DATA MINING

Konsep dan Aplikasi Menggunakan R

Sri Astuti Thamrin
Masjidil Aqsha

“Data will talk to you if you’re willing to listen”

Jim Bergeson

DAFTAR ISI

BAB 1 | PENDAHULUAN

Buku ini memperkenalkan penggunaan R untuk data mining. Dalam buku ini disajikan banyak contoh dari berbagai fungsi data mining di R. Audiens yang diharapkan dari buku ini adalah mahasiswa, peneliti, penambang data, dan ilmuwan data yang tertarik menggunakan R untuk melakukan penelitian dan proyek data mining. Asumsinya bahwa audiens telah memiliki ide dasar tentang data mining dan juga memiliki beberapa pengalaman dasar dengan R. Melalui buku ini diharapkan akan mendorong semakin banyak orang menggunakan R untuk melakukan pekerjaan data mining dalam penelitian dan aplikasinya.

Bab ini memperkenalkan konsep dan teknik dasar untuk data mining, termasuk proses data mining dan teknik data mining yang populer. Dalam Bab ini juga disajikan R dan *package*, fungsi, dan tampilan tugasnya untuk data mining. Selain itu, beberapa dataset yang digunakan dalam buku ini dijelaskan.

1.1 Data Mining

Data mining (penambangan data) adalah topik yang sangat populer saat ini. Tidak seperti beberapa tahun yang lalu, sekarang semuanya terikat dengan data dan jenis data besar ini mampu ditangani dengan baik. Dengan mengumpulkan dan memeriksa data ini, orang dapat menemukan beberapa pola. Meskipun seluruh kumpulan data adalah sampah (*junk*), ada beberapa pola tersembunyi yang dapat diekstraksi dengan menggabungkan beberapa sumber data untuk memberikan wawasan yang berharga. Ini disebut sebagai Data Mining.

Data mining adalah proses untuk menemukan pengetahuan yang menarik dari sejumlah besar data (Han dan Kamber, 2000). Data mining adalah bidang interdisipliner dengan kontribusi dari banyak bidang, seperti statistik, pembelajaran mesin, pengambilan informasi, pengenalan pola, dan bioinformatika. Data mining banyak digunakan di banyak domain, seperti ritel, keuangan, telekomunikasi, dan media sosial.

Data mining sering kali digabungkan dengan berbagai sumber data termasuk data perusahaan yang diamankan oleh organisasi dan memiliki masalah

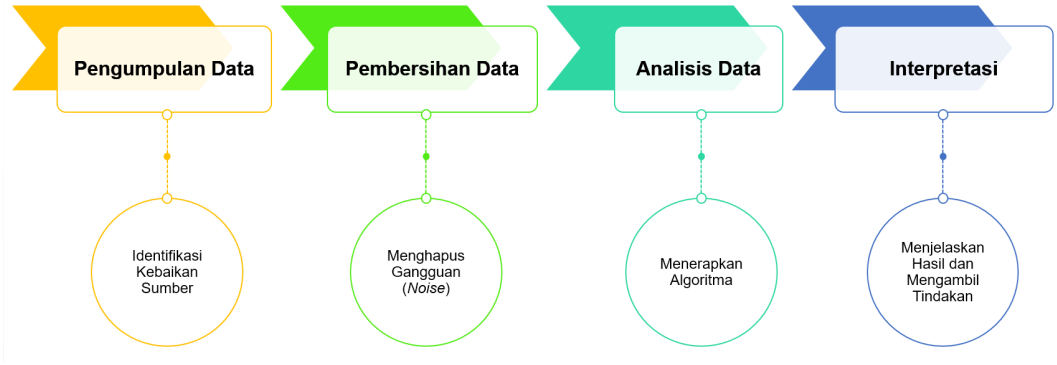
privasi dan terkadang beberapa sumber terintegrasi termasuk data pihak ketiga, demografi pelanggan dan data keuangan, dan lain-lain (Zhou, 2015). Jumlah data yang tersedia merupakan faktor penting disini. Mengapa ini penting? Karena dari jumlah data yang tersedia itu akan ditemukan pola dalam data sekuensial atau non-sekuensial, korelasi, untuk menentukan apakah jumlah data yang diperoleh berkualitas baik, sebanyak data yang tersedia juga baik.

Mari memulai dengan sebuah contoh. Asumsikan kita mendapat beberapa data yang terkait dengan *log* masuk untuk aplikasi web. Secara keseluruhan, kumpulan data ini tidak memiliki nilai. Data ini mungkin berisi nama pengguna, stempel waktu masuk, waktu yang dihabiskan untuk keluar, aktivitas telah dilakukan, dan lain-lain. Jika kita melihat sekilas tentang data ini, ini benar-benar berantakan. Tetapi data ini masih dapat dianalisis untuk mengekstrak beberapa informasi berguna. Misalnya, data ini dapat digunakan untuk mengetahui kebiasaan rutin pengguna tertentu. Selanjutnya, data ini akan membantu untuk mengetahui jam sibuk sistem. Informasi yang diekstraksi ini dapat digunakan untuk meningkatkan efisiensi sistem dan membuatnya lebih ramah bagi pengguna. Namun, data mining bukanlah tugas yang mudah. Proses data mining ini membutuhkan waktu dan prosedur khusus juga.

1.1.1 Langkah-Langkah Dasar Data Mining

Langkah-langkah dasar dari data mining adalah sebagai berikut

1. Pengumpulan data
2. Pembersihan data
3. Analisis data
4. Interpretasi



Gambar 1.1 Langkah-Langkah Dasar dari Data Mining

1. **Pengumpulan data:** Langkah pertama adalah mengumpulkan beberapa data. Semakin banyak informasi yang dimiliki akan semakin bagus untuk mempermudah analisis nantinya. Selain itu, harus pula dipastikan bahwa sumber datanya dapat diandalkan.
2. **Pembersihan data:** Karena data yang didapatkan dalam jumlah besar, perlu dipastikan bahwa hanya disimpan data yang diperlukan dan menghapus yang tidak diinginkan. Jika tidak, kesimpulan yang didapatkan bisa salah.
3. **Analisis Data:** Seperti namanya, analisis dan pola pencarian dilakukan pada langkah ini.
4. **Interpretasi:** Akhirnya data yang telah dianalisis kemudian diinterpretasikan untuk mengambil kesimpulan penting seperti prediksi.

1.1.2 Model Data Mining

Ada berbagai jenis model yang terkait dengan data mining:

1. Pemodelan deskriptif
2. Pemodelan prediktif
3. Pemodelan preskriptif

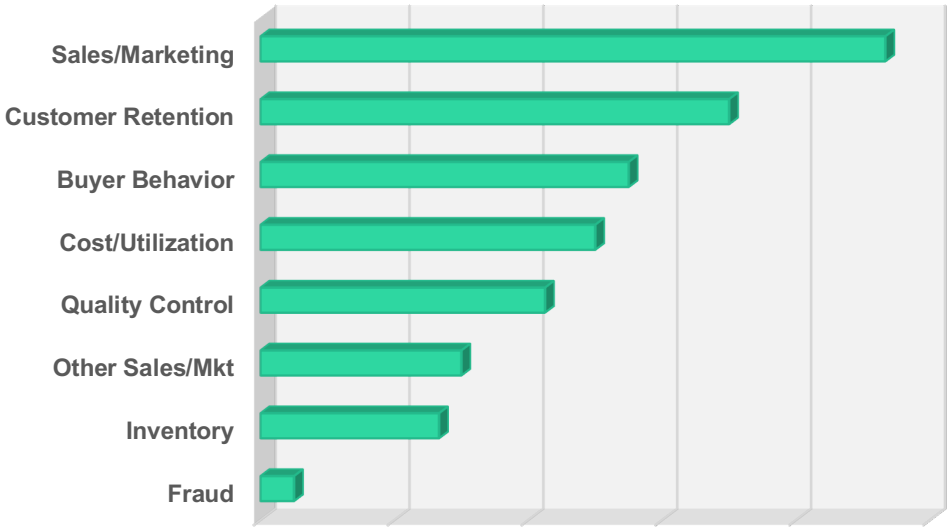
Dalam Pemodelan Deskriptif, hal yang dilakukan adalah mendeteksi kesamaan antara data yang dikumpulkan dan alasan di baliknya. Ini sangat penting dalam membangun kesimpulan akhir dari kumpulan data.

Pemodelan Prediktif digunakan untuk menganalisis data masa lalu dan memprediksi perilaku masa depan. Data masa lalu memberikan semacam petunjuk tentang masa depan.

Dengan perkembangan web yang signifikan, penambangan teks telah ditambahkan sebagai disiplin terkait dengan data mining. Dalam hal ini, data mining diperlukan untuk mengolah, memfilter, dan menganalisis data dengan benar untuk membuat model prediksi tersebut.

1.1.3 Aplikasi Data Mining

Data mining berguna dalam banyak hal. Hal ini dapat dilihat pada Gambar 2. Untuk pemasaran, dapat diterapkan secara efektif. Dengan menggunakan data mining, perilaku pelanggan dapat dianalisis dan periklanan dengan lebih dekat dengan pelanggan dapat juga dilakukan. Ini akan membantu untuk mengidentifikasi *trend* pelanggan untuk barang misal di pasar dan memungkinkan pengecer untuk memahami perilaku pembelian pembeli.



Gambar 1.2 Aplikasi Data Mining

Dalam domain pendidikan, perilaku belajar siswa dapat diidentifikasi dan lembaga pembelajaran dapat meningkatkan modul dan kursus mereka sesuai dengan itu. Data mining dapat juga digunakan untuk mengatasi bencana alam. Jika

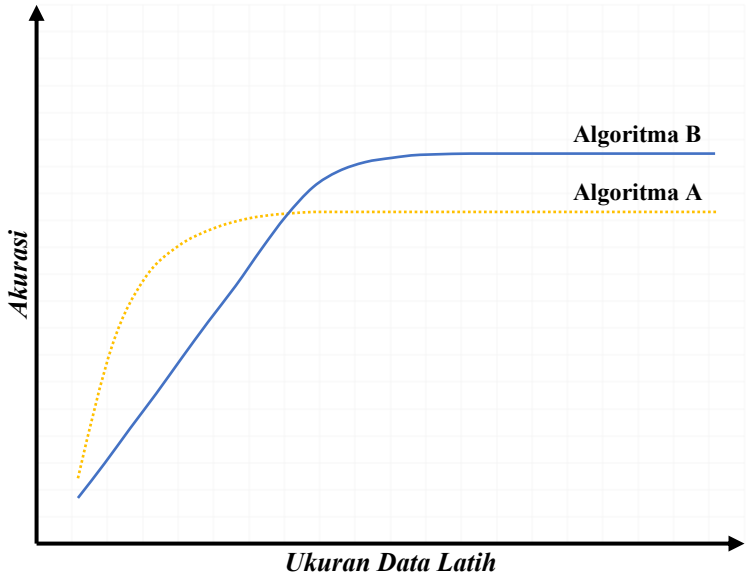
beberapa informasi dapat dikumpulkan, maka informasi itu dapat digunakan untuk memprediksi hal-hal seperti tanah longsor, curah hujan, tsunami, dan lain-lain.

Ada lbanyak aplikasi dalam data mining saat ini. Aplikasinya dapat bervariasi dari hal-hal yang sangat sederhana seperti pemasaran hingga domain yang sangat kompleks seperti membuat prediksi bencana lingkungan, dan lain-lain.

1.1.4 Hal-hal Khusus Terkait Data Mining

Data mining tidak boleh digunakan ketika solusi yang lengkap dan akurat untuk masalah tertentu dimungkinkan. Ketika solusi seperti itu tidak memungkinkan, teknik data mining dengan banyak data dapat digunakan untuk mengkarakterisasi masalah sebagai hubungan masukan-keluaran (*input-output*).

Perlu menganalisis properti masalah untuk menentukan apakah itu masalah klasifikasi (keluaran diskrit, misal: benar atau salah) atau estimasi (keluaran kontinu misalkan: bilangan real antara (0,1)). Masukan harus memiliki informasi yang cukup untuk menghasilkan keluaran yang akurat. Jika tidak, itu akan menyebabkan penurunan yang tak terhindarkan. Harus ada data yang cukup untuk membuat hasil yang akurat. Perlu memilih algoritma yang tepat sesuai dengan data masukan. Beberapa algoritma membutuhkan data dalam jumlah besar untuk mencapai akurasi yang baik, sementara algoritma lainnya dapat menjangkau dengan cepat (Gambar 3).



Gambar 1.3 Perbandingan Algoritma

Teknik utama data mining meliputi klasifikasi dan prediksi, pengelompokan (*clustering*), deteksi outlier, aturan asosiasi (*association rules*), analisis urutan (*sequence analysis*), analisis deret waktu (*time series analysis*) dan penambangan teks (*text mining*), dan juga beberapa teknik baru seperti analisis jaringan sosial (*social networking analysis*) dan analisis sentimen (*sentiment analysis*). Pengenalan rinci dari teknik data mining dapat ditemukan dalam buku teks tentang data mining (Han and Kamber, 2000, Hand dkk., 2001, Witten and Frank, 2005). Dalam aplikasi dunia nyata, proses data mining dapat dibagi menjadi enam fase utama: pemahaman bisnis, pemahaman data, persiapan data, pemodelan, evaluasi, dan penerapan, sebagaimana didefinisikan oleh CRISP-DM (Proses Standar Lintas Industri untuk Data mining) (Adler dkk., 2015). Buku ini berfokus pada tahap pemodelan, dengan eksplorasi data dan evaluasi model yang disajikan dalam beberapa bab.

1.2 Program R

Saat ini R sudah dikenal luas sebagai salah satu *powerful software* untuk analisis data dan *Data Science*. Selain R masih banyak *software* lain yang juga sering digunakan untuk analisis data, misalnya Python. R dibuat dengan tujuan awal untuk komputasi statistika dan grafis. Awalnya digunakan oleh para ilmuwan dalam

riset mereka dan para akademisi. Namun seiring perkembangan teknologi, cakupan kemampuan R sebagai bahasa pemrograman menjadi jauh lebih luas. Karena R didesain untuk analisis data dan perkembangan serta kemampuannya mencakup hampir semua lini dalam analisis data, tidak heran saat ini banyak analis data dan ilmuwan data (*data scientist*) menggunakan R untuk menyelesaikan berbagai masalah mereka (R Core Team, 2020). Berikut ini dijelaskan beberapa kemampuan R.

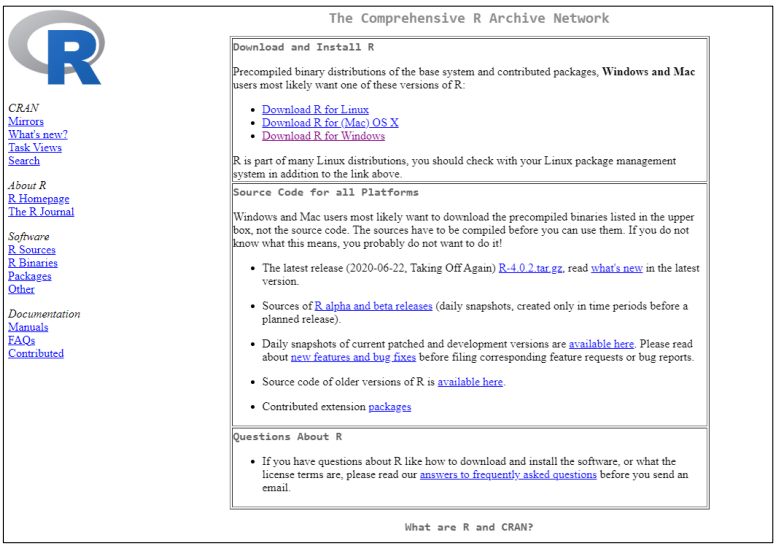
1. **Gratis dan *Open Source*:** Merujuk kepada *opensource.com*, istilah *open source* merujuk kepada sesuatu yang bisa dimodifikasi dan dibagikan. *Open Source Software* (OSS) sendiri berarti *software* yang *source code*-nya dapat diperiksa, dimodifikasi, ditambahkan dan dibagikan oleh siapapun.
2. **Tersedia banyak *package*:** Karena R adalah open source software, hampir semua package yang ada pun dapat digunakan secara bebas. Package adalah kumpulan suatu script yang umumnya berupa *function* atau data yang dapat digunakan untuk kebutuhan tertentu.
3. **Mudah dalam melakukan transformasi dan pemrosesan data:** Karena R adalah program untuk analisis data, maka kemampuan R dalam transformasi data seperti penyiapan data, *import* dan *export* data dalam berbagai format, dan lain-lain.
4. **Mampu menghasilkan grafik yang sangat bagus:** Salah satu keunggulan yang dimiliki oleh R adalah kemampuannya untuk menghasilkan grafik yang sangat bagus. Salah satu yang diunggulkan adalah package {*ggplot2*}. Tentu saja masih banyak package untuk visualisasi selain {*ggplot2*}.

Dan masih banyak lagi kemampuan R yang dapat dimanfaatkan untuk mendukung dan memudahkan pekerjaan pengguna dalam hal analisis data ataupun *data science*.

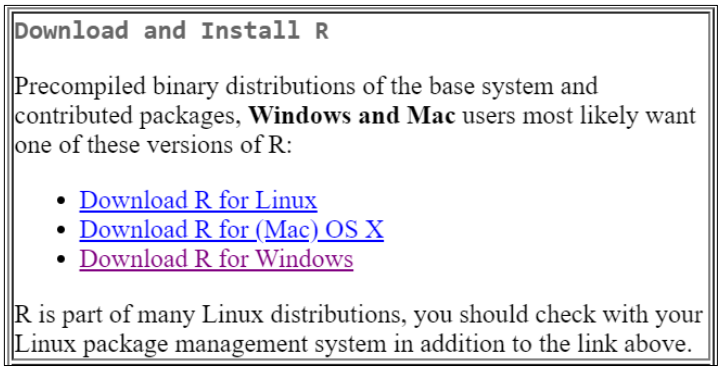
1.2.1 Download dan Install R

Pada *Personal Computer* (PC) dengan *Operating System* (OS) *windows* dapat melakukan langkah-langkah beriku ini:

1. Buka laman **https://cran.r-project.org**

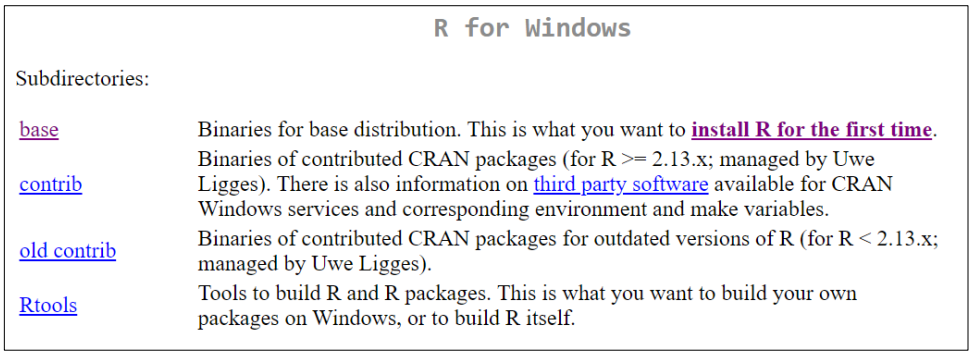


2. Pilih *Download R for Windows*.

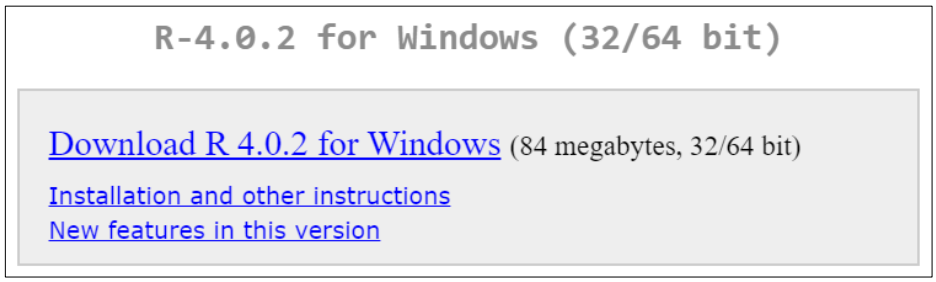


Pengguna OS lain dapat mengikuti petunjuk dengan membuka tautan yang sesuai.

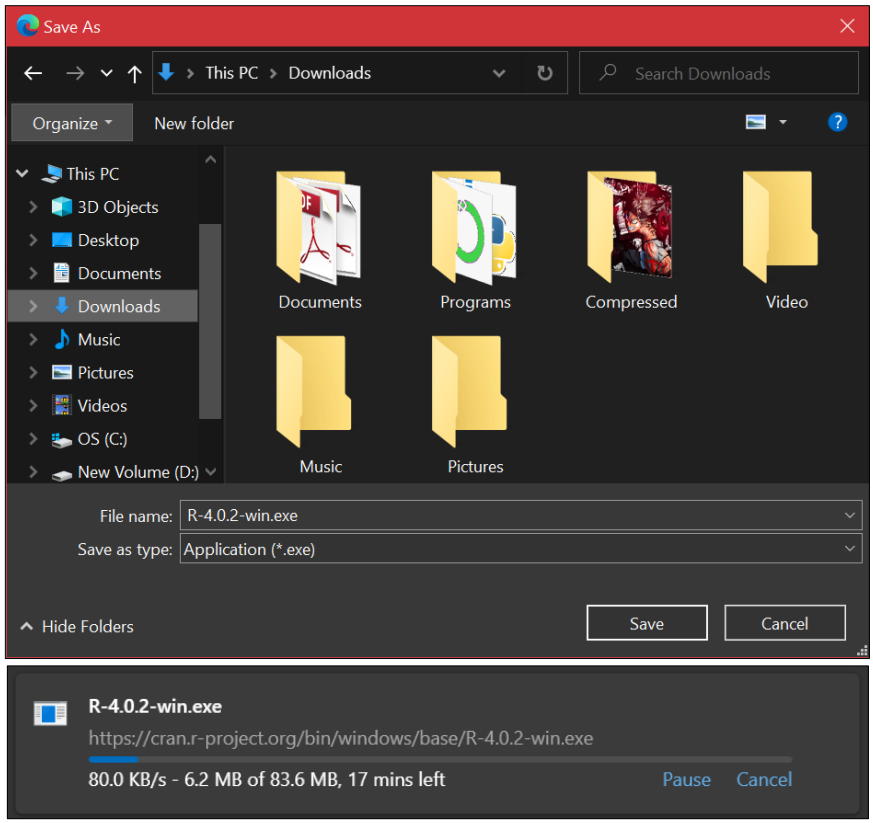
3. Klik *Install R for the first time*.



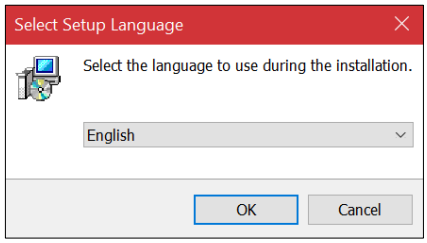
4. Klik *Download R 4.0.2 for Windows* (*R 4.0.2* adalah versi terbaru saat ini).



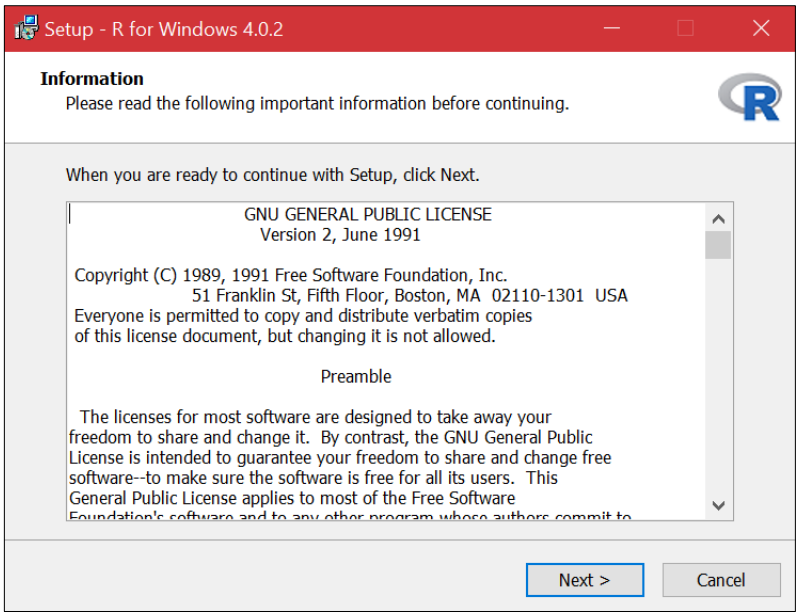
5. Simpan *file installer* dan tunggu hingga proses download selesai.



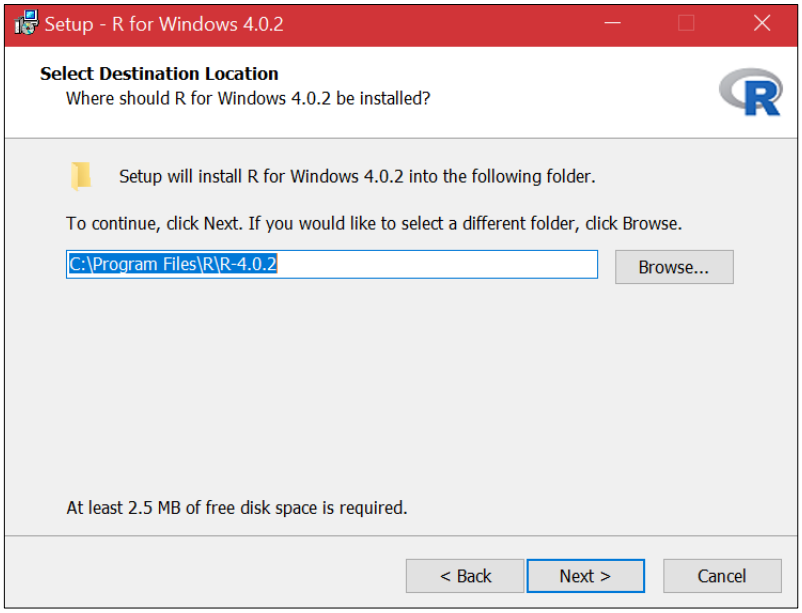
6. Setelah download selesai, jalankan *file R 4.0.2* tersebut dan pilih *English* untuk bahasanya. Klik OK.



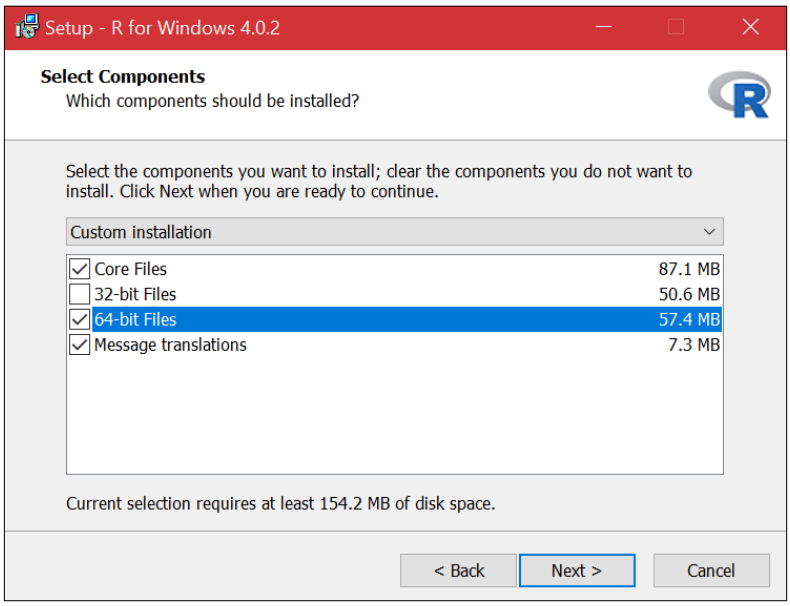
7. Silahkan baca informasi penting bila perlu lalu klik *next*.



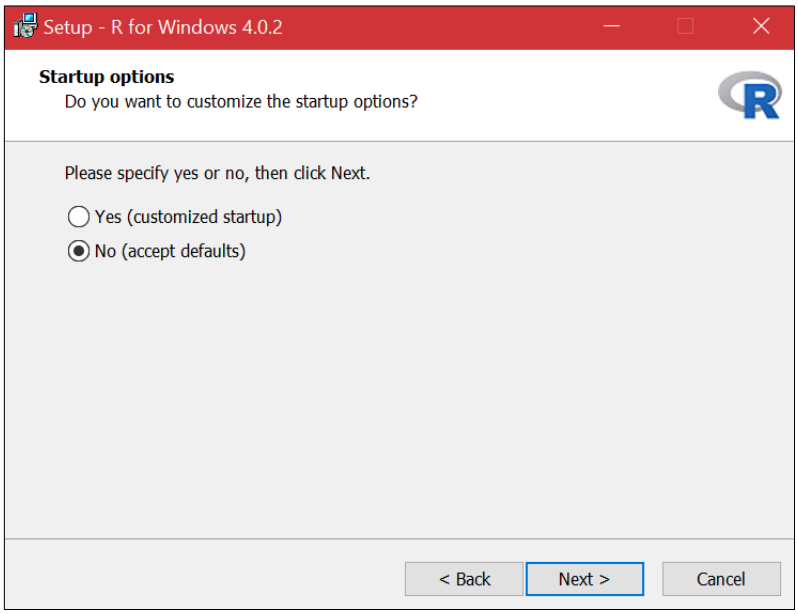
8. Pilih lokasi atau tempat penyimpanan untuk *R* sesuai dengan keinginan pengguna lalu klik *next*.



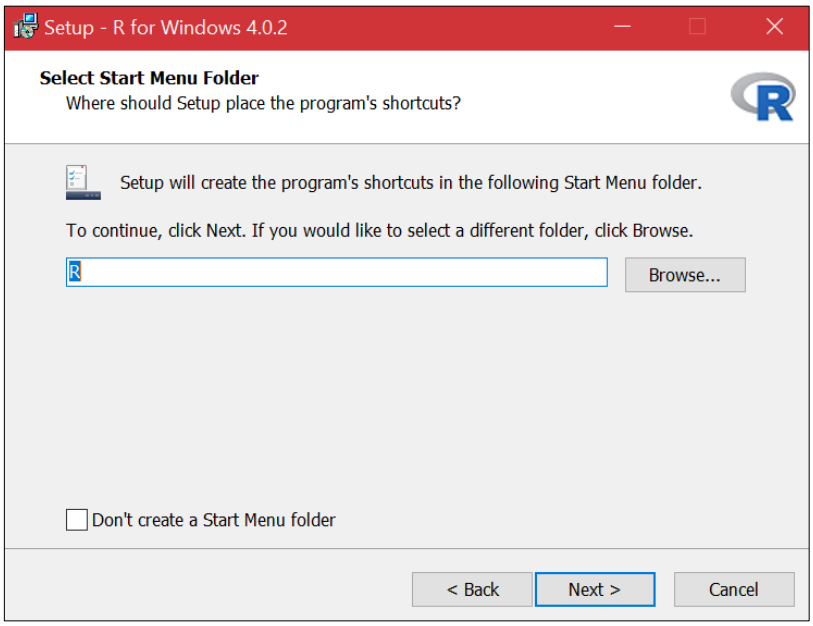
9. Ceklis pada *Core Files*, *Message translation* dan pilih antara *32-bit Files* atau *64-bit Files* sesuai dengan *System Type* pada PC yang digunakan. Klik *Next*.



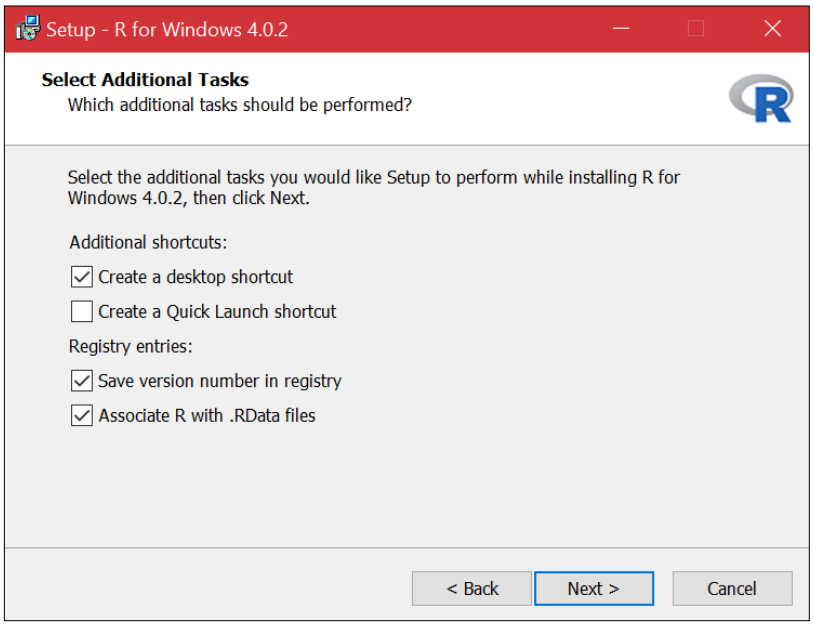
10. Pilih *Yes* untuk menyesuaikan pengaturan awal dan *No* untuk menggunakan pengaturan standar. Klik *Next*.



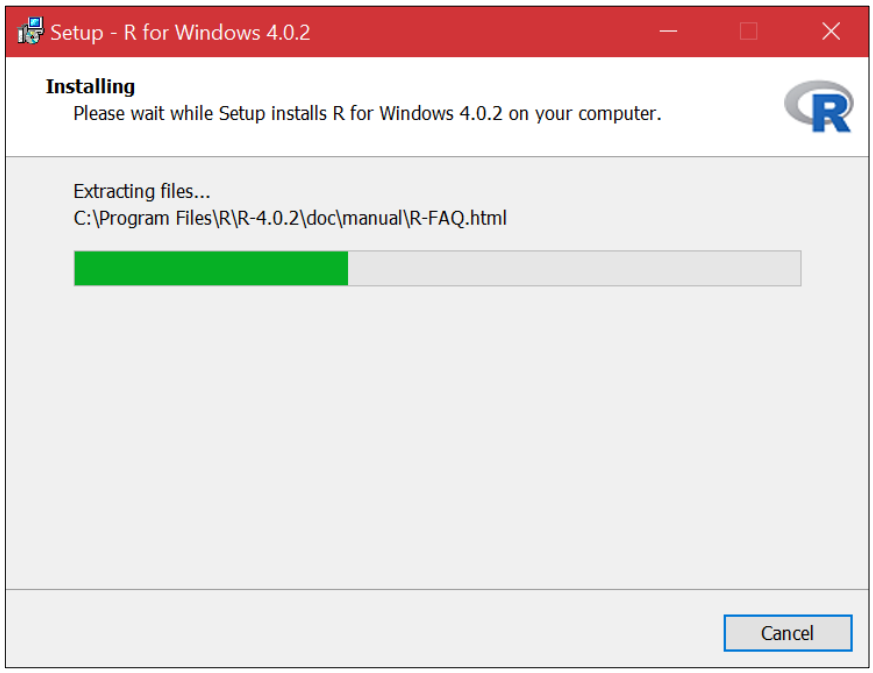
11. Masukkan nama *folder* yang akan digunakan pada *Start Menu* atau ceklis pada *Don't create a Start Menu folder* apabila pengguna tidak ingin melakukan pembuatan *folder* pada *Start Menu*. Klik *Next*.



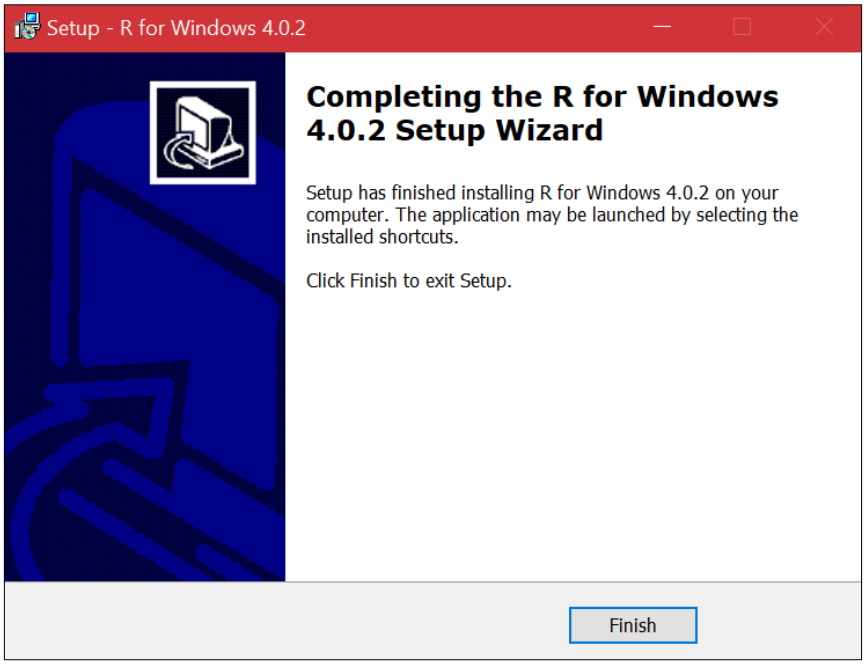
12. Ceklis pada *Create a dekstop shortcuts* apabila pengguna ingin membuat pintasan pada *dekstop* lalu klik *Next*.



13. Tunggu hingga proses instalasi selesai.



14. Klik *Finish* untuk mengakhiri proses instalasi.



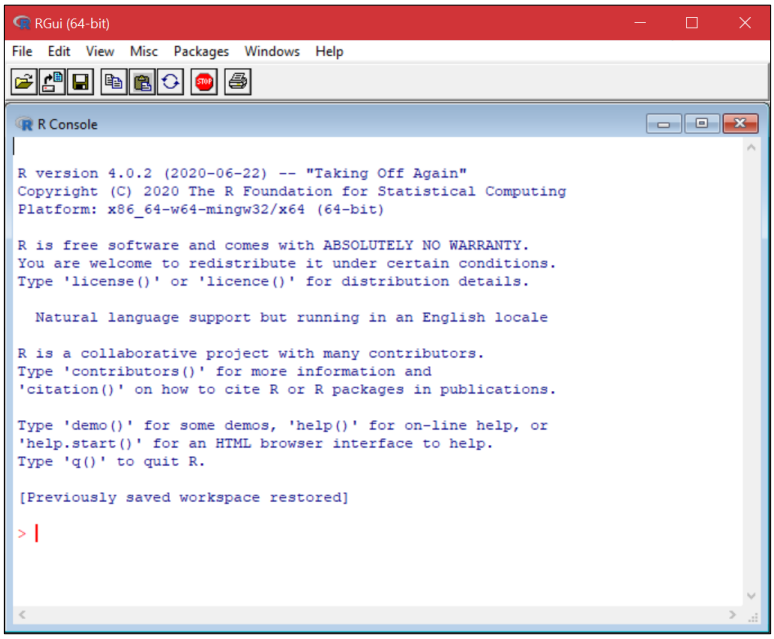
1.2.2 Menjalankan R

Setelah selesai *install*, pengguna bisa membuka *R GUI* yang ada pada *dekstop* atau *Start Menu* pengguna apabila salah satunya dibuat pada saat proses instalasi.

(a) *Icon R* pada *Desktop*

(b) *Icon R* pada *Start Menu*

Berikut ini adalah tampilan awal ketika pengguna membuka program *R*.

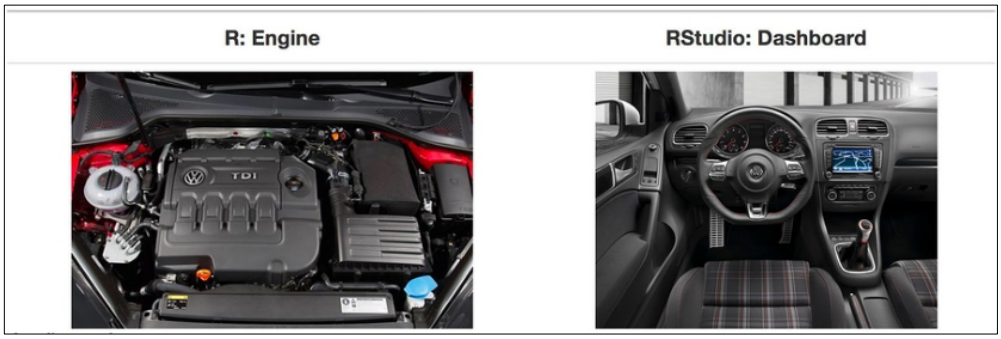


Pengguna sudah bisa menggunakan *R* melalui *R GUI*. Namun, perlu diketahui bahwa untuk penggunaan yang lebih nyaman, mudah dan efisien, pengguna bisa menggunakan *Integrated Development Environtment* (IDE). IDE untuk *R* yang sangat sering digunakan saat ini adalah *Rstudio*.

1.3 RStudio

Sebelum membahas lebih lanjut tentang *R*, sebaiknya pengguna mendownload *RStudio* dan *install* terlebih dahulu. *RStudio* adalah *Integrated Development Environment* (IDE) untuk *R* yang banyak digunakan hingga saat ini.

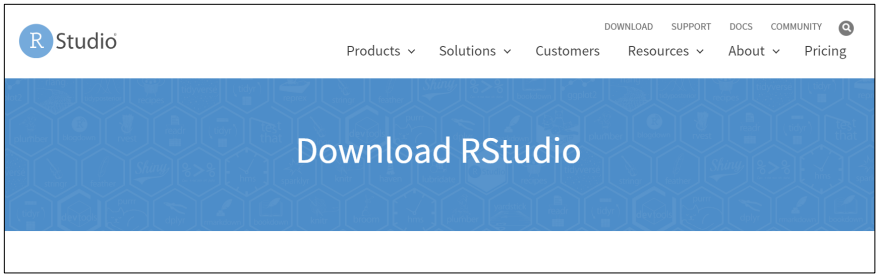
Dapat dikatakan bahwa hampir semua pengguna *R* yang sudah mengetahui *RStudio* akan lebih memilih menggunakan *R* melalui *RStudio* dibandingkan dengan menggunakan *R GUI*.



Sangat disarankan untuk menggunakan *RStudio* dan *R* versi terbaru. *R* dan *RStudio* adalah dua program yang berbeda. Anda tidak harus *install RStudio* untuk dapat menggunakan program *R* (melalui *R GUI*). Tapi pengguna harus *install R* terlebih dahulu sebelum *install* dan menggunakan *RStudio* karena *RStudio* membutuhkan program *R* yang sudah terinstall di PC. Gambar diatas mengilustrasikan perumpamaan *R* ini seperti kerangka mobil dan mesinnya, sedangkan *RStudio* seperti kerangka luar mobil dan interiornya. Pengguna tidak dapat menggunakan mobil tersebut jika pengguna hanya mempunyai kerangka luar dan *dashboard*-nya (*RStudio*) saja.

1.3.1 Instalasi RStudio

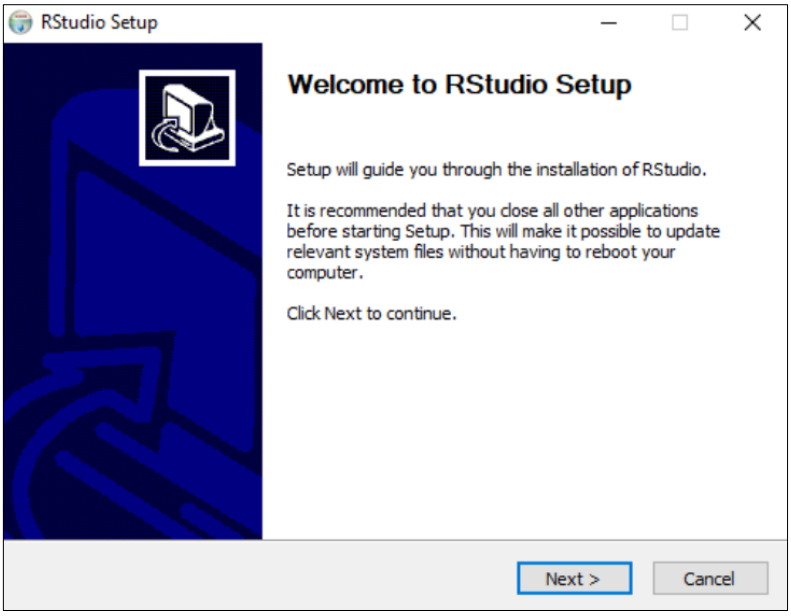
1. Sebelum melakukan instalasi, download *RStudio Desktop* pada laman **https://www.rstudio.com/products/rstudio/download/**



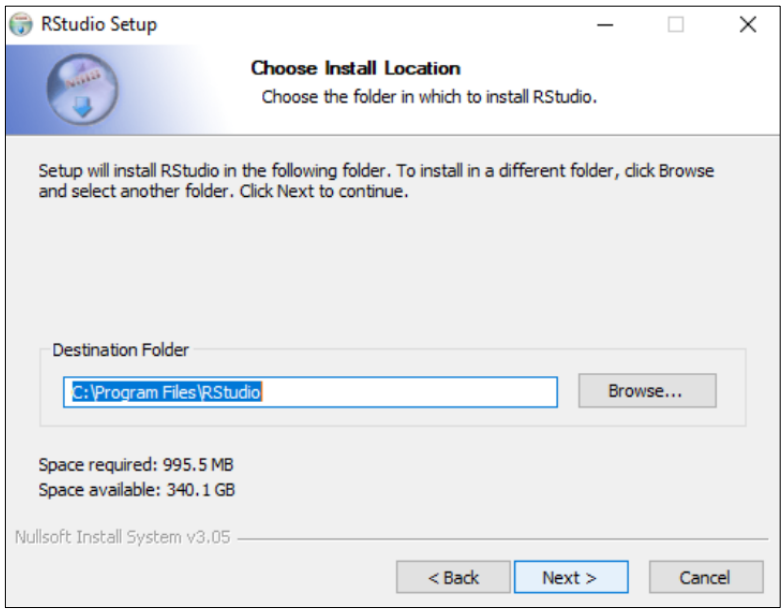
2. *Scroll* sampai pada bagian bawah dan pilih sesuai dengan *System Type* pada PC.

OS	Download	Size	SHA-256
Windows 10/8/7	RStudio-1.3.1056.exe	171.62 MB	a8f1fee5
macOS 10.13+	RStudio-1.3.1056.dmg	148.64 MB	f343c77d
Ubuntu 16	rstudio-1.3.1056-amd64.deb	124.56 MB	cbd5e5e5
Ubuntu 18/Debian 10	rstudio-1.3.1056-amd64.deb	126.50 MB	cd1a9e17
Fedora 19/Red Hat 7	rstudio-1.3.1056-x86_64.rpm	146.86 MB	0b1576bb
Fedora 28/Red Hat 8	rstudio-1.3.1056-x86_64.rpm	150.95 MB	bc4b3f44
Debian 9	rstudio-1.3.1056-amd64.deb	126.65 MB	3fb317e5
SLES/OpenSUSE 12	rstudio-1.3.1056-x86_64.rpm	119.17 MB	1be3540b
OpenSUSE 15	rstudio-1.3.1056-x86_64.rpm	128.14 MB	0e881257

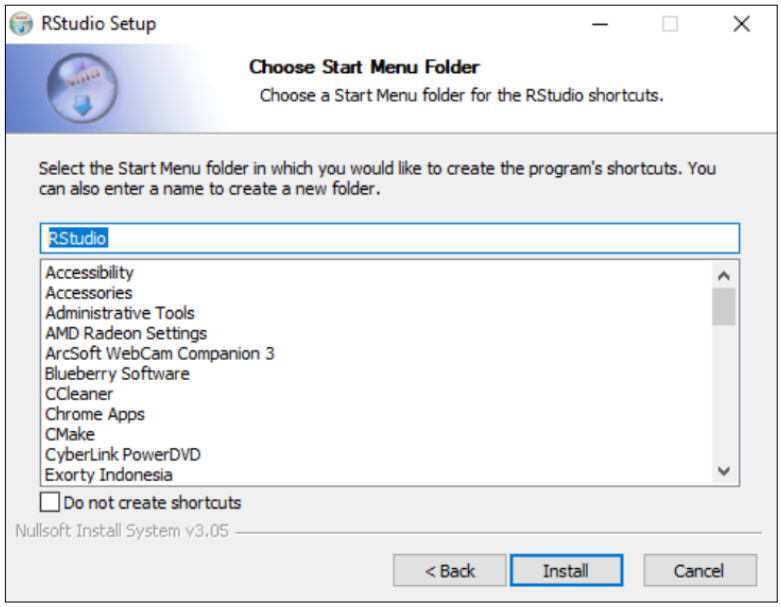
3. Jalankan installer yang sudah didownload dan klik *Next*.



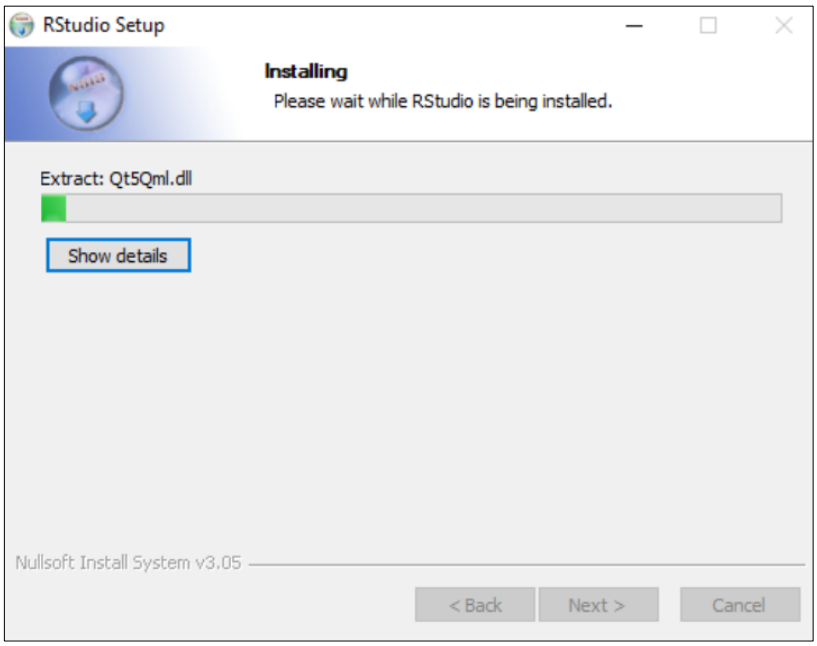
4. Pilih lokasi atau tempat penyimpanan untuk *Rstudio* lalu klik *Next*.



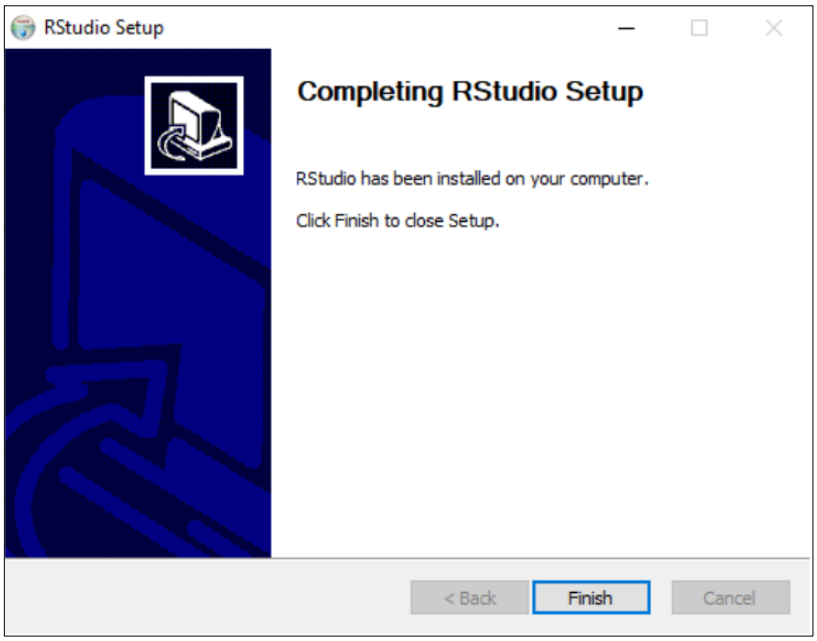
5. Masukkan nama *folder* yang akan digunakan pada *Start Menu* atau ceklis pada *Do not create shortcuts* apabila pengguna tidak ingin melakukan pembuatan *folder* pada *Start Menu*. Klik *Next*.



6. Tunggu hingga proses instalasi selesai.



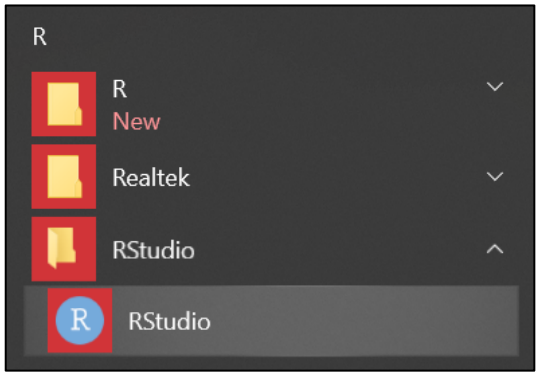
7. Klik *Finish* untuk mengakhiri proses instalasi.



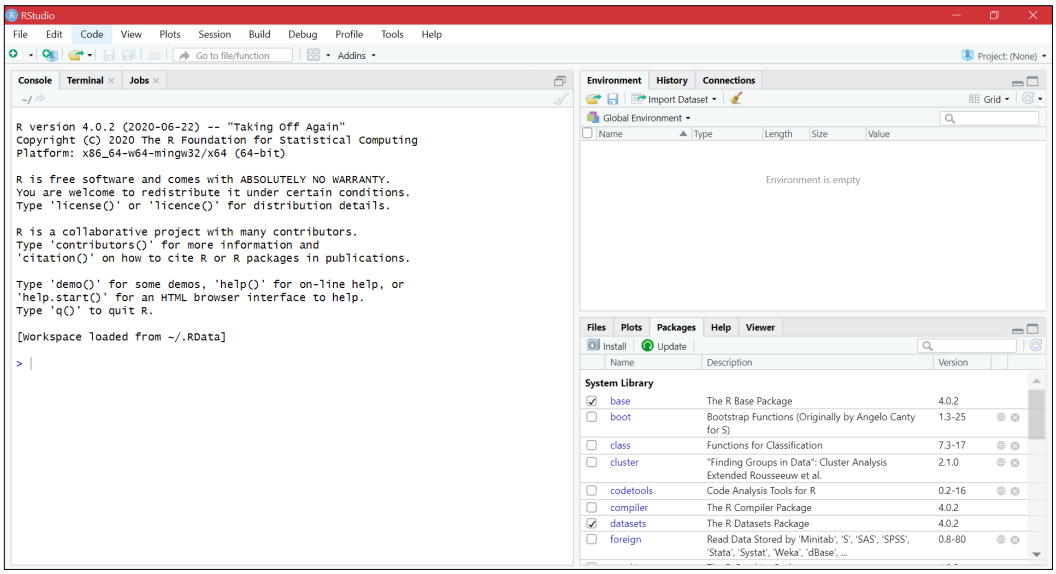
Selanjutnya pada panduan ini akan lebih banyak menggunakan *RStudio* dibandingkan dengan *R GUI* dikarena lebih mudah dan interaktif.

1.4 Penggunaan RStudio

Setelah selesai *install*, pengguna bisa membuka *RStudio* yang ada pada *Start Menu*, hampir sama dengan *R GUI*.



Berikut ini adalah tampilan awal ketika pengguna membuka program *RStudio*.

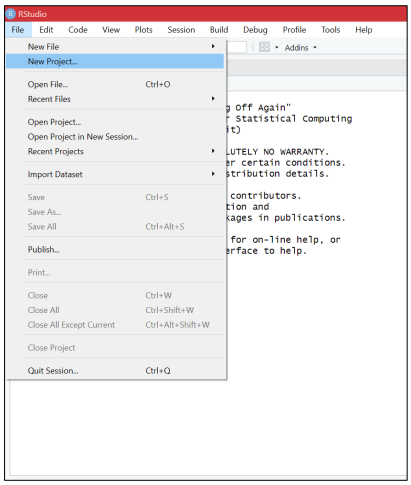


1.4.1 Membuat Sebuah *Project*

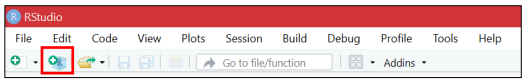
Hal yang sebaiknya menjadi kebiasaan pengguna ketika menggunakan *RStudio* adalah membuat sebuah *Project* untuk setiap pekerjaan yang berbeda. Pengguna dapat membuat sebuah *project* baru di *RStudio* dengan cara:

1. Pilih ***New Project…*** dari 3 cara yang tersedia, a) dari menu *File*, b) dari *toolbar*, atau c) dari menu di pojok kanan atas seperti terlihat pada gambar dibawah.

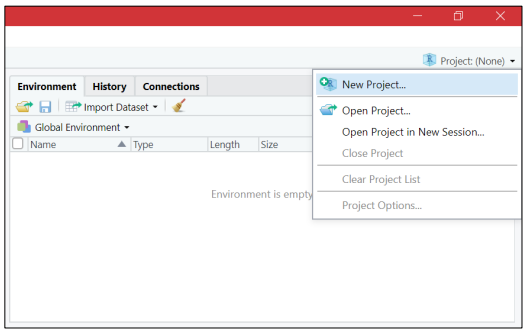
(a) Melalui menu *File*



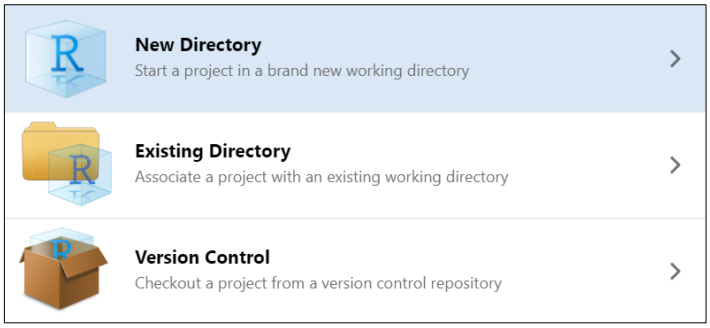
(b) Melalui *Toolbar*



(c) Menu pada pojok kanan atas

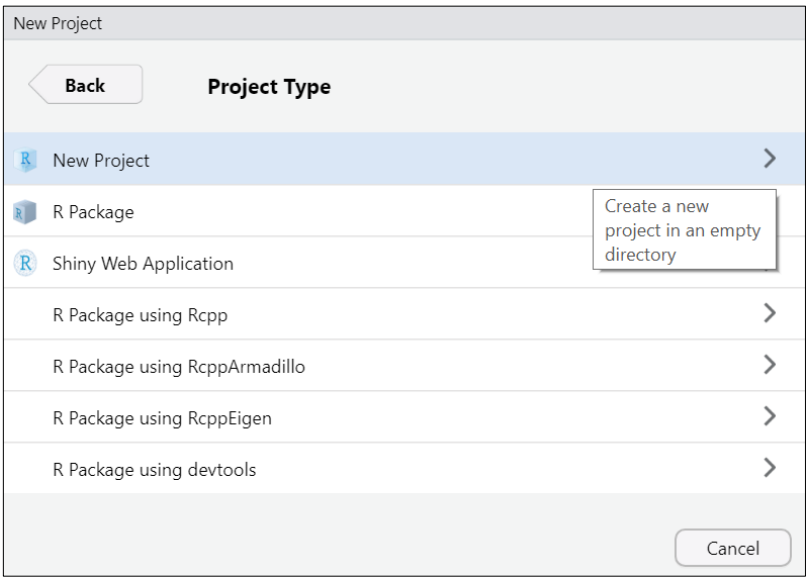


2. Pilih item yang sesuai dengan kebutuhan pengguna:
   a. ***New Directory***: jika pengguna belum mempunyai *folder* yang pengguna jadikan sebagai *working directory* untuk *project* tersebut.
   b. ***Existing Directory***: Jika *folder* pengguna sudah ada sebelumnya.
   c. ***Version Control***: Jika pengguna ingin membuat *project* berdasarkan *repository* yang sudah ada, misalnya di **GiHub** atau **SVN**.

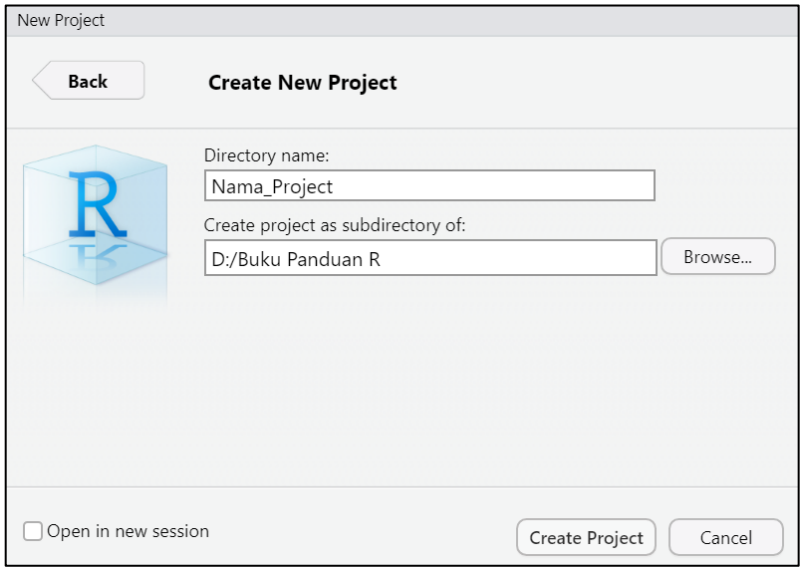


Dikarena pada penjelasan kali ini akan membuat *project* baru dan belum ada *folder* yang sudah disiapkan, maka pilih ***New Directory***.

3. Pilih *Project Type* sesuai dengan *project* yang akan pengguna buat seperti yang terlihat pada gambar dibawah. Sebagai contoh, akan dibuat sebuah *project* kosong, pilih *New Project*.



4. Tuliskan nama *project* yang akan menjadi nama *folder* baru sebagai *subfolder* di dalam *folder* "*Create project as subdirectory of*:" seperti terlihat pada gambar dibawah lalu klik *Create Project*.

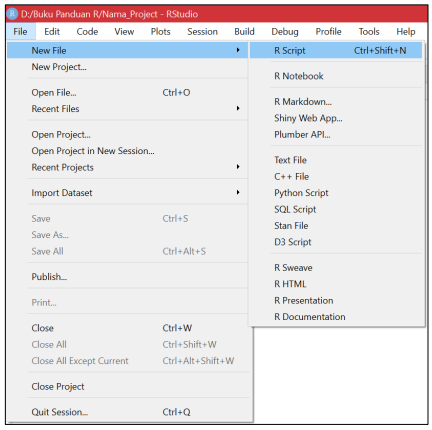


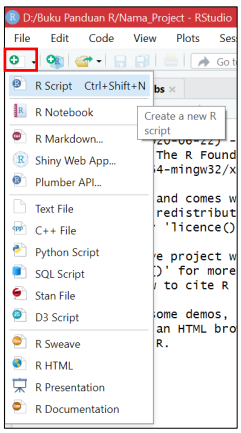
Untuk jenis *project* lainnya, pengguna dapat mencoba dan mengeksplorasi sendiri.

1.4.2 Membuat *File Script* Baru

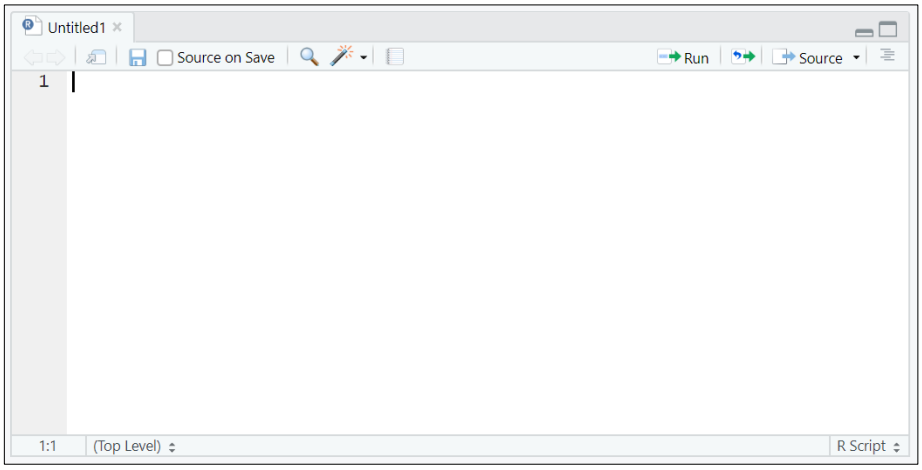
Seperti halnya pada *R GUI*, pengguna juga dapat menuliskan langsung perintah atau *script* pada jendela *console* di *RStudio*. Namun pada penjelasan kali

ini menuliskan semua *script* di bagian *script editor*. Hal ini sebaiknya menjadi kebiasaan pengguna ketika menggunakan *RStudio*. Untuk membuat *script editor* baru di *RStudio*, pengguna dapat klik menu *File >> New File >> R Script*. Cara yang kedua adalah dengan *toolbar New File*. Pengguna dapat lihat pada gambar dibawah. Pengguna juga dapat menggunakan *shortcut RStudio* dengan menekan tombol `Ctrl + Shift + N` jika Anda menggunakan OS *Windows*.





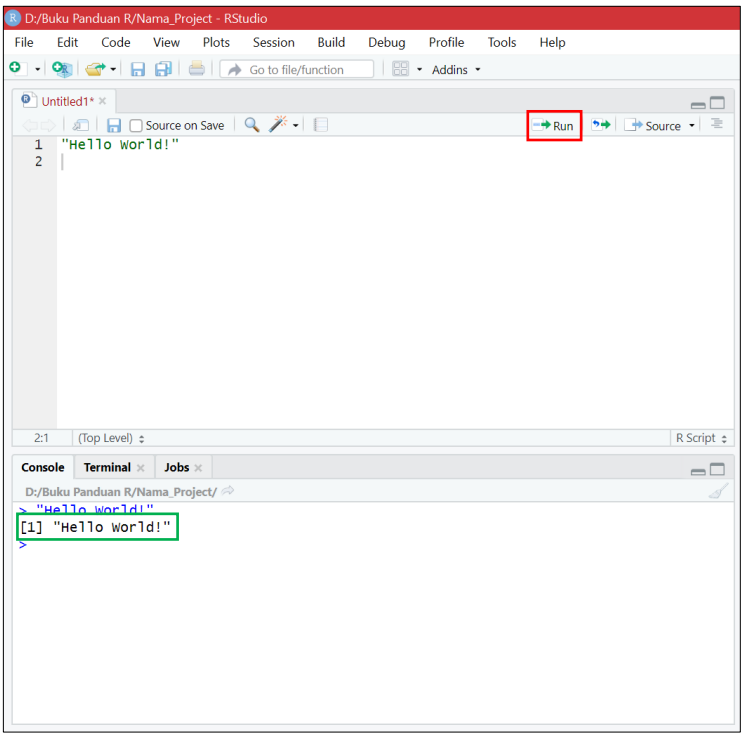
Setelah itu akan muncul *file script* kosong seperti tampilan gambar dibawah.



1.4.3 Menjalankan *Script* Pada *Script Editor*

Setelah berhasil menyiapkan *file script editor* baru yang masih kosong, pengguna dapat menuliskan semua yang akan pengguna lakukan dengan *R*. Sebagai contoh kecil, akan diperlihatkan cara menampilkan kata "*Hello World!*" menggunakan bahasa *R*. Di *R* pengguna cukup menuliskan `"Hello World!"` pada

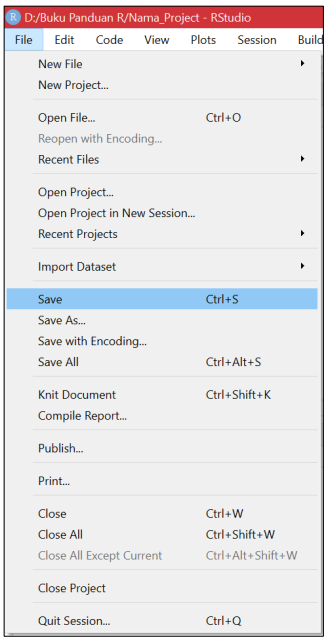
script editor, kemudian klik *Run* atau tekan tombol Ctrl + Enter ketika *cursor* berada pada baris *script* tersebut. Tentu saja pengguna juga dapat melakukannya seperti yang ada di *Python* dengan fungsi print().



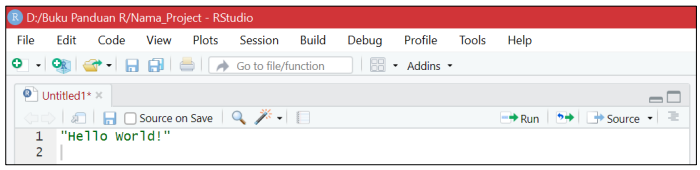

```
print("Hello World!")
```

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

Selanjutnya untuk menyimpan *script* tersebut menjadi sebuah *file*, pengguna dapat melakukan a) klik *icon Save* pada *toolbar*, b) pilih menu *File* >> *Save* atau *Save As...* atau c) menggunakan *shortcut* dengan menekan tombol Ctrl + S.

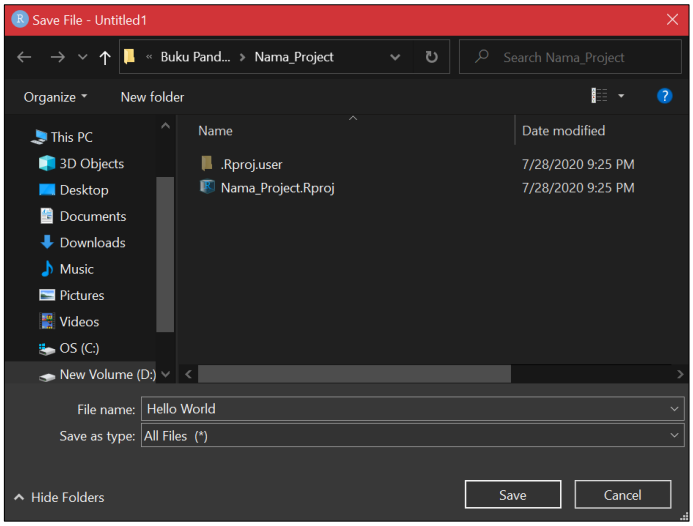


(a) Melalui menu *File*



(b) Melalui *Toolbar*

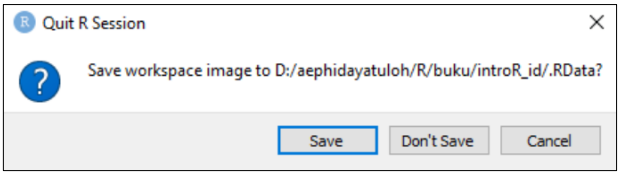
Kemudian tentukan lokasi *folder* tempat menyimpan *file script* tersebut. Selanjutnya berikan nama *file* pada kotak *File name:* seperti pada gambar dibawah. Terakhir klik *Save*.



1.4.4 Menutup *RStudio*

Setelah selesai menggunakan *RStudio*, pengguna dapat menutupnya seperti menutup *software* pada umumnya. Namun yang perlu diperhatikan adalah ketika

muncul konfirmasi seperti gambar dibawah. Sangat disarankan untuk tidak menyimpan *workspace* ketika menutup *RStudio*. Karena ketika pengguna membuka kembali *RStudio* maka akan memuat (*loading*) kembali semua yang disimpan dari pekerjaan sebelumnya. Hal ini akan sangat berpengaruh ketika ukuran *image* dari pekerjaan yang disimpan tersebut relatif besar.



BAB 2| DASAR PEMROGRAMAN R

Pada bab ini akan membahas hal-hal dasar tentang *R* yang harus diketahui dan dikuasai. Hal yang paling sederhana yang dapat dilakukan *R* adalah sebagai kalkulator. Pengguna bisa mengetikan perintah di bawah ini pada *console RStudio* dan tekan tombol *Enter* setelah selesai.

```
2 + 4
```

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

Akan muncul hasil `[1] 6`. Hasil `[1]` menunjukkan bahwa yang ditampilkan adalah dari elemen pertama. Hal ini akan dibahas di bagian 2.4.

Tanda `>` adalah *prompt* yang menunjukkan bahwa *R* sedang dalam posisi siap menerima perintah baru. Jika perintah belum lengkap maka akan berganti menjadi tanda `+`. Artinya ada perintah atau bagian *script* yang belum selesai.

```
> 2 +
+
```

Perhatikan setelah pengguna menekan tombol *Enter* maka kursor pada *R* yang sebelumnya `>` berganti menjadi `+` yang menandakan bahwa perintah belum lengkap. Maka jika pengguna kembali menuliskan bilangan lain, misalkan 4 dan menekan tombol *Enter* maka *prompt* pada *R* akan kembali menjadi `>` setelah menuliskan hasilnya karena perintah sudah lengkap dan selesai.

```
> 2 +
+ 4
## [1] 6
```

R adalah bahasa pemrograman dengan *case-sensitive*. Artinya perbedaan huruf kapital dan huruf kecil sangat berpengaruh. Karena itu, penulisan nama objek

atau nilai berupa karakter sangat tergantung dari kapitalisasinya. Perhatikan perbedaan dari kedua contoh berikut ini.

```
a <- 3
a
## [1] 3
```

```
A
## Error: object 'A' not found
```

Misalkan pengguna membuat sebuah objek bernama `a` dengan nilai sebuah konstanta. Ketika pengguna ingin mengambil nilai dari objek tersebut maka pengguna hanya bisa memanggil dengan nama yang sama persis. Pengguna tidak bisa memanggil objek tersebut dengan nama lain meskipun ketika pengguna mengucapkan dengan suara pengucapan yang sama. Pengguna tidak bisa memanggil objek `a` tersebut dengan `A`.

Ketika objek yang pengguna panggil belum ada pada *session* atau *workingspace R* saat ini, maka akan muncul *error* seperti contoh di atas.

2.1 Assigment

Bahasa pemrograman *R* mempunyai sedikit perbedaan dengan bahasa pemrograman pada umumnya. Salah satunya adalah pada operator *assignment*. Hampir semua bahasa pemrograman lain menggunakan tanda `=` sebagai operator *assignment*. Di *R*, yang utama dan paling banyak digunakan oleh pengguna *R* adalah operator panah kiri (`<-`). `obj <- expr` berarti "memasukkan nilai hasil dari operasi di sisi kanan (`expr`) ke dalam objek di sisi kiri (`obj`)". Pada contoh berikut, akan dimasukkan nilai numerik `5` ke objek yang disebut `x`.

```
x <- 5
x
```

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

Sekarang pertanyaannya adalah apakah tidak bisa menggunakan operator `=` sebagai operator assignment? Tentu saja pengguna juga bisa menggunakannya.

```
x = 5
```

Hal ini sangat membantu jika seorang programmer yang menggunakan bahasa pemrograman lain yang menggunakan operator `=` sebagai operator *assignment*. Jadi tidak perlu bingung dengan "Kapan harus menggunakan `<-` atau `=`?". Tapi pengguna didorong untuk terbiasa menggunakan `<-` saat membuat program menggunakan *R*.

Jika ingin mengetahui nilai suatu objek cukup panggil objek tersebut atau gunakan fungsi `print()`.

```
X
```

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

```
# or
print(x)
```

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

2.2 Operator *Assignment* Lainnya

Ada beberapa operator *assignment* lainnya yang dapat digunakan pada *R*. Di bawah ini adalah daftar operator *assignment*.

Operator	Cakupan	Penjelasan
`<-`	Lokal/Global	Nilai dari sebelah kanan dimasukkan ke dalam objek di sebelah kiri.
`->`	Lokal/Global	Nilai dari sebelah kiri dimasukkan ke dalam objek di sebelah kanan.
`<<-`	Global	Nilai dari sebelah kanan dimasukkan ke dalam objek global di sebelah kiri.
`->>`	Global	Nilai dari sebelah kiri dimasukkan ke dalam objek global di sebelah kanan.

Perhatikan contoh dibawah ini

```
# Contoh 1
x <- 5
5 -> x
X
```

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

```
# Contoh 2
y <- 2 + 4
2 + 4 -> y
Y
```

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

Seperti yang terlihat diatas, operator `->` memiliki sisi yang berlawanan dengan `<-`. Nilai atau ekspresi yang mengembalikan nilai ada di sisi kiri, sedangkan objek di sisi kanan. Untuk dua operator *assignment* yang lain akan dibahas pada pembahasan *function* (Bab 2.10) dalam *R* karena biasanya hanya digunakan dalam sebuah fungsi.

Jika memasukan nilai baru ke dalam sebuah objek yang sama maka nilai yang sebelumnya akan dihapus dan digantikan dengan nilai yang baru.

```
# Nilai x sebelumnya
X
```

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

```
# Nilai x yang baru
x <- 8 + 2
x
```

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

```r
# Nilai x yang baru
x <- x + 5
x
```

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

Ketika menggunakan *R*, setiap yang ada di *R* disebut objek. Jenis-jenis objek data yang ada di *R* adalah *vector*, *factor*, *matriks*, *array*, *data frame*, *list* dan objek berupa *function*.

2.3 Penamaan Objek

Aturan penamaan objek pada *R*, seperti *vector* (2.4), *matrix* (2.6), *data frame* (2.8) dan lain-lain, hampir sama dengan aturan penamaan pada bahasa pemrograman lain. Namun ada beberapa aturan khusus yang terdapat pada *R*. Berikut aturan penamaan objek pada *R*.

1. Menggunakan kombinasi alfabet (a-z, A-Z), angka (0-9), titik atau *underscore*.
2. Diawali alfabet, titik atau *underscore*, tidak boleh diawali dengan angka.
3. Tidak mengandung spasi, tab atau karakter khusus seperti `!`, `@`, `#` dan lainnya.
4. Sebaiknya tidak menggunakan beberapa penamaan atau nilai yang sudah digunakan oleh *R* (*function* dan *keyword* lainnya). Misalnya `c`, `q`, `TRUE`, `FALSE`, `df`, `dt`, `rnorm`, `runif`, `rf`, `exp`, dan lain-lain. Untuk mengetahui nama-nama yang sudah digunakan oleh *R* ketikkan perintah `?reserved` pada *console RStudio*.

2.4 Vektor

Vector adalah objek data paling sederhana yang ada dalam *R*. Secara umum jenis vektor terbagi 2, yaitu *numeric* dan *character*. Ada banyak sekali cara untuk membuat sebuah vektor pada *R*. Di bagian ini akan dibahas beberapa cara yang banyak dan mungkin akan sering digunakan.

2.4.1 Fungsi `c()`

Fungsi yang paling sering digunakan untuk membuat sebuah vektor adalah dengan menggunakan fungsi `c()`.

```
x <- c(2,1,4,2,5)
x
```

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

Pada *script* diatas, dibuat sebuah objek `x` berupa *vector numeric*. Setiap elemen dipisah menggunakan tanda koma (`,`). Fungsi ini dapat digunakan untuk membuat vektor numerik atau karakter. Indeks pada *R* dimulai dari 1, tidak seperti kebanyakan bahasa pemrograman lain yang indeksnya dimulai dari 0. Hal ini cukup memudahkan pengguna karena umumnya manusia menghitung mulai dari 1, bukan dari 0.

Ketika ingin mengambil elemen ke 2 dari vektor `x`, maka dapat menjalankan perintah dibawah ini.

```
x[2]
```

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

2.4.2 Tanda titik dua (`:`)

Untuk membuat sebuah vektor numerik berurutan secara meningkat atau menurun. Perhatikan contoh berikut ini.

```
x <- 1:10 # 1 sampai 10
x
```

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

Fungsi dari operator `:` pada contoh diatas adalah membuat vektor numerik dengan nilai dari 1 s/d 10. Tentu saja operator *increment* ini hanya dapat digunakan

untuk numerik dan meningkat sebesar 1 nilai. Operator ini dapat juga digunakan untuk membuat vektor dengan nilai menurun.

```
# membuat vektor numerik dengan nilai dari 10 s/d -10 secara menurun
x <- 10:-10 # 10 sampai -10
x
```

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

2.4.3 Fungsi `seq()`

Fungsi ini dapat digunakan untuk membuat vektor berurutan dengan *increment* atau peningkatan nilai tertentu.

```
# 1 sampai 10 dengan increment 1 (default by = 1)
x <- seq(from = 1, to = 10)
x
```

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

Secara *default*, *increment* dari fungsi `seq()` adalah `by = 1`. Jika ingin nilai *increment* lain maka hanya perlu mengganti nilai pada argumen `by`. Fungsi ini juga hanya dapat digunakan untuk membuat vektor numerik.

```
# 1 sampai 20 dengan increament 2
x <- seq(from = 1, to = 20, by = 2)
x
```

```
## [1] 1 3 5 7 9 11 13 15 17 19
```

Selain menyesuaikan *increament* pada `seq()` menggunakan `by`, *increament* pada fungsi juga dapat disesuaikan menggunakan `length.out` yang berfungsi untuk meningkatkan nilai elemen berdasarkan panjang vektor yang ditargetkan dan `along.with` yang berfungsi untuk meningkatkan elemen berdasarkan panjang vektor lain. Perhatikan contoh berikut.

```r
# 1 sampai 10, sebanyak 7 elemen, increment mengikut
x <- seq(from = 1, to = 10, length.out = 7)
x
```

```
## [1] 1.0  2.5  4.0  5.5  7.0  8.5 10.0
```

```r
# 1 sampai 10, sebanyak elemen dari vector lain
x <- seq(from = 1, to = 10, along.with = 1:4)
x
```

```
## [1]  1  4  7 10
```

2.4.4 Mengambil satu kolom dari data frame atau matriks

Mengambil sebuah kolom dari sebuah *data frame* akan dibahas lebih jauh di bagian *data frame* (2.8). Dengan menggunakan tanda dolar `$` dan diikuti dengan nama kolom yang akan diambil dari *data frame* tersebut.

```r
# data frame mtcars diambil semua nilai yang ada di kolom mpg
mtcars$mpg
```

```
##  [1] 21.0 21.0 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 17.8 16.4
## [13] 17.3 15.2 10.4 10.4 14.7 32.4 30.4 33.9 21.5 15.5 15.2 13.3
## [25] 19.2 27.3 26.0 30.4 15.8 19.7 15.0 21.4
```

Cara ini akan dicontohkan lebih banyak pada bagian-bagian selanjutnya di bab ini.

2.4.5 Fungsi `unlist()`

Fungsi ini berguna untuk menjadikan sebuah objek *list* menjadi sebuah vektor. Pembahasan lebih lanjut akan dibahas di bagian 2.9.

```r
x <- list(mtcars$mpg, mtcars$disp)
x
```

```
## [[1]]
##  [1] 21.0 21.0 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 17.8 16.4 17.3
## [14] 15.2 10.4 10.4 14.7 32.4 30.4 33.9 21.5 15.5 15.2 13.3 19.2 27.3
```

```
## [27] 26.0 30.4 15.8 19.7 15.0 21.4



## [[2]]
##  [1] 160.0 160.0 108.0 258.0 360.0 225.0 360.0 146.7 140.8 167.6 167.6
## [12] 275.8 275.8 275.8 472.0 460.0 440.0  78.7  75.7  71.1 120.1 318.0
## [23] 304.0 350.0 400.0  79.0 120.3  95.1 351.0 145.0 301.0 121.0
```

```
x <- unlist(x)
x
```

```
##  [1]  21.0  21.0  22.8  21.4  18.7  18.1  14.3  24.4  22.8  19.2  17.8
## [12]  16.4  17.3  15.2  10.4  10.4  14.7  32.4  30.4  33.9  21.5  15.5
## [23]  15.2  13.3  19.2  27.3  26.0  30.4  15.8  19.7  15.0  21.4 160.0
## [34] 160.0 108.0 258.0 360.0 225.0 360.0 146.7 140.8 167.6 167.6 275.8
## [45] 275.8 275.8 472.0 460.0 440.0  78.7  75.7  71.1 120.1 318.0 304.0
## [56] 350.0 400.0  79.0 120.3  95.1 351.0 145.0 301.0 121.0
```

Fungsi `unlist()` menggabungkan semua *list* menjadi sebuah vektor. Walaupun ditampilkan ke samping, dimensi vektor pada R sebenarnya ke bawah. Bayangkan untuk sebuah vektor seperti satu kolom di Ms Excel. Semua contoh diatas untuk membuat vektor adalah vektor numeric.

2.4.6 Fungsi `rep()`

Dengan fungsi `rep()`, pengguna dapat membuat sebuah vektor dengan mengulang-ulang nilai yang diinginkan sebanyak yang dibutuhkan.

```
# vector yang semua elemennya bernilai 3 sebanyak 10 elemen
x <- rep(3, 10)
x
```

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

2.4.7 Vektor Karakter

Vektor karakter adalah vektor yang semua elemennya bertipe character. Perhatikan contoh dibawah ini.

```
y <- c("a", "A", "d", "c")
y
```

```
## [1] "a" "A" "d" "c"
```

Ketika membuat sebuah vektor bernilai numerik namun ada satu elemen yang bertipe karakter, maka semua elemennya akan bertipe karakter. Perhatikan contoh dibawah ini.

```
c(1, 2, 3, 6, "a")
```

```
## [1] "1" "2" "3" "5" "a"
```

Pada aplikasi *R* ada 2 jenis vektor khusus yang bertipe karakter, yaitu `letters` dan `LETTERS`.

```
Letters
```

```
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n"
## [15] "o" "p" "q" "r" "s" "t" "u" "v" "w" "x" "y" "z"
```

```
LETTERS
```

```
##  [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N"
## [15] "O" "P" "Q" "R" "S" "T" "U" "V" "W" "X" "Y" "Z"
```

Dua buah vektor atau lebih dapat digabungkan dengan fungsi `c()`. Namun, jika salah satu vector bertipe karakter, maka vektor hasil gabungan akan menjadi vektor karakter. Contoh dibawah ini menggabungkan vektor `x` dan `y`. Vektor `x` adalah vektor numerik, sedangkan `y` adalah vektor karakter. Karena ada satu atau

lebih elemen yang berupa karakter, maka ketika `x` dan `y` digabungkan akan menjadi vektor karakter.

```
c(x, y)
```

```
## [1] "3" "3" "3" "3" "3" "3" "3" "3" "3" "3" "a" "A" "d" "c"
```

Cara lain yang dapat digunakan untuk membuat vektor karakter adalah menggunakan fungsi `paste()`, `paste0()` atau `sprintf()`. Jalankan dan perhatikan perbedaan dari contoh perintah dibawah ini.

```
paste("A", 1:5)
```

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

```
paste0("A", 1:5)
```

```
## [1] "A1" "A2" "A3" "A4" "A5"
```

```
sprinf("A%s", 1:5)
```

```
## [1] "A1" "A2" "A3" "A4" "A5"
```

2.5 Factor

Factor merupakan bentuk lebih luas dari vektor. Biasanya factor lebih sering digunakan untuk menyimpan data nominal atau ordinal. Misalnya vektor karakter yang berisi `"male"` dan `"female"`. Pada vektor karakter, nilainya adalah `"male"` dan `"female"` seperti terlihat apa adanya. Namun pada factor, tampilan dari isi datanya mungkin `"male"` dan `"female"` tetapi isi dari factor adalah pengkodean numerik. Misal untuk `"female"` nilai sebenarnya adalah `1`, sedangkan `"male"` berniali `2`.

```r
fc <- factor(c("SD", "SMA", "SMP", "SMP", "SD", "SMA", "SD", "SMP"))
fc
```

```
## [1] SD  SMA SMP SMP SD  SMA SD  SMP
## Levels: SD SMA SMP
```

dan nilai sebenarnya dari factor tersebut adalah:

```r
print.default(fc)
```

```
## [1] 1 2 3 3 1 2 1 3
```

Factor mempunyai level, secara *default* levelnya adalah berdasarkan urutan alfabet. Untuk merubah level dari sebuah factor, gunakan argumen `levels =`.

```r
factor(fc, levels = c("SD", "SMA", "SMP"))
```

```
## [1] SD  SMA SMP SMP SD  SMA SD  SMP
## Levels: SD SMP SMA
```

Dengan menambahkan `ordered = TRUE`, maka fungsi factor akan menentukan tingkatan levelnya berdasarkan urutan level yang sebelumnya telah disesuaikan.

```r
factor(fc, levels = c("SD", "SMA", "SMP"), ordered = TRUE)
```

```
## [1] SD  SMA SMP SMP SD  SMA SD  SMP
## Levels: SD < SMP < SMP
```

2.6 Matriks

Matriks adalah objek pada *R* yang memiliki 2 dimensi, baris (*row*) dan kolom (*column*), dan tipe nilainya sama. Ketika membuat sebuah matriks elemennya memiliki minimal 1 elemen bertipe karakter maka seluruh matriks tersebut akan bertipe karakter. Membuat matriks pada *R* menggunakan vektor yang dikonversi dimensinya.

```r
x <- mtcars$mpg # sebuah vector
length(x)
```

```
## [1] 32
```

Karena vektor `x` memiliki 32 elemen, maka dimensi matriks yang dapat dibuat adalah 2 angka hasil perkalian menghasilkan nilai 32. Salah satunya adalah `8 x 4 = 32`.

```
m <- matrix(data = x, nrow = 8, ncol = 4)
m
```

```
##      [,1] [,2] [,3] [,4]
## [1,] 21.0 22.8 14.7 19.2
## [2,] 21.0 19.2 32.4 27.3
## [3,] 22.8 17.8 30.4 26.0
## [4,] 21.4 16.4 33.9 30.4
## [5,] 18.7 17.3 21.5 15.8
## [6,] 18.1 15.2 15.5 19.7
## [7,] 14.3 10.4 15.2 15.0
## [8,] 24.4 10.4 13.3 21.4
```

Argumen `byrow = TRUE` artinya setiap elemen `x` diisikan ke `m` memenuhi baris terlebih dahulu. Jika `byrow = FALSE` maka setiap elemen `x` diisikan ke `m` berdasarkan kolom terlebih dahulu. Perhatikan contoh berikut.

```
matrix(1:10, nrow = 2, ncol = 5, byrow = TRUE)
```

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

```
matrix(1:10, nrow = 2, ncol = 5, byrow = FALSE)
```

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

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

Untuk membuat matriks dengan nilai yang sama seluruhnya, maka dapat dilakukan seperti pada contoh berikut.

```
matrix(data = 0, nrow = 5, ncol = 6)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    0    0    0    0    0    0
## [2,]    0    0    0    0    0    0
## [3,]    0    0    0    0    0    0
## [4,]    0    0    0    0    0    0
## [5,]    0    0    0    0    0    0
```

Untuk mengakses elemen dari suatu matriks, dapat menggunakan indeks dari baris atau kolomnya.

```
# Mengambil elemen matriks `m` di baris 4
m[4 ,]
```

```
## [1] 21.4 16.4 33.9 30.4
```

```
# Mengambil elemen matriks `m` di kolom 3
m[,3]
```

```
## [1] 14.7 32.4 30.4 33.9 21.5 15.5 15.2 13.3
```

```
# Mengambil elemen matriks `m` di baris 4 kolom 3
m[4, 3]
```

```
## [1] 33.9
```

```r
# Mengambil elemen matriks `m` di baris 4 dan 6, dan kolom 3
m[c(4, 6),3]
```

```
## [1] 33.9 15.5
```

R juga menyediakan sebuah fungsi bernama `diag()` untuk mengakses nilai-nilai pada diagonal utama sebuah matriks.

```r
diag(m)
```

```
## [1] 21.0 19.2 30.4 30.4
```

Pengguna juga dapat mengganti nilai dari elemen suatu matriks dengan menggunakan operator *assignment*.

```r
m[4, 3] <- 3
m[4, 3]
```

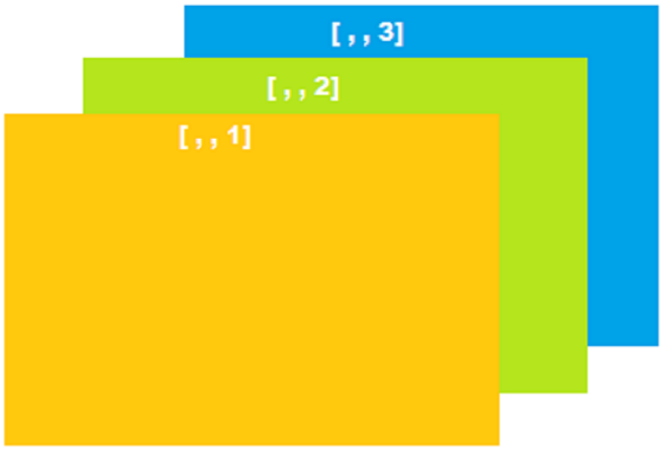
```
## [1] 3
```

```r
# perhatikan elemen di baris 4 kolom 3 sudah berubah jadi 3.0
M
```

```
##      [,1] [,2] [,3] [,4]
## [1,] 21.0 22.8 14.7 19.2
## [2,] 21.0 19.2 32.4 27.3
## [3,] 22.8 17.8 30.4 26.0
## [4,] 21.4 16.4  3.0 30.4
## [5,] 18.7 17.3 21.5 15.8
## [6,] 18.1 15.2 15.5 19.7
## [7,] 14.3 10.4 15.2 15.0
## [8,] 24.4 10.4 13.3 21.4
```

2.7 Array

Array merupakan objek seperti matriks dengan dimensi lebih banyak. Jika matriks hanya mempunyai 2 dimensi, maka array dapat memiliki lebih dari 2 dimensi.



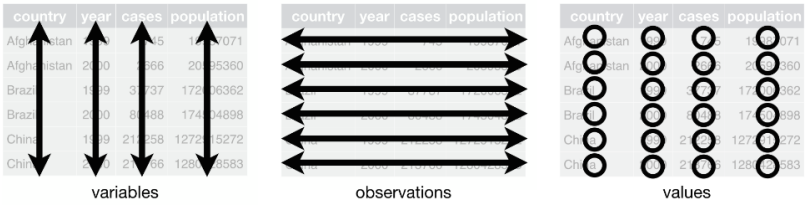

```
# membagi 32 elemen ke dalam array 4 x 4 x 2 (3 dimensi)
array(mtcars$mpg, dim = c(4, 4, 2))
```

```
## , , 1
##
##      [,1] [,2] [,3] [,4]
## [1,] 21.0 18.7 22.8 17.3
## [2,] 21.0 18.1 19.2 15.2
## [3,] 22.8 14.3 17.8 10.4
## [4,] 21.4 24.4 16.4 10.4
##
## , , 2
##
##      [,1] [,2] [,3] [,4]
## [1,] 14.7 21.5 19.2 15.8
## [2,] 32.4 15.5 27.3 19.7
## [3,] 30.4 15.2 26.0 15.0
## [4,] 33.9 13.3 30.4 21.4
```

2.8 Data Frame

Data frame layaknya sebuah tabel pada Ms Excel, terdiri dari baris dan kolom dengan nama masing-masing kolom berbeda. Matriks hanya bisa menyimpan tipe data yang sama, numerik atau karakter seluruhnya. Pada *data frame*, masing-masing kolom boleh memiliki tipe data yang berbeda. *Data frame* seperti umumnya bentuk tabel yang sering digunakan. Contoh data frame yang ada di dalam *R* salah satunya adalah `mtcars`.

Umumnya ketika melakukan analisis data, maka data yang digunakan adalah berupa tabel. Di dalam *R* sebuah tabel yang terdiri dari baris dan kolom disebut *data frame* atau *data.frame*. Baris dalam *data frame* disebut observasi (*observation*) dan kolom disebut variable (*variable*) (Perhatikan Gambar 2.1).



Gambar 2.1. Ilustrasi Data frame

Untuk membuat sebuah *data frame*, pengguna dapat menggunakan fungsi `data.frame()`. Misalkan akan membuat sebuah *data frame* bernama `data1` yang berisi 5 *observation* dan 2 *variable*. Variabel pertama bernama `x1` berisi sebuah vektor numerik dengan nilai {1, 3, 2, 6, 4} dan variabel kedua bernama `v2` berisi vektor karakter dengan nilai {“a”, “A”, “c”, “d”, “E”}. Pengguna dapat membuat vektor `x1` dan `v1` terlebih dahulu menggunakan fungsi `c()` (atau fungsi lain yang sesuai untuk membuat vector). Kemudian membuat *data frame* dari vektor-vektor tersebut. Perhatikan contoh berikut ini.

```
x1 <- c(1, 3, 2, 6, 4)
v1 <- c("a", "A", "c", "d", "E")
data1 <- data.frame(x1 = x1, v1 = v1)
```

Pengguna dapat melihat *data frame* yang sudah dibuat tersebut dengan memanggil nama objek *data frame*.

```
data1
```

```
##   x1 v1
## 1  1  a
## 2  3  A
## 3  2  c
## 4  6  d
## 5  4  E
```

Ketika membuat sebuah *data frame* yang berisi sebuah variabel karakter pada *R* dengan versi 3.6.3 atau sebelumnya, maka secara otomatis variabel dari vektor karakter tersebut akan dirubah menjadi variabel faktor (2.5). Hal ini terjadi karena fungsi `data.frame()` mempunyai sebuah argumen `stringsAsFactors = TRUE`. Pada *R* versi 4.0.0 atau lebih baru, variable dari vektor karakter akan tetap dijadikan sebuah variabel karakter karena argumen `stringsAsFactors = FALSE`. Tentu saja pengguna dapat merubahnya dengan menambahkan argumen tersebut ketika membuat *data frame*.

```
data1 <- data.frame(x1 = x1, v1 = v1, stringAsFactors = TRUE)
data1
```

```
##   x1 v1
## 1  1  a
## 2  3  A
## 3  2  c
## 4  6  d
## 5  4  E
```

Untuk mengetahui ukuran dimensi sebuah *data frame*, pengguna dapat gunakan fungsi `dim()`.

```
dim(data1)
```

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

Hasil dari fungsi `dim()` untuk *data frame* atau matriks adalah sebuah vektor dengan elemen pertama adalah banyaknya observasi, sedangkan elemen kedua adalah banyaknya variabel. Pada contoh diatas berarti *data frame* `data1` memiliki 5 observasi dan 2 variabel.

Untuk mengetahui struktur dari sebuah *data frame* pengguna dapat menggunakan fungsi `str()` (*structure*). Dengan fungsi ini pengguna bisa memperoleh informasi lebih lengkap dari sebuah *data frame* seperti banyaknya observasi dan variabel, nama-nama variabel, tipe variabel, dan beberapa nilai untuk masing-masing variabel.

```
str(data1)
```

```
## 'data.frame':    5 obs. of  2 variables:
##  $ x1: num  1 3 2 6 4
##  $ v1: Factor w/ 5 levels "a","A","c","d",..: 1 2 3 4 5
```

Dari hasil diatas dapat diketahui bahwa objek `data1` adalah sebuah *data frame* berukuran 5 observasi dan 2 variabel. Nama variabel yang ada adalah `x1` dan `v1`. Variabel `x1` adalah variabel bertipe numerik, sedangkan `v1` adalah variabel karakter alias factor. Nilai pada baris pertama *data frame* `data1` untuk variabel `x1` adalah 1 dan variabel `v1` adalah "a". Nilai "a" dikodekan sebagai 1, "A" dikodekan sebagai 2, "c" dikodekan sebagai 3, dan seterusnya.

Selanjutnya untuk mengambil nilai sebuah variabel dari sebuah *data frame* dapat menggunakan tanda dollar (`$`) atau menggunakan indeks. Perhatikan contoh berikut.

```
# menggunakan tanda dollar
data1$x1
```

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

```
# menggunakan indeks urutan posisi variabel
data1[, 1]
```

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

```
# menggunakan nama variable pada indeks
data1[,"x1"]
```

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

Untuk membuat variabel baru pengguna dapat menggunakan cara yang hampir sama dan menggunakan *assignment*. Berikut ini adalah contoh membuat variabel baru bernama `x2` berupa variabel numerik.

```
data1$x2 <- 1:5
data1
```

```
##   x1 v1 x2
## 1  1  a  1
## 2  3  A  2
## 3  2  c  3
## 4  6  d  4
## 5  4  E  5
```

```
str(data1)
```

```
## 'data.frame':    5 obs. of  3 variables:
##  $ x1: num  1 3 2 6 4
##  $ v1: Factor w/ 5 levels "a","A","c","d",..: 1 2 3 4 5
##  $ x2: int  1 2 3 4 5
```

Tipe `num` artinya variabel tersebut adalah tipe double dan tipe `int` adalah tipe integer. Kemudian akan dibuat variabel baru bernama `x3` yang merupakan penjumlahan dari variable `x1` dan `x2`.

```
data1$x3 <- data1$x1 + data1$x2
str(data1)
```

```
## 'data.frame':    5 obs. of  4 variables:
##  $ x1: num  1 3 2 6 4
##  $ v1: Factor w/ 5 levels "a","A","c","d",..: 1 2 3 4 5
##  $ x2: int  1 2 3 4 5
##  $ x3: num  2 5 5 10 9
```

2.9 List

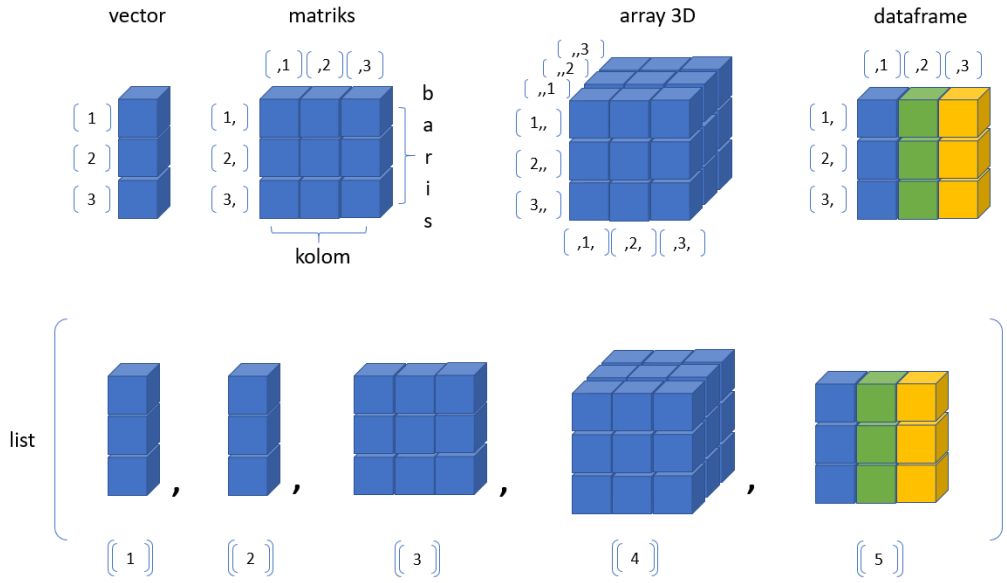
Objek `list` pada dasarnya mirip seperti vektor, hanya saja tipe elemennya bisa berbeda. Jika pada vektor numerik semua elemennya harus berupa numerik dan pada vektor karakter semuanya harus karakter, maka pada objek `list` elemennya dapat berupa vektor, factor, matriks, array, data frame, bahkan list didalam list atau objek lain seperti model prediktif yang dibuat oleh *R*. Contoh membuat list dengan fungsi `list()`.

```
list(2, "x1", c(4, 5, 2), iris[1:5,])
```

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

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
```

Gambar berikut mengilustrasikan objek list:



2.10 *Function* dan *Packages*

Pada dasarnya *R* adalah bahasa pemrograman *functional* atau *Functional Programming* (FP). Wickham (2015) menyatakan bahwa *R* menyediakan banyak alat untuk pembuatan dan manipulasi fungsi. Secara khusus *R* memiliki apa yang dikenal sebagai *first class function*. Pengguna dapat melakukan apa saja dengan fungsi seperti yang dapat pengguna lakukan dengan vektor sebelumnya yaitu pengguna dapat memasukkan mereka ke dalam sebuah objek, menyimpannya dalam list, menjadikannya sebagai argumen pada fungsi yang lain, membuat fungsi di dalam fungsi, dan bahkan menjadikannya sebagai hasil keluaran dari suatu fungsi.

2.10.1 Menggunakan fungsi yang sudah ada

Karena dibuat untuk memudahkan analisis data, *R* mempunyai banyak fungsi yang tersedia untuk melakukan analisis statistik. Mendapatkan nilai rata-rata dari sebuah vektor numerik menggunakan fungsi `mean()`, mendapatkan nilai minimum atau maksimum tersedia fungsi `min()` dan `max()`, fungsi `sd()` yang digunakan untuk mendapatkan nilai standard deviasi atau fungsi `var()` untuk mendapatkan nilai ragam (*varians*). Perhatikan contoh berikut ini.

```
x <- seq(1, 100, by = 4)

# Rata-rata vektor x
mean(x)
```

```
## [1] 49
```

```
# Minimum vektor x
min(x)
```

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

```
# Maksimum vektor x
max(x)
```

```
## [1] 97
```

Jika ingin mencari nilai minimum dan maksimum sekaligus, dapat menggunakan fungsi `range()`. Output dari fungsi ini adalah vektor numerik dengan dua elemen berisi nilai minimum dan maksimum.

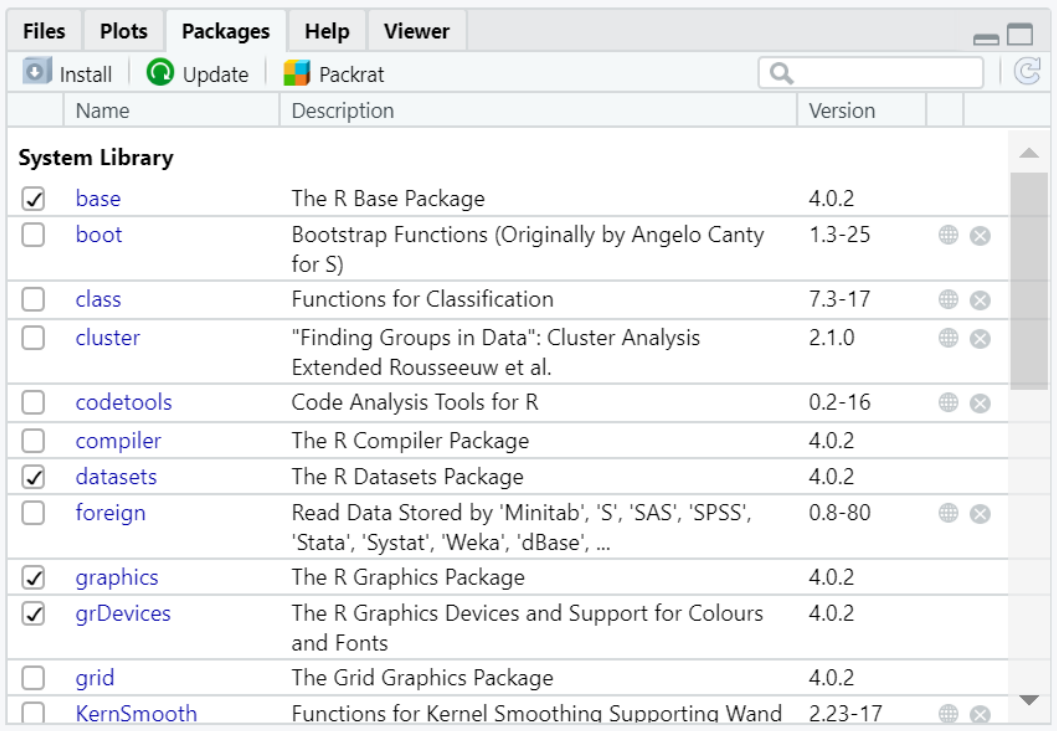
```
range(x)
```

```
## [1]  1 97
```

Jika ingin mencari ringkasan data (minimum, Q1, Q2 atau median, Q3, maksimum), dapat menggunakan fungsi `summary()`.

```
summary(x)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##       1      25      49      49      73      97
```



Jika fungsi atau data yang akan digunakan berasal dari sebuah *package*, maka gunakan fungsi `library()`, `require()` atau beri tanda ceklis pada tab *Packages* seperti pada gambar diatas untuk mengaktifkan *package* tersebut pada *session* yang sedang digunakan. Tentunya *package* tersebut harus sudah diinstall terlebih dahulu. Misalkan ingin menggunakan data `flights` dari *package* `nycflights13`. Hal pertama yang harus dipastikan adalah *package* tersebut sudah terinstall dengan pada *R*. Jika belum menginstal *package* tersebut, maka pengguna dapat melakukan hal berikut.

```
Install.packages("nycflights13")
```

Ketika akan menginstal sebuah *package*, nama *package* tersebut harus berupa *string* dan hanya perlu melakukan satu kali install saja, tidak perlu install setiap kali akan ingin digunakan. Apabila memanggil data `flights` dari package `nycflights13` tanpa mengaktifkan *package* terlebih dahulu, maka akan muncul sebuah error yang menyatakan bahwa objek `flights` tidak ditemukan.

```
Flights
```

```
## Error: object 'flights' not found
```

Karena data `flights` ada di dalam *package* `nycflights13` maka perlu aktifkan terlebih dahulu *package* tersebut. Pengguna dapat mengaktifkannya dengan perintah berikut ini.

```
library("nycflights13")
```

Kemudian panggil data `flights`. Gunakan fungsi `head()` untuk menampilkan beberapa baris pertama saja dari data. Secara *default* fungsi `head()` akan menampilkan 6 baris pertama saja dari data.

```
head(flights)
```

```
## # A tibble: 6 x 19
##    year month   day dep_time sched_dep_time dep_delay arr_time
##   <int> <int> <int>    <int>          <int>     <dbl>    <int>
## 1  2013     1     1      517            515         2      830
## 2  2013     1     1      533            529         4      850
## 3  2013     1     1      542            540         2      923
## 4  2013     1     1      544            545        -1     1004
## 5  2013     1     1      554            600        -6      812
```

```
## 6  2013     1     1      554           558        -4      740
## # ... with 12 more variables: sched_arr_time <int>, arr_delay
<dbl>,
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>,
## #   dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dttm>
```

Karena *package* `nycflights13` sudah diaktifkan sebelumnya maka ketika data `flights` dipanggil tidak akan terjadi error.

2.10.2 Membuat fungsi sendiri

Ketika membuat sebuah *script* yang akan digunakan berkali-kali namun dengan nilai input yang berbeda-beda, maka ada baiknya *script* tersebut dibuat menjadi sebuah fungsi atau *function* sesuai dengan kebutuhan atau *User-Defined Function*. Untuk membuat function di *R*, gunakan fungsi `function()`. Perhatikan contoh berikut.

```
a <- 5
b <- 3
c < a + b
c
```

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

Kemudian ketika ingin menggunakan nilai lain untuk `a` namun dengan nilai `b` yang sama, misalnya `a <- 2`, maka jalankan perintah seperti berikut.

```
a <- 2
c < a + b
c
```

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

Misalkan *script* diatas dibuat menjadi sebuah fungsi, misalnya dengan nama fungsinya adalah `sum_ab`. Fungsi tersebut memiliki 2 buah parameter atau argumen, yaitu `a` dan `b` berupa nilai numerik. Argumen `a` dan `b` harus diisi ketika memanggil fungsi tersebut. Fungsi `return()` dalam fungsi `sum_ab()` menentukan nilai yang akan dijadikan keluaran (output) dari fungsi `sum_ab()`.

```
sum_ab <- function(a, b){
     c <- a + b
     return(c)
}
```

Selanjutnya panggil fungsi tersebut dengan argumen yang diperlukan.

```
sum_ab(a =  5, b = 7)
```

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

Merubah nilai `a <- 2` dapat dilakukan dengan lebih mudah.

```
sum_ab(a =  2, b = 7)
```

```
## [1]  9
```

Bagaimana jika tidak ada argumen yang diberikan ketika memanggil fungsi tersebut? Dalam hal ini akan terjadi *error*. *Error* yang terjadi karena argumen pada fungsi ini adalah tipe argumen yang harus diisi atau tidak mempunyai nilai default.

```
sum_ab()
## Error in sum_ab() : argument "a" is missing, with no default
```

Apa itu nilai *default* argumen pada sebuah fungsi? Lebih mudahnya, akan dilakukan sedikit modifikasi pada fungsi `sum_ab()` yang sudah dibuat sebelumnya.

```
sum_ab <- function(a = 1, b = 2){
     c <- a + b
     return(c)
}
```

Fungsi `sum_ab()` sekarang memiliki nilai *default* untuk setiap argumennya. Argumen `a` memiliki nilai 1 dan `b` juga 2. Ketika memanggil fungsi `sum_ab()` tanpa menuliskan nilai untuk argumennya maka secara *default* nilai-nilai tersebut yang akan digunakan.

```
sum_ab()
```

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

```
sum_ab(a = 17, b = 9)
```

```
## [1] 26
```

2.10.3 *Install Package*

Package adalah sebuah kumpulan fungsi atau data yang dibuat untuk memudahkan proses pada *R* tanpa harus menuliskan ulang *script* yang dibutuhkan. Saat ini *package* menjadi bagian yang sangat penting bagi *Data Analyst* atau *Data Scientist* ketika menggunakan *R*. Karena dengan *package* tambahan (yang belum ada ketika install *R*) pekerjaan dalam mengolah data menjadi lebih efisien.

Untuk dapat menggunakan fungsi atau data dari sebuah *package* tambahan, diperlukan menginstal *package* tersebut terlebih dahulu. Salah satu contohnya sudah disampaikan pada bagian 2.10.1, yaitu package {nycflights13}. Selanjutnya pengguna akan diperlihatkan instalasi *package* {ggplot2}. *Package* ini sangat berguna untuk membuat visualisasi data di *R*. Untuk melihat *help* dari *package* ini pengguna dapat melakukannya dengan cara yang akan di bahas di bagian 2.11.

Install *package* di *R* sangat mudah dengan fungsi `install.packages("namapackage")` atau melalui menu Install pada tab **Packages** di *RStudio*. Yang perlu diperhatikan ketika menginstall *package* adalah koneksi internet, nama *package* dan *repository*-nya. Koneksi internet yang baik sangat dibutuhkan ketika install *package* untuk *R* mendownload *file package* tersebut. Selanjutnya nama *package* juga harus sesuai dari penulisannya, termasuk huruf kapitalnya. Misalkan pengguna ingin menginstall *package* {ggplot2}, maka pengguna harus menuliskannya dengan `install.packages("ggplot2")`. Jika penulisannya tidak sama maka package tersebut tidak akan diinstall. Perhatikan contoh berikut ini.

```
library("ggplot")
```

```
## Installing package into C:/Users/Lenovo/Documents/R/win-library/4.0
## (as lib is unspecified)
## Warning in install.packages :
##   package 'ggplot' is not available (for R version 4.0.2)
```

Contoh di atas adalah pemberitahuan ketika *package* yang ingin diinstall tidak tersedia. Ketika nama package yang dituliskan ada di repository maka akan ada *pop-up* download *file package*. Sekarang coba perhatikan contoh berikut ini.

```
library("ggplot2")
```

```
## Installing package into C:/Users/Lenovo/Documents/R/win-library/4.0
## (as lib is unspecified)
## trying URL 'http://cran.rstudio.com/bin/windows/contrib/4.0/ggplot2_3.3.2.zip'
## Content type 'application/zip' length 4067278 bytes (3.9 MB)
## downloaded 3.9 MB
##
## package 'ggplot2' successfully unpacked and MD5 sums checked
##
## The downloaded binary packages are in
##  C:\Users\Asus\AppData\Local\Temp\RtmpUd5mDA\downloaded_packages
```

Hal yang perlu di perhatikan selanjutnya adalah *repository package*. Ada beberapa *repository* yang digunakan sebagai tempat penyimpanan *package*. *Reposiroty* yang paling umum adalah *Comprehensive R Archieve Network* (CRAN). Ketika menggunakan fungsi `install.packages()` maka secara otomatis *package* yang akan diinstall berasal dari CRAN. Selain itu ada beberapa *repository* lain seperti GitHub, Bitbucket, Bioconductor dan lain-lain. Pada kesempatan ini hanya akan membahas repository GitHub saja.

Umumnya GitHub digunakan sebagai tempat untuk *repository package* yang belum ada di CRAN atau versi pengembangan (*development*) yang belum *submit* ke CRAN. Untuk dapat install *package* dari GitHub pengguna dapat menggunakan *package* {devtools} atau {remotes}. Disarankan untuk menggunakan *package* {devtools} karena dapat digunakan juga untuk membuat *package* jika dibutuhkan. Mungkin perlu install RTools versi yang sesuai dengan versi R pengguna jika menggunakan OS *Windows*.

Misalkan ingin install *package* {DataExplorer} dari Github. Yang perlu diketahui adalah *link/username* dari *repository package* tersebut. *Repository* GitHub *package* {DataExplorer} adalah **https://github.com/boxuancui/DataExplorer**. Argumen yang dibutuhkan adalah *username* dan nama *repository*. Username dari package ini adalah *boxuancui* dan *repository*-nya adalah DataExplorer. Perhatikan contoh berikut ini.

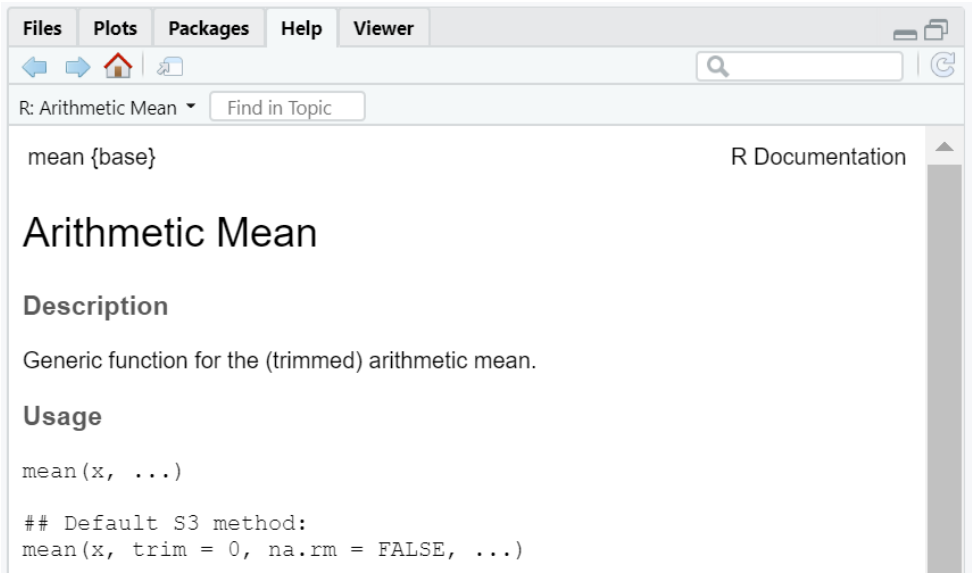
```
Install.packages("devtools")
devtools::install_github("boxuancui/DataExplorer")
```

2.11 Mencari *Help* Sebuah Fungsi

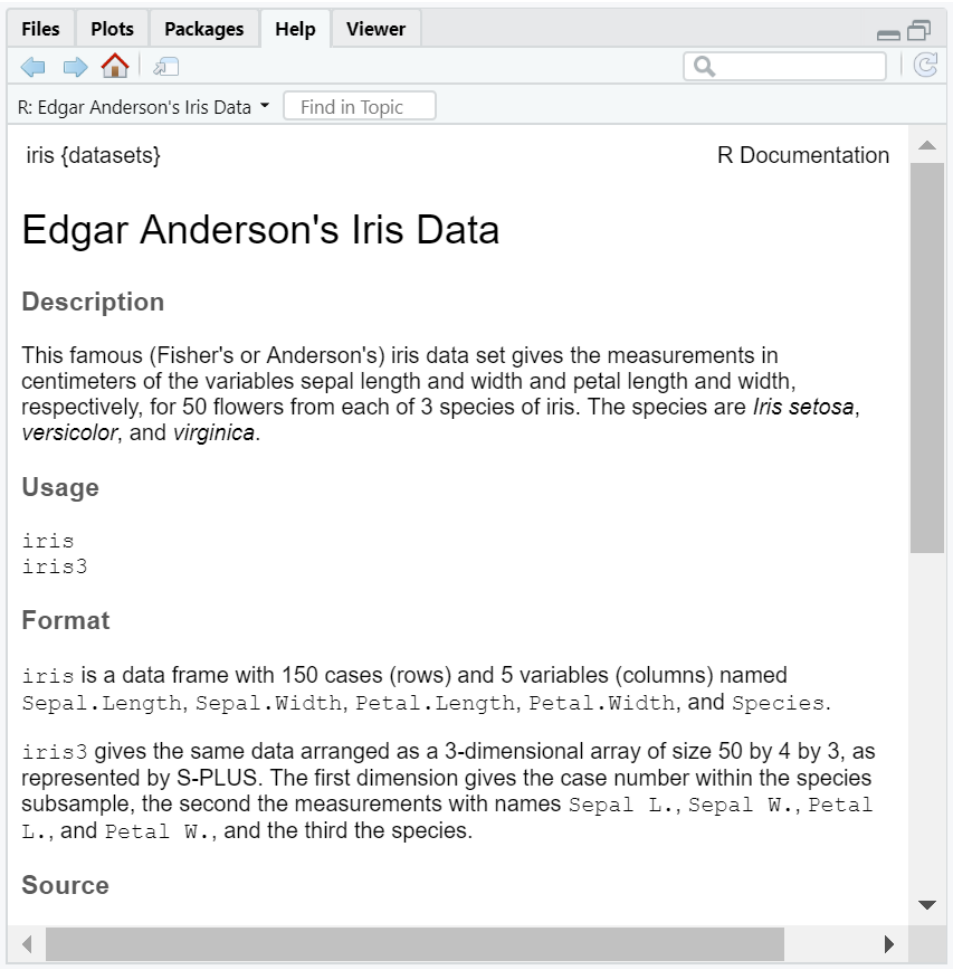
Ketika akan menggunakan sebuah fungsi di *R*, ada baiknya untuk mengetahui beberapa hal tentang fungsi tersebut. Yang harus diketahui dari sebuah fungsi paling tidak adalah untuk apa fungsi tersebut digunakan, apa saja argumen yang diperlukan, bagaimana penggunaannya dan output seperti apa yang dihasilkan.

Misalnya pengguna ingin menggunakan fungsi `mean()`, maka pengguna dapat mengetikkan `?mean` atau `help("mean")`. Jika menggunakan *RStudio* maka akan muncul halaman help dari fungsi `mean()` di tab **Help**.

Umumnya pada sebuah halaman *help* akan berisi informasi fungsi seperti dari *package* mana fungsi tersebut berasal. Pada gambar dibawah diketahui fungsi `mean()` berasal dari *package* {base}. Kemudian deskripsi singkat tentang fungsi (*Description*), penggunaan (*Usage*), penjelasan setiap argumen pada fungsi tersebut (*Arguments*), penjelasan lebih detail (*Value*/*Detail*, jika ada) dan contoh penggunaan fungsi (*Examples*, jika ada).



Hal ini juga berlaku jika ingin melihat halaman *help* dari sebuah data yang ada di *R*, baik itu dari *package* {base} ataupun package tambahan yang lain. Misalnya halaman *help* dari data `iris`. Ketikkan `?iris` pada *console RStudio* untuk menampilkan halaman help data `iris`. Dari gambar dibawah diketahui bahwa data `iris` berasal dari package {datasets}.


Files
Plots
Packages
Help
Viewer
R: Edgar Anderson's Iris Data
Find in Topic
iris {datasets}
R Documentation
Edgar Anderson's Iris Data
Description
This famous (Fisher's or Anderson's) iris data set gives the measurements in centimeters of the variables sepal length and width and petal length and width, respectively, for 50 flowers from each of 3 species of iris. The species are Iris setosa, versicolor, and virginica.
Usage
iris
iris3
Format
iris is a data frame with 150 cases (rows) and 5 variables (columns) named Sepal.Length, Sepal.Width, Petal.Length, Petal.Width, and Species.
iris3 gives the same data arranged as a 3-dimensional array of size 50 by 4 by 3, as represented by S-PLUS. The first dimension gives the case number within the species subsample, the second the measurements with names Sepal L., Sepal W., Petal L., and Petal W., and the third the species.
Source

BAB 3 | IMPORT DAN EXPORT DATA

Bab ini menjelaskan tentang fungsi dasar *R* mendapatkan lembar kerja (*spreedsheet*) ke dalam *R* dan menyimpan data yang telah diolah dalam *R* dalam bentuk `.txt`, `.csv` dan `.xlsx`. Sebelum itu, untuk mempermudah dalam melakukan *import* dan *export*, ada baiknya pengguna memahami fungsi `getwd()` dan `setwd()` terlebih dahulu.

Fungsi `getwd()` adalah fungsi yang digunakan untuk mengetahui lokasi kerja pengguna saat ini. Setelah mengetahui lokasi kerja saat ini, data yang akan di-*import* sebaiknya berada pada lokasi tersebut. Perhatikan contoh berikut ini.

```
getwd()
```

```
## [1] "D:/Buku Panduan R/Nama_Project"
```

Seperti yang dapat dilihat diatas, saat ini lokasi kerja berada pada `"D:/Buku Panduan R/Nama_Project"`. Apabila pengguna mengikuti bab 2 dengan baik maka pengguna pastinya mengetahui penyebab lokasi kerja berada pada lokasi tersebut.

Selanjutnya adalah fungsi `setwd()`. Fungsi `setwd()` berfungsi untuk mengganti lokasi kerja yang diinginkan dengan memasukkan alamat lokasi kerja yang diinginkan. Terkadang fungsi ini digunakan agar lokasi dari data yang akan di-*export* dapat diketahui bahwa akan tersimpan pada lokasi tersebut. Perhatikan contoh berikut ini.

```
setwd("C:/Users/asus/Documents")
```

Pada contoh diatas, perlu diketahui bahwa lokasi kerja saat ini berada di `"C:/Users/asus/Documents"` dan apabila melakukan *export* data maka data tersebut akan tersimpan pada lokasi tersebut.

3.1 *Import Tab Delimited Text File*

Apabila data yang digunakan dalam bentuk `.txt`, pengguna bisa dengan mudah melakukan *import* menggunakan fungsi dasar `read.table()`. Sebagai contoh, pengguna bisa melakukan *import* data yang berasal dari link yang dapat diakses langsung yaitu **https://s3.amazonaws.com/assets.datacamp.com /blog_assets/test.txt**. Perhatikan contoh berikut ini.

```
df <- read.table("https://s3.amazonaws.com/assets.datacamp.com/blog_assets/test.txt")
df
```

```
##    V1 V2 V3
##  1  1  6  a
##  2  2  7  b
##  3  3  8  c
##  4  4  9  d
##  5  5 10  e
```

Data dari link diatas terlihat seperti dibawah ini.

```
1   6   a
2   7   b
3   8   c
4   9   d
5   10  e
```

Namun, terkadang data bentuk `.txt` yang didapat memiliki pemisah dalam bentuk simbol seperti tanda koma (,) atau titik (.). Untuk menangani hal ini, fungsi `read.table()` menyediakan fitur `sep =` yang berguna untuk memisahkan setiap elemen data berdasarkan karakter pemisah yang dimasukkan. Sebagai contoh, perhatikan data berikut ini.

```
1,6,a
2,7,b
3,8,c
4,9,d
5,10,e
```

Data diatas adalah data yang sama dengan data sebelumnya, namun pemisah antar elemen data adalah tanda koma (,). Perlu diketahui bahwa data diatas telah disimpan sebelumnya dengan `sites.txt` pada `Documents` dimana lokasi kerja tersebut sebelumnya telah diganti. Sehingga saat melakukan *import* hanya perlu memasukkan nama *file* dan formatnya. Perhatikan contoh berikut ini.

```
df <- read.table("sites.txt")
```

```
##        V1
##  1  1,6,a
##  2  2,7,b
##  3  3,8,c
##  4  4,9,d
##  5 5,10,e
```

Apabila data seperti diatas di-*import* secara langsung maka hasilnya akan terlihat seperti output diatas. Data yang seharusnya memiliki 3 kolom, namun disimpan dalam 1 kolom saja. Hal ini disebabkan fungsi `read,table()` tidak mengetahui tanda pemisah dari setiap elemen datanya. Oleh karena itu, fitur `sep =` diperlukan untuk menyelesaikan masalah ini. Perhatikan contoh berikut ini.

```
df <- read.table("sites.txt", sep = ",")
```

```
##    V1 V2 V3
##  1  1  6  a
##  2  2  7  b
##  3  3  8  c
##  4  4  9  d
##  5  5 10  e
```

Akhirnya data dapat dikenali dengan benar setelah "memberitahu" fungsi `read.table()` bahwa elemen data dipisahkan oleh tanda koma (,).

Masalah lain yang sering ditemukan adalah data yang akan di-*import* sudah memiliki nama pada setiap kolomnya atau lebih dikenal dengan istilah **HEADER**.

Sama halnya dengan kasus pemisah elemen data diatas, fungsi `read.table()` memiliki fitur `header =` yang berguna untuk memberikan informasi pada fungsi `read.table()` bahwa data yang akan di-*import* sudah memiliki nama kolom atau belum. Nilai *default* dari fitur `header` adalah `FALSE`, sehingga jika tidak diubah ke `TRUE` maka nama kolom akan menjadi data baris pertama. Perhatikan contoh berikut ini.

```
df <- read.table("sites.txt", sep = ",", header = TRUE)
```

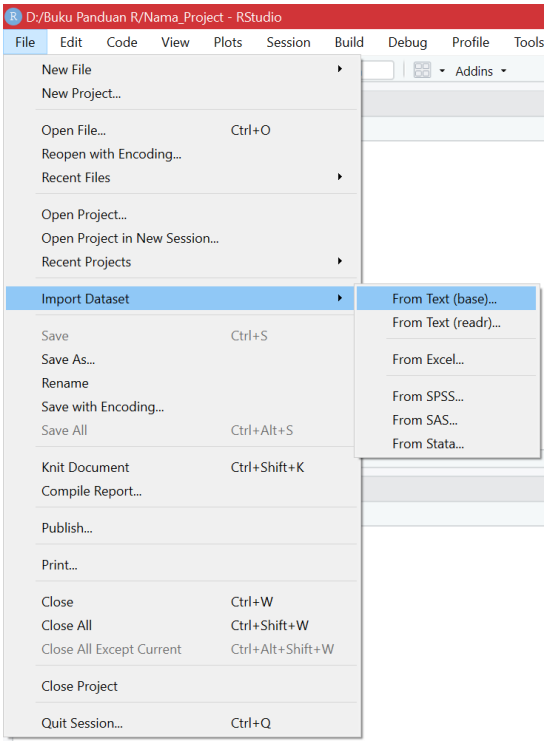
```
##    No Angka Huruf
## 1  1     6     a
## 2  2     7     b
## 3  3     8     c
## 4  4     9     d
## 5  5    10     e
```

Perlu diketahui bahwa data `sites.txt` sebelumnya telah ditambahkan nama kolom sehingga data tersebut terlihat seperti dibawah ini.

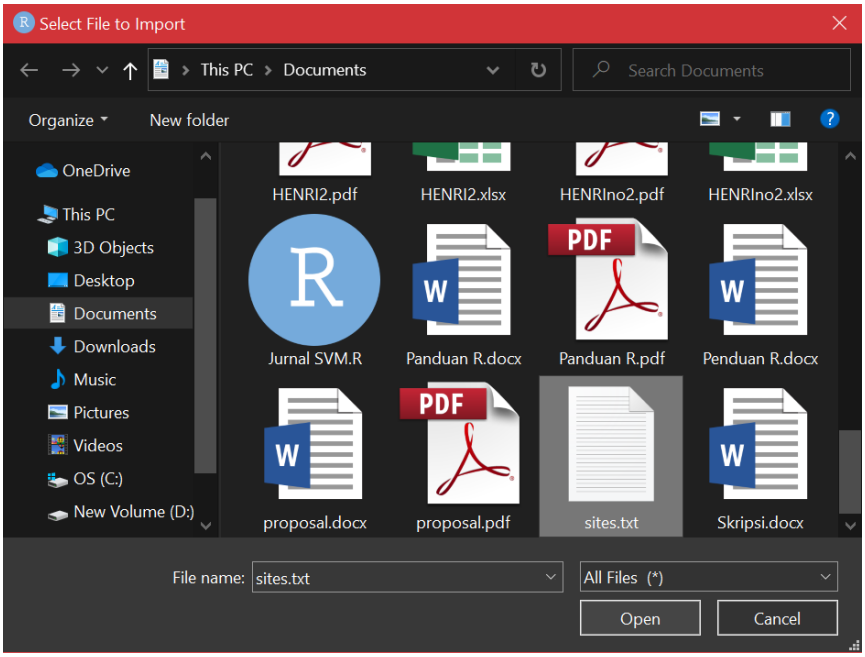
```
No,Angka,Huruf
1,6,a
2,7,b
3,8,c
4,9,d
5,10,e
```

Sekarang melakukan *import* data format .txt menggunakan fungsi read.table() melalui *console R* telah dijelaskan. Selanjutnya akan dijelaskan cara melakukan *import* melalui menu *File*. Berikut langkah-langkah yang harus dilakukan.

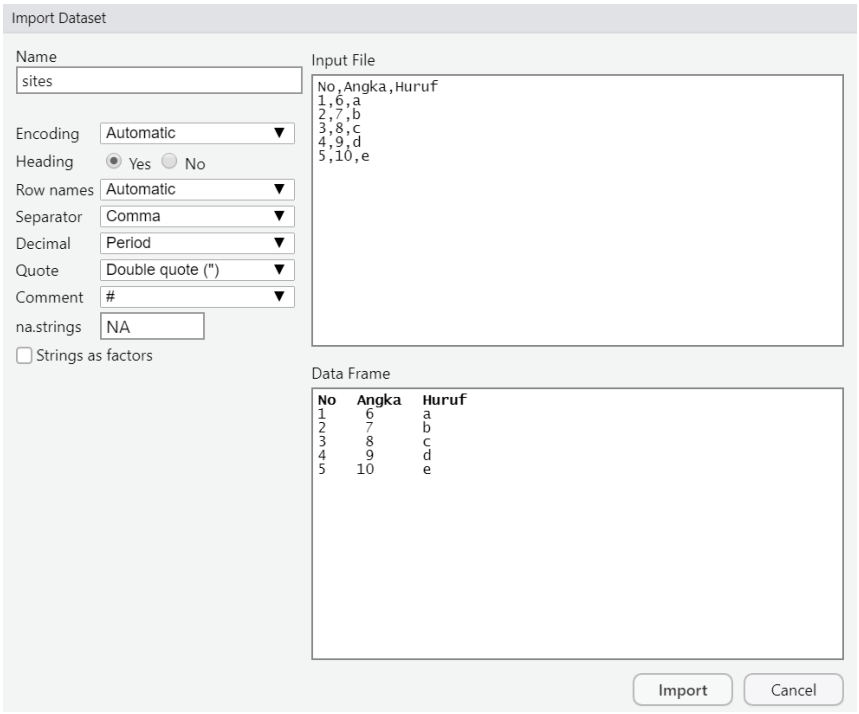
1. Klik menu *File >> Import Dataset >> From Text (base)…*



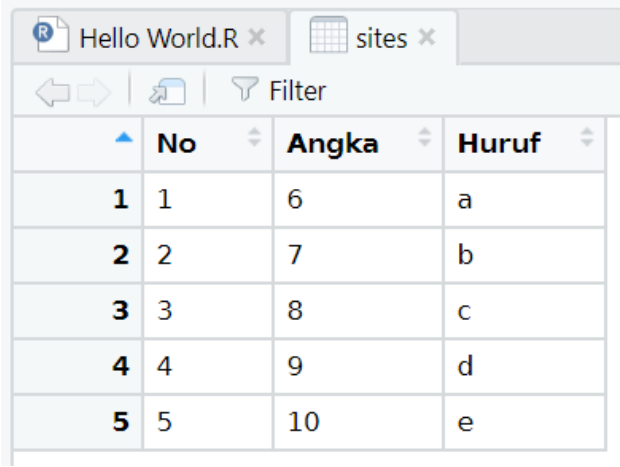
2. Kemudian pilih data yang ingin di-*import* pada lokasi dimana data disimpan.



3. Atur nama objek, pemisah, dan sebagainya. Perhatikan pada bagian ***Data Frame*** sebelum *import*. Apabila data yang akan di-*import* sudah terlihat sesuai dengan keinginan, silahkan klik *Import*.



4. Maka data yang di-*import* akan secara otomatis akan ditampilkan menggunakan fungsi `View()`.



	No	Angka	Huruf
1	1	6	a
2	2	7	b
3	3	8	c
4	4	9	d
5	5	10	e

Perlu diingat bahwa cara ini juga dapat digunakan pada data format `.csv` dikarenakan fungsi yang digunakan adalah `read.csv()`. Selain itu, data yang

sudah di-*import* secara otomatis tersimpan ke dalam objek *data frame*, namun objek tersebut dapat ditransfomasi ke objek lain.

3.2 *Import* CSV dan EXCEL

Sama halnya dengan data dengan format `.txt`, data dengan format `.csv` dan `.xlsx` dapat di-*import* dengan cara yang sama. Yang membedakan adalah, data dengan format `.csv` menggunakan fungsi `read.csv()` dan data dengan format `.xlsx` menggunakan fungsi `read_excel()`.

Untuk kasus data dengan format `read.csv()`, caranya tidak berbeda dengan data dengan format `.txt`. Perhatikan contoh berikut ini.

```
df <- read.csv("sites2.csv", header = TRUE)
```

```
##    No Angka Huruf
##  1  1     6     a
##  2  2     7     b
##  3  3     8     c
##  4  4     9     d
##  5  5    10     e
```

Perlu diketahui bahwa fungsi `read.csv()` juga memiliki fitur `sep =` dan `header =` sehingga tidak perlu khawatir dengan masalah peng-*import*-an data yang tidak benar. Dan cara *import* melalui menu *File* untuk data dengan format `.csv` sama dengan data dengan format `.txt`.

Pada kasus penggunaan fungsi `read_excel()`, diperlukan untuk mendownload *package* `readxl` dan `Rcpp`, dan mengaktifkan *package* `readxl` menggunakan fungsi `library()`. Perhatikan contoh berikut ini.

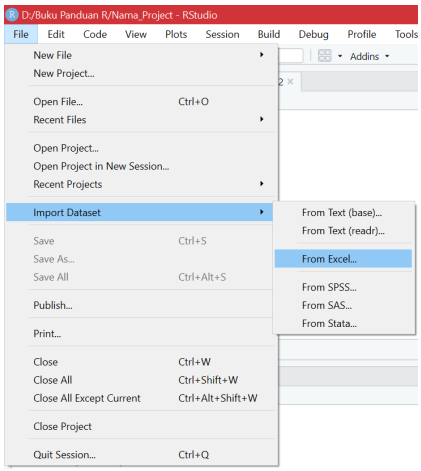
```
library(readxl)
sites3 <- read_excel("sites3.xlsx")
```

```
# A tibble: 5 x 3
```

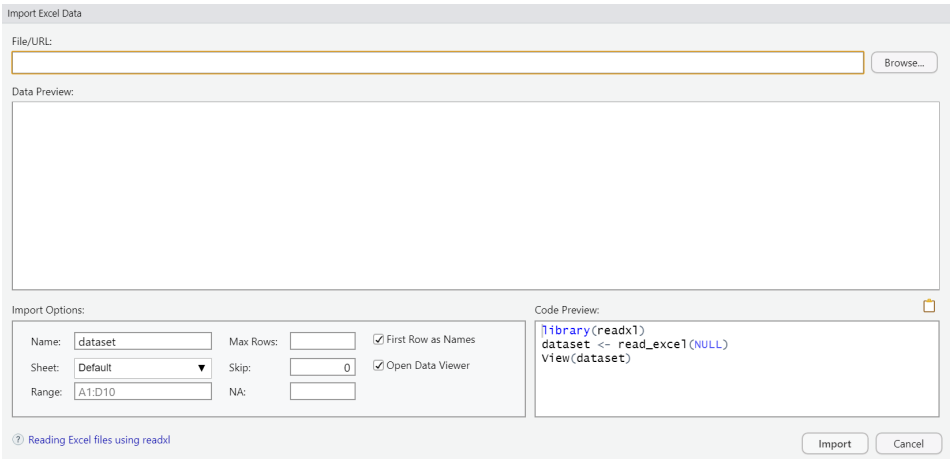
```
     No Angka Huruf
  <dbl> <dbl> <chr>
1     1     6 a
2     2     7 b
3     3     8 c
4     4     9 d
5     5    10 e
```

Selanjutnya akan dijelaskan cara melakukan *import* melalui menu *File* karena cara untuk data format `.xls` dan `.txt` sedikit berbeda. Berikut langkah-langkah yang harus dilakukan.

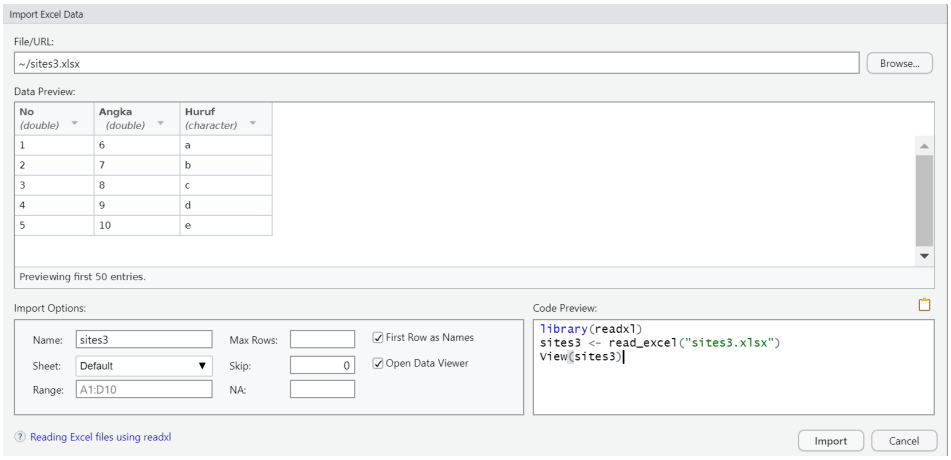
1. Klik menu *File >> Import Dataset >> From Excel…*



2. Klik *Browse..* untuk memilih data yang ingin di-*import* dan pilih data pada lokasi dimana data disimpan.

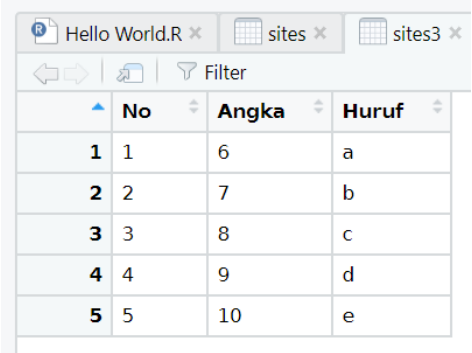


3. Atur nama objek, lembar kerja, dan sebagainya. Perhatikan pada bagian ***Data Preview*** sebelum *import*. Apabila data yang akan di-*import* sudah terlihat sesuai dengan keinginan, silahkan klik *Import*.



Perhatikan bahwa dibawah nama kolom terdapat tipe data yang dapat diganti apabila tipe data tidak sesuai dengan yang seharusnya.

4. Selanjutnya data yang di-*import* akan secara otomatis akan ditampilkan menggunakan fungsi `View()`.



	No	Angka	Huruf
1	1	6	a
2	2	7	b
3	3	8	c
4	4	9	d
5	5	10	e

Selain format data diatas, masih banyak format data yang dapat di-*import* ke dalam *R* sehingga dapat dikatakan bahwa *R* sangat fleksibel digunakan dalam menangani masalah pada data atau melakukan analisis pada data khususnya statistik.

3.3 *Export* Data

Setelah mengolah atau melakukan analisis pada data, *R* juga menyediakan fungsi untuk menyimpan objek (dalam hal ini data) dalam beberapa format seperti *Excel* (xlsx), *Tab Delimited Text File* (txt), bahkan dalam format untuk aplikasi pengolah data lainnya seperti SPSS, SAS, dan Stata.

Karena sebelumnya telah dijelaskan mengenai *import*, maka fungsi-fungsi yang digunakan untuk melakukan *export* seharusnya bisa dipahami dengan melihat beberapa contoh yang akan diberikan berikut ini.

1. Tab Delimited Text File

```
write.table(nama_objek, "nama_data.txt", sep = "\t")
```

`sep = "\t"` artinya memisahkan setiap elemen data dengan tabulasi

2. Excel Spreedsheet

```
library(xlsx)
write.xlsx(nama_objek, "nama_data.xlsx")
```

Perhatikan bahwa dibutuhkan *package* `xlsx` untuk menggunakan fungsi *export* `write.xlsx()`.

3. SPSS

```
library(foreign)
write.foreign(nama_objek, "nama_data.txt", "nama_data.sps")
```

Perhatikan bahwa dibutuhkan *package* `foreign` untuk menggunakan fungsi *export* `write.foreign()`.

4. SAS

```
library(foreign)
write.foreign(nama_objek, "nama_data.txt", "nama_data.sas",
              package = "SAS")
```

5. Stata

```
library(foreign)
write.dta(nama_objek, "nama_data.dta")
```

Perhatikan bahwa dibutuhkan *package* `foreign` untuk menggunakan fungsi *export* `write.dta(`

BAB 4 | VISUALISASI DATA

Pada bab ini akan dijelaskan mengenai visualisasi data menggunakan *R*. Penyajian data dalam bentuk grafik sangat diperlukan dalam berbagai hal sehingga *R* bisa menjadi pilihan yang bagus. Hal ini dikarenakan *R* memiliki banyak *package-package* yang menyediakan fungsi-fungsi yang dapat memvisualisasikan data dalam bentuk yang lebih menarik.

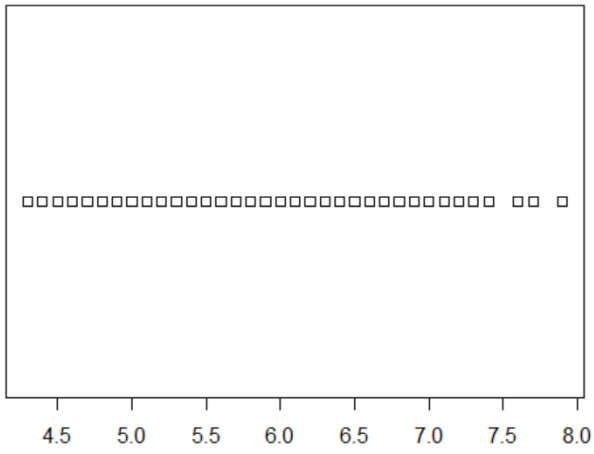
4.1 Dasar Grafik di *R*

R menyediakan fungsi dasar untuk menggambarkan sebuah data seperti membuat bagan garis (*Strip Charts*), histogram, boxplot, scatter plot, dan plot kuantil normal (*Normal QQ Plots*). Berikut cara membuat grafik-grafik tersebut dan fungsi-fungsi yang digunakan dalam menyajikannya.

4.1.1 Bagan Garis (*Strip Charts*)

Bagan garis (*Strip Charts*) adalah jenis plot paling dasar yang disediakan *R*. Fungsi yang digunakan adalah `stripchart()`. Fungsi ini memplot data dalam secara terurut sepanjang garis dengan setiap titik data direpresentasikan sebagai sebuah kotak. Sebagai contoh, digunakan dataset `iris` yang ada dalam *R*, dan mengambil kolom pertama dari dataset `iris` atau `iris$Sepal.Length`. Perhatikan contoh berikut ini.

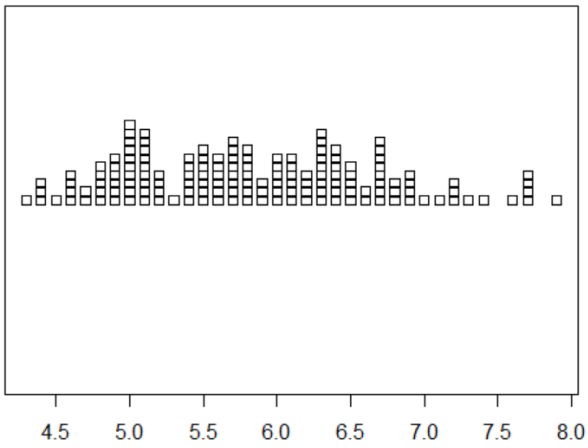
```
stripchart(iris$ Sepal.Length)
```



Gambar 4.1 Bagan Garis

Seperti yang terlihat diatas, tidak ada judul atau label untuk sumbunya. Ini hanya menunjukkan bagaimana data terlihat jika diletakkan sepanjang satu baris dan memberikan tanda kotak disetiap titiknya. Apabila ingin melihat titik mana yang sering diulang, pengguna dapat menyajikannya dalam bentuk tumpukkan menggunakan `method = "stack"`. Perhatikan contoh dibawah ini.

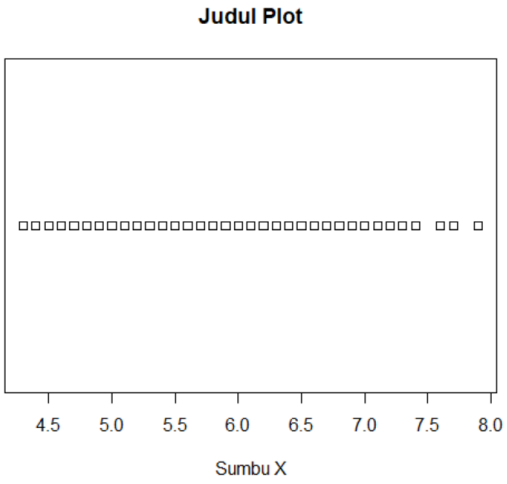
```
stripchart(iris$Sepal.Length, method = "stack")
```



Gambar 4.2 Bagan Garis - *Stack*

Selain *stack*, fungsi `stripchart()` juga menyediakan fitur `method = "jitter"` yang bisa pengguna ekplorasi dan pilih sesuai dengan kebutuhan. Apabila ingin menambahkan judul dan label untuk sumbunya, pengguna bisa menggunakan perintah berikut setelah plot dibuat.

```
title('Judul Plot', xlab = 'Sumbu X')
```

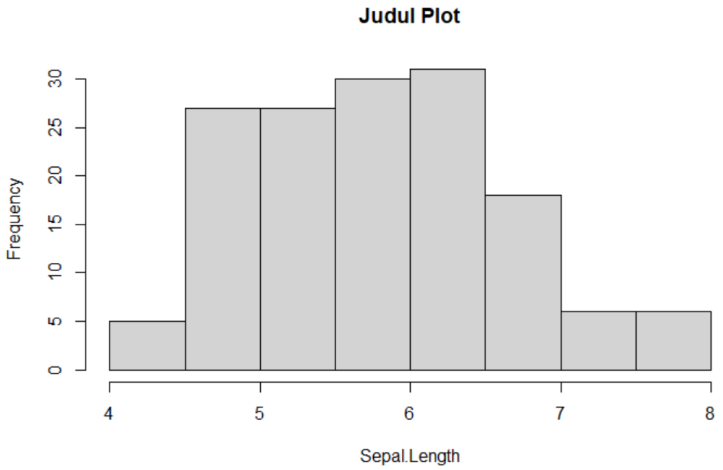


Gambar 4.3 Bagan Garis (menambahkan nama sumbu *X*)

4.1.2 Histogram

Histogram adalah plot yang sangat umum dan cukup sering ditemukan. Plot ini menggambarkan frekuensi yang muncul pada data dalam rentang tertentu. Sebagai contoh, digunakan dataset `iris` yang ada dalam *R*, dan mengambil kolom pertama dari dataset `iris` atau `iris$Sepal.Length`. Perhatikan contoh berikut ini.

```
hist(iris$Sepal.Length, main = "Judul Plot", xlab = "Sepal.Length")
```



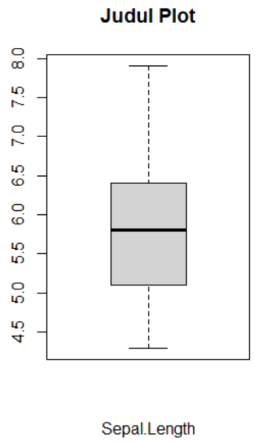
Gambar 4.4 Histogram

Seperti yang terlihat diatas, *R* akan secara otomatos menghitung interval yang digunakan. Ada banyak opsi untuk menentukan interval. Namun pada bab ini hanya menjelaskan cara viasualisasi data menggunakan fungsi `hist()`. Jadi, apabila pengguna ingin tahu lebih banyak tentang fungsi `hist()` ini, silahkan gunakan fungsi `help()` seperti yang telah dijelaskan pada bab 2.1.11.

4.1.3 Boxplot

Boxplot memberikan tampilan grafis dari median, kuartil, maksimum, dan minimum dari dataset. Sebagai contoh, digunakan dataset `iris` yang ada dalam *R*, dan mengambil kolom pertama dari dataset `iris` atau `iris$Sepal.Length`. Perhatikan contoh berikut ini.

```
boxplot(iris$Sepal.Length, main = "Judul Plot", xlab = "Sepal.Length")
```



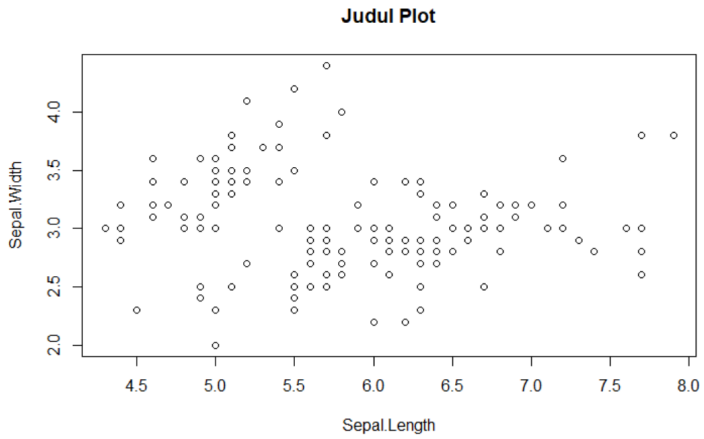
Gambar 4.5 Boxplot

Ada banyak opsi yang terdapat pada fungsi `boxplot()`. Namun pada bab ini hanya menjelaskan cara viasualisasi data menggunakan fungsi `boxplot()`. Jadi, apabila pengguna ingin tahu lebih banyak tentang fungsi `boxplot()` ini, silahkan gunakan fungsi `help()` seperti yang telah dijelaskan pada bab 2.1.11.

4.1.4 Scatter Plot

Scatterplot memberikan tampilan grafis dari hubungan antara dua variabel (kolom). Pada contoh kali ini digunakan 2 variabel (kolom) yang ada pada `iris` yang ada dalam *R* yaitu `iris$Sepal.Length` dan `iris$Sepal.Width`. Perhatikan contoh berikut ini.

```
plot(iris$Sepal.Length, iris$Sepal.Width, main = "Judul Plot",
     xlab = "Sepal.Length", ylab = "Sepal.Width")
```



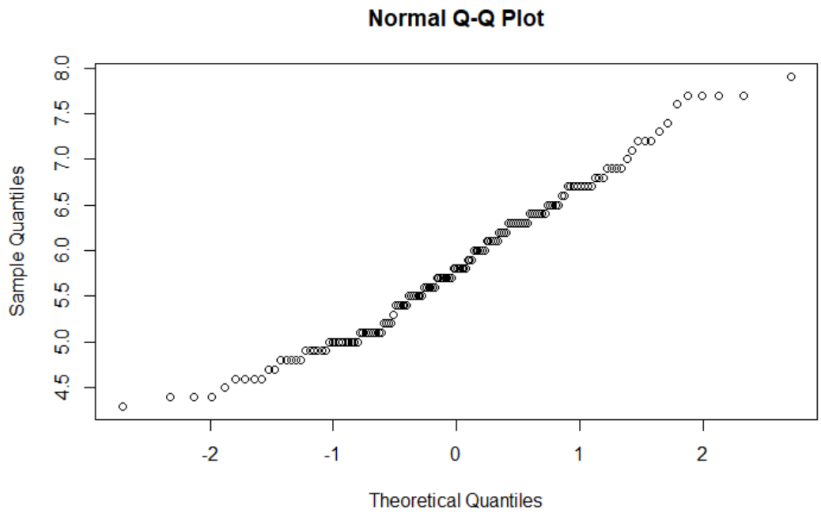
Gambar 4.6 Scatter Plot

Ada banyak opsi yang terdapat pada fungsi `plot()`. Namun pada bab ini hanya menjelaskan cara viasualisasi data menggunakan fungsi `plot()`. Jadi, apabila pengguna ingin tahu lebih banyak tentang fungsi `plot()` ini, silahkan gunakan fungsi `help()` seperti yang telah dijelaskan pada bab 2.1.11.

4.1.5 Plot Kuantil Normal (*Normal QQ Plots*)

Jenis plot terakhir yang akan dilihat adalah plot kuantil normal (*Normal QQ Plots*). Plot ini digunakan untuk menentukan apakah data hampir terdistribusi secara normal. Pengguna tidak dapat memastikan secara pasti bahwa data terdistribusi secara normal, tetapi dapat diketahui jika tidak terdistribusi secara normal. Sebagai contoh, digunakan dataset `iris` yang ada dalam *R*, dan mengambil kolom pertama dari dataset `iris` atau `iris$Sepal.Length`. Perhatikan contoh berikut ini.

```
qqnorm(iris$Sepal.Length)
```



Gambar 4.7 Plot Kuantil Normal

Ada banyak opsi yang terdapat pada fungsi `qqnorm()`. Namun pada bab ini hanya menjelaskan cara viasualisasi data menggunakan fungsi `qqnorm()`. Jadi, apabila pengguna ingin tahu lebih banyak tentang fungsi `qqnorm()` ini, silahkan gunakan fungsi `help()` seperti yang telah dijelaskan pada bab 2.1.11.

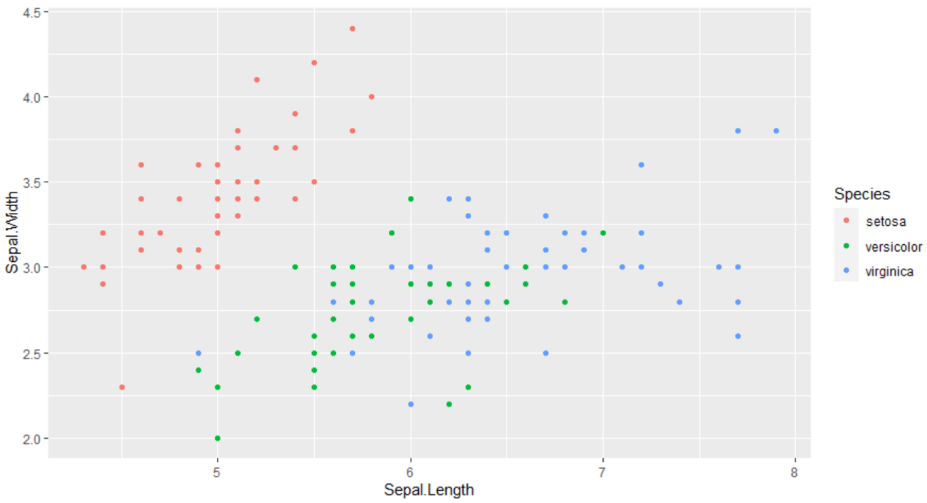
4.2 Visualisasi *Package* {ggplot2}

Package `ggplot2` merupakan *package* yang diciptakan oleh Hadley Wickham (2009) dengan kelebihannya dalam pembuatan gambar yang elegan dan kompleks. Popularitas `ggplot2` di komunitas R tidak diragukan lagi. `ggplot2` memungkinkan untuk membuat grafik yang merepresentasikan data numerik dan kategorik baik univariat maupun multivariat secara simultan. Pengelompokan yang dapat diwakili oleh warna, simbol, ukuran dan ketebalan point. `ggplot2` mempunyai banyak fungsi dan pilihan untuk plot yang akan ditampilkan. Berikut adalah *statement* dari fungsi `ggplot()` yang umum digunakan:

```
library(ggplot2)
ggplot(nama_objek, aes(nama_varibel1, nama_variabel2,
       color = vektor_kelas)) + geom_point()
```

Sama halnya dengan penggunaan fungsi yang ada dalam sebuah *package*, diperlukan untuk mengaktifkan *package* tersebut terlebih dahulu sebelum menggunakan fungsinya. Agar lebih memahami fungsi `ggplot()`, perhatikan contoh pembuat *scatterplot* pada data `iris` untuk variabel `Sepal.Length` dan `Sepal.Width` dengan memberikan warna berdasarkan variabel `Species`.

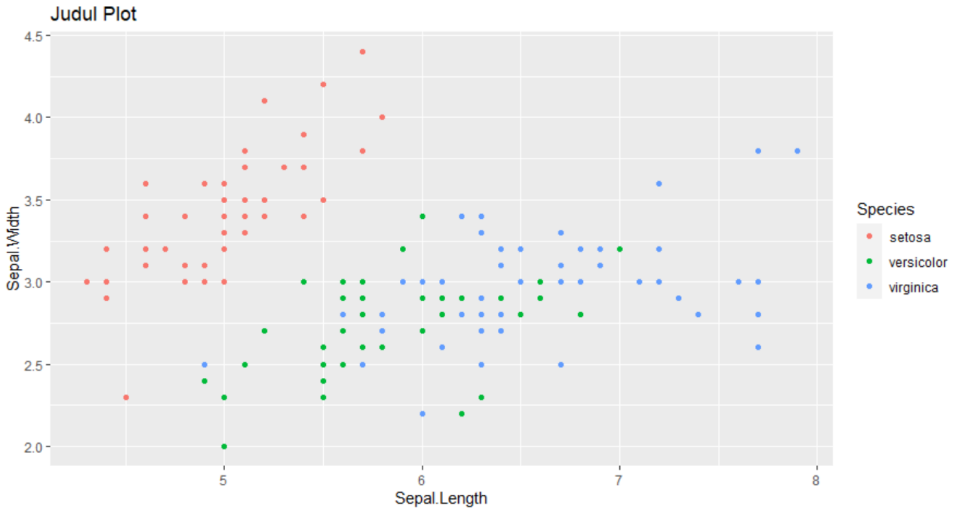
```
ggplot(iris, aes(Sepal.Length, Sepal.Width, color = Species)) +
       geom_point()
```



Gambar 4.8 Scatter Plot - *Package* `ggplot`

Apabila dibandingkan dengan plot pada subbab sebelumnya, plot yang dihasil kan fungsi `ggplot()` terlihat lebih menarik. Inilah kelebihan dari *package* `ggplot2` menyediakan berbagai macam plot dengan gambar yang elegan dan kompleks. Untuk menambah judul pada plot, lengkapi perintah diatas seperti berikut ini.

```
ggplot(iris, aes(Sepal.Length, Sepal.Width, color = Species)) +
       geom_point() + ggtitle("Judul Plot")
```



Gambar 4.9 Scatter Plot dengan Judul - *Package* `ggplot`

Pada dasarnya begitulah cara menggunakan salah satu fungsi yang ada pada *package* `ggplot2`. Yang dijelaskan pada bab ini hanya sebagian kecil dari banyaknya cara menyajikan data dalam bentuk grafik. Jadi, untuk lebih memahaminya, pengguna perlu melakukan percobaan sendiri dengan bantuan fungsi `help()` atau mencari referensi yang ada di internet.

BAB 5 | POHON KEPUTUSAN DAN *RANDOM FOREST*

Bab ini akan ditunjukkan bagaimana membangun model prediktif pada data mining menggunakan *package* `party`, `rpart`, dan `randomForest`. Dimulai dengan membangun pohon keputusan menggunakan *package* `party` dan menggunakan pohon yang dibangun untuk klasifikasi, diikuti dengan cara lain membangun pohon keputusan menggunakan *package* `rpart`. Setelah itu, memperlihatkan contoh dalam melatih model *random forest* menggunakan *package* `randomForest`.

5.1 Pohon Keputusan dengan *Package* `party`

Bagian ini menunjukkan bagaimana membangun pohon keputusan pada data `iris` menggunakan fungsi `ctree()` dalam *package* `party`. Variabel `Sepal.Length`, `Sepal.Width`, `Petal.Length`, dan `Petal.Width` digunakan untuk memprediksi `Species` dari bunga. Dalam *package* `party`, fungsi `ctree()` membangun pohon keputusan, dan `predict()` membuat prediksi untuk data baru.

Sebelum melakukan proses pembuatan model, data `iris` dibagi menjadi dua kumpulan data: `training` (70%) dan `test` (30%).

```
> str(iris)

'data.frame':     150 obs. of  5 variables:
 $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1

> set.seed(1234)
> ind <- sample(2, nrow(iris), replace=TRUE, prob=c(0.7, 0.3))
> trainData <- iris[ind==1,]
> testData <- iris[ind==2,]
```

Selanjutnya memuat *package* `party`, membangun pohon keputusan, dan melakukan pengecekkan pada hasil prediksi. Fungsi `ctree()` memerlukan beberapa paramater, seperti `MinSplit`, `MinBusket`, `MaxSurrogate` dan `MaxDepth`, untuk mengontrol pelatihan pohon keputusan. Dibawah akan

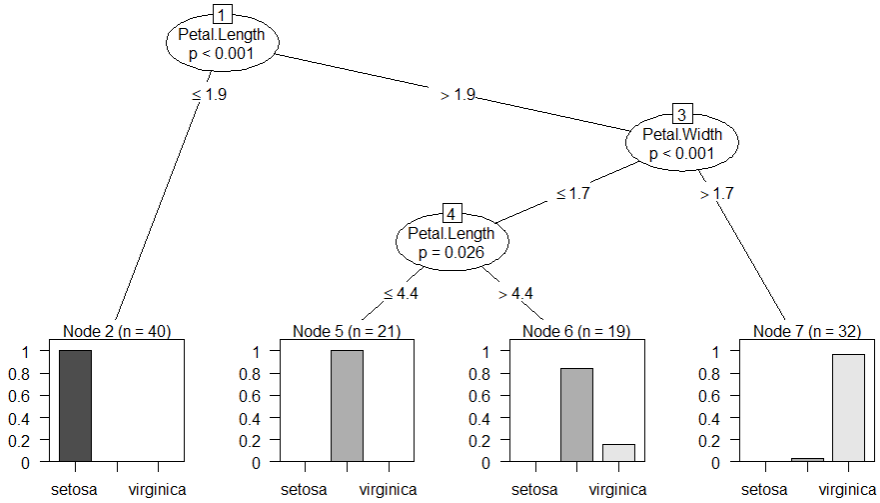
digunakan pengaturan default untuk membangun pohon keputusan. Pada sintaks dibawah, `myFormula` menjelaskan bahwa `Species` adalah variabel target dan semua variabel lain adalah variabel bebas.

```
> library(party)
> myFormula <- Species ~ Sepal.Length + Sepal.Width + Petal.Length
+ Petal.Width
> iris_ctree <- ctree(myFormula, data=trainData)
> # mengecek hasil prediksi
> table(predict(iris_ctree), trainData$Species)

            setosa versicolor virginica
  setosa         40          0         0
  versicolor      0         37         3
  virginica       0          1        31
```

Setelah itu, akan ditunjukkan hasil dari pohon keputusan dengan mencetak aturan dan melakukan plot pada pohon.

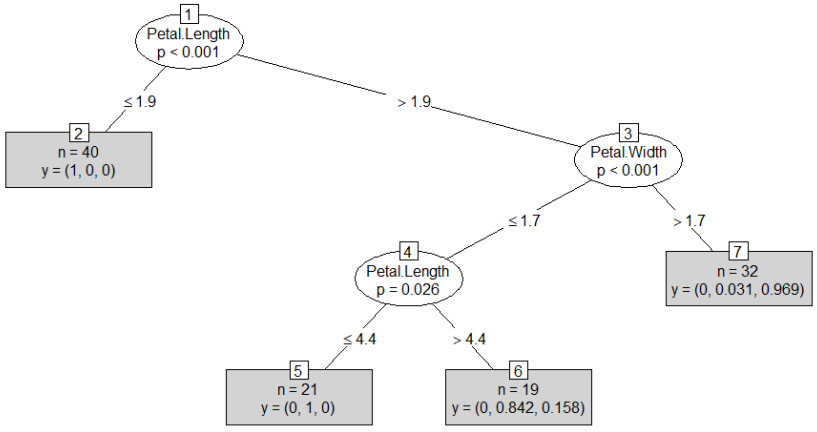
```
> print(iris_ctree)

       Conditional inference tree with 4 terminal nodes

Response:  Species
Inputs:  Sepal.Length, Sepal.Width, Petal.Length, Petal.Width
Number of observations:  112

1) Petal.Length <= 1.9; criterion = 1, statistic = 104.643
  2)*  weights = 40
1) Petal.Length > 1.9
  3) Petal.Width <= 1.7; criterion = 1, statistic = 48.939
    4) Petal.Length <= 4.4; criterion = 0.974, statistic = 7.397
      5)*  weights = 21
    4) Petal.Length > 4.4
      6)*  weights = 19
  3) Petal.Width > 1.7
    7)*  weights = 32
```

```
> plot(iris_ctree)
```



Gambar 5.1 Pohon Keputusan

```
> plot(iris_ctree, type="simple")
```



Gambar 5.2 Pohon Keputusan sederhana (*Simple Style*)

Pada Gambar 5.1, *barplot* setiap *leaf node* memperlihatkan probabilitas dari intens masuk ke dalam tiga spesies. Pada Gambar 7.2 memperlihatkannya sebagai "y" pada *leaf node*. Sebagai contoh, *node* 2 dilabeli dengan "$n = 40, y = (1,0,0)$", yang berarti bahwa didalamnya mengandung 40 intens *training* dan semuanya termasuk dalam kelas pertama "setosa".

Setelah itu, pohon yang telah dibangun perlu diuji pada data *test*.

```
> # prediksi pada data test
> testPred <- predict(iris_ctree, newdata = testData)
> table(testPred, testData$Species)
```

```
testPred     setosa versicolor virginica
  setosa         10          0         0
  versicolor      0         12         2
  virginica       0          0        14
```

Versi sekarang dari `ctree()` tidak dapat menangani nilai hilang (*missing value*) dengan baik, sehingga intens dengan nilai hilang terkadang masuk ke sisi kiri pohon dan terkadang masuk ke sisi kanan pohon. Hal ini kemungkinan disebabkan oleh *surrogate rules*.

Masalah lainnya adalah, ketika sebuah variabel ada dalam data *training* dan dimasukkan dalam fungsi `ctree()`, namun tidak muncul dalam pohon keputusan yang telah dibangun, data *testing* harus mempunyai variabel itu untuk dilakukan prediksi. Jika tidak, maka hasil dari fungsi `predict()` akan mengalami kegagalan (*error*). Selain itu, jika tingkatan nilai variabel kategorik pada data *testing* berbeda dengan data *training*, maka akan terjadi *error* juga pada saat membuat prediksi pada data *testing*. Selain itu, jika tingkat nilai variabel kategorikal pada data uji berbeda dengan data latih, maka akan gagal juga untuk membuat prediksi pada data uji. Salah satu cara untuk mengatasi masalah diatas adalah, setelah membangun pohon keputusan, menggunakan `ctree()` untuk membangun pohon keputusan baru dengan data yang hanya berisi variabel-variabel yang ada di pohon sebelumnya, dan secara eksplisit mengatur tingkat variabel kategorik pada data *testing* ke tingkat variabel terkait pada data *training*.

5.2 Pohon Keputusan dengan *Package* `rpart`

Package `rpart` digunakan pada bagian ini untuk membangun pohon keputusan pada data `bodyfat`. Fungsi `rpart()` digunakan untuk membangun pohon keputusan dan pohon dengan nilai *error* prediksi terendah akan dipilih. Setelah itu, pohon tersebut akan diaplikasikan pada data baru untuk membuat prediksi menggunakan fungsi `predict()`. Pertama, memuat data bodyfat dan melihat rincian dari data.

```
> data("bodyfat", package = "TH.data")
```

```
> dim(bodyfat)
[1] 71 10
> attributes(bodyfat)

$names
 [1] "age"          "DEXfat"       "waistcirc"    "hipcirc"
 [5] "elbowbreadth" "kneebreadth"  "anthro3a"     "anthro3b"
 [9] "anthro3c"     "anthro4"
$row.names
 [1] "47"  "48"  "49"  "50"  "51"  "52"  "53"  "54"  "55"  "56"
[11] "57"  "58"  "59"  "60"  "61"  "62"  "63"  "64"  "65"  "66"
[21] "67"  "68"  "69"  "70"  "71"  "72"  "73"  "74"  "75"  "76"
[31] "77"  "78"  "79"  "80"  "81"  "82"  "83"  "84"  "85"  "86"
[41] "87"  "88"  "89"  "90"  "91"  "92"  "93"  "94"  "95"  "96"
[51] "97"  "98"  "99"  "100" "101" "102" "103" "104" "105" "106"
[61] "107" "108" "109" "110" "111" "112" "113" "114" "115" "116"
[71] "117"
$class
[1] "data.frame"

> bodyfat[1:5,]

   age DEXfat waistcirc hipcirc elbowbreadth kneebreadth anthro3a
47  57  41.68     100.0   112.0          7.1         9.4     4.42
48  65  43.29      99.5   116.5          6.5         8.9     4.63
49  59  35.41      96.0   108.5          6.2         8.9     4.12
50  58  22.79      72.0    96.5          6.1         9.2     4.03
51  60  36.42      89.5   100.5          7.1        10.0     4.24
   anthro3b anthro3c anthro4
47     4.95     4.50    6.13
48     5.01     4.48    6.37
49     4.74     4.60    5.82
50     4.48     3.91    5.66
51     4.68     4.15    5.91
```

Selanjutnya, data dibagi menjadi data latih (*training*) dan data uji (*testing*), dan pohon keputusan dibangun pada data *training*.

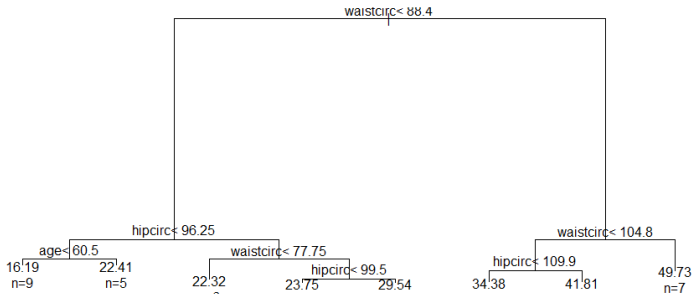
```
> set.seed(1234)
> ind <- sample(2, nrow(bodyfat), replace=TRUE, prob=c(0.7, 0.3))
> bodyfat.train <- bodyfat[ind==1,]
> bodyfat.test <- bodyfat[ind==2,]
> # melatih pohon keputusan
> library(rpart)
> myFormula <- DEXfat ~ age + waistcirc + hipcirc + elbowbreadth +
+     kneebreadth
> bodyfat_rpart <- rpart(myFormula, data = bodyfat.train, control
+     = rpart.control(minsplit = 10))
```

```
> attributes(bodyfat_rpart)

$names
 [1] "frame"               "where"
 [3] "call"                "terms"
 [5] "cptable"             "method"
 [7] "parms"               "control"
 [9] "functions"           "numresp"
[11] "splits"              "variable.importance"
[13] "y"                   "ordered"

$xlevels
named list()

$class
[1] "rpart"

> print(bodyfat_rpart$cptable)

          CP nsplit  rel error    xerror       xstd
1 0.67272638      0 1.00000000 1.0427457 0.19016187
2 0.09390665      1 0.32727362 0.5081173 0.11702581
3 0.06037503      2 0.23336696 0.4522296 0.09801847
4 0.03420446      3 0.17299193 0.3967005 0.09676249
5 0.01708278      4 0.13878747 0.3015476 0.07385485
6 0.01695763      5 0.12170469 0.2929969 0.06850104
7 0.01007079      6 0.10474706 0.2713231 0.06690466
8 0.01000000      7 0.09467627 0.2713231 0.06690466

> print(bodyfat_rpart)
n= 56

node), split, n, deviance, yval
      * denotes terminal node

 1) root 56 7265.0290000 30.94589
   2) waistcirc< 88.4 31  960.5381000 22.55645
     4) hipcirc< 96.25 14  222.2648000 18.41143
       8) age< 60.5 9   66.8809600 16.19222 *
       9) age>=60.5 5   31.2769200 22.40600 *
     5) hipcirc>=96.25 17  299.6470000 25.97000
      10) waistcirc< 77.75 6   30.7345500 22.32500 *
      11) waistcirc>=77.75 11  145.7148000 27.95818
        22) hipcirc< 99.5 3    0.2568667 23.74667 *
        23) hipcirc>=99.5 8   72.2933500 29.53750 *
   3) waistcirc>=88.4 25 1417.1140000 41.34880
     6) waistcirc< 104.75 18  330.5792000 38.09111
      12) hipcirc< 109.9 9   68.9996200 34.37556 *
```

```
      13) hipcirc>=109.9 9   13.0832000 41.80667 *
     7) waistcirc>=104.75 7  404.3004000 49.72571 *
```

Dengan menggunakan sintaks dibawah ini, pohon keputusan yang telah dibangun diplot.

```
> plot(bodyfat_rpart)
> text(bodyfat_rpart, use.n=T)
```

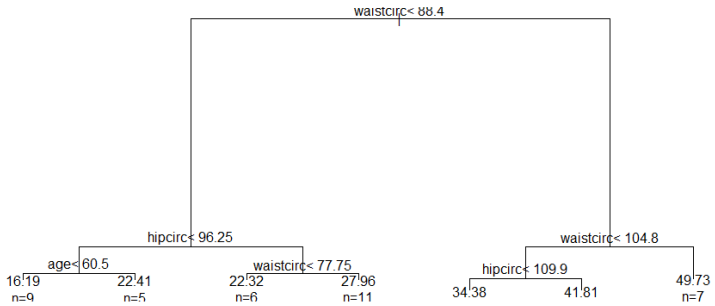


Gambar 5.3 Pohon Keputusan dengan Package `rpart`

Kemudian memilih pohon dengan *error* prediksi terendah dengan sintaks berikut ini.

```
> opt <- which.min(bodyfat_rpart$cptable[,"xerror"])
> cp <- bodyfat_rpart$cptable[opt, "CP"]
> bodyfat_prune <- prune(bodyfat_rpart, cp = cp)
> print(bodyfat_prune)
n= 56

node), split, n, deviance, yval
      * denotes terminal node

 1) root 56 7265.02900 30.94589
   2) waistcirc< 88.4 31  960.53810 22.55645
     4) hipcirc< 96.25 14  222.26480 18.41143
       8) age< 60.5 9   66.88096 16.19222 *
       9) age>=60.5 5   31.27692 22.40600 *
     5) hipcirc>=96.25 17  299.64700 25.97000
      10) waistcirc< 77.75 6   30.73455 22.32500 *
      11) waistcirc>=77.75 11  145.71480 27.95818 *
   3) waistcirc>=88.4 25 1417.11400 41.34880
     6) waistcirc< 104.75 18  330.57920 38.09111
      12) hipcirc< 109.9 9   68.99962 34.37556 *
      13) hipcirc>=109.9 9   13.08320 41.80667 *
     7) waistcirc>=104.75 7  404.30040 49.72571 *
```

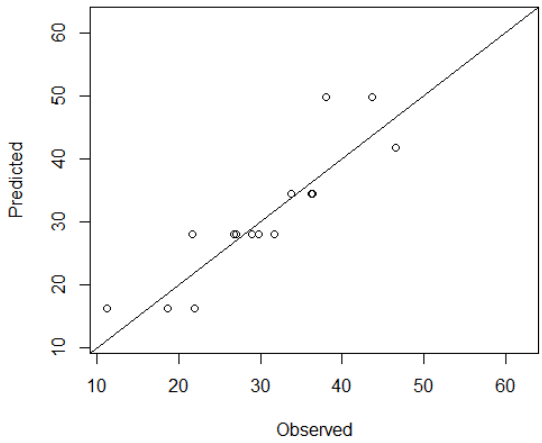
```
> plot(bodyfat_prune)
> text(bodyfat_prune, use.n=T)
```



Gambar 5.4 Pohon Keputusan terpilih

Setelah itu, pohon yang terpilih digunakan untuk membuat prediksi dan hasil prediksi tersebut dibandingkan dengan label aktualnya. Pada sintaks berikut ini, fungsi `abline()` menampilkan sebuah garis diagonal. Hasil prediksi dari model yang bagus diharapkan sama atau sangat dekat dengan nilai aktualnya, yang artinya sebagian besar titik harus berada pada atau dekat dengan garis diagonal.

```
> DEXfat_pred <- predict(bodyfat_prune, newdata=bodyfat.test)
> xlim <- range(bodyfat$DEXfat)
> plot(DEXfat_pred ~ DEXfat, data=bodyfat.test, xlab="Observed",
+     ylab="Predicted", ylim=xlim, xlim=xlim)
> abline(a=0, b=1)
```



Gambar 5.5 Hasil prediksi

5.3 *Random Forest*

Package `randomForest` digunakan untuk membangun model prediktif untuk data `iris`. Terdapat dua batasan menggunakan fungsi `randomForest()`. Pertama, fungsi tersebut tidak dapat menangani data dengan nilai hilang (*missing value*), dan pengguna harus menyesuaikan data sebelum memasukkan ke dalam fungsi. Kedua, terdapat batasan 32 untuk jumlah maksimum level dari setiap atribut kategori. Atribut dengan lebih dari 32 level harus ditransformasi sebelum menggunakan `randomForest()`.

Cara alternatif untuk membangun model random forest adalah menggunakan fungsi `cforest()` dalam package `party`, yang tidak dibatasi level maksimumnya. Namun secara umum, variabel kategorik dengan level yang lebih banyak akan membutuhkan lebih memori dan membutuhkan waktu yang lebih lama untuk membangun modelnya.

Seperti sebelumnya, data `iris` dibagi menjadi dua yaitu data *training* (70%) dan data *testing* (30%).

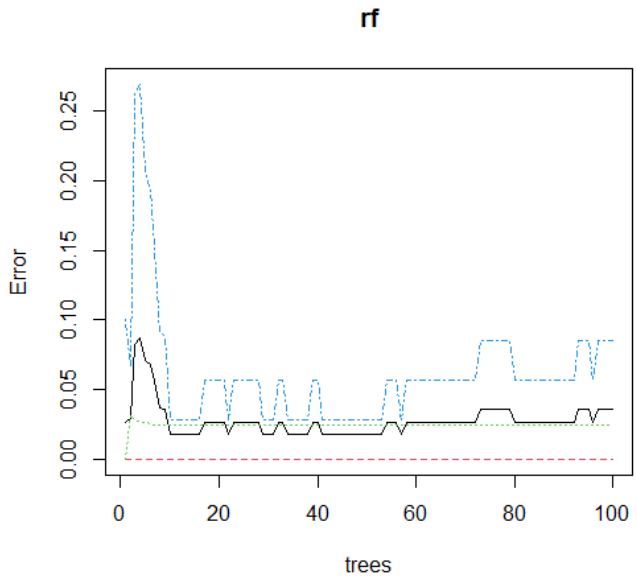
```
> ind <- sample(2, nrow(iris), replace=TRUE, prob=c(0.7, 0.3))
> trainData <- iris[ind==1,]
> testData <- iris[ind==2,]
```

Kemudian memuat *package* `randomForest` dan melatih model random forest. Pada sintaks dibawah ini, rumus diatur sebagai "`Species ~ .`", yang berarti memprediksi `Species` dengan semua variabel lain pada data.

```
> library(randomForest)
> rf <- randomForest(Species ~ ., data=trainData, ntree=100,
+       proximity=TRUE)
> table(predict(rf), trainData$Species)

            setosa versicolor virginica
  setosa         35          0         0
  versicolor      0         40         3
  virginica       0          1        32


> print(rf)

Call:
 randomForest(formula = Species ~ ., data = trainData, ntree = 100,
proximity = TRUE)
               Type of random forest: classification
                     Number of trees: 100
No. of variables tried at each split: 2

        OOB estimate of  error rate: 3.6%
Confusion matrix:
           setosa versicolor virginica class.error
setosa         35          0         0  0.00000000
versicolor      0         40         1  0.02439024
virginica       0          3        32  0.08571429

> attributes(rf)
$names
 [1] "call"           "type"           "predicted"      "err.rate"
 [5] "confusion"      "votes"          "oob.times"      "classes"
 [9]  "importance"          "importanceSD"       "localImportance"
"proximity"
[13] "ntree"           "mtry"            "forest"          "y"
[17] "test"            "inbag"           "terms"

$class
[1] "randomForest.formula" "randomForest"
```

Setelah itu, dilakukan plot pada tingkat *error* berbagai jumlah pohon dengan menggunakan sintaks berikut ini.

```
> plot(rf)
```



Gambar 5.6 Tingkat *error* dari *Random Forest*

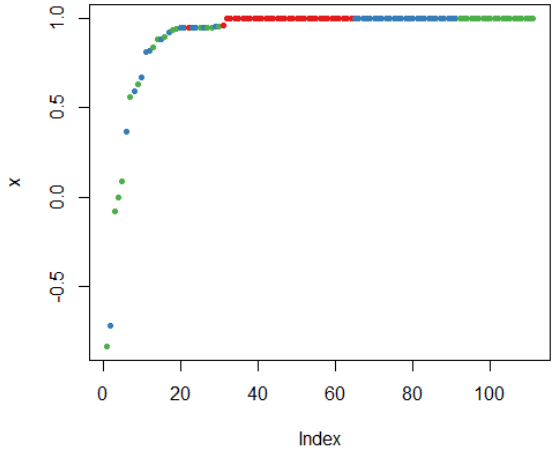
Variabel penting dapat diperoleh menggunakan fungsi `importance()` dan `varImpPlot()`.

```
> importance(rf)
             MeanDecreaseGini
Sepal.Length         7.049879
Sepal.Width          1.841268
Petal.Length        33.097569
Petal.Width         31.214167

> varImpPlot(rf)
```

Gambar 5.7 Variabel penting

Terakhir, model *random forest* yang telah dibangun diuji pada data *testing*, dan hasilnya diperiksa dengan fungsi `table()` dan `margin()`.

```
> irisPred <- predict(rf, newdata=testData)
> table(irisPred, testData$Species)

irisPred     setosa versicolor virginica
  setosa         15          0         0
  versicolor      0          7         0
  virginica       0          2        15

> plot(margin(rf, testData$Species))
```



Gambar 5.8 Margin hasil prediksi

BAB 6 | REGRESI

Regresi adalah teknik membangun sebuah fungsi (model) dari variabel bebas (juga lebih dikenal dengan prediktor) untuk membuat prediksi dari variabel respon. Sebagai contoh, bank menilai tingkat resiko dari pinjaman nasabah berdasarkan umur, pendapatan, pengeluaran, pekerjaan, jumlah tanggungan, batas total kredit, dan sebagainya.

Pada Bab ini akan diperkenalkan konsep dasar dan diperlihatkan contoh dari berbagai macam teknik regresi. Pertama, akan ditunjukkan contoh dalam membagun model regresi linier untuk memprediksi data *Customer Price Index* (CPI). Setelah itu, akan diperkenalkan regresi logistik. Selanjutnya akan diperkenalkan regresi linear diperumum (*Generalized Linear Regression* (GLR)), diikuti dengan pengenalan singkat tentang regresi non-linier.

Kumpulan dari beberapa fungsi R yang berguna dalam analisis regresi tersedia sebagai referensi dalam fungsi R untuk analisis regresi.

6.1 Regresi Linier

Regresi linier adalah teknik untuk memprediksi variabel respon dengan sebuah fungsi linier dari prediktor sebagai berikut:

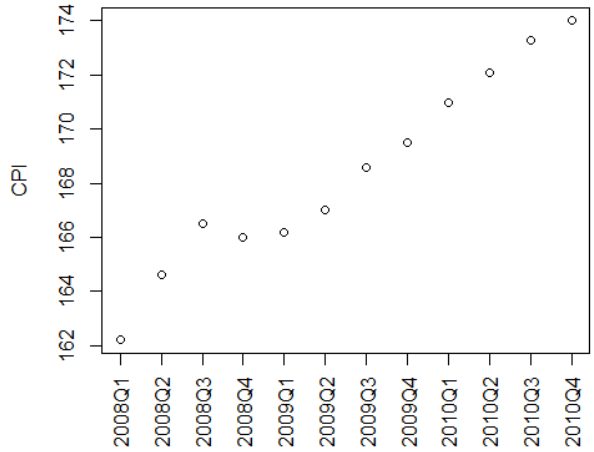
$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_k X_k$$

dimana X_1, X_2, …, X_k adalah prediktor dan Y adalah variabel respon yang akan diprediksi. Regresi linier yang akan ditunjukkan menggunakan fungsi `lm()` pada data `CPI` Australia yang merupakan indeks harga konsumen triwulan dari tahun 2008 hingga 2010.

Pertama, data dibuat dan dibuatkan plot. Pada sintaks berikut ini, sumbu x ditambahkan secara manual dengan fungsi `axis()`, dimana `las=3` membuat teks menjadi vertikal.

```
> year <- rep(2008:2010, each=4)
> quarter <- rep(1:4, 3)
> cpi <- c(162.2, 164.6, 166.5, 166.0,
+          166.2, 167.0, 168.6, 169.5,
+          171.0, 172.1, 173.3, 174.0)
```

```
> plot(cpi, xaxt="n", ylab="CPI", xlab="")
> #Menambahkan sumbu x
> axis(1, labels=paste(year,quarter,sep="Q"), at=1:12, las=3)
```



Gambar 6.1 CPI Australia pada tahun 2008 hingga 2010

Selanjutnya akan dilakukan pengecekkan korelasi antara `CPI` dan variabel lain, yaitu `year` dan `quarter`.

```
> cor(year,cpi)
[1] 0.9096316
> cor(quarter,cpi)
[1] 0.3738028
```

Kemudian sebuah model regresi linier dibangun menggunakan fungsi `lm()` pada data diatas, yaitu menggunakan variabel `year` dan `quarter` sebagai prediktor dan `CPI` sebagai variabel respon.

```
> fit <- lm(cpi ~ year + quarter)
> fit

Call:
lm(formula = cpi ~ year + quarter)

Coefficients:
(Intercept)         year      quarter
  -7644.488        3.888        1.167
```

Berdasarkan model linier diatas, CPI dapat dihitung sebagai:

$$\text{cpi} = \beta_0 + \beta_1 * year + \beta_2 * quarter$$

dimana β_0, β_1, dan β_2 adalah koefisien dari pencocokan model (*model fit*). Oleh karena itu, CPI pada tahun 2011 dapat dihitung seperti berikut ini. Untuk mempermudah proses ini digunakan fungsi `predict()`, yang akan diperlihatkan pada bagian akhir bagian ini.

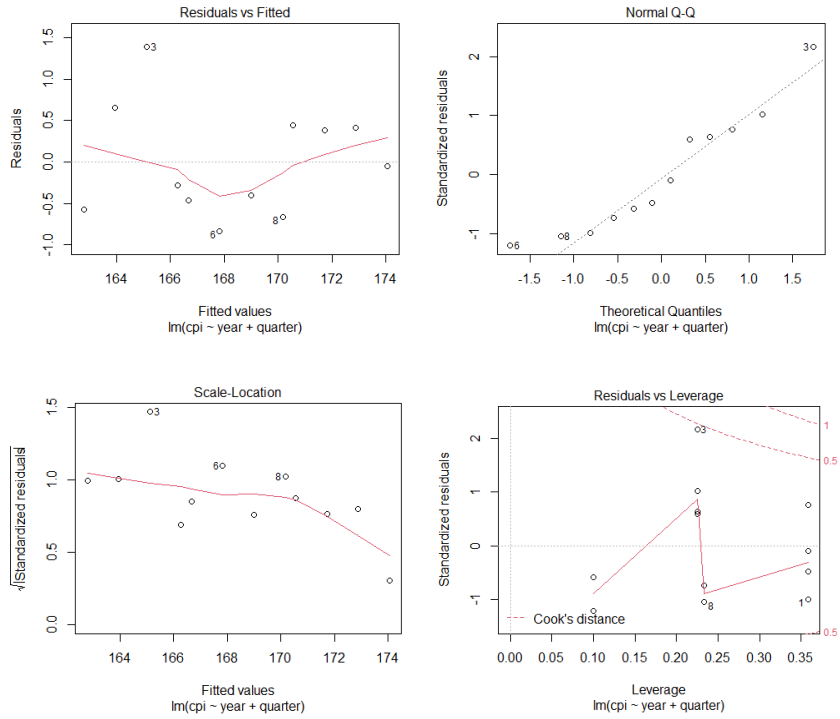
```
> (cpi2011 <- fit$coefficients[[1]] + fit$coefficients[[2]]*2011 +
+             fit$coefficients[[3]]*(1:4))
[1] 174.4417 175.6083 176.7750 177.9417
```

Lebih detail dari model dapat diperoleh dengan menggunakan sintaks dibawah ini.

```
> attributes(fit)

$names
 [1] "coefficients"  "residuals"     "effects"       "rank"
 [5] "fitted.values" "assign"        "qr"            "df.residual"
 [9] "xlevels"       "call"          "terms"         "model"

$class
[1] "lm"

> fit$coefficients

 (Intercept)         year      quarter
-7644.487500     3.887500     1.166667

> # perbedaan antara nilai aktual dan nilai prediksi
> residuals(fit)

          1           2           3           4           5
-0.57916667  0.65416667  1.38750000 -0.27916667 -0.46666667
          6           7           8           9          10
-0.83333333 -0.40000000 -0.66666667  0.44583333  0.37916667
         11          12
 0.41250000 -0.05416667

> summary(fit)

Call:
lm(formula = cpi ~ year + quarter)

Residuals:
```

```
    Min      1Q  Median      3Q     Max
-0.8333 -0.4948 -0.1667  0.4208  1.3875
Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept) -7644.4875   518.6543 -14.739 1.31e-07 ***
year            3.8875     0.2582  15.058 1.09e-07 ***
quarter         1.1667     0.1885   6.188 0.000161 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.7302 on 9 degrees of freedom
Multiple R-squared:  0.9672,  Adjusted R-squared:  0.9599
F-statistic: 132.5 on 2 and 9 DF,  p-value: 2.108e-07
```

Kemudian plot dari model dapat dilihat pada Gambar 6.2 ini.

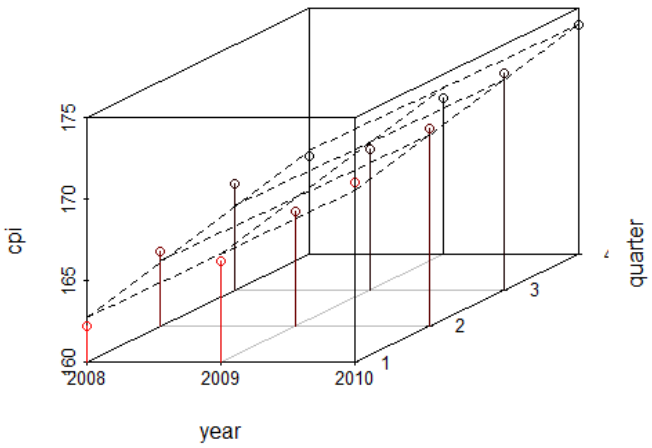


Gambar 6.2 Prediksi dengan model Regresi Linier

Model juga dapat diperlihatkan dalam bentuk plot 3D seperti dibawah ini, dimana fungsi `scatterplot3d()` membuat sebuah *scatter* plot 3D dan `plane3d()` menggambar bidang prediksi. Parameter `lab` menjelaskan jenis tanda pada sumbu x dan y.

```
> library(scatterplot3d)
> s3d <- scatterplot3d(year, quarter, cpi, highlight.3d=T, type="h",
+         lab=c(2,3))
```

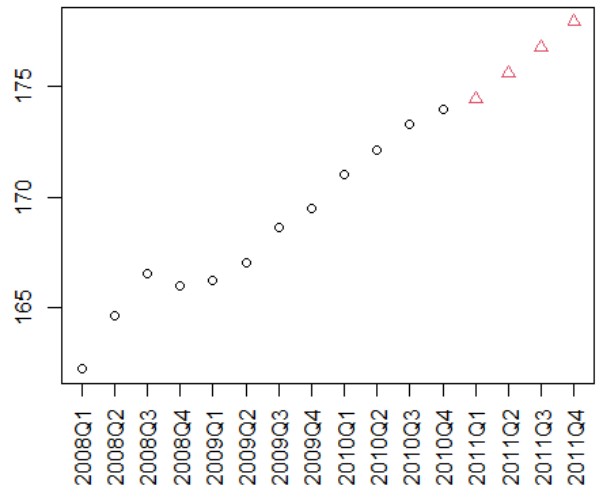
```
> s3d$plane3d(fit)
```



Gambar 6.3 Plot 3D dari model

Dengan model yang diperoleh, nilai `CPI` tahun 2011 dapat dipresiksi seperti dibawah sintaks dibawah ini, dan nilai prediksi akan ditandai dengan segitiga merah pada Gambar 6.4.

```
> data2011 <- data.frame(year=2011, quarter=1:4)
> cpi2011 <- predict(fit, newdata=data2011)
> style <- c(rep(1,12), rep(2,4))
> plot(c(cpi, cpi2011), xaxt="n", ylab="CPI", xlab="", pch=style,
+      col=style)
> axis(1, at=1:16, las=3, labels=c(paste(year,quarter,sep="Q"),
+      "2011Q1", "2011Q2", "2011Q3", "2011Q4"))
```



Gambar 6.4 Prediksi CPI tahun 2011 dengan model Regresi Linier

6.2 Regresi Logistik

Regresi logistik adalah regresi yang digunakan untuk memprediksi peluang terjadinya sebuah kejadian dengan menyesuaikan data ke sebuah kurva logistik. Sebuah model regresi logistik dibangun seperti persamaan berikut:

$$\text{logit}(Y) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_k X_k$$

dimana X_1, X_2, …, X_k adalah prediktor, Y adalah variabel respon yang akan diprediksi, dan $\text{logit}(Y) = \ln\left(\frac{y}{1-y}\right)$. Persamaan diatas juga bisa dituliskan sebagai:

$$Y = \frac{1}{1 + e^{-(\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_k X_k)}}$$

Regresi logistik dapat dibangun dengan menggunakan fungsi `glm()` dengan mengatur `family` dengan `binomial("logit")`. Pada bagian ini akan diilustrasikan menggunakan data Cedegren yang dapat diakses pada websitenya.

```
> cedegren <- read.table("http://www-nlp.stanford.edu/manning/
              courses/ling289/cedegren.txt", header=T)
```

Selanjutnya membuat matriks 2 kolom dari jumlah sukses/gagal untuk variabel respon.

```
> attach(cedegren)
> ced.del <- cbind(sDel, sNoDel)
```

Selanjutnya, membuat model regresi logistik. Bentuk kedua yang lebih pendek dibawah ekuivalen dengan bentuk pertama, namun `family` tidak boleh diabaikan.

```
> ced.logr <- glm(ced.del ~ cat + follows + factor(class),
+                 family=binomial("logit"))
> ced.logr <- glm(ced.del ~ cat + follows + factor(class),
+                 family=binomial)
> ced.logr

Call:  glm(formula = ced.del ~ cat + follows + factor(class), family
= binomial)

Coefficients:
   (Intercept)            catd            catm            catn
       -1.3183         -0.1693          0.1786          0.6667
          catv        followsP        followsV  factor(class)2
       -0.7675          0.9525          0.5341          1.2704
factor(class)3  factor(class)4
        1.0480          1.3742

Degrees of Freedom: 51 Total (i.e. Null);  42 Residual
Null Deviance:        958.7
Residual Deviance: 198.6      AIC: 446.1

> summary(ced.logr)

Call:
glm(formula = ced.del ~ cat + follows + factor(class), family =
binomial)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-3.2438  -1.3432   0.0495   1.0149   6.4009

Coefficients:
               Estimate Std. Error z value Pr(>|z|)
(Intercept)    -1.31827    0.12221 -10.787  < 2e-16 ***
catd           -0.16931    0.10032  -1.688 0.091459 .
catm            0.17858    0.08952   1.995 0.046053 *
catn            0.66672    0.09651   6.908 4.91e-12 ***
catv           -0.76754    0.21844  -3.514 0.000442 ***
followsP        0.95255    0.07400  12.872  < 2e-16 ***
followsV        0.53408    0.05660   9.436  < 2e-16 ***
factor(class)2  1.27045    0.10320  12.310  < 2e-16 ***
factor(class)3  1.04805    0.10355  10.122  < 2e-16 ***
factor(class)4  1.37425    0.10155  13.532  < 2e-16 ***
```

```
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 958.66  on 51  degrees of freedom
Residual deviance: 198.63  on 42  degrees of freedom
AIC: 446.1

Number of Fisher Scoring iterations: 4

> anova(ced.logr, test="Chisq")

Analysis of Deviance Table

Model: binomial, link: logit

Response: ced.del

Terms added sequentially (first to last)


              Df Deviance Resid. Df Resid. Dev  Pr(>Chi)
NULL                             51     958.66
cat            4   314.88        47     643.79 < 2.2e-16 ***
follows        2   228.86        45     414.93 < 2.2e-16 ***
factor(class)  3   216.30        42     198.63 < 2.2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

6.3 *Generalized Linear Regression* (GLR)

Generalized Linear Regression (GLR) menggeneralisasi regresi linier dengan memungkinkan model linier dikaitkan dengan variabel respon melalui fungsi link dan memungkinkan besaran variansi dari setiap pengukuran menjadi fungsi dari nilai prediksi. Ini menggabungkan berbagai model statistik lainnya, termasuk regresi linier, regresi logistik dan regresi Poisson. Fungsi `glm()` digunakan untuk menyesuaikan model linier umum, ditentukan dengan memberikan deskripsi simbolis dari prediktor linier dan deskripsi distribusi *error*.

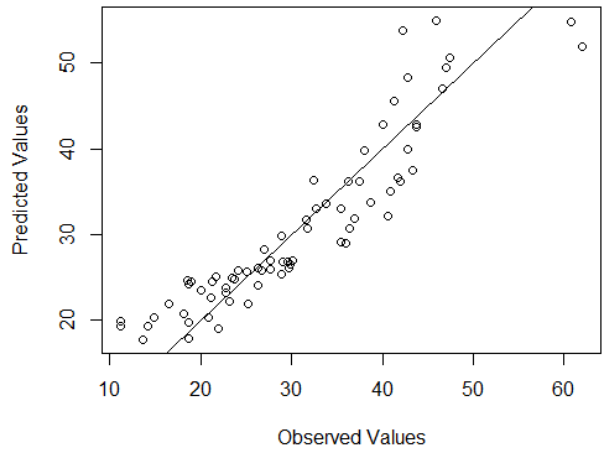
Model GLR dibuat dengan menggunakan fungsi `glm()` pada data `bodyfat`.

```
> data("bodyfat", package="TH.data")
> myFormula <- DEXfat ~ age + waistcirc + hipcirc + elbowbreadth +
+              kneebreadth
```

```
> bodyfat.glm <- glm(myFormula, family = gaussian("log"), data =
+                 bodyfat)
> summary(bodyfat.glm)

Call:
glm(formula = myFormula, family = gaussian("log"), data = bodyfat)

Deviance Residuals:
     Min        1Q    Median        3Q       Max
-11.5688   -3.0065    0.1266    2.8310   10.0966

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.734293   0.308949   2.377  0.02042 *
age          0.002129   0.001446   1.473  0.14560
waistcirc    0.010489   0.002479   4.231 7.44e-05 ***
hipcirc      0.009702   0.003231   3.003  0.00379 **
elbowbreadth 0.002355   0.045686   0.052  0.95905
kneebreadth  0.063188   0.028193   2.241  0.02843 *
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for gaussian family taken to be 20.31433)

    Null deviance: 8536.0  on 70  degrees of freedom
Residual deviance: 1320.4  on 65  degrees of freedom
AIC: 423.02

Number of Fisher Scoring iterations: 5

> pred <- predict(bodyfat.glm, type="response")
```

Pada sintaks diatas, `type` mengindikasikan jenis prediksi yang dibutuhkan. Defaultnya adalah skala prediktor linier, dan pilihan alternatif “`response`” adalah skala variabel respon.

```
> plot(bodyfat$DEXfat, pred, xlab="Observed Values",
+      ylab="Predicted Values")
> abline(a=0, b=1)
```



Gambar 6.5 Prediksi dengan Model GLR

Pada sintaks diatas, jika `family=gaussian("identity")` digunakan, maka model yang terbentuk serupa dengan regresi linier. Model juga dapat berbentuk regresi logistik dengan mengatur `family` dengan `binomial("logit")`.

6.4 Regresi Non-linier

Regresi non linier merupakan suatu metode analisis regresi untuk mendapatkan model non linier yang digunakan untuk mengetahui hubungan antara variabel terikat dan variabel bebas.

```
> DNase1 <- subset(DNase, Run == 1)
> fm1DNase1 <- nls(density ~ SSlogis(log(conc), Asym, xmid, scal),
+              DNase1)
> summary(fm1DNase1)

Formula: density ~ SSlogis(log(conc), Asym, xmid, scal)

Parameters:
     Estimate Std. Error t value Pr(>|t|)
Asym  2.34518    0.07815   30.01 2.17e-13 ***
xmid  1.48309    0.08135   18.23 1.22e-10 ***
scal  1.04146    0.03227   32.27 8.51e-14 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.01919 on 13 degrees of freedom
```

```
Number of iterations to convergence: 0
Achieved convergence tolerance: 8.283e-06

> coef(fm1DNase1)

    Asym     xmid     scal
2.345182 1.483092 1.041455

> coef(summary(fm1DNase1))

     Estimate Std. Error  t value     Pr(>|t|)
Asym 2.345182 0.07815410 30.00715 2.165539e-13
xmid 1.483092 0.08135333 18.23025 1.218541e-10
scal 1.041455 0.03227082 32.27236 8.506932e-14

> fm2DNase1 <- nls(density ~ 1/(1 + exp((xmid - log(conc))/scal)),
+                  data = DNase1,
+                  start = list(xmid = 0, scal = 1),
+                  algorithm = "plinear")
> summary(fm2DNase1)

Formula: density ~ 1/(1 + exp((xmid - log(conc))/scal))

Parameters:
     Estimate Std. Error t value Pr(>|t|)
xmid  1.48309    0.08135   18.23 1.22e-10 ***
scal  1.04145    0.03227   32.27 8.51e-14 ***
.lin  2.34518    0.07815   30.01 2.17e-13 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.01919 on 13 degrees of freedom

Number of iterations to convergence: 5
Achieved convergence tolerance: 1.032e-06

> fm3DNase1 <- nls(density ~ Asym/(1 + exp((xmid –
+                  log(conc))/scal)),
+                  data = DNase1,
+                  start = list(Asym = 3, xmid = 0, scal = 1))
> summary(fm3DNase1)

Formula: density ~ Asym/(1 + exp((xmid - log(conc))/scal))

Parameters:
     Estimate Std. Error t value Pr(>|t|)
Asym  2.34518    0.07815   30.01 2.17e-13 ***
xmid  1.48309    0.08135   18.23 1.22e-10 ***
```

```
scal  1.04145    0.03227   32.27 8.51e-14 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.01919 on 13 degrees of freedom

Number of iterations to convergence: 6
Achieved convergence tolerance: 1.989e-06
```

BAB 7 | PENGELOMPOKAN (*CLUSTERING*)

Bab ini menjelaskan contoh dari beberapa teknik pengelompokkan (*Clustering*) pada R, diantaranya *k-means clustring*, *k-medoids clustering*, *hierarchical clustering and density-based clustering*. Dua bagian pertama memperlihatkan bagaimana algoritma *k-means* dan *k-medoids* untuk mengelompokkan pada data `iris`. Bagian ketiga memperlihatkan sebuah contoh menggunakan *hierarchical clustering* pada data yang sama. Bagian terakhir mendeskripsikan gagasan pengelompokkan berbasis kepadatan (*density*) dan algoritma DBSCAN, serta menunjukkan cara pengelompokkan dengan DBSCAN dan kemudian memberi label data baru dengan model pengelompokkan.

7.1 *K-Means Clustering*

Bagian ini menunjukkan *k-means clustring* pada data `iris`. Pertama, variabel `Species` akan dihilangkan untuk dikelompokkan. Setelah itu, menerapkan fungsi `kmeans()` pada data `iris2` dan menyimpan hasil pengelompokkan dalam variabel `kmeans.result`. Jumlah pengelompokkan ditetapkan 3 seperti pada sintaks dibawah ini.

```
> iris2 <- iris
> iris2$Species <- NULL
> (kmeans.result <- kmeans(iris2, 3))

K-means clustering with 3 clusters of sizes 33, 21, 96

Cluster means:
  Sepal.Length Sepal.Width Petal.Length Petal.Width
1     5.175758    3.624242     1.472727   0.2727273
2     4.738095    2.904762     1.790476   0.3523810
3     6.314583    2.895833     4.973958   1.7031250

Clustering vector:
  [1] 1 2 2 2 1 1 1 1 2 2 1 1 2 2 1 1 1 1 1 1 1 1 1 2 2 1 1 1
 [30] 2 2 1 1 1 2 1 1 1 2 1 1 2 2 1 1 2 1 2 1 1 3 3 3 3 3 3 3 2
 [59] 3 3 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 [88] 3 3 3 3 3 3 2 3 3 3 3 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
[117] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
[146] 3 3 3 3 3
```

```
Within cluster sum of squares by cluster:
[1]   6.432121  17.669524 118.651875
 (between_SS / total_SS =  79.0 %)

Available components:
[1] "cluster"      "centers"      "totss"        "withinss"
[5] "tot.withinss" "betweenss"    "size"         "iter"
[9] "ifault"
```

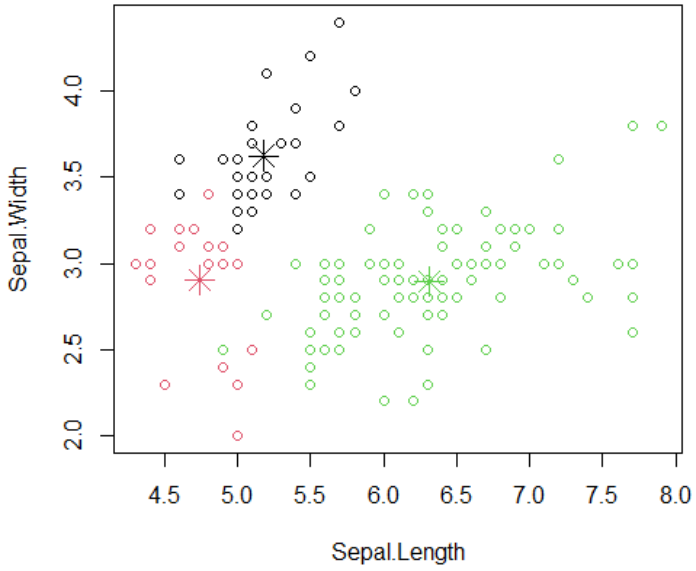
Hasil pengelompokkan kemudian dibandingkan dengan label kelasnya (`Species`) untuk mengecek apakah objek yang serupa dikelompokkan pada kelompok yang sama.

```
> table(iris$Species, kmeans.result$cluster)

             1  2  3
  setosa     33 17  0
  versicolor  0  4 46
  virginica   0  0 50
```

Hasil diatas menunjukkan bahwa kelompok "`setosa`" dapat dipisahkan dengan mudah dari kelompok lain, dan bahwa kelompok "`versicolor`" dan "`virginica`" sedikit tumpang tindih satu sama lain.

Selanjutnya, kelompok dan pusatnya diplot (lihat Gambar 7.1). Perlu diketahui bahwa ada 4 dimensi pada data dan hanya dua dimensi pertama yang digunakan untuk menggambar plot dibawah. Beberapa titik hitam yang dekat dengan titik pusat hijau sebenarnya lebih dekat dengan titik pusat hitam diruang 4 dimensi. Perlu juga diketahui bahwa hasil pengelompokkan `k-means` dapat bervariasi dari satu proses ke proses lainnya, karena pemilihan acak pusat kelompok awalnya.

```
> plot(iris2[c("Sepal.Length", "Sepal.Width")], col =
+      kmeans.result$cluster)
> # plot pusat kelompok
> points(kmeans.result$centers[,c("Sepal.Length", "Sepal.Width")],
+        col = 1:3,pch = 8, cex=2)
```



Gambar 7.1 Hasil dari *k-Means Clustering*

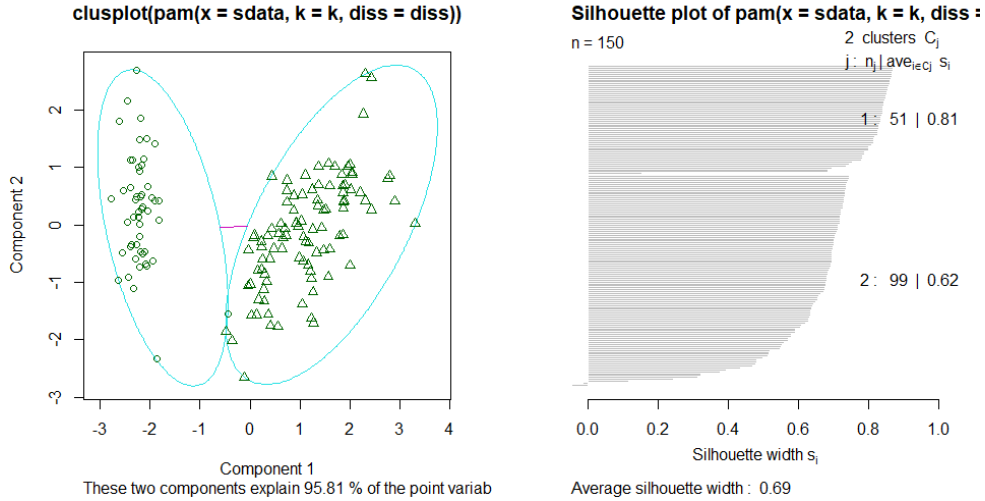
7.2 *K-Medoids Clustering*

Bagian ini menunjukkan *k-medoids clustering* dengan fungsi `pam()` dan `pamk()`. *K-medoids clustering* sangar mirip atau serupa dengan *k-means*, dan perbedaan besar antara keduanya adalah: apabila sebuah kelompok direpresentasikan dengan pusatnya dalam algoritma *k-means*, maka *k-medoids* direpresentasikan dengan objek yang paling dekat dengan pusat kelompoknya. Pengelompokkan *k-medoids* lebih kuat daripada *k-means* dengan adanya pencilan. PAM (*Partitioning Around Medoids*) adalah algortima klasik untuk pengelompokkan *k-medoids*. Sementara algortima PAM tidak efisien untuk mengelompokkan data besar. Algoritma CLARA adalah teknik PAM yang disempurnakan dengan menggambar beberapa sampel data, menerapkan PAM pada setiap sampel dan kemudian mengambil pengelompokkan terbaik. Hal tersebut memiliki kinerja yang lebih baik daripada PAM pada data yang lebih besar. Fungsi `pam()` dan `clara()` dalam *package* `cluster` (Maechler dkk, 2015) masing-masing adalah implementasi dari algoritma PAM dan CLARA pada R. Untuk kedua algoritma itu, pengguna harus menentukan nilai *k*, yaitu jumlah kelompok yang akan dituju. Sebagai versi `pam()` yang telah ditingkatkan, fungsi `pamk()` dalam

package `fpc` (Hennig, 2015) tidak mengharuskan pengguna untuk menentukan nilai *k*.

Dengan menggunakan sintaks dibawah, akan ditunjukkan bagaimana menentukan kelompok (*clusters*) menggunakan fungsi `pam()` dan `pamk()`.

```
> library(fpc)
> pamk.result <- pamk(iris2)
> # jumlah kelompok
> pamk.result$nc

[1] 2

> table(pamk.result$pamobject$clustering, iris$Species)

    setosa versicolor virginica
  1     50          1         0
  2      0         49        50

> layout(matrix(c(1,2),1,2))
> plot(pamk.result$pamobject)
> layout(matrix(1))
```



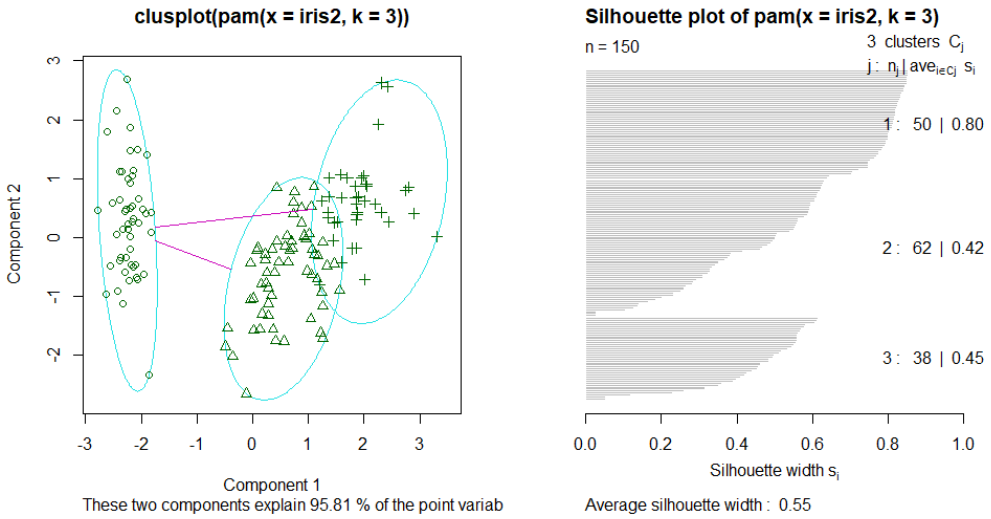
Gambar 7.2 Pengelompokkan dengan algoritma *k-medoids* - I

Berdasarkan contoh diatas, `pamk()` menghasilkan 2 kelompok yaitu "`setosa`" dan campuran dari "`versicolor`" dan "`virginica`". Pada Gambar 9.2, grafik sebelah kiri adalah plot cluster 2 dimensi dari 2 kelompok dan garis merah menunjukkan jarak antara kedua kelompok. Grafik sebelah kanan menunjukkan *silhouettes* keduanya. Pada *silhouettes*, nilai s_i yang besar

(mendekati 1) menunjukkan bahwa pengamatan terkelompokkan dengan sangat baik, nilai s_i yang kecil (mendekati 0) berarti pengamatan terletak diantara 2 kelompok, dan pengamatan dengan nilai s_i negatif kemungkinan terletak pada kelompok yang salah. Karena rata-rata dari s_i masing-masing adalah 0,81 dan 0,62 pada *silhouettes* diatas, maka kedua kelompok yang teridentifikasi dikelompokkan dengan baik.

Selanjutnya, akan digunakan fungsi `pam()` dengan $k = 3$.

```
> library(cluster)
> pam.result <- pam(iris2, 3)
> table(pam.result$clustering, iris$Species)

   setosa versicolor virginica
1      50          0         0
2       0         48        14
3       0          2        36

> layout(matrix(c(1,2),1,2))
> plot(pam.result)
```



Gambar 7.3 Pengelompokkan dengan algoritma *k-medoids* - II

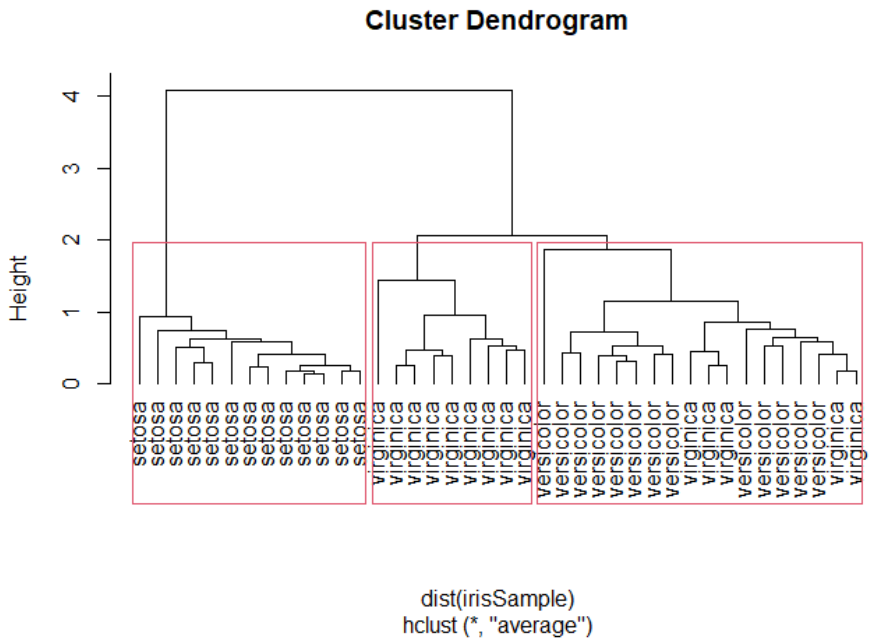
Dengan hasil yang dihasilkan `pam()`, terdapat tiga kelompok. Kelompok 1 adalah spesies "`setosa`" dan dipisahkan dengan baik dari dua kelompok lainnya. Kelompok 2 sebagian besar terdiri dari "`versicolor`", ditambah beberapa kasus dari "`virginca`". Kemudian mayoritas kelompok 3 adalah "`virginica`", dengan 2 kasus dari "`versicolor`".

Sulit untuk mengatakan mana yang lebih baik dari dua pengelompokkan diatas yang dihasilkan oleh `pamk()` dan `pam()`. Hal ini tergantung pada target masalah. Pada contoh ini, hasil dari `pam()` terlihat lebih baik, karena `pam()` mengidentifikasi tiga kelompok, sesuai dengan tiga spesies pada data `iris`. Oleh karena itu, mengidentifikasi jumlah kelompok pada `pamk()` belum tentu memberikan hasil terbaik. Perhatikan bahwa sebelumnya telah diberikan $k = 3$ ketika menggunakan `pam()`, yang sebelumnya telah diketahui bahwa itu adalah jumlah spesies pada data `iris`.

7.3 *Hierarchical Clustering*

Bagian ini memperlihatkan *hierarchical clustering* menggunakan `hclust()` pada data `iris`. Pertama-tama, akan diambil sampel sebanyak 40 dari data `iris`, sehingga plot tidak akan terlalu banyak. Sama seperti sebelumnya, variabel `Species` dihapus dari data. Setelah itu, pada data `iris` diterapkan *hierarchical clustering*.

```
> idx <- sample(1:dim(iris)[1], 40)
> irisSample <- iris[idx,]
> irisSample$Species <- NULL
> hc <- hclust(dist(irisSample), method="ave")
> plot(hc, hang = -1, labels=iris$Species[idx])
> # potong pohon menjadi 3 kelompok
> rect.hclust(hc, k=3)
> groups <- cutree(hc, k=3)
```



Gambar 7.4 Cluster Dendogram

Mirip dengan *k-means clustering*, Gambar 7.4 juga menunjukkan bahwa kelompok "setosa" dapat dengan mudah dipisahkan dari dua kelompok lainnya, dan kelompok "versicolor" dan "virginica" sedikit tumpang tindih satu sama lain.

7.4 *Density-based Clustering*

Algoritma DBSCAN (Ester dkk, 1996) dalam *package* `fpc` (Hennig, 2015) menyediakan *density-based clustering* untuk data numerik. Ide utama dari *density-based clustering* adalah untuk mengelompokkan objek ke dalam sebuah kelompok jika objek terhubung satu sama lain berdasarkan daerah kepadatannya. Terdapat dua parameter utama dalam DBSCAN:

- **`eps:`** jarak jangkauan, yang menentukan ukuran ketetanggaan; dan
- **`MinPts:`** jumlah titik minimum

Jika jumlah titik pada ketetanggaan titik α kurang dari `MinPts`, maka α adalah titik padat. Semua titik dalam ketetanggan dari titik padat adalah jangkaun kepadatan dari α dan dimasukkan dalam kelompok yang sama dengan α.

Kekuatan dari *density-based clustering* adalah dapat menemukan kelompok dengan berbagai bentuk dan ukuran serta tidak sensitif terhadap *noise*. Sebagai

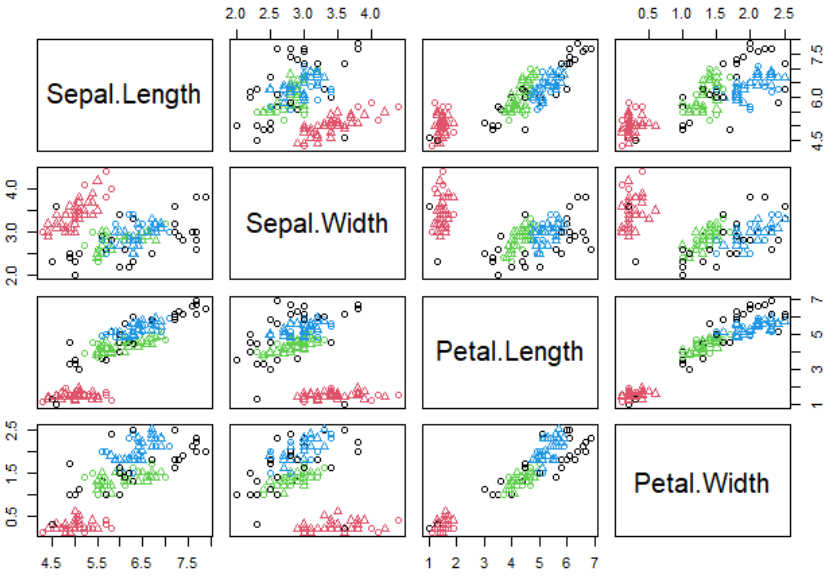
pembanding, algortima *k-means* cenderung mencari kelompok yang berbentuk bulatan dan berukuran serupa.

Dibawah ini adalah contoh *density-based clustering* pada data `iris`.

```
> library(fpc)
> iris2 <- iris[-5] # menghilangkan tag kelas
> ds <- dbscan(iris2, eps=0.42, MinPts=5)
> # membandingkan kelompok dengan kelas aslinya
> table(ds$cluster, iris$Species)

   setosa versicolor virginica
0       2         10        17
1      48          0         0
2       0         37         0
3       0          3        33
```

Berdasarkan tabel diatas, "1" hingga "3" pada kolom pertama adalah tiga kelompok yang teridentifikasi. Selanjutnya "0" adalah *noise* atau *outlier*, yaitu objek yang tidak dapat dikelompokkan pada kelompok mana pun. *Noise* tersebut ditampilkan sebagai lingkaran hitam pada Gambar 9.5.

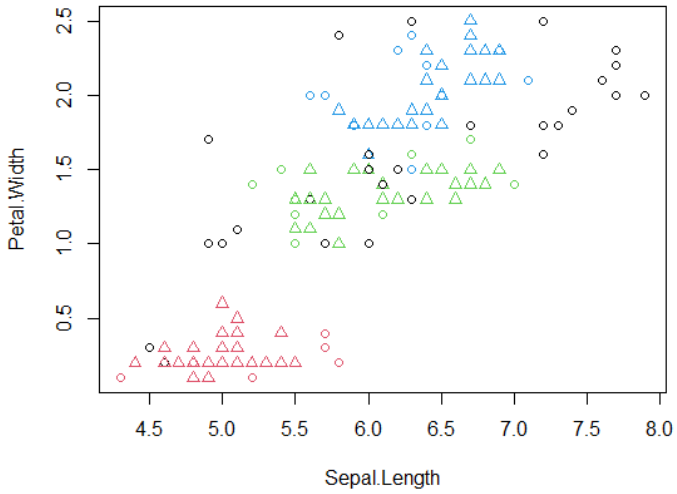
```
> plot(ds, iris2)
```



Gambar 7.5 *Density-based Clustering* - I

Kelompok yang diperoleh diperlihatkan dibawah dalam bentuk *scatter plot* menggunakan kolom pertama dan keempat pada data.

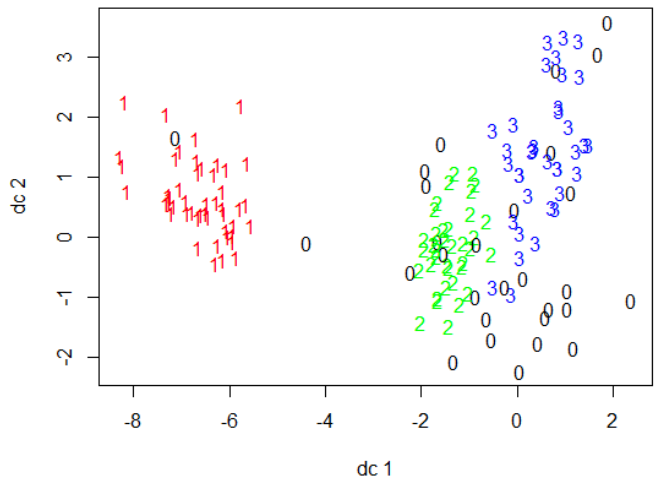
```
> plot(ds, iris2[c(1,4)])
```



Gambar 7.6 *Density-based Clustering* - II

Cara lain untuk menunjukkan kelompok yang diperoleh adalah menggunakan fungsi `plotcluster()` dalam *package* `fpc`. Perlu diketahui bahwa data diproyeksi untuk membedakan kelasnya.

```
> plotcluster(iris2, ds$cluster)
```

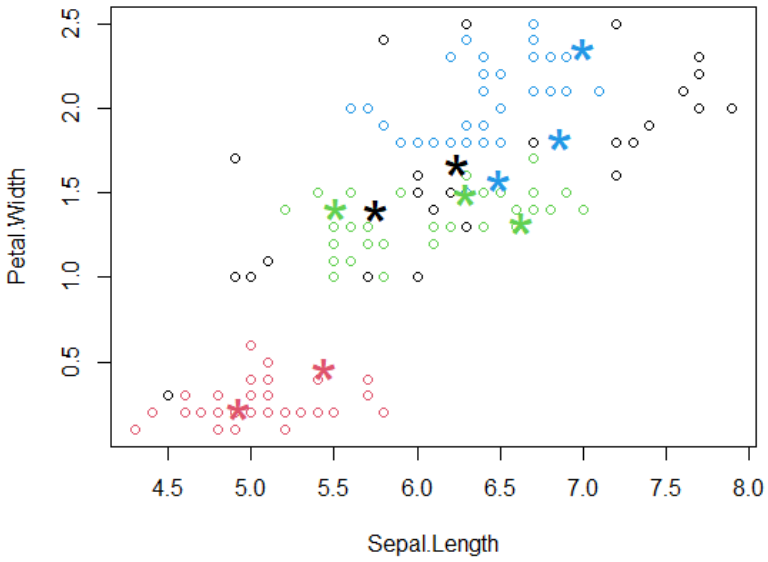


Gambar 7.7 *Density-based Clustering* - III

Model *clustering* dapat digunakan untuk mengelompokkan data baru, berdasarkan kesamaan antara data baru dan kelompok yang telah dibentuk. Contoh

berikut mengambil sampel sebanyak 10 dari data `iris` dan menambahkan *small noise* kedalamnya untuk membuat data baru untuk dikelompokkan. *Noise* acak dibangkitkan dengan distribusi segaram (*uniform*) menggunakan fungsi `runif()`.

```
> # membuat data baru untuk dikelompokkan
> set.seed(435)
> idx <- sample(1:nrow(iris), 10)
> newData <- iris[idx,-5]
> newData <- newData + matrix(runif(10*4, min=0, max=0.2), nrow=10,
ncol=4)
> # mengelompokkan data baru
> myPred <- predict(ds, iris2, newData)
> # plot hasil
> plot(iris2[c(1,4)], col=1+ds$cluster)
> points(newData[c(1,4)], pch="*", col=1+myPred, cex=3)
> # mengecek hasil pengelompokkan


> table(myPred, iris$Species[idx])

myPred setosa versicolor virginica
     0      0          2         0
     1      2          0         0
     2      0          3         0
     3      0          2         1
```



Gambar 7.8 Prediksi dengan model *clustering*

Seperti yang terlihat pada hasil diatas, selain 10 data baru yang belum dikelompokkan, 8 telah dikelompokkan dengan benar (perhatikan diagonal matriks). Data baru ditampilkan dengan tanda "*" dan warnanya menandakan kelompoknya pada Gambar 7.8.

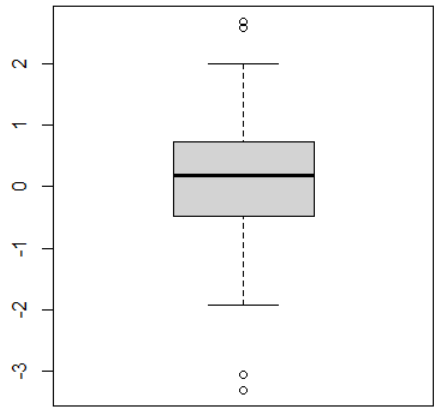
BAB 8 | DETEKSI PENCILAN

Bab ini menjelaskan contoh dari deteksi pencilan menggunakan R. Pertama-tama, bab ini mendemonstrasikan pendeteksian pencilan univariat. Setelah itu, diberikan contoh pendeteksian pencilan dengan memberikan LOF (*Local Outlier Factor*), diikuti dengan contoh pendeteksian pencilan menggunakan pengelompokkan (*clustering*). Terakhir, bab ini mendemonstrasikan pendeteksian pencilan dari data runtun waktu.

8.1 Deteksi Pencilan Univariat

Bagian ini menunjukkan contoh deteksi pencilan univariat, dan menunjukkan cara menerapkannya ke data multivariat. Dalam contoh, deteksi pencilan univariat dilakukan dengan fungsi `boxplot.stats()`, yang mengembalikan statistik untuk menghasilkan *boxplot*. Hasil yang dikembalikan oleh fungsi tersebut adalah sebuah komponen yang memberikan daftar pencilan. Lebih khusus lagi, ini mencantumkan titik data yang berada di luar batas ekstrim. Argumen `coef` dapat digunakan untuk mengontrol seberapa jauh *whiskers* keluar dari kotak *boxplot*. Rincian lebih lanjut tentang itu dapat diperoleh dengan menjalankan perintah `?boxplot.stats` di R. Gambar 8.1 menunjukkan *boxplot*, di mana keempat lingkaran tersebut merupakan pencilan.

```
> set.seed(3147)
> x <- rnorm(100)
> summary(x)

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
-3.3154 -0.4837  0.1867  0.1098  0.7120  2.6859

> # pencilan
> boxplot.stats(x)$out

[1] -3.315391  2.685922 -3.055717  2.571203

> boxplot(x)
```

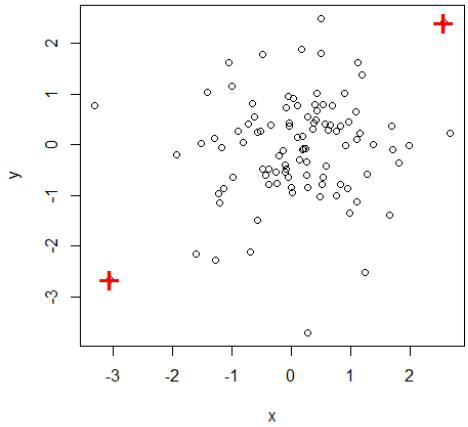


Gambar 8.1 Deteksi Pencilan Univariat dengan *Boxplot*

Deteksi pencilan univariat di atas dapat digunakan untuk menemukan pencilan dalam data multivariat dengan cara *ensemble* sederhana. Pada contoh di bawah ini, pertama-tama dibuat sebuah `dataframe` `df`, yang memiliki dua kolom, `x` dan `y`. Setelah itu, pencilan dideteksi secara terpisah dari `x` dan `y`. Kemudian mengambil pencilan sebagai data yang merupakan pencilan untuk kedua kolom. Pada Gambar 8.2, pencilan diberi label "+" berwarna merah.

```
> y <- rnorm(100)
> df <- data.frame(x, y)
> rm(x, y)
> head(df)
           x          y
1 -3.31539150  0.7619774
2 -0.04765067 -0.6404403
3  0.69720806  0.7645655
4  0.35979073  0.3131930
5  0.18644193  0.1709528
6  0.27493834 -0.8441813

> attach(df)
> # mencari indeks pencilan dari x
> (a <- which(x %in% boxplot.stats(x)$out))

[1]  1 33 64 74

> # mencari indeks pencilan dari y
> (b <- which(y %in% boxplot.stats(y)$out))
[1] 24 25 49 64 74

> detach(df)
> # pencilan pada kedua x dan y
> (outlier.list1 <- intersect(a,b))
[1] 64 74
```

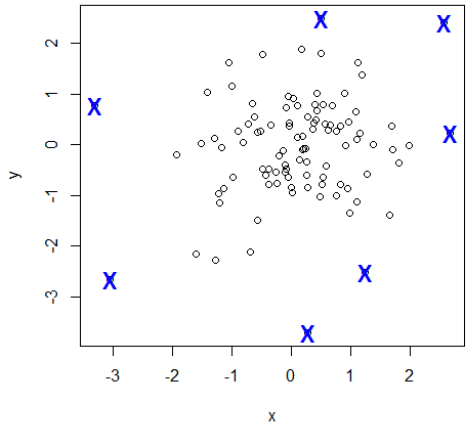
```
> plot(df)
> points(df[outlier.list1,], col="red", pch="+", cex=2.5)
```



Gambar 8.2 Deteksi Pencilan - I

Dengan cara yang sama, pengguna juga bisa mengambil pencilan sebagai data yang pencilannya berasal dari `x` atau `y`. Pada Gambar 8.3, pencilan diberi label dengan "x" dengan warna biru.

```
> # pencilan pada x maupun y
> (outlier.list2 <- union(a,b))
[1]  1 33 64 74 24 25 49

> plot(df)
> points(df[outlier.list2,], col="blue", pch="x", cex=2)
```



Gambar 8.3 Deteksi Pencilan - II

Ketika ada tiga atau lebih variabel dalam sebuah pengaplikasian, daftar akhir pencilan dapat dibuat dengan *vote* suara mayoritas pencilan yang terdeteksi

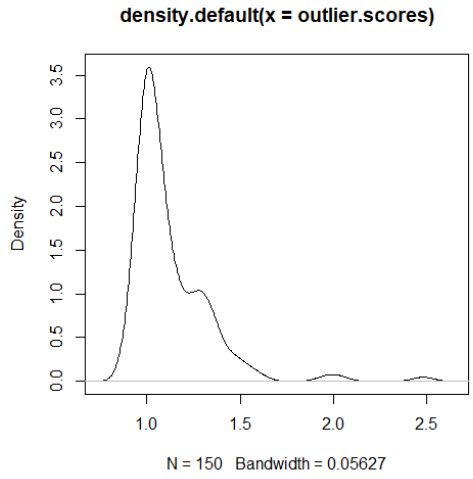
dari setiap variabel. Pengetahuan domain harus dilibatkan saat memilih cara optimal untuk melakukan *ensembel* dalam aplikasi dunia nyata.

8.2 Deteksi Pencilan dengan LOF

LOF (*Local Outlier Factor*) adalah algoritma untuk mengidentifikasi pencilan berbasis kepadatan lokal (Breunig dkk, 2000). Dengan LOF, kepadatan lokal suatu titik dibandingkan dengan tetangganya. Jika yang pertama secara signifikan lebih rendah dari yang terakhir (dengan nilai LOF lebih besar dari satu), titik tersebut berada di wilayah yang lebih jarang daripada tetangganya, yang menunjukkan bahwa titik tersebut adalah pencilan. Kekurangan LOF adalah bekerja pada data numerik saja.

Fungsi `lofactor()` menghitung faktor pencilan lokal menggunakan algoritma LOF, dan tersedia dalam *package* `DMwR` (Torgo, 2010) dan `dprep`. Contoh deteksi pencilan dengan LOF diberikan dibawah ini, di mana `k` adalah jumlah tetangga yang digunakan untuk menghitung faktor pencilan lokal. Gambar 8.4 menunjukkan plot kepadatan (*density plot*) dari skor pencilan.

```
> library(DMwR)
> # menghilangkan "Species" (kolom kategorik)
> iris2 <- iris[,1:4]
> outlier.scores <- lofactor(iris2, k=5)
> plot(density(outlier.scores))
```

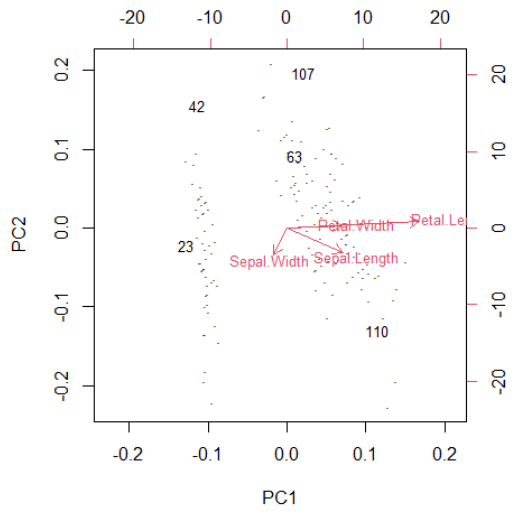


Gambar 8.4 Plot Kepadatan dari Faktor Pencilan

```
> # mengambil 5 pencilan teratas
> outliers <- order(outlier.scores, decreasing=T)[1:5]
> # menampilkan pencilan
> print(outliers)

[1]  42 107  23 110  63

> print(iris2[outliers,])

    Sepal.Length Sepal.Width Petal.Length Petal.Width
42           4.5         2.3          1.3         0.3
107          4.9         2.5          4.5         1.7
23           4.6         3.6          1.0         0.2
110          7.2         3.6          6.1         2.5
63           6.0         2.2          4.0         1.0
```

Selanjutnya, akan diperlihatkan pencilan menggunakan *biplot* dari dua komponen utama pertama (Gambar 10.5).

```
> n <- nrow(iris2)
> labels <- 1:n
> labels[-outliers] <- "."
> biplot(prcomp(iris2), cex=.8, xlabs=labels)
```

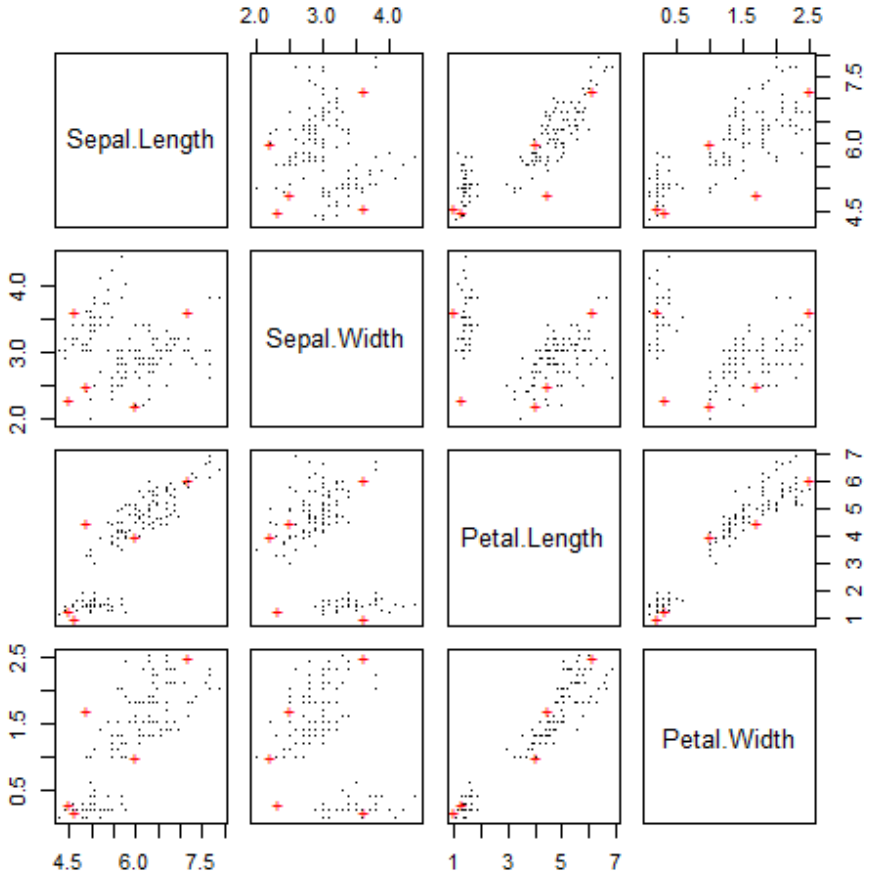


Gambar 8.5 Pencilan pada *Biplot* dari Dua Kompunen Utama Pertama

Pada sintaks diatas, `prcomp()` melakukan analisis komponen utama, dan `biplot()` melakukan plot data menggunakan dua komponen utama pertama. Pada Gambar 8.5, sumbu `x` dan `y` masing-masing merupakan komponen utama pertama dan kedua, panah menunjukkan kolom asli (variabel), dan lima pencilan diberi label dengan nomor barisnya.

Pengguna juga bisa menampilkan pencilan dengan plot berpasangan seperti dibawah ini, di mana pencilan diberi label dengan "+" berwarna merah.

```
> pch <- rep(".", n)
> pch[outliers] <- "+"
> col <- rep("black", n)
> col[outliers] <- "red"
> pairs(iris2, pch=pch, col=col)
```



Gambar 8.6 Pencilan dalam Matriks dari *Scatter Plot*

Package `Rlof` menyediakan fungsi `lof()`, implementasi paralel dari algoritma LOF. Penggunaannya mirip dengan `lofactor()`, namun `lof()` memiliki dua fitur tambahan untuk mendukung beberapa nilai `k` dan beberapa pilihan metrik jarak. Di bawah ini adalah contoh `lof()`. Setelah menghitung skor pencilan, pencilan dapat dideteksi dengan memilih yang teratas.

```
> library(Rlof)
> outlier.scores <- lof(iris2, k=5)
> # coba menggunakan angka ketetanggan berbeda (k = 5,6,7,8,9,10)
> outlier.scores <- lof(iris2, k=c(5:10))
```

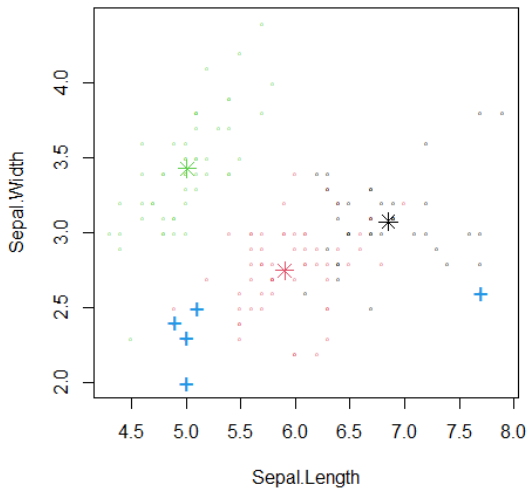
8.3 Deteksi Pencilan dengan *Clustering*

Cara lain untuk mendeteksi pencilan adalah pengelompokan (*clustering*). Dengan mengelompokkan data ke dalam beberapa kelompok, data yang tidak ditetapkan dalam kelompok mana pun akan dianggap sebagai pencilan. Misalnya, dengan pengelompokan berbasis kepadatan seperti DBSCAN (Ester dkk, 1996), objek dikelompokkan ke dalam satu kelompok jika mereka terhubung satu sama lain dengan area kepadatan populasi. Oleh karena itu, objek yang tidak ditetapkan dalam kelompok mana pun terisolasi dari objek lain dan dianggap sebagai pencilan. Contoh DBSCAN dapat ditemukan di Bagian 7.4 *Density-based Clustering*.

Pengguna juga bisa mendeteksi pencilan dengan algoritma *k-means*. Dengan *k-means*, data dipartisi menjadi grup *k* dengan menetapkannya ke pusat kelompok terdekat. Setelah itu, menghitung jarak (atau ketidaksamaan) antara setiap objek dan pusat kelompoknya, dan memilih objek dengan jarak terbesar sebagai pencilan. Contoh deteksi pencilan dengan *k-means* dari data `iris` diberikan di bawah ini.

```
> # menghilangkan “Species” dari data untuk dikelompokkan
> iris2 <- iris[,1:4]
> kmeans.result <- kmeans(iris2, centers=3)
> # pusat kelompok
> kmeans.result$centers

  Sepal.Length Sepal.Width Petal.Length Petal.Width
1     6.850000    3.073684     5.742105    2.071053
2     5.901613    2.748387     4.393548    1.433871
3     5.006000    3.428000     1.462000    0.246000

> # ID (label) kelompok
> kmeans.result$cluster

  [1] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 [26] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 [51] 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 [76] 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
[101] 1 2 1 1 1 1 2 1 1 1 1 1 2 2 1 1 1 1 2 1 2 1 2 1
[126] 1 2 2 1 1 1 1 1 2 1 1 1 1 2 1 1 1 2 1 1 1 2 1 1 2

> # hitung jarak antara objek dan pusat kelompoknya
> centers <- kmeans.result$centers[kmeans.result$cluster, ]
> distances <- sqrt(rowSums((iris2 - centers)^2))
```

```
> # mengamnil 5 pencilan dengan jarak terbesar
> outliers <- order(distances, decreasing=T)[1:5]
> # menampilkan pencilan
> print(outliers)

[1]  99  58  94  61 119

> print(iris2[outliers,])

    Sepal.Length Sepal.Width Petal.Length Petal.Width
99           5.1         2.5          3.0         1.1
58           4.9         2.4          3.3         1.0
94           5.0         2.3          3.3         1.0
61           5.0         2.0          3.5         1.0
119          7.7         2.6          6.9         2.3

> # plot kelompok
> plot(iris2[,c("Sepal.Length", "Sepal.Width")], pch="o",
+      col=kmeans.result$cluster, cex=0.3)

> # plot pusat kelompok
> points(kmeans.result$centers[,c("Sepal.Length", "Sepal.Width")],
+      col=1:3,pch=8, cex=1.5)

> # plot pencilan
> points(iris2[outliers, c("Sepal.Length", "Sepal.Width")],
+      pch="+", col=4, cex=1.5)
```



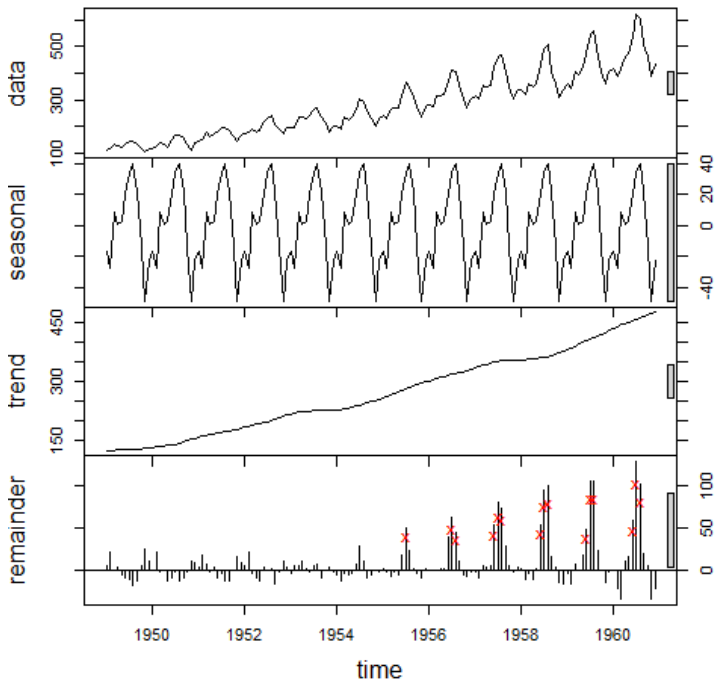
Gambar 8.7 Pencilan dengan *k-means Clustering*

Pada gambar diatas, pusat kelompok diberi label dengan tanda bintang dan pencilan dengan “+”.

8.4 Deteksi Pencilan dari Runtun Waktu

Bagian ini menyajikan contoh deteksi pencilan dari data runtun waktu. Dalam contoh, data runtun waktu pertama kali diuraikan dengan *robust regression* menggunakan fungsi `stl()` dan kemudian pencilan diidentifikasi.

```
> # menggunakan robust fitting
> f <- stl(AirPassengers, "periodic", robust=TRUE)
> (outliers <- which(f$weights<1e-8))

 [1]  79  91  92 102 103 104 114 115 116 126 127 128 138 139 140

> # mengatur tata letak
> op <- par(mar=c(0, 4, 0, 3), oma=c(5, 0, 4, 0), mfcol=c(4, 1))
> plot(f, set.pars=NULL)
> sts <- f$time.series

> # plot pencilan
> points(time(sts)[outliers], 0.8*sts[,"remainder"][outliers],
+        pch="x", col="red")
> par(op) # reset tata letak
```



Gambar 8.8 Pencilan pada Data Runtun Waktu

Pada gambar diatas, pencilan diberi label “x” dengan warna merah.

BAB 9 | ATURAN ASOSIASI (*ASSOCIATION RULES*)

Bab ini menjelaskan contoh dari *association rules mining* dengan R. Bab ini dimulai dengan konsep dasar dari *association rules*, kemudian menjelaskan *association rules mining* menggunakan R. Setelah itu, bab ini memperlihatkan contoh memangkas *rules* yang berlebihan (*redundant*) dan menginterpretasi serta visualisasi *association rules*.

9.1 Dasar *Association Rules*

Association rules adalah aturan yang menyajikan asosiasi atau korelasi antara kumpulan item. Sebuah *association rules* dalam bentuk $A \Rightarrow B$, dimana A dan B adalah dua kumpulan item *disjoint* (saling terpisah), yang masing-masing disebut sebagai aturan **lhs** (*left-hand side*, sisi kiri) dan **rhs** (*right-hand side*, sisi kanan). Tiga ukuran yang paling banyak digunakan untuk memilih aturan (*rules*) menarik adalah *support*, *confidence* dan *lift*. *Support* adalah persentase kasus dalam data yang berisi A dan B, *confidence* adalah persentase kasus yang berisi A yang juga berisi B, dan *lift* adalah perbandingan antara *confidence* dan persentase kasus yang berisi B. Rumus untuk menghitungnya adalah:

$$\text{support}(A \Rightarrow B) = P(A \cup B)$$

$$\text{confidence}(A \Rightarrow B) = P(B|A)$$

$$= \frac{P(A \cup B)}{P(A)}$$

$$\text{lift}(A \Rightarrow B) = \frac{\text{confidence}(A \Rightarrow B)}{P(B)}$$

$$= \frac{P(A \cup B)}{P(A)P(B)}$$

dimana $P(A)$ adalah persentase (atau probabilitas) dari kasus yang berisi A. Selain *support*, *confidence* dan *lift*, masih ada banyak ukuran menarik lainnya, seperti *chi-square*, *conviction*, *gini* dan *leverage*.

9.2 Dataset `Titanic`

Dataset `Titanic` dalam *package* `dataset` adalah data 4 dimensi dengan ringkasan informasi tentang keadaan penumpang `Titanic` berdasarkan kelas sosial, jenis kelamin, umur, dan ketahanan hidup. Agar data cocok untuk *association rules mining*, data mentah tersebut direkonstruksi dan disimpan dengan nama `titanic.raw`, dimana setiap baris mewakili seseorang. Data yang telah direkonstruksi juga bisa didownload di **http://www.rdatamining.com/data** dengan nama `titanic.raw.rdata`.

```
> str(Titanic)

 'table' num [1:4, 1:2, 1:2, 1:2] 0 0 35 0 0 0 17 0 118 154 ...
 - attr(*, "dimnames")=List of 4
  ..$ Class   : chr [1:4] "1st" "2nd" "3rd" "Crew"
  ..$ Sex     : chr [1:2] "Male" "Female"
  ..$ Age     : chr [1:2] "Child" "Adult"
  ..$ Survived: chr [1:2] "No" "Yes"

> df <- as.data.frame(Titanic)
> head(df)

  Class    Sex   Age Survived Freq
1   1st   Male Child       No    0
2   2nd   Male Child       No    0
3   3rd   Male Child       No   35
4  Crew   Male Child       No    0
5   1st Female Child       No    0
6   2nd Female Child       No    0

> titanic.raw <- NULL
> for(i in 1:4) {
+   titanic.raw <- cbind(titanic.raw, rep(as.character(df[,i]),
+                        df$Freq))
+ }
> titanic.raw <- as.data.frame(titanic.raw)
> names(titanic.raw) <- names(df)[1:4]
> dim(titanic.raw)

[1] 2201    4
```

```
> str(titanic.raw)

'data.frame':     2201 obs. of  4 variables:
 $ Class   : Factor w/ 4 levels "1st","2nd","3rd",..: 3 3 3 3 3 ...
 $ Sex     : Factor w/ 2 levels "Female","Male": 2 2 2 2 2 2 2 2 ...
 $ Age     : Factor w/ 2 levels "Adult","Child": 2 2 2 2 2 2 2 2 ...
 $ Survived: Factor w/ 2 levels "No","Yes": 1 1 1 1 1 1 1 1 1 ...

> head(titanic.raw)

  Class  Sex   Age Survived
1   3rd Male Child       No
2   3rd Male Child       No
3   3rd Male Child       No
4   3rd Male Child       No
5   3rd Male Child       No
6   3rd Male Child       No

> summary(titanic.raw)

  Class         Sex          Age       Survived
 1st :325   Female: 470   Adult:2092   No :1490
 2nd :285   Male  :1731   Child: 109   Yes: 711
 3rd :706
 Crew:885
```

Dataset Titanic yang telah diolah seperti diatas juga dapat didownload pada laman **http://www.cs.toronto.edu/~delve/data/titanic/desc.html**.

9.3 *Association Rule Mining*

Algoritma klasik untuk *association rule mining* adalah APRIORI (Agrawal and Srikant, 1984). Implementasi algoritma ini tersedia pada fungsi `apriori()` dalam *package* `arules` (Hahsler, 2014). Algortima lain untuk *association rule mining* adalah algoritma ECLAT (Zaki, 2000) yang tersedia pada fungsi `eclat()` dalam *package* yang sama.

Dibawah ini akan diperlihatkan *association rule mining* menggunakan fungsi `apriori()`. Fungsi ini memiliki pengaturan default sebagai berikut: 1) `supp=0.1`, dimana merepresentasikan nilai *support* minimum dari aturan (*rules*); 2) `conf=0.8`, dimana merepresentasikan nilai *confidence* minimum dari aturan (*rules*); dan 3) `maxlen=10`, dimana merepresentasikan nilai panjang maksimum dari aturan (*rules*).

```
> library(arules)
> rules.all <- apriori(titanic.raw)

Parameter specification:
 confidence minval smax arem  aval originalSupport maxtime
        0.8    0.1    1 none FALSE            TRUE       5
 support minlen maxlen target  ext
     0.1      1     10  rules TRUE

Algorithmic control:
 filter tree heap memopt load sort verbose
    0.1 TRUE TRUE  FALSE TRUE    2    TRUE

Absolute minimum support count: 220

set item appearances ...[0 item(s)] done [0.00s].
set transactions ...[10 item(s), 2201 transaction(s)] done [0.00s].
sorting and recoding items ... [9 item(s)] done [0.00s].
creating transaction tree ... done [0.00s].
checking subsets of size 1 2 3 4 done [0.00s].
writing ... [27 rule(s)] done [0.00s].
creating S4 object  ... done [0.00s].

> rules.all

set of 27 rules

> inspect(rules.all)

     lhs                     rhs          support    confidence  lift
[1]  {}                   => {Age=Adult}   0.9504771  0.9504771   1.0000000
[2]  {Class=2nd}          => {Age=Adult}   0.1185825  0.9157895   0.9635051
[3]  {Class=1st}          => {Age=Adult}   0.1449341  0.9815385   1.0326798
[4]  {Sex=Female}         => {Age=Adult}   0.1930940  0.9042553   0.9513700
[5]  {Class=3rd}          => {Age=Adult}   0.2848705  0.8881020   0.9343750
[6]  {Survived=Yes}       => {Age=Adult}   0.2971377  0.9198312   0.9677574
[7]  {Class=Crew}         => {Sex=Male}    0.3916402  0.9740113   1.2384742
[8]  {Class=Crew}         => {Age=Adult}   0.4020900  1.0000000   1.0521033
[9]  {Survived=No}        => {Sex=Male}    0.6197183  0.9154362   1.1639949
[10] {Survived=No}        => {Age=Adult}   0.6533394  0.9651007   1.0153856
[11] {Sex=Male}           => {Age=Adult}   0.7573830  0.9630272   1.0132040
[12] {Sex=Female,
      Survived=Yes}       => {Age=Adult}   0.1435711  0.9186047   0.9664669
[13] {Class=3rd,
      Sex=Male}           => {Survived=No} 0.1917310  0.8274510   1.2222950
[14] {Class=3rd,
      Survived=No}        => {Age=Adult}   0.2162653  0.9015152   0.9484870
[15] {Class=3rd,
       Sex=Male}          => {Age=Adult}   0.2099046  0.9058824   0.9530818
[16] {Sex=Male,
       Survived=Yes}      => {Age=Adult}   0.1535666  0.9209809   0.9689670
```

```
[17] {Class=Crew,
       Survived=No}     => {Sex=Male}    0.3044071  0.9955423   1.2658514
[18] {Class=Crew,
       Survived=No}     => {Age=Adult}   0.3057701  1.0000000   1.0521033
[19] {Class=Crew,
       Sex=Male}        => {Age=Adult}   0.3916402  1.0000000   1.0521033
[20] {Class=Crew,
       Age=Adult}       => {Sex=Male}    0.3916402  0.9740113   1.2384742
[21] {Sex=Male,
       Survived=No}     => {Age=Adult}   0.6038164  0.9743402   1.0251065
[22] {Age=Adult,
       Survived=No}     => {Sex=Male}    0.6038164  0.9242003   1.1751385
[23] {Class=3rd,
       Sex=Male,
       Survived=No}     => {Age=Adult}   0.1758292  0.9170616   0.9648435
[24] {Class=3rd,
       Age=Adult,
       Survived=No}     => {Sex=Male}    0.1758292  0.8130252   1.0337773
[25] {Class=3rd,
       Sex=Male,
       Age=Adult}       => {Survived=No} 0.1758292  0.8376623   1.2373791
[26] {Class=Crew,
       Sex=Male,
       Survived=No}     => {Age=Adult}   0.3044071  1.0000000   1.0521033
[27] {Class=Crew,
       Age=Adult,
       Survived=No}     => {Sex=Male}    0.3044071  0.9955423   1.2658514
```

Fenomena umum untuk *association rule mining*, banyak aturan yang dibuat diatas yang tidak menarik. Misalkan pengguna yang hanya tertarik pada pada aturan `rhs` yang menunjukkan kelangsungan hidup (`Survival`), jadi pengguna bisa mengatur `rhs=c("Survived=No", "Survived=Yes")` pada `appearance` untuk memastikan bahwa hanya "`Survived=No`" dan "`Survived=Yes`" yang muncul pada aturan `rhs`. Semua item lainnya bisa muncul pada bagian `lhs`, sebagaimana diatur dengan `default="lhs"`. Pada hasil `rules.all` diatas, dapat dilihat bahwa sisi kiri (`lhs`) dari aturan pertama kosong. Untuk menghindari aturan seperti itu, pengguna dapat menetapkan `minlen=2` pada sintaks dibawah ini. Setelah *association rule mining*, aturan diurutkan berdasarkan `lift` agar rules dengan nilai `lift` tertinggi muncul pertama.

```
> rules <- apriori(titanic.raw, control = list(verbose=F),
+    parameter = list(minlen=2, supp=0.005, conf=0.8),
+    appearance = list(rhs=c("Survived=No", "Survived=Yes"),
+    default="lhs"))
> quality(rules) <- round(quality(rules), digits=3)
> rules.sorted <- sort(rules, by="lift")
> inspect(rules.sorted)
```

```
     lhs             rhs            support confidence coverage  lift count
[1]  {Class=2nd,
      Age=Child}  => {Survived=Yes}   0.011      1.000    0.011 3.096    24
[2]  {Class=2nd,
      Sex=Female,
      Age=Child}  => {Survived=Yes}   0.006      1.000    0.006 3.096    13
[3]  {Class=1st,
      Sex=Female} => {Survived=Yes}   0.064      0.972    0.066 3.010   141
[4]  {Class=1st,
      Sex=Female,
      Age=Adult}  => {Survived=Yes}   0.064      0.972    0.065 3.010   140
[5]  {Class=2nd,
      Sex=Female} => {Survived=Yes}   0.042      0.877    0.048 2.716    93
[6]  {Class=Crew,
      Sex=Female} => {Survived=Yes}   0.009      0.870    0.010 2.692    20
[7]  {Class=Crew,
      Sex=Female,
      Age=Adult}  => {Survived=Yes}   0.009      0.870    0.010 2.692    20
[8]  {Class=2nd,
      Sex=Female,
      Age=Adult}  => {Survived=Yes}   0.036      0.860    0.042 2.663    80
[9]  {Class=2nd,
      Sex=Male,
      Age=Adult}  => {Survived=No}    0.070      0.917    0.076 1.354   154
[10] {Class=2nd,
      Sex=Male}   => {Survived=No}    0.070      0.860    0.081 1.271   154
[11] {Class=3rd,
      Sex=Male,
      Age=Adult}  => {Survived=No}    0.176      0.838    0.210 1.237   387
[12] {Class=3rd,
      Sex=Male}   => {Survived=No}    0.192      0.827    0.232 1.222   422
```

Ketika pengaturan lain tidak berubah, dengan nilai *support* minimum yang lebih rendah, lebih banyak aturan yang dihasilkan, dan *association* antara kumpulan item yang terlihat pada aturan akan lebih mungkin terjadi. Pada sintaks diatas, nilai *support* minimum diatur dengan nilai 0.005, sehingga setiap aturan di *support* setidaknya sebesar 12 (`=ceilling(0.005 * 2201)`) kasus, yang dapat diterima untuk 2201 populasi.

Support, *confidence*, dan *lift* adalah tiga ukuran umum untuk memilih *assosiation rules* yang menarik. Selain itu, masih banyak ukuran dalam memilih *assosiation rules* yang menarik, sseperti *chi-square*, *conviction*, *gini*, dan *leverage*. Lebih dari 20 ukuran yang dapat dihitung dengan fungsi `interestMeasure()` dalam *package* `arules`.

9.4 Menghilangkan *Redudancy*

Beberapa aturan yang dihasilkan pada bagian sebelumnya mengandung sedikit atau tidak ada informasi ekstra meskipun aturan yang lain mengandung informasi ekstra pada hasil diatas. Misalnya, aturan kedua diatas tidak memberikan informasi tambahan karena aturan pertama telah memberikan informasi bahwa semua semua anak dengan kelas "*2nd-class*" selamat (*survived*). Secara umum, ketika sebuah aturan (seperti aturan 2) lebih spesifik dari aturan lain namun memiliki nilai *lift* yang sama atau bahkan lebih rendah, maka aturan itu (aturan 2) dikatakan *redundant* (berlebihan). Aturan yang *redundent* lainnya pada hasil sebelumnya adalah aturan 4, 7 dan 8, apabila masing-masing dibandingkan dengan aturan 3, 6 dan 5.

Dibawah ini dilakukan pemangkasan aturan yang *redundant*. Perlu diketahui bahwa aturan telah diurutkan menurun berdasarkan nilai *lift*-nya.

```
> redundant <- is.redundant(rules.sorted)
> which(redundant)

[1] 2 4 7 8

> rules.pruned <- rules.sorted[!redundant]
> inspect(rules.pruned)
```

```
    lhs                                rhs        support confidence    lift
[1] {Class=2nd,Age=Child}          => {Survived=Yes}   0.011      1.000   3.096
[2] {Class=1st,Sex=Female}         => {Survived=Yes}   0.064      0.972   3.010
[3] {Class=2nd,Sex=Female}         => {Survived=Yes}   0.042      0.877   2.716
[4] {Class=Crew,Sex=Female}        => {Survived=Yes}   0.009      0.870   2.692
[5] {Class=2nd,Sex=Male,Age=Adult} => {Survived=No}    0.070      0.917   1.354
[6] {Class=2nd,Sex=Male}           => {Survived=No}    0.070      0.860   1.271
[7] {Class=3rd,Sex=Male,Age=Adult} => {Survived=No}    0.176      0.838   1.237
[8] {Class=3rd,Sex=Male}           => {Survived=No}    0.192      0.827   1.222
```

Pada sintaks diatas, fungsi `is.redundant` melakukan pemeriksaan pada setiap aturan dan mengembalikan nilai *logical* **TRUE** untuk aturan yang *redundant* dan **FALSE** untuk sebaliknya. Berdasarkan hasil diatas, dapat dilihat bahwa aturan 2, 4, 7 dan 8 telah berhasil dipangkas.

9.5 Interpretasi Aturan (*Rules*)

Walaupun *association rules* mudah untuk mencari aturan dengan nilai *lift* tertinggi dari data, bukan hal yang mudah untuk mengerti aturan-aturan yang telah dibuat. Tidak jarang bahwa *association rules* disalahartikan maknanya. Misalnya, dalam daftar aturan diatas, aturan pertama "`{Class=2nd,Age=Child}`" memiliki nilai *confidence* sebesar 1 dan nilai *lift* sebesar 3 serta tidak ada aturan mengenai anak dengan kelas "*1st*" dan "*3rd*". Sehingga, pengguna mengartikan bahwa "*anak-anak dengan kelas '2nd' memiliki tingkat ketahanan hidup yang lebih tinggi dari anak lainnya*". Hal ini keliru, karena aturan hanya menegaskan bahwa semua anak dengan kelas "*2nd*" selamat (*survived*), namun tidak ada informasi untuk membandingkan tingkat ketahanan hidup dari kelas yang berbeda. Untuk menyelidiki masalah diatas, pengguna dapat menggunakan sintaks dibawah untuk mencari aturan dengan rhs "`Survived=Yes`" dan lhs yang hanya berisi "`Class=1st`", "`Class=2nd`", "`Class=3rd`", "`Age=Child`" dan "`Class=Adult`", dan yang tidak mengandung item lain (`default="none"`). Akan digunakan nilai *support* dan *confidence* yang lebih rendah dari sebelumnya untuk mencari semua aturan untuk anak-anak dari kelas yang berbeda.

```
> rules <- apriori(titanic.raw, parameter = list(minlen=3,
+   supp=0.002, conf=0.2), appearance = list(rhs=c("Survived=Yes"),
+  lhs=c("Class=1st",  "Class=2nd",  "Class=3rd",  "Age=Child",
+   "Age=Adult"), default="none"), control = list(verbose=F))
> rules.sorted <- sort(rules, by="confidence")
> inspect(rules.sorted)
    lhs                     rhs            support     confidence  lift
[1] {Class=2nd,Age=Child} => {Survived=Yes} 0.010904134 1.0000000   3.0956399
[2] {Class=1st,Age=Child} => {Survived=Yes} 0.002726034 1.0000000   3.0956399
[3] {Class=1st,Age=Adult} => {Survived=Yes} 0.089504771 0.6175549   1.9117275
[4] {Class=2nd,Age=Adult} => {Survived=Yes} 0.042707860 0.3601533   1.1149048
[5] {Class=3rd,Age=Child} => {Survived=Yes} 0.012267151 0.3417722   1.0580035
[6] {Class=3rd,Age=Adult} => {Survived=Yes} 0.068605179 0.2408293   0.7455209
```

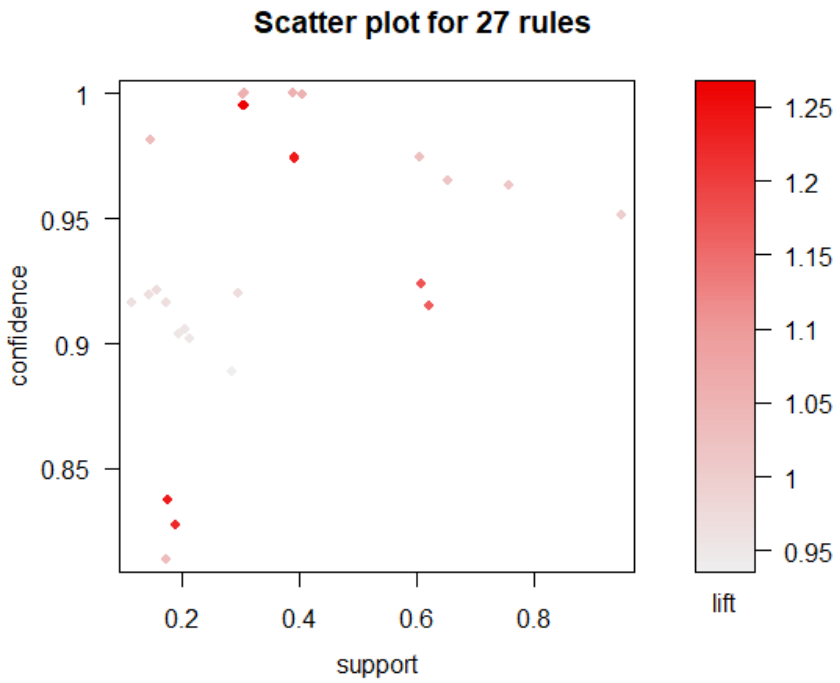
Berdasarkan hasil diatas, dua aturan pertama menunjukkan bahwa anak-anak dengan kelas "*1st*" memiliki tingkat ketahanan hidup yang sama dengan anak-anak dengan kelas "*2nd*" dan semuanya selamat (*survived*). Aturan untuk anak-anak dengan kelas "*1st*" sebelumnya tidak muncul, dikarenakan nilai *support*-nya berada dibawah nilai *support* minimum yang dimasukkan sebelumnya pada pembahasan dan bagian 9.3. Aturan kelima memperlihatkan fakta yang menyedihkan, bahwa

anak-anak dengan kelas "*3rd*" memiliki tingkat ketahanan hidup yang rendah yaitu 34%. Sama halnya dengan orang dewasa dengan kelas "*2nd*" (aturan keempat) yang memiliki tingkat ketahanan hidup yang lebih rendah dari orang dewasa dengan kelas "*1st*" (aturan 3).

9.6 Visualisasi *Association Rules*

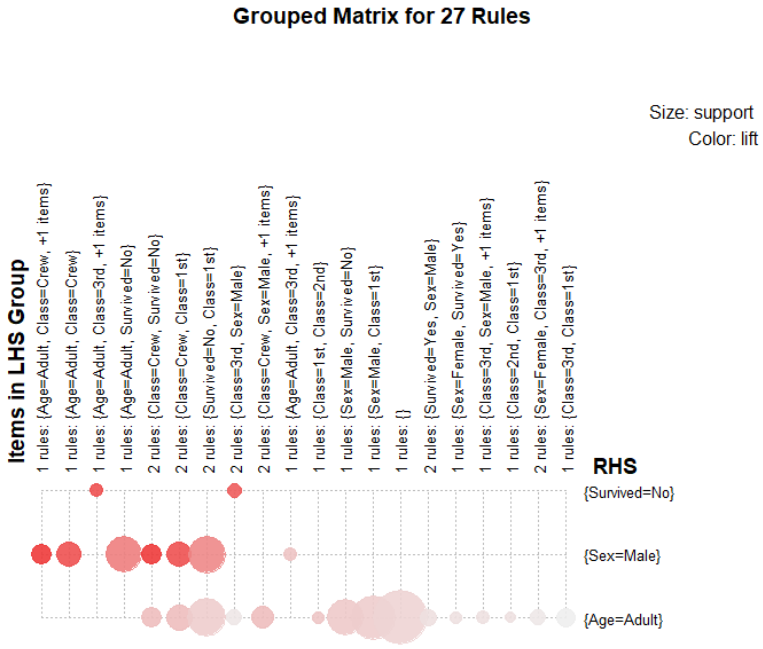
Selanjutnya akan diperlihatkan beberapa cara melakukan visualisasi association rules, diantaranya scatter plot, balloon plot, graph dan parallel coordinates plot. Lebih banyak contoh tentang memvisualisasikan aturan asosiasi dapat ditemukan di sketsa package arulesViz (Hahsler and Chelluboina, 2014) pada CRAN http://cran. r-project.org/web/packages/arulesViz/vignettes/arulesViz.pdf.

```
> library(arulesViz)
> plot(rules.all)
```



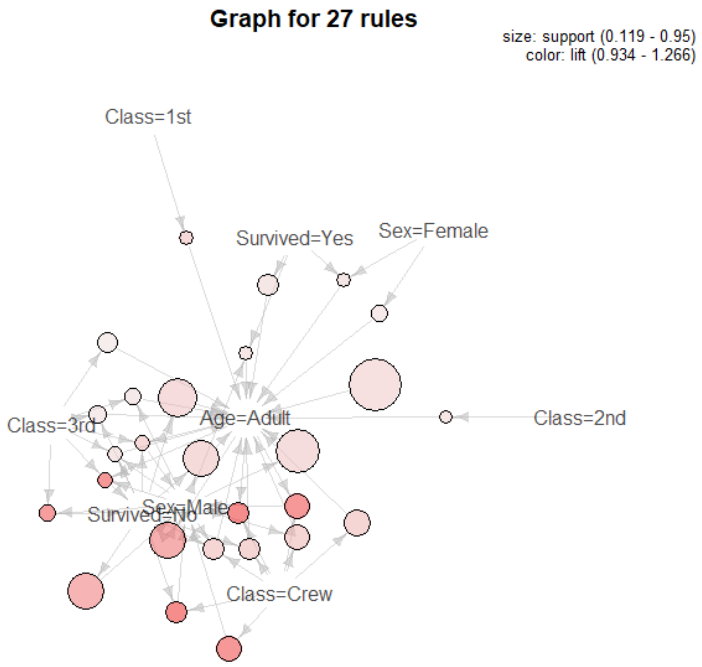
Gambar 9.1 *Scatter Plot Association Rules*

```
> plot(rules.all, method="grouped")
```



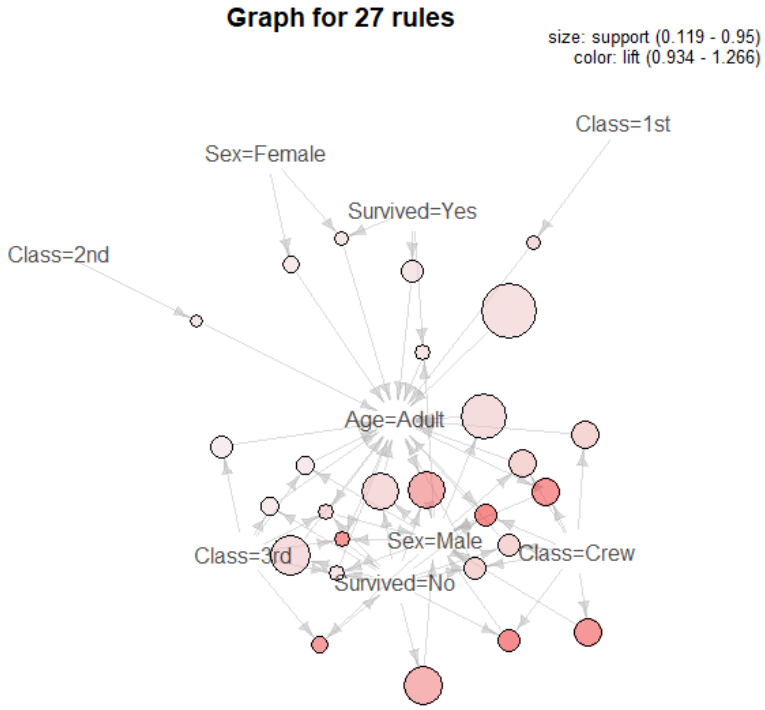
Gambar 9.2 *Ballon Plot Association Rules*

```
> plot(rules.all, method="graph")
```



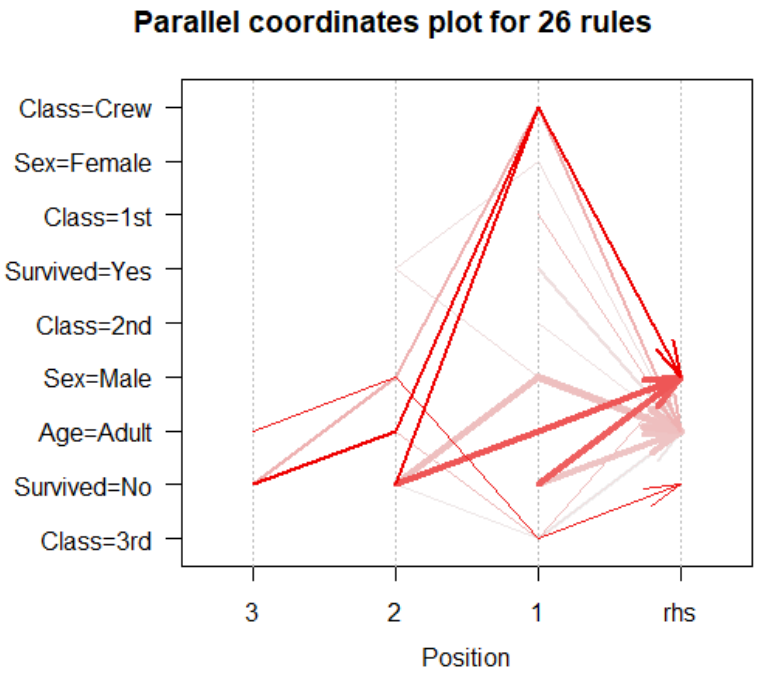
Gambar 9.3 *Graph Association Rules*

```
> plot(rules.all, method="graph", control=list(type="items"))
```



Gambar 9.4 *Graph Item*

```
> plot(rules.all, method="paracoord", control=list(reorder=TRUE))
```



Gambar 9.5 *Parallel Coordinates Plot Association Rules*

BAB 10 | ANALISIS RUNTUN WAKTU DAN MINING

Bab ini menjelaskan contoh penguraian runtun waktu, peramalan, pengelompokkan (*clustering*), dan klasifikasi. Bagian pertama memperkenalkan secara singkat data runtun waktu di R. Bagian kedua menunjukkan contoh penguraian runtun waktu menjadi komponen tren, musiman, dan acak. Bagian ketiga menyajikan bagaimana membangun model *autoregressive integrated moving average* (ARIMA) di R dan menggunakannya untuk peramalan. Bagian keempat memperkenalkan *Dynamic Time Warping* (DTW) dan pengelompokan hierarki (*hierarchical clustering*) data runtun waktu dengan jarak *Euclidean* dan dengan jarak DTW. Bagian kelima menunjukkan tiga contoh klasifikasi runtun waktu: satu dengan data asli, yang lain dengan data transformasi DWT (*Discrete Wavelet Transform*), dan satu lagi dengan klasifikasi *k-NN*.

10.1 Data Runtun Waktu di R

Kelas `ts` merepresentasikan data yang diambil pada titik waktu yang sama. Frekuensi 7 mengindikasikan bahwa sebuah runtun waktu terdiri dari data mingguan dan, frekuensi 12 dan 4 digunakan masing-masing untuk runtun bulanan dan triwulanan. Contoh dibawah menunjukkan konstruksi dari sebuah runtun waktu dengan 30 nilai (1 sampai 30). `Frequency=12` dan `start=c(2011,3)` menjelaskan bahwa data adalah data runtun waktu bulanan dimulai dari bulan Maret 2011.

```
> a <- ts(1:30, frequency=12, start=c(2011,3))
> print(a)

     Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
2011           1   2   3   4   5   6   7   8   9  10
2012  11  12  13  14  15  16  17  18  19  20  21  22
2013  23  24  25  26  27  28  29  30

> str(a)

 Time-Series [1:30] from 2011 to 2014: 1 2 3 4 5 6 7 8 9 10 ...
```

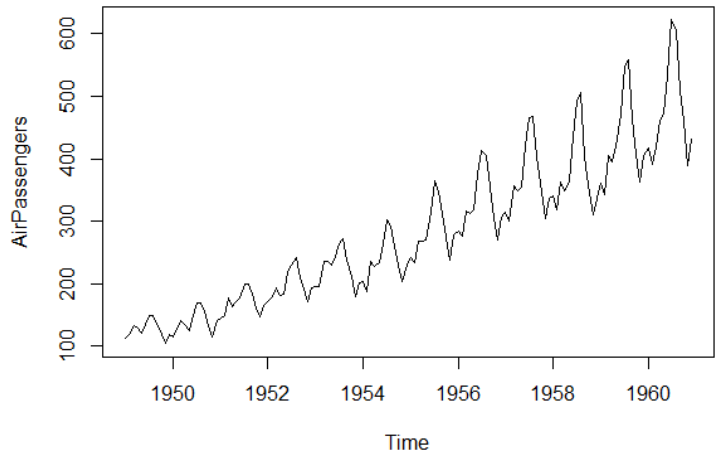
```
> attributes(a)

$tsp
[1] 2011.167 2013.583   12.000

$class
[1] "ts"
```

10.2 Dekomposisi Runtun Waktu

Dekomposisi runtun waktu adalah penguraian sebuah runtun waktu ke dalam komponen *trend*, musiman, siklus, dan *irreguler* (tidak teratur). Komponen *trend* yang dimaksud untuk *trend* jangka panjang, komponen musiman adalah variasi musiman, komponen siklus berulang tetapi fluktuasi non-periodik, dan residu adalah komponen tidak teratur.

Data runtun waktu `AirPassengers` digunakan sebagai contoh untuk demonstrasi penguraian runtun waktu. Data tersebut terdiri dari total bulanan penumpang penerbangan internasional Box & Jenkins dari tahun 1949 sampai 1960 yang memiliki 144 (`=12*12`) nilai.

```
> plot(AirPassengers)
```

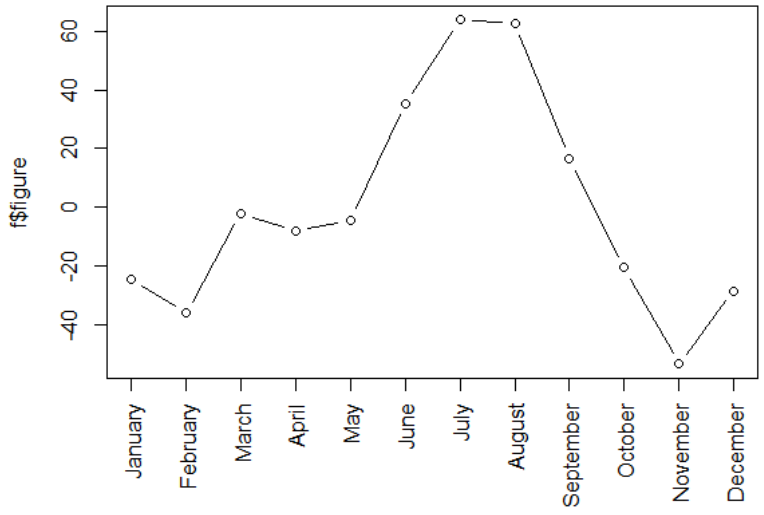


Gambar 10.1 Runtun Waktu `AirPassengers`

```
> # Penguraian runtun waktu
> apts <- ts(AirPassengers, frequency=12)
> f <- decompose(apts)
```

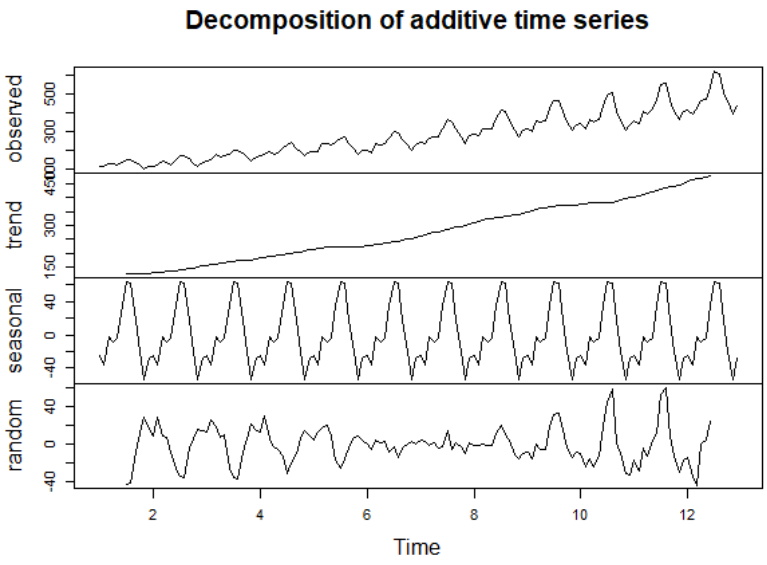
Jika *package* `igraph` sudah dimuat sebelumnya, kemungkinan mendapatkan error yang mengatakan "*Error in decompose(apts) : Not a graph object*". Hal ini dikarenakan fungsi `decompose()` yang berasal dari *package* `stats` tertimpa oleh fungsi dengan nama yang sama pada *package* `igraph`. Untuk mengatasi masalah tersebut, silahkan jalankan sintaks dengan menambahkan informasi bahwa fungsi `decompose()` yang ingin dijalankan berasal dari *package* `stats`.

```
> apts <- ts(AirPassengers, frequency=12)
> f <- stats::decompose(apts)
> # gambar musiman
> f$figure

 [1] -24.748737 -36.188131  -2.241162  -8.036616  -4.506313
 [6]  35.402778  63.830808  62.823232  16.520202 -20.642677
[11] -53.593434 -28.619949

> plot(f$figure, type="b", xaxt="n", xlab="")
```



Gambar 10.2 Komponen musiman

```
> # memperoleh nama dari 12 bulan dalam kosa kata bahasa inggris
> monthNames <- months(ISOdate(2011,1:12,1))
> # label x-axis dengan nama bulan
> # las diatur dengan nilai 2 untuk orientasi label vertikal
> axis(1, at=1:12, labels=monthNames, las=2)
> plot(f)
```



Gambar 10.3 Dekomposisi Runtun Waktu

Pada Gambar 10.3, diagram pertama adalah data runtun waktu asli. Diagram kedua adalah *trend* dari data, diagram ketiga menunjukkan faktor musiman, dan diagram terakhir adalah komponen yang tersisa setelah menghilangkan faktor *trend* dan musiman.

Beberapa fungsi lain untuk dekomposisi runtun waktu adalah `stl()` dalam *package* `stats` (R Core Team, 2015), `decomp()` dalam *package* `timsac` (The Institute of Statistical Mathematics, 2012), dan `tsr()` dalam *package* `ast`.

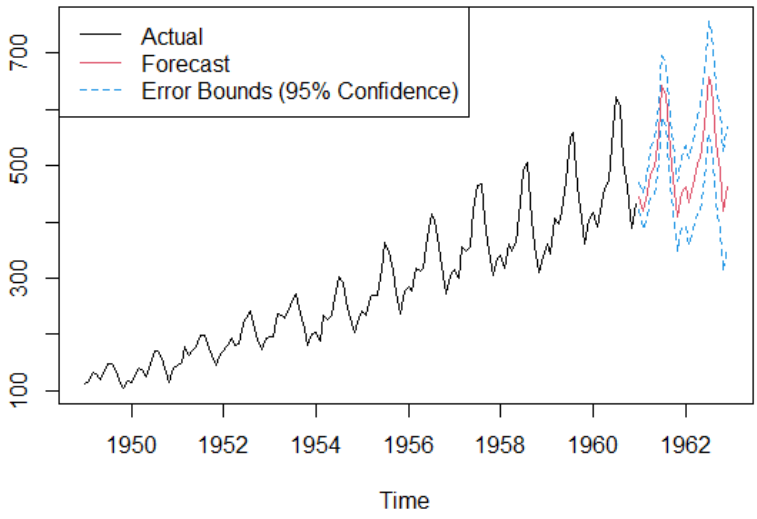
10.3 Meramalkan Runtun Waktu

Meramalkan runtun waktu adalah proses peramalan kejadian masa akan datang berdasarkan data historis. Sebagai contoh, melakukan prediksi harga pembukaan dari sebuah stok berdasarkan kinerja sebelumnya. Dua model populer untuk peramalan runtun waktu adalah *autoregressive moving average* (ARMA) dan *autoregressive integrated moving average* (ARIMA).

Berikut adalah contoh untuk menyesuaikan sebuah model ARIMA pada runtun waktu univariat dan menggunakannya untuk meramalkan.

```
> fit <- arima(AirPassengers, order=c(1,0,0), list(order=c(2,1,0),
+         period=12))
> fore <- predict(fit, n.ahead=24)
```

```
> # tingkat/level kepercayaan 95%
> U <- fore$pred + 2*fore$se
> L <- fore$pred - 2*fore$se
> ts.plot(AirPassengers, fore$pred, U, L, col=c(1,2,4,4),
+         lty = c(1,1,2,2))
> legend("topleft", c("Actual", "Forecast",
+    "Error Bounds (95% Confidence)"), col=c(1,2,4), lty=c(1,1,2))
```



Gambar 10.4 Peramalan Runtun Waktu

Pada Gambar 10.4, garis merah tebal menunjukkan nilai yang telah diramalkan, dan garis titik biru adalah batas kesalahan pada tingkat kepercayaan 95%.

10.4 Pengelompokkan (*Clustering*) Runtun Waktu

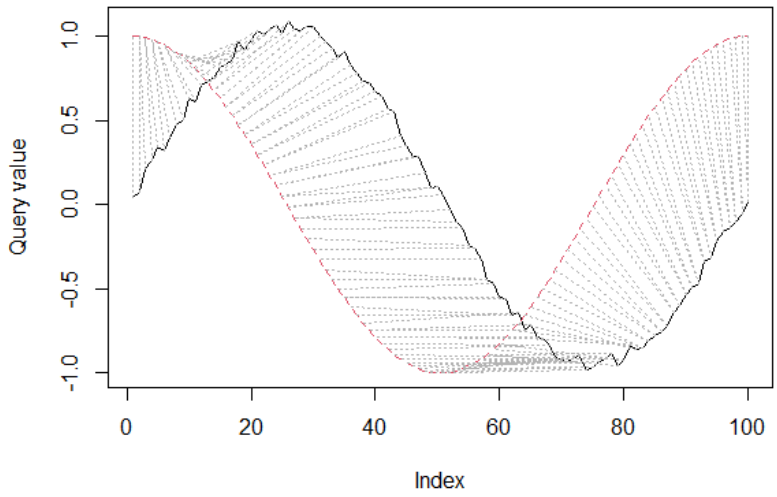
Pengelompokkan runtun waktu dilakukan untuk mempartisi data runtun waktu menjadi kelompok-kelompok berdasarkan kemiripan atau jarak, sehingga runtun waktu dalam kluster yang sama mirip satu sama lain. Ada berbagai macam cara pengukuran jarak atau ketidaksamaan, seperti jarak *Euclidean*, jarak *Manhattan*, *Maximum norm*, jarak *Hamming*, sudut antara dua vektor (perkalian dalam), dan jarak *Dynamic Time Warping* (DTW).

10.4.1 *Dynamic Time Warping*

Dynamic Time Warping (DTW) mencari *alignment* optimal antara dua runtun waktu (Keogh and Pazzani, 2001) dan implementasinya dalam R ada dalam

package `dtw` (Giorgino, 2009). Dalam *package* tersebut, fungsi `dtw(x,y,…)` menghitung *time warp* dinamis dan mencari *alignment* optimal antara runtun waktu `x` dan `y`, dan `dtwDist(mx,my-mx,…)` atau `dist(mx,my-mx,method="DTW",…)` menghitung jarak antara runtun waktu `mx` dan `my`.

```
> library(dtw)
> idx <- seq(0, 2*pi, len=100)
> a <- sin(idx) + runif(100)/10
> b <- cos(idx)
> align <- dtw(a, b, step=asymmetricP1, keep=T)
> dtwPlotTwoWay(align)
```



Gambar 10.5 *Alignment* dengan DTW

10.4.2 Data Runtun Waktu Diagram Kendali Sintetis

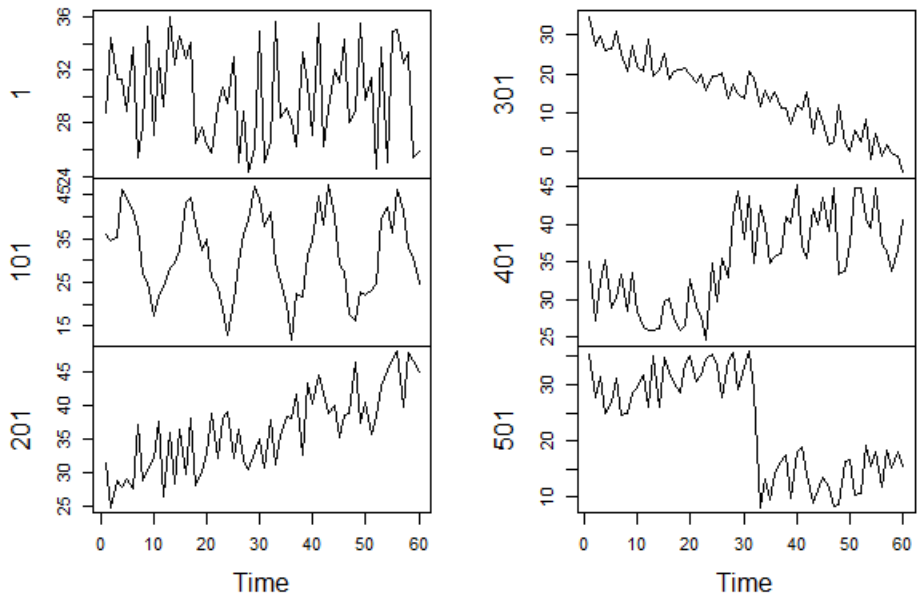
Runtun waktu diagram kendali sintetis yang digunakan pada contoh dibawah ini dapat diakses pada **http://kdd.ics.uci.edu/databases/synthetic_control/synthetic_control.data**. Dataset berisi 600 contoh diagram kendali yang dihasilkan secara sintentis yang oleh proses yang dilakukan Alcock dan Manolopoulos (1999). Setiap diagram kendali adalah sebuah runtun waktu dengan 60 nilai, dan terdapat 6 kelas:

- 1-100: Normal;
- 101-200: Siklus;
- 201-300: *Trend* meningkat;

- 301-400: *Trend* menurun;
- 401-500: *Upward shift*; dan
- 501-600: *Downward shift*.

Pertama, data dibaca ke dalam R dengan fungsi `read.table()`. Parameter `sep` diatur " " (tanpa spasi diantara dua tanda kutip), yang digunakan ketika pemisahnya adalah *white space*, yaitu satu atau beberapa spasi, tab, baris baru atau tanda kutip.

```
> sc <- read.table("http://kdd.ics.uci.edu/databases/synthetic_
       control/synthetic_control.data", header=F, sep="")
> # Menunjukkan satu sampel dari setiap kelas
> idx <- c(1,101,201,301,401,501)
> sample1 <- t(sc[idx,])
> plot.ts(sample1, main="")
```



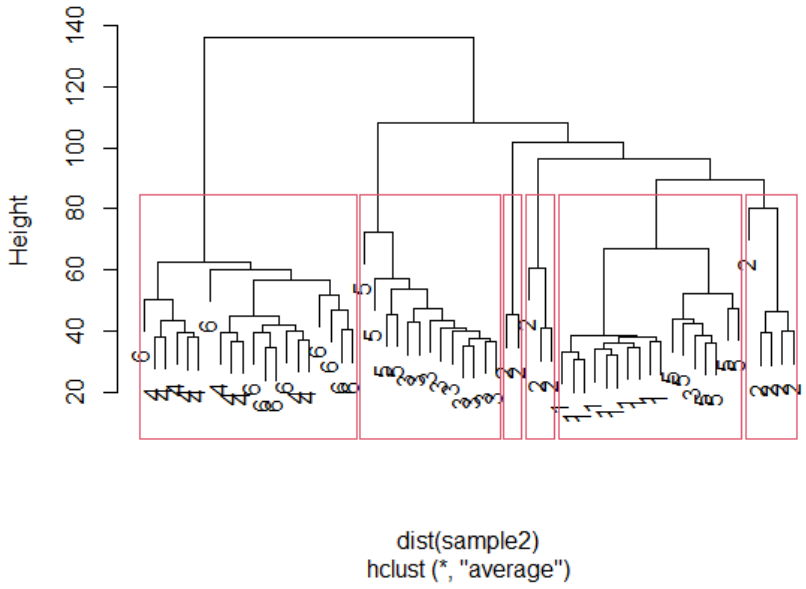
Gambar 10.6 Enam kelas dalam Diagram Kendali Sintetis Runtun Waktu

10.4.3 Pengelompokkan *Hierarchical* dengan Jarak *Euclidean*

Pertama, akan dipilih 10 kasus secara acak dari setiap kelas. Jika tidak, akan ada terlalu banyak kasus dan plot pengelompokkan *Hierarchical* akan terlalu penuh.

```
> set.seed(6218)
> n <- 10
> s <- sample(1:100, n)
> idx <- c(s, 100+s, 200+s, 300+s, 400+s, 500+s)
> sample2 <- sc[idx,]
> observedLabels <- rep(1:6, each=n)
> # hierarchical clustering dengan Euclidean distance
> hc <- hclust(dist(sample2), method="average")
> plot(hc, labels=observedLabels, main="")
> # memotong tree untuk memperoleh 6 kelompok
> rect.hclust(hc, k=6)
> memb <- cutree(hc, k=6)
> table(observedLabels, memb)
```

```
                memb
observedLabels  1  2  3  4  5  6
             1 10  0  0  0  0  0
             2  0  3  2  5  0  0
             3  1  0  0  0  9  0
             4  0  0  0  0  0 10
             5  6  0  0  0  4  0
             6  0  0  0  0  0 10
```



Gambar 10.7 Pengelompokkan *Hierarchical* dengan Jarak *Euclidean*

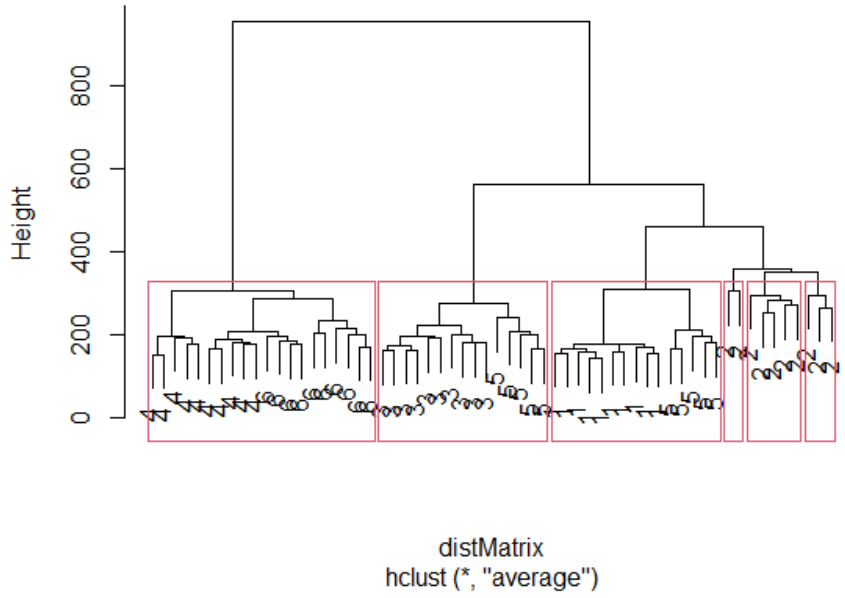
Hasil pengelompokkan pada Gambar 12.7 menunjukkan bahwa, *trend* meningkat (kelas 3) dan *upperward shift* (kelas 5) tidak terpisah dengan baik, dan *trend* menurun (kelas 4) dan *downwarp shift* (kelas 6) juga tercampur.

10.4.4 Pengelompokkan *Hierarchical* dengan Jarak DTW

Selanjutnya, akan dilakukan pengelompokkan *Hierarchical* jarak DTW.

```
> library(dtw)
> distMatrix <- dist(sample2, method="DTW")
> hc <- hclust(distMatrix, method="average")
> plot(hc, labels=observedLabels, main="")
> # memotong tree untuk memperoleh 6 kelompok
> rect.hclust(hc, k=6)
> memb <- cutree(hc, k=6)
> table(observedLabels, memb)
```

```
               memb
observedLabels  1  2  3  4  5  6
             1 10  0  0  0  0  0
             2  0  3  2  5  0  0
             3  0  0  0  0 10  0
             4  0  0  0  0  0 10
             5  5  0  0  0  5  0
             6  0  0  0  0  0 10
```



Gambar 10.8 Pengelompokkan *Hierarchical* dengan Jarak DTW

Dengan membandingkan Gambar 10.7 dan Gambar 10.8, dapat dilihat bahwa jarak DTW lebih baik dibanding jarak *Euclidean* untuk mengukur kemiripan antara runtun waktu.

10.5 Klasifikasi Runtun Waktu

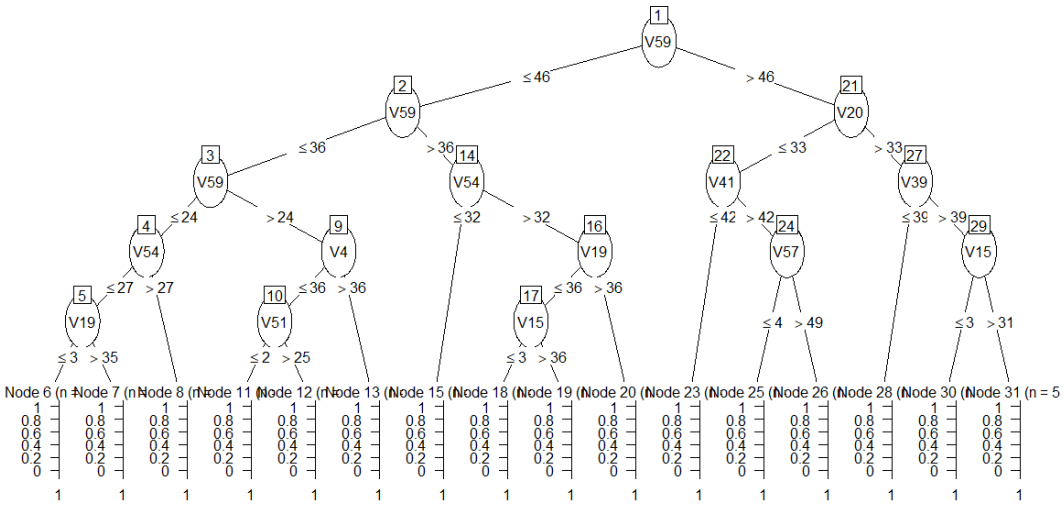
Klasifikasi runtun waktu adalah proses membangun model klasifikasi berdasarkan runtun waktu yang telah dilabeli dan selanjutnya menggunakan model untuk memprediksi label dari runtun waktu yang belum dilabeli. Fitur baru dari runtun waktu mungkin kemungkinan membantu untuk meningkatkan kinerja model klasifikasi. Teknik untuk ekstraksi fitur antara lain *Singular Value Decomposition* (SVD), *Discrete Fourier Transform* (DFT), *Discrete Wavelet Transform* (DWT), *Piecewise Aggregate Approximation* (PAA), *Perpetually Important Points* (PIP), *Piecewise Linear Representation* dan *Symbolic Representation*.

10.5.1 Klasifikasi dengan Data Asli

Fungsi `ctree()` dalam *package* `party` (Hothorn, 2015) digunakan untuk mendemonstrasikan klasifikasi runtun waktu dengan data asli. Label kelas diubah ke dalam bentuk kategorik sebelum memasukkan data ke dalam `ctree()`, sehingga tidak akan ditemui label kelas dengan bilangan riil seperti 1,35. Pohon keputusan yang dibangun diperlihatkan pada Gambar 10.9.

```
> classId <- as.factor(rep(1:6, each=100))
> newSc <- data.frame(cbind(classId, sc))
> library(party)
> ct <- ctree(classId ~ ., data=newSc, controls =
ctree_control(minsplit=30, minbucket=10, maxdepth=5))
> pClassId <- predict(ct)
> table(classId, pClassId)

       pClassId
classId  1  2  3  4  5  6
      1 97  0  0  0  0  3
      2  1 93  2  0  0  4
      3  0  0 96  0  4  0
      4  0  0  0 100  0  0
      5  4  0 10  0 86  0
      6  0  0  0 87  0 13
```

```
> # akurasi
> (sum(classId==pClassId)) / nrow(sc)

[1] 0.8083333

> plot(ct, ip_args=list(pval=FALSE), ep_args=list(digits=0))
```



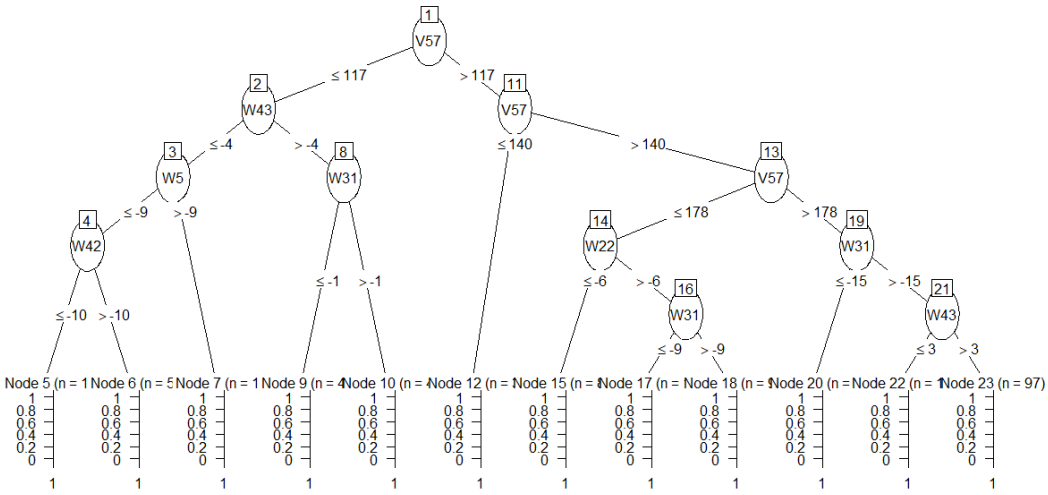
Gambar 10.9 Pohon Keputusan

10.5.2 Klasifikasi dengan Fitur Terekstraksi

Contoh dalam mengekstraksi koefisien DWT (dengan penyaring *Haar*) akan diperlihatkan pada bagian ini. *Package* `wavelet` (Aldrich, 2013) digunakan untuk DWT. Dalam *package*, fungsi `dwt(X,filter,n.levels,…)` menghitung koefisien DWT, di mana `X` adalah runtun waktu univariat atau multivariat, `filter` menunjukkan penyaring *wavelet* mana yang akan digunakan, dan `n.levels` menentapkan level dekomposisi. Fungsi tersebut menhasilkan sebuah objek kelas `dwt`, di mana slot `W` berisi koefisien *wavelet* dan `V` berisi koefisien penskalaan. Runtun waktu asli data dibangun ulang menggunakan invers DWT dengan fungsi `idwt()` dalam *package* yang sama. Model yang dihasilkan diperlihatkan pada Gambar 10.10.

```
> library(wavelets)
> wtData <- NULL
> for (i in 1:nrow(sc)) {
+     a <- t(sc[i,])
+     wt <- dwt(a, filter="haar", boundary="periodic")
+     wtData <- rbind(wtData, unlist(c(wt@W, wt@V[[wt@level]])))
+ }
```

```
> wtData <- as.data.frame(wtData)
> wtSc <- data.frame(cbind(classId, wtData))
> # membangun pohon keputusan dengan koefisien DWT
> ct <- ctree(classId ~ ., data=wtSc, controls = ctree_control
+       (minsplit=30, minbucket=10, maxdepth=5))
> pClassId <- predict(ct)
> table(classId, pClassId)

       pClassId
classId  1  2  3  4  5  6
      1 97  3  0  0  0  0
      2  1 99  0  0  0  0
      3  0  0 81  0 19  0
      4  0  0  0 63  0 37
      5  0  0 16  0 84  0
      6  0  0  0  1  0 99

> (sum(classId==pClassId)) / nrow(wtSc)

[1] 0.8716667

> plot(ct, ip_args=list(pval=FALSE), ep_args=list(digits=0))
```



Gambar 10.10 Pohon Keputusan dengan DWT

10.5.3 Klasifikasi *k*-NN

Klasifikasi *k*-NN dapat juga digunakan untuk klasifikasi runtun waktu. *K*-NN menemukan *k* tetangga terdekat dari instens baru dan melabelinya berdasarkan voting terbanyak. Namun, untuk menemukan tetangga terdekat, kompleksitas waktu menggunakan cara *naïve* adalah $O(n^2)$, di mana n adalah ukuran dari data. Oleh karena itu, struktur *indexing* yang efisien diperlukan untuk data yang

berukuran besar. *Package* RANN mendukung pencarian tetangga terdekat yang cepat menggunakan kompleksitas waktu $O(n \log n)$. Berikut adalah contoh dari klasifikasi *k*-NN pada data runtun waktu tanpa *indexing*.

```
> k <- 20
> # membuat runtun waktu baru dengan menambahkan noise to runtun
waktu 501
> newTS <- sc[501,] + runif(100)*15
> distances <- dist(newTS, sc, method="DTW")
> s <- sort(as.vector(distances), index.return=TRUE)
> # kelas ID dari k nearest neighbors
> table(classId[s$ix[1:k]])

 1  2  3  4  5  6
 0  0  0  3  0 17
```

Untuk 20 tetangga terdekat dari runtun waktu baru, 3 diantaranya adalah kelas 4, dan 17 lainnya adalah kelas 6. Dengan voting terbanyak, yaitu mengambil label yang terbanyak sebagai pemenang, sehingga label runtun waktu baru ditetapkan sebagai kelas 6.

BAB 11 | MENAMBANG TEKS (*TEXT MINING*)

Bab ini menjelaskan contoh dari *text mining* dengan R. Teks Twitter dari *@RdataMining* digunakan sebagai data yang akan dianalisis pada bab ini. Hal ini dimulai dengan mengekstraksi teks dari Twitter. Teks yang telah diekstrak kemudian ditransformasi untuk membangun sebuah matriks *term-document*. Setelah itu, kata-kata dan asosiasi yang sering muncul ditemukan dari matriks. *Word cloud* digunakan untuk menunjukkan kata-kata penting dalam dokumen. Pada akhirnya, kata dan *tweet* dikelompokkan menjadi kelompok kata dan kelompok *tweet*. Dalam bab ini, "*tweet*" dan "*document*" akan digunakan secara bergantian, begitu juga dengan "*word*" dan "*term*".

Ada tiga *package* penting yang digunakan pada contoh yang akan diperlihatkan pada bab ini, yaitu `twitteR`, `tm`, dan `wordcloud`. *Package* `twitteR` (Gentry, 2015) menyediakan akses ke data Twitter, `tm` (Feinerer and Hornik, 2015) menyediakan fungsi untuk *text mining*, dan `wordcloud` (Fellows, 2014) memvisualisasikan hasil dalam bentuk *word cloud*.

11.1 Mengambil Teks dari Twitter

Teks Twitter digunakan pada bab ini untuk mendemonstrasikan *text mining*. *Tweet* diekstrasi dari twitter dengan sintaks dibawah menggunakan fungsi `userTimeline()` dalam *package* `twitteR` (Gentry, 2015). *Package* `twitteR` bergantung pada *package* `Rcurl` (Lang and CRAN team, 2015), yang tersedia pada laman **http://www.stats.ox.ac.uk/pub/RWin/bin/windows/contrib/**. Cara lain untuk mengambil teks dari twitter adalah menggunakan *package* `XML`, dan contohnya dapat dilihat pada laman **http://heuristically.wordpress.com/2011/04/08/text-data-mining-twitter-r/**.

Untuk pembaca yang tidak memiliki akses ke Twitter, data *tweet* "`rdmTweets.RData`" dapat diunduh pada laman **http://www.rdatamining.com/data**. Kemudian pembaca bisa melewati bagian ini dan langsung ke bagian 8.2.

```
> library(twitteR)
> ## Download @RDataMining tweets dari RDataMining.com
> url <- "http://www.rdatamining.com/data/rdmTweets.RData"
> download.file(url, destfile = "rdmTweets.RData")
```

```
> load("~/rdmTweets.RData")
> rdmTweets[11:15]
```

Dengan sintaks diatas, setiap *tweet* dicetak dalam satu baris yang mungkin melebihi batas kertas. Oleh karena itu, sintaks berikut digunakan dalam buku ini untuk mencetak lima *tweet* dengan membungkus teks menjadi selebar kertas. Metode yang sama digunakan untuk mencetak *tweet* dalam sintaks lain di bab ini.

```
> for (i in 11:15) {
+   cat(paste0("[[", i, "]] ", sep=""))
+   writeLines(strwrap(rdmTweets[[i]]$getText(), width=73))
+ }

[[11]] Slides on massive data, shared and distributed memory,and
concurrent
programming: bigmemory and foreach http://t.co/a6bQzxj5
[[12]] The R Reference Card for Data Mining is updated with
functions & packages for handling big data & parallel computing.
http://t.co/FHoVZCyk
[[13]] Post-doc on Optimizing a Cloud for Data Mining primitives,
INRIA, France http://t.co/cA28STPO
[[14]] Chief Scientist - Data Intensive Analytics, Pacific
Northwest National Laboratory (PNNL), US http://t.co/0Gdzq1Nt
[[15]] Top 10 in Data Mining http://t.co/7kAuNvuf
```

11.2 Transformasi Teks

Pertama, *tweet* dikonversi ke dalam bentuk *data frame* dan kemudian ke dalam bentuk *corpus*, yang merupakan kumpulan dokumen teks. Setelah itu, *corpus* dapat diproses dengan fungsi yang tersedia dalam *package* `tm`.

```
> # konersi tweets ke data frame
> df <- twListToDF(rdmTweets)
> dim(df)

[1] 154  10

> library(tm)
> # membangun corpus
> myCorpus <- Corpus(VectorSource(df$text))
```

Setelah itu, *corpus* membutuhkan beberapa transformasi, termasuk mengubah huruf menjadi huruf kecil, dan menghapus tanda baca, angka, dan *stop word*. Daftar *stop-word* dalam bahasa Ingris umum disesuaikan dengan

menambahkan "*available*" dan "*via*" dan menghapus "*r*" dan "*big*". *Hyperlink* juga dihapus pada contoh dibawah ini.

```
> # mengubah ke huruf kecil
> myCorpus <- tm_map(myCorpus, tolower)

> # menghapus URLs
> removeURL <- function(x) gsub("http[^[:space:]]*", "", x)
> myCorpus <- tm_map(myCorpus, removeURL)

> # menghapus selain kata bahasa Inggris atau spasi
> removeNumPunct<-function(x) gsub("[^[:alpha:][:space:]]*","", x)
> myCorpus <- tm_map(myCorpus,content_transformer(removeNumPunct))

> # menghapus tanda baca
> # myCorpus <- tm_map(myCorpus, removePunctuation)

> # menghapus angka
> # myCorpus <- tm_map(myCorpus, removeNumbers)

> # menambah dua stop words tambahan: "available" dan "via"
> myStopwords <- c(stopwords("english"), "available", "via")

> # menghapus "r" dan "big" dari daftar stop word
> myStopwords <- setdiff(myStopwords, c("r", "big"))

> # remove stopwords from corpus
> myCorpus <- tm_map(myCorpus, removeWords, myStopwords)
> myCorpus <- tm_map(myCorpus, stripWhitespace)
```

Pada sintaks diatas, `tm_map()` adalah antarmuka untuk menerapkan transformasi (pemetaan) ke *corpora*. Daftar transformasi yang tersedia dapat diperoleh dari `getTransformations()`, dan yang paling sering digunakan adalah `as.PlainTextDocument()`, `removeNumbers()`, `removePunctuation()`, `removeWords()`, `stemDocument()` dan `stripWhitespace()`. Fungsi `removeURL()` didefinisikan di atas untuk menghapus *hyperlink*, dengan pola `"http [[:alnum:]]*"` yang cocok dengan *string* yang dimulai dengan "*http*" lalu diikuti dengan sejumlah karakter alfabet dan digit. *String* yang cocok dengan pola ini akan dihapus dengan `gsub()`. Pola di atas ditetapkan sebagai ekspresi reguler, dan detailnya dapat ditemukan dengan menjalankan `?regex` di R.

11.3 Pemangkasan Kata

Dalam banyak penerapan, kata perlu dipangkas untuk mengambil intinya, sehingga berbagai bentuk yang telah diturunkan akan dianggap sama pada saat menghitung frekuensi kata. Misalnya, kata "*update*", "*updated*" dan "*updating*" semuanya akan dipangkas menjadi "*updat*". Pemangkasan kata dapat dilakukan dengan *snowball stemmer*, yang membutuhkan *package* `Snowball`, `RWeka`, `rJava` dan `RWekajars`. Setelah itu, melengkapinya ke bentuk aslinya, yaitu *"update"* untuk contoh di atas, sehingga kata-kata tersebut akan terlihat normal. Hal ini bisa dilakukan dengan fungsi `stemCompletion()`.

```
> library(SnowballC)
> myCorpusCopy <- myCorpus
> # stem words
> myCorpus <- tm_map(myCorpus, stemDocument)

> # memeriksa dakumen (tweet) bernomor 11 sampai 15
> # sintaks dibawah membuat teks cocok untuk lebar kertas
> for (i in 11:15) {
+   cat(paste("[[", i, "]] ", sep=""))
+   writeLines(strwrap(myCorpus[[i]], width=73))
+ }

[11] slide massiv data share distribut memoryand concurr program
bigmemori foreach
[12] r refer card data mine updat function packag handl big data
parallel comput
[13] postdoc optim cloud data mine primit inria franc
[14] chief scientist data intens analyt pacif northwest nation
laboratori pnnl us
[15] top data mine
```

Setelah itu, menggunakan `stemCompletion()` untuk melengkapi *stem* dengan *unstemmed corpus* myCorpusCopy sebagai kamus. Dengan pengaturan default, fungsi tersebut mengambil kecocokan paling sering dalam kamus sebagai penyelesaian.

```
> stemCompletion2 <- function(x, dictionary) {
+   x <- unlist(strsplit(as.character(x), " "))
+   x <- x[x != ""]
+   x <- stemCompletion(x, dictionary=dictionary)
+   x <- paste(x, sep="", collapse=" ")
+   PlainTextDocument(stripWhitespace(x))
```

```
+ }
> myCorpus <- lapply(myCorpus, stemCompletion2, dictionary=
              myCorpusCopy)

> myCrText <- NULL
> for (i in 1:154) {
+    myCrText[i] <- myCorpus[[i]][["content"]]
+ }
> myCorpus <- Corpus(VectorSource(myCrText))
```

Kemudian melihat dokumen bernomor 11 sampai 15 dalam *corpus* yang telah dibangun.

```
> for (i in 11:15) {
+   cat(paste0("[", i, "] "))
+   writeLines(strwrap(as.character(myCorpus[[i]]), 60))
+ }

[11] slides massive data share distributed memoryand concurrent
programming foreach
[12] r reference card data miners updated functions package
handling big data parallel computing
[13] postdoctoral optimizing cloud data miners primitives inria
france
[14] chief scientist data intensive analytics pacific northwest
national pnnl using
[15] top data miners
```

Seperti yang dapat dilihat dari hasil diatas, ada beberapa hal yang tak terduga pada hasil pemangkasan dan pelengkapannya.

1. Kedua *Corpus* yang telah dipangkas dan setelah dilengkapi, *"memoryand"* berasal dari *"... memory,and ..."* pada *tweet* 11 asli.
2. Pada *tweet* 11, kata *"bigmemory"* dipangkas menjadi *"bigmemori"*, dan terhapus pada tahap pelengkapan.
3. Kata *"mining"* pada *tweet* 12, 13 dan 15 pada awalnya dipangkas menjadi *"mine"* dan dilengkapi menjadi *"miners"*.
4. Kata *"Laboratory"* pada *tweet* 14 dipangkas menjadi *"laboratori"* dan juga hilang pada tahap pelengkapan.

Dalam masalah diatas, poin 1 disebabkan oleh tidak adanya spasi setelah tanda koma. Ini dapat dengan mudah diperbaiki dengan mengganti koma dengan spasi sebelum menghilangkan tanda baca pada Bagian 10.2. Untuk poin 2 & 4,

belum diketahui mengapa bisa terjadi seperti itu. Namun, kata-kata yang terlibat dalam poin 1, 2 dan 4 tidak penting dalam tweet *@RDataMining* dan mengabaikannya tidak akan membahayakan demonstrasi *text mining* ini.

Di bawah ini difokuskan pada poin 3, dimana kata *"mining"* pertama kali berasal dari *"mine"* dan kemudian dilengkapkan menjadi *"miners"*, bukan *"mining"*, meskipun ada banyak contoh *"mining"* pada *tweet*, dibandingkan dengan hanya dua contoh *"miners"*. Mungkin ada solusi untuk masalah di atas dengan mengubah parameter dan/atau kamus untuk pemangkasan dan pelengkapan. Cara sederhana untuk menyelesaikan masalah pada poin 3 adalah dengan mengganti *"miners"* dengan *"mining"*, karena yang terakhir memiliki lebih banyak kasus daripada yang sebelumnya dalam *corpus*. Sintaks untuk penggantian diberikan dibawah ini.

```
> # menghitung jumlah frekuensi "mining"
> miningCases <- lapply(myCorpusCopy,function(x){grep(as.char
                 acter(x), pattern = "\\<mining")})
> sum(unlist(miningCases))

[1] 47

> # menghitung jumlah frekuensi "miner"
> minerCases <- lapply(myCorpusCopy,function(x){grep(as.character
               (x), pattern = "\\<miner")} )
> sum(unlist(minerCases))

[1] 2

> # mengganti "miner" dengan "mining"
> myCorpus <- tm_map(myCorpus, content_transformer(gsub),
            pattern = "miners", replacement = "mining")
```

Pada pemanggilan pertama fungsi `tm_map()` pada di atas, `grep()` diterapkan ke setiap dokumen (*tweet*) dengan argumen `pattern="\\<mining"`. Pola tersebut cocok dengan kata-kata yang dimulai dengan *"mining"*, dimana "`\\<`" menandakan *string* kosong di awal kata. Hal ini memastikan bahwa teks *"rdatamining"* tidak akan berkontribusi pada penghitungan *"mining"* di atas.

11.4 Membangun Matriks *Term-Document*

Matriks *term-document* merepresentasikan hubungan antara suku kata dan dokumen, dimana setiap baris mewakili suku kata dan setiap kolom mewakili dokumen, dan entri adalah jumlah kemunculan suku kata dalam dokumen. Alternatifnya, seseorang juga bisa membangun matriks *term-document* dengan menukar baris dan kolomnya. Pada bagian ini, akan diperlihatkan cara membangun matriks *term-document* dari *corpus* yang diproses di atas dengan fungsi `TermDocumentMatrix()`. Dengan pengaturan default, suku kata kurang dari tiga karakter akan dibuang. Untuk mempertahankan *"r"* dalam matriks, akan ditetapkan rentang `wordLengths` seperti contoh dibawah.

```
> tdm <- TermDocumentMatrix(myCorpus, control = list(wordLengths =
        c(1,Inf)))
> tdm

<<TermDocumentMatrix (terms: 479, documents: 154)>>
Non-/sparse entries: 1161/72605
Sparsity           : 98%
Maximal term length: 27
Weighting          : term frequency (tf)
```

Seperti yang dapat dilihat dari hasil diatas, matriks *term-document* terdiri dari 479 suku kata dan 154 dokumen. Kemudian melihat enam suku kata pertama yang dimulai dengan *"r"* dan *tweet* bernomor 101 sampai 110.

```
> sortind <- order(dimnames(tdm)$Terms)
> idx <- which((dimnames(tdm)$Terms)[sortind] == "r")
> inspect(tdm[sortind[idx+(0:5)],101:110])

<<TermDocumentMatrix (terms: 6, documents: 10)>>
Non-/sparse entries: 9/51
Sparsity           : 85%
Maximal term length: 12
Weighting          : term frequency (tf)
Sample             :
             Docs
Terms         101 102 103 104 105 106 107 108 109 110
  r             1   1   0   0   2   0   0   1   1   1
  ramachandran  0   0   0   0   0   0   0   0   0   0
  random        0   0   0   0   0   0   0   0   0   0
  ranked        0   0   0   0   0   0   0   0   1   0
  rapidminer    1   0   0   0   0   0   0   0   0   0
```

```
  rdatamining    0   0   0   0   0   0   0   1   0   0
```

Daftar suku kata dapat diambil dengan `rownames(tdm)`. Berdasarkan matriks diatas, ada banyak kasus data mining yang dapat dilakukan, misalnya *clustering*, klasifikasi, dan analisis asosiasi. Jika terdapat terlalu banyak suku kata, ukuran dari matriks *term-document* dapat dikurangi dengan memilih suku kata yang muncul dalam jumlah minimum pada dokumen, atau menyaring suku kata dengan TF-IDF (*term frequency-inverse document frequency*).

11.5 Suku Kata (*Term*) Paling Sering dan Asosiasi

Selanjutnya, hal perlu diperhatikan adalah kata populer atau yang sering muncul dan asosiasi antar kata.

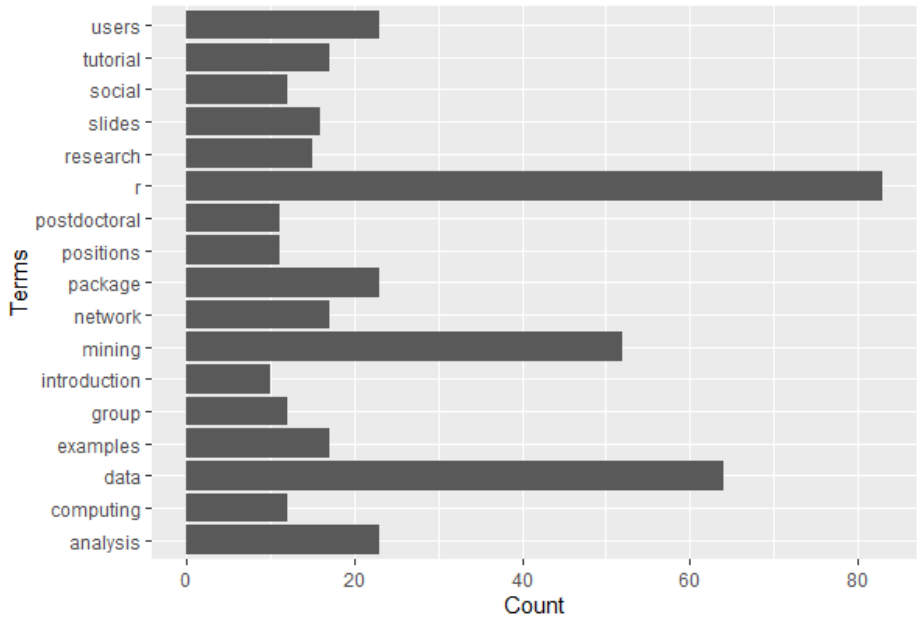
```
> # inspect frequent words
> findFreqTerms(tdm, lowfreq=10)
 [1] "data"         "positions"    "research"     "computing"
 [5] "r"            "package"      "tutorial"     "users"
 [9] "slides"       "mining"       "postdoctoral" "analysis"
[13] "network"      "social"       "introduction" "examples"
[17] "group"
```

Pada sintaks diatas, `findFreqTerms()` menemukan suku kata yang sering dengan frekuensi tidak kurang dari sepuluh. Perhatikan bahwa hasil diatas diurutkan bukan berdasarkan frekuensi atau popularitas.

Untuk menampilkan kata-kata yang paling sering secara visual, selanjutnya dibuat diagram batang untuk hasil diatas. Dari matriks *term-document*, pengguna dapat merubahnya menjadi matriks frekuensi suku kata dengan `rowSums()`. Kemudian memilih suku kata yang muncul dalam sepuluh atau lebih dokumen dan menyajikannya dalam bentuk diagram batang menggunakan *package* `ggplot2` (Wicham,2009). Pada sintaks dibawah ini, `geom="bar"` berarti menampilkannya dalam bentuk diagram batang dan `coord_flip()` menukar sumbu x dan y. Diagram batang pada Gambar 12.1 dengan jelas menunjukkan bahwa tiga kata yang paling sering digunakan adalah *"r"*, *"data"* dan *"mining"*.

```
> termFrequency <- rowSums(as.matrix(tdm))
> termFrequency <- subset(termFrequency, termFrequency>=10)
> library(ggplot2)
> df <- data.frame(term=names(termFrequency), freq=termFrequency)
```

```
> ggplot(df, aes(x=term, y=freq)) + geom_bar(stat="identity") +
+       xlab("Terms") + ylab("Count") + coord_flip()
```



Gambar 11.1 Suku kata paling sering digunakan

Sebagai alternatif, plot diatas juga dapat digambar dengan `barplot()` seperti dibawah ini, dimana arah label sumbu x menjadi vertikal.

```
> barplot(termFrequency, las=2)
```

Fungsi `findAssocs()` digunakan untuk melihat tingkat keterkaitan suku kata dengan suku kata lainnya. Di bawah ini akan diperlihatkan suku kata yang terkait dengan *"r"* (atau *"mining"*) dengan korelasi tidak kurang dari 0,25, dan kata-kata tersebut diurutkan berdasarkan korelasinya dengan *"r"* (atau *"mining"*).

```
> # Kata mana yang terkait dengan "r"?
> findAssocs(tdm, "r", 0.25)
$r
   users canberra     list     cran examples
    0.34     0.26     0.26     0.26     0.25

> # Kata mana yang terkait dengan "mining"?
> findAssocs(tdm, "mining", 0.25)

$mining
          data         mahout recommendation           sets
          0.54           0.39           0.39           0.39
      supports       frequent        itemset           card
```

```
      0.39          0.35          0.34          0.29
 functions     reference          text
      0.29          0.29          0.26
```

11.6 *Word Cloud*

Setelah membangun matriks *term-document*, bisa dilihat pentingnya kata-kata disajikan dalam bentuk *word cloud* (juga dikenal sebagai *tag cloud*), yang dapat dengan mudah dibuat dengan *package* `wordcloud`. Pada sintaks dibawah ini, pertama-tama ubah matriks *term-document* menjadi matriks normal, kemudian menghitung frekuensi kata. Setelah itu, menetapkan tingkat abu-abu berdasarkan frekuensi kata dan gunakan `wordcloud()` untuk membuat plotnya. Dengan `wordcloud()`, dua parameter pertama memberikan daftar kata dan frekuensinya. Kata-kata dengan frekuensi dibawah tiga tidak diplot, seperti yang ditentukan pada `min.freq=3`. Dengan menyetel `random.order=F`, kata-kata yang paling sering digunakan diplot terlebih dahulu, yang membuatnya muncul di tengah awan. Pengaturan warna ke tingkat abu-abu berdasarkan frekuensi. Awan warna-warni dapat dihasilkan dengan mengatur warna dengan `rainbow()`.

```
> library(wordcloud)
> m <- as.matrix(tdm)

> # menghitung frekuensi kata dan urutkan dari frekuensi terbesar
> wordFreq <- sort(rowSums(m), decreasing=TRUE)

> # warna
> pal <- brewer.pal(9, "BuGn")
> pal <- pal[-(1:4)]

> # word cloud
> set.seed(375)
> grayLevels <- gray( (wordFreq+10) / (max(wordFreq)+10) )


> wordcloud(words=names(wordFreq), freq=wordFreq, min.freq=3,
+       random.order=F, colors=pal)
```

Gambar 11.2 *Word Cloud*

Word cloud diatas dengan jelas menunjukkan bahwa *"r"*, *"data"* dan *"mining"* adalah tiga kata teratas, yang menjelaskan bahwa *tweet @RDataMining* menyajikan informasi tentang R dan data mining. Beberapa kata penting lainnya adalah *"analysis"*, *"examples"*, *"slides"*, *"tutorial"* dan *"package"*, yang menunjukkan fokusnya pada dokumen dan contoh pada analisis dan *package* R. Kumpulan kata lain yang sering digunakan, *"research"*, *"postdoctoral"* dan *"position"*, berasal dari *tweet* tentang program doktoral dan posisi penelitian. Ada juga beberapa *tweet* tentang topik analisis jejaring sosial, seperti yang ditunjukkan oleh kata *"network"* dan *"social"* dalam *cloud*.

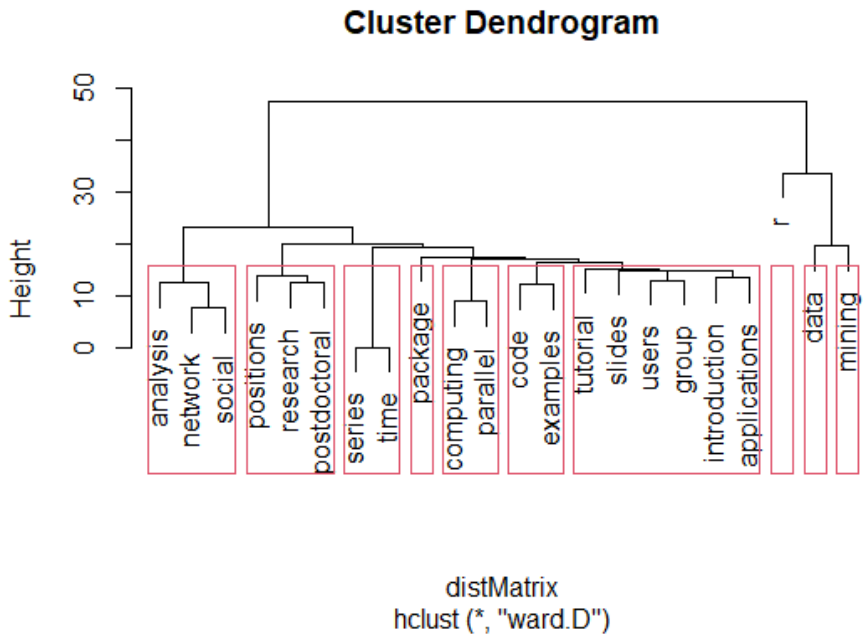
11.7 Pengelompokkan (*Clustring*) Kata

Selanjutnya akan diperlihatkan cara menemukan kelompok kata dengan pengelompokan hierarkis (*hierarchical clustering*). Suku kata yang jarang muncul akan dihilangkan, sehingga plot *clustering* tidak akan dipenuhi kata-kata. Kemudian jarak antar suku kata dihitung dengan `dist()` setelah penskalaan. Setelah itu, suku kata tersebut dikelompokkan dengan `hclust()` dan dendrogramnya dipotong menjadi 10 cluster. Metode aglomerasi diatur menjadi *ward*, yang menunjukkan peningkatan varians ketika dua cluster digabungkan.

Beberapa opsi lain adalah *single linkage*, *complete*, *linkage*, *average linkage*, *median* dan *centroid*. Detail tentang berbagai metode aglomerasi dapat ditemukan di buku teks data mining (Han and Kamber, 2000; Hand dkk, 2001; Witten and Frank, 2005)

```
> # menghilangkan sparse terms
> tdm2 <- removeSparseTerms(tdm, sparse=0.95)
> m2 <- as.matrix(tdm2)

> # pengelompokan suku
> distMatrix <- dist(scale(m2))
> fit <- hclust(distMatrix, method="ward.D")

> plot(fit)
> # potong pohon kedalam 10 kelompok
> rect.hclust(fit, k=10)
> (groups <- cutree(fit, k=10))
```

```
        data    positions     research    computing     parallel
           1            2            2            3            3
           r      package     tutorial        users       slides
           4            5            6            6            6
      mining postdoctoral     analysis      network       social
           7            2            8            8            8
introduction         code     examples applications        group
           6            9            9            6            6
      series         time
          10           10
```



Gambar 11.3 Pengelompokan Kata

Dari dendrogram diatas, bisa dilihat topik pada *tweet*. Kata *"analysis"*, *"network"* dan *"social"* dikelompokkan menjadi satu kelompok, karena ada beberapa *tweet* pada analisis jaringan sosial. Kelompok kedua dari kiri terdiri dari *"positions"*, *"postdoctoral"* dan *"research"*, dan dikelompokkan menjadi satu kelompok karena *tweet* tentang lowongan penelitian dan posisi *postdoctoral*. Pengguna juga bisa melihat kelompok runtun waktu (*time series*), *package* R, komputasi paralel (*parallel computing*), kode (*code*) dan contoh (*examples*) R, serta *tutorial* dan *slides*. Tiga cluster paling kanan terdiri dari *"r"*, *"data"* dan *"mining"*, yang merupakan kata kunci dari *tweet @RDataMining*.

11.8 Pengelompokkan (*Clustring*) *Tweet*

Tweet dibawah dikelompokkan dengan algoritma *k-means* dan algoritma *k-medoids*.

11.8.1 Pengelompokkan (*Clustring*) *Tweet* dengan Algoritma *k-means*

Pertama, mengubah matriks *term-document* menjadi matriks *document-term*. *Tweet* tersebut kemudian dikelompokkan dengan `kmeans()` dengan jumlah kelompok diatur menjadi delapan. Setelah itu, memeriksa kata-kata populer di setiap kelompok dan juga pusat *cluster*.

```
> # transpos matriks
> m3 <- t(m2)
> set.seed(122)
> # k-means clustering dari tweets
> k <- 8
> kmeansResult <- kmeans(m3, k)
> # pusat kelompok
> round(kmeansResult$centers, digits=3)

   data positions research computing parallel     r package tutorial
1 0.500     0.400    1.300     0.000    0.000 0.000   0.100    0.000
2 0.000     0.000    0.000     0.000    0.000 1.333   0.250    0.083
3 1.296     0.037    0.037     0.037    0.037 0.444   0.148    0.000
4 0.938     0.000    0.000     0.000    0.000 1.000   0.188    0.062
5 0.000     0.000    0.000     0.267    0.200 1.200   0.200    0.167
6 0.125     0.100    0.000     0.075    0.025 0.000   0.125    0.000
7 0.800     0.000    0.000     0.000    0.000 0.000   0.000    1.200
8 0.000     0.143    0.071     0.000    0.000 0.214   0.071    0.286


  users slides mining postdoctoral analysis network social
1 0.000  0.000  0.100        0.400    0.000   0.000  0.100
2 1.000  0.333  0.250        0.000    0.250   0.083  0.000
3 0.148  0.037  1.185        0.111    0.000   0.000  0.000
4 0.125  0.062  0.312        0.000    0.250   0.062  0.000
5 0.067  0.100  0.033        0.000    0.033   0.033  0.000
6 0.050  0.125  0.125        0.050    0.075   0.000  0.000
7 0.000  0.200  0.800        0.000    0.000   0.000  0.000
8 0.071  0.071  0.071        0.143    0.857   1.000  0.786

  introduction  code examples applications group series  time
1        0.000 0.000    0.000        0.100 0.000  0.000 0.000
2        0.000 0.000    0.167        0.000 0.667  0.167 0.167
3        0.037 0.000    0.000        0.222 0.037  0.000 0.000
4        0.062 0.312    0.562        0.000 0.000  0.188 0.188
5        0.033 0.100    0.100        0.033 0.033  0.000 0.000
6        0.125 0.025    0.050        0.025 0.025  0.075 0.075
7        0.000 0.000    0.000        0.000 0.200  0.000 0.000
8        0.143 0.000    0.071        0.000 0.000  0.000 0.000
```

Untuk memudahkan mengetahui tentang klusternya, dilakukan pemeriksaan tiga kata teratas di setiap kluster.

```
> for (i in 1:k) {
+   cat(paste("cluster ", i, ": ", sep=""))
+   s <- sort(kmeansResult$centers[i,], decreasing=T)
+   cat(names(s)[1:3], "\n")
+ }
```

```
cluster 1: research data positions
cluster 2: r users group
cluster 3: data mining r
cluster 4: r data examples
cluster 5: r computing parallel
cluster 6: data package slides
cluster 7: tutorial data mining
cluster 8: network analysis social
```

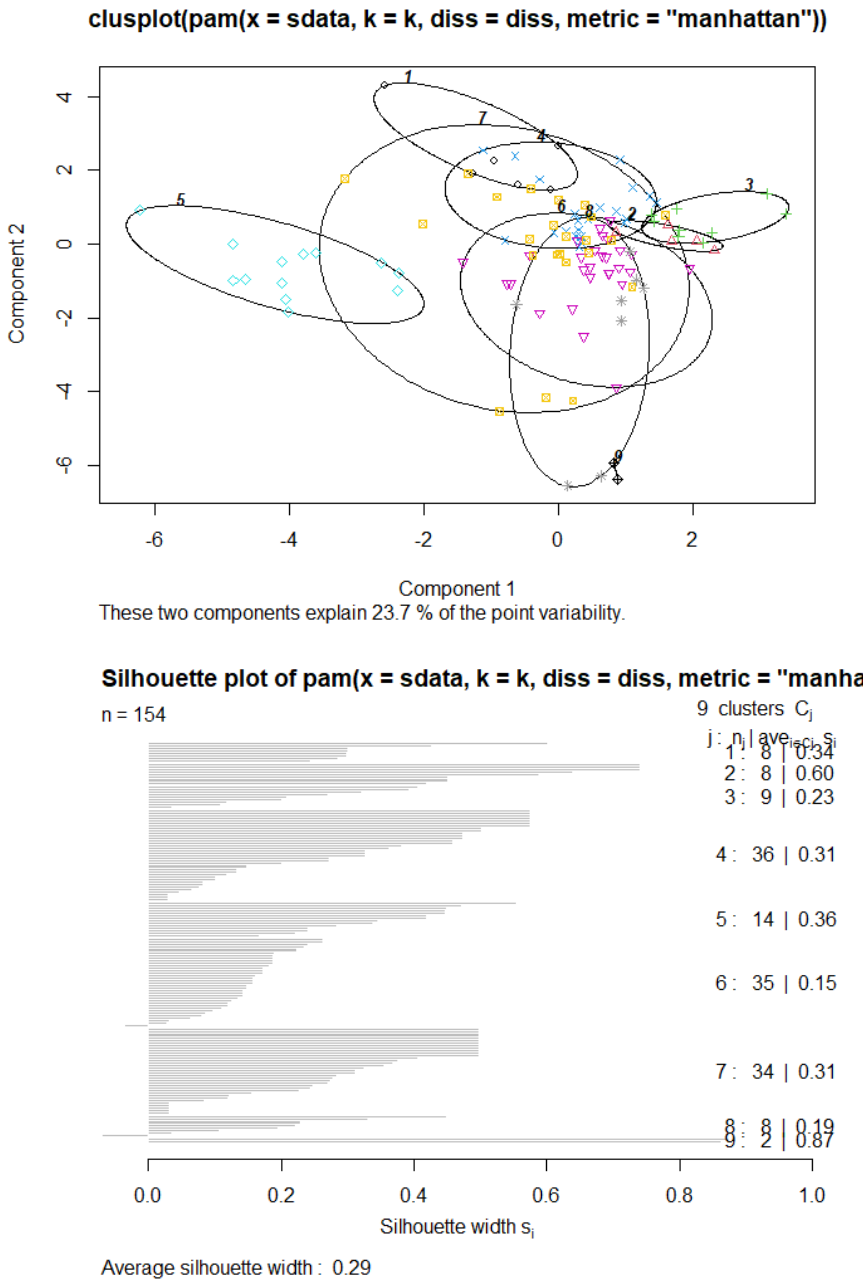
Dari kata-kata teratas dan pusat *cluster* di atas, dapat dilihat bahwa *cluster* memiliki topik yang berbeda. Misalnya, *cluster* 4 berfokus pada *data* dan *examples* R, *cluster* 3 fokus pada *data mining* dengan R, *cluster* 5 fokus pada *computing parallel* di R, *cluster* 6 fokus pada *package* R dan *cluster* 7 fokus pada *tutorial*. Dapat juga dilihat bahwa *cluster* 1, 7 & 8 tidak fokus pada R. *Cluster* 1, 7 & 8 adalah tentang informasi umum tentang data mining dan tidak dibatasi oleh R. *Cluster* 8 fokus pada analisis jejaring sosial, *cluster* 7 tentang *data mining tutorial*, dan cluster 1 tentang posisi untuk penelitian data mining.

11.8.2 Pengelompokkan (*Clustring*) *Tweet* dengan Algortima *k-medoids*

Selanjutnya akan diperlihatkan cara pengelompokan *k-medoids* dengan algoritma *Partitioning Around Medoids* (PAM), yang menggunakan *medoid* (objek perwakilan) sebagai ganti rata-rata untuk mewakili *cluster*. Pengelompokan ini lebih tahan terhadap *noise* dan *outlier* daripada pengelompokan *k-means*, dan memberikan tampilan plot *silhouette* untuk menunjukkan kualitas pengelompokan. Pada contoh dibawah ini, akan digunakan fungsi `pamk()` dari *package* `fpc` (Hennig, 2015), yang memanggil fungsi `pam()` dengan jumlah *cluster* yang diperkirakan dengan *silhouette* rata-rata optimal.

```
> library(fpc)
> pamResult <- pamk(m3, metric="manhattan")
> # jumlah cluster teridentifikasi
> (k <- pamResult$nc)

[1] 9

> pamResult <- pamResult$pamobject
> # cetak cluster medoids
> for (i in 1:k) {
+    cat(paste("cluster", i, ": "))
```

```
+    cat(colnames(pamResult$medoids)[
+          which(pamResult$medoids[i,]==1)], "\n")
+ }

cluster 1 : data positions research
cluster 2 : computing parallel r
cluster 3 : r package mining
cluster 4 : data mining
cluster 5 : tutorial analysis network social
cluster 6 : r
cluster 7 :
cluster 8 : r mining code examples
cluster 9 : users mining analysis group series time

> # plot hasil clustering
> layout(matrix(c(1,2),2,1)) # mengatur 2 grafik per halaman
> plot(pamResult, color=F, labels=4, lines=0, cex=.8, col.clus=1,
+      col.p=pamResult$clustering)
> layout(matrix(1))
```



Gambar 11.4 *Cluster Tweet*

Pada Gambar 11.4, diagram pertama adalah *"clusplot"* (plot pengelompokan) 2D dari *k* cluster, dan diagram kedua menunjukkan *silhouette*nya. Dengan *silhouette* tersebut, nilai s_i besar (hampir 1) menunjukkan bahwa pengamatan terkait terkluster dengan sangat baik , nilai s_i kecil (sekitar 0) berarti pengamatan berada di antara dua *cluster*, dan pengamatan dengan nilai s_i negatif kemungkinan ditempatkan pada *cluster* yang salah. Lebar *silhouette* rata-rata

adalah 0.29, yang menunjukkan bahwa *cluster* tidak terpisah dengan baik satu sama lain.

Hasil diatas dan Gambar 11.4 menunjukkan bahwa terdapat sembilan *cluster tweet*. *Cluster* 2 dan 9 adalah kelompok yang terpisah dengan baik, dengan masing-masing berfokus pada topik tertentu. *Cluster* 8 memiliki lebar *silhouette* negatif, yang berarti kemungkinan lebih baik berada dalam *cluster* lain daripada di *cluster* 8. Untuk meningkatkan kualitas *clustering*, lakukan pengaturan range jumlah *cluster* `krange=2:8` saat memanggil `pamk()`.

DAFTAR PUSTAKA

Agrawal, R. and Srikant, R. (1994). Fast algorithms for mining as- sociation rules in large databases. In Proc. of the 20th International Conference on Very Large Data Bases, pages 487–499, Santiago, Chile.

Aldrich, E. (2013). wavelets: A package of functions for computing wavelet filters, wavelet transforms and multiresolution analyses. R package version 0.3-0.

Breunig, M. M., Kriegel, H.-P., Ng, R. T., Sander, J. 2000. LOF: identifying density-based local outliers. In SIGMOD '00: Proceedings of the 2000 ACM SIG-MOD international conference on Management of data. New York. ACM Press.

Ester, M., Kriegel, H.-P., Sander, J., and Xu, X. 1996. A density-based algorithm for discovering clusters in large spatial databases with noise.

Feinerer, I. and Hornik, K. (2015). tm: Text Mining Package. R package version 0.6-2.

Fellows, I. (2014). wordcloud: Word Clouds. R package version 2.5.

Gentry, J. (2015). twitteR: R based Twitter Client. R package version 1.1.9.

Giorgino, T. (2009). Computing and visualizing dynamic time warping alignments in R: The dtw package. Journal of Statistical Software, 31(7):1–24.

Han, J. and Kamber, M. (2000). Data Mining: Concepts and Techniques. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Hand, D. J., Mannila, H., and Smyth, P. (2001). Principles of Data Mining (Adaptive Computation and Machine Learning). The MIT Press.

Hahsler, M., Gruen, B., and Hornik, K. (2014). arules: Mining Association Rules and Frequent Itemsets. R package version 1.1-6.

Hennig, C. (2015). fpc: Flexible Procedures for Clustering. R package version 2.1-10.

Hothorn, T., Hornik, K., Strobl, C., and Zeileis, A. (2015). Party: A lab- oratory for recursive partytioning. http://cran.r-project.org/web/packages/party/. R package version 1.0-23.

Keogh, E. J. and Pazzani, M. J. (2001). Derivative dynamic time warping. In the 1st SIAM Int. Conf. on Data Mining (SDM-2001), Chicago, IL, USA.

Lang, D. T. and the CRAN team (2015). RCurl: General Network (HTTP/FTP/...) Client Interface for R. R package version 1.95-4.7.

Maechler, M., Rousseeuw, P., Struyf, A., Hubert, M., and Hornik, K. (2015). cluster: Cluster Analysis Basics and Extensions. R package version 2.0.3 — For new features, see the 'Changelog' file (in the package source).

R Core Team (2015). R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria.

R Core Team. 2020. R: A Language and Environment for Statistical Computing. Vienna, Austria: R Foundation for Statistical Computing. https://www.R-project.org/.

Torgo, L. (2010). Data Mining with R, learning with case studies. Chapman and Hall/CRC.

The Institute of Statistical Mathematics (2012). timsac: TIMe Series Analysis and Control package. R package version 1.2.7.

Wickham, H. (2009). ggplot2: elegant graphics for data analysis. Springer New York.

Wickham, Hadley. 2015. Advance R. Boca Raton, Florida: Chapman & Hall/CRC. http://adv-r.had.co.nz/.

Witten, I. and Frank, E. (2005). Data mining: Practical machine learning tools and techniques. Morgan Kaufmann, San Francisco, CA., USA, second edition.

Zaki, M. J. (2000). Scalable algorithms for association mining. IEEE Transactions on Knowledge and Data Engineering, 12(3):372–390.

Zhou, Yanchang. 2015. R and Data Mining: Examples and Case Studies. Elsevier.