MODUL 1
PENGENALAN R, RSTUDIO DAN DASAR R

A. Tujuan Praktikum

- Memahami fitur bahasa pemrograman R
- Memahami area kerja pada RStudio
- Mampu mendefinisikan fungsi dasar R

B. Alokasi Waktu

1 x pertemuan = 120 menit

C. Dasar Teori

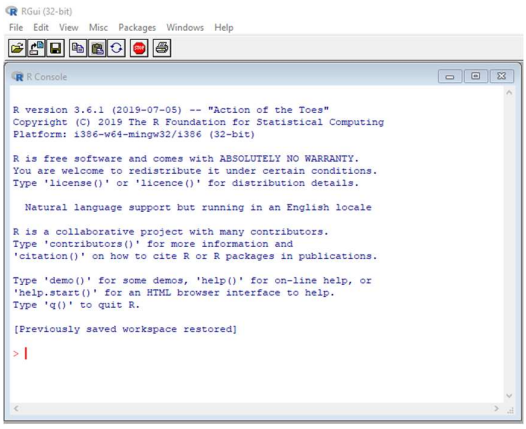
Tidak sama seperti seperti C atau Java, pada awalnya bahasa pemrograman R tidak dibuat oleh ahli perangkat lunak untuk pengembangan perangkat lunak. R dikembangkan oleh ahli statistik sebagai sarana yang interaktif untuk analisis data. Interaktivitas adalah fitur yang sangat diperlukan dalam *data science* karena kemampuan untuk mengeksplorasi data dengan cepat adalah salah satu tujuan utama dari *data science*. Dalam aplikasinya, R biasanya digunakan untuk proses analisis data dan, khususnya, visualisasi data.

Fitur menarik lainnya dari R adalah:

1. R gratis dan bersifat *open source*.
2. Dapat digunakan pada semua platform: Windows, Mac Os, UNIX / Linux.
3. *Scripts* dan *data objects* dapat dibagikan dan digunakan di seluruh platform.
4. Adanya komunitas pengguna R yang cukup besar, berkembang, dan aktif, sehingga ada banyak sumber untuk belajar dan mengajukan pertanyaan.
5. Kemudahan berbagi-pakai *add-ons* yang memungkinkan pengembang dan pengguna untuk berbagi implementasi metodologi *data science* terbaru. Sehingga seluruh pengguna memiliki akses terhadap metode terbaru dan alat-alat yang dikembangkan menggunakan bahasa R pada berbagai disiplin ilmu, seperti: ekologi, biologi molekuler, ilmu sosial, dan geografi, dan lain-lain.

R *Console*

Analisis data biasanya dilakukan pada R *console* yang akan mengeksekusi perintah secara langsung setelah diakhiri dengan tombol *enter*. Untuk mendapatkan akses ke R *console*, caranya adalah dengan memulai RGui. Tampilan R *console* terlihat seperti gambar dibawah ini:



Contoh yang paling mudah, silahkan coba gunakan R *console* untuk menghitung diskon 15% dari laptop yang harganya 7.500.000:

```
0.15 * 7500000
#> [1] 1125000
```

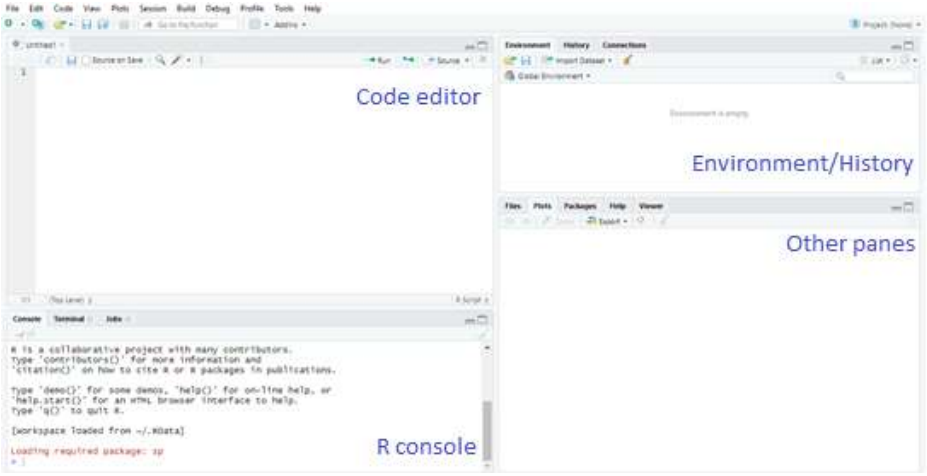
Sebagai catatan, pada modul ini, kotak abu-abu digunakan untuk menunjukkan R *script*. Simbol #> digunakan untuk menunjukkan *output* yang dihasilkan R *console*.

R *Scripts*

Selain menggunakan R *console*, sekumpulan kode atau pekerjaan dapat pula disimpan sebagai R *Scripts*. Dengan menggunakan R *Scripts*, penulisan kode atau *script* dapat diedit dan disimpan melalui editor teks. Pada modul ini, untuk meng-*compile script*, kita akan menggunakan *integrated development environment* (IDE) RStudio Versi 1.2.5019. RStudio memiliki fasilitas editor dengan berbagai fitur spesifik untuk pemrograman R, diantaranya R *console* untuk mengeksekusi kode, dan panel-panel lainnya, yang salah satunya dapat digunakan untuk menampilkan gambar (*plot*). Area kerja RStudio akan dibahas lebih detail pada bagian selanjutnya.

RStudio

RStudio akan menjadi area kerja yang digunakan untuk pembelajaran *data science* pada modul ini. RStudio tidak hanya menyediakan editor untuk membuat dan mengedit *script*, namun terdapat juga beberapa *tools* bermanfaat lainnya yang dapat digunakan dalam proses analisis data. Di bagian ini, akan dibahas beberapa fungsi-fungsi dan panel dasar yang dapat digunakan.



1. *Tool Panes*

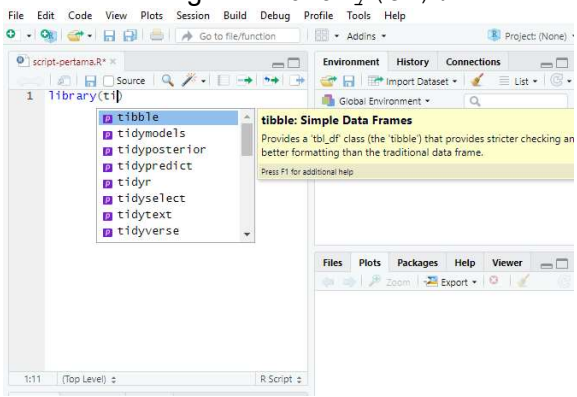
Saat pertama kali memulai RStudio, hanya akan muncul tiga panel. Panel kiri merupakan R *console*. Di sebelah kanan, panel atas terdiri dari tab: *Environment* dan *History*, sedangkan panel bawah memperlihatkan lima tab: *File*, *Plot*, *Packages*, *Help*, dan *Viewer*. Untuk memulai *script* baru, klik File - New File - R *script*. Sehingga, akan muncul panel baru di sebelah kiri yang dapat digunakan untuk menulis *script* seperti yang ditunjukkan pada gambar diatas.

2. Menjalankan Perintah saat Mengedit *Script*

Beberapa editor yang dibuat khusus untuk *coding* biasanya akan memvisualisasikan *script* menggunakan warna dan indentasi yang ditambahkan secara otomatis agar kode lebih mudah dibaca. RStudio termasuk salah satu editor yang menawarkan kemudahan tersebut, dan secara khusus dikembangkan untuk bahasa pemrograman R. Salah satu keuntungan utama yang diberikan oleh RStudio adalah adanya fitur untuk menjalankan *script* dengan mudah saat proses mengedit *script* sedang dilakukan. Untuk lebih mudah memahaminya, silahkan coba langkah berikut.

Langkah pertama, buat *script* baru seperti langkah yang telah dijelaskan sebelumnya. Selanjutnya beri nama *script*. Langkah kedua ini bisa dilakukan kapan saja, jika tidak ingin dilakukan saat ini (membuat *script* tanpa mendefinisikan nama *script* yang dibuat terlebih dahulu), *script* baru sementara akan bernama *Untitled*. Untuk memberi nama *script*, klik pada ikon *save* atau gunakan tombol Ctrl + S. Kemudian, simpan *script* pertama pada modul ini dengan nama "*script-pertama.R"*.

Setelah memberi nama lembar kerja *script*, kita dapat melakukan *editing* pada "*script-pertama.R"*. Baris pertama kode dalam R *script* didedikasikan untuk memuat *library* yang akan digunakan. Fitur RStudio lain yang bermanfaat adalah saat kita ketik `library(`, maka dapat dilakukan pelengkapan *script* otomatis, sesuai dengan *library* yang telah diinstal. Perhatikan apa terjadi ketika kita mengetik `library(ti)`:



Fitur lain yang mungkin dapat perhatikan adalah ketika mengetik `library(`, maka tanda kurung kedua akan ditambahkan secara otomatis.

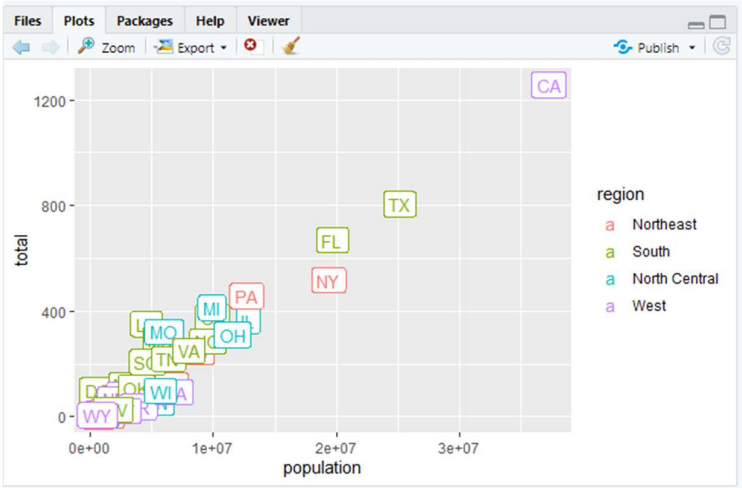
Selanjutnya, kita akan mencoba menampilkan plot pada panel kiri bawah. Sebagai contoh, kita akan membuat grafik yang menunjukkan pembunuhan total versus total populasi berdasarkan negara. *Script* yang digunakan adalah:

```
library(tidyverse)
library(dslabs)
data(murders)

murders %>%
  ggplot(aes(population, total, label = abb, color = region)) +
  geom_label()
```

Setelah selesai menulis *script* untuk menampilkan plot dari contoh data "US *murders"* yang terdapat pada `library(dslab)`, untuk mengeksekusi kode, klik tombol *Run* sisi kanan atas *code editor*. Sebagai alternatif, dapat pula menggunakan *key binding*: Ctrl + Shift + Enter untuk menjalankan *script* secara keseluruhan.

Untuk melihat tampilan hasil dari fungsi `ggplot`, klik Plot pada panel kanan bawah (area *Other Panes*). Perhatikan bahwa konsol plot akan memvisualisasikan data secara interaktif dalam bentuk plot. Beberapa fungsi lain yang bermanfaat pada konsol plot diantaranya: tombol panah untuk klik mundur dan maju antar plot yang telah dibuat, tombol *zoom* untuk memperbesar plot, dan *export* untuk menyimpan plot. Untuk menjalankan *script* per baris, dapat pula digunakan kombinasi *key binding*: Control + Enter.



3. Menginstal *R Packages*

Instalasi baru R hanya akan memiliki sebagian kecil paket dasar matematis. Fungsionalitas tambahan berasal dari *add-on* yang tersedia dari pengembang. Saat ini terdapat ratusan *library* tersedia di CRAN dan banyak repositori lain seperti GitHub. Namun, karena tidak semua orang membutuhkan semua fungsionalitas yang tersedia, R memberikan kemudahan dengan menyediakan komponen yang disebut *package*. Langkah menginstal *R Packages* melalui RStudio sangat mudah. Misalnya, untuk menginstal `dslabs` yang digunakan pada contoh *script* sebelumnya, maka dapat digunakan *script*:

```
install.packages("dslabs")
```

Cara lain untuk menambahkan *library* di RStudio, adalah dengan klik tab *Tools* dan memilih *install packages*, kemudian memasukkan paket apa yang ingin diinstal. Setelah proses instalasi berhasil, selanjutnya *library* dapat memuat *package* pada R *script* menggunakan fungsi:

```
library(dslabs)
```

Untuk menginstal lebih dari satu paket sekaligus, dapat digunakan *script* berikut:

```
install.packages (c ("tidyverse", "dslabs"))
```

Yang perlu diperhatikan adalah, dengan menginstal `tidyverse`, akan dilakukan pula instalasi terhadap beberapa paket lain. Hal ini terjadi ketika sebuah paket memiliki dependensi, atau menggunakan dan memiliki keterkaitan dengan fungsi dari paket lain. Saat memuat paket menggunakan *library*, maka akan dimuat pula dependensinya. Setelah paket diinstal, paket dapat dimuat kembali ke R pada *script* lain dan tidak perlu menginstalnya lagi, kecuali jika sebelumnya melakukan instalasi versi baru R. Paket *add-on* diinstal dalam R bukan RStudio. Untuk melihat semua paket yang telah diinstal, gunakan fungsi berikut:

```
installed.packages()
```

Dasar R

Sebelum memulai dengan analisis *dataset*, perlu dipahami terlebih dahulu mengenai dasar-dasar R.

1. Objek

Kita menggunakan istilah objek untuk mendefinisikan hal-hal yang disimpan dalam R. Sebagai contoh: Variabel. Tetapi objek juga bisa menjadi entitas yang lebih rumit seperti fungsi.

Untuk menetapkan nilai ke variabel, dapat digunakan tanda (`<-`). Selain itu, dapat pula menggunakan tanda (=). Sebagai contoh, ketik kode berikut pada R *console* untuk mendefinisikan tiga variabel dan nilainya.

```
a <- 1
b <- 1
c <- -1
```

Perhatikan bahwa R tidak mencetak hasil apa pun saat *script* dijalankan. Untuk melihat nilai yang disimpan oleh variabel, kita cukup meminta R untuk mengevaluasi nilai *'a'* dan menampilkan nilai yang disimpan

```
a
#> [1] 1
```

Cara yang lebih eksplisit untuk meminta R menampilkan nilai yang disimpan dalam *'a'* adalah dengan menggunakan fungsi `print` seperti ini:

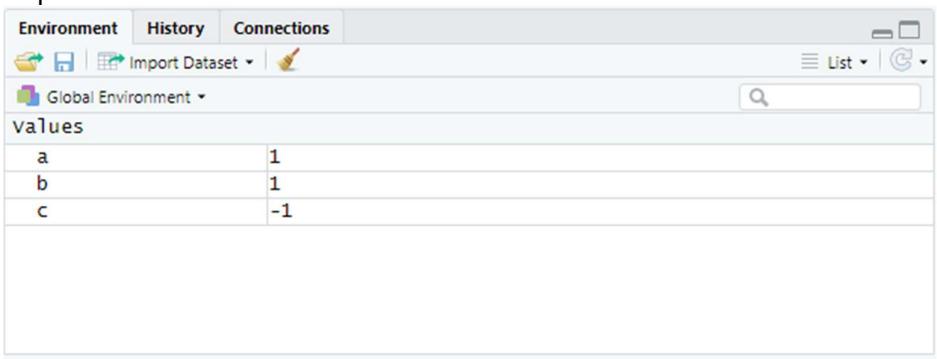
```
print(a)
#> [1] 1
```

2. *Workspace*

Saat kita mendefinisikan objek di R *console*, kita sebenarnya juga telah melakukan perubahan terhadap *workspace*. Untuk melihat semua variabel yang disimpan di *workspace*, dapat dilakukan dengan perintah:

```
ls()
#> [1] "a" "b" "c"
```

Di RStudio, tab *environment* pada panel kanan atas akan menunjukkan nilai-nilai yang tersimpan pada suatu sesi R:



Jika kita mencoba mencari nilai variabel yang tidak ada di *workspace*, maka kita akan menerima pesan kesalahan. Misalnya, jika kita mengetik *'x'*, maka akan muncul pesan berikut: "*Error: object 'x' not found*".

Karena kita telah memiliki nilai-nilai variabel *'a'*, *'b'*, dan *'c'* yang telah disimpan dalam variabel, Kita dapat melakukan komputasi untuk menyelesaikan suatu persamaan kuadratik seperti contoh berikut:

$$\frac{-b - \sqrt{b^2 - 4ac}}{2a} \quad \text{and} \quad \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

dengan menuliskan *script* seperti dibawah ini:

```
(-b + sqrt(b^2 - 4*a*c) ) / ( 2*a )
#> [1] 0.618034

(-b - sqrt(b^2 - 4*a*c) ) / ( 2*a )
#> [1] -1.618034
```

3. Fungsi

Proses analisis data biasanya dapat digambarkan sebagai serangkaian fungsi yang diterapkan pada data. R telah memiliki beberapa fungsi dasar dan sebagian besar analisis yang akan dilakukan juga akan memanfaatkan penggunaan berbagai macam fungsi dalam satu *script*. Contoh fungsi dasar yang dimaksud diantaranya: fungsi `install.packages`, `library`, dan `ls` yang telah kita coba gunakan sebelumnya. Selain itu, kita juga telah menggunakan fungsi dasar R, yaitu `sqrt` untuk menyelesaikan persamaan kuadrat di atas. Ada banyak fungsi *prebuilt* yag telah dimiliki oleh R dan banyak fungsi lainnya dapat ditambahkan melalui paket.

Secara umum, kita perlu menggunakan tanda kurung untuk mengevaluasi suatu fungsi. Jika kita mengetik `ls`, fungsi tersebut tidak akan menampilkan hasil evaluasi berupa *list* variabel pada *workspace*, sebaliknya R akan menunjukkan *script* yang mendefinisikan fungsi itu sendiri. Tidak seperti `ls`, sebagian besar fungsi memerlukan satu atau lebih argumen. Di bawah ini adalah contoh bagaimana kita menetapkan objek ke argument fungsi `log`. Yang perlu diingat adalah bahwa kita telah menetapkan sebelumnya bahwa nilai variabel *'a'* adalah 1:

```
log (8)
#> [1] 2.08

log (a)
#> [1] 0
```

Untuk mengetahui argumen yang dibutuhkan oleh fungsi dan kegunaan fungsi, dapat digunakan fungsi `help` yang sangat berguna pada R. Dengan fungsi tersebut, kita dapat memperoleh bantuan jika kurang yakin dengan kegunaan suatu fungsi tertentu, atau jika saat implementasi ternyata masih ditemukan pesan *error* sehingga *script* tidak berhasil dijalankan:

```
help(log)
```

Untuk sebagian besar fungsi, sebagai alternatif, dapat digunakan pula langkah ini:

```
?(log)
```

Halaman bantuan akan menunjukkan argumen apa yang dibutuhkan oleh fungsi. Sebagai contoh, `log` membutuhkan *'x'* dan *basis*. Pada fungsi tertentu, yang perlu diperhatikan adalah beberapa argumen wajib ada saat fungsi didefinisikan, dan ada pula argumen lainnya yang sifatnya opsional.

Argumen opsional berisi nilai *default* yang ditandai dengan (=) dalam dokumen `help`. Sebagai contoh, argumen *base* dari fungsi `log` secara *default* bernilai = `exp(1)`. Jika ingin melihat argumen tanpa membuka dokumen `help`, dapat digunakan fungsi `args`:

```
args (log)
#> function (x, base = exp (1))
#> NULL
```

Nilai *default* dapat diubah dengan mendefinisikan nilai objek seperti contoh berikut:

```
log (8, base = 2)
#> [1] 3
```

Sebagai alternatif, argumen *x* dapat pula didefinisikan sebagai berikut:

```
log (x = 8, base = 2)
#> [1] 3
```

Kedua potongan kode di atas akan menghasilkan nilai *output* yang sama, tetapi untuk menghindari beberapa kesalahan pengetikan atau pendefinisian argumen: jika tidak ada nama argumen yang didefinisikan, R mengasumsikan bahwa argumen telah dimasukkan dalam urutan yang ditunjukkan pada dokumen `help` atau `args`. Sehingga, dengan tidak mendefinisikan nama variabel, diasumsikan fungsi `log` berisi argumen *x* diikuti oleh basis:

```
log (8,2)
#> [1] 3
```

Jika nama argumen dedifinisikan, maka argumen fungsi dapat dimasukkan dalam urutan yang bebas:

```
log (basis = 2, x = 8)
#> [1] 3
```

Untuk menentukan argumen, kita harus menggunakan (=), dan tidak dapat menggunakan <- .

Terdapat beberapa pengecualian pada aturan bahwa tanda kurung dibutuhkan untuk mengeksekusi suatu fungsi. Diantaranya, yang paling umum digunakan adalah operator aritmatika dan relasional. Sebagai contoh:

```
2 ^ 3
#> [1] 8
```

Untuk melihat bantuan mengenai operator aritmatika dapat dilakukan dengan:

```
help("+")
```

atau

```
? "+"
```

Sedangkan untuk operator relasional dapat dilakukan dengan:

```
help(">")
```

atau

```
? ">"
```

4. *Prebuilt Objects*

Terdapat beberapa *dataset* yang telah disediakan R untuk *user* agar memberikan kemudahan dalam berlatih dan menguji fungsi yang diinginkan. Untuk melihat semua *dataset* contoh yang tersedia dapat digunakan:

```
data()
```

Hasil dari perintah diatas adalah berupa tampilan nama-nama *dataset* yang tersedia dalam paket R. Kumpulan data tersebut merupakan suatu objek yang bisa dipanggil dengan hanya mendefinisikan nama *dataset* yang diinginkan. Misalnya:

```
CO2
```

Hasil dari perintah diatas akan menampilkan data penyerapan karbon dioksida pada tanaman rumput.

Objek *prebuilt* lain diantaranya berupa atribut-atribut matematis, seperti konstanta π dan ∞:

```
pi
#> [1] 3.14
Inf+1
#> [1] Inf
```

5. *Comment* pada *Script*

Jika suatu baris R *script* dimulai dengan simbol #, maka baris tersebut tidak akan dievaluasi. Kita dapat menggunakan # untuk menulis *comment* yang berisi definisi dari kumpulan baris yang akan dieksekusi. Misalnya, dengan menggunakan *script* sebelumnya kita akan menambahkan *comment* sebagai berikut:

```
## Script untuk menghitung persamaan kuadrat
## Variabel yang digunakan
a <- 3
b <- 2
c <- -1
## Persamaan kuadrat yang dihitung
(-b + sqrt(b^2 - 4*a*c)) / (2*a)
(-b - sqrt(b^2 - 4*a*c)) / (2*a)
```

D. Latihan

1. Berapa jumlah dari 100 bilangan bulat positif pertama?
   Rumus untuk jumlah bilangan bulat 1 sampai *n* adalah `n (n + 1) / 2`. Tentukan n = 100 dan kemudian gunakan R untuk menghitung jumlah 1 hingga 100 menggunakan rumus.
2. Gunakan rumus yang sama untuk menghitung jumlah bilangan bulat dari 1 hingga 1.000.
3. Lihatlah hasil potongan kode berikut dalam R:
   a. n <- 1000
   b. x <- seq (1, n)
   c. jumlah (x)
4. Coba implementasikan dalam potongan kode beberapa perintah dibawah ini, dan jelaskan hasil percobaan yang dilakukan. Sebagai panduan, dapat digunakan fungsi `help`.
   a. `seq` membuat daftar angka dan `sum` menjumlahkannya.
   b. `seq` membuat daftar angka acak dan `sum` menghitung jumlah angka ke-1 hingga 1.000.
5. Dalam matematika dan pemrograman, kita dapat mendefinisikan bahwa kita akan mengevaluasi suatu fungsi ketika nilai argumen telah diisi. Sehingga, saat kita menjalankan perintah `sqrt(4)`, secara otomatis fungsi `sqrt` akan dievaluasi dengan nilai argumen = 4. Pada R, kita juga dapat mengevaluasi fungsi di dalam fungsi lain. Evaluasi akan dilakukan dari dalam ke luar.
   Gunakan satu baris kode untuk menghitung `log` dengan nilai basis 10, pada akar kuadrat dari 100.
6. Manakah dari contoh berikut ini yang akan selalu mengembalikan nilai numerik yang disimpan dalam variabel *x*?
   a. `log (10 ^ x)`
   b. `log10 (x ^ 10)`
   c. `log (exp (x))`
      `exp (log (x, base = 2))`