Vectors

> Created by Woo Hyun (Ray) Hwang

Vectors

Scalars and vectors

Vectors are basic forms of data in R. Scalars of vectors with length of 1. use `typeof()` to check.

```
x <- c(1); is.vector(x)
print(pi, 6) #some scalars are built in
```

Declaration

Vector’s length needs to be declared beforehand.

```
# The fibonacci seq
f <- vector(length=10) #f is now a empty vector with length 10
f[1] <- 1; f[2] <- 1
for(i in 1:length(f)) f[i+2] <- f[i] + f[i+1]
f
```

To add or delete elements of a vector, we need to newly declare the vector.

```
x <- c(55, 5, 11, 12)
x <- c(x[1:3], 100, x[4]) #add 100 before 12 in x
x
```

Matrix

Matrix is a 2 dimensional array of 2 vectors.

```
M <- matrix(1:12, nrow=4, ncol=3) #matrix(data, nrow, ncol) #byrow=F (default)
M.1 <- matrix(1:12, 4, 3, byrow=T) #data will go by rows
```

Recycling

If data is insufficient, recycle from the start of data.

```
M.2 <- matrix(1:10, 4, 3) #will show warning
M.2
M.3 <- matrix(1:10, 4, 3, byrow=T) #still recycles by row
M.3
```

When calculating vectors, the shorter vector will be recycled to match the length of the longer vector.

```
c(1, 2, 4) + c(6, 0, 9, 20, 22) #1 and 2 is recycled
c(1, 2, 4, 1, 2) + c(6, 0, 9, 20, 22)
```

Indexing

Using a part of a vector, subsetting.

```
#the fibonacci seq made above
f[10]
f[3:10]
f[-c(1, 2)] #excluding 1, 2 elements
f[-1:-2] #same as above
f[c(2, 4, 6, 8, 10)] #can index using vector
```

```
#Using the length() function
z <- c(5, 12, 13)
z[1:(length(z)-1)] #excluding the last element
z[-length(z)] #same as above
```

```
#Creating useful vectors using operators
5:8
5:1
#Need to be aware of the order of operators
i <- 2
1:i-1 #calculated as (1:i)-1
1:(i-1) #calculated as 1:1
```

`seq()` and `rep()`

```
seq(0, 10, by=2.5) #seq(start, end, by)
seq(0, 10, length=10) #seq(start, end, length)
rep(0, times=10) #rep(factor, times)
rep(NA, 10)
rep(c(1, 2, 3), 2) #repeat vector (1, 2, 3) 2 times
rep(c(1, 2, 3), c(1, 2, 3)) #1 once, 2 twice, 3 three times
x <- c(5, 12, 13)
seq(x) #produces 1, 2, 3
x <- NULL
seq(x) #x is empty, does not repeat
```

NA and NULL

NA is not available (missing value), NULL is does not exist (no value)

```
x <- c(1, 2, NULL, 6)
x
mean(x) #calculates, as NULL is non-existent
y <- c(1, 2, NA, 6)
mean(y) #NA, as there is a missing value
mean(y, na.rm=T) #option na.rm=T removes NA
mean(y[!is.na(y)]) #index values that are not NA in y
u <- NULL
length(u) #0
v <- NA
length(v) #1
```

Some ways to delete NA

`is.na()` checks for NAs / `table(is.na())` gives frequency of NAs

`(, na.rm=T)` removes NAs for funcitons / `[is.na()] <- 0` changes NAs to 0

`library(dplyr); df %>% filter(is.na(df))` prints all data that is NAs

`library(dplyr); df %>% filter(!is.na(df))` prints all data that is not NAs

Using NULL: for repetitions

```
z <- NULL
for(i in 1:10) if(i %% 2 == 0) z <- c(z, i)
z
seq(2, 10, 2) #using seq for the example above
2*1:5 #simpler yet
```

If we use NA, then we need to remove NA

```
z <- NA
for(i in 1:10) if(i %% 2 == 0) z <- c(z, i)
z
```

Using the `purrr` package and the function `compact()` to delete NULL

```
library(purrr)
lst <- list("Moe", NULL, "Curly")
compact(lst) #deletes NULL
```

Using `discard()` to filter under conditions

```
lst <- list(NA, 0, NA, 1, 2)
lst %>% discard(is.na) #to discard characters use discard(is.character)
```

filtering

Filter to get only the data we want to use

```
z <- c(1,8,9,2,7,10,5,6,4,3)
z[z %% 2 == 1] #indexing only odd numbers
index <- (z %% 2 == 1) #creating index
z[index]
subset(z, z %% 2 == 1) #subset(data, condition)
x <- c(6, 1:3, NA, 12)
subset(x, x>5) #NA is not included
```

`ifelse()`

`ifelse(test, yes, no)` is the format

```
x <- c(-0.6, 0.2,-0.8, 1.6, 0.3,-0.8, 0.5, 0.7, 0.6,-0.3)
x.1 <- ifelse(x>0, x, 2*x)
rbind(x, x.1)
```

Using `if` is not efficient

```
x.1 <- x
for (i in 1:length(x)){
  if (x[i] > 0) x.1[i] <- x[i]
  else x.1[i] <- 2*x[i]
}
rbind(x, x.1)
```

`identical()`

Shows if two objects are identical.

```
x <- 1:3 #integer
y <- c(1, 2, 3) #numeric
z <- seq(1, 3, 1) #numeric
x == y #TRUE
identical(x, y) #FALSE
identical(y, z) #TRUE
all(x == y) #if all x = y: TRUE, for x to compare with y, x change to numeric
typeof(x)
typeof(y)
```

`names()`

Gives names to individual vectors.

```
era <- c(5, 4, 3, 4, 5, 6)
names(era) <- paste("y", 2001:2006, sep=".")
era
era[2]
era["y.2002"]
names(era) <- NULL #delete all names
```

Application: run lengths

```
set.seed(1)
z <- rbinom(20, size=1, prob=0.5) #binomial dist, n=20
z
n <- length(z)
a <- z[2:n]
b <- z[1:(n-1)]
d <- a-b #if there is run, then 0 / else 1 or -1
sum(d!=0)+1 #total runs
ind <- c(which(d!=0), n)
ind
c(ind[1], diff(ind))
run <- rle(z)
run$lengths
```

Random number generation

`set.seed(k)` to generate seed

`rnorm(n, mu, sigma)`: normal dist.

`runif(n, a, b)` : uniform dist.

`rexp(n, lambda)` : exponential dist.

`rbinom(n, k, p)` : binomial dist.

`rpoisson(n, mu)` : poisson dist.

`sample(x, n, replace, prob)` : random sample, replace=T for sampling with replacement, prob=k for weighted sampling