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

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)

réaliser un calcul simple

10**2 #carre

sqrt(100) #racine

100**(1/2) #puissance

sqrt(100) #racine

pi # pi = 3.1416

cos(pi) #cosinus

sin(pi/2) # sinus

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$x$), $sin(pix)$et tan(pi*x).

exp(1) #exponentielle

log(1) #log neperien

round(2.566) #arrondi à un entier

round(pi,2) # arrondi 2 chiffres apres ,

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

v <- c(10,20,30) # un vector

length(v) # longueur du vector

2*v+1 # sur chaque composante du vector

v**2 # carre de chaque composante

log(v) # log de chaque composante

w <- c(1,2,3) # un autre vector

v-w # soustraction membre a membre

v*w # multiplication membre a membre

v/w # division membre a membre

v%*%w # produit scalaire

sum(v) # =somme

mean(v) # moyenne

min(v) # =minimum

max(v) # =maximum

sd(v) # =ecart type

median(v) # =medianne

autres type de function :

- La fonction REP pour indiquer la valeur à répéter

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

u <- c(1,2,3,4,5,6,7,8) # un autre vector u[2] # deuxieme composante

u[3:5] # nouveau vector

u[8] <- 80 # affectation une composante

u[1:5] <- 1 # affectation 5 composantes u

v <- c(10,20,30,30,60,50) # jeux avec les vectors

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.

w <- c(20,10,31,31,61,51) # un autre vector

u <- c(5 ,5 ,5 ,32,62,49) # un autre vector

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

str(v) # jeter un oeil sur les data n

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

sum(is.na(v)) # nb de valeurs manquantes

v_ <- c(NA,v,NA,NA) # un vecteur avec 3 valeur manquantes

sum(is.na(v_)) # nb valeurs manquantes

range(v) # min et max du vector

range(v_) # min et max du vector ECHEC !

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

range(v_ , na.rm = TRUE) # sans tenir compte des NA

quantile(v) # quartiles de v

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

sd(v, na.rm = TRUE) # ecart type

cor(v,w) # coeff correlation entre vectors

sort(v) # vector tri ordre croissant

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

sort(v, decreasing = TRUE) # vector tri ordre decroissant

On peut trier selon plusieurs variables

order(w) # donne pointer tri sur elements

rank(w, ties.method=“min”) # vecteur des rangs

rank(w, ties.method=“max”) # vecteur des rangs

pmax(v,w,u) # valeurs max membre a membre

pmin(v,w,u) # valeurs min membre a membre

cumsum(v) # sommes cumulees

cumprod(v) # produits successifs

cummax(w) #maximum entre membre

cummin(w) # idem avec min

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 :

strictement supérieur

< strictement inférieur

>= supérieur ou égal

<= inférieur ou égal

!= différent

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

a <- 1 b <- 2 (a == 1) # TRUE

(a == b) # FALSE

(a <= b) # 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 :

A & B # table de verite de "et"

A | B # table de verite de "ou"

! A # non-A

xor(A,B) # table verite ou exclusif

!A|B # table de l'implication A==>B

str(A) # vector compose de logical

c <- (a > b) # stocker le resultat d'un test

v <- c(10,20,30,30,60,50) # un vector t <- (v > 30) # vecteur resultant du test t # membre a membre w <- v[(v>30)] # on ne garde que les membres # avec expression TRUE

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

which(v == 30) # trouve les indices ou membre egal a 30

which(v == max(v)) # trouve les indices ou

which(v == min(v)) # idem mais recherche min

s <- 1*t # transformation en vecteur 1,0

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

any(v < 5) # ?une valeur inf a 5