PROJECT R

Code_R_1

install.packages("matrixcalc")

pour manipuler des matrices Un ensemble de fonctions pour prendre en charge les calculs matriciels pour l'analyse probabiliste, économétrique et numérique

```
library(matrixcalc)
version
```

```
##                _
## platform       x86_64-pc-linux-gnu
## arch           x86_64
## os             linux-gnu
## system         x86_64, linux-gnu
## status
## major          4
## minor          0.2
## year           2020
## month          06
## day            22
## svn rev        78730
## language       R
## version.string R version 4.0.2 (2020-06-22)
## nickname       Taking Off Again
```

```
rm(list =ls())
```

Pour nous assurer que nous disposons d'un environnement propre en R avant de soumettre le traitement par lot

10*(1+1+1.5)

```
rm(list =ls())
```

réaliser un calcul simple

- *carre*

```
10**2
```

```
## [1] 100
```

- *racine*

```
sqrt(100)
```

```
## [1] 10
```

- *puissance*

```
100**(1/2)
```

```
## [1] 10
```

- *pi = 3.1416*

```
pi
```

```
## [1] 3.141593
```

- *cosinus*

```
cos(pi)
```

```
## [1] -1
```

- *sinus*

```
sin(pi/2)
```

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

Ces fonctions donnent les fonctions trigonométriques évidentes. Ils calculent respectivement le cosinus, le sinus, la tangente, l'arc-cosinus, l'arc-sinus, l'arc-tangent et l'arc-tangent à deux arguments.

cospi(x), sinpi(x)Et tanpi(x), compute cos(pi*x), sin(pi*x)et tan(pi*x).

- *exponentielle*

```
exp(1)
```

```
## [1] 2.718282
```

- log neperien

```
log(1)
```

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

- arrondi à un entier

```
round(2.566)
```

```
## [1] 3
```

- arrondi 2 chiffres apres

```
round(pi,2)
```

```
## [1] 3.14
```

Pour créer un vecteur, on utilisera la fonction c la lettre c étant un raccourci du mot anglais combine puisque cette fonction permet de combiner des valeurs individuelles dans un vecteur unique

- un vector

```
v <- c(10,20,30)
```

- longueur du vector

```
length(v)
```

```
## [1] 3
```

- sur chaque composante du vector

```
2*v+1
```

```
## [1] 21 41 61
```

- carre de chaque composante

```
v**2
```

```
## [1] 100 400 900
```

- log de chaque composante

```
log(v)
```

```
## [1] 2.302585 2.995732 3.401197
```

- un autre vector

```
w <- c(1,2,3)
```

- soustraction membre a membre

```
v-w
```

```
## [1]  9 18 27
```

- multiplication membre a membre

```
v*w
```

```
## [1] 10 40 90
```

- division membre a membre

```
v/w
```

```
## [1] 10 10 10
```

- produit scalaire

```
v%*%w
```

```
##      [,1]
## [1,]  140
```

- $=somme$

```
sum(v)
```

```
## [1] 60
```

- $moyenne$

```
mean(v)
```

```
## [1] 20
```

- $=minimum$

```
min(v)
```

```
## [1] 10
```

- $=maximum$

```
max(v)
```

```
## [1] 30
```

- $=ecart\ type$

```
sd(v)
```

```
## [1] 10
```

- $=medianne$

```
median(v)
```

```
## [1] 20
```

autres type de function :

- La fonction REP pour indiquer la valeur à répéter

- La fonction SEQ pour un vecteur avec une suite de valeur

- La fonction GLIMPSE ce qui signifie aperçu en anglais), qui permet de visualiser rapidement et de manière condensée le contenu d'un tableau de données.

- un autre vector u[2] #deuxieme composante

```
u <- c(1,2,3,4,5,6,7,8)
```

- nouveau vector

```
u[3:5]
```

```
## [1] 3 4 5
```

- affectation une composante

```
u[8] <- 80
```

- affectation 5 composantes u

```
u[1:5] <- 1
```

- jeux avec les vectors

```
v <- c(10,20,30,30,60,50)
```

COMBINER DES VECTEURS

Pour combiner des vecteurs, rien de plus simple. Il suffit d'utiliser *c!* Les valeurs des différents vecteurs seront mises bout à bout pour créer un unique vecteur.

- un autre vector

```
w <- c(20,10,31,31,61,51)
```

- un autre vector

```
u <- c(5 ,5 ,5 ,32,62,49)
```

str afficher de maniére compacte la structure d'un objet R

jeter un oeil sur les data n

```
str(v)
```

```
##  num [1:6] 10 20 30 30 60 50
```

Valeurs manquantes lorsqu'on travaille avec des données d'enquêtes, il est fréquent que certaines données soient manquantes, en raison d'un refus du participant de réponse à une question données ou d'un oubli ou d'un dysfonctionnement du matériel de mesure

- nb de valeurs manquantes

```
sum(is.na(v))
```

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

- un vecteur avec 3 valeur manquantes

```
v_ <- c(NA,v,NA,NA)
```

- nb valeurs manquantes

```
sum(is.na(v_))
```

```
## [1] 3
```

- min et max du vector

```
range(v)
```

```
## [1] 10 60
```

- min et max du vector ECHEC !

```
range(v_)
```

```
## [1] NA NA
```

il ne faut pas confondre NA avec un autre objet qu'on rencontre sous R qui s'appelle NULL et représente l'objet vide. NULL ne contient absolument rien.

NA (Not Available) indique une valeur manquante

- sans tenir compte des NA

```
range(v_ , na.rm = TRUE)
```

```
## [1] 10 60
```

- quartiles de v

```
quantile(v)
```

```
##   0%  25%  50%  75% 100%
## 10.0 22.5 30.0 45.0 60.0
```

- resume

```
quantile(v, probs =c(0,0.1,0.9,1))  # 80/20 0 summary(v)
```

```
##   0%  10%  90% 100%
##   10   15   55   60
```

- *ecart type*

```
sd(v, na.rm = TRUE)
```

```
## [1] 18.61899
```

- *coeff correlation entre vectors*

```
cor(v,w)
```

```
## [1] 0.9433573
```

- *vector tri ordre croissant*

```
sort(v)
```

```
## [1] 10 20 30 30 50 60
```

On peut trier par ordre décroissant en utilisant l'option decreasing=TRUE

- *vector tri ordre decroissant*

```
sort(v, decreasing = TRUE)
```

```
## [1] 60 50 30 30 20 10
```

On peut trier selon plusieurs variables

- *donne pointer tri sur elements*

```
order(w)
```

```
## [1] 2 1 3 4 6 5
```

- vecteur des rangs

```
rank(w, ties.method='min')
```

```
## [1] 2 1 3 3 6 5
```

- vecteur des rangs

```
rank(w, ties.method='max')
```

```
## [1] 2 1 4 4 6 5
```

- valeurs max membre a membre

```
pmax(v,w,u)
```

```
## [1] 20 20 31 32 62 51
```

- valeurs min membre a membre

```
pmin(v,w,u)
```

```
## [1]  5  5  5 30 60 49
```

- sommes cumulees

```
cumsum(v)
```

```
## [1]  10  30  60  90 150 200
```

- produits successifs

```
cumprod(v)
```

```
## [1] 1.00e+01 2.00e+02 6.00e+03 1.80e+05 1.08e+07 5.40e+08
```

- maximum entre membre

```
cummax(w)
```

```
## [1] 20 20 31 31 61 61
```

- idem avec min

```
cummin(w)
```

```
## [1] 20 10 10 10 10 10
```

Logique Booléenne Dans R il est possible d'effectuer des comparaisons ou des tests qui vont sortir la valeur TRUE si vrai et FALSE si faux

Voici les opérateur que l'on peut utiliser :

1) strictement supérieur

2) < strictement inférieur

3) >= supérieur ou égal

4) <= inférieur ou égal

5) != différent

6) == égal (oui il faut mettre == et pas =)

- TRUE

```
a<-1
b<-2
(a==1)
```

```
## [1] TRUE
```

- FALSE

```
(a == b)
```

```
## [1] FALSE
```

- *TRUE*

```
(a <= b)
```

```
## [1] TRUE
```

```
A <- c(TRUE,TRUE,FALSE,FALSE)
B <- c(TRUE,FALSE,TRUE,FALSE)
```

Les connecteurs usuels nommés NON, ET, OU s'écrivent respectivement !, &, |. Il est très fortement conseillé d'utiliser des parenthèses pour séparer ces comparaisons logiques. Voici quelques exemples d'utilisation :

- *Table de verite de "et"*

```
A & B
```

```
## [1]  TRUE FALSE FALSE FALSE
```

- *table de verite de "ou"*

```
A | B
```

```
## [1]  TRUE  TRUE  TRUE FALSE
```

- *non-A*

```
 ! A
```

```
## [1] FALSE FALSE  TRUE  TRUE
```

- *table verite ou exclusif*

```
xor(A,B)
```

```
## [1] FALSE  TRUE  TRUE FALSE
```

- *table de l'implication A==>B*

```
!A|B
```

```
## [1]  TRUE FALSE  TRUE  TRUE
```

- vector compose de logical

```
str(A)
```

```
##  logi [1:4] TRUE TRUE FALSE FALSE
```

- stocker le resultat d'un test

```
c <- (a > b)
```

- un vector

```
v <- c(10,20,30,30,60,50)
```

- vecteur resultant du test

```
t <- (v > 30)
```

- membre a membre

```
t
```

```
## [1] FALSE FALSE FALSE FALSE  TRUE  TRUE
```

- on ne garde que les membres avec expression TRUE

```
w <- v[(v>30)]
w #avec expression True
```

```
## [1] 60 50
```

WHICH récupère ou définit un attribut c'est aussi pour la manipulation et la sélection de données

- trouve les indices ou membre egal a 30

```
which(v == 30)
```

```
## [1] 3 4
```

- trouve les indices ou

```
which(v == max(v))
```

```
## [1] 5
```

- idem mais recherche min

```
which(v == min(v))
```

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

- transformation en vecteur 1,0

```
s <- 1*t
s
```

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

- un vector

```
v <- c(10,20,70,30,60,50)
all(v > 5) # ?toutes les val sont sup
```

```
## [1] TRUE
```

- ?une valeur inf a 5

```
any(v < 5)
```

```
## [1] FALSE
```

Merci pour votre attention