Data Science Hacking Basics

avec Linux, R, et Chrome

Master IM, Paris 5

@comeetie

Révisions de R

Master IM, Paris 5

@comeetie

R

- language de haut niveau
- support natif des valeur manquantes
- programation objet
- éco-système vivant : beaucoup de packages
- ! plutôt permisif
- http://cran.r-project.org/doc/contrib/Lam-IntroductionToR_LHL.pdf
- ...

Les types de base

les vecteurs :

```
# vecteur d'entier
a = c(1,5,10)
class(a)
# de chaine de caractère
b = c("a","g","t","t","c","g","g")
class(b)
```

- permet de stocker des éléments de même type
- opérations de bases c, length, seq, rep, indexation logique

Les types de base

les vecteurs, manipulations de bases :

```
length(a)
a[1:2]
i = 1:2
a[i]
i = seq(1,length(b),2)
b[i]
i = rep(1,5)
b[i]
i = rep(c(1,2),5)
b[i]
i = rep(c(1,2),each=3)
b[i]
i = (b=="g")
b[i]
```

Les types de base

les facteurs :

```
b = c("a","g","t","t","c","g","g")
c = factor(b,levels=c("a","t","g","c"))
levels(c)
unclass(c)
```

- type particulier de vecteurs pour coder des catégories "les niveaux (levels)"
- opérations de bases c, length, levels, unclass
- ! interprétation des chaines de caractères comme des facteurs lors de la création d'une data.frame, cf option stringAsFactor

Les types de base

les matrices :

```
# matrice d'entier
a = matrix(c(1,5,10,10),2,2)
# de chaine de caractère
b = rbind(c("a","g"),c("t","t"),c("c","g"),c("t","g"))
c = cbind(c("a","g"),c("t","t"),c("c","g"),c("t","g"))
dim(b)
t(b)
dim(t(b))
a[1,]
b[,2]
c[c[,1]=="a",]
```

- permet de stocker des éléments de même type
- opérations de bases dim, rbind, cbind, indexation logique

Les types de base

les array :

```
# tenseur de dimension 3
a = array(runif(50),dim=c(5,5,2))
a[1,,]
a[,5,]
a[,2,1]
```

- permet de stocker des éléments de même type
- opérations de bases dim, indexation logique

Les types de base

les listes :

```
l = list(a,b,c)
length(l)
l[[2]]
l = list(a=a,b=b,c=c)
l$a
l$c
l[[1]]
```

- permet de stocker des éléments de type différents
- opérations de bases length

Les types de base

les data.frame :

```
d = data.frame(v1=rep("a",10),v2=1:10,v3=runif(10))
dim(d)
d$v1
d[,3]
d$v4 = factor(rep(c("a","b"),5),levels=c("a","b"))
d[d$v4=="a",]
d[,"v2"]
d[,c(3,1)]
d[,c("v2","v4")]
names(d)
summary(d)
```

- permet de stocker des éléments de type différents
- liste de vecteurs només indexable et manipulable comme une matrice
- opérations de bases dim, cbind, rbind, names, summary

Les fonctions

```
f = function(a,b){
  return(a-b)
}
f(5,6)
f(b=5,a=6)
f = function(a=32,b=12){
  a-b
}
f()
f(5,6)
f(b=5,a=6)
```

- une variable comme une autre ?
- argument nomé et valeur par défaut
- pas besoin de return explicite

Les structures de contrôle

```
if(a == b){}
for (i in 1:length(a)){}
while(i > 4){i=i-1}
```

! éviter les boucles for, while préférer les opérations vectorielle

```
a=runif(100000)
t=Sys.time()
for (i in 1:length(a)){a[i]=a[i]+5}
t1=Sys.time()-t
t1
```

Version vectorielle

```
t=Sys.time()
a=a+5
t2=Sys.time()-t
t2
as.numeric(t1)/as.numeric(t2)
```

Quelques fonctions vectorielles

somme (sum), somme cumulée (cumsum), différences finies (diff), max, min ...

```
a=data.frame(v1=runif(5000),v2=rnorm(5000),v3=rbinom(5000,5,0.2))
# opération algébrique de base
a$v1+a$v2;a$v1*a$v2;a$v1/a$v2
# produit matriciel
t(a$v1)%*%a$v2
# somme et somme cumulé
sum(a$v2);cumsum(a$v1)
# différence
diff(a$v2)
# max, min ...
max(a$v3)
which.max(a$v1)
which(a$v1>0.2)
# concatenation de chaine de caractères
paste(a$v1,a$v2)
# sommes sur les matrices
b=matrix(runif(100),10,10)
sum(b);rowSums(b);colSums(b)
```

Apply, lapply, sapply

Appliquer une fonction à chaque élément d'un objet

```
a=data.frame(v1=runif(5000),v2=rnorm(5000),v3=rbinom(5000,5,0.2))
# appliquer à chaque lignes
r=apply(a,1,sum)
head(r);class(r);dim(r)
# appliquer à chaque colonnes
r=apply(a,2,function(col){c(max(col),which.max(col))})
r;class(r);dim(r)
# appliquer à tous les éléments d'une liste
b=list(v1=runif(5000),v2=rnorm(5000),v3=rbinom(5000,5,0.2))
r=lapply(b,which.max)
r;class(r)
r=sapply(b,which.max)
r;class(r)
```

à préférer aux boucles...

Subset : sample, logical indexing

Sélectionner une partie des données

```
#logical indexing
a[a$v1>0.5 & a$v3==3,]
#fonction subset
subset(a,v1>0.5 & v3==3)
```

Binning : cut

Prétraiter les variables pour construires des facteurs // intervalles

```
r=cut(a$v2,c(-Inf,-3,-2,2,1,Inf))
class(r);head(r)
```

Jointure : merge, %in%, match

```
a=data.frame(id=1:500,val1=runif(500))
b=data.frame(id=sample(500,500),val2=runif(500))
# jointure par colonne de même nom
c=merge(a,b)
# recherche des indices de correspondances
match(a$id,b$id)
d=cbind(a,b$v2[match(a$id,b$id)])
sum(d!=c)
# matching multiples
b=data.frame(id=sample(500,1000,replace=T),val2=runif(1000))
match(a$id,b$id)
match(b$id,a$id)
a$id %in% b$id
c=merge(a,b)
head(c)
c=merge(a,b,all.x=T)
head(c)
```

Aggrégation : tapply, by, aggregate

```
a=data.frame(id=1:500,val1=runif(500),val2=factor(rbinom(500,5,0.4)))
aggregate(a$val1,list(a$val2),sum)
tapply(a$val1,list(a$val2),summary)
by(a$val1,list(a$val2),summary)
```

Comptage : table

```
table(a$val2)
a$val3=rep(c("a","t","g","c"),500/4)
table(a[,c('val2','val3')])
```

Dédupliquer : unique, duplicated

```
val=rbinom(500,5,0.4)
unique(val)
duplicated(val)
```

Ordonner : sort, order

```
a=runif(500)
#selon une variable
sort(a)
order(a)
# selon plusieur variables
a=data.frame(id=1:500,val1=runif(500),val2=rbinom(500,5,0.4))
a[order(a$val2,a$val1),]
```