul du temps de suivi en années et fractions d'années: fonction *cal.yr*()

```
> Nutrition$y = cal.yr(Nutrition$dox)-cal.yr(Nutrition$doe)
```

Exercice

- Créer la variable âge à l'entrée dans l'étude AgeEntree

```
> Nutrition$AgeEntree = cal.yr(Nutrition$doe)-cal.yr(Nutrition$dob)
```

- Créer la variable BMI

```
> Nutrition$BMI = Nutrition$weight/((Nutrition$height/100)^2)
```

Création d'une variable Event

Le suivi est interrompu soit parce que le sujet a atteint la date de point ou a été perdu de vue ($fail = 0$), soit parce qu'un événement est survenu ($fail > 0$). Dans ce dernier cas, les événements d'intérêt correspondent aux codes 1, 3 ou 13. On veut regrouper ces trois événements en une seule variable Event codée $(0/1)$: utilisation de la fonction logique %in%()

```
> Nutrition$Event = Nutrition$fail %in% c(1,3,13)
```

Quelques fonctions permettant de résumer le tableau de données

- *summary*
- *table*

Sélection d'un sous groupe de sujets

La variable *job* permet de sélectionner les individus en fonction d'un métier particulier

```
> Nutrition.farmer = Nutrition[Nutrition$job == "Farmer",]
```

On souhaite visualiser les 10 premières lignes et les colonnes 1 (id), 6 (job), 12 (y), 13 (AgeEntree),14 (BMI), 15 (Event) de ce sous jeu de données.

```
> Nutrition.farmer[1:10,c(1,6,12:15)]
```

```
    id    job          y AgeEntree      BMI Event
1  102 Farmer 10.8747433  36.87885 26.73564 FALSE
2   59 Farmer  8.9691992  61.02943 21.31916 FALSE
4   16 Farmer  0.6269678  62.66119 30.50212  TRUE
10   2 Farmer 11.9589322  50.53799 25.96401 FALSE
22  30 Farmer 17.3798768  46.93498 21.97072 FALSE
23  74 Farmer 16.2929500  37.09788 27.01750 FALSE
25  26 Farmer 17.3798768  47.17043 20.28399 FALSE
29  23 Farmer  4.9609856  65.03765 26.18556 FALSE
32  55 Farmer  8.8186174  61.18001 15.87416 FALSE
34  31 Farmer 17.2950034  48.47091 18.55736 FALSE
```

Les conditions de sélection des lignes ou colonnes dans un tableau ou data.frame peuvent être associées en utilisant les symboles logiques $==$ $|$ **& $\leq$ $\geq$ etc...**

Création de variables catégorielles à partir de variables continues: fonction *cut*()

La variable energy correspond à la consommation quotidienne énergétique. Son unité est la *kcal*/100. On souhaite créer une variable catégorielle en trois classes de niveaux de consommation énergétique par rapport aux deux limites 1800 et 2500 kcalories/jour

```
> Nutrition$energy.classe = cut(Nutrition$energy,breaks=c(0,18,25,100),right=FALSE)
```

Exercice: Créer une variable BMI.classe en 4 classes par rapport aux bornes 20,25,30

Bilan des variables du jeu de données Nutrition

```
> des(Nutrition)
```

```
 No. of observations =  337
   Variable      Class           Description
1  id            integer
2  doe           Date
3  dox           Date
4  dob           Date
5  fail          integer
6  job           factor
7  energy        numeric
8  height        numeric
9  weight        numeric
10 fat           numeric
11 fibre         numeric
12 y             numeric
13 AgeEntree     numeric
14 BMI           numeric
15 Event         logical
16 energy.classe factor
17 BMI.classe    factor
```

Tableaux descriptifs: fonction *table*(), *stat.table*() (package Epi)

Tableau de contingence simple entre 2 variables fonction *table*()

```
> table(Nutrition$job,Nutrition$Event)
```

```
        FALSE TRUE
  Clerk    131   20
  Farmer    90   12
  Worker    70   14
```

Tableau récapitulatif complexe: fonction *stat.table*()

```
> stat.table(job,list(sum(Event),sum(y),ratio(Event,y,1000)),data=Nutrition,margin=TRUE)
```

```
 ----------------------------------------
 job      sum(Event)  sum(y) ratio(Event,
                                 y, 1000)
 ----------------------------------------
 Clerk         20.00 2333.12         8.57
 Farmer        12.00 1227.10         9.78
 Worker        14.00 1043.45        13.42

 Total         46.00 4603.67         9.99
 ----------------------------------------
```

Ce tableau peut s'améliorer en donnant des intitulés spécifiques aux colonnes et aux lignes

```
> stat.table(list(Metier=job,BMI=BMI.classe),
+ list(Nombre=sum(Event),PA=sum(y),Incidence=ratio(Event,y,1000)),data=Nutrition,margin=TRUE)
```

```
 ------------------------------------------------
        -------------------BMI--------------------
 Metier    [0,20) [20,25) [25,30) [30,100)   Total
 ------------------------------------------------
 Clerk       1.00    8.00    9.00     1.00   20.00
           102.79 1170.02  914.73    87.18 2333.12
             9.73    6.84    9.84    11.47    8.57

 Farmer      2.00    4.00    4.00     2.00   12.00
           133.10  447.38  595.10    51.51 1227.10
            15.03    8.94    6.72    38.83    9.78

 Worker      1.00   11.00    2.00     0.00   14.00
           142.63  687.55  178.68    22.89 1043.45
             7.01   16.00   11.19     0.00   13.42


 Total       4.00   23.00   15.00     3.00   46.00
           378.52 2304.95 1688.51   161.59 4603.67
            10.57    9.98    8.88    18.57    9.99
 ------------------------------------------------
```

Tris croisés multiples de facteurs: fonction *ftable*()

On veut dénombrer les survenues d'événements en fonction du métier et des classes de BMI

```
> # On commence par r\'ecup\'erer les variables factor
> Essai = ftable(Nutrition[,c(6,15,16,17)])
> # On s\'electionne les variables en ligne (row.vars) et celles en colonnes (col.vars)
> ftable(Essai,row.vars=c(1,4),col.vars=2)
```

```
                  Event FALSE TRUE
job    BMI.classe
Clerk  [0,20)              8    1
       [20,25)            65    8
       [25,30)            50    9
       [30,100)            5    1
Farmer [0,20)             10    2
       [20,25)            33    4
       [25,30)            44    4
       [30,100)            3    2
Worker [0,20)             11    1
       [20,25)            45   11
       [25,30)            11    2
       [30,100)            2    0
```

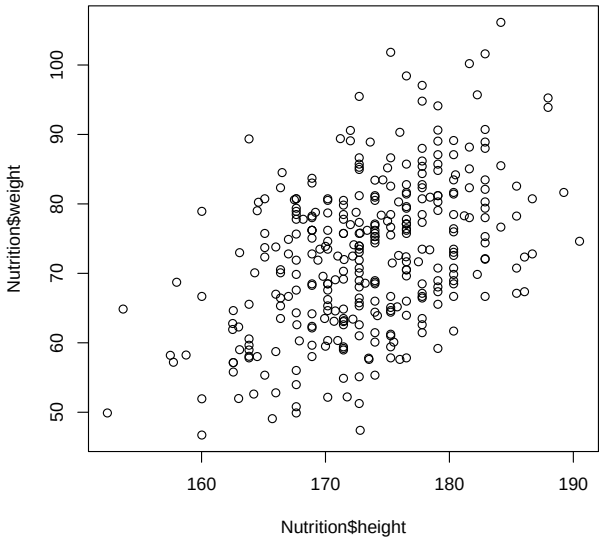
Graphes

Fonction *plot*()

Exercice: graphe du poids en fonction de la taille

- Graphe élémentaire

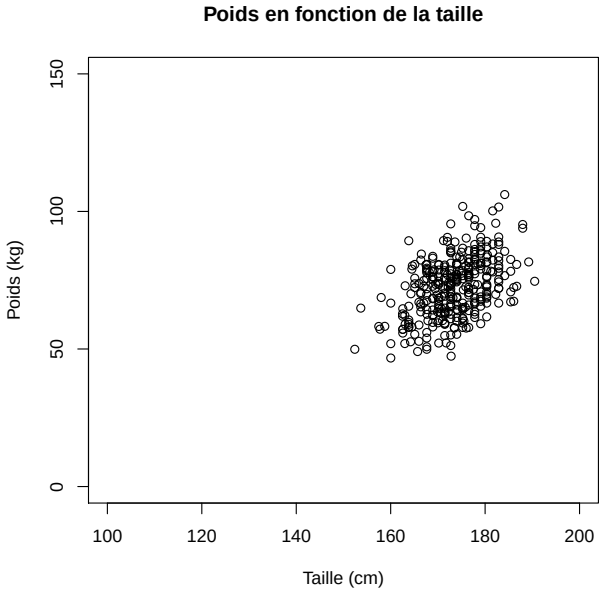
```
> plot(Nutrition$height,Nutrition$weight)
```



- Type de graphique

- Titre, sous- titres, Echelles et labels des axes, légende

```
> plot(Nutrition$height,Nutrition$weight,main="Poids en fonction de la taille",
+ xlab="Taille (cm)",ylab="Poids (kg)",xlim=c(100,200),ylim=c(0,150))
```

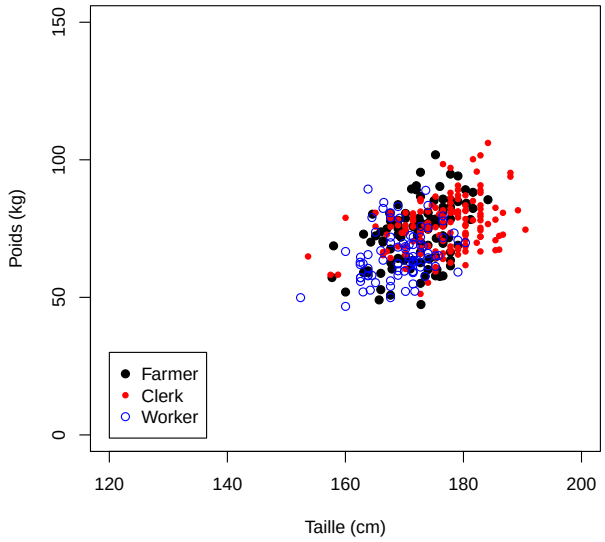


- Type de graphique type="p","l","b"....

Superposition de graphes

On veut superposer sur un même graphe les poids en fonction de la taille selon le type d'occupation

```
> plot(Nutrition$height[Nutrition$job=="Farmer"],Nutrition$weight[Nutrition$job=="Farmer"],
+ xlab="Taille (cm)",ylab="Poids (kg)",xlim=c(120,200),ylim=c(0,150),pch=19)
> points(Nutrition$height[Nutrition$job=="Clerk"],Nutrition$weight[Nutrition$job=="Clerk"],
+ main="Clerk",xlab="Taille (cm)",ylab="Poids (kg)",col="red",pch=20)
> points(Nutrition$height[Nutrition$job=="Worker"],Nutrition$weight[Nutrition$job=="Worker"],
+ main="Worker",xlab="Taille (cm)",ylab="Poids (kg)",col="blue",pch=21)
> #
> # On peut rajouter une legende
> #
> legend(x=120,y=30,legend = c("Farmer","Clerk","Worker"), pch=c(19,20,21),col=c("black","red","blue"))
```



Ajout d'une courbe sur un graphique: fonction *lines*()

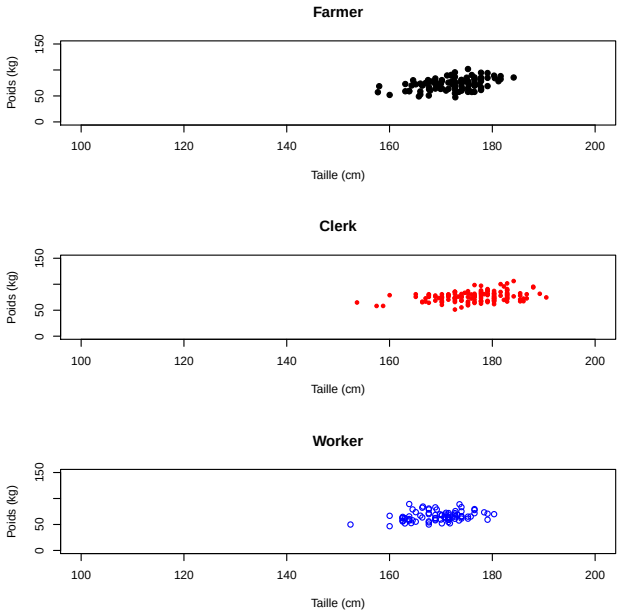
On veut rajouter sur le graphique la courbe des sujets dont le BMI vaut 30, correspondant donc au sujet de poids égal à

$$Limite = 30 * (Taille^2)$$

```
> x = seq(100,200,by=0.1)
> Limite = 30*(x/100)^2
> lines(x,Limite,col="pink")
> legend(x=120,y=100,legend = c("Limite BMI =30"), col=c("pink"),lty="solid",box.lty=0)
```

Les graphes multiples et la partition de la fenêtre graphique: la fonction *par*()

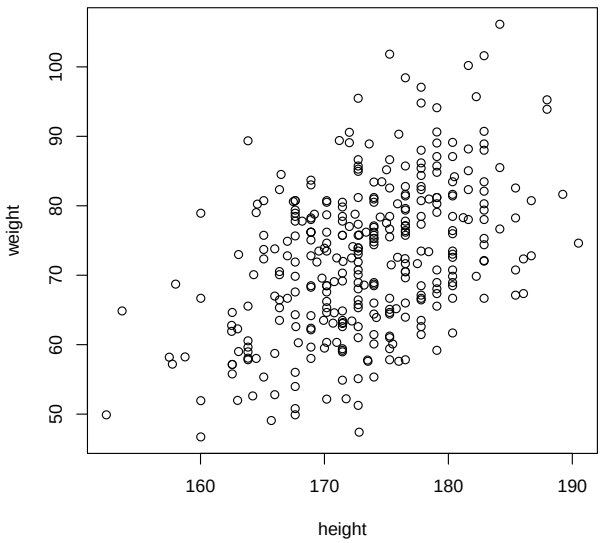
```
> op= par(mfrow=c(3,1))
> plot(Nutrition$height[Nutrition$job=="Farmer"],Nutrition$weight[Nutrition$job=="Farmer"],
+ main="Farmer",xlab="Taille (cm)",ylab="Poids (kg)",xlim=c(100,200),ylim=c(0,150),pch=19,col="black")
> plot(Nutrition$height[Nutrition$job=="Clerk"],Nutrition$weight[Nutrition$job=="Clerk"],
+ main="Clerk",xlab="Taille (cm)",ylab="Poids (kg)",xlim=c(100,200),ylim=c(0,150),pch=20,col="red")
> plot(Nutrition$height[Nutrition$job=="Worker"],Nutrition$weight[Nutrition$job=="Worker"],
+ main="Worker",xlab="Taille (cm)",ylab="Poids (kg)",xlim=c(100,200),ylim=c(0,150),pch=21,col="blue")
> par(op)
```



Au passage, les fonctions *attach*() et *detach*()

Passer son temps à écrire tout le chemin d'une variable peut paraitre fastidieux: il est possible d'attacher temporairement un data.file et rendre ainsi les noms des variables connus directement dans toutes les expressions les utilisant

```
> attach(Nutrition)
> plot(height,weight)
> detach(Nutrition)
```



Autres représentations graphiques: histogrammes, boxplot, etc...

- *hist*
- *barplot*

- *boxplot*
- *dotplot*

Bref aperçu d'épidémiologie analytique avec R: fonctions *lm*(), *glm*()

Etude de cohorte et modèle logistique sur critère binaire

- Le modèle

```
> res = glm(Event ~ job+ BMI.classe, data= Nutrition, family=binomial(link=logit))
> summary(res)


Call:
glm(formula = Event ~ job + BMI.classe, family = binomial(link = logit),
    data = Nutrition)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-0.8315  -0.5209  -0.5167  -0.4916   2.1410

Coefficients:
                  Estimate Std. Error z value Pr(>|z|)
(Intercept)        -2.0793     0.5885  -3.533 0.000410 ***
jobFarmer          -0.1061     0.3970  -0.267 0.789312
jobWorker           0.3362     0.3945   0.852 0.394110
BMI.classe[20,25)   0.1503     0.5825   0.258 0.796420
BMI.classe[25,30)   0.1329     0.6118   0.217 0.827989
BMI.classe[30,100)  0.8587     0.8543   1.005 0.314853
---
Signif. codes:  0 âĂŸ***âĂŹ 0.001 âĂŸ**âĂŹ 0.01 âĂŸ*âĂŹ 0.05 âĂŸ.âĂŹ 0.1 âĂŸ âĂŹ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 263.47  on 331  degrees of freedom
Residual deviance: 261.31  on 326  degrees of freedom
  (5 observations deleted due to missingness)
AIC: 273.31

Number of Fisher Scoring iterations: 4
```

- Fonctions associées utiles
  - coeff(res)
  - ci.lin(res,alpha=0.05,Exp=T)
  - fitted.values(res)
  - plot(res)
  - anova(res)
- Il convient en fait ici de prendre en compte le temps de suivi (variable y) en effectuant une régression de type Posson qui s'écrit

```
> res.poisson = glm(Event ~ job+ BMI.classe+offset(log(y)), data= Nutrition, family=poisson)
> summary(res.poisson)
```

```
Call:
glm(formula = Event ~ job + BMI.classe + offset(log(y)), family = poisson,
    data = Nutrition)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-0.8073  -0.5622  -0.5216  -0.4505   3.1320

Coefficients:
                    Estimate Std. Error z value Pr(>|z|)
(Intercept)        -4.812563   0.555818  -8.659   <2e-16 ***
jobFarmer           0.157129   0.371932   0.422    0.673
jobWorker           0.493573   0.362034   1.363    0.173
BMI.classe[20,25)   0.003623   0.545533   0.007    0.995
BMI.classe[25,30)  -0.030989   0.573469  -0.054    0.957
BMI.classe[30,100)  0.691049   0.769918   0.898    0.369
---
Signif. codes:  0 âĂŸ***âĂŹ 0.001 âĂŸ**âĂŹ 0.01 âĂŸ*âĂŹ 0.05 âĂŸ.âĂŹ 0.1 âĂŸ âĂŹ 1

(Dispersion parameter for poisson family taken to be 1)

    Null deviance: 255.01  on 331  degrees of freedom
Residual deviance: 252.03  on 326  degrees of freedom
  (5 observations deleted due to missingness)
AIC: 354.03

Number of Fisher Scoring iterations: 7
```

Etude de cohorte et données de suivi: package survival

Etude cas- témoins

La fonction glm est utilisée de la même façon, mais le coefficient d'intercept n'a pas de sens et ne doit pas être utilisé.

Cas particulier: données agglomérées

res.aggrege = glm(***cbind(cas,controles)*** covariable1+ covariable2,..., data = fichier, family= binomial(link=logit))