Statistika Lingkungan Menggunakan R

Moh. Rosidi

2019-03-11

Contents

Pengantar **5**

1 Mengenal Bahasa R **7**

1.1 Sejarah R 7
1.2 Fitur dan Karakteristik R 8
1.3 Kelebihan dan Kekurangan R 8
1.4 RStudio 9
1.5 Menginstall R dan RStudio 9
1.6 Working Directory 9
1.7 Fasilitas Help 12
1.8 Referensi 16

2 Sintaks Bahasa R **17**

2.1 Operator Aritmatika 17
2.2 Fungsi Aritmetik 18
2.3 Operator Relasi 19
2.4 Operator Logika 20
2.5 Memasukkan Nilai Kedalam Variabel 21
2.6 Tipe Data 23
2.7 Vektor 26
2.8 Matriks 30
2.9 Faktor 36
2.10 Data Frames 38
2.11 List 41
2.12 Loop 44
2.13 Decision Making 47
2.14 Fungsi 49
2.15 Referensi 51

3 Methods **53**

4 Applications **55**

4.1 Example one 55

4.2 Example two 55

5 Final Words **57**

Pengantar

Buku ini menyajikan penerapan program `R` dalam `Statistika Lingkungan`. Buku ini akan disajikan secara ringkas menggunakan sejumlah contoh kasus yang relevan dalam bidang lingkungan.

Penulis berharap buku ini dapat menjadi referensi sumber terbuka bagi mahasiswa yang ingin menggunakan `R` untuk kegiatan analisa data. Sehingga dapat mengurangi ketergantungan pada penggunaan aplikasi yang berlisensi.

Chapter 1

Mengenal Bahasa R

1.1 Sejarah R

`R` Merupakan bahasa yang digunakan dalam komputasi **statistik** yang pertama kali dikembangkan oleh **Ross Ihaka** dan **Robert Gentlement** di University of Auckland New Zealand yang merupakan akronim dari nama depan kedua pembuatnya. Sebelum `R` dikenal ada `S` yang dikembangkan oleh **John Chambers** dan rekan-rekan dari **Bell Laboratories** yang memiliki fungsi yang sama untuk komputasi statistik. Hal yang membedakan antara keduanya adalah `R` merupakan sistem komputasi yang bersifat gratis.Logo `R` dapat dilihat pada Figure 1.1.

`R` dapat dibilang merupakan aplikasi sistem **statistik** yang kaya. Hal ini disebabkan banyak sekali paket yang dikembangkan oleh pengembang dan komunitas untuk keperluan analisa statistik seperti *linear regression*, *clustering*, *statistical test*, dll. Selain itu, `R` juga dapat ditambahkan paket-paket lain yang dapat meningkatkan fiturnya.

Sebagai sebuah bahasa pemrograman yang banyak digunakan untuk keperluan analisa data, `R` dapat dioperasikan pada berbagai sistem operasi pada komputer. Adapun sistem operasi yang didukung antara lain: `UNIX`, `Linux`, `Windows`, dan `MacOS`.

Figure 1.1: Logo R.

1.2 Fitur dan Karakteristik R

R memiliki karakteristik yang berbeda dengan bahasa pemrograman lain seperti C++,python, dll. R memiliki aturan/sintaks yang berbeda dengan bahasa pemrograman yang lain yang membuatnya memiliki ciri khas tersendiri dibanding bahasa pemrograman yang lain.

Beberapa ciri dan fitur pada R antara lain:

1. **Bahasa R bersifat case sensitif**. maksudnya adalah dalam proses input R huruf besar dan kecil sangat diperhatikan. Sebagai contoh kita ingin melihat apakah objek A dan B pada sintaks berikut:

```
A <- "Andi"
B <- "andi"

# cek kedua objek A dan B
A == B
```

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

```
# Kesimpulan : Kedua objek berbeda
```

2. **Segala sesuatu yang ada pada program R akan diangap sebagai objek**. konsep objek ini sama dengan bahasa pemrograma berbasis objek yang lain seperti Java, C++, python, dll.Perbedaannya adalah bahasa R relatif lebih sederhana dibandingkan bahasa pemrograman berbasis obejk yang lain.
3. **interpreted language atau script**. Bahasa R memungkinkan pengguna untuk melakukan kerja pada R tanpa perlu kompilasi kode program menjadi bahasa mesin.
4. Mendukung proses **loop**, **decision making**, dan menyediakan berbagai jenis **operstor** (aritmatika, logika, dll).
5. **Mendukung export dan import berbagai format file**, seperti:TXT, CSV, XLS, dll.
6. **Mudah ditingkatkan melalui penambahan fungsi atau paket**. Penambahan paket dapat dilakukan secara online melalui CRAN atau melalui sumber seperti github.
7. **Menyedikan berbagai fungsi untuk keperluan visualisasi data**. Visualisasi data pada R dapat menggunakan paket bawaan atau paket lain seperti ggplo2,ggvis, dll.

1.3 Kelebihan dan Kekurangan R

Selain karena R dapat digunakan secara gratis terdapat **kelebihan** lain yang ditawarkan, antara lain:

1. **Protability**. Penggunaan software dapat digunakan kapanpun tanpa terikat oleh masa berakhirnya lisensi.
2. **Multiplatform**. R bersifat *Multiplatform Operating Systems*, dimana *software* R lebih kompatibel dibanding *software* statistika lainnya. Hal in berdampak pada kemudahan dalam penyesuaian jika pengguna harus berpindah sistem operasi karena R baik pada sistem operasi seperti windows akan sama pengoperasiannya dengan yang ada di Linux (paket yang digunakan sama).
3. **General** dan **Cutting-edge**. Berbagai metode statistik baik metode klasik maupun baru telah diprogram kedalam R. Dengan demikian *software* ini dapat digunakan untuk analisis statistika dengan pendekatan klasik dan pendekatan modern.
4. **Programable**. Pengguna dapat memprogram metode baru atau mengembangakan modifikasi dari analisis statistika yang telah ada pada sistem R.
5. **Berbasis analisis matriks**. Bahasa R sangat baik digunakan untuk *programming* dengan basis matriks.

6. Fasiltas grafik yang lengkap.

Adapun kekurangan dari R antara lain:

1. **Point and Click GUI**. Interaksi utama dengan R bersifat *CLI* (*Command Line Interface*), walaupun saat ini telah dikembangkan paket yang memungkinkan kita berinteraksi dengan R menggunakan *GUI* (*Graphical User Interface*) sederhana menggunakan paket `R-Commander` yang memiliki fungsi yang terbatas. `R- Commander` sendiri merupakan *GUI* yang diciptakan dengan tujuan untuk keperluan pengajaran sehingga analisis statistik yang disediakan adalah yang klasik. Meskipun terbatas paket ini berguna jika kita membutuhkan analisis statistik sederhana dengan cara yang simpel.
2. **Missing statistical function**. Meskipun analisis statistika dalam R sudah cukup lengkap, namun tidak semua metode statistika telah diimplementasikan ke dalam R. Namun karena R merupakan *lingua franca* untuk keperluan komputasi statistika modern staan ini, dapat dikatakan ketersediaan fungsi tambahan dalam bentuk paket hanya masalah waktu saja.

1.4 RStudio

Aplikasi R pada dasarnya berbasis teks atau *command line* sehingga pengguna harus mengetikkan perintah-perintah tertentu dan harus hapal perintah-perintahnya. Setidaknya jika kita ingin melakukan kegiatan analisa data menggunakan R kita harus selalu siap dengan perintah-perintah yang hendak digunakan sehingga buku manual menjadi sesuatu yang wajib adasaat berkeja dengan R.

Kondisi ini sering kali membingunkan bagi pengguna pemula maupun pengguna mahir yang sudah terbiasa dengan aplikasi statistik lain seperti SAS, SPSS, Minitab, dll. Alasan itulah yang menyebabkan pengembang R membuat berbagai *frontend* untuk R yang berguna untuk memudahkan dalam pengoperasian R.

`RStudio` merupakan salah satu bentuk *frontend* R yang cukup populer dan nyaman digunakan. Selain nyaman digunakan, `RStudio` memungkinkan kita melakukan penulisan laporan menggunakan `Rmarkdown` atau `RNotebook` serta membuat berbagai bentuk project seperti shyni, dll. Pada R studio juga memungkinkan kita mengatur *working directory* tanpa perlu mengetikkan sintaks pada Commander, yang diperlukan hanya memilihnya di menu `RStudio`. Selain itu, kita juga dapat meng-import file berisikan data tanpa perlu mengetikkan pada Commander dengan cara memilih pada menu `Environment`.

1.5 Menginstall R dan RStudio

Pada tutorial ini hanya akan dijelaskan bagaimana menginstal R dan `RStudio` pada sistem operasi `windows`. Sebelum memulai menginstal sebaiknya pembaca mengunduh terlebih dahulu *installer* R dan RStudio.

1. Jalankan proses pemasangan dengan meng-klik *installer* aplikasi R dan `RStudio`.
2. Ikuti langkah proses pemasangan aplikasi yang ditampilkan dengan klik `OK` atau `Next`.
3. Apabila pemasangan telah dilakukan, jalankan aplikasi yang telah terpasang untuk menguji jika aplikasi telah berjalan dengan baik.

Jendela aplikasi yang telah terpasang ditampilkan pada Figure 1.2 dan Figure 1.3.

> **Note:** Sebaiknya install R terlebih dahulu sebelum `RStudio`

1.6 Working Directory

Setiap pengguna akan bekerja pada tempat khusus yang disebut sebagai *working directory*. *working directory* merupakan sebuah folder dimana R akan membaca dan menyimpan file kerja kita. Pada pengguna `windows`, *working directory* secara default pada saat pertama kali menginstall R terletak pada folder `c:\\Document`.

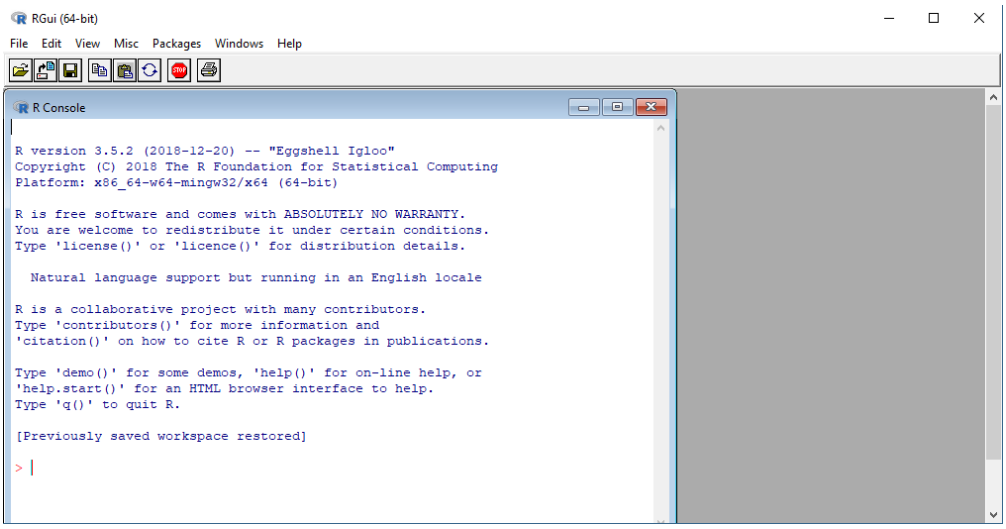

Figure 1.2: Jendela R.

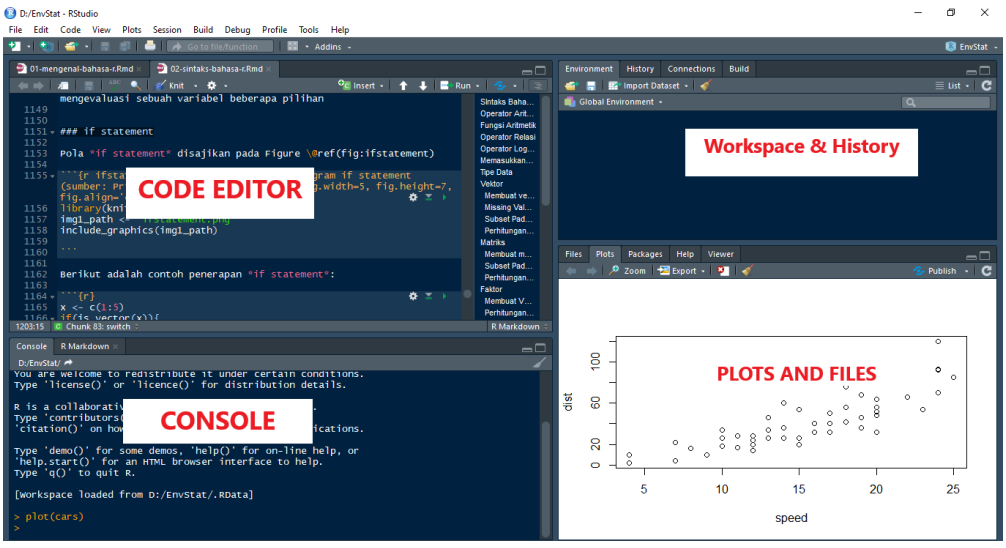

Figure 1.3: Jendela RStudio.

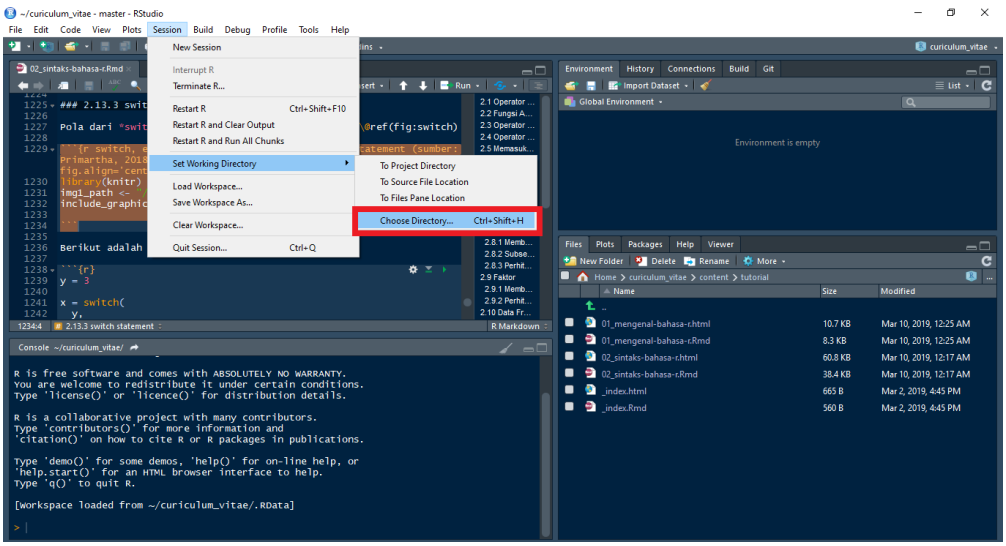

Figure 1.4: Mengubah working directory.

1.6.1 Mengubah Lokasi Working Directory

Kita dapat mengubah lokasi *working directory* berdasarkan lokasi yang kita inginkan, misalnya letak data yang akan kita olah tidak ada pada folder default atau kita ingin pekerjaan kita terkait `R` dapat berlangsung pada satu folder khusus.

Berikut adalah cara mengubah *working directory* pada `R`.

1. Buatlah folder pada drive (kita bisa membuat folder pada selain drive c) dan namai dengan nama yang kalian inginkan. Pada tutorial ini penulis menggunakan nama folder `R`.
2. Jika pengguna menggunakan `RStudio`, pada menu `RStudio` pilih **Session > Set Working Directory > Chooses Directory**. Proses tersebut ditampilkan pada Figure 1.4
3. Pilih folder yang telah dibuat pada step 1 sebagai *working directory.

> **Note:** Data atau file yang hendak dibaca selama proses kerja pada `R` harus selalu diletakkan pada working directory. Jika tidak maka data atau file tidak akan terbaca.

Untuk mengecek apakah proses perubahan telah terjadi, kita dapat mengeceknya dengan menjalankan perintah berikut untuk melihat lokasi *working directory* kita yang baru.

```
getwd()
```

Selain itu kita dapat mengubah *working directory* menggunakan perintah berikut:

```
# Ubah working directori pada folder R
setwd("/Documents/R")
```

> **Note:** Pada proses pengisian lokasi folder pastikan pemisah pada lokasi folder menggunakan tanda “/” bukan “”

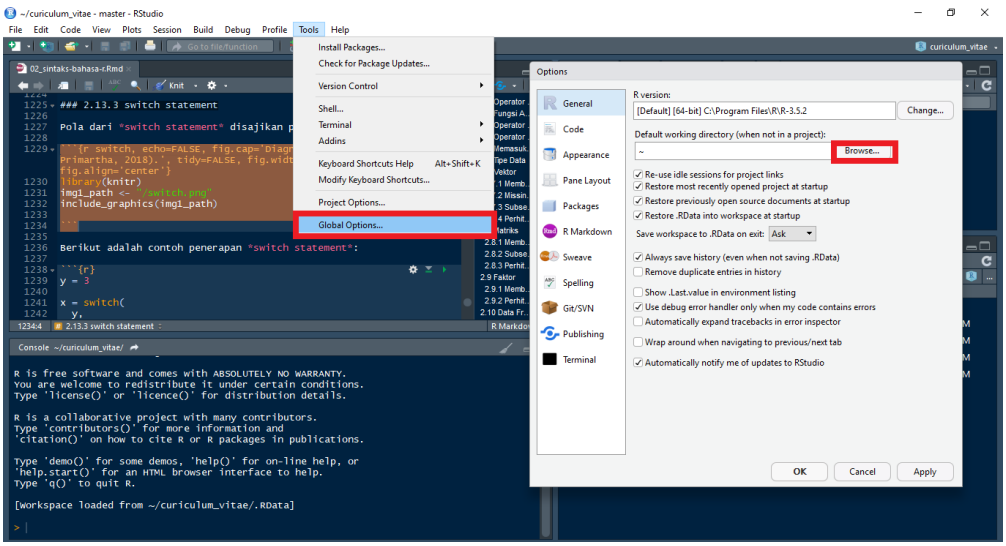

Figure 1.5: Merubah working directory melalui Global options.

1.6.2 Mengubah Lokasi Working Directory Default

Pada proses yang telah penulis jelaskan sebelumnya. Proses perubahan *working directory* hanya berlaku pada saat pekerjaan tersebut dilakukan. Setelah pekerjaan selesai dan kita menjalankan kembali `R` maka *working directory* akan kembali secara default pada working directory lama.

Untuk membuat lokasi default *working directory* pindah, kita dapat melakukannya dengan memilih pada menu: **Tools > Global options > pada "General" klik pada "Browse" dan pilih lokasi working directory yang diinginkan**. Proses tersebut ditampilkan pada Figure 1.5

1.7 Fasilitas Help

Agar dapat menggunakan `R` dengan secara lebih baik, pengetahuan untuk mengakses fasilitas *help* in cukup penting untuk disampaikan. Adapun cara yang dapat digunakan adalah sebagai berikut.

1.7.1 Mencari Help dari Suatu Perintah Tertentu

Untuk memperoleh bantuan terkait suatu perintah tertentu kita dapat menggunakan fungsi `help()`. Secara umum format yang digunakan adalah sebagai berikut:

```
help(nama_perintah)
```

atau dapat juga menggunakan tanda tanya (?) pada awal `nama_perintah` seperti berikut:

```
?nama_perintah
```

Misalkan kita kebingungan terkait bagaimana cara menuliskan perintah untuk menghitung rata-rata suatu vektor. Kita dapat mengetikkan perintah berikut untuk mengakses fasilitas *help*.

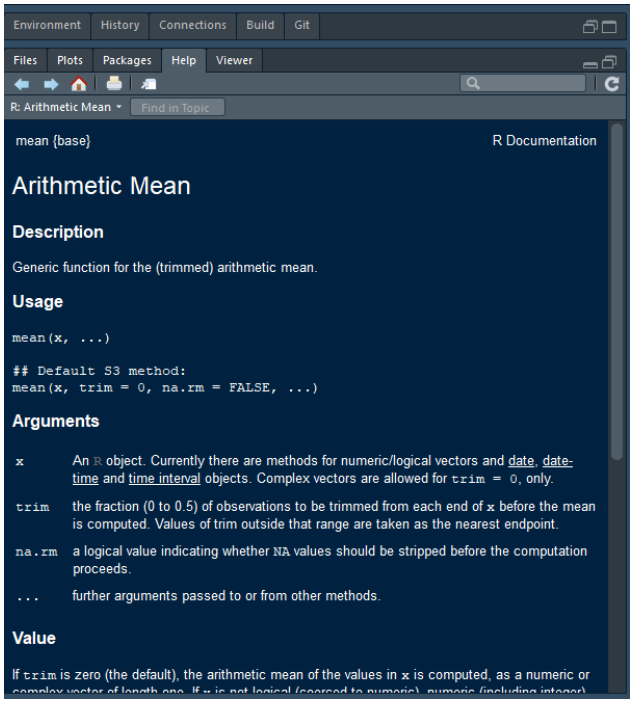

Figure 1.6: Jendela help dokumentasi fungsi mean().

```
help(mean)

#atau
?mean
```

Perintah tersebut akan memunculkan hasil berupa dokumentasi yang ditampilkan pada Figure 1.6.

Keterangan pada jendela pada Figure 1.6 adalah sebagia berikut:

1. Pada bagian jendela kiri atas jendela *help*, diberikan keterangan nama dari perintah yang sedang ditampilkan.
2. Selanjutnya, pada bagian atas dokumen, ditampilkan infomasi terkait nama perintah, dan nama *library* yang memuat perintah tersebut. Pada gambar diatas informasi terkait perintah dan nama *library* ditunjukkan pada teks `mean {base}` yang menunjukkan perintah `mean()` pada paket (*library*) *base* (paket bawaan `R`).
3. Setiap jendela *help* dari suatu perintah tertentu selanjutnya akan memuat bagian-bagian berikut:

- *Title*
- *Description* : deskripsi singkat tentang perintah.
- *Usage* : menampilkan sintaks perintah untuk penggunaan perintah tersebut.
- *Arguments* : keterangan mengenai *argument/input*yang diperlukan pada perintah tersebut.
- *Details* : keterangan lebih lengkap lengkap tentang perintah tersebut.
- *Value* : keterangan tentang *output* suatu perintah dapat diperoleh pada bagian ini.
- *Author(s)* : memberikan keterangan tentang *Author* dari perintah tersebut.
- *References* : seringkali referensi yang dapat digunakan untuk memperoleh keterangan lebih lanjut terhadap suatu perintah ditampilkan pada bagian ini.
- *See also*: bagian ini berisikan daftar perintah/fungsi yang berhubungan erat dengan perintah tersebut.
- *Example* : berisikan contoh-contoh penggunaan perintah tersebut.

Kita juga dapat melihat contoh penggunaan dari perintah tersebut. Untuk melakukannya kita dapat menggunakan fungsi `example()`. Fungsi tersebut akan menampilkan contoh kode penerapan dari fungsi yang kita inginkan. Secara sederhana fungsi tersebut dapat dituliskan sebagai berikut:

```
example(nama_perintah)
```

Untuk mengetahui contoh kode fungsi `mean()`, ketikkan sintaks berikut:

```
example(mean)
```

```
##
## mean> x <- c(0:10, 50)
##
## mean> xm <- mean(x)
##
## mean> c(xm, mean(x, trim = 0.10))
## [1] 8.75 5.50
```

kita juga dapat mencoba kode yang dihasilkan pada console `R`. Berikut adalah contoh penerapannya:

```
# Menghitung rata-rata bilangan 1 sampai 10 dan 50
# membuat vektor
x <- c(0:10, 50)

# Print
x
```

```
##  [1]  0  1  2  3  4  5  6  7  8  9 10 50
```

```
# mean
mean(x)
```

```
## [1] 8.75
```

Pembaca dapat mencoba melakukanya sendiri dengan mengganti nilai yang telah ada serta mencoba contoh kode yang lain.

1.7.2 General Help

Kita juga dapat membaca beberapa dokumen manual yang ada pada `R`. Untuk melakukannya jalankan perintah berikut:

```
help.start()
```

Output yang dihasilkan berupa link pada sejumlah dokumen yang dapat kita klik. Tampilan halaman yang dihasilkan disajikan pada Figure 1.7.

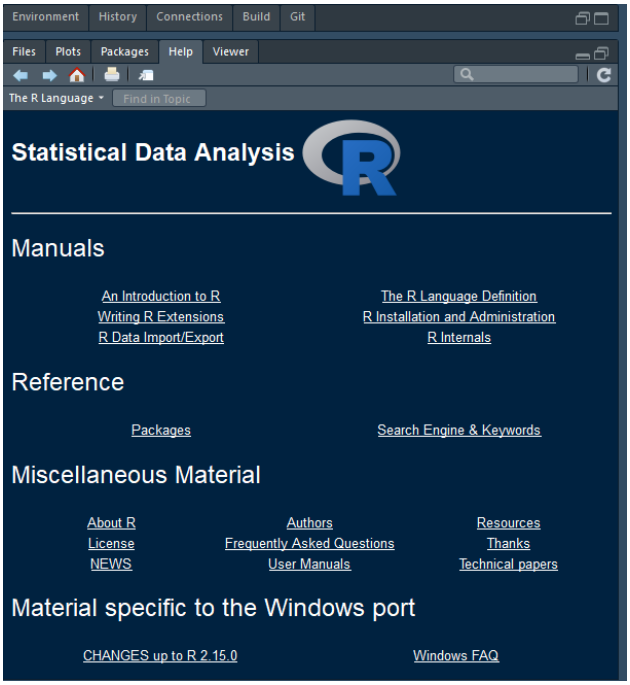

Figure 1.7: Jendela general help dokumentasi fungsi mean().

1.7.3 Fasilitas Help Lainnya

Selain yang telah penulis sebutkan sebelumnya. Kita juga dapat memanfaatkan fasilitas *help* lainnya melalui fungsi `apropos()` dan `help.search()`.

`apropos ()`: mengembalikan daftar objek, berisi pola yang pembaca cari, dengan pencocokan sebagian. Ini berguna ketika pembaca tidak ingat persis nama fungsi yang akan digunakan. Berikut adalah contoh ketika penulis ingin mengetahui fungsi yang digunakan untuk menghitung median.

```
apropos("med")
```

```
## [1] "elNamed"         "elNamed<-"       "median"          "median.default"
## [5] "medpolish"       "runmed"
```

List yang dihasilkan berupa fungsi-fungsi yang memiliki elemen kata "med". Berdasarkan pencaria tersebut penulis dapat mencoba menggunakan fungsi "median" untuk menghitung median.

`help.search ()` (sebagai alternatif ??): mencari dokumentasi yang cocok dengan karakter yang diberikan dengan cara yang berbeda. Ini mengembalikan daftar fungsi yang mengandung istilah yang pembaca cari dengan deskripsi singkat dari fungsi.

Berikut adalah contoh penerapan dari fungsi tersebut:

```
help.search("mean")

# atau
??mean
```

Output yang dihasilkan akan tampak seperti pada Figure 1.8.

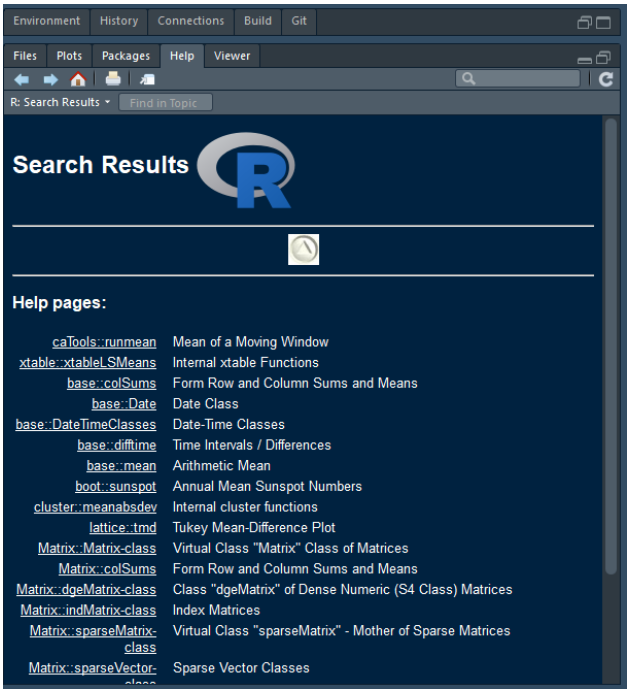

Figure 1.8: Jendela help search dokumentasi fungsi mean().

1.8 Referensi

1. Primartha, R. 2018. **Belajar Machine Learning Teori dan Praktik**. Penerbit Informatika : Bandung
2. Rosadi,D. 2016. **Analisis Statistika dengan R**. Gadjah Mada University Press: Yogyakarta
3. STHDA. Running RStudio and Setting Up Your Working Directory - Easy R Programming .http://www.sthda.com/english/wiki/running-rstudio-and-setting-up-your-working-directory-easy-r-programming#set-your-working-directory
4. STDHA. **Getting Help With Functions In R Programming**. http://www.sthda.com/english/wiki/getting-help-with-functions-in-r-programming .
5. Venables, W.N. Smith D.M. and R Core Team. 2018. **An Introduction to R**. R Manuals.

Chapter 2

Sintaks Bahasa R

2.1 Operator Aritmatika

Proses perhitungan akan ditangani oleh fungsi khusus. `R` akan memahami urutannya secara benar. Kecuali kita secara eksplisit menetapkan yang lain. Sebagai contoh jalankan sintaks berikut:

```
2+4*2
```

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

Bandingkan dengan sintaks berikut:

```
(2+4)*2
```

```
## [1] 12
```

> `R` dapat digunakan sebagai kalkulator

Berdasarkan kedua hasil tersebut dapat disimpulkan bahwa ketika kita tidak menetapkan urutan perhitungan menggunakan tanda kurung, `R` akan secara otomatis akan menghitung terlebih dahulu perkalian atau pembangian.

Operator aritmatika yang disediakan `R` adalah sebagai berikut:

Table 1 Operator Aritmatika `R`

Simbol	Keterangan
+	*Addition*, untuk operasi penjumlahan
-	*Substraction*, untuk operasi pengurangan
*	*Multiplication*, untuk operasi pembagian
/	*Division*, untuk operasi pembagian
^	*Eksponentiation*, untuk operasi pemangkatan
%%	*Modulus*, Untuk mencari sisa pembagian
%/%	*Integer*, Untuk mencari bilangan bulat hasil pembagian saja dan tanpa sisa pembagian

Untuk lebih memahaminya berikut contoh sintaks penerapan operator tersebut.

```
# Addition
5+3
```

```
## [1] 8
```

```
# Substraction
5-3
```

```
## [1] 2
```

```
# Multiplication
5*3
```

```
## [1] 15
```

```
# Division
5/3
```

```
## [1] 1.666667
```

```
# Eksponetiation
5^3
```

```
## [1] 125
```

```
# Modulus
5%%3
```

```
## [1] 2
```

```
# Integer
5%/%3
```

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

> *Note:* Pada `R` tanda `#` berfungsi menambahkan keterangan untuk menjelaskan sebuah sintaks pada `R`.

2.2 Fungsi Aritmetik

Selain fungsi operator aritmetik, pada `R` juga telah tersedia fungsi aritmetik yang lain seperti logaritmik, ekponensial, trigonometri, dll.

1. Logaritma dan eksponensial

Untuk contoh fungsi logaritmik dan eksponensial jalankan sintaks berikut:

```
log2(8) # logaritma basis 2 untuk 8
```

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

```
log10(8) # logaritma basis 10 untuk 8
```

```
## [1] 0.90309
```

```
exp(8) # eksponensial 8
```

```
## [1] 2980.958
```

2. Fungsi trigonometri

fungsi trigonometri yang ditampilkan seperti sin,cos, tan, dll.

```
cos(x) # cos x
sin(x) # Sin x
tan(x) # Tan x
acos(x) # arc-cos x
asin(x) # arc-sin x
atan(x) #arc-tan x
```

Note: x dalam fungsi trigonometri memiliki satuan radian

Berikut adalah salah satu contoh penggunaannya:

```
cos(pi)
```

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

3. Fungsi matematik lainnya

Fungsi lainnya yang dapat digunakan adalah fungsi absolut, akar kuadrat, dll. Berikut adalah contoh sintaks penggunaan fungsi absolut dan akar kuadrat.

```
abs(-2) # nilai absolut -2
```

```
## [1] 2
```

```
sqrt(4) # akar kuadrat 4
```

```
## [1] 2
```

2.3 Operator Relasi

Operator relasi digunakan untuk membandingkan satu objek dengan objek lainnya. Operator yang disediakan `R` disajikan pada Table 2.

Table 2 Operator Relasi `R`

Simbol	Keterangan
“>”	Lebih besar dari
“<”	Lebih Kecil dari
“==”	Sama dengan
“>=”	Lebih besar sama dengan
“<=”	Lebih kecil sama dengan
“!=”	Tidak sama dengan

Berikut adalah penerapan operator pada tabel tersebut:

```
x <- 34
y <- 35

# Operator >
x > y
```

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

```
# Operator <
x < y
```

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

```
# operator ==
x == y
```

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

```
# Operator >=
x >= y
```

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

```
# Operator <=
x <= y
```

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

```
# Operator !=
x != y
```

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

2.4 Operator Logika

Operator logika hanya berlaku pada vektor dengan tipe logical, numeric, atau complex. Semua angka bernilai 1 akan dianggap bernilai logika `TRUE`. Operator logika yang disediakan `R` dapat dilihat pada Table 3.

Table 3 Operator logika `R`

Simbol	Keterangan
&&	Operator logika AND
!	Opeartor logika NOT
&	Operator logika AND element wise
|	Operator logika OR element wise

Penerapannya terdapat pada sintaks berikut:

```
v <- c(TRUE,TRUE, FALSE)
t <- c(FALSE,FALSE,FALSE)

# Operator &&
print(v&&t)
```

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

```
# Operator ||
print(v||t)
```

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

```
# Operator !
print(!v)
```

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

```
# operator &
print(v&t)
```

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

```
# Operator |
print(v|t)
```

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

Note:

operator & dan | akan mengecek logika tiap elemen pada vektor secara berpesangan (sesuai urutan dari kiri ke kanan).

Operator %% dan || hanya mengecek dari kiri ke kanan pada observasi pertama. Misal saat menggunakan && jika observasi pertama TRUE maka observasi pertama pada vektor lainnya akan dicek, namun jika observasi pertama FALSE maka proses akan segera dihentikan dan menghasilkan FALSE.

2.5 Memasukkan Nilai Kedalam Variabel

Variabel pada R dapat digunakan untuk menyimpan nilai. Sebagai contoh jalankan sintaks berikut:

```
# Harga sebuah lemon adalah 500 rupiah
lemon <- 500

# Atau
500 -> lemon

# dapat juga menggunakan tanda "="
lemon = 500
```

Note:

1. `R` memungkinkan penggunaan <-,->, atau = sebagai perintah pengisi nilai variabel
2. `R` bersifat *case-sensitive*. Maksudnya adalah variabel Lemon tidak sama dengan lemon (Besar kecil huruf berpengaruh)

Untuk mengetahui nilai dari objek `lemon` kita dapat menggunakan fungsi `print()` atau mengetikkan nama objeknya secara langsung.

```
# Menggunakan fungsi print()
print(lemon)
```

```
## [1] 500
```

```
# Atau
lemon
```

```
## [1] 500
```

`R` akan menyimpan variabel `lemon` sebagai objek pada memori. Sehingga kita dapat melakukan operasi terhadap objek tersebut seperti mengalikannya atau menjumlahkannya dengan bilangan lain. Sebagai contoh jalankan sintaks berikut:

```
# Operasi perkalian terhadap objek lemon
5*lemon
```

```
## [1] 2500
```

Kita dapat juga mengubah nilai dari objek `lemon` dengan cara menginput nilai baru terhadap objek yang sama. `R` secara otomatis akan menggatikan nilai sebelumnya. Untuk lebih memahaminya jalankan sintaks berikut:

```
lemon <- 1000

# Print lemon
print(lemon)
```

```
## [1] 1000
```

Untuk lebih memahaminya berikut adalah sintaks untuk menghitung volume suatu objek.

```
# Dimensi objek
panjang <- 10
lebar <- 5
tinggi <- 5

# Menghitung volume
volume <- panjang*lebar*tinggi

# Print objek volume
print(volume)
```

```
## [1] 250
```

Untuk mengetahui objek apa saja yang telah kita buat sepanjang artikel ini kita dapang menggunakan fungsi `ls()`.

```
ls()
```

```
##  [1] "A"         "B"         "img1_path" "lebar"     "lemon"
##  [6] "panjang"   "t"         "tinggi"    "v"         "volume"
## [11] "x"         "xm"        "y"
```

Kumpulan objek yang telah tersimpan dalam memori disebut sebagai **workspace**

Untuk menghapus objek pada memori kita dapat menggunakan fungsi `rm()`. Pada sintaks berikut penulis hendak menghapus objek `lemon` dan `volume`.

```
# Menghapus objek lemon dan volume
rm(lemon, volume)

# Tampilkan kembali objek yang tersisa
ls()
```

```
##  [1] "A"         "B"         "img1_path" "lebar"     "panjang"
##  [6] "t"         "tinggi"    "v"         "x"         "xm"
## [11] "y"
```

Note: Setiap variabel atau objek yang dibuat akan menempati sejumlah memori pada komputer sehingga jika kita bekerja dengan jumlah data yang banyak pastikan kita menghapus seluruh objek pada memori sebelum memulai kerja.

2.6 Tipe Data

Data pada `R` dapat dikelompokan berdasarkan beberapa tipe. Tipe data pada `R` disajikan pada Table 4.

Table 4 Tipe Data `R`

Tipe Data	Contoh	Keterangan
Logical	TRUE, FALSE	Nilai Boolean

Tipe Data	Contoh	Keterangan
Numeric	12.3, 5, 999	Segala jenis angka
Integer	23L, 97L, 3L	Bilangan integer (bilangan bulat)
Complex	2i, 3i, 9i	Bilangan kompleks
Character	‘a’, “b”, “123”	Karakter dan string
Raw	Identik dengan “hello”	Segala jenis data yang disimpan sebagai raw bytes

Sintaks berikut adalah contoh dari tipe data pada `R`. Untuk mengetahui tipa data suatu objek kita dapat menggunakan perintah `class()`

```
# Logical
apel <- TRUE
class(apel)
```

```
## [1] "logical"
```

```
# Numeric
x <- 2.3
class(x)
```

```
## [1] "numeric"
```

```
# Integer
y <- 2L
class(y)
```

```
## [1] "integer"
```

```
# Compleks
z <- 5+2i
class(z)
```

```
## [1] "complex"
```

```
# string
w <- "saya"
class(w)
```

```
## [1] "character"
```

```
# Raw
xy <- charToRaw("hello world")
class(xy)
```

```
## [1] "raw"
```

Keenam jenis data tersebut disebut sebagai tipe data atomik. Hal ini disebabkan karena hanya dapat menangani satu tipe data saja. Misalnya hanya numeric atau hanya integer.

Selain menggunakan fungsi `class()`, kita dapat pula menggunakan fungsi `is_numeric()`, `is.character()`, `is.logical()`, dan sebagainya berdasarkan jenis data apa yang ingin kita cek. Berbeda dengan fungsi `class()`, ouput yang dihasilkan pada fungsi seperti `is_numeric()` adalah nilai Boolean sehingga fungsi ini hanya digunakan untuk mengecek apakah jenis data pada objek sama seperti yang kita pikirkan. Sebagai contoh disajikan pada sintaks berikut:

```
data <- 25

# Cek apakah objek berisi data numerik
is.numeric(data)
```

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

```
# Cek apakah objek adalah karakter
is.character(data)
```

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

Kita juga dapat mengubah jenis data menjadi jenis lainnya seperti integer menjadi numerik atau sebaliknya. Fungsi yang digunakan adalah `as.numeric()` jika ingin mengubah suatu jenis data menjadi numerik. Fungsi lainnya juga dapat digunakan sesuai dengan kita ingin mengubah jenis data objek menjadi jenis data lainnya.

```
# Integer
apel <- 2L

# Ubah menjadi numerik
as.numeric(apel)
```

```
## [1] 2
```

```
# Cek
is.numeric(apel)
```

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

```
# Logical
nangka <- TRUE

# Ubah logical menjadi numeric
as.numeric(nangka)
```

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

```
# Karakter
minum <- "minum"

# ubah karakter menjadi numerik
as.numeric(minum)
```

```
## Warning: NAs introduced by coercion
```

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

> **Note:** Konversi karakter menjadi numerik akan menghasilkan output NA (*not available*). `R` tidak mengetahui bagaimana cara merubah karakter menjadi bentuk numerik.

Berdasarkan Tabel 2, vektor karakter dapat dibuat menggunakan tanda kurung baik *double quote* (“”) maupun *single quote* (’).Jika pada teks yang kita tuliskan mengandung *quote* maka kita harus menghentikannya menggunakan tanda ( ). Sbegai contoh kita ingin menuliskan ‘**My friend’s name is “Adi”**, pada sintaks akan dituliskan:

```
'My friend\`s name is "Adi"'
```

```
## [1] "My friend`s name is \"Adi\""
```

```
# Atau

"My friend's name \"Adi\""
```

```
## [1] "My friend's name \"Adi\""
```

2.7 Vektor

Vektor merupakan kombinasi berbagai nilai (numerik, karakter, logical, dan sebagainya berdasarkan jenis input data) pada objek yang sma. Pada contoh kasus berikut, pembaca akan memiliki sesuai jenis data input yaitu**vektor numerik**, **vector karakter**, **vektor logical**, dll.

2.7.1 Membuat vektor

Vektor dibuat dengan menggunakan fungsi `c()`(concatenate) seperti yang disajikan pada sintaks berikut:

```
# membuat vektor numerik
x <- c(3,3.5,4,7)
x # print vektor
```

```
## [1] 3.0 3.5 4.0 7.0
```

```
# membuat vektor karakter
y <- c("Apel", "Jeruk", "Rambutan", "Salak")
y # print vektor
```

```
## [1] "Apel"     "Jeruk"    "Rambutan" "Salak"
```

```
# membuat vektor logical
t <- c("TRUE", "FALSE", "TRUE")
t # print vektor
```

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

selain menginput nilai pada vektor, kita juga dapat memberi nama nilai setiap vektor menggunakan fungsi `names()`.

```
# Membuat vektor jumlah buah yang dibeli
Jumlah <- c(5,5,6,7)
names(Jumlah) <- c("Apel", "Jeruk", "Rambutan", "Salak")

# Atau
Jumlah <- c(Apel=5, Jeruk=5, Rambutan=6, Salak=7)

# Print
Jumlah
```

```
##     Apel    Jeruk Rambutan    Salak
##        5        5        6        7
```

> **Note:** Vektor hanya dapat memuat satu buah jenis data. Vektor hanya dapat mengandung jenis data numerik saja, karakter saja, dll.

Untuk menentukan panjang sebuah vektor kita dapat menggunakan fungsi `lenght()`.

```
length(Jumlah)
```

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

2.7.2 Missing Values

Seringkali nilai pada vektor kita tidak lengkap atau terdapat nilai yang hilang (*missing value*) pada vektor. *Missing value* pada `R` dilambangkan oleh `NA`(*not available*). Berikut adalah contoh vektor dengan *missing value*.

```
Jumlah <- c(Apel=5, Jeruk=NA, Rambutan=6, Salak=7)
```

Untuk mengecek apakah dalam objek terdapat *missing value* dapat menggunakan fungsi `is.na()`. ouput dari fungsi tersebut adalah nilai Boolean. Jika terdapat *Missing value*, maka output yang dihasilkan akan memberikan nilai `TRUE`.

```
is.na(Jumlah)
```

```
##     Apel    Jeruk Rambutan    Salak
##    FALSE     TRUE    FALSE    FALSE
```

> **Note:**
>
> Selain NA terdapat NaN (*not a number*) sebagai *missing value8*. Nilai tersebut muncul ketika fungsi matematika yang digunakan pada proses perhitungan tidak bekerja sebagaimana mestinya. Contoh: $0/0 = \text{NaN}$
>
> `is.na()` juga akan menghasilkan nilai `TRUE` pada NaN. Untuk membedakannya dengan NA dapat digunakan fungsi `is.nan()`.

2.7.3 Subset Pada Vektor

Subseting vector terdiri atas tiga jenis, yaitu: *positive indexing*, *Negative Indexing*, dan .

- **Positive indexing**: memilih elemen vektor berdasarkan posisinya (indeks) dalam kurung siku.

```
# Subset vektor pada urutan kedua
Jumlah[2]
```

```
## Jeruk
##    NA
```

```
# Subset vektor pada urutan 2 dan 4
Jumlah[c(2, 4)]
```

```
## Jeruk Salak
##    NA     7
```

Selain melalui urutan (indeks), kita juga dapat melakukan subset berdasarkan nama elemen vektornya.

```
Jumlah["Jeruk"]
```

```
## Jeruk
##    NA
```

> **Note:** Indeks pada `R` dimulai dari 1. Sehingga kolom atau elemen pertama vektor dimulai dari [1]

- **Negative indexing**: mengecualikan (*exclude*) elemen vektor.

```
# mengecualikan elemen vektor 2 dan 4
Jumlah[-c(2,4)]
```

```
##     Apel Rambutan
##        5        6
```

```
# mengecualikan elemen vektor 1 sampai 3
Jumlah[-c(1:3)]
```

```
## Salak
##     7
```

- **Subset berdasarkan vektor logical**: Hanya, elemen-elemen yang nilai yang bersesuaian dalam vektor pemilihan bernilai TRUE, akan disimpan dalam subset.

> **Note:** panjang vektor yang digunakan untuk subset harus sama.

```
Jumlah <- c(Apel=5, Jeruk=NA, Rambutan=6, Salak=7)

# selecting vector
merah <- c(TRUE, FALSE, TRUE, FALSE)

# Subset
Jumlah[merah==TRUE]
```

```
##     Apel Rambutan
##        5        6
```

```
# Subset untuk elemen vektor bukan missing value
Jumlah[!is.na(Jumlah)]
```

```
##     Apel Rambutan    Salak
##        5        6        7
```

2.7.4 Perhitungan Menggunakan Vektor

Jika pembaca melakukan operasi dengan vektor, operasi akan diterapkan ke setiap elemen vektor. Contoh disediakan pada sintaks di bawah ini:

```
pendapatan <- c(2000, 1800, 2500, 3000)
names(pendapatan) <- c("Andi", "Joni", "Lina", "Rani")
pendapatan
```

```
## Andi Joni Lina Rani
## 2000 1800 2500 3000
```

```
# Kalikan pendapatan dengan 3
pendapatan*3
```

```
## Andi Joni Lina Rani
## 6000 5400 7500 9000
```

Seperti yang dapat dilihat, `R` mengalikan setiap elemen dengan bilangan pengali.

Kita juga dapat mengalikan vektor dengan vektor lainnya.Contohnya disajikan pada sintaks berikut:

```
# membuat vektor dengan panjang sama dengan dengan vektor pendapatan
coefs <- c(2, 1.5, 1, 3)

# Mengalikan pendapatan dengan vektor coefs
pendapatan*coefs
```

```
## Andi Joni Lina Rani
## 4000 2700 2500 9000
```

Berdasarkan sintaks tersebut dapat terlihat bahwa operasi matematik terhadap masing-masing vektor dapat berlangsung jika panjang vektornya sama.

Berikut adalah fungsi lain yang dapat digunakan pada operasi matematika vektor.

```
max(x) # memperoleh nilai maksimum x
min(x) # memperoleh nilai minimum x
range(x) # memperoleh range vektor x
length(x) # memperoleh jumlah elemen vektor x
sum(x) # memperoleh total penjumlahan elemen vektor x
prod(x) # memeperoleh produk elemen vektor x
mean(x) # memperoleh nilai rata-rata seluruh elemen vektor x
sd(x) # standar deviasi vektor x
var(x) # varian vektor x
sort(x) # mengurutkan elemen vektor x dari yang terbesar
```

Contoh penggunaan fungsi tersebut disajikan beberapa pada sintaks berikut:

```
# Menghitung range pendapatan
range(pendapatan)
```

```
## [1] 1800 3000
```

```
# menghitung rata-rata dan standar deviasi pendapatan
mean(pendapatan)
```

```
## [1] 2325
```

```
sd(pendapatan)
```

```
## [1] 537.7422
```

2.8 Matriks

Matriks seperti Excel sheet yang berisi banyak baris dan kolom (kumpulan bebrapa vektor). Matriks digunakan untuk menggabungkan vektor dengan tipe yang sama, yang bisa berupa numerik, karakter, atau logis. Matriks digunakan untuk menyimpan tabel data dalam R. Baris-baris matriks pada umumnya adalah individu / pengamatan dan kolom adalah variabel.

2.8.1 Membuat matriks

Untuk membuat matriks kita dapat menggunakan fungsi `cbind()` atau `rbind()`. Berikut adalah contoh sintaks untuk membuat matriks.

```
# membuat vektor numerik
col1 <- c(5, 6, 7, 8, 9)
col2 <- c(2, 4, 5, 9, 8)
col3 <- c(7, 3, 4, 8, 7)

# menggabungkan vektor berdasarkan kolom
my_data <- cbind(col1, col2, col3)
my_data
```

```
##      col1 col2 col3
## [1,]    5    2    7
## [2,]    6    4    3
## [3,]    7    5    4
## [4,]    8    9    8
## [5,]    9    8    7
```

```
# Mengubah atau menambahkan nama baris
rownames(my_data) <- c("row1", "row2", "row3", "row4", "row5")
my_data
```

```
##      col1 col2 col3
## row1    5    2    7
## row2    6    4    3
## row3    7    5    4
## row4    8    9    8
## row5    9    8    7
```

Note:

- **cbind()**: menggabungkan objek `R` berdasarkan kolom
- **rbind()**: menggabungkan objek `R` berdasarkan baris
- **rownames()**: mengambil atau menetapkan nama-nama baris dari objek seperti-matriks
- **colnames()**: mengambil atau menetapkan nama-nama kolom dari objek seperti-matriks

Kita dapat melakukan tranpose (merotasi matriks sehingga kolom menjadi baris dan sebaliknya) menggunakan fungsi `t()`. Berikut adalah contoh penerapannya:

```
t(my_data)
```

```
##      row1 row2 row3 row4 row5
## col1    5    6    7    8    9
## col2    2    4    5    9    8
## col3    7    3    4    8    7
```

Selain melalui pembentukan sejumlah objek vektor, kita juga dapat membuat matriks menggunakan fungsi `matrix()`. Secara sederhana fungsi tersebut dapat dituliskan sebagai berikut:

```
matrix(data = NA, nrow = 1, ncol = 1, byrow = FALSE,
       dimnames = NULL)
```

Note:

- **data**: vektor data opsional
- **nrow**, **ncol**: jumlah baris dan kolom yang diinginkan, masing-masing.
- **byrow**: nilai logis. Jika FALSE (default) matriks diisi oleh kolom, jika tidak, matriks diisi oleh baris.
- **dimnames**: Daftar dua vektor yang memberikan nama baris dan kolom masing-masing.

Dalam kode `R` di bawah ini, data input memiliki panjang 6. Kita ingin membuat matriks dengan dua kolom. Kita tidak perlu menentukan jumlah baris (di sini `nrow = 3`). `R` akan menyimpulkan ini secara otomatis. Matriks diisi kolom demi kolom saat argumen `byrow = FALSE`. Jika kita ingin mengisi matriks dengan baris, gunakan `byrow = TRUE`. Berikut adalah contoh pembuatan matriks menggunakan fungsi `matrix()`.

```
data <- matrix(
           data = c(1,2,3, 11,12,13),
           nrow = 2, byrow = TRUE,
           dimnames = list(c("row1", "row2"), c("C.1", "C.2", "C.3"))
           )
data
```

```
##      C.1 C.2 C.3
## row1   1   2   3
## row2  11  12  13
```

Untuk mengetahui dimensi dari suatu matriks, kita dapat menggunakan fungsi `ncol()` untuk mengetahui jumlah kolom matriks dan `nrow()` untuk mengetahui jumlah baris pada matriks. Berikut adalah contoh penerapannya:

```
# mengetahui jumlah kolom
ncol(my_data)
```

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

```
# mengetahui jumlah baris
nrow(my_data)
```

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

Jika ingin memperoleh ringkasan terkait dimensi matriks kita juga dapat mengunakan fungsi `dim()` untuk mengetahui jumlah baris dan kolom matriks. Berikut adalah contoh penerapannya:

```
dim(my_data) # jumlah baris dan kolom
```

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

2.8.2 Subset Pada Matriks

Sama dengan vektor, subset juga dapat dilakukan pada matriks. Bedanya subset dilakukan berdasarkan baris dan kolom pada matriks.

- **Memilih baris/kolom** berdasarkan pengindeksan positif

baris atau kolom dapat diseleksi menggunakan format `data[row, col]`. Cara selesi ini sama dengan vektor, bedanya kita harus menetukan baris dan kolom dari data yang akan kita pilih. Berikut adalah contoh penerapannya:

```
# Pilih baris ke-2
my_data[2,]
```

```
## col1 col2 col3
##    6    4    3
```

```
# Pilih baris 2 sampai 4
my_data[2:4,]
```

```
##      col1 col2 col3
## row2    6    4    3
## row3    7    5    4
## row4    8    9    8
```

```
# Pilih baris 2 dan 4
my_data[c(2,4),]
```

```
##      col1 col2 col3
## row2    6    4    3
## row4    8    9    8
```

```
# Pilih baris 2 dan kolom 3
my_data[2, 3]
```

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

- **Pilih berdasarkan nama baris/kolom**

Berikut adalah contoh subset berdasarkan nama baris atau kolom.

```
# Pilih baris 1 dan kolom 3
my_data["row1","col3"]
```

```
## [1] 7
```

```
# Pilih baris 1 sampai 4 dan kolom 3
baris <- c("row1","row2","row3")
my_data[baris, "col3"]
```

```
## row1 row2 row3
##    7    3    4
```

- **Kecualikan baris/kolom** dengan pengindeksan negatif

Sama seperti vektor pengecualian data dapat dilakukan di matriks menggunakan pengindeksan negatif. Berikut cara melakukannya:

```
# Kecualikan baris 2 dan 3 serta kolom 3
my_data[-c(2,3), -3]
```

```
##      col1 col2
## row1    5    2
## row4    8    9
## row5    9    8
```

- **Pilihan dengan logik**

Dalam kode `R` di bawah ini, misalkan kita ingin hanya menyimpan baris di mana col3> = 4:

```
col3 <- my_data[, "col3"]
my_data[col3 >= 4, ]
```

```
##      col1 col2 col3
## row1    5    2    7
## row3    7    5    4
## row4    8    9    8
## row5    9    8    7
```

2.8.3 Perhitungan Menggunakan Matriks

_ Kita juga dapat melakukan operasi matematika pada matriks. Pada operasi matematika pada matriks proses yang terjadi bisa lebih kompleks dibanding pada vektor, dimana kita dapat melakukan operasi untuk memperoleh gambaran data pada tiap kolom atau baris.

Berikut adalah contoh operasi matematika sederhana pada matriks:

```
# mengalikan masing-masing elemen matriks dengan 2
my_data*2
```

```
##      col1 col2 col3
## row1   10    4   14
## row2   12    8    6
## row3   14   10    8
## row4   16   18   16
## row5   18   16   14
```

```
# memperoleh nilai log basis 2 pada masing-masing elemen matriks
log2(my_data)
```

```
##          col1     col2     col3
## row1 2.321928 1.000000 2.807355
## row2 2.584963 2.000000 1.584963
## row3 2.807355 2.321928 2.000000
## row4 3.000000 3.169925 3.000000
## row5 3.169925 3.000000 2.807355
```

Seperti yang telah penulis jelaskan sebelumnya, kita juga dapat melakukan operasi matematika untuk memperoleh hasil penjumlahan elemen pada tiap baris atau kolom dengan menggunakan fungsi `rowSums()` untuk baris dan `colSums()` untuk kolom.

```
# Total pada tiap kolom
colSums(my_data)
```

```
## col1 col2 col3
##   35   28   29
```

```
# Total pada tiap baris
rowSums(my_data)
```

```
## row1 row2 row3 row4 row5
##   14   13   16   25   24
```

Jika kita tertarik untuk mencari nilai rata-rata tiap baris arau kolom kita juga dapat menggunakan fungsi `rowMeans()` atau `colMeans()`. Berikut adalah contoh penerapannya:

```
# Rata-rata tiap baris
rowMeans(my_data)
```

```
##     row1     row2     row3     row4     row5
## 4.666667 4.333333 5.333333 8.333333 8.000000
```

```
# Rata-rata tiap kolom
colMeans(my_data)
```

```
## col1 col2 col3
##  7.0  5.6  5.8
```

Kita juga dapat melakukan perhitungan statistika lainnya menggunakan fungsi `apply()`. Berikut adalah format sederhananya:

```
apply(x, MARGIN, FUN)
```

Note:

- x : data matriks
- MARGIN : Nilai yang dapat digunakan adalah 1 (untuk operasi pada baris) dan 2 (untuk operasi pada kolom)
- FUN : fungsi yang diterapkan pada baris atau kolom

untuk mengetahui fungsi (`FUN`) apa saja yang dapat diterapkan pada fungsi `apply()` jalankan sintaks bantuan berikut:

```
help(apply)
```

Berikut adalah contoh penerapannya:

```
# Rata-rata pada tiap baris
apply(my_data, 1, mean)
```

```
##     row1     row2     row3     row4     row5
## 4.666667 4.333333 5.333333 8.333333 8.000000
```

```
# Median pada tiap kolom
apply(my_data, 2, median)
```

```
## col1 col2 col3
##    7    5    7
```

2.9 Faktor

Dalam bahasa `R` , faktor merupakan verktor dengan level. Level disimpan sebagai `R` Character. Jika kita menggunakan SPSS maka factor ini akan sama dengan jenis data numerik atau ordinal.

Faktor merepresentasikan kategori atau grup pada data. Untuk membuat faktor pada `R`, kita dapat menggunakan fungsi `factor()`.

2.9.1 Membuat Variabel Faktor

Berikut adalah contoh sintaks pembuatan variabel faktor.

```
# membuat variabel faktor
faktor <- factor(c(1,2,1,2))
faktor
```

```
## [1] 1 2 1 2
## Levels: 1 2
```

Pada sintaks tersebut objek faktor terdiri atas dua buah kategori atau pada `R` disebut sebagai **factor levels**. Kita dapat mengecek factor levels menggunakan fungsi `levels()`.

```
levels(faktor)
```

```
## [1] "1" "2"
```

Kita juga dapat memberikan label atau mengubah level pada faktor. Berikut adalah contoh bagaimana kita melakukannya:

```
# Ubah level
levels(faktor) <- c("baik","tidak_baik")
faktor
```

```
## [1] baik       tidak_baik baik       tidak_baik
## Levels: baik tidak_baik
```

```
# Ubah urutan level
faktor <- factor(faktor,
                 levels = c("tidak_baik","baik"))
faktor
```

```
## [1] baik       tidak_baik baik       tidak_baik
## Levels: tidak_baik baik
```

Note:

- Fungsi `is.factor()` dapat digunakan untuk mengecek apakah sebuah variabel adalah faktor. Hasil yang dimunculkan dapat berupa TRUE (jika faktor) atau FALSE (jika bukan)
- Fungsi `as.factor()` dapat digunakan untuk merubah sebuah variabel menjadi faktor.

```
# Cek jika objek faktor adalah faktor
is.factor(faktor)
```

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

```
# Cek jika objek Jumlah adalah faktor
is.factor(Jumlah)
```

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

```
# Ubah objek Jumlah menjadi faktor
as.factor(Jumlah)
```

```
##     Apel    Jeruk Rambutan    Salak
##        5     <NA>        6        7
## Levels: 5 6 7
```

2.9.2 Perhitungan Menggunakan Faktor

Jika kita ingin mengetahui jumlah masing-masing observasi pada masing-masing faktor, kita dapat menggunakan fungsi `summary()`. Berikut adalah contoh penerapannya:

```
summary(faktor)
```

```
## tidak_baik       baik
##          2          2
```

Pada contoh perhitungan menggunakan vektor kita telah membuat objek `pendapatan`. Pada objek tersebut kita ingin menghitung nilai rata-rata pendapatan berdasarkan objek faktor. Untuk melakukannya kita dapat menggunakan fungsi `tapply()`.

```
pendapatan
```

```
## Andi Joni Lina Rani
## 2000 1800 2500 3000
```

```
faktor
```

```
## [1] baik       tidak_baik baik       tidak_baik
## Levels: tidak_baik baik
```

```
# Rata-rata pendapatan dan simpan sebagai objek dengan nama:
# mean_pendapatan
mean_pendapatan <- tapply(pendapatan, faktor, mean)
mean_pendapatan
```

```
## tidak_baik       baik
##       2400       2250
```

```r
# Hitung ukuran/panjang masing-masing grup
tapply(pendapatan, faktor, length)
```

```
## tidak_baik       baik
##          2          2
```

Untuk mengetahui jumlah masing-masing observasi masing-masing factor levels kita juga dapat menggunakan fungsi `table()`. Fungsi tersebut akan membuat frekuensi tabel pada masing-masing factor levels atau yang dikenal sebagai *contingency table*.

```r
table(faktor)
```

```
## faktor
## tidak_baik       baik
##          2          2
```

```r
# Cross-tabulation antara
# faktor dan pendapatan
table(pendapatan, faktor)
```

```
##           faktor
## pendapatan tidak_baik baik
##       1800          1    0
##       2000          0    1
##       2500          0    1
##       3000          1    0
```

2.10 Data Frames

Data frame merupakan kumpulan vektor dengan panjang sama atau dapat pula dikatan sebagai matriks yang memiliki kolom dengan jenis data yang berbeda-beda (numerik, karakter, logical). Pada data frame terdapat baris dan kolom. Baris disebut sebagai observasi, sedangkan kolom disebut sebagai variabel. Sehingga dapat dikatakan bahwa setiap observasi akan memiliki satu atau beberapa variabel.

2.10.1 Membuat Data Frame

Data frame dapat dibuat menggunakan fungsi `data.frame()`. Berikut adalah contoh cara membuat data frame:

```r
# Membuat data frame
nama <- c("Andi","Rizal","Ani","Ina")
pendapatan <- c(1000, 2000, 3500, 500)
tinggi <- c(160, 155, 170, 146)
usia <- c(35, 40, 25, 27)
menikah <- c(TRUE, FALSE, TRUE, TRUE)

data_teman <- data.frame(nama = nama,
                         gaji = pendapatan,
                         tinggi = tinggi,
```

```
                        menikah = menikah)

data_teman
```

```
##    nama gaji tinggi menikah
## 1  Andi 1000    160    TRUE
## 2 Rizal 2000    155   FALSE
## 3   Ani 3500    170    TRUE
## 4   Ina  500    146    TRUE
```

Untuk mengecek apakah objek `data_teman` merupakan data frame, kita dapat menggunakan fungsi `is.data.frame()`. Jika hasilnya TRUE, maka objek tersebut adalah data frame. Berikut adalah contoh penerapannya:

```
is.data.frame(data_teman)
```

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

> **Note:** untuk konversi objek menjadi data frame, kita dapat menjalankan fungsi `as.data.frame()`.

2.10.2 Subset Pada Data Frame

Subset pada data frame sebenarnya tidak berbeda dengan subset pada matriks. Bedanya adalah kita juga bisa melakukan subset langsung terhadap nama variabel menggunakan dollar sign. Untuk lebih memahaminya berikut adalah jenis subset pada data frame.

- **Pengindeksan positif** menggunakan nama dan lokasi.

```
# Subset menggunakan dollar sign
data_teman$nama
```

```
## [1] Andi  Rizal Ani   Ina
## Levels: Andi Ani Ina Rizal
```

```
# atau
data_teman[, "nama"]
```

```
## [1] Andi  Rizal Ani   Ina
## Levels: Andi Ani Ina Rizal
```

```
# subset baris 1 sampai 3 serta kolom 1 dan 3
data_teman[1:3, c(1,3)]
```

```
##    nama tinggi
## 1  Andi    160
## 2 Rizal    155
## 3   Ani    170
```

- **Pengindeksan negatif**

```
# Kecualikan kolom nama
data_teman[,-1]
```

```
##   gaji tinggi menikah
## 1 1000    160    TRUE
## 2 2000    155   FALSE
## 3 3500    170    TRUE
## 4  500    146    TRUE
```

- **Pengideksan berdasarkan karakteristik**

Kita ingin memilih data dengan kriteria teman yang telah menikah

```
data_teman[data_teman$menikah==TRUE, ]
```

```
##   nama gaji tinggi menikah
## 1 Andi 1000    160    TRUE
## 3  Ani 3500    170    TRUE
## 4  Ina  500    146    TRUE
```

```
# Tampilkan hanya kolom nama dan gaji untuk yang telah menikah
data_teman[data_teman$menikah==TRUE, 1:2]
```

```
##   nama gaji
## 1 Andi 1000
## 3  Ani 3500
## 4  Ina  500
```

kita juga dapat menggunakan fungsi `subset()` agar lebih mudah. Berikut adalah contoh penerapannya:

```
# subset terhadap teman yang berusia >=30 tahun
subset(data_teman, usia>=30)
```

```
##    nama gaji tinggi menikah
## 1  Andi 1000    160    TRUE
## 2 Rizal 2000    155   FALSE
```

Opsi lain adalah menggunakan fungsi `attach()` dan `detach()`. Fungsi `attach()` mengambil data frame dan membuat kolomnya dapat diakses hanya dengan memberikan nama mereka.

```
# attach data frame
attach(data_teman)
```

```
## The following objects are masked _by_ .GlobalEnv:
##
##     menikah, nama, tinggi
```

```
# ==== memulai data manipulation ====
data_teman[usia>=30]
```

```
##    nama gaji
## 1  Andi 1000
## 2 Rizal 2000
## 3   Ani 3500
## 4   Ina  500
```

```
# ==== mengakhiri data manipulation ====
# detach data frame

detach(data_teman)
```

2.10.3 Memperluas Data Frame

Kita dapat juga memperluas data frame dengan cara menambahkan variabel atau kolombaru pada data frame. Pada contoh kali ini penulis akan menambahkan kolom pendidikan terakhir pada objek `data_teman`. Berikut adalah sintaks yang digunakan.

```
# membuat vektor pendidikan
pendidikan <- c("S1","S2","D3","D1")

# menambahkan variabel pendidikan pada data frame
data_teman$pendidikan <- pendidikan
```

```
# atau
cbind(data_teman, pendidikan=pendidikan)
```

2.10.4 Perhitungan Pada Data Frame

Perhitungan pada variabel numerik data frame pada dasarnya sama dengan perhitungan pada matriks. kita dapat menggunakan fungsi `rowSums()`, `colSums()`, `rowMeans()` dan `apply()`. Proses perhitungan dan manipulasi pada data frame akan dibahas pada sesi yang lain secara lebih detail.

2.11 List

List adalah kumpulan objek yang diurutkan, yang dapat berupa vektor, matriks, data frame, dll. Dengan kata lain, daftar dapat berisi semua jenis objek `R`.

2.11.1 Membuat List

List dapat dibuat menggunakan fungsi `list()`. Berikut disajikan contoh sebuah list sebuah keluarga:

```
# Membuat list keluarga
keluarga <- list(
  ayah = "Budi",
  usia_ayah = 48,
```

```
  ibu  = "Ani",
  usia_ibu = "47",
  anak = c("Andi", "Adi"),
  usia_anak = c(15,10)
  )

# Print
keluarga
```

```
## $ayah
## [1] "Budi"
##
## $usia_ayah
## [1] 48
##
## $ibu
## [1] "Ani"
##
## $usia_ibu
## [1] "47"
##
## $anak
## [1] "Andi" "Adi"
##
## $usia_anak
## [1] 15 10
```

```
# Nama elemen dalam list
names(keluarga)
```

```
## [1] "ayah"      "usia_ayah" "ibu"       "usia_ibu"  "anak"      "usia_anak"
```

```
# Jumlah elemen pada list
length(keluarga)
```

```
## [1] 6
```

2.11.2 Subset List

Kita dapat memilih sebuah elemen pada list dengan menggunakan nama elemen atau indeks dari elemen tersebut. Berikut adalah contoh penerapannya:

```
# Subset berdasarkan nama
# mengambil elemen usia_ayah
keluarga$usia_ayah
```

```
## [1] 48
```

```
# Atau
keluarga[["usia_ayah"]]
```

```
## [1] 48
```

```
# Subset berdasarkan indeks
keluarga[[2]]
```

```
## [1] 48
```

```
# subset elemen pertama pada keluarga[[5]]
keluarga[[5]][1]
```

```
## [1] "Andi"
```

2.11.3 Memperluas List

Kita juga dapat menambahkan elemen pada list yang telah kita buat. Pada contoh list sebelumnya penulis akan menambahkan elemen keluarga yang lain seperti berikut:

```
# Menambahkan kakek dan nenek pada list
keluarga$kakek <- "Suprapto"
keluarga$nenek <- "Sri"

# Print
keluarga
```

```
## $ayah
## [1] "Budi"
##
## $usia_ayah
## [1] 48
##
## $ibu
## [1] "Ani"
##
## $usia_ibu
## [1] "47"
##
## $anak
## [1] "Andi" "Adi"
##
## $usia_anak
## [1] 15 10
##
## $kakek
## [1] "Suprapto"
##
## $nenek
## [1] "Sri"
```

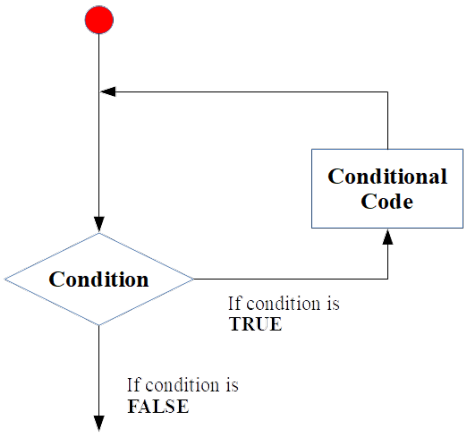

Figure 2.1: Diagram umum loop (sumber: Primartha, 2018).

Kita juga dapat menggabungkan beberapa list menjadi satu. Berikut adalah format sederhana bagaimana cara menggabungkan beberapa list menjadi satu:

```
list_baru <- c(list_a, list_b, list_c, ...)
```

2.12 Loop

Loop merupakan kode program yang berulang-ulang. *Loop* berguna saat kita ingin melakukan sebuah perintah yang perlu dijalankan berulang-ulang seperti melakukan perhitungan maupaun melakukan visualisasi terhadap banyak variabel secara serentak. Hal ini tentu saja membantu kita karena kita tidak perlu menulis sejumlah sintaks yang berulang-ulang. Kita hanya perlu mengatur *statement* berdasarkan hasil yang kita harapkan.

Pada `R` bentuk *loop* dapat bermacam-macam ("*for loop*","*while loop*", dll). `R` menyederhanakan bentuk *loop* ini dengan menyediakan sejumlah fungsi seperti `apply()`,`tapply()`, dll. Sehingga `loop` jarang sekali muncul dalam kode `R`. Sehingga `R` sering disebut sebagai *loopless loop*.

Meski *loop* jarang muncul bukan berarti kita tidak akan melakukannya. Terkadang saat kita melakukan komputasi statistik atau matematik dan belum terdapat paket yang mendukung proses tersebut, sering kali kita akan membuat sintaks sendiri berdasarkan algoritma metode tersebut. Pada algoritma tersebut sering pula terdapat *loop* yang diperlukan selama proses perhitungan. Secara sederhana diagram umum loop ditampilkan pada Figure 2.1

2.12.1 For Loop

Mengulangi sebuah *statement* atau sekelompok *statement* sebanyak nilai yang ditentukan di awal. Jadi operasi akan terus dilakukan sampai dengan jumlah yang telah ditetapkan di awal atau dengan kata lain tes kondisi (Jika jumlah pengulangan telah cukup) hanya akan dilakukan di akhir. Secara sederhana bentuk dari *for loop* dapat dituliskan sebagai berikut:

```
for (value in vector){
  statements
}
```

Berikut adalah contoh sintaks penerapan *for loop*:

```
# Membuat vektor numerik
vektor <- c(1:5)

# loop
for(i in vektor){
  print(i)
}
```

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

Loop akan dimulai dari blok *statement for* sampai dengan `print(i)`. Berdasarkan *loop* pada contoh tersebut, *loop* hanya dilakukan sebanyak 5 kali sesuai dengan jumlah vektor yang ada.

2.12.2 While Loop

While loop merupakan loop yang digunakan ketika kita telah menetapkan *stop condition* sebelumnya. Blok *statement*/kode yang sama akan terus dijalankan sampai *stop condition* ini tercapai. *Stop condition* akan di cek sebelum melakukan proses *loop*. Berikut adalah pola dari *while loop* dapat dituliskan sebagai berikut:

```
while (test_expression){
  statement
}
```

Berikut adalah contoh penerapan dari *while loop*:

```
coba <- c("Contoh")
counter <- 1

# loop
while (counter<5){
  # print vektor
  print(coba)
  # tambahkan nilai counter sehingga proses terus berlangsung sampai counter = 5
  counter <- counter + 1
}
```

```
## [1] "Contoh"
## [1] "Contoh"
## [1] "Contoh"
## [1] "Contoh"
```

Loop akan dimulai dari blok *statement while* sampai dengan *counter* <- 1. *Loop* hanya akan dilakukan sepanjang nilai *counter* < 5.

2.12.3 Repeat Loop

Repeat loop akan menjalankan *statement*/kode yang sama berulang-ulang hingga *stop condition* tercapai. Berikut adalah pola dari *repeat loop*.

```
repeat {
  commands
  if(condition){
    break
  }
}
```

Berikut adalah contoh penerapan dari *repeat loop*:

```
coba <- c("contoh")
counter <- 1
repeat {
  print(coba)
  counter <- counter + 1
  if(counter < 5){
break
  }
}
```

```
## [1] "contoh"
```

Loop akan dimulai dari blok *statement while* sampai dengan *break*. *Loop* hanya akan dilakukan sepanjang nilai *counter* < 5. Hasil yang diperoleh berbeda dengan *while loop*, dimana kita memperoleh 4 buah kata "contoh". Hal ini disebabkan karena *repeat loop* melakukan pengecekan *stop condition* tidak di awal loop seperti *while loop* sehingga berapapun nilainya, selama nilainya sesuai dengan *stop condition* maka *loop* akan dihentikan. Hal ini berbeda dengan *while loop* dimana proses dilakukan berulang-ulang sampai jumlahnya mendekati *stop condition*.

2.12.4 Break

Break sebenarnya bukan bagian dari *loop*, namun sering digunakan dalam *loop*. *Break* dapat digunakan pada *loop* manakala dirasa perlu, yaitu saat kondisi yang disyaratkan pada *break* tercapai.

Berikut adalah contoh penerapan *break* pada beberapa jenis *loop*.

```
# for loop
a = c(2,4,6,8,10,12,14)
for(i in a){
  if(i>8){
    break
  }
  print(i)
}
```

```
## [1] 2
## [1] 4
## [1] 6
## [1] 8
```

```
# while loop
a = 2
b = 4
while(a<7){
  print(a)
  a = a +1
  if(b+a>10){
    break
  }
}
```

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

```
# repeat loop
a = 1
repeat{
  print(a)
  a = a+1
  if(a>6){
    break
  }
}
```

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

2.13 Decision Making

Decicion Making atau sering disebut sebagai *if then else statement* merupakan bentuk percabagan yang digunakan manakala kita ingin agar program dapat melakukan pengujian terhadap syarat kondisi tertentu. Pada Table 5 disajikan daftar percabangan yang digunakan pada R.

Table 5 Daftar percabangan pada R

Statement	Keterangan
if statement	*if statement* hanya terdiri atas sebuah ekspresi *Boolean*, dan diikuti satu atau lebih *statement*
if…else statement	*if else statement* terdiri atas beberapa buah ekspresi *Boolean*. Ekspressi *Boolean* berikutnya akan dijalankan jika ekspresi *Boolan sebelumnya bernilai FALSE
switch statement	*switch statement* digunakan untuk mengevaluasi sebuah variabel beberapa pilihan

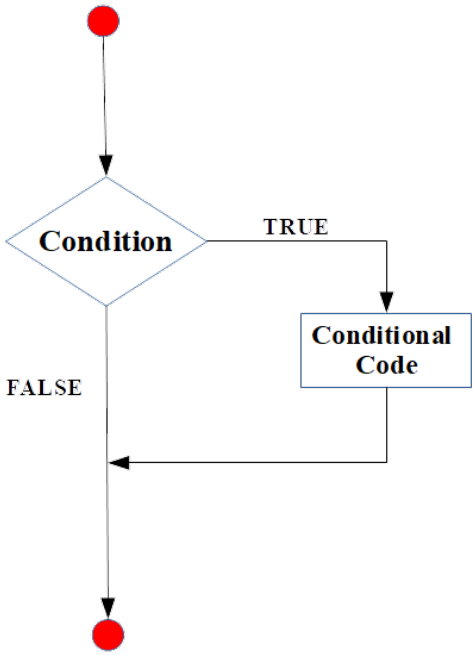

Figure 2.2: Diagram if statement (sumber: Primartha, 2018).

2.13.1 if statement

Pola *if statement* disajikan pada Figure 2.2

Berikut adalah contoh penerapan *if statement*:

```
x <- c(1:5)
if(is.vector(x)){
  print("x adalah sebuah vector")
}
```

```
## [1] "x adalah sebuah vector"
```

2.13.2 if else statement

Pola dari *if else statement* disajikan pada Figure 2.3

Berikut adalah contoh penerapan *if else statement*:

```
x <- c("Andi","Iwan", "Adi")
if("Rina" %in% x){
  print("Rina ditemukan")
} else if("Adi" %in% x){
  print("Adi ditemukan")
} else{
  print("tidak ada yang ditemukan")
}
```

```
## [1] "Adi ditemukan"
```

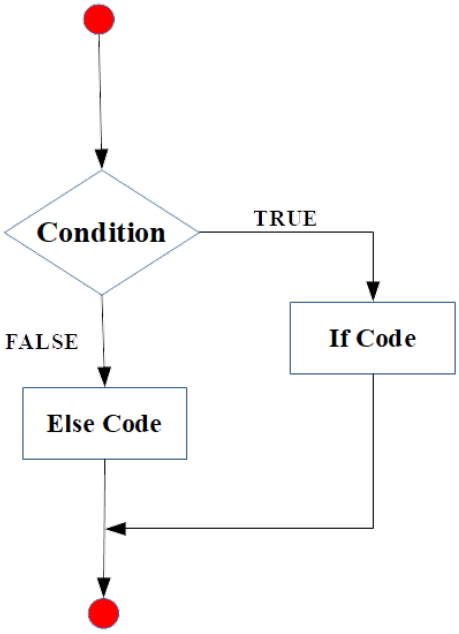

Figure 2.3: Diagram if else statement (sumber: Primartha, 2018).

2.13.3 switch statement

Pola dari *switch statement* disajikan pada Figure 2.4

Berikut adalah contoh penerapan *switch statement*:

```
y = 3

x = switch(
  y,
  "Selamat Pagi",
  "Selamat Siang",
  "Selamat Sore",
  "Selamat Malam"
)

print(x)
```

```
## [1] "Selamat Sore"
```

2.14 Fungsi

Fungsi merupakan sekumpulan instruksi atau *statement* yang dapat melakukan tugas khusus. Sebagai contoh fungsi perkalian untuk menyelesaikan operasi perkalian, fungsi pemangkatan hanya untuk operasi pemangkatan, dll.

Pada `R` terdapat 2 jenis fungsi, yaitu: *build in fuction* dan *user define function*. *build in fnction* merupakan fungsi bawaan `R` saat pertama kita menginstall `R`. Contohnya adalah `mean()`, `sum()`, `ls()`, `rm()`, dll. Sedangkan *user define fuction* merupakan fungsi-fungsi yang dibuat sendiri oleh pengguna.

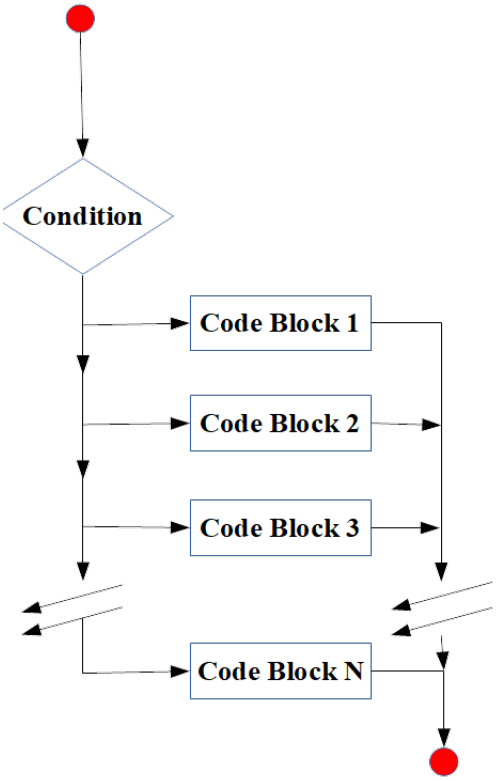

Figure 2.4: Diagram switch statement (sumber: Primartha, 2018).

Fungsi-fungsi buatan pengguna haruslah dideklarasikan (dibuat) terlebih dahulu sebelum dapat dijalankan. Pola pembentukan fungsi adalah sebagai berikut:

```
function_name <- function(argument_1, argument_2, ...){
  function body
}
```

Note:

- **function_name** : Nama dari fungsi `R`. `R` akan menyimpan fungsi tersebut sebagai objek
- **argument_1, argument_2,...** : *Argument* bersifat opsional (tidak wajib). *Argument* dapat digunakan untuk memberi inputan kepada fungsi
- **function body** : Merupakan inti dari fungsi. Fuction body dapat terdiri atas 0 statement (kosong) hingga banyak statement.
- **return** : Fungsi ada yang memiliki *output* atau *return value* ada juga yang tidak. Jika fungsi memiliki *return value* maka *return value* dapat diproses lebih lanjut

Berikut adalah contoh penerapan *user define function*:

```
# Fungsi tanpa argument
bilang <- function(){
  print("Hello World!!")
}

# Print
bilang()
```

```
## [1] "Hello World!!"
```

```
# Fungsi dengan argumen
tambah <- function(a,b){
  print(a+b)
}

# Print
tambah(5,3)
```

```
## [1] 8
```

```
# Fungsi dengan return value
kali <- function(a,b){
  return(a*b)
}

# Print
kali(4,3)
```

```
## [1] 12
```

2.15 Referensi

1. Primartha, R. 2018. **Belajar Machine Learning Teori dan Praktik**. Penerbit Informatika : Bandung.
2. Rosadi,D. 2016. **Analisis Statistika dengan R**. Gadjah Mada University Press: Yogyakarta.
3. STHDA. **Easy R Programming Basics**. http://www.sthda.com/english/wiki/easy-r-programming-basics
4. Venables, W.N. Smith D.M. and R Core Team. 2018. **An Introduction to R**. R Manuals.
5. The R Core Team. 2018. **R: A Language and Environment for Statistical Computing**. R Manuals.

Chapter 3

Methods

We describe our methods in this chapter.

Chapter 4

Applications

Some *significant* applications are demonstrated in this chapter.

4.1 Example one

4.2 Example two

Chapter 5

Final Words

We have finished a nice book.