



Intro to R - Volume 2.2

Edisi Revisi

ikanx101.github.io

Agustus 6, 2020

Contents

Untuk apa saya belajar R?	8
1 Pendahuluan	10
1.1 Sejarah	10
1.2 Fitur dan Karakteristik	10
1.3 Kelebihan dan Kekurangan R	11
1.4 R vs R Studio	11
1.5 Mengenal operator dasar	13
1.6 Working Directory	14
1.6.1 Bagaimana mengubah <i>working directory</i> ?	14
1.6.2 Apa keuntungan mengubah-ubah <i>working directory</i> ?	14
1.7 Mengenal <i>packages</i> atau <i>library</i>	14
1.7.1 Instalasi <i>Packages</i>	15
1.7.2 Mengaktifkan <i>Packages</i>	15
1.7.3 Serba-Serbi Tentang <i>Packages</i>	15
1.7.4 <i>Help</i>	16
1.7.5 <i>Example</i>	16
2 Mulai Bekerja dengan R	18
2.1 Mengenal Data	18
2.1.1 Tipe Data (statistika)	18
2.1.2 Tipe Data di R	18
2.1.3 Struktur Data di R	19
2.1.4 Apa gunanya kita mengetahui jenis dan struktur data di R ?	19
2.2 Tata Cara Memberikan Nama <i>Object</i> atau Variabel	19
3 Memulai dengan R	21
3.1 Fungsi Awal	21
3.1.1 Pendefinisian <i>object</i>	21
3.1.2 Operasi Aritmatika dan Matematika	21
3.1.3 Operasi Relasi dan <i>Logical (Boolean)</i>	22
3.1.4 <i>If Conditional</i>	23
3.2 Bekerja dengan data	24
3.2.1 <i>Single Value</i>	24
3.2.2 <i>Vector</i>	24
3.2.3 <i>Tibble</i> atau <i>Data Frame</i>	28

3.2.4	<i>Missing values NA</i>	28
3.3	Beberapa Fungsi yang Berguna	30
3.3.1	<i>Paste</i>	30
3.3.2	<i>Print</i>	30
3.3.3	<i>str</i>	31
3.3.4	<i>Summary</i>	31
3.3.5	<i>Class</i>	31
3.3.6	<i>View</i>	31
3.4	<i>Looping</i>	32
3.4.1	<i>For</i>	32
3.4.2	<i>While</i>	33
3.5	<i>Regex</i>	33
3.5.1	<i>Pattern Matching</i>	34
3.5.2	<i>Replace Pattern</i>	36
4	Membuat <i>function</i> di R.	38
4.1	<code>function()</code> tanpa <i>entry variable</i>	38
4.2	<code>function()</code> dengan <i>entry variable</i>	39
5	Membaca Data dari Excel	41
5.1	Membaca Data dari Format File Lain	41
6	Berkenalan dengan Tidyverse	43
6.1	<code>filter()</code>	43
6.2	<code>arrange()</code>	45
6.2.1	<i>Descending</i>	45
6.2.2	<i>Ascending</i>	45
6.3	<code>select()</code>	46
6.4	<code>mutate()</code>	47
6.5	<code>group_by()</code> dan <code>summarise()</code>	48
6.6	<code>separate()</code>	49
7	Beberapa Fungsi Statistik	52
7.1	<code>sum()</code>	52
7.2	<code>mean()</code>	52
7.3	<code>median()</code>	52
7.4	<code>sd()</code>	52

8 Menggabungkan Data	54
8.1 merge()	54
8.1.1 Cara Klasik	54
8.1.2 Menggunakan Tidyverse	55
8.2 rbind()	55
8.2.1 Cara Klasik	56
8.2.2 Menggunakan Tidyverse	56
9 Libraries yang Berguna	59
9.1 library(janitor)	59
9.1.1 Function make_clean_names()	59
9.1.2 Function clean_names()	60
9.2 library(lubridate)	61
9.3 library(ggplot2)	63
9.4 library(reshape2)	64
9.5 library(tidytext)	65
9.6 library(rvest)	65
10 APLIKASI R DI DUNIA REAL	67
10.1 Mencari Harga Mobil Bekas	67
Link dan Function yang Digunakan	67
Membereskan Harga	69
Membereskan Variabel nama dan Mengekstrak tahun	69
10.2 Membuat Alat Rapid Test COVID-19	72
Problem Statements	72
Analisa ROC	72
biomarker1	73
biomarker2	73
Kesimpulan dari nilai AUC	74
Cut Off Points	74
biomarker1	74
biomarker2	75
Confusion Matrix	75
biomarker1	75
biomarker2	76
Kesimpulan Confusion Matrix	76
KESIMPULAN Final	76
Referensi	78



Bayangkan Anda berada di restoran *all you can eat* dimana semua makanan yang Anda mau ada semua di sana... Ikang

Untuk apa saya belajar R?

Di era digitalisasi ini, disadari atau tidak data tersebar di mana-mana. Data juga dihasilkan dengan *volume* yang besar dalam waktu singkat.

Analoginya seperti ada sungai yang memiliki arus yang deras dan kencang. Seperti itulah kondisi saat ini.

Tools analisa data klasik seperti **Ms. Excel** dan **SPSS** sudah tidak mampu lagi melakukan analisa *big data* yang seringkali berbentuk *unstructured data*.

R tidak sendirian, ada juga *software* lain bernama **Python**. Keduanya digunakan untuk membuat algoritma *artificial intelligence* (bahasa keren dari *machine learning*. Bahasa kerennya dari *computational science*).

1 Pendahuluan

R merupakan salah satu bahasa pemrograman yang biasa digunakan untuk menyelesaikan permasalahan terkait dengan data. Kita bisa membuat model prediksi (*machine learning*, *artificial intelligence*, dan *deep learning*) sampai membuat algoritma automasi menggunakan **R**.

Apa perbedaan R dan Python? Salah satu kelebihan **R** adalah:

R is made by statistician for statistician. Setiap *package* atau *library* yang di-*launching* di **R** biasanya disertakan dengan jurnal ilmiah sehingga kita bisa dengan yakin memakainya.

R tersedia secara *open source* sehingga *software* ini gratis dan dikembangkan secara massal oleh komunitas-komunitas di seluruh dunia. Sehingga *package* atau *library* yang disediakan untuk analisis statistika dan analisa numerik juga sangat lengkap dan terus bertambah setiap saat.

Bagaimana dengan **Python**?

Sejatinya Python digunakan untuk membangun aplikasi. Namun, belakangan ini ternyata Python disadari bisa untuk melakukan pengolahan data. Berbeda dengan **R** yang memang dibangun untuk kebutuhan *data science*, **Python** membutuhkan *libraries* setiap kali melakukan pengolahan data.

Materi *training* ini saya kumpulkan dari berbagai sumber dan saya *customize* sesuai dengan kebutuhan **Nutrifood** berdasarkan pengalaman selama ini berkutat dengan data yang ada (dari mulai data pabrik hingga *finance*). Semoga menjadi manfaat bagi *Nutrifooders* semua.

1.1 Sejarah

R Merupakan bahasa yang digunakan dalam komputasi statistik yang pertama kali dikembangkan oleh Ross Ihaka dan Robert Gentleman 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.

1.2 Fitur dan Karakteristik

Sama halnya dengan bahasa pemrograman lainnya. Berbeda bahasa berarti berbeda peraturan / cara menulis *code* (algoritma). Tapi jangan khawatir, dengan memanfaatkan *tidy principle* di **R**, kita bisa menulis algoritma dengan mudah (bagi kita dan pembaca algoritamanya).

Oleh karena itu, menurut saya **R** menawarkan *learning curve* yang jauh lebih baik dibandingkan *Python*. Beberapa karakter dari **R** adalah sebagai berikut:

1. Bahasa **R** bersifat *case sensitive*. Setiap perbedaan cara penulisan (kapital vs non kapital) akan membedakan suatu objek. Contoh:

```
x = 'Nutrifood'
y = 'nutrifood'
x == y
```

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

2. Segala sesuatu yang ada pada program **R** akan dianggap sebagai objek. konsep objek ini sama dengan bahasa pemrograman berbasis objek yang lain seperti *Java*, *C++*, *Python*, dll. Perbedaanannya adalah bahasa **R** relatif lebih sederhana dibandingkan bahasa pemrograman berbasis objek yang lain.
3. *Interpreted language* atau *script*. Bahasa **R** memungkinkan pengguna untuk melakukan kerja pada **R** tanpa perlu melakukan *compile* menjadi *executable* file (.exe).
4. Mendukung proses *loop*, *decision making*, dan menyediakan berbagai jenis operator (aritmatika, logika, dll).
5. Mendukung *export* dan *import* berbagai *format file*, seperti: *.txt*, *.xlsx*, *.csv*, *.json*, *sql*, dll.
6. Mudah ditingkatkan melalui penambahan fungsi atau *library*. Penambahan ini dapat dilakukan secara *online* melalui **CRAN** atau melalui sumber seperti **github**.
7. Menyediakan berbagai fungsi untuk keperluan visualisasi data. Visualisasi data pada **R** dapat menggunakan *library* bawaan atau lainnya seperti **ggplot2**, **ggvis**, **plotly**, 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 **R** bisa dijalankan di OS manapun. Baik Windows, iOS, Linux, Raspbian, bahkan Android! Dengan fitur yang sama (tidak ada perbedaan fitur di semua OS).
3. *Programable*, pengguna dapat membuat fungsi dan metode baru atau mengembangkan modifikasi dari analisis statistika yang telah ada pada sistem **R**.
4. Fasilitas grafik yang lengkap.

Adapun kekurangan dari **R** antara lain:

- *Point and Click GUI*, interaksi utama dengan **R** bersifat **CLI** (*Command Line Interface*), walaupun saat ini telah dikembangkan *library* yang memungkinkan kita berinteraksi dengan **R** menggunakan **GUI** (*Graphical User Interface*) sederhana menggunakan **library(R-Commander)** yang memiliki fungsi yang terbatas.

1.4 **R** vs **R Studio**

Pada dasarnya, *software R* bisa di-*download* dan di-*install* langsung dari situs CRAN. *Software R* ini bersifat **CLI**.

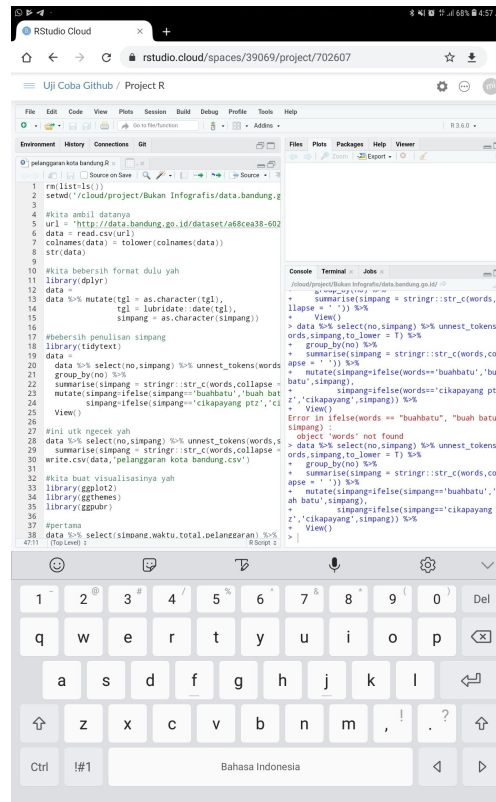
Bayangkan Anda membuka aplikasi **notepad**. Putih dan bersih kan? Seperti itulah *software R*.

Bagi Kamu yang kaget dan tidak terbiasa melihat tampilan yang *intimidating* seperti itu, Kamu bisa meng-*install software R Studio*. Sebuah *software* GUI yang bisa membuat **R** terlihat lebih *user friendly*. **R Studio** bisa di-*download* di sini.

Tapi tolong diperhatikan bahwa **R Studio** hanya tambahan tampilan dari **R** standar. Jadi Kamu tetap harus meng- *install R* yah!

Kelebihan R Studio antara lain:

1. *Free*, kita bisa memilih versi gratis dari **R Studio** tanpa ada pengurangan fitur dasar dari **R**.
2. *R Studio Cloud*, tersedia layanan *cloud* sehingga bisa diakses dan digunakan menggunakan *browser* di *gadget* manapun. Layanan *cloud* ini bisa diakses di sini dan dikoneksikan ke akun **github** Kamu. Kira-kira seperti ini tampilannya jika dibuka di *Chrome for Android*:



3. *Shiny Apps*, kita bisa membuat *apps* berbasis *web* dari **R**. *Apps* ini bisa dijadikan *dashboard* atau mesin kalkulasi otomatis. Tergantung seberapa jauh Kamu membuat *coding* algoritmanya.
4. *R Markdown*, ini fitur yang paling saya sukai. Bahkan untuk menulis *web* ini, saya menggunakan *R Markdown*. Output files -nya beragam, mulai dari *docx*, *pptx*, *pdf*, *html*, *md*, dll. Bahkan kita bisa membuat *e-book* dengan memanfaatkan `library(bookdown)`.

Jadi, setelah membaca bagian ini pastikan Kamu sudah meng- *install* **R** dan **R Studio** yah.

Jangan sampai terbalik urutan instalasinya!

1.5 Menenal operator dasar

Beberapa operator dasar di **R** antara lain:

1. `=` atau `<-`, digunakan untuk melakukan pendefinisian suatu objek. Contoh:

```
a = 10
b <- 3
a + b
```

```
## [1] 13
```

2. `' '` atau `" "`, digunakan untuk menandai tipe variabel berupa **character**. Lalu apa beda penggunaan `' '` dengan `" "`? `" "` digunakan saat `' '` dibutuhkan dalam suatu **character**. Contoh:

```
a = 'saya hendak pergi ke pasar'
b = "i don't want to buy it"
a
```

```
## [1] "saya hendak pergi ke pasar"
```

```
b
```

```
## [1] "i don't want to buy it"
```

3. `==`, `<`, `>`, `<=`, atau `>=`, digunakan untuk mengecek apakah dua variabel itu memiliki kesamaan atau tidak. *Output* dari operator ini adalah **logic** (*TRUE or FALSE*). Contoh:

```
a = 5
b = 3
a == b
```

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

```
a > b
```

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

4. `;` atau `,` digunakan untuk memisahkan baris kode pada skrip algoritma. Contoh:

```
a = 5;b = 3;a*b
```

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

1.6 Working Directory

Apa itu *working directory*?

Working directory adalah *folder path default* untuk **R** melakukan *import* dan *export* data. Untuk mengetahui di mana *working directory* kita, bisa digunakan perintah:

```
getwd()
```

```
## [1] "/home/ikanx101githubio/Documents/belajar/Materi Training/Day 1 - R Series"
```

Secara *default*, **R** menggunakan C:\\My Documents sebagai *working directory*.

1.6.1 Bagaimana mengubah *working directory*?

Working directory bisa diubah sesuai kemauan kita memanfaatkan perintah `setwd()`, tanda dalam kurung diisi dengan *folder path* yang diinginkan.

```
setwd("~/Documents/belajar/Materi Training/Day 1 - R Series")
```

1.6.2 Apa keuntungan mengubah-ubah *working directory*?

Perubahan *working directory* akan sangat berguna saat kita ingin mengambil data dari *folder path* tertentu dan menyimpan hasil analisa kita ke *folder path* yang berbeda.

1.7 Mengenal *packages* atau *library*

packages atau *library* adalah sekumpulan fungsi yang telah dibuat dan dibakukan untuk kemudian disertakan di halaman *web* CRAN atau github. *library* bisa kita *install* dan gunakan dengan mudah.

Seperti yang sudah saya infokan di bagian pendahuluan. Banyak orang atau komunitas yang mengembangkan berbagai macam *library* sehingga memudahkan kita untuk menyelesaikan masalah di data kita. Kita tidak perlu lagi membuat algoritma dari nol. Cukup memanfaatkan *library* yang tepat saja.

Beberapa contoh *library* yang sering saya gunakan:

1. **dplyr**: *data carpentry* menggunakan *tidy principle*.
2. **ggplot2**: *data visualization*.
3. **rvest**: *web scraping*.
4. **tidytext**: *text analysis*.
5. **reshape2**: *data manipulation*.
6. **readxl** atau **openxlsx**: *export dan import excel files*.
7. **officer**: membuat *Ms. Office files* seperti *excel*, *docx*, dan *powerpoint*.
8. **expss**: **SPSS** di **R**.
9. **xaringan**: membuat *file* presentasi berformat *html*.

1.7.1 Instalasi *Packages*

`library` di **R** bisa di-*install* dengan mudah dengan menggunakan perintah `install.packages('nama packages')`. Tanda dalam kurung diisi `character` nama `library`. Bisa menggunakan `" "` atau `' '`.

Proses instalasi `library` ini membutuhkan koneksi internet karena **R** akan otomatis terhubung ke dalam situs *web CRAN*. Setelah proses instalasi selesai, maka koneksi internet tidak diperlukan lagi (kecuali untuk melakukan *web scraping*).

Contoh:

```
install.packages('readxl')  
install.packages("rvest")
```

1.7.2 Mengaktifkan *Packages*

`library` yang sudah di-*install* bisa diaktifkan dengan menggunakan perintah `library(nama packages)` tanpa menggunakan tanda `" "` atau `' '`.

Pengaktifan `library` cukup dilakukan sekali saja di awal pengerjaan *project* (tidak perlu dilakukan berulang kali). Contoh:

```
library(dplyr)
```

Beberapa `library` saat diaktifkan akan menghasilkan pesan tertentu seperti di atas ini. Hal ini merupakan sesuatu yang **normal** terjadi.

1.7.3 Serba-Serbi Tentang *Packages*

Untuk beberapa `library` ada kemungkinan (kecil) ditemukan kasus saat mereka tidak kompatibel. Akibatnya beberapa fungsi perintah di `library` tersebut akan menjadi kacau.

Misalnya pada saat kita memanggil `library(tidyverse)` dan `library(plyr)`, maka perintah `filter()` yang dimiliki `tidyverse` akan tidak berjalan dengan baik.

Ada beberapa solusi yang bisa kita lakukan:

1. Selalu mengaktifkan `library` sesuai dengan urutannya. Biasanya setiap kali kita mengaktifkan `library` akan muncul *warnings* mengenai kompatibilitas `library` tersebut dengan `library` lain.
2. Menonaktifkan `library` yang sudah tidak perlu digunakan dengan perintah:

```
detach("package:tidytext", unload = TRUE)
```

3. Memanggil `library` tanpa harus mengaktifkannya. Kita bisa melakukannya dengan menggunakan tanda `nama packages::`. Contoh:

```
reshape2::melt(data)
```

1.7.4 Help

Setiap `library` yang telah di-*install* dan aktif disertai dengan fitur *help* yang berfungsi sebagai informasi kepada *user*. Jika kita ingin mengetahui bagaimana isi dari perintah suatu fungsi, kita bisa gunakan perintah `help(nama fungsi)` atau `?nama fungsi`. *Help* akan muncul pada tab *help* di **R Studio**. Contoh:

```
help(sum)
```

atau

```
?sum
```

1.7.5 Example

Selain *help*, kita bisa melihat contoh pemakaian dari suatu fungsi di **R** dengan menggunakan perintah `example()`. Contoh:

```
example(sum)
```

```
##
## sum> ## Pass a vector to sum, and it will add the elements together.
## sum> sum(1:5)
## [1] 15
##
## sum> ## Pass several numbers to sum, and it also adds the elements.
## sum> sum(1, 2, 3, 4, 5)
## [1] 15
##
## sum> ## In fact, you can pass vectors into several arguments, and everything gets added.
## sum> sum(1:2, 3:5)
## [1] 15
##
## sum> ## If there are missing values, the sum is unknown, i.e., also missing, ....
## sum> sum(1:5, NA)
## [1] NA
##
## sum> ## ... unless we exclude missing values explicitly:
## sum> sum(1:5, NA, na.rm = TRUE)
## [1] 15
```

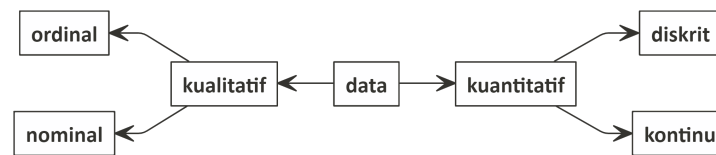

2 Mulai Bekerja dengan R

2.1 Mengenal Data

Sebelum memulai bekerja dengan **R**, ada baiknya saya jelaskan dan ingatkan kembali beberapa hal penting terkait data. Dengan demikian, kita bisa memilih jenis analisa statistika apa yang tepat untuk tipe-tipe data yang berbeda.

2.1.1 Tipe Data (statistika)

Secara statistika, berikut adalah pembagian data berdasarkan tipenya:



1. Data kualitatif: adalah data yang tidak bisa dilakukan operasi aritmatika (penjumlahan, pengurangan, pembagian, dan perkalian). Data seperti ini, kita akan sebut sebagai **data kategorik**.
 - **Nominal**; Representasi dari sesuatu. Contoh: **gender**, 1 saya tulis sebagai **pria** dan 2 saya tulis sebagai **wanita**.
 - **Ordinal**; Urutan dari data menjadi penting. Contoh: skala *likert* 1 - 6.
2. Data kuantitatif: adalah data yang bisa dilakukan operasi aritmatika (penjumlahan, pengurangan, pembagian, dan perkalian). Data seperti ini, kita akan sebut sebagai **data numerik**.
 - **Diskrit**; bilangan bulat (*integer*).
 - **Kontinu**; bilangan *real* (mengandung koma).

2.1.2 Tipe Data di R

Di **R** ada beberapa tipe data yang sering digunakan. Secara hierarki, bisa diurutkan sebagai berikut:

```
character > numeric > integer > logical
```

Oke, saya coba jelaskan satu persatu yah:

1. **character**: merupakan tipe data berupa karakter atau **string**. Semua data bisa dilihat sebagai **character**. Oleh karena itu, secara hierarki tipe data ini ditempatkan di urutan paling atas. Namun, data tipe ini tidak bisa dilakukan operasi aritmatika *yah*.
2. **numeric**: merupakan tipe data angka berupa bilangan *real*. Kalau saya boleh bilang, tipe data ini mirip dengan data numerik di poin **2.1.1**.
3. **integer**: merupakan tipe data angka berupa bilangan bulat. Sekilas mirip dengan tipe data diskrit di poin **2.1.1**. Namun di beberapa kondisi, tipe data ini bisa dijadikan data **kategorik** sehingga kita bisa sebut tipenya menjadi **factor**.
4. **logical**: merupakan tipe data *boolean*. Hanya berisi **TRUE** atau **FALSE**. Tipe data ini sangat berguna saat kita melakukan *if conditional*, *looping*, atau membuat *regex (regular expression)*.

2.1.3 Struktur Data di R

Ada beberapa bentuk struktur data di **R**, yakni:

1. *Single value*; satu objek yang berisi satu *value* saja.
2. *Vector*; kumpulan dari beberapa *single value(s)* yang menjadi satu objek. Bayangkan sebagai satu buah kolom di *file Ms. Excel*.
3. *Data frame* atau *tibble*; merupakan kumpulan dari beberapa *vectors* yang memiliki ukuran sama. Bayangkan sebagai satu tabel di *Ms. Excel* yang banyaknya baris di setiap kolom sama.
4. *List*; merupakan bentuk struktur data yang sangat kompleks. Berisi *multiple data* dengan struktur bermacam-macam.

2.1.4 Apa gunanya kita mengetahui jenis dan struktur data di R?

Beberapa algoritma yang tersedia di *library* mengharuskan kita memiliki *input* yang ter-standar, baik dari segi jenis dan strukturnya.

Dengan mengetahui jenis dan struktur data, kita bisa lebih mudah bekerja dengan algoritma yang ada di *library*.

Contoh:

Algoritma analisa *simple linear regression* (`lm()`) memerlukan input berupa `data.frame()` dengan masing-masing *variables* yang ada di dalamnya berjenis *numeric*.

2.2 Tata Cara Memberikan Nama *Object* atau Variabel

Setiap *object* atau variabel di **R** bisa diberikan nama sesuai dengan keinginan kita. Tidak ada aturan baku dalam memberikan nama.

Tapi, dengan memberikan nama yang **tepat** kita bisa bekerja dengan lebih cepat dan efisien. Berikut adalah tata cara pemberian nama yang akan membuat pekerjaan kita lebih efisien:

1. Seragamkan kapital atau non kapital dari nama variabel kita. Jika menggunakan *lowercase*, maka harus konsisten di setiap data yang ada di *environment R*.
 2. Hindari penggunaan spasi " ". Jika memang tidak bisa dihindari, gunakan tanda "." atau "_".
- Contoh: variabel `tinggi badan` akan lebih baik ditulis dalam bentuk `tinggi.badan` atau `tinggi_badan`. Jika sudah terlanjur memiliki nama variabel yang tidak seragam atau mengandung spasi (biasanya terjadi saat kita meng- *import* data dari sumber lain seperti: *excel*), kita bisa merapkannya dengan otomatis dengan memanfaatkan `library(janitor)` fungsi `make_clean_names()` atau `clean_names()`.

3 Memulai dengan R

Oke, kita akan memulai bekerja dengan **R**, dimulai dari fungsi-fungsi awal sebagai berikut:

3.1 Fungsi Awal

3.1.1 Pendefinisian *object*

Setiap data yang akan kita masukkan ke dalam memori **R**, akan saya sebut sebagai *object*. Setiap *object* yang ingin dimasukkan ke dalam memori perlu didefinisikan terlebih dahulu menggunakan perintah `=` atau `<-`.

Contoh:

Jika saya ingin mendefinisikan dua buah *objects*, yakni **a** dan **b** sebagai berikut:

```
a = 6
b <- 8
```

Maka:

```
a
```

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

```
b
```

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

3.1.2 Operasi Aritmatika dan Matematika

Setiap *object* yang sudah masuk ke dalam memori **R** sudah bisa dilakukan analisa atau dilakukan operasi aritmatika: `+`, `-`, `/`, dan `*` ATAU diberikan fungsi matematika seperti **trigonometri**, **logaritmik**, dan lain-lain.

Pada *section* 3.1.1 kita telah mendefinisikan *objects* **a** dan **b**, maka kita bisa lakukan perintah sebagai berikut:

```
a + b
```

```
## [1] 14
```

```
a / b
```

```
## [1] 0.75
```

```
c = a * b
c
```

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

```
sin(c)
```

```
## [1] -0.7682547
```

```
log(a+b/c)
```

```
## [1] 1.819158
```

3.1.3 Operasi Relasi dan *Logical (Boolean)*

Pada *section* 1.5 poin 3, kita telah mengetahui operator relasi seperti ==, >, <, <=, >=, dan !=. Hasil dari operator relasi ini adalah *logical value* (**TRUE** atau **FALSE**).

Logical value yang dihasilkan memiliki sifat sebagai berikut:

1. **TRUE**, berarti **benar**. Bisa disingkat menjadi T. Tidak bisa ditulis dalam huruf kecil (harus kapital).
2. **FALSE**, berarti **salah**. Bisa disingkat menjadi F. Tidak bisa ditulis dalam huruf kecil (harus kapital).

Operator logical yang biasa digunakan di **R** antara lain:

1. & menandakan **AND**
2. | menandakan **OR**
3. ! menandakan **NOT**

Contoh:

Misalkan saya memiliki dua pernyataan sebagai berikut:

```
pernyataan_1 = T  
pernyataan_2 = F
```

Maka:

```
!pernyataan_1
```

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

```
pernyataan_1 & pernyataan_2
```

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

```
pernyataan_1 | pernyataan_2
```

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

3.1.4 If Conditional

Mungkin teman-teman bertanya-tanya:

Apa sih gunanya *logical value* dan *logical operator*? *Logical value* merupakan unsur utama saat kita hendak membuat fungsi *conditional* dan *looping*. Masih ingat fungsi di **Ms. Excel** untuk membuat *conditional*? Nah, mirip pengerjaannya di **R**.

Setidaknya ada tiga fungsi *conditional* di **R**, yakni:

1. `ifelse()`: bawaan dari *package base*.
2. `if_else()`: fungsi dari *package dplyr* (perlu di- *install* dulu *package*-nya).
3. `case_when()`: fungsi dari *package dplyr* (perlu di- *install* dulu *package*-nya).

Apa perbedaan ketiganya?

`ifelse()` dengan `if_else()` berdasarkan pengalaman saya tidak ada perbedaannya. Selama ini saya cukup memilih salah satu saja.

Sedangkan `case_when()` digunakan bersamaan dengan *pipe %>%* pada saat *tidying data*. Berguna saat kita hendak mem- *vector*-kan *conditional*.

Masih bingung? Saya akan bahas fungsi `ifelse()` dulu *yah*. Pembahasan mengenai `case_when()` akan saya jelaskan pada *section* khusus mengenai *tidyverse*.

Contoh paling mudah untuk *conditional* seperti ini:

Misalkan saya memiliki dua buah *objects*, yakni `a` dan `b`.

```
a = 10
b = 10 + sin(pi/3)
```

Maka:

```
ifelse(a < b, 'hari ini cerah','hari ini mendung')
```

```
## [1] "hari ini cerah"
```

```
ifelse(a == b, 'sama-sama','tidak bersama')
```

```
## [1] "tidak bersama"
```

3.2 Bekerja dengan data

Pada *section 2.1.2* telah dijelaskan beberapa struktur data di **R**, sekarang kita akan melihat bagaimana bentuk *real*-nya di **R**.

Untuk data berbentuk `list`, akan saya jelaskan sekalian bersamaan dengan materi `tidyverse` *yah*.

3.2.1 Single Value

Contoh:

```
a = 100
x = 50
z = 'Indonesia'
```

3.2.2 Vector

Vector didefinisikan dengan menggunakan perintah `c()`; merupakan

Contoh:

```
tinggi_badan = c(164,149,180,184,153,90,139,199,186,158,197)
tinggi_badan
```

```
## [1] 164 149 180 184 153 90 139 199 186 158 197
```

3.2.2.1 Elemen Vector Ada yang sadar *gak* dengan tanda `[1]` setiap kali kita *running* suatu skrip di **R**.

Apa *sih* artinya?

Itu adalah tanda posisi pertama dari *vector*. Tanda `[]` digunakan untuk memanggil isi *vector* di posisi tertentu. Istilah kerennya adalah *subset* dari suatu *vector*.

Contoh:

```
tinggi_badan[1]
```

```
## [1] 164
```

```
tinggi_badan[7]
```

```
## [1] 139
```

```
tinggi_badan[10]
```

```
## [1] 158
```

```
tinggi_badan[3:5]
```

```
## [1] 180 184 153
```



```
tinggi_badan[c(1,7,10)]
```

```
## [1] 164 139 158
```

```
tinggi_badan[-c(1,7,10)] #pengeluaran
```

```
## [1] 149 180 184 153 90 199 186 197
```

3.2.2.2 Operasi Aritmatika Pada *Vector* *Vector* yang berupa numerik bisa dilakukan operasi aritmatik.

Contoh:

```
status = (tinggi_badan - 100)/50  
status
```

```
## [1] 1.28 0.98 1.60 1.68 1.06 -0.20 0.78 1.98 1.72 1.16 1.94
```

3.2.2.3 Fungsi Pada *Vector* *Vector* berupa numerik juga bisa dikenakan fungsi perhitungan seperti:

```
max(tinggi_badan) # memperoleh nilai maksimum x
```

```
## [1] 199
```

```
min(tinggi_badan) # memperoleh nilai minimum x
```

```
## [1] 90
```

```
range(tinggi_badan) # memperoleh range vektor x
```

```
## [1] 90 199
```

```
length(tinggi_badan) # memperoleh jumlah vektor x
```

```
## [1] 11
```

```
sum(tinggi_badan) # memperoleh total penjumlahan vektor x
```

```
## [1] 1799
```

```
mean(tinggi_badan) # memperoleh nilai mean vektor x
```

```
## [1] 163.5455
```

```
sd(tinggi_badan) # standar deviasi vektor x
```

```
## [1] 31.5194
```

```
var(tinggi_badan) # varian vektor x
```

```
## [1] 993.4727
```

```
sort(tinggi_badan) # mengurutkan elemen vektor x dari yang terbesar
```

```
## [1] 90 139 149 153 158 164 180 184 186 197 199
```

3.2.2.4 Generating Sequences *Sequences* atau deret bisa kita bangun menggunakan **R** dengan dua cara:

1. Menggunakan `:`.
2. Menggunakan fungsi `seq()`.

Contoh:

```
nomor_1 = c(1:10)  
nomor_1
```

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

```
# perintah untuk menghitung gcumulative sum  
cumsum(nomor_1)
```

```
## [1] 1 3 6 10 15 21 28 36 45 55
```

```
nomor_2 = seq(1,10,0.4) # generating sequence dari 1 hingga 10 dengan jeda 0.4  
nomor_2
```

```
## [1] 1.0 1.4 1.8 2.2 2.6 3.0 3.4 3.8 4.2 4.6 5.0 5.4 5.8 6.2 6.6 7.0 7.4 7.8 8.2  
## [20] 8.6 9.0 9.4 9.8
```

Apa sih gunanya deret? Percayalah, suatu saat nanti deret bisa digunakan untuk membantu perhitungan atau analisa kita. Seolah-olah berperan sebagai **katalis** pada reaksi kimia.

3.2.2.5 Random sampling dari suatu vector Ada suatu masa saat kita membutuhkan untuk mengambil sampel (mengambil subset) dari suatu *vector* secara acak. Kita bisa menggunakan fungsi `sample()`.

Contoh:

Kita memiliki data variabel `nama_orang` berisi 20 nama orang. Kita hanya ingin memilih 3 nama orang saja secara acak. Bagaimana caranya?

```
nama_orang = randomNames::randomNames(20)
nama_orang
```

```
## [1] "Wilson, Ayreonah"      "el-Yasin, Salwa"      "Olson, Laura"
## [4] "Wilson, Anthony"      "al-Latif, Fuaad"      "Beston, Tyler"
## [7] "Martin, Latasha"      "Mcgee, Symone"        "Ramirez Benzor, Victor"
## [10] "el-Khatib, Najwa"     "Mangisel, Sepehr"     "Chavez, Steven"
## [13] "Evers, Jennifer"      "Vogltanz, Jowan"      "Fernandez, Robert"
## [16] "el-Mir, Ramzi"        "Cisneros, Tyler"      "Medhin, Jasirii"
## [19] "Dinh, Abbygale"       "Tseng, Carly"
```

```
sample(nama_orang,3,replace = F)
```

```
## [1] "Medhin, Jasirii" "al-Latif, Fuaad" "Tseng, Carly"
```

`replace = F` digunakan saat kita tidak ingin ada pemilihan yang berulang. Sedangkan `replace = T` digunakan saat diperbolehkan hasil pemilihan berulang. Coba *run* sendiri *yah*.

Perintah `sample()` ini akan sangat berguna saat kita hendak menggunakan prinsip simulasi **Monte Carlo**.

3.2.2.6 Repeat Adakalanya kita hendak melakukan pengulangan yang simpel. *Instead of using looping*, kita bisa menggunakan perintah `rep()`. Misalkan:

```
rep('belajar R',3)
```

```
## [1] "belajar R" "belajar R" "belajar R"
```

```
rep(c(4:8),10)
```

```
## [1] 4 5 6 7 8 4 5 6 7 8 4 5 6 7 8 4 5 6 7 8 4 5 6 7 8 4 5 6 7 8 4 5 6
## [39] 7 8 4 5 6 7 8 4 5 6 7 8
```

```
tinggi_badan = c(120,132,142,90)
rep(mean(tinggi_badan),4)
```

```
## [1] 121 121 121 121
```

Jadi perintah `rep()` tidak hanya bisa untuk mengulang suatu single variabel atau *vector* saja tapi bisa digunakan untuk mengulang suatu fungsi.

Apa perbedaan dengan fungsi `repeat()`? Fungsi `repeat()` biasanya digunakan dalam *looping* dan baru akan berhenti saat diberikan perintah `break`.

3.2.3 *Tibble* atau *Data Frame*

Tibble atau *data frame* adalah struktur data di **R** berupa tabel. Analogi sederhananya adalah mirip dengan tabel di **Ms. Excel files**.

Data frame bisa dibentuk dari beberapa *vector* yang memiliki `length()` yang sama. Contohnya berikut ini: Kita akan membuat *data frame* dari 4 buah *vector*, yakni: `id`, `nama`, dan `tinggi_badan`.

```
id = c(1:10)
nama = randomNames::randomNames(10,gender = 0,which.names = 'first')
tinggi_badan = sample(c(150:199),10,replace = F)
absensi = data.frame(id,nama,tinggi_badan)
```

Hasilnya seperti ini:

```
absensi
```

```
##      id      nama tinggi_badan
## 1     1     Shawn          176
## 2     2     Keith          182
## 3     3     Louie          175
## 4     4     Sergio          178
## 5     5 Aucktaveon          161
## 6     6     Lance          162
## 7     7  Jeremiah          163
## 8     8    Deontey          151
## 9     9  Nicholas          177
## 10    10   Matthew          159
```

Bentuk *data frame* kelak akan menjadi primadona dalam setiap analisa yang digunakan di **R**. Nanti saat kita belajar *data carpentry* menggunakan *tidyverse*, struktur *data frame* mudah dimanipulasi dengan *piping operator*: `%>%`.

3.2.4 *Missing values* NA

Missing values adalah suatu nilai yang kosong pada suatu data. Kosong berarti tidak berisi data apapun. Bedakan dengan nilai 0 yah!

NA tidak akan diikutsertakan dalam perhitungan sedangkan 0 diikutsertakan. Di **R**, nilai NA pada data numerik akan membuat *error* setiap kali dihitung.

Contoh:

```
data_1 = c(3,5,0,6,8,3)
mean(data_1)
```

```
## [1] 4.166667
```

Berikut adalah contoh saat ada data berisi NA, maka data tersebut tidak akan bisa dihitung:

```
data_2 = c(3,5,NA,6,8,3)
mean(data_2)
```

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

Bagaimana cara mengecek keberadaan NA di data kita?

Kita bisa menggunakan fungsi `is.na()`. *Output* dari fungsi ini adalah *boolean variable* berupa TRUE atau FALSE.

Contoh: mengecek apakah ada NA di `data_2`.

```
is.na(data_2)
```

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

Contoh: membuat tabulasi dari *function* `is.na()`.

```
table(is.na(data_2))
```

```
##
## FALSE  TRUE
##      5      1
```

Contoh: menghitung seberapa banyak yang TRUE.

```
sum(is.na(data_2))
```

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

Contoh: mengecek apakah ada data yang **TIDAK** NA di `data_2`.

```
!is.na(data_2)
```

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

3.3 Beberapa Fungsi yang Berguna

3.3.1 *Paste*

Apakah kalian mengetahui fungsi bernama **CONCATENATE** di **Ms. Excel**? Fungsi `paste()` mirip penggunaannya dengan perintah **CONCATENATE**, yakni menggabungkan beberapa data menjadi satu.

Contoh: membuat *vector* berisi `nama_toko` yang berisi urutan nama toko.

```
nama_toko = paste('toko ke',c(1:10),sep='-')
nama_toko
```

```
## [1] "toko ke-1" "toko ke-2" "toko ke-3" "toko ke-4" "toko ke-5"
## [6] "toko ke-6" "toko ke-7" "toko ke-8" "toko ke-9" "toko ke-10"
```

Contoh: membuat *vector* dari `nama_toko` yang sudah pernah dibuat sebelumnya dengan *rules* 5 toko pertama dari Bandung dan 5 toko selanjutnya dari Bekasi.

```
nama_toko = paste(nama_toko,rep(c('Bandung','Bekasi'),5))
nama_toko
```

```
## [1] "toko ke-1 Bandung" "toko ke-2 Bekasi" "toko ke-3 Bandung"
## [4] "toko ke-4 Bekasi" "toko ke-5 Bandung" "toko ke-6 Bekasi"
## [7] "toko ke-7 Bandung" "toko ke-8 Bekasi" "toko ke-9 Bandung"
## [10] "toko ke-10 Bekasi"
```

`sep` = berguna untuk mendefinisikan *separator* apa yang hendak digunakan. Secara *default*, *separator* yang digunakan adalah spasi.

Coba kalian ganti sendiri bagian *separator*-nya.

3.3.2 *Print*

Fungsi `print()` digunakan untuk menampilkan data ke layar. Biasanya digunakan pada proses *looping* agar hasil iterasi dapat tampil ke layar.

```
print(nama_toko)
```

```
## [1] "toko ke-1 Bandung" "toko ke-2 Bekasi" "toko ke-3 Bandung"
## [4] "toko ke-4 Bekasi" "toko ke-5 Bandung" "toko ke-6 Bekasi"
## [7] "toko ke-7 Bandung" "toko ke-8 Bekasi" "toko ke-9 Bandung"
## [10] "toko ke-10 Bekasi"
```

3.3.3 *str*

Fungsi `str()` digunakan untuk melihat tipe dan struktur *object* yang ada di **R**. Sebagai contoh, kita akan pakai data `absensi` dari *section 3.2.3*.

```
str(absensi)
```

```
## 'data.frame': 10 obs. of 3 variables:
## $ id : int 1 2 3 4 5 6 7 8 9 10
## $ nama : chr "Shawn" "Keith" "Louie" "Sergio" ...
## $ tinggi_badan: int 176 182 175 178 161 162 163 151 177 159
```

Terlihat bahwa data `absensi` memiliki struktur **data.frame** dengan ada 3 *variables* dan 10 *observations* (baris data).

3.3.4 *Summary*

Fungsi `summary()` digunakan untuk melihat statistik deskriptif dari suatu data (tergantung dari tipe datanya). Contoh:

```
summary(absensi$tinggi_badan)
```

```
##      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    151.0   161.2   169.0   168.4   176.8   182.0
```

3.3.5 *Class*

Fungsi `class()` digunakan untuk melihat tipe atau struktur dari suatu data. Mirip dengan fungsi `str()`, tapi tidak sampai menampilkan dengan detail. Contoh:

```
class(absensi)
```

```
## [1] "data.frame"
```

```
class(absensi$tinggi_badan)
```

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

3.3.6 *View*

Fungsi `View()` (dengan huruf **V** kapital) digunakan untuk menampilkan **dataset** dalam *pop-up windows*.

3.4 *Looping*

Looping berarti pengulangan namun berbeda dengan *repeat* yang pernah kita bahas sebelumnya. Ada dua fungsi *looping* yang biasa digunakan, yakni: `for()` dan `while()`. Keduanya memiliki manfaat yang berbeda.

3.4.1 *For*

Fungsi `for()` biasa dilakukan untuk melakukan *looping* dengan syarat iterasi yang didefinisikan terlebih dahulu. Jadi kita sudah mengetahui terlebih dahulu berapa kali kita akan melakukan *looping*.

Contoh:

```
for(i in 1:10){  
  print(paste('pertanyaan ke',i))  
}
```

```
## [1] "pertanyaan ke 1"  
## [1] "pertanyaan ke 2"  
## [1] "pertanyaan ke 3"  
## [1] "pertanyaan ke 4"  
## [1] "pertanyaan ke 5"  
## [1] "pertanyaan ke 6"  
## [1] "pertanyaan ke 7"  
## [1] "pertanyaan ke 8"  
## [1] "pertanyaan ke 9"  
## [1] "pertanyaan ke 10"
```

Contoh:

```
for(i in 1:10){  
  i = 1/i  
  print(round(i,3))  
}
```

```
## [1] 1  
## [1] 0.5  
## [1] 0.333  
## [1] 0.25  
## [1] 0.2  
## [1] 0.167  
## [1] 0.143  
## [1] 0.125  
## [1] 0.111  
## [1] 0.1
```


3.4.2 While

Fungsi `while()` digunakan untuk melakukan *looping* dengan sampai syarat iterasi terpenuhi. Jadi kita belum mengetahui berapa kali kita akan melakukan *looping*.

Contoh: misalkan dalam satu ruangan ada 100 orang. Saya akan membagi mereka menjadi kelompok - kelompok berisi 1 sampai 5 orang. Kira - kira ada berapa banyak kelompok yang bisa saya dapatkan?

Caranya, kita set dulu kondisi awalnya.

```
orang = 100
i = 0 # berapa banyak kelompok? awalnya nol dulu
```

Lalu kita buat iterasi dengan `while()`. Yakni mengurangi secara berkala 100 orang dengan kelompok berisi 1 - 5 orang lalu menghitung ada berapa banyak iterasi yang terjadi.

```
while(orang>0){
  n = sample(c(1:5),1)
  orang = orang - n
  i = i+1
}
```

Berapa banyak iterasi (kelompok) yang mungkin muncul:

```
i
```

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

3.5 Regex

Regex adalah kepanjangan dari *regular expression*, yakni mencari *pattern* dari data berupa *string*. *Cheatsheet* untuk *regex* bisa dilihat di sini.

Selain menggunakan `base` dari **R**, kita juga bisa menggunakan `library(stringr)`.

Setidaknya ada dua manfaat utama dari *regular expression*, yakni:

1. *Pattern Matching*; mencari kecocokan *pattern* dari suatu data bertipe **character**.
2. *Replace Pattern*; mencari kecocokan *pattern* dan mengubahnya dari suatu data bertipe **character**.

Perbedaan cara penulisan (kapital atau *lower*) bisa kita pertimbangkan untuk dijadikan syarat pencarian atau tidak, yakni dengan penambahan `ignore.case = T` atau `ignore.case = F`.

Berikut ini adalah *syntaxes* yang ada dan digunakan untuk mencari apa:

Character Classes	
<code>[:digit:]</code> or <code>\d</code>	Digits; [0-9]
<code>\D</code>	Non-digits; [^0-9]
<code>[:lower:]</code>	Lower-case letters; [a-z]
<code>[:upper:]</code>	Upper-case letters; [A-Z]
<code>[:alpha:]</code>	Alphabetic characters; [A-z]
<code>[:alnum:]</code>	Alphanumeric characters [A-z0-9]
<code>\w</code>	Word characters; [A-z0-9_]
<code>\W</code>	Non-word characters
<code>[:xdigit:]</code> or <code>\x</code>	Hexadec. digits; [0-9A-Fa-f]
<code>[:blank:]</code>	Space and tab
<code>[:space:]</code> or <code>\s</code>	Space, tab, vertical tab, newline, form feed, carriage return
<code>\S</code>	Not space; [^[:space:]]
<code>[:punct:]</code>	Punctuation characters; !"#\$%&'()*+,-./:;<=>?@[]^_`{ }~
<code>[:graph:]</code>	Graphical char.; [[:alnum:]][[:punct:]]
<code>[:print:]</code>	Printable characters; [[:alnum:]][[:punct:]]\s
<code>[:cntrl:]</code> or <code>\c</code>	Control characters; \n, \r etc.

Character Classes and Groups	Anchors	Quantifiers
<code>.</code> Any character except \n	<code>^</code> Start of the string	<code>*</code> Matches at least 0 times
<code> </code> Or, e.g. (a b)	<code>\$</code> End of the string	<code>+</code> Matches at least 1 time
<code>[...]</code> List permitted characters, e.g. [abc]	<code>\b</code> Empty string at either edge of a word	<code>?</code> Matches at most 1 time; optional string
<code>[a-z]</code> Specify character ranges	<code>\B</code> NOT the edge of a word	<code>{n}</code> Matches exactly n times
<code>[^...]</code> List excluded characters	<code>\<</code> Beginning of a word	<code>{n,}</code> Matches at least n times
<code>(...)</code> Grouping, enables back referencing using \N where N is an integer	<code>\></code> End of a word	<code>{n}</code> Matches at most n times
		<code>{n,m}</code> Matches between n and m times

3.5.1 Pattern Matching

Sebagai contoh, saya akan gunakan data berikut ini:

Variabel *string* yang diketahui:

```
string = c('Market Research', 'market riset', 'survey', 'responden', 'mickey mouse')
```

Berikut *pattern* yang diinginkan:

```
pattern = 'm..ke'
```

Berikut adalah beberapa fungsi yang sering digunakan.

- *Function* `grep()`

Perhatikan *output* dari masing-masing perintah sebagai berikut:

```
grep(pattern,string)
```

```
## [1] 2 5
```

Output function ini adalah nomor urut / elemen dari *vector* yang sesuai dengan *pattern* yang diinginkan.

```
grep(pattern,string,value = T)
```

```
## [1] "market riset" "mickey mouse"
```

Output function ini adalah isi elemen dari *vector* yang sesuai dengan *pattern* yang diinginkan.

```
grep(pattern,string,ignore.case = T)
```

```
## [1] 1 2 5
```

Output function ini adalah isi elemen dari *vector* yang sesuai dengan *pattern* yang diinginkan dengan menghiraukan *uppercase* atau *lowercase*.

- *Function* `grepl()`

Output dari fungsi ini berupa *logic (boolean)*:

```
grepl(pattern,string,ignore.case = T)
```

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

- *Function* menggunakan `stringr::` dan fungsi `str_detect()`

Output dari fungsi ini berupa *logic (boolean)*:

```
stringr::str_detect(string,pattern)
```

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

- *Function* menggunakan `stringr::` dan fungsi `str_locate()`

Find starting and end position of all matches.

```
stringr::str_locate(string, pattern)
```

```
##      start end
## [1,]    NA  NA
## [2,]     1   5
## [3,]    NA  NA
## [4,]    NA  NA
## [5,]     1   5
```

- *Function* menggunakan `stringr::` dan fungsi `str_extract()`

Extract first match.

```
stringr::str_extract(string, pattern)
```

```
## [1] NA      "marke" NA      NA      "micke"
```

3.5.2 Replace Pattern

Kita akan gunakan contoh data berikut ini:

Ini adalah `string` yang digunakan:

```
string = c("This is a sentence about axis",  
           "A second pattern is also listed here")
```

Berikut adalah `pattern` dan `replacement` yang hendak dilakukan:

```
pattern = 'is'  
replace = 'XY'
```

Berikut adalah beberapa fungsi yang sering digunakan:

```
function sub()
```

```
sub(pattern, replace, string)
```

```
## [1] "ThXY is a sentence about axis"  
## [2] "A second pattern XY also listed here"
```

```
sub(pattern, replace, string, ignore.case = T)
```

```
## [1] "ThXY is a sentence about axis"  
## [2] "A second pattern XY also listed here"
```

```
function gsub()
```

```
gsub(pattern, replace, string)
```

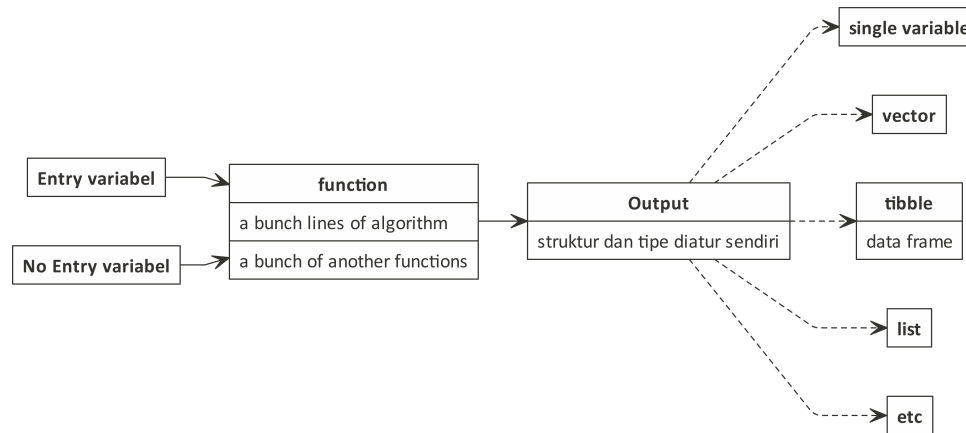
```
## [1] "ThXY XY a sentence about axXY"  
## [2] "A second pattern XY also lXYted here"
```

```
gsub(pattern, replace, string, ignore.case = T)
```

```
## [1] "ThXY XY a sentence about axXY"  
## [2] "A second pattern XY also lXYted here"
```


4 Membuat *function* di R.

R memungkinkan kita untuk membuat fungsi yang *custom* secara mandiri. Jika kita sering menggunakan perintah-perintah tertentu dan ingin menghemat penulisan algoritma, kita bisa membuat *custom function()* sendiri.



`function()` bisa memerlukan *entry variable* atau tidak memerlukan *entry variable* sama sekali.

4.1 `function()` tanpa *entry variable*

Ada kalanya kita membuat `function()` tanpa ada *entry variable*. *Lho kok gitu?*

Tergantung dari kebutuhan kita *yah*.

Sebagai contoh, kita akan membuat `function()` yang akan men- *generate* nama orang, umur, tinggi, dan berat badan:

```

demografi = function(){
  nama = randomNames::randomNames(1)
  umur = sample(c(20:60),1)
  tinggi = rnorm(1,mean = 150, sd = 20)
  tinggi = round(tinggi,1)
  berat = rnorm(1,mean = 40, sd = 5)
  berat = round(berat,1)
  data = c(nama,umur,tinggi,berat)
  return(data)
}
demografi()
  
```

```
## [1] "Dang, Kabee" "37"      "142"      "39.4"
```

4.2 `function()` dengan *entry variable*

Entry variable yang digunakan bisa berbentuk macam-macam dan bisa lebih dari satu.

Contoh, kita akan membuat `function()` untuk mencari modus dari sebuah *vector*:

```
modus = function(x) {  
  ux = unique(x)  
  tab = tabulate(match(x, ux))  
  ux[tab == max(tab)]  
}
```

Mari kita cek apakah `function`-nya berhasil atau tidak:

```
nama = c('a','b','a','c','d')  
modus(nama)
```

```
## [1] "a"
```

Contoh berikutnya kita akan buat `function()` untuk menghitung rumus pitagoras dengan dua *entry variables*, yakni `a` dan `b` sebagai berikut:

Ini adalah rumusnya:

```
pytagoras = function(a,b){  
  sqrt((a^2) + (b^2))  
}
```

Mari kita hitung pitagoras dengan `a = 3` dan `b = 4`, yakni:

```
pytagoras(3,4)
```

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


5 Membaca Data dari Excel

Data yang kita temui sehari-hari biasanya memiliki format `.xlsx` atau `.xls`. Sekarang kita akan *import* data dari **Excel** untuk masuk ke dalam **R**.

Library yang digunakan adalah `library(readxl)`

Contoh data yang digunakan:

```
library(readxl)
data = read_excel('Contoh_Data.xlsx')
head(data)
```

```
## # A tibble: 6 x 8
##   dept  nama.karyawan status bulan tardines.freque~ permit.frequency
##   <chr> <chr>         <chr> <chr>          <dbl>          <dbl>
## 1 JES   Jimenez         Tetap Jan              2              2
## 2 JES   Mccarthy         Tetap Mar              0              0
## 3 JES   Topaha           Tetap Jan              0              1
## 4 JES   el-Ghanem        Tetap Mar              3              1
## 5 JES   Topaha           Tetap Mei              0              1
## 6 JES   Lofton           Tetap Mei              0              2
## # ... with 2 more variables: sick.frequency <dbl>, leave.frequency <dbl>
```

5.1 Membaca Data dari Format File Lain

R memiliki kemampuan untuk membaca data dalam format lain seperti `csv`, `sav` (SPSS), `txt`, dan lainnya. Secara *default*, **R** bisa membaca file dalam bentuk `csv` dan `txt` tanpa menggunakan `library()` lainnya.

Gunakan fungsi `read.csv()` untuk membaca file `csv` dan `readLines()` untuk file `txt`.

6 Berkenalan dengan Tidyverse

Salah satu fungsi utama **R** adalah kemampuannya melakukan *data carpentry* untuk *dataset* berukuran besar dengan cepat. Salah satu **library** yang sering digunakan untuk melakukan *data carpentry* adalah **tidyverse**.

Salah satu ciri utama pada **tidyverse** adalah penggunaan *piping*: `%>%` yang memiliki arti *then*.

Beberapa fungsi yang sering digunakan antara lain:

1. `filter()`
2. `select()`
3. `mutate()`
4. `group_by()` dan `summarise()`
5. `separate()`

6.1 `filter()`

Digunakan untuk melakukan filter pada data. Kita bisa menggunakan tanda `==`, `>`, `>=`, `<`, `<=`, atau `!=`.

Contohnya:

Kita hendak melakukan *filter* terhadap karyawan yang telat (*tardines*) lebih dari 10 kali dalam bulan Januari.

```
# melakukan filter:
# keterlambatan > 10
# bulan == Jan
data %>% filter(tardines.frequency>10, bulan == 'Jan')
```

```
## # A tibble: 6 x 8
##   dept  nama.karyawan status bulan tardines.freque~ permit.frequency
##   <chr> <chr>      <chr> <chr>          <dbl>          <dbl>
## 1 JES   Herrera      Tetap Jan           14             1
## 2 JES   Garcia        Tetap Jan           19             3
## 3 ABD   Gomez          Tetap Jan           14             2
## 4 ABD   Sanchez        Tetap Jan           11             0
## 5 ABD   Archibeque     Tetap Jan           12             3
## 6 VIK   Sweetwater     Tetap Jan           14             1
## # ... with 2 more variables: sick.frequency <dbl>, leave.frequency <dbl>
```

Misalkan kita hendak melakukan *filter* untuk beberapa `dept` tertentu, kita bisa melakukan cara berikut:

```
dept_filter = c('JES','ELL','OSH')
```

Melakukan *filter* untuk `dept` tersebut:

```
data %>% filter(dept %in% dept_filter)
```

```
## # A tibble: 160 x 8
##   dept nama.karyawan status bulan tardines.freque~ permit.frequency
##   <chr> <chr>      <chr> <chr>          <dbl>          <dbl>
## 1 JES   Jimenez      Tetap Jan           2             2
## 2 JES   Mccarthy      Tetap Mar           0             0
## 3 JES   Topaha        Tetap Jan           0             1
## 4 JES   el-Ghanem     Tetap Mar           3             1
## 5 JES   Topaha        Tetap Mei          0             1
## 6 JES   Lofton        Tetap Mei          0             2
## 7 JES   Porambo       Tetap Apr           9             1
## 8 JES   Porambo       Tetap Feb           8             2
## 9 JES   Porambo       Tetap Mei          3             2
## 10 JES   Jimenez       Tetap Mei          0             2
## # ... with 150 more rows, and 2 more variables: sick.frequency <dbl>,
## #   leave.frequency <dbl>
```

Melakukan *filter* untuk **BUKAN** `dept` tersebut:

```
data %>% filter(!dept %in% dept_filter)
```

```
## # A tibble: 3,729 x 8
##   dept nama.karyawan status bulan tardines.freque~ permit.frequency
##   <chr> <chr>      <chr> <chr>          <dbl>          <dbl>
## 1 JAA   Rayford     Tetap Apr           3             1
## 2 JAA   Rayford     Tetap Jan           3             1
## 3 JAA   Rayford     Tetap Mei          0             2
## 4 JAA   Rayford     Tetap Mar           4             0
## 5 JAA   Rayford     Tetap Feb           4             3
## 6 LOR   Schwalger    Tetap Mei          0             2
## 7 LOR   Xue          Tetap Jan           2             0
## 8 LOR   Xue          Tetap Apr           0             1
## 9 LOR   Schwalger    Tetap Feb           0             2
## 10 LOR   Kwan         Tetap Mei          0             2
## # ... with 3,719 more rows, and 2 more variables: sick.frequency <dbl>,
## #   leave.frequency <dbl>
```

Perhatikan penggunaan tanda seru ! pada skrip untuk mengaktifkan **NOT** sebelum `grepl()`.

6.2 arrange()

Digunakan untuk melakukan *sort* pada data dengan menggunakan *piping*. Kita akan *sort* berdasarkan angka *tardines* tersebut.

6.2.1 Descending

```
data %>%
  filter(tardines.frequency>10, bulan == 'Jan') %>%
  arrange(desc(tardines.frequency))
```

```
## # A tibble: 6 x 8
##   dept  nama.karyawan status  bulan  tardines.freque~ permit.frequency
##   <chr> <chr>          <chr> <chr>          <dbl>          <dbl>
## 1 JES   Garcia          Tetap Jan           19             3
## 2 JES   Herrera          Tetap Jan           14             1
## 3 ABD   Gomez            Tetap Jan           14             2
## 4 VIK   Sweetwater        Tetap Jan           14             1
## 5 ABD   Archibeque        Tetap Jan           12             3
## 6 ABD   Sanchez           Tetap Jan           11             0
## # ... with 2 more variables: sick.frequency <dbl>, leave.frequency <dbl>
```

6.2.2 Ascending

```
data %>%
  filter(tardines.frequency>10, bulan == 'Jan') %>%
  arrange(tardines.frequency)
```

```
## # A tibble: 6 x 8
##   dept  nama.karyawan status  bulan  tardines.freque~ permit.frequency
##   <chr> <chr>          <chr> <chr>          <dbl>          <dbl>
## 1 ABD   Sanchez           Tetap Jan           11             0
## 2 ABD   Archibeque        Tetap Jan           12             3
## 3 JES   Herrera          Tetap Jan           14             1
## 4 ABD   Gomez            Tetap Jan           14             2
## 5 VIK   Sweetwater        Tetap Jan           14             1
## 6 JES   Garcia          Tetap Jan           19             3
## # ... with 2 more variables: sick.frequency <dbl>, leave.frequency <dbl>
```

6.3 `select()`

Digunakan untuk memilih variabel dari *dataset*. Jika `filter()` dilakukan untuk melakukan pemilihan atas variabel tertentu, sedangkan `select()` digunakan untuk memilih variabel yang akan digunakan.

Contoh:

Kita akan memilih variabel `dept` dan `sick.frequency` dari data tersebut.

```
data %>% select(dept,sick.frequency)
```

```
## # A tibble: 3,889 x 2
##   dept sick.frequency
##   <chr>         <dbl>
## 1 JES             0
## 2 JES             0
## 3 JES             1
## 4 JES             0
## 5 JES             0
## 6 JES             0
## 7 JES             5
## 8 JES             0
## 9 JES             0
## 10 JES            1
## # ... with 3,879 more rows
```

Misalkan kita hendak memilih semua variabel yang mengandung kata *frequency*, kita bisa lakukan hal berikut:

```
data %>% select(contains('frequency'))
```

```
## # A tibble: 3,889 x 4
##   tardines.frequency permit.frequency sick.frequency leave.frequency
##   <dbl>         <dbl>         <dbl>         <dbl>
## 1             2             2             0             3
## 2             0             0             0             3
## 3             0             1             1             1
## 4             3             1             0             2
## 5             0             1             0             1
## 6             0             2             0             1
## 7             9             1             5             0
## 8             8             2             0             3
## 9             3             2             0             0
## 10            0             2             1             3
## # ... with 3,879 more rows
```

6.4 mutate()

Digunakan untuk membuat dan menghitung variabel baru atau *existing*.

Misalkan kita hendak membuat variabel baru bernama `telat.y.n` yang gunanya untuk mengecek apakah karyawan di bulan tersebut pernah telat atau tidak:

```
data %>%
  mutate(telat.y.n = ifelse(tardines.frequency>0,'Yes','No')) %>%
  select(dept,nama.karyawan,bulan,telat.y.n)
```

```
## # A tibble: 3,889 x 4
##   dept  nama.karyawan bulan telat.y.n
##   <chr> <chr>         <chr> <chr>
## 1 JES   Jimenez       Jan   Yes
## 2 JES   Mccarthy       Mar   No
## 3 JES   Topaha         Jan   No
## 4 JES   el-Ghanem       Mar   Yes
## 5 JES   Topaha         Mei   No
## 6 JES   Lofton         Mei   No
## 7 JES   Porambo        Apr   Yes
## 8 JES   Porambo        Feb   Yes
## 9 JES   Porambo        Mei   Yes
## 10 JES  Jimenez       Mei   No
## # ... with 3,879 more rows
```

Misalkan kita hendak me- *replace* variabel `tardines.frequency` dan menggantinya menjadi 3 kelompok kelas (low, med, high):

```
data %>%
  filter(tardines.frequency>0) %>% # melakukan filter hanya utk karyawan yag telat
  filter(status == "Tetap") %>% # hanya yang statusnya tetap
  filter(bulan == 'Jan') %>% # hanya pada bulan Jan
  mutate(tardines.frequency = cut(tardines.frequency,
                                   3,
                                   labels=c('low','med','high')))) # membagi tardines menjadi 3 kelas
```

```
## # A tibble: 185 x 8
##   dept  nama.karyawan status bulan tardines.freque~ permit.frequency
##   <chr> <chr>         <chr> <chr> <fct>                <dbl>
## 1 JES   Jimenez       Tetap Jan   low                2
## 2 JES   el-Ghanem     Tetap Jan   low                0
## 3 JES   Porambo       Tetap Jan   low                1
## 4 JES   Nicklas       Tetap Jan   low                0
## 5 JES   Marsh         Tetap Jan   low                3
## 6 JES   Noon          Tetap Jan   low                3
## 7 JES   Herrera       Tetap Jan   high               1
## 8 JES   Littrell      Tetap Jan   low                2
## 9 JES   Garcia        Tetap Jan   high               3
## 10 JES  Kim          Tetap Jan   low                0
## # ... with 175 more rows, and 2 more variables: sick.frequency <dbl>,
## #   leave.frequency <dbl>
```

Fungsi `cut()` membagi data numerik menjadi kelas-kelas tertentu Untuk melakukan *vectorize* dari *conditional ifelse()*, kita bisa menggunakan fungsi `case_when()`.

6.5 `group_by()` dan `summarise()`

Digunakan untuk melakukan pengelompokkan serta membuat dan menghitung variabel baru atau *existing* berdasarkan pengelompokkan tersebut.

Contoh:

Menghitung berapa banyak karyawan setiap bulannya:

```
data %>%
  group_by(bulan) %>%
  summarise(number_of_employee = length(unique(nama.karyawan)))
```

```
## # A tibble: 5 x 2
##   bulan number_of_employee
##   <chr>           <int>
## 1 Apr             667
## 2 Feb             657
## 3 Jan             659
## 4 Mar             660
## 5 Mei             658
```

Menghitung berapa banyak karyawan yang sakit setiap bulannya:

```
data %>%
  filter(sick.frequency > 0) %>%
  group_by(bulan) %>%
  summarise(number_of_sick_employee = length(unique(nama.karyawan)))
```

```
## # A tibble: 5 x 2
##   bulan number_of_sick_employee
##   <chr>           <int>
## 1 Apr             211
## 2 Feb             177
## 3 Jan             190
## 4 Mar             211
## 5 Mei             213
```


Menghitung berapa rata-rata frekuensi cuti dari karyawan setiap bulannya:

```
data %>%  
  filter(leave.frequency > 0) %>%  
  group_by(bulan) %>%  
  summarise(leave_avg = mean(leave.frequency))
```

```
## # A tibble: 5 x 2  
##   bulan leave_avg  
##   <chr>      <dbl>  
## 1 Apr        1.79  
## 2 Feb        1.92  
## 3 Jan        2.14  
## 4 Mar        1.92  
## 5 Mei        1.63
```

6.6 `separate()`

Digunakan untuk memecah satu variabel ke dua atau lebih variabel.

Contoh: misalkan kita memiliki data sebagai berikut:

Table 1: Absensi Karyawan

id	nama
1	Khan, Leah
2	Ray, Logan
3	Robinson, China
4	Antwi, Jereek
5	Wall, Naomi
6	al-Rahman, Rawda
7	Halstead, Brittania
8	Ayala, Kali
9	Flowers, Aliya
10	Foshag, Maxwell

Kita hendak memisahkan antara `first.name` dan `last.name` berdasarkan koma (,).

```
data_new %>%
  separate(nama,
           into = c('first.name', 'last.name'),
           sep = '\\,')
```

```
##   id first.name last.name
## 1  1      Khan      Leah
## 2  2       Ray     Logan
## 3  3  Robinson     China
## 4  4     Antwi    Jereek
## 5  5      Wall     Naomi
## 6  6 al-Rahman    Rawda
## 7  7  Halstead  Britannia
## 8  8     Ayala      Kali
## 9  9   Flowers    Aliya
## 10 10   Foshag   Maxwell
```

Jika diperhatikan ada penggunaan `\\` pada saat `separate` (`sep =`). Ini artinya kita hanya ingin menggunakan simbol setelah penggunaan `\\` yakni (,).

7 Beberapa Fungsi Statistik

Beberapa fungsi statistik yang kita ketahui di **Ms. Excel** juga memiliki nama yang sama di **R**. Biasanya, yang sering kita gunakan itu adalah:

7.1 `sum()`

Menghitung *sum* dari data berupa *vector*.

7.2 `mean()`

Menghitung *mean* dari data berupa *vector*.

7.3 `median()`

Menghitung *median* dari data berupa *vector*.

7.4 `sd()`

Menghitung *standar deviasi* dari data berupa *vector*.

8 Menggabungkan Data

Seringkali kita berurusan dengan beberapa *datasets* dan mengharuskan kita untuk menggabungkan beberapa *datasets* tersebut.

Contoh paling sederhana adalah melakukan *vlookup* seperti yang biasa kita lakukan pada Ms. Excel.

Di **R**, kita tidak hanya bisa melakukan *vlookup* saja tapi bisa juga teknik penggabungan data yang lain.

8.1 `merge()`

`merge()` biasa digunakan untuk menggabungkan dua data dengan prinsip yang sama dengan *vlookup*, yakni harus ada *key id* variabel yang sama antara dua data tersebut.

Misalkan saya punya *dataset* pertama (`data_1`) sebagai berikut:

Table 2: Dataset Pertama

bulan	number_of_employee
Apr	667
Feb	657
Jan	659
Mar	660
Mei	658

Saya ingin menggabungkan *dataset* di atas dengan *dataset* kedua (`data_2`) berikut ini:

Table 3: Dataset Kedua

bulan	number_of_chairs
Jan	687
Feb	690
Mar	642
Apr	651
Mei	659

Jika kita hendak melakukan `merge()`, kita bisa lakukan dengan dua cara:

8.1.1 Cara Klasik

```
merge(data_1,data_2)
```

```
##   bulan number_of_employee number_of_chairs
## 1   Apr                667                651
## 2   Feb                657                690
## 3   Jan                659                687
## 4   Mar                660                642
## 5   Mei                658                659
```

8.1.2 Menggunakan Tidyverse

```
data_1 %>% merge(data_2)
```

```
##   bulan number_of_employee number_of_chairs
## 1   Apr                667                651
## 2   Feb                657                690
## 3   Jan                659                687
## 4   Mar                660                642
## 5   Mei                658                659
```

Perintah `merge()` ini juga memiliki banyak fitur lainnya. Coba kalian cek dengan perintah `?merge` untuk melihat apa saja yang bisa dilakukan.

8.2 rbind()

`rbind()` dilakukan jika kita hendak menggabungkan dua *datasets* yang memiliki *variable names* yang sama. Berbeda dengan prinsip *vlookup*, penggabungan ini adalah hanya menaruh data kedua dibawah urutan data pertama.

Contohnya, saya punya dataset pertama (`data_3`) sebagai berikut:

Table 4: Dataset Pertama

bulan	number_of_sick_employee
Apr	211
Feb	177
Jan	190
Mar	211
Mei	213

Lalu kita hendak menggabungkannya dengan *dataset* kedua (`data_4`) berikut ini:

Table 5: Dataset Kedua

bulan	number_of_sick_employee
Jun	212
Jul	195
Agu	188
Sep	191
Okt	216

Maka caranya adalah:

8.2.1 Cara Klasik

```
rbind(data_3,data_4)
```

```
## # A tibble: 10 x 2
##   bulan number_of_sick_employee
##   <chr>             <int>
## 1 Apr               211
## 2 Feb               177
## 3 Jan               190
## 4 Mar               211
## 5 Mei               213
## 6 Jun               212
## 7 Jul               195
## 8 Agu               188
## 9 Sep               191
## 10 Okt              216
```

8.2.2 Menggunakan Tidyverse

```
data_3 %>% rbind(data_4)
```

```
## # A tibble: 10 x 2
##   bulan number_of_sick_employee
##   <chr>             <int>
## 1 Apr               211
## 2 Feb               177
## 3 Jan               190
## 4 Mar               211
## 5 Mei               213
## 6 Jun               212
## 7 Jul               195
## 8 Agu               188
## 9 Sep               191
## 10 Okt              216
```

Perintah `rbind()` ini juga bisa dilakukan untuk menggabungkan dua *datasets* yang *variable names*-nya ada yang berbeda. Jadi tidak harus sama, tapi minimal ada satu yang sama.

Misalkan dua *datasets* ini:

```
data_new_1
```

```
##   bulan number_of_employee number_of_chairs
## 1 Apr           667           651
## 2 Feb           657           690
## 3 Jan           659           687
## 4 Mar           660           642
## 5 Mei           658           659
```



```
data_new_2
```

```
## # A tibble: 10 x 2
##   bulan number_of_sick_employee
##   <chr>           <int>
## 1 Apr             211
## 2 Feb             177
## 3 Jan             190
## 4 Mar             211
## 5 Mei             213
## 6 Jun             212
## 7 Jul             195
## 8 Agu             188
## 9 Sep             191
## 10 Okt            216
```

```
data_new_1[setdiff(names(data_new_2), names(data_new_1))] = NA
data_new_2[setdiff(names(data_new_1), names(data_new_2))] = NA
rbind(data_new_1, data_new_2)
```

```
##   bulan number_of_employee number_of_chairs number_of_sick_employee
## 1   Apr             667             651             NA
## 2   Feb             657             690             NA
## 3   Jan             659             687             NA
## 4   Mar             660             642             NA
## 5   Mei             658             659             NA
## 6   Apr              NA              NA             211
## 7   Feb              NA              NA             177
## 8   Jan              NA              NA             190
## 9   Mar              NA              NA             211
## 10  Mei              NA              NA             213
## 11  Jun              NA              NA             212
## 12  Jul              NA              NA             195
## 13  Agu              NA              NA             188
## 14  Sep              NA              NA             191
## 15  Okt              NA              NA             216
```

Ingat *yah*, hanya menaruh *dataset* kedua secara berurut ada di bawah *dataset* pertama.

9 *Libraries* yang Berguna

Pada *section 1.7* saya sempat menyebutkan beberapa *libraries* yang berguna. Faedah dari `library(dplyr)` sudah termasuk dalam pembahasan `library(tidyverse)`. Oleh karena itu, saya akan coba bahas *libraries* lainnya satu- persatu.

9.1 `library(janitor)`

Pada *section 2.2* saya telah menyebutkan *library* yang satu ini. Salah satu faedahnya yang sering saya pakai adalah untuk membersihkan nama variabel (`colnames()`) dari suatu `data.frame`. Tapi tidak menutup kemungkinan digunakan juga untuk membersihkan *text* pada saat kita hendak melakukan *text analysis*.

9.1.1 *Function* `make_clean_names()`

Function ini berguna untuk membuat membersihkan data berbentuk *character* dengan cara:

1. Membuat *character* menjadi *lowercase*.
2. Mengubah spasi menjadi *underscore* `_`.
3. Menghilangkan tanda baca dan *non alpha numeric character*.

Contohnya: misalkan saya memiliki *vector* berisi *string* sebagai berikut:

```
kata = c('Pulang pergi', 'Selamat Pagi', 'kamu siapa?', 'Nama (lengkap...)', 'Bersiap! 1,2,3!')
kata
```

```
## [1] "Pulang pergi"      "Selamat Pagi"      "kamu siapa?"
## [4] "Nama (lengkap...)" "Bersiap! 1,2,3!"
```

Perhatikan *output* dari *function* ini:

```
library(janitor)
```

```
##
## Attaching package: 'janitor'

## The following objects are masked from 'package:stats':
##
##      chisq.test, fisher.test
```

```
make_clean_names(kata)
```

```
## [1] "pulang_pergi"  "selamat_pagi"  "kamu_siapa"    "nama_lengkap"
## [5] "bersiap_1_2_3"
```

9.1.2 Function clean_names()

Function ini digunakan untuk membersihkan colnames() dari suatu data.frame.

Misalkan: saya memiliki data seperti ini:

Table 6: Data Harian COVID-19

Tanggal Lapor	Kasus baru	Kasus Impor	Kasus Lokal	Total kasus	Kasus aktif	% kasus aktif
2020-03-02	2	0	2	2	2	1.0000000
2020-03-03	0	0	0	2	2	1.0000000
2020-03-04	0	0	0	2	2	1.0000000
2020-03-05	0	0	0	2	2	1.0000000
2020-03-06	2	0	2	4	4	1.0000000
2020-03-07	0	0	0	4	4	1.0000000
2020-03-08	2	1	1	6	6	1.0000000
2020-03-09	13	7	6	19	19	1.0000000
2020-03-10	8	5	3	27	27	1.0000000
2020-03-12	0	0	0	34	27	0.7941176

Seperti yang telah disebutkan sebelumnya, penamaan variabel sebisa mungkin dibuat simpel agar penulisan *script* menjadi lebih cepat dan mudah. Berikut adalah nama variabel (colnames()) dari data.frame tersebut:

```
colnames(data)
```

```
## [1] "Tanggal Lapor" "Kasus baru"    "Kasus Impor"   "Kasus Lokal"
## [5] "Total kasus"   "Kasus aktif"   "% kasus aktif"
```

Untuk mengubahnya, kita bisa menggunakan salah satu dari dua cara berikut ini:

- Cara tidyverse

```
data_new_1 = data %>% clean_names()
head(data_new_1,10)
```

```
## # A tibble: 10 x 7
##   tanggal_lapor      kasus_baru kasus_impor kasus_lokal total_kasus
##   <dtm>          <dbl>      <dbl>      <dbl>      <dbl>
## 1 2020-03-02 00:00:00         2         0         2         2
## 2 2020-03-03 00:00:00         0         0         0         2
## 3 2020-03-04 00:00:00         0         0         0         2
## 4 2020-03-05 00:00:00         0         0         0         2
## 5 2020-03-06 00:00:00         2         0         2         4
## 6 2020-03-07 00:00:00         0         0         0         4
## 7 2020-03-08 00:00:00         2         1         1         6
## 8 2020-03-09 00:00:00        13         7         6        19
## 9 2020-03-10 00:00:00         8         5         3        27
## 10 2020-03-12 00:00:00         0         0         0        34
## # ... with 2 more variables: kasus_aktif <dbl>, percent_kasus_aktif <dbl>
```

- Cara klasik

```
data_new_2 = clean_names(data)
head(data_new_2,10)
```

```
## # A tibble: 10 x 7
##   tanggal_lapor      kasus_baru kasus_impор kasus_lokal total_kasus
##   <dtm>          <dbl>      <dbl>      <dbl>      <dbl>
## 1 2020-03-02 00:00:00         2         0         2         2
## 2 2020-03-03 00:00:00         0         0         0         2
## 3 2020-03-04 00:00:00         0         0         0         2
## 4 2020-03-05 00:00:00         0         0         0         2
## 5 2020-03-06 00:00:00         2         0         2         4
## 6 2020-03-07 00:00:00         0         0         0         4
## 7 2020-03-08 00:00:00         2         1         1         6
## 8 2020-03-09 00:00:00        13         7         6        19
## 9 2020-03-10 00:00:00         8         5         3        27
## 10 2020-03-12 00:00:00         0         0         0        34
## # ... with 2 more variables: kasus_aktif <dbl>, percent_kasus_aktif <dbl>
```

9.2 library(lubridate)

Beberapa kali saya dihadapkan pada data tanggal dan waktu. Untuk itu, saya biasa menggunakan `library(lubridate)` untuk membantu saya untuk melakukannya. Mari kita lihat kembali data COVID-19 yang sudah kita bersihkan sebelumnya:

Table 7: 5 Data Teratas dari Data COVID-19 Cleaned

tanggal_lapor	kasus_baru	kasus_impор	kasus_lokal	total_kasus	kasus_aktif	percent_kasus_aktif
2020-03-02	2	0	2	2	2	1
2020-03-03	0	0	0	2	2	1
2020-03-04	0	0	0	2	2	1
2020-03-05	0	0	0	2	2	1
2020-03-06	2	0	2	4	4	1

Kita coba cek tipe data dari masing-masing variabel yang ada pada data tersebut:

```
str(data_new_2)
```

```
## tibble [139 x 7] (S3: tbl_df/tbl/data.frame)
## $ tanggal_lapor      : POSIXct[1:139], format: "2020-03-02" "2020-03-03" ...
## $ kasus_baru          : num [1:139] 2 0 0 0 2 0 2 13 8 0 ...
## $ kasus_impор         : num [1:139] 0 0 0 0 0 0 1 7 5 0 ...
## $ kasus_lokal         : num [1:139] 2 0 0 0 2 0 1 6 3 0 ...
## $ total_kasus         : num [1:139] 2 2 2 2 4 4 6 19 27 34 ...
## $ kasus_aktif         : num [1:139] 2 2 2 2 4 4 6 19 27 27 ...
## $ percent_kasus_aktif : num [1:139] 1 1 1 1 1 1 1 ...
```

Ternyata variabel `tanggal_lapor` tidak bertipe `date`. Oleh karena itu, kita akan *convert* ke tipe `date` dengan *function* yang ada di `library(lubridate)` yakni `date()`.

```
library(lubridate)

data_new_2 =
  data_new_2 %>%
  mutate(tanggal_lapor = date(tanggal_lapor))

str(data_new_2)

## tibble [139 x 7] (S3: tbl_df/tbl/data.frame)
##  $ tanggal_lapor      : Date[1:139], format: "2020-03-02" "2020-03-03" ...
##  $ kasus_baru          : num [1:139] 2 0 0 0 2 0 2 13 8 0 ...
##  $ kasus_impор        : num [1:139] 0 0 0 0 0 0 1 7 5 0 ...
##  $ kasus_lokal         : num [1:139] 2 0 0 0 2 0 1 6 3 0 ...
##  $ total_kasus         : num [1:139] 2 2 2 2 4 4 6 19 27 34 ...
##  $ kasus_aktif         : num [1:139] 2 2 2 2 4 4 6 19 27 27 ...
##  $ percent_kasus_aktif: num [1:139] 1 1 1 1 1 1 ...
```

Sekarang `tanggal_lapor` sudah bertipe `date`, oleh karena itu kita bisa mengekstrak beberapa informasi lain terkait waktu sebagai berikut:

```
data_tanggal =
  data_new_2 %>%
  mutate(
    bulan = month(tanggal_lapor,label = T),
    tahun = year(tanggal_lapor),
    tanggal = day(tanggal_lapor),
    hari = wday(tanggal_lapor,label = T),
    total_hari_sebulan = days_in_month(tanggal_lapor),
    minggu_ke = epiweek(tanggal_lapor)
  ) %>%
  select(tanggal_lapor,bulan,tahun,tanggal,hari,total_hari_sebulan,minggu_ke)
```

Table 8: Hasil Ekstrak Data Tanggal

tanggal_lapor	bulan	tahun	tanggal	hari	total_hari_sebulan	minggu_ke
2020-03-02	Mar	2020	2	Sen	31	10
2020-03-03	Mar	2020	3	Sel	31	10
2020-03-04	Mar	2020	4	Rab	31	10
2020-03-05	Mar	2020	5	Kam	31	10
2020-03-06	Mar	2020	6	Jum	31	10

Selain *functions* yang saya gunakan di atas, silakan di-*explore functions* apalagi yang bisa digunakan dari `library(lubridate)`.

9.3 library(ggplot2)

Seringkali kita harus membuat grafik atau visualisasi dari data yang kita olah. Salah satu *library* yang paling *powerful* untuk melakukan itu di **R** adalah `library(ggplot2)`.

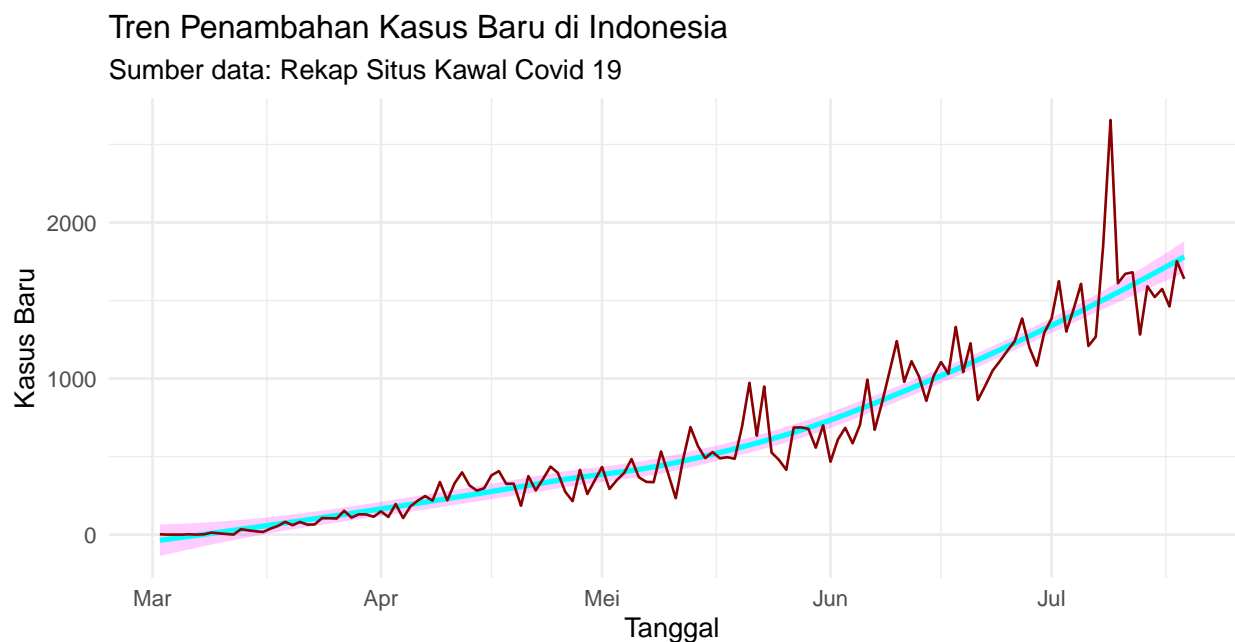
Hampir semua elemen dalam grafik bersifat *customize* dan bisa diubah melalui *script*. Kalian bisa mengecek semua yang bisa dilakukan di situs berikut ini:

1. *Ggplot2 essential*.
2. *Top 50 ggplot visualization*.
3. *R graph gallery*.

Salah satu contohnya dari data COVID-19 di atas adalah sebagai berikut:

```
library(ggplot2)

data_new_2 %>%
  ggplot(aes(x = tanggal_lapor,
             y = kasus_baru)) +
  geom_smooth(method = 'loess',
             color = 'cyan',
             fill = 'magenta',
             alpha = .2) +
  geom_line(color = 'darkred') +
  labs(x = 'Tanggal',
       y = 'Kasus Baru',
       title = 'Tren Penambahan Kasus Baru di Indonesia',
       subtitle = 'Sumber data: Rekap Situs Kawal Covid 19',
       caption = 'Visualized using R\nikanx101.github.io') +
  theme_minimal()
```



9.4 `library(reshape2)`

Seringkali kita berhadapan dengan bentuk data yang menyamping. Bentuk data tersebut tidak tabular dan akan menyulitkan bagi kita untuk melakukan analisa.

Kali ini saya akan menggunakan data contoh COVID-19 yang saya simpan di *link* berikut ini. Silakan diunduh untuk bisa mencobanya juga bersama-sama.

Mari kita *import* datanya ke dalam **R**. Berikut tampilannya:

Table 9: Contoh Data Tidak Tabular (hanya ditampilkan 10 kolom pertama saja)

Total Kasus	18-Mar	19-Mar	20-Mar	21-Mar	22-Mar	23-Mar	24-Mar	25-Mar	26-Mar	27-Mar
Jakarta	158	210	215	267	307	353	424	463	515	598
Jabar	24	26	41	55	59	59	60	73	78	98
Jateng	8	12	12	14	15	15	19	38	40	43
Jatim	8	9	15	26	41	41	51	51	59	66

Sebagai informasi, data ini berisi 125 kolom dengan kolom pertama adalah **Total Kasus** sedangkan 124 kolom lainnya adalah tanggal.

```
library(reshape2)
colnames(data)[1] = 'provinsi'

data_baru =
  data %>%
  melt(id.vars = 'provinsi')
```

Table 10: 15 Data Teratas dari Hasil Konversi Ke Tabular

provinsi	variable	value
Jakarta	18-Mar	158
Jabar	18-Mar	24
Jateng	18-Mar	8
Jatim	18-Mar	8
Jakarta	19-Mar	210
Jabar	19-Mar	26
Jateng	19-Mar	12
Jatim	19-Mar	9
Jakarta	20-Mar	215
Jabar	20-Mar	41
Jateng	20-Mar	12
Jatim	20-Mar	15
Jakarta	21-Mar	267
Jabar	21-Mar	55
Jateng	21-Mar	14

Didapatkan data_baru berisi 496 baris hasil konversi.

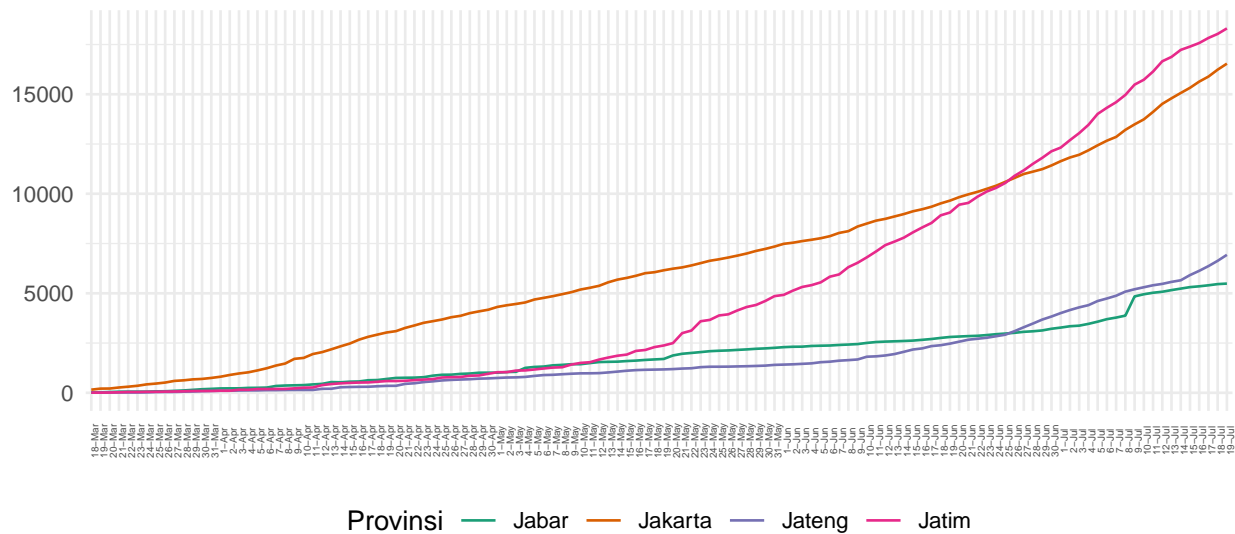
```
str(data_baru)
```

```
## 'data.frame':  496 obs. of  3 variables:
## $ provinsi: chr  "Jakarta" "Jabar" "Jateng" "Jatim" ...
## $ variable: Factor w/ 124 levels "18-Mar","19-Mar",...: 1 1 1 1 2 2 2 2 3 3 ...
## $ value   : num  158 24 8 8 210 26 12 9 215 41 ...
```

Dengan bentuk data seperti ini, kita bisa melakukan analisa *pivot* berdasarkan provinsi.

Penambahan Kasus Baru Per Provinsi

Sumber data: Situs Kawal Covid-19



Visualized using R
ikanx101.github.io

9.5 library(tidytext)

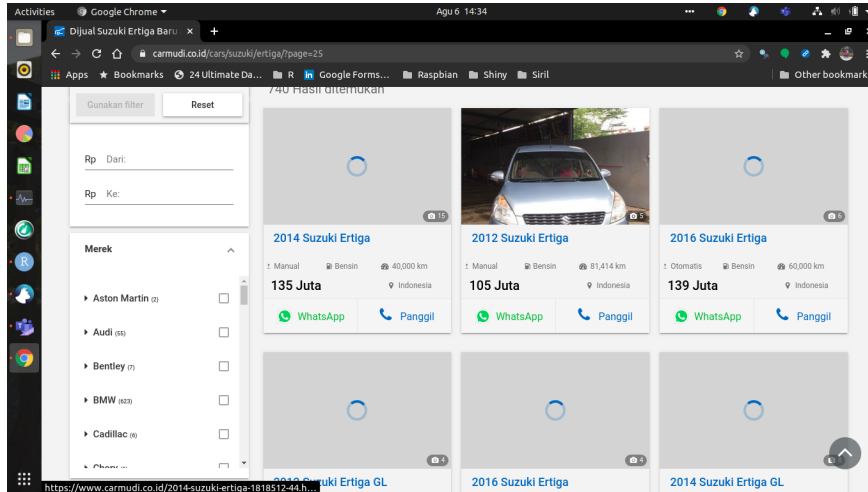
9.6 library(rvest)

10 APLIKASI R DI DUNIA REAL

Berikut akan saya sampaikan beberapa contoh kasus yang diselesaikan menggunakan R.

10.1 Mencari Harga Mobil Bekas

Salah satu keuntungan menggunakan **R** adalah saya bisa mengambil data dari *web* (*web scraping*) lalu langsung bisa membuat *report* nya menggunakan *markdown*. Kali ini saya akan mengambil data mobil **Suzuki Ertiga** yang di-*listing* pada situs **carmudi**.



Link dan Function yang Digunakan

Setelah dicek, ada 25 *pages* terkait **Suzuki Ertiga**.

```
## [1] "https://www.carmudi.co.id/cars/suzuki/ertiga/?page=1"
## [2] "https://www.carmudi.co.id/cars/suzuki/ertiga/?page=2"
## [3] "https://www.carmudi.co.id/cars/suzuki/ertiga/?page=3"
## [4] "https://www.carmudi.co.id/cars/suzuki/ertiga/?page=4"
## [5] "https://www.carmudi.co.id/cars/suzuki/ertiga/?page=5"
## [6] "https://www.carmudi.co.id/cars/suzuki/ertiga/?page=6"
## [7] "https://www.carmudi.co.id/cars/suzuki/ertiga/?page=7"
## [8] "https://www.carmudi.co.id/cars/suzuki/ertiga/?page=8"
## [9] "https://www.carmudi.co.id/cars/suzuki/ertiga/?page=9"
## [10] "https://www.carmudi.co.id/cars/suzuki/ertiga/?page=10"
## [11] "https://www.carmudi.co.id/cars/suzuki/ertiga/?page=11"
## [12] "https://www.carmudi.co.id/cars/suzuki/ertiga/?page=12"
## [13] "https://www.carmudi.co.id/cars/suzuki/ertiga/?page=13"
## [14] "https://www.carmudi.co.id/cars/suzuki/ertiga/?page=14"
## [15] "https://www.carmudi.co.id/cars/suzuki/ertiga/?page=15"
## [16] "https://www.carmudi.co.id/cars/suzuki/ertiga/?page=16"
## [17] "https://www.carmudi.co.id/cars/suzuki/ertiga/?page=17"
## [18] "https://www.carmudi.co.id/cars/suzuki/ertiga/?page=18"
## [19] "https://www.carmudi.co.id/cars/suzuki/ertiga/?page=19"
## [20] "https://www.carmudi.co.id/cars/suzuki/ertiga/?page=20"
## [21] "https://www.carmudi.co.id/cars/suzuki/ertiga/?page=21"
## [22] "https://www.carmudi.co.id/cars/suzuki/ertiga/?page=22"
```

```
## [23] "https://www.carmudi.co.id/cars/suzuki/ertiga/?page=23"
## [24] "https://www.carmudi.co.id/cars/suzuki/ertiga/?page=24"
## [25] "https://www.carmudi.co.id/cars/suzuki/ertiga/?page=25"
```

Fungsi *web scrape* yang saya buat adalah sebagai berikut:

```
scrap = function(url){
  data =
    read_html(url) %>% {
      tibble(
        nama = html_nodes(., '.title-blue') %>% html_text(),
        harga = html_nodes(., '.price a') %>% html_text(),
        lokasi = html_nodes(., '.catalog-listing-item-location span') %>%
          html_text()
      )
    }
  return(data)
}
```

Jadi setelah saya *scrape* semua 25 *pages*, hasilnya seperti ini:

Table 11: 10 Data Pertama dari Hasil Scrape Ertiga

nama	harga	lokasi
2020 Suzuki Ertiga PROMO XL7 DISCOUNT SA...	234 Juta	Bekasi
2020 Suzuki Ertiga PROMO ERTIGA DISKON S...	135 Juta	Kota Jakarta Timur
2020 Suzuki Ertiga PROMO SUZUKI ERTIGA D...	150 Juta	Depok
2020 Suzuki Ertiga BIG SALE SUZUKI NO TI...	175 Juta	Kota Jakarta Selatan
2019 Suzuki Ertiga SUZUKI ALL NEW ERTIGA...	135 Juta	Bekasi
2020 Suzuki Ertiga	135 Juta	Depok
2020 Suzuki Ertiga PROMO ALL NEW ERTIGA ...	145 Juta	Tangerang Selatan
2020 Suzuki Ertiga PROMO ...	155.5 Juta	Kota Jakarta Selatan
2020 Suzuki Ertiga Bandun...	200.5 Juta	Bandung
2020 Suzuki Ertiga PROMO ANGSURAN & DISK...	150 Juta	Depok

Cara melakukan *scrape* -nya, kalian bisa menggunakan *looping* `for(i in 1:25)` dengan fungsi `scrap(url[i])`. Jangan lupa menambahkan `rbind()` untuk menggabungkan data antar iterasi ke bawah. Mari kita lihat tipe variabel dari data tersebut:

```
str(ertiga.data)
```

```
## tibble [740 x 3] (S3: tbl_df/tbl/data.frame)
## $ nama : chr [1:740] "\n2020 Suzuki Ertiga PROMO XL7 DISCOUNT SA..." "\n2020 Suzuki Ertiga PROMO ..."
## $ harga : chr [1:740] "234 Juta" "135 Juta" "150 Juta" "175 Juta" ...
## $ lokasi: chr [1:740] "\nBekasi " "\nKota Jakarta Timur " "\nDepok " "\nKota Jakarta Selatan " ...
```

Data ini bisa kalian dapatkan di *link* berikut dalam format *.rda*.

Data bentuk ini belum bisa saya analisa karena masih *mix* antara *structured* dan *unstructured*. Oleh karena itu, saya akan bersihkan datanya sebagai berikut:

Membereskan Harga

```
ertiga.data =
  ertiga.data %>%
  mutate(harga = gsub('juta','',harga,ignore.case = T),
         harga = gsub('\\ ','',harga),
         harga = as.numeric(harga))
```

Table 12: 10 Data Pertama dari Hasil Scrape Ertiga: Cleaned Harga

nama	harga	lokasi
2020 Suzuki Ertiga PROMO XL7 DISCOUNT SA...	234.0	Bekasi
2020 Suzuki Ertiga PROMO ERTIGA DISKON S...	135.0	Kota Jakarta Timur
2020 Suzuki Ertiga PROMO SUZUKI ERTIGA D...	150.0	Depok
2020 Suzuki Ertiga BIG SALE SUZUKI NO TI...	175.0	Kota Jakarta Selatan
2019 Suzuki Ertiga SUZUKI ALL NEW ERTIGA...	135.0	Bekasi
2020 Suzuki Ertiga	135.0	Depok
2020 Suzuki Ertiga PROMO ALL NEW ERTIGA ...	145.0	Tangerang Selatan
2020 Suzuki Ertiga PROMO ...	155.5	Kota Jakarta Selatan
2020 Suzuki Ertiga Bandun...	200.5	Bandung
2020 Suzuki Ertiga PROMO ANGSURAN & DISK...	150.0	Depok

Membereskan Variabel nama dan Mengekstrak tahun

Oke, sekarang kita akan membereskan variabel **nama**. Sebelum melakukannya, saya akan menghapuskan tanda `\n` dan menambahkan variabel **id** untuk memudahkan proses ekstrak **tahun** nantinya.

```
ertiga.data =
  ertiga.data %>%
  mutate(nama = gsub('\\\\n','',nama),
         id = c(1:length(nama)))
```

Table 13: 10 Data Pertama dari Hasil Scrape Ertiga: Cleaned Nama

nama	harga	lokasi	id
2020 Suzuki Ertiga PROMO XL7 DISCOUNT SA...	234.0	Bekasi	1
2020 Suzuki Ertiga PROMO ERTIGA DISKON S...	135.0	Kota Jakarta Timur	2
2020 Suzuki Ertiga PROMO SUZUKI ERTIGA D...	150.0	Depok	3
2020 Suzuki Ertiga BIG SALE SUZUKI NO TI...	175.0	Kota Jakarta Selatan	4
2019 Suzuki Ertiga SUZUKI ALL NEW ERTIGA...	135.0	Bekasi	5
2020 Suzuki Ertiga	135.0	Depok	6
2020 Suzuki Ertiga PROMO ALL NEW ERTIGA ...	145.0	Tangerang Selatan	7
2020 Suzuki Ertiga PROMO ...	155.5	Kota Jakarta Selatan	8
2020 Suzuki Ertiga Bandun...	200.5	Bandung	9
2020 Suzuki Ertiga PROMO ANGSURAN & DISK...	150.0	Depok	10

Nah, jika diperhatikan. Angka tahun pada variabel `nama` tidak berpola sama sekali. Kadang bisa muncul di depan, kadang bisa muncul di belakang, bahkan bisa muncul di pertengahan. Tapi ada kesamaannya yakni merupakan tipe data `numerik`. Oleh karena itu untuk mengekstrak `tahun` saya akan gunakan metode yang sama untuk membuat *word cloud* atau *word counting*. Pandang variabel `nama` sebagai satu kalimat utuh yang kemudian akan dipisah-pisah per kata. Setiap tipe data berupa `numerik` yang muncul akan kita jadikan variabel `tahun`.

```
new =
  ertiga.data %>% select(id,nama) %>%
  unnest_tokens('words',nama) %>%
  mutate(words = as.numeric(words)) %>%
  filter(!is.na(words),words>2000)

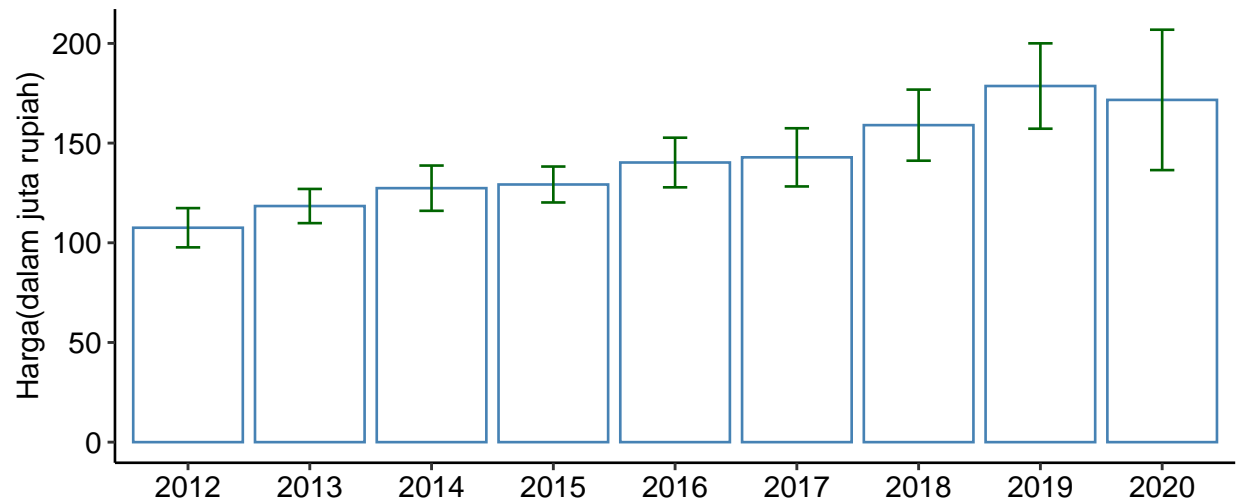
ertiga.data = merge(ertiga.data,new)
colnames(ertiga.data)[5] = 'tahun'
```

Table 14: 10 Data Pertama dari DATA FINAL

id	nama	harga	lokasi	tahun
1	2020 Suzuki Ertiga PROMO XL7 DISCOUNT SA...	234.0	Bekasi	2020
2	2020 Suzuki Ertiga PROMO ERTIGA DISKON S...	135.0	Kota Jakarta Timur	2020
3	2020 Suzuki Ertiga PROMO SUZUKI ERTIGA D...	150.0	Depok	2020
4	2020 Suzuki Ertiga BIG SALE SUZUKI NO TI...	175.0	Kota Jakarta Selatan	2020
5	2019 Suzuki Ertiga SUZUKI ALL NEW ERTIGA...	135.0	Bekasi	2019
6	2020 Suzuki Ertiga	135.0	Depok	2020
7	2020 Suzuki Ertiga PROMO ALL NEW ERTIGA ...	145.0	Tangerang Selatan	2020
8	2020 Suzuki Ertiga PROMO ...	155.5	Kota Jakarta Selatan	2020
9	2020 Suzuki Ertiga Bandun...	200.5	Bandung	2020
10	2020 Suzuki Ertiga PROMO ANGSURAN & DISK...	150.0	Depok	2020

Harga Suzuki Ertiga Bekas 2012 – 2020

source: situs Carmudi Indonesia



Scraped 6 Agustus 2020 15:00
Visualised using R
i k a n x

10.2 Membuat Alat Rapid Test COVID-19

Beberapa orang peneliti sedang mengembangkan dua alat pendeteksi cepat COVID-19. Cara kerja alat ini adalah untuk mendeteksi kadar *biomarker* dalam darah.

Apa sih *biomarker* itu?

Biomarker is a measurable substance in an organism whose presence is indicative of some phenomenon such as disease, infection, or enviromental exposure.

Diharapkan salah satu dari kedua *biomarker* tersebut bisa dengan akurat menentukan seseorang terkena COVID-19.

Berikut adalah datanya:

Table 15: 15 Data Pertama dari Data Biomarker

id	diagnosis	biomarker1	biomarker2
1	Tidak	7.801	13.422
2	Tidak	9.344	14.210
3	Tidak	9.066	13.490
4	Tidak	6.792	12.233
5	Ya	14.342	29.493
6	Tidak	6.622	14.325
7	Ya	12.458	20.632
8	Tidak	6.981	12.958
9	Ya	27.030	34.564
10	Tidak	8.720	13.752
11	Tidak	8.376	13.249
12	Tidak	9.919	14.813
13	Tidak	9.614	15.963
14	Ya	10.799	19.094
15	Tidak	8.711	14.656

Problem Statements

1. Berapa *cut off points* dari masing-masing *biomarker*? Maksudnya pada level berapa kadar *biomarker* bisa mengatakan positif COVID-19? Pada level berapa kadar *biomarker* bisa mengatakan negatif COVID-19? Misalkan jika kadar `biomarker1` ≥ 10 maka pasien disebut positif COVID-19 sedangkan kebalikannya negatif.
2. *Biomarker* mana yang lebih bagus?
3. Apakah kedua atau salah satu *biomarker* ini sudah baik untuk menggantikan diagnosis yang sebelumnya dilakukan?

Untuk semua pertanyaan di atas, mari kita lakukan analisa berikut ini:

Analisa ROC

Definisi ROC itu?

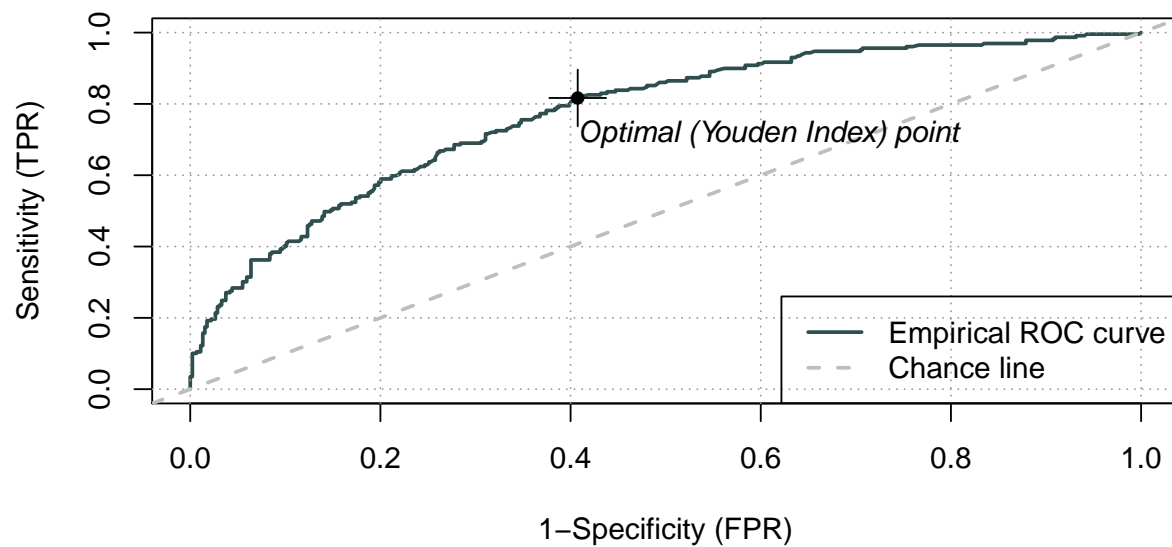
A receiver operating characteristic curve, or ROC curve, is a graphical plot that illustrates the diagnostic ability of a binary classifier system as its discrimination threshold is varied.

Menggunakan ROC ini, ada dua informasi yang bisa saya dapatkan:

1. Menentukan *cut off points* untuk masing - masing *biomarker*.
2. Membuat **ROC Curve** lalu menghitung *Area Under Curve* (AUC). **AUC** ini adalah nilai yang memberikan indikasi *goodness of fit* apakah suatu variabel bisa digunakan untuk menebak variabel lain. **AUC** ini juga yang biasa dijadikan acuan dalam setiap kompetisi *data science*. Semakin nilainya menuju 1, maka akan semakin bagus hasil prediksinya.

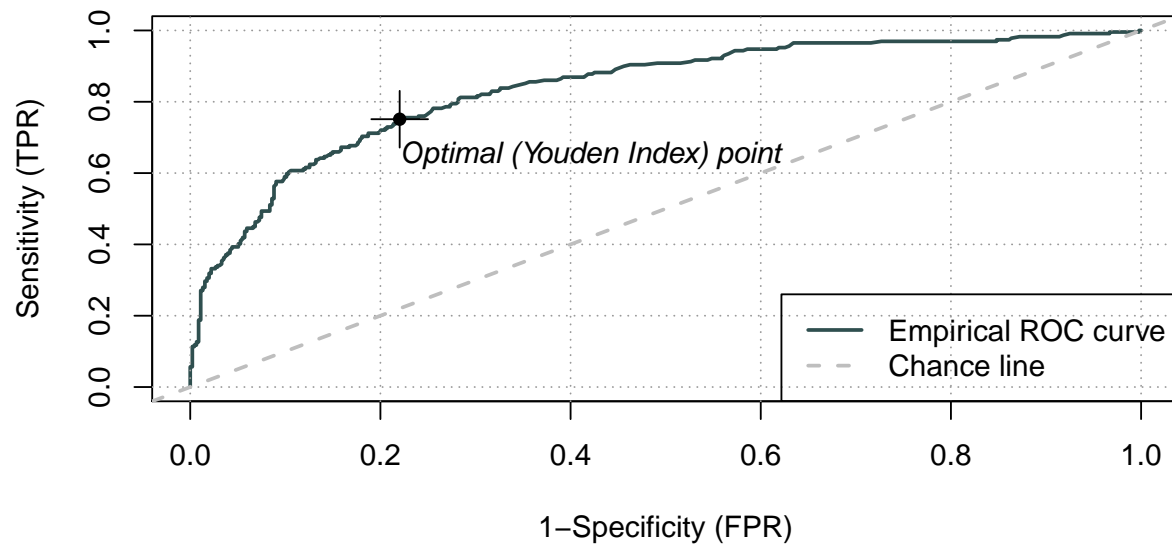
```
biomarker1
```

```
##
## Method used: empirical
## Number of positive(s): 229
## Number of negative(s): 454
## Area under curve: 0.7735
```



```
biomarker2
```

```
##
## Method used: empirical
## Number of positive(s): 229
## Number of negative(s): 454
## Area under curve: 0.8375
```



Kesimpulan dari nilai AUC

Dari nilai AUC kedua **ROC Curve** di atas, **biomarker2** menunjukkan angka yang terbaik dibandingkan **biomarker1**. Oleh karena itu, dugaan awal saya: **biomarker2** memiliki kemampuan lebih baik untuk mendiagnosis penyakit dibandingkan **biomarker1**.

Cut Off Points

Dari data numerik **biomarker1** dan **biomarker2**, saya akan menentukan batas berapa kadar bisa dibilang positif dan negatif. Untuk itu, saya akan menggunakan Youden's J statistic, yakni:

$$J = \text{sensitivity} + \text{specificity} - 1$$

Cut off points akan dipilih saat nilai J mencapai maksimum.

biomarker1

```
##
## Method used: empirical
## Number of positive(s): 229
## Number of negative(s): 454
## Area under curve: 0.7735
```

Cut off points untuk **biomarker1** terjadi pada 9.147, saat **TPR** = 0.8165939; **FPR** = 0.407489 sehingga menghasilkan $J = 0.4091$.

```
biomarker2
```

```
##
## Method used: empirical
## Number of positive(s): 229
## Number of negative(s): 454
## Area under curve: 0.8375
```

Cut off points untuk **biomarker2** terjadi pada 16.981, saat **TPR** = 0.7510917; **FPR** = 0.2202643 sehingga menghasilkan $J = 0.5308$.

Dari *cut off points* tersebut, saya membuat *rules* yakni jika kadar **biomarker** \geq *cut off points* maka akan ditulis **sakit**. Sedangkan jika kadar **biomarker** $<$ *cut off points* maka akan ditulis **tidak sakit**.

Confusion Matrix

Sekarang saya akan buat *confusion matrix* dari hasil transformasi tersebut lalu kita komparasi nilai *precision* dan *recall*.

Misalkan *precision* saya definisikan sebagai:

$$precision = \frac{jumlah.pasien.diprediksi.sakit.benar}{jumlah.pasien.diprediksi.sakit}$$

Sedangkan *recall* saya definisikan sebagai:

$$recall = \frac{jumlah.pasien.diprediksi.sakit.benar}{jumlah.pasien.sakit}$$

```
biomarker1
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Tidak Ya
##      Tidak    269  42
##      Ya       185 187
##
##           Accuracy : 0.6676
##           95% CI : (0.6309, 0.7029)
##      No Information Rate : 0.6647
##      P-Value [Acc > NIR] : 0.4534
##
##           Kappa : 0.3543
##
##  McNemar's Test P-Value : <2e-16
##
##           Sensitivity : 0.5925
##           Specificity : 0.8166
##           Pos Pred Value : 0.8650
##           Neg Pred Value : 0.5027
##           Prevalence : 0.6647
##           Detection Rate : 0.3939
```

```
## Detection Prevalence : 0.4553
## Balanced Accuracy : 0.7046
##
## 'Positive' Class : Tidak
##

biomarker2

## Confusion Matrix and Statistics
##
##           Reference
## Prediction Tidak Ya
##      Tidak   354  57
##      Ya      100 172
##
##           Accuracy : 0.7701
##           95% CI : (0.7367, 0.8012)
##      No Information Rate : 0.6647
##      P-Value [Acc > NIR] : 1.125e-09
##
##           Kappa : 0.5072
##
## Mcnemar's Test P-Value : 0.0008024
##
##           Sensitivity : 0.7797
##           Specificity : 0.7511
##      Pos Pred Value : 0.8613
##      Neg Pred Value : 0.6324
##           Prevalence : 0.6647
##      Detection Rate : 0.5183
##      Detection Prevalence : 0.6018
##      Balanced Accuracy : 0.7654
##
##      'Positive' Class : Tidak
##
```

Kesimpulan *Confusion Matrix*

1. biomarker1 memberikan *precision* = 50.27 dan *recall* = 81.66.
2. biomarker2 memberikan *precision* = 63.24 dan *recall* = 75.11.

KESIMPULAN Final

Melihat dari **AUC**, sebenarnya biomarker2 cukup bagus dibandingkan dengan biomarker1. Tapi dari nilai *precision* dan *recall* saya tidak merekomendasikan sama sekali kedua biomarker untuk digunakan.

Referensi

1. Metode Numerik Menggunakan R Untuk Teknik Lingkungan, Mohammad Rosidi: R bookdown.
2. Cara Install R di Android.
3. Install R base for Windows.
4. Install R Studio.
5. R Studio Cloud.
6. Bookdown, e-book from R Markdown.
7. Menggunakan R Studio Cloud di Android browser.
8. NOMNOML: How to make diagram in R.
9. Beberapa *puzzles* yang bisa diselesaikan dengan simulasi **Monte Carlo**.