MODUL 6
TIDYVERSE
TIDY DATA, MANIPULASI DATA FRAME, OPERATOR PIPE, MENYIMPULKAN & SORTING DATA

A. Tujuan Praktikum
- Memahami bentuk data yang bersifat *tidy*
- Mengimplementasikan fungsi-fungsi yang berkaitan dengan manipulasi *data frame*
- Mengimplementasikan penggunaan operator *pipe* dalam `tidyverse`
- Mengimplementasikan fungsi-fungsi yang berkaitan dengan penarikan kesimpulan pada tipe *data frame*
- Mengimplementasikan fungsi-fungsi yang berkaitan dengan pengurutan *data frame*

B. Alokasi Waktu
1 x pertemuan = 120 menit

C. Dasar Teori
Dengan adanya kebutuhan analisis data yang lebih kompleks, tipe data yang lebih banyak digunakan untuk penyimpanan data bukan lagi berupa vektor, melainkan *data frame*. Dalam modul 6-7 kita akan mempelajari lebih lanjut mengenai *data frame*, yang dapat memfasilitasi kemudahan pengorganisasian informasi. Maka dari itu, dimulai dari modul ini, sebagian besar data yang digunakan memiliki tipe *data frame*. Pada modul 6-7 juga akan dibahas mengenai data yang bersifat *tidy* (rapi) dan kumpulan paket yang dapat digunakan pada *tidy data*, yaitu `tidyverse`.

Kita dapat memuat semua paket `tidyverse` sekaligus dengan menginstal dan memanggil paket `tidyverse` melalui *script* berikut:

```
library(tidyverse)
```

Secara keseluruhan, tujuan rangkaian praktikum data science adalah untuk memahami langkah-langkah yang diperlukan dalam analisis data. Pada modul ini, akan dibahas mengenai bagaimana penerapan `tidyverse` yang sering digunakan dalam *data science.* Kita akan membahas beberapa fungsi `tidyverse` yang paling banyak digunakan, dimulai dengan paket `dplyr` untuk memanipulasi *data frame* dan paket `purrr` untuk proses data analisis yang berkaitan dengan fungsi. `tidyverse` juga memiliki paket khusus untuk visualisasi data, yaitu: `ggplot2`, yang akan dibahas lebih lanjut pada modul 8 mengenai Visualisasi Data; dan masih banyak lagi. Pertama, kita akan membahas mengenai konsep data yang bersifat *tidy* dan kemudian dilanjutkan dengan implementasi `tidyverse` pada tipe *data frame*.

TIDY DATA
Suatu data dapat dikatakan bersifat *tidy* apabila tiap baris tabelnya merepresentasikan satu nilai observasi dan tiap kolomnya terdiri dari variabel yang berbeda. *Dataset* "*murders*" yang telah kita gunakan pada modul ini adalah contoh *data frame* yang bersifat *tidy*.

```
#>    state      abb   region population total
#> 1  Alabama    AL    South  4779736    135
#> 2  Alaska     AK    West   710231     19
#> 3  Arizona    AZ    West   6392017    232
#> 4  Arkansas   AR    South  2915918    93
#> 5  California CA    West   37253956   1257
#> 6  Colorado   CO    West   5029196    65
```

Setiap baris mewakili negara dan kelima kolomnya memiliki variabel yang berbeda, tiap kolom berisi variabel-variabel terkait yang mendefinisikan karakteristik atau atribut dari tiap baris data, yaitu: *state*, *abb*, *region*, *population*, dan *total*.

Untuk memahami bagaimana menampilkan informasi yang sama dalam format yang berbeda, silahkan amati dua contoh data berikut:

```
#>   country     year fertility
#> 1 Germany     1960 2.41
#> 2 South Korea 1960 6.16
#> 3 Germany     1961 2.44
#> 4 South Korea 1961 5.99
#> 5 Germany     1962 2.47
#> 6 South Korea 1962 5.79
```

Dataset diatas mendefinisikan nilai *fertility* pada dua negara (Germany dan South Korea) dalam beberapa tahun. *Dataset* tersebut juga merupakan *dataset* yang bersifat *tidy* karena setiap baris menyajikan satu nilai pengamatan yang berisi tiga variabel, yaitu: *country*, *year*, dan *fertility*. Namun, *dataset* diatas sebenarnya merupakan *dataset* yang telah ditransformasi dari format berikut:

```
#>   country     1960 1961 1962
#> 1 Germany     2.41 2.44 2.47
#> 2 South Korea 6.16 5.99 5.79
```

Informasi yang sama dapat disediakan dalam beberapa bentuk yang berbeda, namun terdapat dua perbedaan penting dalam format kedua data diatas: 1) Pada data pertama, setiap baris mendefinisikan beberapa hasil pengamatan, 2) Pada data kedua, salah satu variabel, yaitu: tahun, disimpan sebagai *header*. Agar paket `tidyverse` dapat digunakan secara optimal, data yang digunakan perlu dipastikan telah bersifat *tidy*. Jika belum, data perlu ditransformasikan terlebih dahulu. Proses transformasi data menjadi data yang *tidy* akan dipelajari lebih lanjut pada modul mengenai *data wraggling*. Sehingga, untuk sementara, pada modul ini kita akan menggunakan contoh *dataset* yang sudah dalam format *tidy*.

MEMANIPULASI *DATA FRAME*

Paket `dplyr` yang terdapat pada `tidyverse` memiliki fungsi-fungsi yang dapat digunakan dalam proses manipulasi data dengan tipe *data frame* yang nama fungsinya relatif mudah diingat. Misalnya, untuk mengubah tabel dengan menambahkan kolom baru, dapat digunakan fungsi `mutate`. Sedangkan untuk memfilter tabel sehingga dapat menghasilkan suatu subset baris, dapat digunakan fungsi `filter`. Selanjutnya, untuk mengelompokkan data dengan memilih kolom tertentu, dapat digunakan fungsi `select`.

1. `mutate`

Misal, kita ingin semua informasi yang diperlukan untuk analisis dimasukkan dalam tabel data yang sudah ada, maka dapat digunakan fungsi `mutate` untuk menambahkan satu variabel baru

pada tabel data. Sebagai contoh, hasil penghitungan *murder_rate* akan kita tambahkan ke dalam *data frame* "*murders*" yang kita miliki dengan nama variabel baru "*rate*". Untuk menyelesaikan permasalahan ini, akan kita gunakan fungsi `mutate` untuk mengambil *data frame* sebagai argumen pertama dan nama serta nilai variabel baru yang ingin dimasukkan dalam *data frame* sebagai sebagai argument kedua (pada contoh kasus ini, argument kedua akan memiliki bentuk: "nama variabel = nilai"). Sehingga, untuk menambahkan variabel "*rate*", kita tuliskan *script* sebagai berikut:

```
library(dslabs)
data("murders")
murders <- mutate(murders, rate = total / population * 100000)
```

Yang perlu diperhatikan pada *script* diatas adalah, variabel *total* dan *population* di dalam fungsi `mutate` merupakan objek yang sebelumnya belum pernah didefinisikan di *workspace*. Namun, mengapa *script* diatas tidak menampilkan pesan *error*? Jawabannya adalah karena kita telah menggunakan library `dplyr`. Fungsi `mutate` dalam paket `dplyr`, dapat mengidentifikasi variabel-variabel dalam *data frame* yang telah didefinisikan dalam argumen pertama, sehingga *script* akan lebih mudah dibaca.

Setelah menjalankan script diatas, dapat dilihat menggunakan fungsi `head` bahwa kolom variabel baru telah berhasil ditambahkan:

```
head(murders)
#> state abb region population total rate
#> 1 Alabama AL South 4779736 135 2.82
#> 2 Alaska AK West 710231 19 2.68
#> 3 Arizona AZ West 6392017 232 3.63
#> 4 Arkansas AR South 2915918 93 3.19
#> 5 California CA West 37253956 1257 3.37
#> 6 Colorado CO West 5029196 65 1.29
```

Penambahan variabel yang dilakukan pada langkah-langkah diatas tidak akan mengubah data asli "*murders*". Hal ini dikarenakan, perubahan yang dilakukan hanya disimpan pada *workspace*. Sehingga jika selanjutnya kita menginputkan perintah `data(murders)` baru pada *script* selanjutnya, data asli yang belum ditambahkan variabel "*murder_rate*" akan me-*replace* hasil implementasi fungsi `mutate` diatas.

2. `filter`

Sekarang, anggaplah kita ingin menggunakan fungsi `filter` pada tabel, agar data yang ditampilkan hanya terdiri dari data-data yang nilai "*rate*"-nya kurang dari sama dengan 0,71. Fungsi `filter`, akan membutuhkan *data frame* yang akan dievaluasi sebagai argumen pertama dan kemudian pernyataan kondisional yang diinginkan sebagai argumen kedua. Sama halnya dengan fungsi `mutate`, kita tidak perlu mendefinisikan terlebih dahulu nama variabel yang belum dipanggil di *workspace* saat menggunakan fungsi ini. Namun, `filter` akan tetap mengenali variabel yang kita maksud.

```
filter(murders, rate <= 0.71)
#> state abb region population total rate
#> 1 Hawaii HI West 1360301 7 0.515
#> 2 Iowa IA North Central 3046355 21 0.689
#> 3 New Hampshire NH Northeast 1316470 5 0.380
#> 4 North Dakota ND North Central 672591 4 0.595
#> 5 Vermont VT Northeast 625741 2 0.320
```

3. `select`

Meskipun contoh *data frame* yang kita gunakan hanya memiliki enam kolom, tidak menutup kemungkinan bahwa suatu saat kita akan menemui tabel data yang memiliki puluhan hingga ratusan variabel. Jika kita hanya ingin menampilkan beberapa variabel dari keseluruhan yang ada, dapat menggunakan fungsi `select` yang terdapat pada `dplyr`. Contohnya, pada *script* di bawah ini, kita akan memilih tiga kolom dari *data frame* yang kita miliki, dan menyimpan tiga kolom yang kita pilih tersebut pada objek baru, selanjutnya, kita juga akan menggunakan fungsi `filter` objek baru agar data yang ditampilkan hanya terdiri dari data-data yang nilai "*rate*"-nya kurang dari sama dengan 0,71:

```
new_table <- select(murders, state, region, rate)
filter(new_table, rate <= 0.71)
#> state region rate
#> 1 Hawaii West 0.515
#> 2 Iowa North Central 0.689
#> 3 New Hampshire Northeast 0.380
#> 4 North Dakota North Central 0.595
#> 5 Vermont Northeast 0.320
```

Dapat dilihat pada *script* diatas, bahwa argumen pertama fungsi `select` adalah *data frame* yang kita miliki, selanjutnya argumen kedua, ketiga, dan seterusnya diisi dengan variabel yang kita pilih untuk disimpan pada objek *data frame* baru, yaitu: *state*, *region*, dan *rate*.

OPERATOR *PIPE*: %>%

Dengan `dplyr` kita juga dapat melakukan serangkaian operasi seperti yang telah kita lakukan di atas, misalnya memilih dan kemudian menyaring data, dengan mengirimkan hasil dari satu fungsi ke fungsi lain menggunakan operator *pipe* (%>%). Agar lebih mudah memahami operator *pipe*, kita akan mencoba untuk menghasilkan *output* yang sama dengan yang telah kita lakukan pada *script* diatas, yaitu: memilih tiga variabel (*state*, *region*, dan *rate*) dan hanya menampilkan *state* yang memiliki nilai *rate* <= 0,71. Dengan menggunakan `dplyr` kita dapat menulis *script* yang lebih sederhana sesuai dengan deskripsi alur yang kita inginkan (original data → `select` → `filter`) tanpa perlu membuat objek perantara:

Implementasi *script* nya adalah sebagai berikut:

```
murders %>% select(state, region, rate) %>% filter(rate <= 0.71)
#> state region rate
#> 1 Hawaii West 0.515
#> 2 Iowa North Central 0.689
#> 3 New Hampshire Northeast 0.380
#> 4 North Dakota North Central 0.595
#> 5 Vermont Northeast 0.320
```

Satu baris kode menggunakan operator *pipe* dapat menghasilkan output yang sama dengan dua baris kode pada contoh *script* sebelumnya. Operator *pipe* akan mengirimkan hasil operasi fungsi yang dilakukan pada sisi kiri %>% untuk selanjutnya menjadi argumen pertama dari fungsi di sisi kanannya. Contoh sederhananya,:

```
16 %>% sqrt()
#> [1] 4
```

Kemudian kita akan tambahkan operator *pipe*:

```
16 %>% sqrt() %>% log2()
#> [1] 2
```

Kita akan kembali lagi pada contoh alur (original data → `select` → `filter`) sebelumnya untuk identifikasi alur operasi *pipe* lebih rinci. *Data frame* "*murders*" adalah argumen pertama dari fungsi `select`, dan *data frame* baru (sebelumnya *new_table*) adalah argumen pertama dari fungsi `filter`. Dengan menggunakan operator *pipe*, *data frame* baru: '*new_table*' secara eksplisit akan diteruskan pada operasi fungsi selanjutnya tanpa perlu disimpan menjadi variabel baru terlebih dahulu.

MENYIMPULKAN DATA

Bagian yang cukup penting dari analisis dan eksplorasi data adalah menyimpulkan data. Rata-rata dan standar deviasi adalah dua contoh statistik yang banyak digunakan dalam analisis data. Di bagian ini, kita akan membahas dua fungsi `dplyr` lainnya yang dapat digunakan untuk menganalisis kesimpulan dari data yang dimiliki menggunakan `summarize` dan `group_by`.

1. `summarize`

Fungsi `summarize` di `dplyr` menyediakan cara untuk menghitung statistik dalam kode yang mudah dibaca. Kita akan mulai mengimplementasikan fungsi ini pada contoh *dataset* sederhana: "*heights*" yang berisi data tinggi siswa dan jenis kelamin. Sebelumnya, definisikan terlebih dahulu *library* dan *dataset* yang akan digunakan pada *script* selanjutnya:

```
library(dplyr)
library(dslabs)
data(heights)
```

Untuk menganalisa rata-rata dan standar deviasi tinggi badan untuk data dengan jenis kelamin wanita, *script* yang digunakan:

```
s <- heights %>%
filter(sex == "Female") %>%
summarize(average = mean(height), standard_deviation = sd(height))
s
#> average standard_deviation
#> 1 64.9 3.76
```

Alur dari program diatas dimulai dengan mengambil tabel data "*heights*" sebagai input, kemudian fungsi `filter` digunakan untuk menampilkan variabel *sex* yang nilainya *female* saja, selanjutnya, fungsi `summarize` akan menghasilkan *data frame* baru yang terdiri dari nilai rata-rata dan standar deviasi tinggi badan pada contoh *dataset* "*heights*" yang jenis kelaminnya adalah wanita. Kita juga dapat memilih nama variabel baru pada *data frame* yang dihasilkan. Contoh, di atas menggunakan nama variabel *average* untuk menyimpan hasil rata-rata tinggi siswa wanita dan *standard_deviation* untuk hasil komputasi statistik standa deviasi tinggi siswa wanita. Karena *data frame* yang dihasilkan disimpan dalam objek baru '*s*', kita dapat mengakses variabel yang dimiliki '*s*' dengan operator aksesor ($):

```
s$average
#> [1] 64.9
s$standard_deviation
#> [1] 3.76
```

2. `group_by`

Operasi umum lain yang dapat dilakukan dalam eksplorasi data adalah membagi data menjadi beberpa kelompok, kemudian dilanjutkan dengan menyimpulkan data dari tiap kelompok. Misalnya, kita akan menghitung rata-rata dan standar deviasi ketinggian pria dan wanita secara terpisah. Fungsi yang dapat digunakan untuk menyelesaikan skenario analisis diatas adalah `group_by`.

Untuk memperoleh kesimpulan dari studi kasus diatas, pertama kita akan kelompokkan terlebih dahulu data pada *dataset* "*heights*" berdasarkan jenis kelaminnya (*sex*) menggunakan *script* berikut:

```
heights %>% group_by(sex)
#> # A tibble: 1,050 x 2
#> # Groups: sex [2]
#> sex height
#>  <fct> <dbl>
#> 1 Male 75
#> 2 Male 70
#> 3 Male 68
#> 4 Male 74
#> 5 Male 61
#> # ... with 1,045 more rows
```

Hasil pengelompokan data berdasarkan jenis kelamin dapat dilihat pada keterangan `Groups: sex [2]` pada hasil diatas. Selanjutnya, ketika fungsi `summarize` ditambahkan pada alur program yang datanya telah dikelompokkan, hasilnya adalah:

```
heights %>%
group_by(sex) %>%
summarize(average = mean(height), standard_deviation = sd(height))
#> # A tibble: 2 x 3
#> sex average standard_deviation
#> <fct> <dbl> <dbl>
#> 1 Female 64.9 3.76
#> 2 Male 69.3 3.61
```

SORTING DATA FRAMES

Saat menganalisa *dataset*, sering kali kita membutuhkan tampilan data tabel yang telah diurutkan berdasarkan kolom tertentu. Pada modul sebelumnya, kita telah mempelajari fungsi `sort` dan `order`, namun fungsi tersebut tidak dapat digunakan pada tipe *data frame*. Untuk mengurutkan data pada tipe *data frame*, `dplyr` telah menyediakan fungsi `arrange`. Sebagai contoh, pada *script* selanjutnya, kita akan mengurutkan nama-nama negara bagian (*state*) berdasarkan jumlah populasinya:

```
murders %>%
arrange(population) %>%
head()
#> state abb region population total rate
#> 1 Wyoming WY West 563626 5 0.887
#> 2 District of Columbia DC South 601723 99 16.453
#> 3 Vermont VT Northeast 625741 2 0.320
#> 4 North Dakota ND North Central 672591 4 0.595
#> 5 Alaska AK West 710231 19 2.675
#> 6 South Dakota SD North Central 814180 8 0.983
```

Dengan menggunakan fungsi `arrange`, kita dapat menentukan argumen kolom mana yang akan digunakan sebagai dasar proses pengurutan *data frame*. Untuk menampilkan *state* berdasarkan populasi yang terkecil hingga terbesar, *script* yang digunakan adalah sebagai berikut:

```
murders %>%
arrange(rate) %>%
head()
#> state abb region population total rate
#> 1 Vermont VT Northeast 625741 2 0.320
#> 2 New Hampshire NH Northeast 1316470 5 0.380
#> 3 Hawaii HI West 1360301 7 0.515
#> 4 North Dakota ND North Central 672591 4 0.595
#> 5 Iowa IA North Central 3046355 21 0.689
#> 6 Idaho ID West 1567582 12 0.766
```

Secara *default*, fungsi `arrange` akan menampilkan hasil pengurutan dari nilai yang terkecil hingga terbesar. Dalam `dplyr`, terdapat pula fungsi `desc` yang dapat digunakan untuk mentransformasi vektor atau *data frame* sehingga urutan yang dihasilkan adalah dari nilai yang terbesar ke terkecil. Contohnya adalah sebagai berikut:

```
murders %>%
arrange(desc(rate))
```

1. *Nested sorting*

Penggunaan fungsi `arrange` juga dapat dilakukan pada lebih dari satu argumen. Pada contoh selanjutnya, kita akan mencoba menggunakan fungsi `arrange` pada dua argumen yang berbeda. Misalnya, kita akan mengurutkan data berdasarkan wilayah, kemudian data akan diurutkan berdasarkan besarnya tingkat pembunuhan yang terjadi '*rate*':

```
murders %>%
arrange(region, rate) %>%
head()
#>   state         abb region    population total rate
#> 1 Vermont       VT  Northeast 625741     2     0.320
#> 2 New Hampshire NH  Northeast 1316470    5     0.380
#> 3 Maine         ME  Northeast 1328361    11    0.828
#> 4 Rhode Island  RI  Northeast 1052567    16    1.520
#> 5 Massachusetts MA  Northeast 6547629    118   1.802
#> 6 New York      NY  Northeast 19378102   517   2.668
```

2. Top *n*

Dalam *script* di atas, kita telah menggunakan fungsi `head` untuk membatasi jumlah data yang ditampilkan pada hasil. Secara *default*, fungsi `head` akan membatasi jumlah hasil yang ditampilkan sebanyak 6 baris saja. Jika kita ingin menampilkan hasil dalam jumlah yang lebih besar, dapat digunakan fungsi `top_n`. Fungsi `top_n` membutuhkan *data frame* yang akan dievaluasi sebagai argumen pertama, jumlah baris yang ingin ditampilkan sebagai argumen yang kedua, dan variabel yang akan dijadikan acuan sebagai argumen yang ketiga. Pada contoh *script* selanjutnya, akan dilakukan *filter* 5 baris teratas yang memiliki nilai `rate` tertinggi:

```
murders %>% top_n(5, rate)
#>   state                abb region        population total rate
#> 1 District of Columbia DC  South         601723     99    16.45
#> 2 Louisiana            LA  South         4533372    351   7.74
#> 3 Maryland             MD  South         5773552    293   5.07
#> 4 Missouri             MO  North Central 5988927    321   5.36
#> 5 South Carolina       SC  South         4625364    207   4.48
```

Dengan menggunakan fungsi `top_n`, hasil baris yang ditampilkan tidak diurutkan berdasarkan argumen ke tiga (*rate*), hanya difilter. Sehingga, jika menginginkan hasil ditampilkan dalam bentuk yang telah diurutkan, perlu ditambahkan pula fungsi `arrange` sebelum argumen ketiga fungsi `top_n`. Secara *default*, jika argumen ketiga dibiarkan kosong, `top_n` akan melakukan *filter* data berdasarkan kolom terakhir *data frame*.

D. Latihan

Manipulasi *data frame*

1. Gunakan paket `dplyr` dan *dataset* "US *murders*".

```
library(dplyr)
library(dslabs)
data(murders)
```

Tambahkan kolom baru dengan nama '*rate*' menggunakan fungsi `mutate` pada paket `dplyr` seperti pada contoh kode di bawah ini.

```
rate <- mutate(murders, population_in_millions = population / 10^6)
```

2. `rank(x)` menghasilkan pemeringkatan '*x*' dari nilai terendah ke tertinggi. Gunakan fungsi `mutate` untuk menambahkan kolom baru yang berisi hasil pemeringkatan dari nilai tingkat pembunuhan tertinggi ke terendah.
3. Dengan `dplyr`, kita dapat menggunakan fungsi `select` untuk menampilkan kolom tertentu saja. Misalnya dengan contoh *script* ini, kita hanya akan menampilkan kolom *state* dan *population*:

```
select(murders, state, population) %>% head()
```

Gunakan `select` untuk menampilkan nama negara (*state*) dan singkatan (*abb*) dalam *dataset* "US *murders*".

4. Fungsi `filter` pada `dplyr` dapat digunakan untuk memilih baris tertentu dari *data frame* yang akan disimpan. Berbeda dengan `select` yang digunakan untuk memilih tampilan kolom, `filter` digunakan untuk memilih tampilan baris. Misalnya, kita ingin hanya menampilkan baris yang berisi dengan *state* = *New York* seperti contoh ini

```
filter(murders, state == "New York")
```

Gunakan filter untuk menampilkan 5 negara bagian teratas dengan tingkat pembunuhan tertinggi.

5. Buat script yang dapat menampilkan hasil sesuai kondisi berikut: seseorang ingin tinggal di regional *Northeast* atau *West* dan ingin calon tempat tinggal yang dipilih memiliki tingkat pembunuhan kurang dari 1.
   Gunakan `filter` untuk hanya menampilkan hasil yang terdiri dari: *state*, *rate*, dan peringkatnya.

Operator *pipe*

1. *Reset dataset "US murders"* ke tabel aslinya dengan melakukan *update* dengan perintah: `data(murders)`. Gunakan operator *pipe* untuk membuat *data frame* baru dengan nama '*my_states*' yang hanya berisi negara-negara di regional *Northeast* atau *Eastwest* yang memiliki tingkat pembunuhan kurang dari 1, dan hanya menampilkan kolom: state, *tingkat*, dan *rate*. *Script* yang dibuat seharusnya terdiri dari empat komponen yang dipisahkan oleh tiga %>%. Seperti contoh kerangka ini:

```
my_states <- murders %>%
mutate _____________ %>%
filter _____________ %>%
select _____________
```