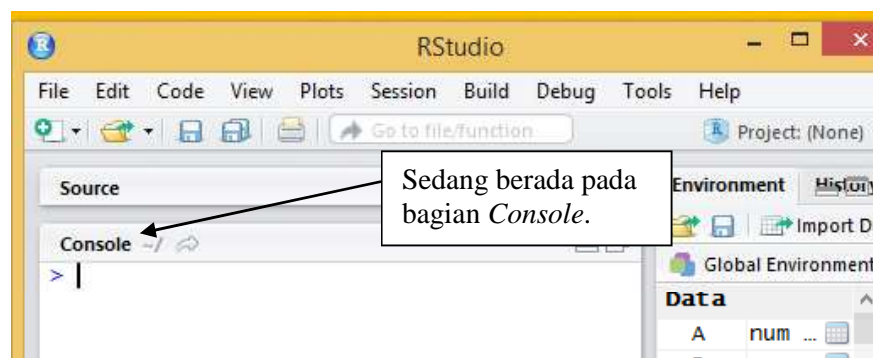


BAB 2

VEKTOR

2.1 Membuat Vektor

Pertama-tama, aktifkan *software* RStudio terlebih dahulu (Gambar 2.1). Lambang RStudio diperlihatkan pada Gambar 2.2. Pada Gambar 2.1 sedang berada pada bagian *Console*. Pada bagian *R Console*, perintah R dibuat. Terdapat tanda “>” pada *R Console*. Tanda tersebut disebut *prompt*. Di depan *prompt* perintah R ditulis.

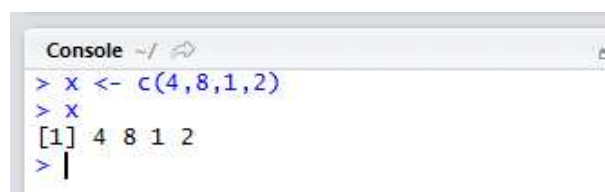


Gambar 2.1 Tampilan RStudio pada Bagian *Console*



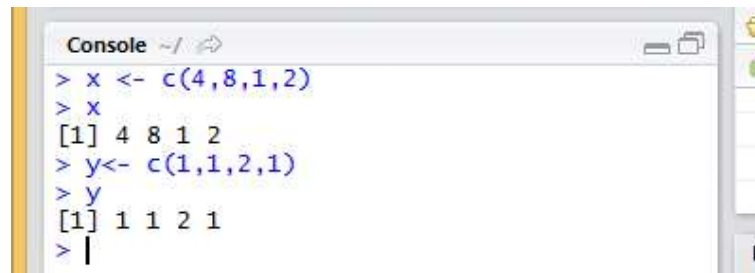
Gambar 2.2 Lambang RStudio

Berdasarkan Gambar 2.3, perintah R pada baris pertama `x <- c(4,8,1,2)` dapat diartikan menugaskan (*assign*) bilangan 4,8,1,2 ke vektor dengan nama `x`, kemudian perintah R pada baris kedua `x` menampilkan seluruh elemen dalam vektor `x`. Dalam membuat vektor, melibatkan fungsi `c()`. Suatu fungsi ditandai dengan tanda buka kurung “(” dan tutup kurung “)”. Huruf “c” merupakan singkatan dari “*concatenate*”, dalam bahasa Indonesia dapat berarti “menggabungkan”.



Gambar 2.3

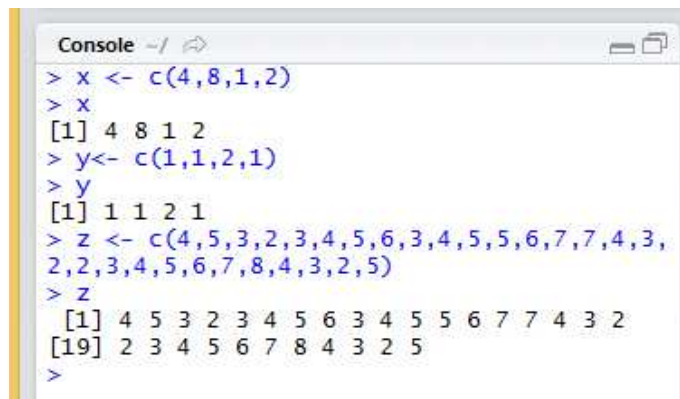
Pada Gambar 2.4, perintah R `y <- c(1,1,2,1)` dapat diartikan menugaskan (*assign*) bilangan 1,1,2,1 ke vektor dengan nama `y`, kemudian perintah R `y` menampilkan seluruh elemen dalam vektor `y`.



```
Console ~/\n> x <- c(4,8,1,2)\n> x\n[1] 4 8 1 2\n> y<- c(1,1,2,1)\n> y\n[1] 1 1 2 1\n> |
```

Gambar 2.4

2.2 Hasil Eksekusi Perintah R



```
Console ~/\n> x <- c(4,8,1,2)\n> x\n[1] 4 8 1 2\n> y<- c(1,1,2,1)\n> y\n[1] 1 1 2 1\n> z <- c(4,5,3,2,3,4,5,6,3,4,5,5,6,7,7,4,3,2,2,3,4,5,6,7,8,4,3,2,5)\n> z\n[1] 4 5 3 2 3 4 5 6 3 4 5 5 6 7 7 4 3 2\n[19] 2 3 4 5 6 7 8 4 3 2 5\n>
```

Gambar 2.5

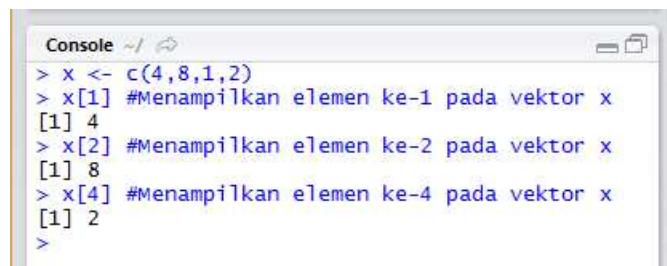
Perhatikan Gambar 2.5. Pada Gambar 2.5, terdapat tanda “[]”, yang merupakan hasil dari eksekusi perintah R. Perhatikan bahwa “[1]” berarti hasil atau *output* yang ditampilkan dimulai dari *output* yang ke-1. Sementara “[19]” berarti hasil atau *output* yang ditampilkan dimulai dari *output* yang ke-19.

2.3 Indeks Vektor

Indeks vektor dimulai dari 1. Berdasarkan Gambar 2.5, pada vektor `x`, diketahui:

- ⇒ Nilai 4 berada pada indeks ke-1 dari vektor `x`.
- ⇒ Nilai 8 berada pada indeks ke-2 dari vektor `x`.
- ⇒ Nilai 1 berada pada indeks ke-3 dari vektor `x`.
- ⇒ Nilai 2 berada pada indeks ke-4 dari vektor `x`.

2.4 Menampilkan Elemen Vektor berdasarkan Indeks

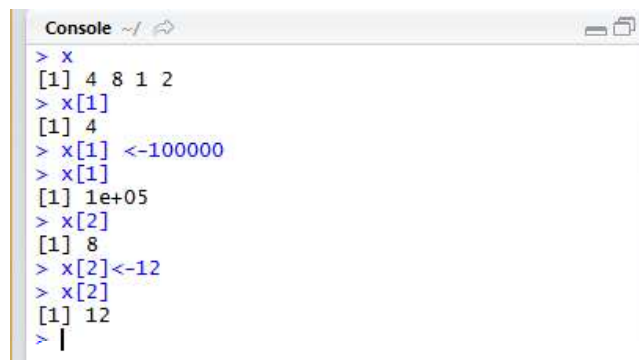


```
Console ~/
> x <- c(4,8,1,2)
> x[1] #Menampilkan elemen ke-1 pada vektor x
[1] 4
> x[2] #Menampilkan elemen ke-2 pada vektor x
[1] 8
> x[4] #Menampilkan elemen ke-4 pada vektor x
[1] 2
>
```

Gambar 2.6

Perhatikan Gambar 2.6. Pada Gambar 2.6, perintah R **x[1]** berarti menampilkan elemen pada indeks ke-1 dari vektor **x**, perintah R **x[4]** berarti menampilkan elemen pada indeks ke-4 pada vektor **x**, dan sebagainya. Pada Gambar 2.6, terdapat tanda “#”. Kalimat di depan tanda “#” dimaksudkan sebagai pemberi keterangan (*comment*).

2.5 Mengganti Elemen Vektor



```
Console ~/
> x
[1] 4 8 1 2
> x[1]
[1] 4
> x[1] <- 100000
> x[1]
[1] 1e+05
> x[2]
[1] 8
> x[2] <- 12
> x[2]
[1] 12
> |
```

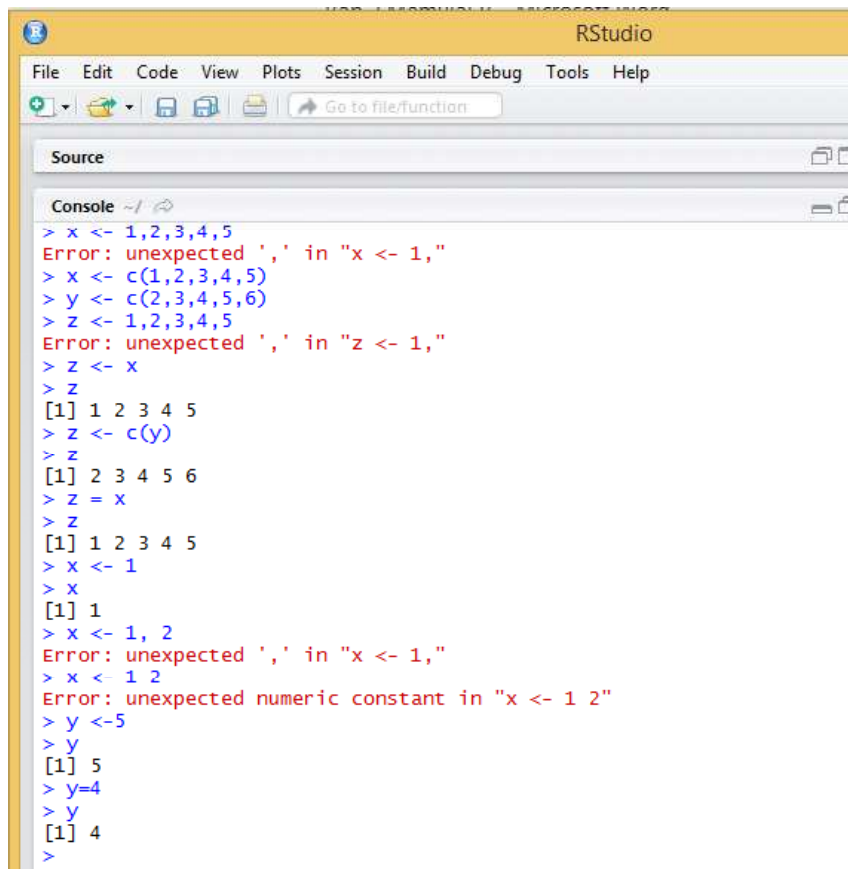
Gambar 2.7

Pada Gambar 2.7, diketahui nilai 4 berada pada indeks ke-1 di vektor **x**. Andaikan nilai 4 akan diganti dengan 100000. Perintah R untuk mengganti nilai 4 menjadi nilai 100000 adalah **x[1] <- 100000** (Perhatikan Gambar 2.7). Pada Gambar 2.7, *output* 1e+05 berarti $10^5 = 100000$. Selanjutnya diketahui nilai 8 berada pada indeks ke-2 di vektor **x**. Andaikan nilai 8 akan diganti dengan 12. Perintah R untuk mengganti nilai 8 menjadi nilai 12 adalah **x[2] <- 12** (Perhatikan Gambar 2.7).

2.6 Operator Penugasan

Perhatikan Gambar 2.8. Perintah R **x <- 1,2,3,4,5** memberikan pesan kesalahan. Sementara perintah R **x <- c(1,2,3,4,5)** tidak memberikan pesan kesalahan, begitu juga perintah R **y <- c(2,3,4,5,6)** tidak memberikan pesan kesalahan. Namun perintah R **z <- 1,2,3,4,5** memberikan pesan kesalahan. Perintah R **z <- y** dapat diartikan menugaskan seluruh elemen pada vektor **y** ke vektor **z**. Jadi, seluruh elemen

dari vektor **z** sekarang adalah seluruh elemen dari vektor **y**. Perhatikan bahwa perintah **R x <- 1** tidak memberikan pesan kesalahan, namun perintah **R x <- 1, 2** memberikan pesan kesalahan.



```

RStudio
File Edit Code View Plots Session Build Debug Tools Help
Source
Console
> x <- 1,2,3,4,5
Error: unexpected ',' in "x <- 1,"
> x <- c(1,2,3,4,5)
> y <- c(2,3,4,5,6)
> z <- 1,2,3,4,5
Error: unexpected ',' in "z <- 1,"
> z <- x
> z
[1] 1 2 3 4 5
> z <- c(y)
> z
[1] 2 3 4 5 6
> z = x
> z
[1] 1 2 3 4 5
> x <- 1
> x
[1] 1
> x <- 1, 2
Error: unexpected ',' in "x <- 1,"
> x <- 1 2
Error: unexpected numeric constant in "x <- 1 2"
> y <-5
> y
[1] 5
> y=4
> y
[1] 4
>

```

Gambar 2.8

Operator standar penugasan (*standard assignment operator*) dalam R adalah “<-”, meskipun terdapat beberapa operator penugasan seperti “->”, “=”, dan fungsi **assign()**. Norman Matloff (2009) menyatakan sebagai berikut.

“The standard assignment operator in R is <-. However, there are also ->, =, and even the assign() function.”

Perhatikan Gambar 2.9. Gambar 2.9 diberikan ilustrasi penggunaan operator penugasan “<-”, “->”, “=”, dan fungsi **assign()**. Perintah **R x <- c(1,2,3)** dapat diartikan menugaskan bilangan 1,2,3 ke vektor **x**. Perintah **R x -> y** dapat diartikan menugaskan bilangan-bilangan pada vektor **x** ke vektor **y**. Dengan kata lain, seluruh elemen dari vektor **y** adalah seluruh elemen dari vektor **x**. Perintah **R assign(“z”, y)** dapat diartikan menugaskan bilangan-bilangan pada vektor **y** ke vektor **z**. Dengan kata lain, seluruh elemen dari vektor **z** adalah seluruh elemen dari vektor **y**. Perintah **R assign(“a”,c(1,2,3,4,5,6,7,8))** dapat diartikan menugaskan bilangan 1,2,3,4,5,6,7,8 ke vektor **a**.

```

Source
Console ~/
> x <- c(1,2,3)
> x
[1] 1 2 3
> x -> y
> y
[1] 1 2 3
> assign("z",y)
> z
[1] 1 2 3
> assign("a",c(1,2,3,4,5,6,7,8))
> a
[1] 1 2 3 4 5 6 7 8
> b=a
> b
[1] 1 2 3 4 5 6 7 8
> |

```

Gambar 2.9

2.7 Berbagai Cara Menampilkan Elemen Vektor berdasarkan Indeks

Gambar 2.10 diberikan ilustrasi berbagai cara untuk menampilkan elemen suatu vektor. Perintah R `x <- c(4,5,2,1,9,8,5,6,10,13)` berarti menugaskan bilangan 4,5,2,1,9,8,5,6,10,13 ke vektor `x`. Sebagai contoh diketahui:

- ⇒ Elemen pada indeks ke-1 dari vektor `x` adalah 4.
- ⇒ Elemen pada indeks ke-2 dari vektor `x` adalah 5.
- ⇒ Elemen pada indeks ke-3 dari vektor `x` adalah 2.
- ⇒ Elemen pada indeks ke-10 dari vektor `x` adalah 13.

```

Console ~/
> x <- c(4,5,2,1,9,8,5,6,10,13)
> y <- c(1,2,6)
> z <- c(9,3,4,5)
> x
[1] 4 5 2 1 9 8 5 6 10 13
> x[4]
[1] 1
> x[c(1,3)]
[1] 4 2
> x[c(8,10)]
[1] 6 13
> x[-c(1,2,3,4,5)]
[1] 8 5 6 10 13
> x[3:7]
[1] 2 1 9 8 5
> x[-3]
[1] 4 5 1 9 8 5 6 10 13
> x[c(-3)]
[1] 4 5 1 9 8 5 6 10 13
> x[-3,1]
Error in x[-3, 1] : incorrect number of dimensions
> x[3,1]
Error in x[3, 1] : incorrect number of dimensions
> x[y]
[1] 4 5 8
> x[-y]
[1] 2 1 9 5 6 10 13
> x[z]
[1] 10 2 1 9
> z[-z]
[1] 9 3
>

```

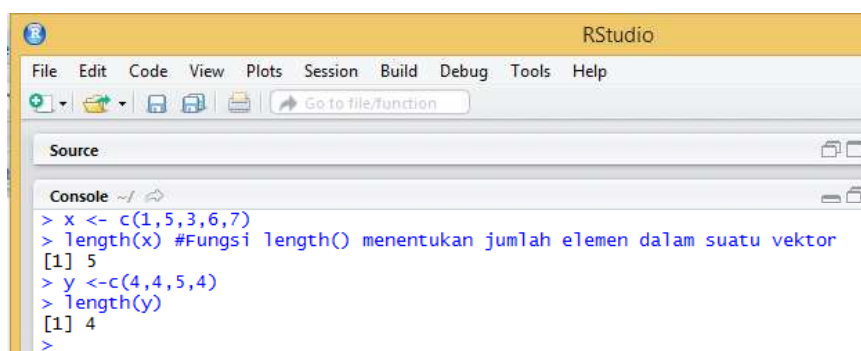
Gambar 2.10

Perhatikan bahwa:

- ⇒ Perintah R `x[4]` berarti menampilkan elemen dari vektor `x` pada indeks ke-4, yakni 1.
- ⇒ Perintah R `x[c(1,3)]` berarti menampilkan elemen dari vektor `x` pada indeks ke-1 dan indeks ke-3, yakni 4 dan 2.
- ⇒ Perintah R `x[c(8,10)]` berarti menampilkan elemen dari vektor `x` pada indeks ke-8 dan indeks ke-10, yakni 6 dan 13.
- ⇒ Perintah R `x[-c(1,2,3,4,5)]` berarti menampilkan elemen dari vektor `x` yang bukan pada indeks ke-1,2,3,4,5. Dengan kata lain, menampilkan elemen dari vektor `x` pada indeks ke-6,7,8,9,10, yakni 8,5,6,10,13.
- ⇒ Perintah R `x[3:7]` berarti menampilkan elemen dari vektor `x` mulai dari indeks ke-3 sampai dengan indeks ke-7. Dengan kata lain, menampilkan elemen dari vektor `x` pada indeks ke-3,4,5,6,7, yakni 2,1,9,8,5.
- ⇒ Perintah R `x[-3]` berarti menampilkan elemen dari vektor `x` yang bukan pada indeks ke-3. Dengan kata lain, menampilkan elemen dari vektor `x` pada indeks ke-1,2,4,5,6,7,8,9,10. Perhatikan bahwa perintah R `x[-3]` memberikan hasil yang sama dengan perintah R `x[-c(3)]`. Namun perintah R `x[-3,1]` dan perintah R `x[3,1]` memberikan pesan kesalahan.
- ⇒ Perintah R `x[y]` sama saja dengan `x[c(1,2,6)]`. Perhatikan bahwa 1, 2, dan 6 merupakan elemen dari vektor `y`.
- ⇒ Perintah R `x[-y]` sama saja dengan `x[-c(1,2,6)]` atau sama saja dengan `x[c(3,4,5,7,8,9,10)]`.
- ⇒ Perintah R `x[z]` sama saja dengan `x[c(9,3,4,5)]`. Perhatikan bahwa 9, 3, 4, dan 5 merupakan elemen dari vektor `z`. Perintah R `x[z]` berarti menampilkan elemen dari vektor `x` pada indeks ke-9,3,4,5.

2.8 Mengetahui Panjang Vektor dengan Fungsi `length()`

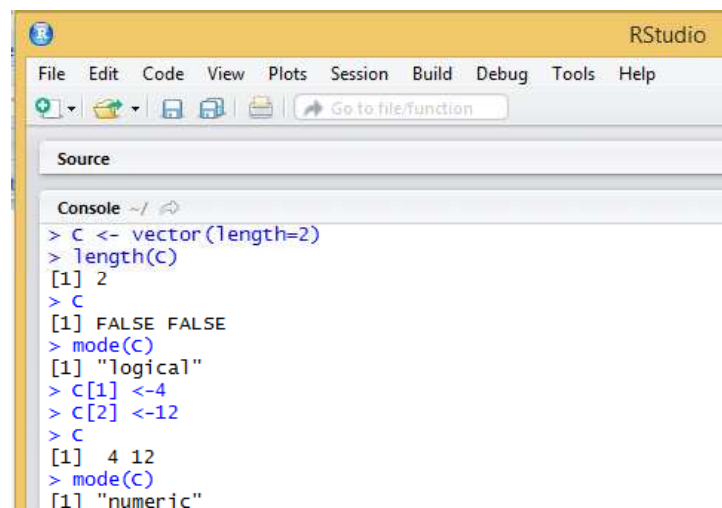
Untuk mengetahui panjang (*length*) suatu vektor, dapat digunakan fungsi `length()`. Perhatikan Gambar 2.11. Berdasarkan Gambar 2.11, diketahui panjang dari vektor `x` adalah 5, yang mana elemennya adalah 1, 5, 3, 6, dan 7. Panjang dari vektor `y` adalah 4, yang mana elemennya adalah 4, 4, 5, dan 4. Pada Gambar 2.11, digunakan fungsi `length()` untuk mengetahui panjang dari suatu vektor.



Gambar 2.11

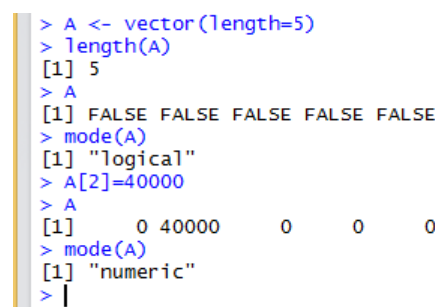
2.9 Membuat Vektor dengan Fungsi `vector()`

Fungsi `vector()` digunakan untuk membuat suatu vektor. Perhatikan Gambar 2.12. Berdasarkan Gambar 2.12, perintah `R C <- vector(length=2)` berarti membuat vektor `C` dengan panjang 2. Perintah `R length(C)` untuk mengetahui panjang vektor `C`, yang mana panjang vektor `C` adalah 2. Jika elemen dari vektor `C` ditampilkan, maka kedua elemen tersebut adalah `FALSE FALSE`. Secara bawaan (*default*), ketika suatu vektor dibuat dengan fungsi `vector()` dengan panjang tertentu, vektor tersebut berjenis logika dengan nilai elemen seluruhnya `FALSE`. Fungsi `mode()` digunakan untuk menentukan jenis data yang tersimpan dalam suatu vektor. Setelah vektor `C` dibuat dengan panjang 2, selanjutnya mengisi bilangan 4 pada indeks ke-1 dari vektor `C`, mengisi bilangan 12 pada indeks ke-2 dari vektor `C`. Setelah itu dicek kembali jenis dari vektor `C` dengan fungsi `mode()`. Jenis vektor `C` sekarang adalah *numeric*, karena elemen dari vektor `C` saat ini adalah angka, yakni 4 dan 12.



```
> C <- vector(length=2)
> length(C)
[1] 2
> C
[1] FALSE FALSE
> mode(C)
[1] "logical"
> C[1] <-4
> C[2] <-12
> C
[1] 4 12
> mode(C)
[1] "numeric"
```

Gambar 2.12



```
> A <- vector(length=5)
> length(A)
[1] 5
> A
[1] FALSE FALSE FALSE FALSE FALSE
> mode(A)
[1] "logical"
> A[2]=40000
> A
[1] 0 40000 0 0 0
> mode(A)
[1] "numeric"
> |
```

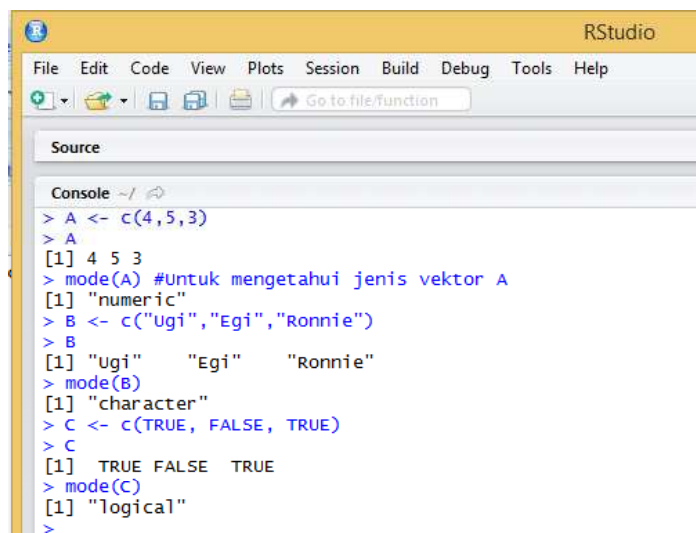
Gambar 2.13

Berdasarkan Gambar 2.13, perintah `R A<- vector(length=5)` berarti membuat vektor `A` dengan panjang 5. Perintah `R length(A)` untuk mengetahui panjang vektor `A`, yang mana panjang vektor `A` adalah 5. Jika elemen dari vektor `A` ditampilkan, maka kelima elemen tersebut adalah `FALSE FALSE FALSE FALSE FALSE`. Secara bawaan (*default*), ketika suatu vektor dibuat dengan fungsi `vector()` dengan

panjang tertentu, vektor tersebut berjenis logika dengan nilai elemen seluruhnya FALSE. Fungsi *mode()* digunakan untuk menentukan jenis data yang tersimpan dalam suatu vektor. Setelah vektor **A** dibuat dengan panjang 5, selanjutnya mengisi bilangan 40000 pada indeks ke-2 dari vektor **A**. Untuk indeks ke-1,3,4,5 tidak diisi bilangan. Selanjutnya ditampilkan seluruh nilai dari vektor **A**, yakni 0, 40000, 0, 0, dan 0. Secara bawaan (*default*) nilai vektor **A** pada indeks ke-1,3,4,5, diisi dengan 0. Selanjutnya diperiksa kembali jenis dari vektor **A** dengan fungsi *mode()*. Jenis vektor **A** sekarang adalah *numeric*, karena elemen dari vektor **A** saat ini adalah angka, yakni 0, 40000, 0, 0, dan 0.

2.9 Fungsi *mode()* Untuk Mengetahui Jenis Data Suatu Vektor

Untuk memeriksa jenis data suatu vektor, dapat digunakan fungsi *mode()*. Perhatikan ilustrasi Gambar 2.14. Berdasarkan Gambar 2.14, diketahui jenis data dari vektor **A** adalah *numeric*. Data dari vektor **A** adalah angka, yakni 4, 5, dan 3. Jenis data dari vektor **B** adalah *character*, dengan data “Ugi”, “Egi”, dan “Ronnie”. Setiap elemen dari vektor berjenis *character* diapit tanda petik ganda “ ”. Jenis data dari vektor **C** adalah *logical*. Data pada jenis *logical* hanya ada dua, yakni TRUE dan FALSE.



```

RStudio
File Edit Code View Plots Session Build Debug Tools Help
Source
Console ~/
> A <- c(4,5,3)
> A
[1] 4 5 3
> mode(A) #Untuk mengetahui jenis vektor A
[1] "numeric"
> B <- c("Ugi","Egi","Ronnie")
> B
[1] "Ugi" "Egi" "Ronnie"
> mode(B)
[1] "character"
> C <- c(TRUE, FALSE, TRUE)
> C
[1] TRUE FALSE TRUE
> mode(C)
[1] "logical"
>

```

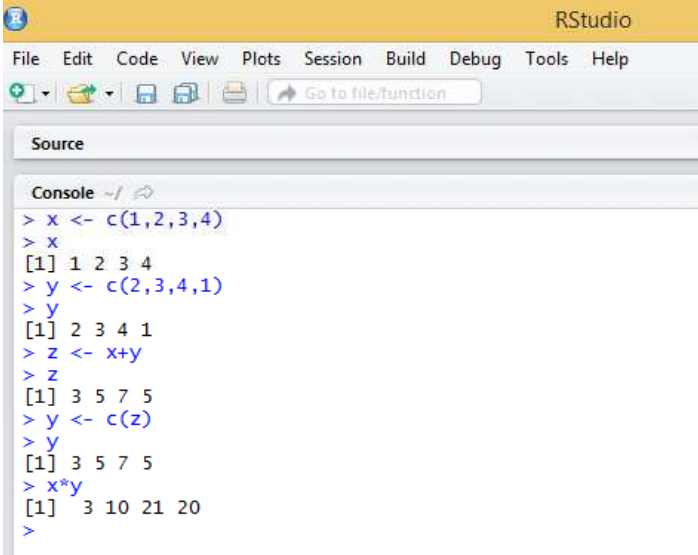
Gambar 2.14

2.10 Operator Penjumlahan +, Pengurangan -, Perkalian *, Pembagian /, Pangkat ^, dan Sisa %%

Perhatikan Gambar 2.15. Pada Gambar 2.15, diketahui elemen-elemen dari vektor **z** adalah 3, 5, 7, 5, yang diperoleh dari perintah R **z <- x+y**. Diketahui:

- ⇒ Elemen pertama dari vektor **z** adalah 3, yang diperoleh dari hasil jumlah antara elemen pertama dari vektor **x**, yakni 1, dan elemen pertama dari vektor **y**, yakni 2.
- ⇒ Elemen kedua dari vektor **z** adalah 5, yang diperoleh dari hasil jumlah antara elemen kedua dari vektor **x**, yakni 2, dan elemen kedua dari vektor **y**, yakni 3.

- ⇒ Elemen ketiga dari vektor **z** adalah 7, yang diperoleh dari hasil jumlah antara elemen ketiga dari vektor **x**, yakni 3, dan elemen ketiga dari vektor **y**, yakni 4.
- ⇒ Elemen keempat dari vektor **z** adalah 5, yang diperoleh dari hasil jumlah antara elemen keempat dari vektor **x**, yakni 4, dan elemen keempat dari vektor **y**, yakni 1.



```

RStudio
File Edit Code View Plots Session Build Debug Tools Help
Source
Console
> x <- c(1,2,3,4)
> x
[1] 1 2 3 4
> y <- c(2,3,4,1)
> y
[1] 2 3 4 1
> z <- x+y
> z
[1] 3 5 7 5
> y <- c(z)
> y
[1] 3 5 7 5
> x*y
[1] 3 10 21 20
>

```

Gambar 2.15

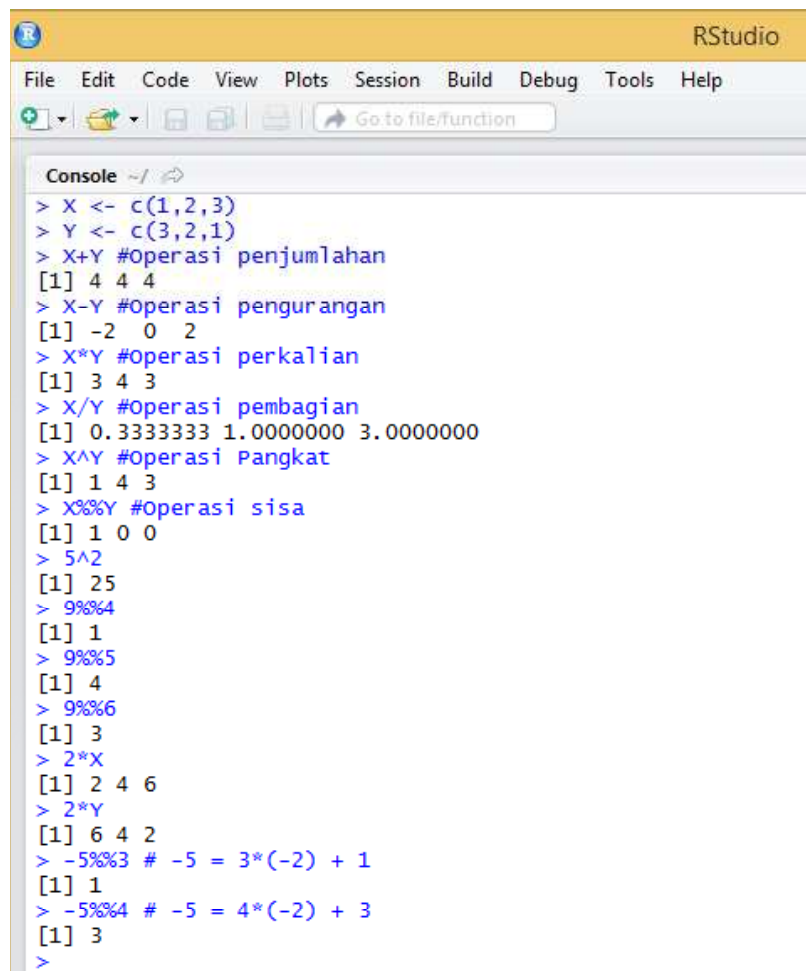
Berdasarkan Gambar 2.15, perintah R **y <- c(z)** dapat diartikan menugaskan seluruh elemen dari vektor **z** ke vektor **y**. Dengan kata lain, sekarang seluruh elemen pada vektor **y** merupakan seluruh elemen pada vektor **z**. Perintah R **x*y** dapat diartikan:

- ⇒ Mengalikan elemen pertama pada vektor **x** dengan elemen pertama pada vektor **y** ($1 \times 3 = 3$).
- ⇒ Mengalikan elemen kedua pada vektor **x** dengan elemen kedua pada vektor **y** ($2 \times 5 = 10$).
- ⇒ Mengalikan elemen ketiga pada vektor **x** dengan elemen ketiga pada vektor **y** ($3 \times 7 = 21$).
- ⇒ Mengalikan elemen keempat pada vektor **x** dengan elemen keempat pada vektor **y** ($4 \times 5 = 20$).

Gambar 2.16 diberikan berbagai contoh perhitungan. Berdasarkan Gambar 2.16, diketahui:

- ⇒ Elemen dari vektor **X** adalah 1, 2, dan 3. Elemen dari vektor **Y** adalah 3, 2, dan 1.
- ⇒ Pada Gambar 2.16, terdapat perintah R **X^Y**, yang berarti $1^3, 2^2, 3^1$ atau 1, 4, 3.

Hasil dari perintah R **9%%4** adalah 1 ($9 = 4 * 2 + 1$), hasil dari perintah R **9%%5** adalah 4 ($9 = 5 * 1 + 4$), hasil dari perintah R **-5%%3** adalah 1 ($-5 = 3 * -2 + 1$), dan hasil dari perintah R **-5%%4** adalah 3 ($-5 = 4 * -2 + 3$).



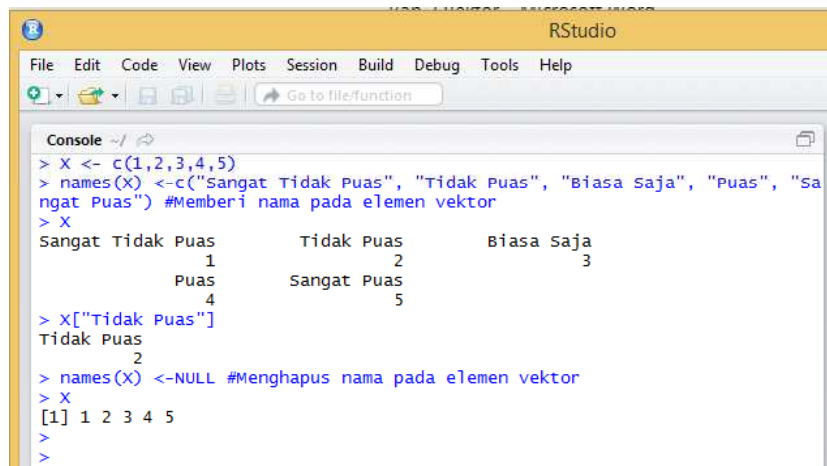
```
RStudio
File Edit Code View Plots Session Build Debug Tools Help
Go to file/function

Console
> X <- c(1,2,3)
> Y <- c(3,2,1)
> X+Y #Operasi penjumlahan
[1] 4 4 4
> X-Y #Operasi pengurangan
[1] -2 0 2
> X*Y #Operasi perkalian
[1] 3 4 3
> X/Y #Operasi pembagian
[1] 0.3333333 1.0000000 3.0000000
> X^Y #Operasi Pangkat
[1] 1 4 3
> X%%Y #Operasi sisa
[1] 1 0 0
> 5^2
[1] 25
> 9%%4
[1] 1
> 9%%5
[1] 4
> 9%%6
[1] 3
> 2*X
[1] 2 4 6
> 2*Y
[1] 6 4 2
> -5%%3 # -5 = 3*(-2) + 1
[1] 1
> -5%%4 # -5 = 4*(-2) + 3
[1] 3
>
```

Gambar 2.16

2.11 Memberi Nama pada Elemen Vektor dengan Fungsi *names()*

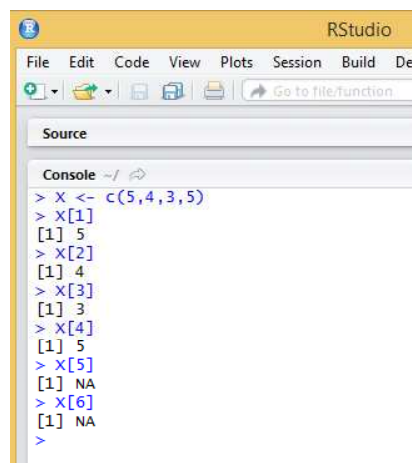
Elemen dalam vektor juga dapat diberi nama dengan fungsi *names()*. Perhatikan ilustrasi pada Gambar 2.17. Berdasarkan Gambar 2.17, digunakan fungsi *names()* untuk memberi nama pada elemen vektor **X**. Elemen vektor **X** pada indeks ke-1diberi nama “Sangat Tidak Puas”, elemen vektor **X** pada indeks ke-2diberi nama “Tidak Puas”, elemen vektor **X** pada indeks ke-3diberi nama “Biasa Saja”, elemen vektor **X** pada indeks ke-4diberi nama “Puas”, dan elemen vektor **X** pada indeks ke-5diberi nama “Sangat Puas”. Kemudian perhatikan perintah R *names(X) <- NULL* bermaksud untuk menghapus nama pada elemen di vektor **X**.



Gambar 2.17

2.12 Tidak Tersedia (*Not Available* (NA))

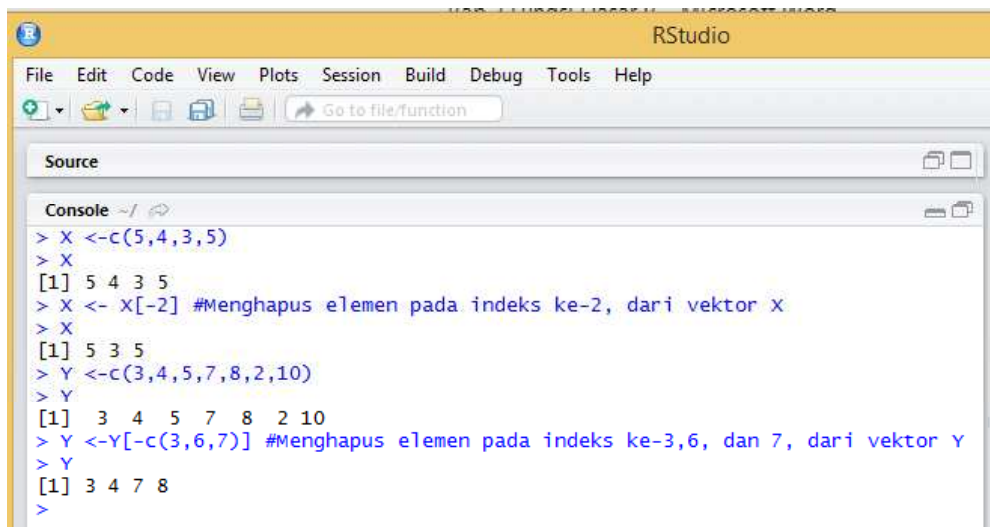
Perhatikan Gambar 2.18. Perintah R `X <- c(5,4,3,5)` berarti menugaskan bilangan 5, 4, 3 dan 5 ke vektor `X`. Nilai pada indeks ke-1 dari vektor `X` adalah 5, nilai pada indeks ke-2 dari vektor `X` adalah 4, nilai pada indeks ke-3 dari vektor `X` adalah 3, nilai pada indeks ke-4 dari vektor `X` adalah 5, nilai pada indeks ke-5 dari vektor `X` adalah tidak ada atau tidak tersedia atau *not available* (disingkat “NA”), nilai pada indeks ke-6 dari vektor `X` adalah tidak ada atau tidak tersedia atau *not available* (disingkat “NA”).



Gambar 2.18

2.13 Menghapus Elemen Vektor

Perhatikan Gambar 2.19. Perintah R `X <- c(5,4,3,5)` berarti menugaskan bilangan 5, 4, 3 dan 5 ke vektor `X`. Andaikan nilai 4 pada indeks ke-2 dari vektor `X` ingin dihapus, sehingga elemen dari vektor `X` sekarang adalah 5, 3, dan 5. Maka perintah R untuk menghapus nilai 4 pada indeks ke-2 dari vektor `X` adalah `X <- X[-2]`. Perintah R `X <- X[-2]` dapat juga diartikan menugaskan bilangan selain pada indeks ke-2, ke vektor `X`. Sehingga elemen pada vektor `X` saat ini adalah 5, 3, dan 5. Selanjutnya perintah R `Y <- Y[-c(3,6,7)]` berarti menghapus elemen dari vektor `Y` pada indeks ke-3, 6, dan 7.



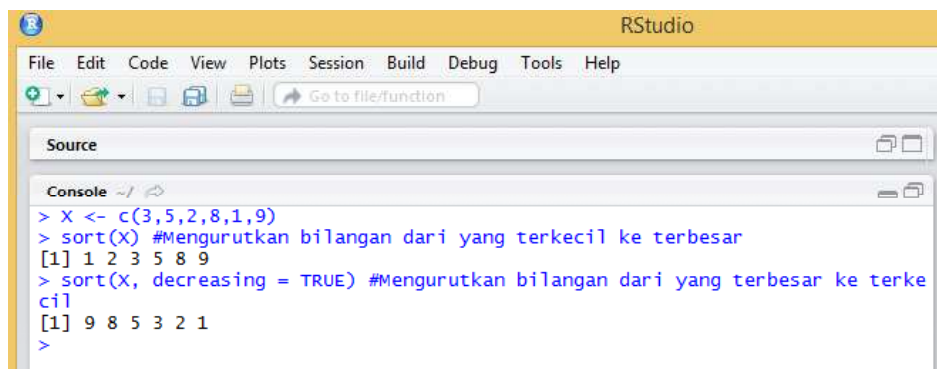
Gambar 2.19

BAB 3

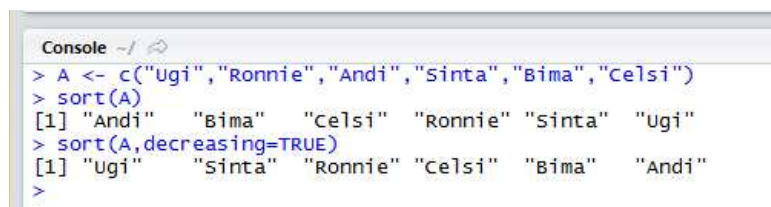
FUNGSI DASAR R

3.1 Mengurutkan Elemen Vektor dengan Fungsi *sort()*

Fungsi *sort()* dapat digunakan untuk mengurutkan elemen vektor. Gambar 3.1 diberikan ilustrasi mengurutkan bilangan dari yang terkecil ke terbesar, dan sebaliknya dari yang terbesar ke terkecil. Sementara pada Gambar 3.2, mengurutkan elemen vektor berjenis *character*, dimulai dari huruf abjad yang terkecil ke terbesar ($A \rightarrow Z$), dan sebaliknya dari huruf abjad yang terbesar ke terkecil ($Z \rightarrow A$).



Gambar 3.1



Gambar 3.2

3.2 Menghitung Selisih dari Data Berurutan dalam Vektor dengan Fungsi *diff()*

Misalkan diberikan data 100, 50, 70, 80, 60. Misalkan dilakukan perhitungan sebagai berikut.

- $\Rightarrow 50 - 100 = -50$
- $\Rightarrow 70 - 50 = 20$
- $\Rightarrow 80 - 70 = 10$
- $\Rightarrow 60 - 80 = -20$

Sehingga hasil akhirnya adalah $-50, 20, 10, -20$. Perhatikan Gambar 3.3. Perintah R ***diff(X)*-1*** dimaksudkan untuk melakukan perhitungan sebagai berikut.

- ⇒ $100 - 50 = 50$
- ⇒ $50 - 70 = -20$
- ⇒ $70 - 80 = -10$
- ⇒ $80 - 60 = 20$



```

> x <- c(100,50,70,80,60)
> diff(x)
[1] -50  20  10 -20
> diff(x)*-1
[1]  50 -20 -10  20
>

```

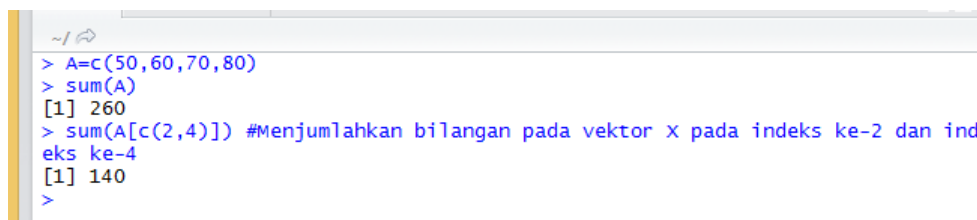
Gambar 3.3

3.3 Menjumlahkan Seluruh Bilangan dalam Vektor dengan Fungsi *sum()*

Fungsi *sum()* dapat digunakan untuk menjumlahkan seluruh bilangan dalam suatu vektor. Misalkan bilangan 50, 60, 70, dan 80 ditugaskan ke vektor **A**. Maka jumlah dari seluruh bilangan dalam vektor **A** adalah 260.

$$50 + 60 + 70 + 80 = 260$$

Perhatikan Gambar 3.4. Pada Gambar 3.4, perintah R *sum(A[c(2,4)])* berarti menjumlahkan bilangan dalam vektor **A** pada indeks ke-2 dan indeks ke-4.



```

> A=c(50,60,70,80)
> sum(A)
[1] 260
> sum(A[c(2,4)]) #Menjumlahkan bilangan pada vektor x pada indeks ke-2 dan indeks ke-4
[1] 140
>

```

Gambar 3.4

3.4 Menghitung Akar Pangkat Dua dalam Vektor dengan Fungsi *sqrt()*

Fungsi *sqrt()* dapat digunakan untuk menghitung akar pangkat dua dari suatu bilangan dalam vektor. Perhatikan ilustrasi pada Gambar 3.5. Perintah *sqrt(B)* berarti $\sqrt{4}, \sqrt{9}, \sqrt{16}, \sqrt{25}$ atau 2,3,4,5. Selanjutnya perintah R *C^(1/3)* berarti menghitung akar pangkat tiga dari bilangan-bilangan pada vektor **C**, yakni $\sqrt[3]{8}, \sqrt[3]{27}, \sqrt[3]{64}$ atau 2, 3, 4.

```

> B <- c(4,9,16,25)
> sqrt(B) #Menghitung akar pangkat dua untuk setiap bilangan di vektor B
[1] 2 3 4 5
> C <- c(8,27,64)
> C^(1/3) #Menghitung akar pangkat tiga untuk setiap bilangan di vektor C
[1] 2 3 4
>

```

Gambar 3.5

3.5 Fungsi *max()* dan *min()* untuk Menentukan Nilai Maksimum dan Minimum dalam Vektor

Fungsi *max()* dan *min()* dapat digunakan untuk menentukan nilai maksimum dan minimum dalam vektor. Misalkan bilangan 10,25,90, 75, 95, 57 ditugaskan ke vektor A. Nilai maksimum bilangan di vektor A adalah 95, sementara nilai minimum bilangan di vektor A adalah 10. Perhatikan Gambar 3.6.

```

> A=c(10,25,90,75,95,57)
> max(A)
[1] 95
> min(A)
[1] 10
>

```

Gambar 3.6

Fungsi *max()* dan *min()* juga dapat digunakan untuk vektor berjenis *character*. Perhatikan ilustrasi pada Gambar 3.7.

```

> B <- c("Nanas","Jeruk","Anggur","Stroberi","Markisa")
> max(B) #Hasilnya adalah stroberi, dikarenakan huruf "s" adalah huruf awal kata paling besar dari nama buah di vektor B
[1] "stroberi"
> min(B) #Hasilnya adalah anggur, dikarenakan huruf "A" adalah huruf awal kata paling kecil dari nama buah di vektor B
[1] "Anggur"
>

```

Gambar 3.7

3.6 Fungsi *exp()* (Eksponensial)

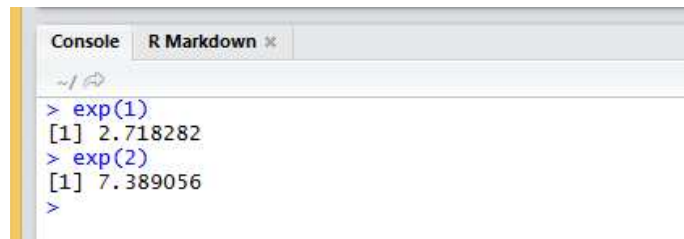
Exp merupakan singkatan dari *exponential* atau eksponensial. Nilai dari eksponensial adalah 2,71828182845...

$$\exp = 2,71828182845$$

$$\exp^1 = 2,71828182845^1 = 2,71828182845$$

$$\exp^2 = 2,71828182845^2 = 7,389056096$$

Berikut diberikan contoh penggunaan fungsi *exp()* dalam R (Gambar 3.8).



```
Console R Markdown x
> exp(1)
[1] 2.718282
> exp(2)
[1] 7.389056
>
```

Gambar 3.8

3.7 *pi* atau π

Pi atau π bernilai 3,141593 ... Berikut diberikan contoh penggunaan dari *pi* dalam R (Gambar 3.9).



```
Source
Console R Markdown x
> pi
[1] 3.141593
> pi^2
[1] 9.869604
>
```

Gambar 3.9

3.8 Menentukan Jumlah Angka di belakang Koma dengan Fungsi *options()*

Nilai pi adalah 3.141593... Misalkan hanya ingin ditampilkan 2 digit angka di belakang koma dari nilai pi. Gambar 3.10 ilustrasi dalam R menggunakan fungsi *options()* untuk menampilkan hanya 2 digit angka di belakang koma dari bilangan pi.



```
Console R Markdown x
> pi
[1] 3.141592654
> options(digits=3) #Menetapkan 2 angka di belakang koma
> pi
[1] 3.14
> options(digits=5) #Menetapkan 4 angka di belakang koma
> pi
[1] 3.1416
>
```

Gambar 3.10

3.9 Menampilkan Serangkaian Bilangan dengan Fungsi *seq()* dan Pengulangan Serangkaian Bilangan dengan Fungsi *rep()*

Fungsi *seq()* ("sequence" atau serangkaian) dapat digunakan untuk menampilkan serangkaian bilangan. Misalkan ingin ditampilkan bilangan dari 1 sampai 11, dengan jarak 2, yakni 1, 3, 5, 7, 9, 11. Gambar 3.11 digunakan fungsi *seq()* untuk menampilkan bilangan tersebut.


```

Console ~/
> seq(from=1, to=11, by=2)
[1] 1 3 5 7 9 11
> |

```

Gambar 3.11

Selanjutnya misalkan ingin ditampilkan bilangan dari 1 sampai 2, dengan jarak 0,2, yakni 1, 1,2, 1,4, 1,6, 1,8, dan 2. Gambar 3.12 digunakan fungsi *seq()* untuk menampilkan bilangan tersebut.

```

Console ~/
> seq(from=1, to=2, by=0.2)
[1] 1.0 1.2 1.4 1.6 1.8 2.0
>

```

Gambar 3.12

Misalkan hanya ingin ditampilkan 4 bilangan, dimulai dari 2 kemudian 2,5, 3 dan 3,5, dengan jarak adalah 0,5. Maka perintah R untuk menampilkan bilangan-bilangan tersebut adalah **seq(from=2, by=0.5, length.out=4)**. Kemudian misalkan ingin ditampilkan 6 bilangan, dimulai dari 2 kemudian 2,5, 3, 3,5, 4, dan 4,5, dengan jarak adalah 0,5. Maka perintah R untuk menampilkan bilangan-bilangan tersebut adalah **seq(from=2, by=0.5, length.out=6)**. Perhatikan Gambar 3.13.

```

Console ~/
> seq(from=2, by=0.5, length.out=4)
[1] 2.0 2.5 3.0 3.5
> seq(from=2, by=0.5, length.out=6)
[1] 2.0 2.5 3.0 3.5 4.0 4.5
> length(seq(from=2, by=0.5, length.out=4))
[1] 4
> length(seq(from=2, by=0.5, length.out=6))
[1] 6
>

```

Gambar 3.13

Selanjutnya fungsi *seq()* dan *rep()* akan digunakan secara bersama, untuk menampilkan bilangan 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3. Fungsi *rep()* (“repeat” atau mengulangi) di sini digunakan untuk melakukan pengulangan rangkaian bilangan 1, 2, 3 sebanyak 3 kali. Perhatikan Gambar 3.14.

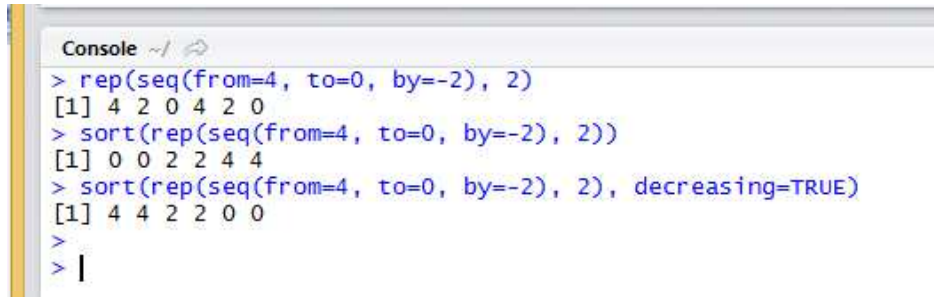
```

Console ~/
> rep(1:3,4) #Menampilkan bilangan 1, 2, 3, yang diulang sebanyak 4 kali
[1] 1 2 3 1 2 3 1 2 3 1 2 3
> sort(rep(1:3,4))
[1] 1 1 1 1 2 2 2 2 3 3 3 3
> sort(rep(1:3,4), decreasing=TRUE)
[1] 3 3 3 3 2 2 2 2 1 1 1 1
> |

```

Gambar 3.14

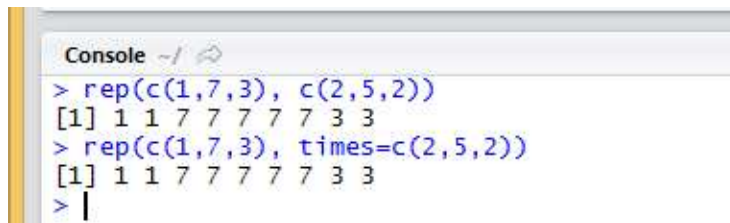
Misalkan akan ditampilkan bilangan 4, 2, 0, 4, 2, 0. Perhatikan Gambar 3.15.



```
Console ~/
> rep(seq(from=4, to=0, by=-2), 2)
[1] 4 2 0 4 2 0
> sort(rep(seq(from=4, to=0, by=-2), 2))
[1] 0 0 2 2 4 4
> sort(rep(seq(from=4, to=0, by=-2), 2), decreasing=TRUE)
[1] 4 4 2 2 0 0
>
> |
```

Gambar 3.15

Misalkan lagi akan ditampilkan bilangan 1, 1, 7, 7, 7, 7, 7, 3, 3. Perhatikan Gambar 3.16.

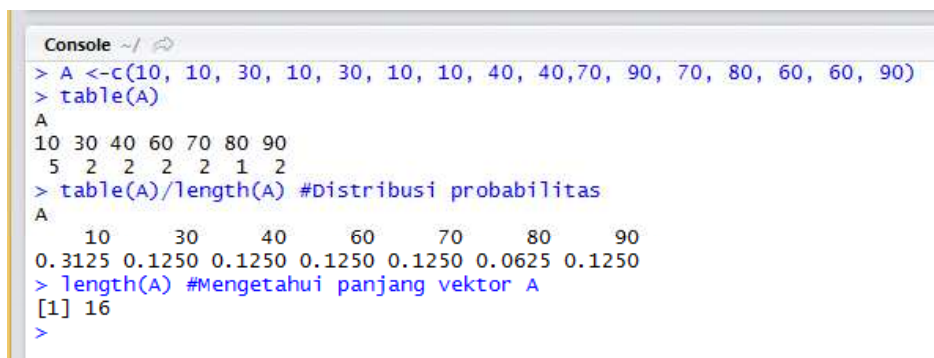


```
Console ~/
> rep(c(1,7,3), c(2,5,2))
[1] 1 1 7 7 7 7 7 3 3
> rep(c(1,7,3), times=c(2,5,2))
[1] 1 1 7 7 7 7 7 3 3
>
> |
```

Gambar 3.16

3.10 Membuat Distribusi Frekuensi dan Distribusi Probabilitas dengan Fungsi *table()*

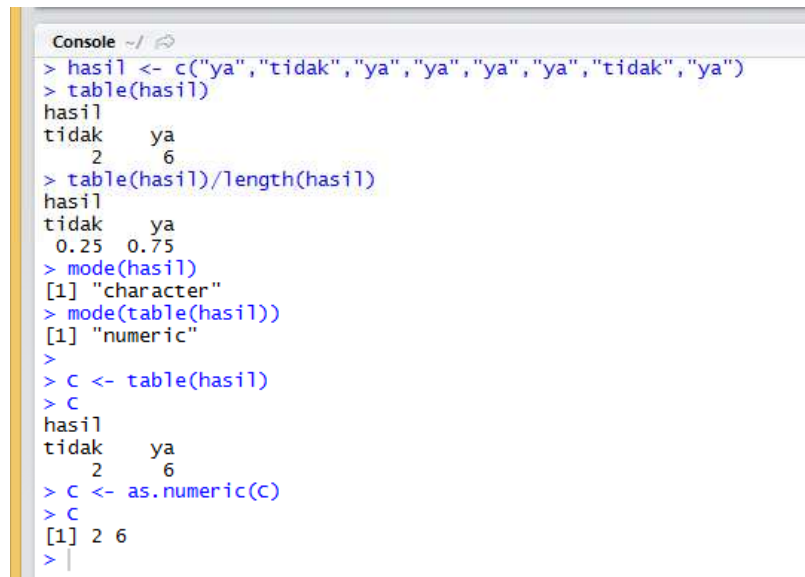
Fungsi *table()* dapat digunakan untuk membuat suatu distribusi frekuensi. Misalkan suatu vektor bernama **A** menyimpan bilangan 10, 10, 30, 10, 30, 10, 10, 40, 40, 70, 90, 70, 80, 60, 60, 90. Diketahui nilai 10 sebanyak 5, nilai 30 sebanyak 2, nilai 40 sebanyak 2, nilai 60 sebanyak 2, nilai 70 sebanyak 2, nilai 80 sebanyak 1, dan nilai 90 sebanyak 2. Gambar 3.17 diperlihatkan penggunaan fungsi *table()* untuk menyajikan distribusi frekuensi dan probabilitas dari data pada vektor **A**. Perhatikan bahwa probabilitas dari nilai 10 adalah 0,3125 yang diperoleh $\frac{5}{16} = 0,3125$, probabilitas dari nilai 80 adalah 0,0625 yang diperoleh $\frac{1}{16} = 0,0625$, dan seterusnya.



```
Console ~/
> A <-c(10, 10, 30, 10, 30, 10, 10, 40, 40, 70, 90, 70, 80, 60, 60, 90)
> table(A)
A
10 30 40 60 70 80 90
 5  2  2  2  2  1  2
> table(A)/length(A) #Distribusi probabilitas
A
 10   30   40   60   70   80   90
0.3125 0.1250 0.1250 0.1250 0.1250 0.0625 0.1250
> length(A) #Mengetahui panjang vektor A
[1] 16
>
```

Gambar 3.17

Gambar 3.18 disajikan contoh distribusi frekuensi dan probabilitas untuk vektor berjenis *character*.



```
Console ~/\n> hasil <- c("ya","tidak","ya","ya","ya","ya","tidak","ya")\n> table(hasil)\nhasil\ntidak    ya\n      2     6\n> table(hasil)/length(hasil)\nhasil\ntidak    ya\n 0.25 0.75\n> mode(hasil)\n[1] "character"\n> mode(table(hasil))\n[1] "numeric"\n>\n> C <- table(hasil)\n> C\nhasil\ntidak    ya\n      2     6\n> C <- as.numeric(C)\n> C\n[1] 2 6\n> |
```

Gambar 3.18

3.11 Fungsi *factor()* dan Fungsi *level()* untuk Mengetahui Keragaman Data

Fungsi *factor()* dalam R dapat digunakan untuk mengetahui keragaman *level* pada data vektor. Misalkan diberikan data sebagai berikut.

ikan, ikan, udang, ikan, udang, ikan, ikan, udang

Berdasarkan data tersebut, terdapat dua *level*, yakni ikan dan udang. Misalkan diberikan data sebagai berikut.

sarjana,diploma, sarjana, pengangguran, sarjana, diploma, diploma, pengangguran

Berdasarkan data tersebut, terdapat tiga *level*, yakni sarjana, diploma, dan pengangguran. Misalkan diberikan data sebagai berikut.

1, 2, 3, 2, 1, 4, 6, 3, 2, 6, 2, 4, 5, 3, 3, 2

Berdasarkan data tersebut, terdapat tiga *level*, yakni 1, 2, 3, 4, 5, dan 6. Ilustrasi dalam R disajikan pada Gambar 3.19 dan Gambar 3.20.

```

Console ~/
> A <- c("ikan","ikan","udang","ikan","udang","ikan","ikan","udang")
> factor(A)
[1] ikan ikan udang ikan udang ikan ikan udang
Levels: ikan udang
> levels(factor(A))
[1] "ikan" "udang"
> B <- levels(factor(A))
> B
[1] "ikan" "udang"
> length(B)
[1] 2
> mode(B)
[1] "character"
>
>
> C <- c("sarjana","diploma","sarjana","pengangguran","sarjana","diploma","diploma","pengangguran")
> factor(C)
[1] ikan udang
Levels: ikan udang
> levels(factor(C))
[1] "diploma" "pengangguran" "sarjana"
> D <- levels(factor(C))
> D
[1] "diploma" "pengangguran" "sarjana"
> length(D)
[1] 3
> mode(D)
[1] "character"
>
>

```

Gambar 3.19

```

Console ~/
> E <- c(1, 2, 3, 2, 1, 4, 6, 3, 2, 6, 2, 4, 5, 3, 3, 2)
> factor(E)
[1] 1 2 3 2 1 4 6 3 2 6 2 4 5 3 3 2
Levels: 1 2 3 4 5 6
> F <- levels(factor(E))
> F
[1] "1" "2" "3" "4" "5" "6"
> mode(F)
[1] "character"
> F <- as.numeric(F)
> F
[1] 1 2 3 4 5 6
> mode(F)
[1] "numeric"
> |

```

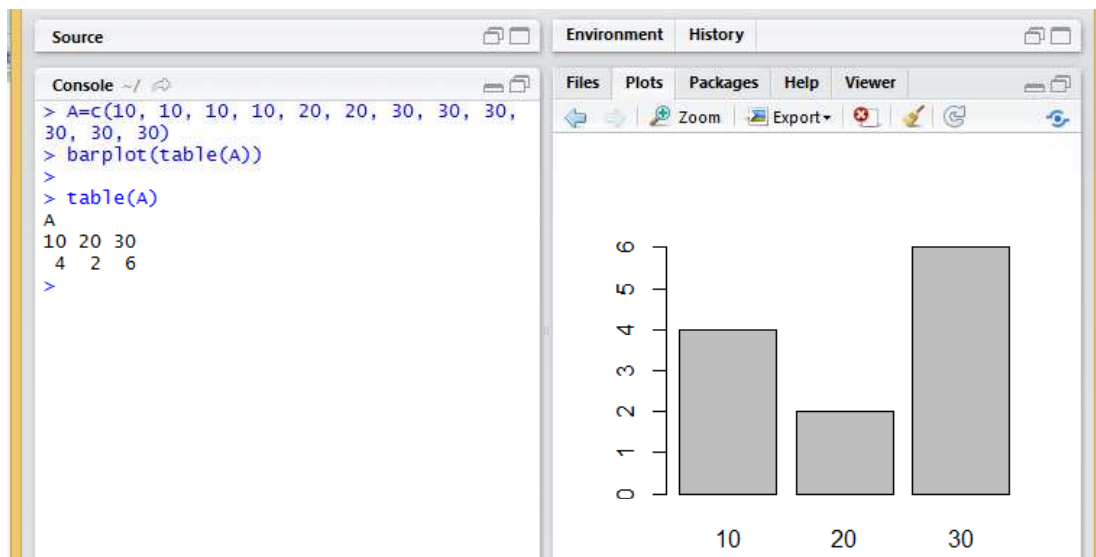
Gambar 3.20

3.12 Mengkonversi Vektor *Character* Menjadi Vektor *Numeric* dengan Fungsi *as.numeric()*

Fungsi *as.numeric()* dapat digunakan untuk mengkonversi vektor berjenis *character* menjadi vektor berjenis *numeric*. Ilustrasi diberikan pada Gambar 3.20.

3.13 Membuat Grafik Batang dengan Fungsi *barplot()*

Fungsi *barplot()* dalam R berfungsi untuk menyajikan data dalam bentuk grafik batang. Misalkan vektor **A** menyimpan data 10, 10, 10, 10, 20, 20, 30, 30, 30, 30, 30, 30. Gambar 3.21 diberikan ilustrasi dalam R untuk menyajikan data pada vektor **A** dalam bentuk grafik batang.



Gambar 3.21

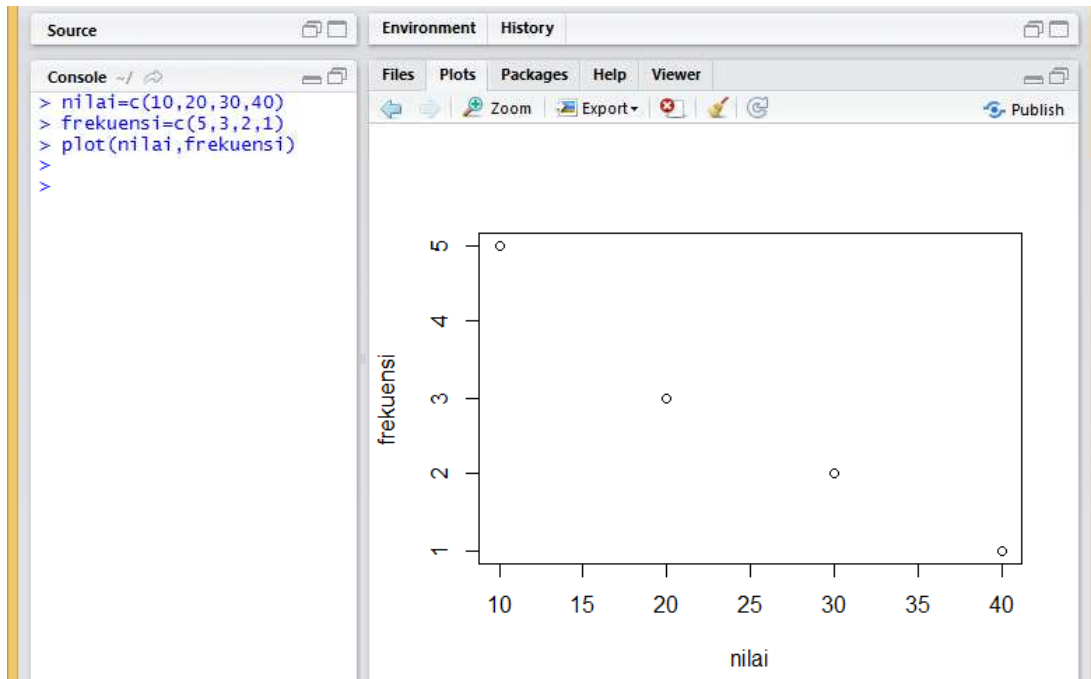


Gambar 3.22

Berdasarkan Gambar 3.21, perhatikan bahwa untuk data dengan nilai 10 mempunyai frekuensi sebanyak 4, data dengan nilai 20 mempunyai frekuensi sebanyak 2, dan data dengan nilai 30 mempunyai frekuensi sebanyak 6. Grafik batang di atas dapat diatur agar disajikan berdasarkan probabilitas (Gambar 3.22). Perhatikan bahwa nilai 0,3333, 0,16667, dan 0,5 masing-masing merupakan probabilitas dari nilai 10, 20, dan 30.

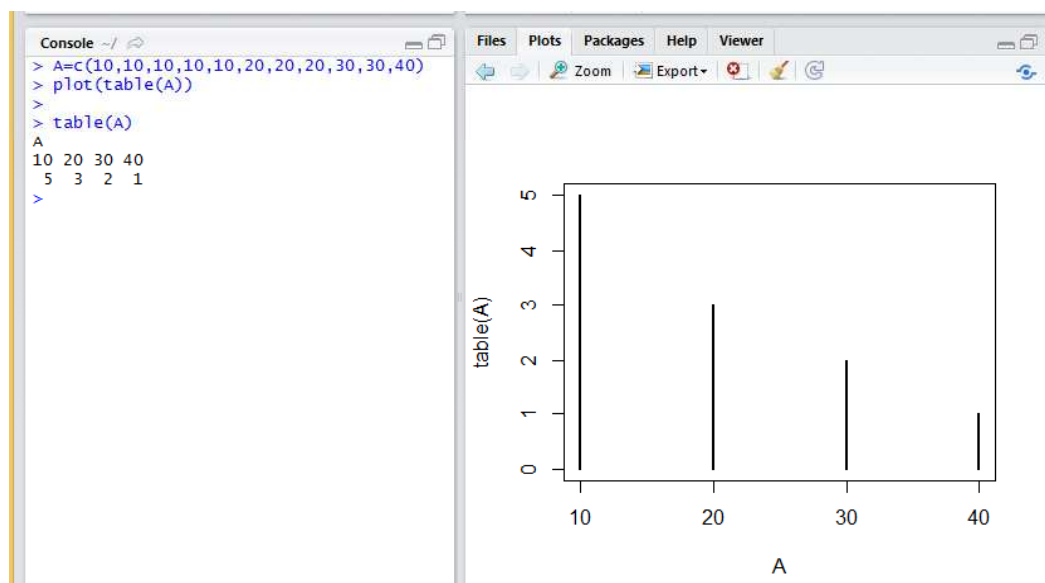
3.14 Memplot Data dengan Fungsi *plot()*

Misalkan vektor bernama **A** menyimpan data 10,10,10,10,10,20,20,20,30,30,40. Diketahui nilai 10 muncul sebanyak 5, nilai 20 sebanyak 3, nilai 30 sebanyak 2, dan nilai 40 sebanyak 1. Gambar 3.23 digunakan fungsi *plot()* untuk memplot data yang tersimpan dalam vektor **A**.



Gambar 3.23

Alternatif lain disajikan pada Gambar 3.24.



Gambar 3.24


3.15 Membulatkan Bilangan dengan Fungsi *round()*, Fungsi *ceiling()*, Fungsi *floor()*, dan Fungsi *signif()*

Gambar 3.25 dan Gambar 3.26 diberikan contoh ilustrasi dalam R penggunaan fungsi *round()*, fungsi *ceiling()*, fungsi *floor()*, dan fungsi *signif()*. Pada penggunaan fungsi *round()*, 2,45 dibulatkan menjadi 2, namun 2,54 dibulatkan menjadi 3. Pada bilangan 2,45, satu angka di belakang koma adalah $4 \leq 5$, maka 2,45 dibulatkan menjadi 2. Pada bilangan 2,54, satu angka di belakang koma adalah $5 \geq 5$, maka 2,54 dibulatkan menjadi 3. Sementara pada penggunaan fungsi *ceiling()*, 2,12 dan 2,94 dibulatkan menjadi bilangan 3, meskipun pada bilangan 2,12, satu angka di belakang koma adalah $1 < 5$, bilangan 2,12 tetap dibulatkan menjadi 3. Namun pada fungsi *floor()*, 2,45 dan 2,656 dibulatkan menjadi 2.



```
Console ~/
> round(2.45)
[1] 2
> round(2.54)
[1] 3
> ceiling(2.12)
[1] 3
> ceiling(2.94545)
[1] 3
> floor(2.45)
[1] 2
> floor(2.656)
[1] 2
> round(2.17645, digits=2)
[1] 2.18
> round(2.172324, digits=2)
[1] 2.17
>
```

Gambar 3.25



```
Console ~/
> signif(2.34567,digits=2)
[1] 2.3
> signif(2.34567,digits=3)
[1] 2.35
> signif(2.34567,digits=1)
[1] 2
> round(2.34567,digits=1)
[1] 2.3
> round(2.34567,digits=2)
[1] 2.35
> round(2.34567,digits=3)
[1] 2.346
>
```

Gambar 3.26

3.16 Mengkonversi Vektor *Numeric* Menjadi Vektor *Character* dengan Fungsi *as.character()*

Fungsi *as.character()* dapat digunakan untuk mengkonversi vektor berjenis *numeric* menjadi vektor berjenis *character*. Ilustrasi diberikan pada Gambar 3.27.

```

Console ~1
> A <- c(1,2,3,4,2,4)
> A
[1] 1 2 3 4 2 4
> mode(A)
[1] "numeric"
> A <- as.character(A)
> A
[1] "1" "2" "3" "4" "2" "4"
> mode(A)
[1] "character"
> |

```

Gambar 3.27

3.17 Mengambil Beberapa Karakter dengan Fungsi *substr()* dan Fungsi *substring()*

Fungsi *substr()* dan fungsi *substring()* dapat digunakan untuk mengambil beberapa karakter dari serangkaian karakter. Ilustrasi diberikan pada Gambar 3.28 dan Gambar 3.29. Berdasarkan Gambar 3.28, diketahui vektor **Buah** menyimpan data “Mangga”, “Anggur”, “Pepaya”, “Semangka”. Selanjutnya perintah R **substr(Buah,1,1)** berarti mengambil satu huruf yang dimulai dari huruf pertama nama buah di vektor **Buah**, yang mana hasilnya “M”, “A”, “P”, dan “S”. Selanjutnya perintah R **substr(Buah,1,3)** berarti mengambil tiga huruf yang dimulai dari huruf pertama nama buah di vektor **Buah** (dari huruf pertama sampai huruf ketiga), yang mana hasilnya “Man”, “Ang”, “Pep”, dan “Sem”. Perintah R **substr(Buah,3,4)** berarti mengambil dua huruf yang dimulai dari huruf ketiga nama buah di vektor **Buah** (dari huruf ketiga sampai huruf keempat).

```

Console ~1
> Buah <- c("Mangga","Anggur","Pepaya","Semangka")
> substr(Buah,1,1) #Mengambil satu huruf yang dimulai dari huruf pertama nama buah
[1] "M" "A" "P" "S"
> substr(Buah,1,3) #Mengambil tiga huruf yang dimulai dari huruf pertama nama buah
[1] "Man" "Ang" "Pep" "Sem"
> substr(Buah,3,4) #Mengambil dua huruf yang dimulai dari huruf ketiga nama buah
[1] "ng" "gg" "pa" "ma"
>
> A <- c(6584, 1834, 5323, 9534)
> substr(A,1,1)
[1] "6" "1" "5" "9"
> substr(A,1,2)
[1] "65" "18" "53" "95"
>

```

Gambar 3.28

Pada Gambar 3.29 digunakan fungsi *substring()*. Perintah R **substring(Buah,3)** berarti mengambil seluruh huruf yang dimulai dari huruf ketiga nama buah di vektor **Buah**, yang mana hasilnya adalah “ngga”, “ggur”, “paya”, dan “mangka”.

```

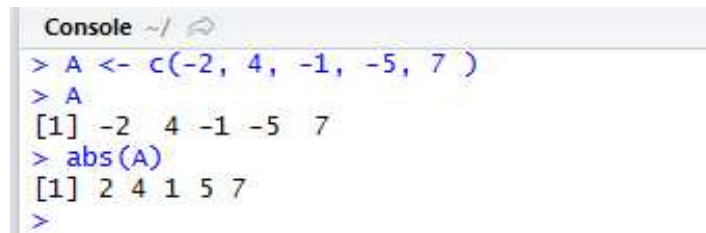
Console ~1
> Buah <- c("Mangga","Anggur","Pepaya","Semangka")
> substring(Buah,3) #Mengambil seluruh huruf yang dimulai dari huruf ketiga nama buah
[1] "ngga" "ggur" "paya" "mangka"
>

```

Gambar 3.29

3.18 Mengkonversi Bilangan Negatif Menjadi Bilangan Tak Negatif dalam Vektor dengan Fungsi *abs()*

Misalkan bilangan -2, 4, -1, -5, 7 ditugaskan ke vektor bernama **A**. Andaikan diinginkan seluruh elemen di vektor **A** bernilai tak negatif, yakni menjadi 2, 4, 1, 5, dan 7. Fungsi *abs()* dapat digunakan untuk melakukan hal tersebut. Perhatikan Gambar 3.30.



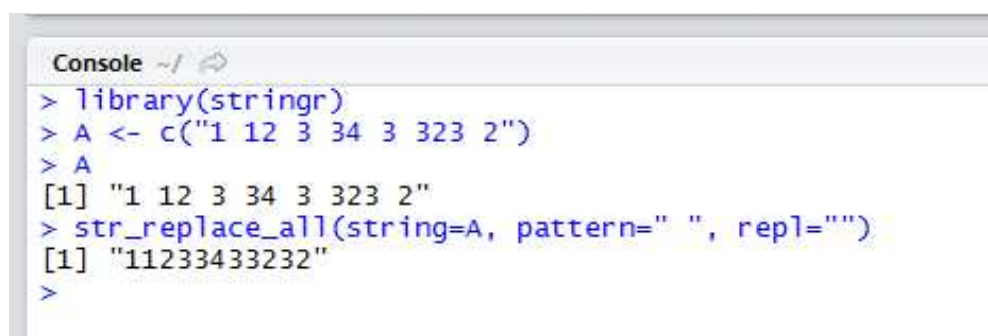
```
Console ~/
> A <- c(-2, 4, -1, -5, 7)
> A
[1] -2  4 -1 -5  7
> abs(A)
[1] 2 4 1 5 7
>
```

Gambar 3.30

3.19 Fungsi *str_replace_all()* dan Fungsi *strsplit()*

Gambar 3.31 diberikan ilustrasi penggunaan fungsi *str_replace_all()*. Penggunaan fungsi *str_replace_all()* pada Gambar 3.31 dimaksudkan untuk menghapus spasi, sehingga “1 12 3 34 3 323 2” menjadi “11233433232”. Pada Gambar 3.32 diberikan ilustrasi penggunaan fungsi *strsplit()*. Diketahui elemen dari vektor **A** adalah “1 23 3234 43 4 534”.

Jenis data dari vektor **A** adalah *character*. Selanjutnya digunakan fungsi *strsplit()* untuk memisahkan angka (masih dalam jenis *character*) di antara spasi, sehingga menjadi “1” “23” “3234” “43” “4” “534”. Hasil dari perintah R `mode(strsplit(A, " "))` adalah *list*, yang kemudian dikonversi menjadi *character* dengan perintah R `unlist(strsplit(A, " "))`.



```
Console ~/
> library(stringr)
> A <- c("1 12 3 34 3 323 2")
> A
[1] "1 12 3 34 3 323 2"
> str_replace_all(string=A, pattern=" ", repl="")
[1] "11233433232"
>
```

Gambar 3.31

```

Source

Console ~/
> A <- c("1 23 3234 43 4 534")
> A
[1] "1 23 3234 43 4 534"
> mode(A)
[1] "character"
> strsplit(A, " ")
[[1]]
[1] "1" "23" "3234" "43" "4" "534"

> mode(strsplit(A, " "))
[1] "list"
> unlist(strsplit(A, " "))
[1] "1" "23" "3234" "43" "4" "534"
> unlist(strsplit(A, "[ ]"))
[1] "1" "23" "3234" "43" "4" "534"
> unlist(strsplit(A, " ", fixed=TRUE))
[1] "1" "23" "3234" "43" "4" "534"
> mode(unlist(strsplit(A, " ")))
[1] "character"
> B <- unlist(strsplit(A, " "))
> mode(B)
[1] "character"
> B <- as.numeric(B)
> B
[1] 1 23 3234 43 4 534
> mode(B)
[1] "numeric"
> |

```

Gambar 3.32

Gambar 3.33, Gambar 3.34, dan Gambar 3.35 diberikan ilustrasi tambahan mengenai penggunaan fungsi *strsplit()* .

```

Console ~/
> A <- c("1 23 3234 43 4 534 5 43434 32323")
> A
[1] "1\t23\t3234\t43\t4\t534\t5\t43434\t32323"
> mode(A)
[1] "character"
> strsplit(A, "\t")
[[1]]
[1] "1" "23" "3234" "43" "4" "534" "5" "43434" "32323"

> B <- unlist(strsplit(A, "\t"))
> mode(B)
[1] "character"
> B <- as.numeric(B)
> B
[1] 1 23 3234 43 4 534 5 43434 32323
> mode(B)
[1] "numeric"
>

```

Gambar 3.33

```

Console ~/
> A <- c("1 23 3234" 43 4 34 5 43434 32323")
> A <- strsplit(A, "\t")
> A <- unlist(A)
> A <- strsplit(A, " ")
> A
[[1]]
[1] "1" "23" "3234"

[[2]]
[1] "43"

[[3]]
[1] "4" "34"

[[4]]
[1] "5"

[[5]]
[1] "43434"

[[6]]
[1] "32323"

> mode(A)
[1] "list"
> B <- unlist(A)
> mode(B)
[1] "character"
> B <- as.numeric(B)
> B
[1] 1 23 3234 43 4 34 5 43434 32323
> mode(B)
[1] "numeric"
>

```

Gambar 3.34

```

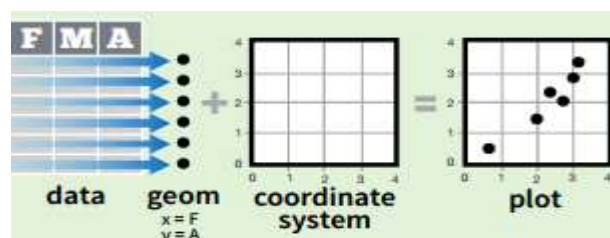
Console ~/
> A <- c("ddfsfsdfdf5")
> strsplit(A, NULL)[[1]]
[1] "d" "d" "f" "d" "s" "f" "s" "d" "f" "d" "f" "5"
>
>

```

Gambar 3.35

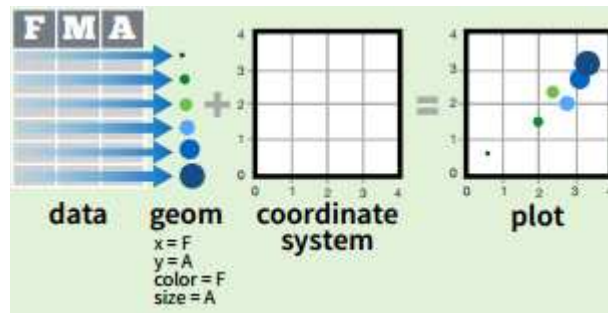
3.20 ggplot2

Huruf "gg" pada *ggplot2* merupakan kependekan dari *grammar of graphics*. Ide dalam menggunakan package ini adalah kita bisa membuat *graph* dari komponent: **dataset**, **geom** yang merupakan tanda visual yang mewakili data dan **coordinate system**. Untuk menampilkan



Gambar 3.36

nilai datanya, gunakan variabel sebagai *aesthetic properties* dari **geom** seperti *size*, *color* dan lokasi **x** dan **y**.

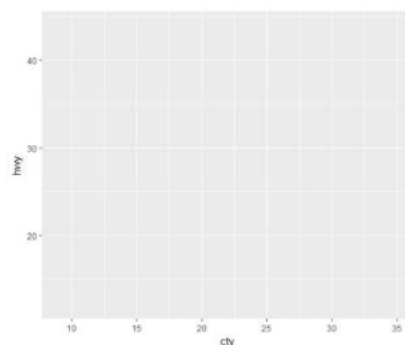


Gambar 3.37

Untuk memudahkan pemahaman, ketikkan perintah berikut ,

```
ggplot(data = mpg, aes(x = cty, y = hwy))
```

maka akan menghasilkan plot yang tidak ada isinya atau bisa kita sebut sebagai layer pertama yang nantinya dapat kita tambahkan layer-layer berikutnya.

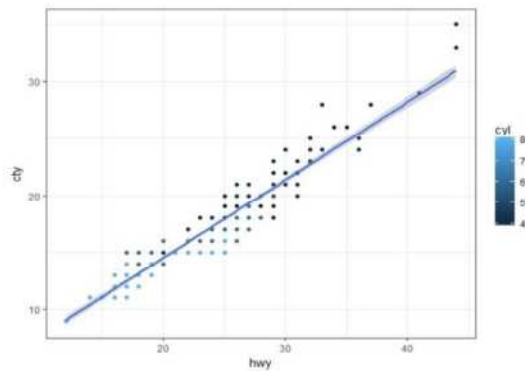


Gambar 3.38

Untuk menambahkan layer baru ke plot, kita dapat menggunakan **geom_*()** atau **stat_*()**. Seperti di bawah ini,

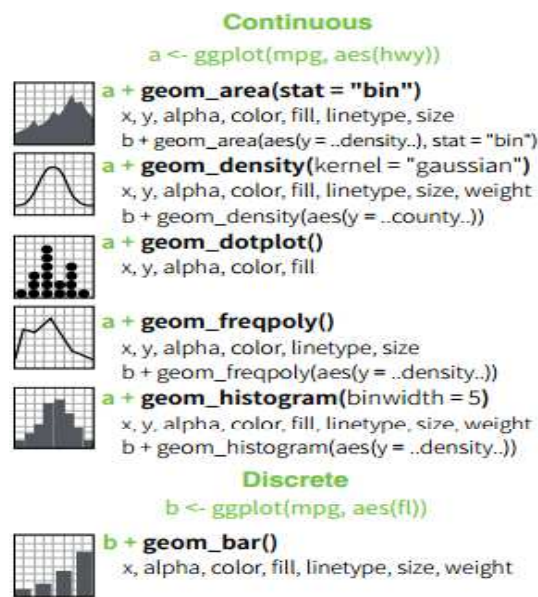
```
ggplot(mpg, aes(hwy, cty)) +  
  geom_point(aes(color=cyl)) +  
  geom_smooth(method = "lm") +  
    coord_cartesian() +  
    scale_color_gradientn() +  
    theme_bw()
```

yang akan menghasilkan,



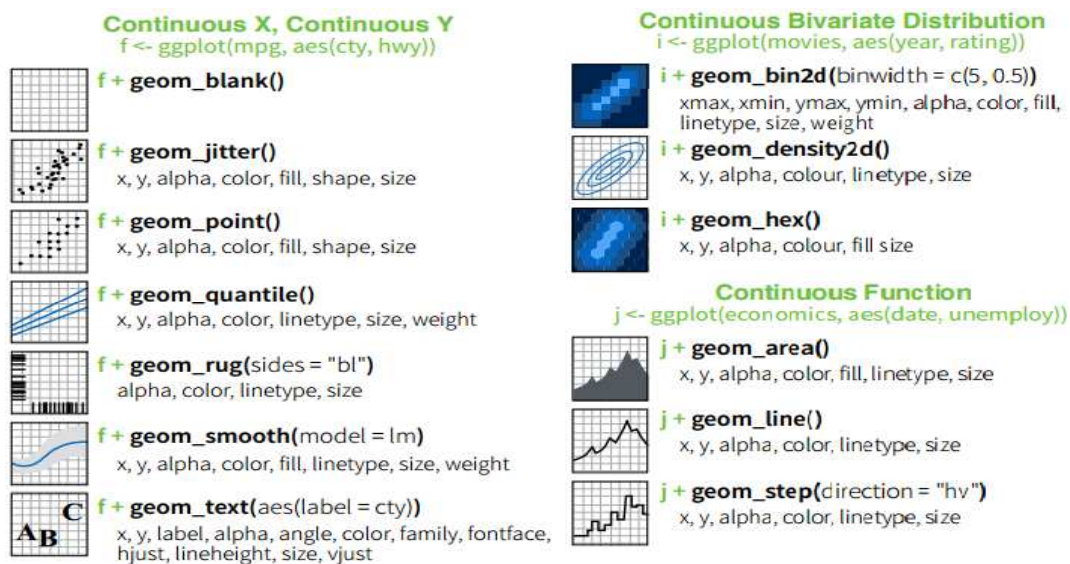
Gambar 3.39

3.21 Lanjutan ggplot2 (Satu Variabel)



Gambar 3.40

3.22 Lanjutan ggplot2 (Dua Variabel)



Discrete X, Continuous Y

```
g <- ggplot(mpg, aes(class, hwy))
```



g + geom_bar(stat = "identity")
x, y, alpha, color, fill, linetype, size, weight



g + geom_boxplot()
lower, middle, upper, x, ymax, ymin, alpha, color, fill, linetype, shape, size, weight



g + geom_dotplot(binaxis = "y", stackdir = "center")
x, y, alpha, color, fill



g + geom_violin(scale = "area")
x, y, alpha, color, fill, linetype, size, weight

Discrete X, Discrete Y

```
h <- ggplot(diamonds, aes(cut, color))
```

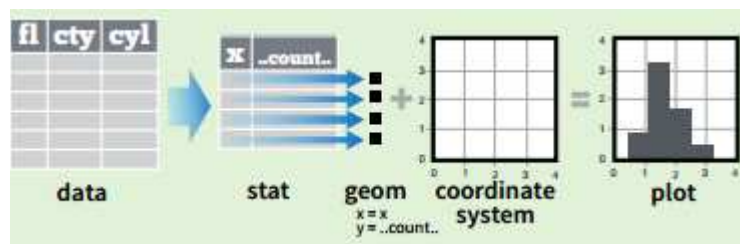


h + geom_jitter()
x, y, alpha, color, fill, shape, size

Gambar 3.41

3.23 Stats

Selain menggunakan **geom**, untuk membuat layer dapat digunakan **stat**. Tak jarang dijumpai adanya keinginan untuk membuat plot yang merupakan transformasi dari asli.



Gambar 3.42

BAB 4

OPERATOR PEMBANDING DAN OPERATOR LOGIKA

4.1 Operator Pembanding atau Operator Relasional

Operator pembanding, atau disebut juga dengan istilah operator relasional, biasa digunakan untuk membandingkan dua bilangan. Tabel 4.1 disajikan berbagai macam operator pembanding.

Tabel 4.1 Operator Pembanding

Operator Pembanding	Keterangan
==	Sama dengan (bukan penugasan)
!=	Tidak sama dengan
>	Lebih dari
<	Kurang dari
>=	Lebih dari atau sama dengan
<=	Kurang dari atau sama dengan

Gambar 4.1 diberikan ilustrasi penggunaan operator pembanding.

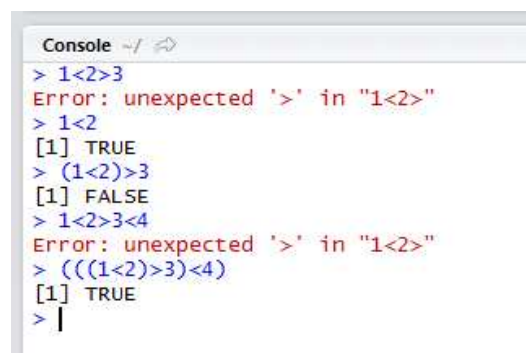
```
Console -/ ↗
> A <- c(1,2,3,4)
> B <- c(2,2,2,2)
> A>B
[1] FALSE FALSE TRUE TRUE
> A<B
[1] TRUE FALSE FALSE FALSE
> B>A
[1] TRUE FALSE FALSE FALSE
> B<A
[1] FALSE FALSE TRUE TRUE
> A==B
[1] FALSE TRUE FALSE FALSE
> A!=B
[1] TRUE FALSE TRUE TRUE
> A>=B
[1] FALSE TRUE TRUE TRUE
> 2>3
[1] FALSE
> 2<3
[1] TRUE
> 2!=3
[1] TRUE
> 2==3
[1] FALSE
> 2==2
[1] TRUE
>
```

Gambar 4.1

Berdasarkan Gambar 4.1, vektor **A** ditugaskan untuk menyimpan bilangan 1, 2, 3, dan 4, sementara vektor **B** ditugaskan untuk menyimpan bilangan 2, 2, 2, dan 2. Perintah R **A>B** berarti:

- ⇒ $1 > 2$ (salah atau FALSE). 1 merupakan elemen pertama dari vektor **A**, sementara 2 merupakan elemen pertama dari vektor **B**.
- ⇒ $2 > 2$ (salah atau FALSE). 2 merupakan elemen kedua dari vektor **A**, sementara 2 merupakan elemen kedua dari vektor **B**.
- ⇒ $3 > 2$ (benar atau TRUE). 3 merupakan elemen ketiga dari vektor **A**, sementara 2 merupakan elemen ketiga dari vektor **B**.
- ⇒ $4 > 2$ (benar atau TRUE). 4 merupakan elemen keempat dari vektor **A**, sementara 2 merupakan elemen keempat dari vektor **B**.

Perhatikan bahwa hasil dari perintah R $\mathbf{A > B}$ adalah FALSE, FALSE, TRUE, dan TRUE (perhatikan Gambar 4.1). Perhatikan juga bahwa perintah R $\mathbf{2 \neq 3}$ (2 tidak sama dengan 3) memberikan hasil TRUE. Perintah R $\mathbf{2 == 3}$ (2 sama dengan 3) memberikan hasil FALSE. Hasil dari penggunaan operator pembandingan memberikan hasil nilai logika, yakni TRUE atau FALSE. Gambar 4.2 diberikan contoh penggunaan operator pembandingan lebih dari 1 kali dalam satu perintah R.



```

Console ~/
> 1<2>3
Error: unexpected '>' in "1<2>"
> 1<2
[1] TRUE
> (1<2)>3
[1] FALSE
> 1<2>3<4
Error: unexpected '>' in "1<2>"
> (((1<2)>3)<4)
[1] TRUE
> |

```

Gambar 4.2

4.2 Operator Logika

Operator logika pada umumnya digunakan untuk membuat keadaan logika dengan menggunakan dua buah kondisi atau sebuah kondisi, yang mana bergantung pada operator logika yang digunakan. Suatu kondisi dapat bernilai benar, yakni **TRUE**, atau dapat bernilai salah, yakni **FALSE**.

Operator logika “dan” yang dilambangkan “&&” menggunakan dua buah kondisi, yakni

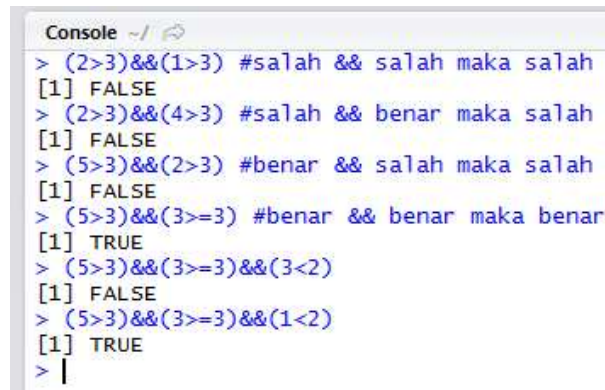
(kondisi1)&&(kondisi2)

Suatu kondisi dapat bernilai benar atau salah. Tabel 4.2 disajikan nilai kebenaran untuk operasi logika “dan”.

Tabel 4.2 Nilai Kebenaran untuk Operasi “dan”

Kondisi Pertama	Kondisi Kedua	Hasil &&
Salah	Salah	Salah
Salah	Benar	Salah
Benar	Salah	Salah
Benar	Benar	Benar

Pada operasi “dan” akan menghasilkan nilai benar, jika kedua kondisi bernilai benar. Selain dari pada itu, bernilai salah. Gambar 4.3 diberikan ilustrasi dalam R terkait operasi “dan”.



```
Console ~/
> (2>3)&&(1>3) #salah && salah maka salah
[1] FALSE
> (2>3)&&(4>3) #salah && benar maka salah
[1] FALSE
> (5>3)&&(2>3) #benar && salah maka salah
[1] FALSE
> (5>3)&&(3==3) #benar && benar maka benar
[1] TRUE
> (5>3)&&(3==3)&&(3<2)
[1] FALSE
> (5>3)&&(3==3)&&(1<2)
[1] TRUE
> |
```

Gambar 4.3

Operator logika “atau” yang dilambangkan dengan “||” juga menggunakan dua buah kondisi, yakni

(kondisi1)||(kondisi2)

Suatu kondisi dapat bernilai benar atau salah. Tabel 4.3 disajikan nilai kebenaran untuk operasi logika “atau”.

Tabel 4.3 Nilai Kebenaran untuk Operasi “atau”

Kondisi Pertama	Kondisi Kedua	Hasil
Salah	Salah	Salah
Salah	Benar	Benar
Benar	Salah	Benar
Benar	Benar	Benar

Pada operasi “||” akan menghasilkan nilai salah, jika kedua kondisi bernilai salah. Selain dari pada itu, bernilai benar. Gambar 4.4 diberikan ilustrasi dalam R terkait operasi “||”.

```

Console ~/
> (2>3)||(1>3) #salah || salah maka salah
[1] FALSE
> (2>3)||(4>3) #salah || benar maka benar
[1] TRUE
> (5>3)||(2>3) #benar || salah maka salah
[1] TRUE
> (5>3)||(3>=3) #benar || benar maka benar
[1] TRUE
> (5>3)||(3>=3)||(3<2)
[1] TRUE
> (5>3)||(3>=3)||(1<2)
[1] TRUE
>

```

Gambar 4.4

Operator logika “bukan” yang dilambangkan dengan “!” berguna untuk membalikkan suatu nilai kebenaran dari suatu kondisi. Perintah R `!(2>3)` akan menghasilkan nilai kebenaran TRUE. Perhatikan Gambar 4.5.

```

Console ~/
> 2>5
[1] FALSE
> !(2>5)
[1] TRUE
> 3==3
[1] TRUE
> !(3==3)
[1] FALSE
>

```

Gambar 4.5

4.3 Lebih Lanjut Penggunaan Operator Pembandingan dan Operator Logika

Perhatikan Gambar 4.6. Perintah R `A[A>5]` berarti menampilkan elemen dari vektor **A** dengan syarat lebih besar dari 5, yakni 8 (pada indeks ke-1), 7 (pada indeks ke-4), dan 6 (pada indeks ke-8).

```

Console ~/
> A <- c(8,4,5,7,2,3,1,6)
> A
[1] 8 4 5 7 2 3 1 6
> A[A>5]
[1] 8 7 6
> |

```

Gambar 4.6

Perintah R `A[B[B>3]]` pada Gambar 4.7 sama saja dengan `A[4]`, yang akan menghasilkan 7. Perintah R `B[B>3]` akan menghasilkan 4 (Perhatikan Gambar 4.8).

```

Console ~/
> A <- c(8,4,5,7,2,3,1,6)
> B <- c(4,3,1,2)
> A[B[B>3]]
[1] 7
> |

```

Gambar 4.7

```

Console ~/
> A <- c(8,4,5,7,2,3,1,6)
> B <- c(4,3,1,2)
> A[B[B>3]]
[1] 7
> B[B>3]
[1] 4
> |

```

Gambar 4.8

Perhatikan Gambar 4.9. Perintah R pada Gambar 4.9 `A[A>=2 & A<7]` berarti menampilkan bilangan dari vektor **A** dengan syarat bilangan tersebut lebih besar atau sama dengan 2 dan lebih kecil dari 7, yang mana bilangan-bilangan tersebut adalah 4, 5, 2, 3, 6, 5, 2, 2, dan 6.

```

Console ~/
> A <- c(8,4,5,7,2,3,1,6,5,2,8,2,9,6)
> A[A>=2 & A<7]
[1] 4 5 2 3 6 5 2 2 6
> table(A[A>=2 & A<7])

 2  3  4  5  6 
 3  1  1  2  2 
> |

```

Gambar 4.9

Namun jika perintah R nya adalah `A[A>=2 | A<7]` akan menampilkan seluruh elemen dari vektor **A** (Gambar 4.10).

```

Console ~/
> A <- c(8,4,5,7,2,3,1,6,5,2,8,2,9,6)
> A[A>=2 & A<7]
[1] 4 5 2 3 6 5 2 2 6
> table(A[A>=2 & A<7])

 2  3  4  5  6 
 3  1  1  2  2 
> A[A>=2 | A<7]
[1] 8 4 5 7 2 3 1 6 5 2 8 2 9 6
> |

```

Gambar 4.10

Perintah R `A[A<2 | A>9]` menampilkan hasil 1. Elemen dari vektor **A** yang lebih kecil dari 2 adalah 1, sementara elemen dari vektor **A** yang lebih besar dari 9 tidak ada. Perintah R `A[A<2 | A>=9]` menampilkan hasil 1 dan 9, sementara perintah R `A[A<=2 | A>=9]` menampilkan hasil 2 (pada indeks ke-5), 1 (pada indeks ke-7), 2 (pada indeks ke-10), 2 (pada indeks ke-12), dan 9 (pada indeks ke-13).

```

Console
> A <- c(8,4,5,7,2,3,1,6,5,2,8,2,9,6)
> A[A>=2 & A<7]
[1] 4 5 2 3 6 5 2 2 6
> table(A[A>=2 & A<7])

 2 3 4 5 6 
3 1 1 2 2 
> A[A>=2 | A<7]
[1] 8 4 5 7 2 3 1 6 5 2 8 2 9 6
> A[A<2 | A>9]
[1] 1
> A[A<2 | A>=9]
[1] 1 9
> A[A<=2 | A>=9]
[1] 2 1 2 2 9
>

```

Gambar 4.11