

# DATA SCIENCE WITH R TEXT MINING



Course Book / Module  
Data Science with R  
Text Mining

# Text Mining With R

By Sadasa Academy

## Chapter 1

### Pengenalan Dasar R

Pada dasarnya R merupakan program pengolahan statistik menggunakan pendekatan coding. Namun, dalam prakteknya R juga banyak digunakan untuk mengolah data tidak terstruktur, termasuk data dari media sosial. R merupakan salah satu *open source* yang saat ini cukup populer dan banyak digunakan oleh berbagai organisasi dengan skala besar hingga kecil. Pengguna R tersebar mulai dari perusahaan teknologi besar dunia (*tech giants*) seperti Google dan Facebook, pemerintahan, akademisi, hingga usaha kecil menengah. R sangat popular dikalayak umum dikarenakan R menyediakan berbagai macam library yang dapat digunakan oleh siapa saja. Dengan demikian, kita dapat menganalisis data dan memvisualisasikannya dengan mudah. Dalam pembelajaran ini, kita akan menggunakan model yang sudah umum digunakan pada data science.

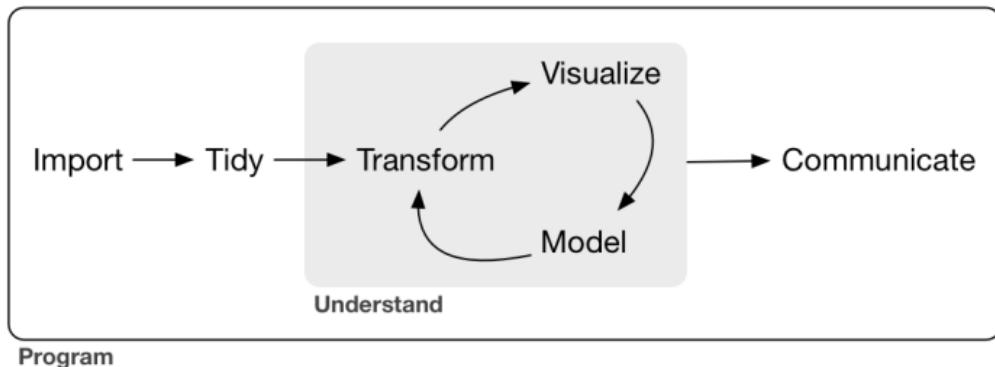


Figure 1: Diagram alir project data science

Pertama, kita harus meng-**import** data ke R. Artinya, Anda harus mengambil data yang sudah disimpan sebelumnya di dalam sebuah folder file, database, atau web API, dan memuatnya ke dalam sebuah *dataframe* atau objek di R.

Sesudah berhasil mengimpor data, langkah selanjutnya adalah membereskan data, atau sering disebut dengan **tidy**. *Tidying data* berarti menyimpan data kita ke dalam sebuah bentuk yang konsisten yang sesuai antara semantik dari data yang Anda miliki dengan cara penyimpanan data tersebut. Secara singkat, jika data Anda telah *tidy* (rapi, teratur), setiap kolom adalah sebuah variabel, dan setiap baris adalah sebuah observasi. Data yang *tidy* sangatlah penting karena dengan struktur data yang konsisten, kita hanya perlu berkonsentrasi untuk menjawab pertanyaan-pertanyaan seputar data Anda, bukannya untuk mengambil data ke dalam format yang tepat untuk fungsi-fungsi yang berbeda.

Setelah mendapat data yang rapi dan teratur (*tidy*), langkah selanjutnya adalah melakukan **transformasi** terhadap data tersebut. Transformasi data dapat digambarkan sebagai observasi dari kepentingan kita (misalnya data semua orang dalam suatu kota atau semua data dengan batasan waktu tertentu), membuat variabel baru yang merupakan fungsi dari variabel yang sudah ada sebelumnya (misalnya menghitung variabel kecepatan dari variabel jarak dan waktu), dan menghitung fungsi-fungsi statistik (mean, median, modus). Proses **tidy** dan **transformasi** ini sering disebut dengan **wangling**, karena mengubah bentuk data Anda menjadi bentuk yang siap diolah seringkali terasa seperti sebuah pertarungan (*wangling*)!

Setelah mendapatkan data yang rapi dan teratur (*tidy*) dengan variabel yang dibutuhkan, ada dua cara untuk melakukan penggalian **knowledge** dari data: visualisasi dan pemodelan. Kedua cara ini memiliki kelebihan dan kekurangan yang saling berkomplemen, sehingga dalam analisis yang riil biasanya akan berulang di antara kedua cara ini.

**Visualization** adalah kegiatan fundamental manusia. Visualisasi yang baik akan menunjukkan sesuatu yang tidak diduga sebelumnya, atau menimbulkan pertanyaan baru seputar data Anda. Visualisasi yang baik juga akan memberikan isyarat bahwa Anda telah mengajukan pertanyaan yang salah atau harus mengumpulkan data lain lagi.

**Model** adalah perkakas komplementer dari visualisasi. Ketika Anda dapat mengajukan pertanyaan yang cukup presisi, Anda dapat menggunakan sebuah model untuk menjawab pertanyaan tersebut. Model secara fundamental adalah fungsi matematis, sehingga dapat *scaling* dengan baik.

Langkah terakhir dari *data science* adalah **communication** yang merupakan bagian paling kritis dari seluruh proyek analisis data. Tidak peduli sebaik apa pemodelan atau visualisasi yang Anda buat, jika Anda tidak bisa mengkomunikaskan hasilnya kepada orang lain, berarti proyek data analisis Anda berakhiran sebagai sebuah kegagalan.

## Menggunakan R dan RStudio

R dan RStudio adalah dua hal yang berbeda tapi digunakan untuk fungsi yang sama. RStudio di sini berfungsi untuk menjalankan skrip-skrip R yang kita buat. Saat mayoritas pengguna R menggunakan RStudio. Dua-duanya dapat diunduh dan digunakan secara gratis melalui tautan berikut.

1. R dapat diunduh di: <https://cran.r-project.org/>
2. RStudio dapat diunduh di: <https://www.rstudio.com/>

Setelah kedua perangkat lunak di atas terunduh lalu install R **terlebih dahulu**, kemudian RStudio. Setelah keduanya terinstall, kita akan lebih banyak menggunakan RStudio. Untuk itu langkah selanjutnya adalah mengenal bagian-bagian dari R Studio, yang secara umum memiliki empat bagian antar muka seperti dapat dilihat pada gambar @ref(fig:RstudioInt).

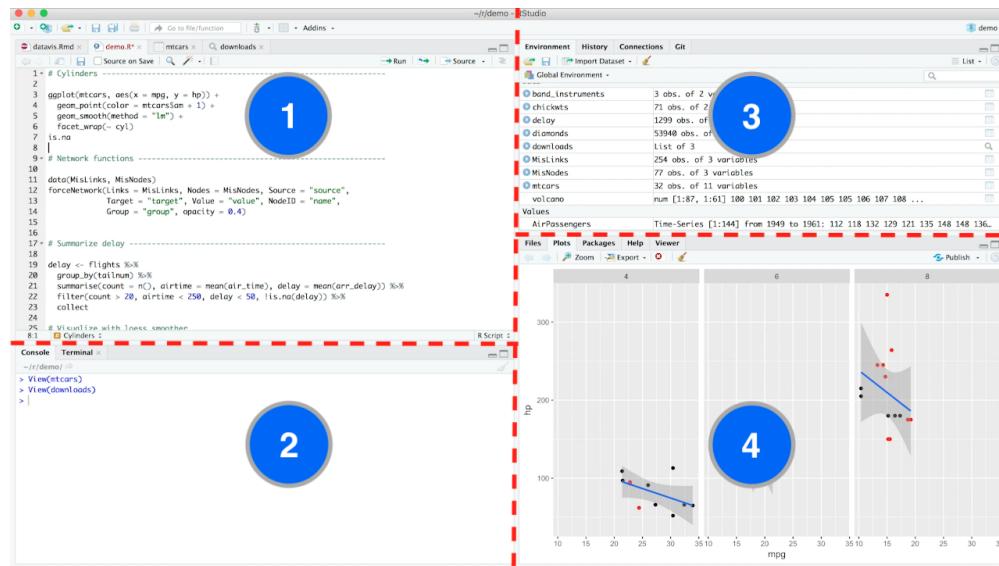


Figure 2: RStudio Interface terdiri empat bagian utama.

Seperti dapat dilihat pada gambar @ref(fig:RstudioInt), RStudio memiliki empat bagian utama. Namun, ketika pertama kali membukanya atau tidak ada skrip yang terbuka RStudio hanya menunjukkan tiga bagian, yaitu bagian 2, 3, dan 4. Bagian 1 merupakan bagian untuk menampilkan skrip. Untuk membuat skrip di RStudio kita dapat menggunakan cara pintas, yaitu dengan menekan tombol **command + shift + N** di Mac atau **control + shift + N** di Windows. Walaupun demikian, agar pekerjaan kita terkelola dengan baik kita bisa mengikuti langkah berikut dalam menggunakan RStudio.

1. **Membuat project** dengan menuju menu **File** (Bagian kiri atas)  $\rightarrow$  **New Project**  $\rightarrow$  **New Directory**  $\rightarrow$  **New Project**  $\rightarrow$  Pilih Disk/Folder untuk menyimpan pekerjaan  $\rightarrow$  **Isi bagian directory name** dengan nama project  $\rightarrow$  tekan tombol **create new**. Dengan membuat project tersebut, semua pekerjaan kita akan berada dalam satu folder yang dibuat pada kolom **directory name**.

- Membuat skrip baru.** Untuk membuat skrip baru kita akan menuju menu **File -> R Script** untuk membuat file **.R** (dot R). Selain itu, kita juga membuat jenis file lain seperti **R Markdown** maupun bahasa lain, misalnya **C++**. File skrip yang dibuat akan tampil pada bagian antar muka **nomor 1**.
- Membuat folder baru** khusus untuk menyimpan data mentah, data yang telah diolah, dan untuk menyimpan visualisasi yang dihasilkan. Folder-folder ini dapat dibuat seperti umumnya membuat folder sehari-hari yaitu dengan **klik kanan -> new folder** akan memudahkan kita dalam mengerjakan project di R.

Project yang kita buat dengan urutan di atas, dapat dilihat pada bagian antar muka **nomor 4**. Bagian antar muka **nomor 2** berfungsi untuk melihat skrip dan aktivitas yang dijalankan di RStudio. Sementara bagian antar muka **nomor 3** dapat digunakan untuk melihat data yang digunakan atau ada di lingkungan *project* yang dibuat. Di R hal semua aktivitas yang dilakukan tersimpan dalam sebuah tempat yang disebut **global environment**.

Selain hal-hal yang disebutkan di atas, dalam setiap halaman antarmuka juga memiliki beberapa fitur lain yang berfungsi untuk melihat dan mengontrol pekerjaan. Misalnya, pada bagian nomor 4 kita juga melihat hasil visualisasi data. Sementara pada bagian nomor 3 kita bisa mengatur koneksi langsung dengan aplikasi *database* seperti **spark** dan version control seperti **github** dan atau **gitlab**. Namun hal-hal tersebut tidak akan dibahas dalam modul ini.

## Mengenal Bahasa Pemrograman

Bahasa pemrograman adalah sebuah kumpulan perintah-perintah yang bisa dipahami oleh komputer. Mengapa menggunakan istilah **bahasa**? Karena hal tersebut dibuat agar manusia dapat berkomunikasi dengan komputer. Kita dapat memberikan perintah ke komputer untuk melakukan suatu tugas tertentu. Ada banyak sekali jenis bahasa pemrograman saat ini. Beberapa yang cukup populer sekarang di antaranya: C, C++, PHP, Java, Python, Swift, dan lain-lain. Memasuki era revolusi industri ke-4 ini, kebutuhan penggunaan komputer semakin masif, sehingga setiap orang diharapkan menguasai setidaknya satu bahasa pemrograman untuk mampu bersaing. R merupakan sebuah bahasa pemrograman yang relatif mudah dan sederhana untuk dikuasai orang-orang yang tidak memiliki latar belakang keilmuan terkait komputer. R mulai populer sekitar 5-10 tahun terakhir dalam kaitannya sebagai alat untuk mengolah data serta membuat visualisasi yang menarik.

## Dasar-dasar Bahasa pemrograman R

### R Sebagai Kalkulator

Sama seperti beberapa bahasa pemrograman lainnya, R melalui RStudio juga dapat berfungsi sebagai kalkulator. Untuk mencobanya, kita dapat langsung menuliskan skrip di bawah pada bagian Consol lalu tekan enter.

```
2 + 3 # the space around '+' is optional
2 * 3 #=> 6
sqrt(36) #=> 6, square root
log10(100) #=> 2, log base 10
10 / 3 #=> 3.3, 10 by 3
10 %/ 3 #=> 3, quotient of 10 by 3
10 %% 3 #=> 1, remainder of 10 by 3
```

Selain operasi sederhana seperti di atas, kita juga dapat menggunakan R untuk melakukan tugas yang lebih rumit sama seperti kalkulator saintifik. Untuk kesempatan ini, kita cukup tahu bahwa R bisa mengoperasikan berbagai fungsi matematika.

### Package dan Library

Packages atau Paket R merupakan kumpulan fungsi dan dataset yang dibuat oleh komunitas pengguna dan dapat digunakan secara gratis. Karena siapa saja dapat membuat paket, saat ini jumlahnya adalah puluhan

ribu dan bisa dicek pada tautan di atas. Di sini paket merupakan salah satu kekuatan *open source* karena dapat memudahkan pekerjaan kita dalam menyelesaikan sebuah proyek dengan R.

Sebagai contoh, ketika kita ingin membuat sebuah visualisasi dari data maka tanpa bantuan paket. Maka kita membutuhkan puluhan baris dan bahkan ratusan baris kode untuk satu gambar. Namun dengan bantuan paket, kita bisa menghemat energi dan terkadang hanya butuh beberapa baris kode saja.

Contohnya dalam pelatihan ini, misalnya kita ingin mengekstrak probabilitas topik yang terdapat dalam sebuah data teks. Di sini kita bisa menggunakan algoritma *Latent Dirichelet Allocation*(LDA). Tanpa bantuan paket, kita harus menyesuaikan persyaratan dan perhitungan serta jumlah iterasi yang dibutuhkan hingga sebuah model cukup meyakinkan. Tapi kita beruntung karena ada orang yang telah membuat paket dengan nama `topicmodels` [@topicmodels] yang sama persis menjalankan algoritma LDA. Oleh karena itu, kita bisa saja hanya butuh mempelajari fungsi dari paket tersebut.

### Cara Menginstall Paket

Terdapat dua cara untuk memasang atau menginstall paket dalam R. Tentunya di sini kita berbicara dalam konteks penggunaan RStudio. Pertama, dengan menggunakan *user interface*. Kedua, menggunakan perintah `install.packages()`. Kedua cara tersebut dapat dilakukan saat perangkat yang digunakan terkoneksi dengan internet.

Untuk menggunakan *user interface*, kita bisa menuju ke bagian atas dan mencari menu `Tools -> Install Pacakages... -> Isi nama paket -> Biarkan install dependencies tercentang`. Hal tersebut merupakan hal umum. Selain itu, di sini kita juga dapat menginstall dari sumber lain selain dari CRAN.

Untuk menginstall dengan menggunakan perintah `install.packages()` kita bisa mengetikkannya baik dalam skrip maupun langsung dalam console. Tanda kurung dalam fungsi di atas diisi dengan nama paket dalam format string atau karakter yang di tandai dengan tanda petik (Contoh: "itsAsAString"). Contohnya adalah sebagai berikut.

```
# install secara individual
install.packages("tidyverse")

# install secara bersamaan
paket <- c("tidyverse", "tidytext", "reshape2", "RColorBrewer", "scales", "haven",
          "topicmodels", "plotly", "DT", "knitr", "readxl", "missForest", "wordcloud")
install.packages(paket)
```

Perlu kita tahu bahwa tidak semua paket tersedia dalam CRAN, kadang kala kita harus melakukan instalasi dari sumber lain. Di sini kita juga bisa menginstall paket dari sumber lain, misalnya github atau biconductor. Berikut ini adalah beberapa fungsi yang dapat digunakan untuk menginstall paket.

- `install_bioc()` from Bioconductor,
- `install_bitbucket()` from Bitbucket,
- `install_cran()` from CRAN,
- `install_git()` from a git repository,
- `install_github()` from GitHub,
- `install_local()` from a local file,
- `install_svn()` from a SVN repository,
- `install_url()` from a URL, and
- `install_version()` from a specific version of a CRAN package.

### Cara Memanggil Paket

Kita harus memahami bahwa paket hanya perlu diinstall sekali, namun untuk menggunakannya perlu dipanggil setiap kali kita hendak menjalankan skrip yang menggunakan fungsi dari paket tertentu. Fungsi yang dapat digunakan untuk memanggil paket adalah `library(nama_paket)`. `nama_paket` diisi sesuai dengan nama paket yang akan digunakan. Beberapa fungsi tambahan terkait dengan paket berikut suatu waktu mungkin akan dibutuhkan.

- `installed.packages()` untuk mengecek paket yang terinstall
- `remove.packages("vioplot")` untuk menghapus paket yang terinstall
- `old.packages()` untuk mengecek paket yang perlu di perbarui
- `update.packages()` untuk mengupdate paket
- `detach()` untuk meng-*unload* sebuah paket

Karena banyaknya paket yang tersedia, terkadang ada fungsi yang sama dan menyebabkan konflik. Salah satu tanda yang sering muncul (dapat dilihat pada bagian **Console**) adalah adanya keterangan yang menyatakan fungsi tidak ada. Padahal kita sudah memanggil paket yang memiliki fungsi tersebut. Solusinya adalah adalah dengan memanggil fungsi dari paket yang spesifik dengan cara `package::function()`. Contohnya, `dplyr::count()` untuk menggunakan fungsi `count` dari paket `dplyr`.

### Cara Mendapatkan Paket yang Tepat

Cara utama untuk mendapatkan paket yang tepat sebenarnya hanya bisa dilakukan melalui percobaan secara terus-menurus hingga menemukan paket yang tepat dan sesuai dengan kebutuhan dan perangkat yang digunakan. Tapi mungkin beberapa langkah berikut ini bisa sedikit memudahkan menemukan data tentang paket yang dapat digunakan.

1. Membaca dokumentasi paket R di: [https://cran.r-project.org/web/packages/available\\_packages\\_by\\_date.html](https://cran.r-project.org/web/packages/available_packages_by_date.html)
2. Mencari paket berdasarkan kategori di: <https://cran.r-project.org/web/views/>
3. Mencari dokumentasi di: <https://www.rdocumentation.org/>

### Alur Kerja dalam R: Projects

Suatu hari, Anda menutup aplikasi R, kemudian mengerjakan hal lain, dan kembali ke pekerjaan di aplikasi R keesokan harinya. Pada waktu tertentu, Anda akan bekerja dengan banyak proyek bersama R secara simultan dan Anda menginginkan proyek tersebut diperlakukan secara terpisah. Atau di lain kesempatan, Anda ingin membawa data dari “dunia luar” ke R dan mengirimkan hasil analisisnya kembali ke “dunia luar tersebut”. Untuk menangani situasi di kehidupan nyata tersebut, Anda harus membuat dua keputusan:

1. Apa yang akan Anda simpan di dalam R sebagai *history* tentang semua yang terjadi?
2. Di mana analisis Anda “hidup”?

### Lingkungan kerja dalam RStudio

Sebagai pengguna R pemula, baik untuk menganggap lingkungan (seperti *object* yang ada di panel *environment*) sebagai hal yang “real”. Namun, dalam jangka panjang, lebih baik untuk menganggap R *scripts* sebagai yang “real”.

Dengan skrip R (dan berkas data), Anda dapat membuat ulang sebuah *environment*. Akan tetapi, sebaliknya, lebih sulit membuat ulang *script* R dari *environment* Anda! Anda harus mengetik ulang sebarisan panjang kode yang diambil dari ingatan anda (di mana kesalahan menjadi tidak terhindarkan) atau Anda harus menggali R *history* dengan tekun untuk mengembalikan *environment* yang Anda inginkan tersebut. Untuk mebiasakan hal tersebut, disarankan untuk mematikan fungsi preservasi *workspace* di antara *sessions*. Cara untuk melakukannya dapat mengikuti langkah pada gambar @ref(fig:RestoreData).

Di awal, hal ini mungkin akan menimbulkan *short-term pain*, karena ketika Anda me-restart RStudio, sekarang RStudio tidak akan mengingat lagi hasil-hasil analisis Anda beserta kode-kode programnya. Akan tetapi, “rasa sakit” jangka pendek ini akan menyelamatkan Anda dari “penderitaan” jangka panjang karena hal ini memaksa Anda untuk menangkap seluruh interaksi penting dalam seluruh proses pengkodean program Anda.

Ada dua *shortcut* keyboard yang akan berguna untuk menangkap seluruh bagian penting dari kode Anda di dalam editor:

1. Tekan `Cmd/Ctrl-Shift-F10` untuk me-restart RStudio.
2. Tekan `Cmd/Ctrl-Shift-S` untuk me-rerun skrip yang sedang dijalankan (*current script*).

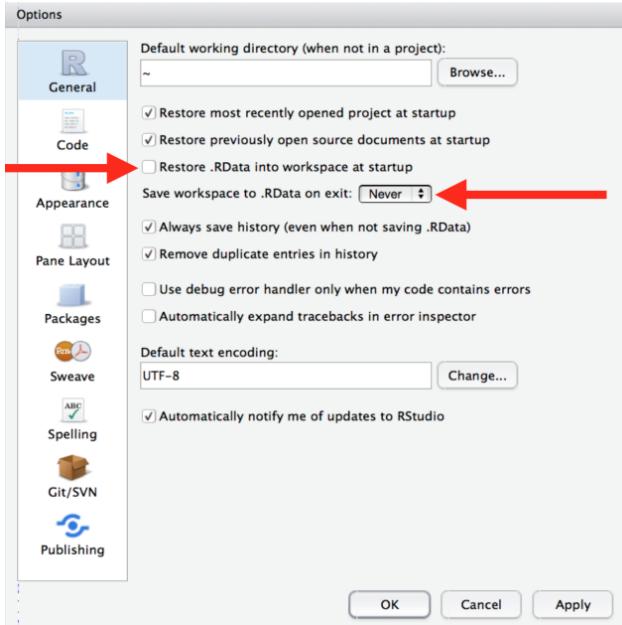


Figure 3: Uncheck pilihan 'Restore .RData into workspace at startup'

### Mengenal Direktori Kerja dalam RStudio

R memiliki *working directory*. *Working directory* adalah tempat di mana R mencari berkas-berkas yang ingin Anda buka dan lokasi seluruh penyimpanan file Anda. RStudio menampilkan *working directory* yang sedang aktif di atas Console seperti dapat dilihat pada gambar @ref(fig:workingdirectory), serta dapat dilihat melalui menjalankan fungsi `getwd()`.

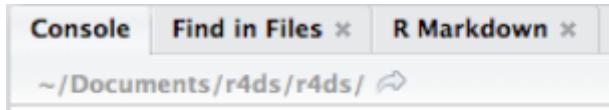


Figure 4: Letak Working directory di RStudio

Mulai sekarang, Anda harus mengorganisasikan project-project *data science* Anda ke dalam direktori-direktori yang terpisah dan ketika bekerja dalam sebuah R Projects, atur R *working directory* ke folder yang bersangkutan.

### Gaya Penulisan Direktori di RStudio

*Path* dan *directories* agak sedikit rumit karena ada dua gaya penulisan *path*, yakni antara sistem operasi Linux atau Windows. Berikut adalah perbedaan di antara keduanya:

1. Perbedaan yang utama adalah cara memisahkan komponen di dalam *path*. Pada Mac dan Linux, mereka menggunakan *slash* (contoh: `plots/diamonds.pdf`), sedangkan Windows menggunakan *backslash* (**contoh: `plots\iamonds.pdf`**). R dapat bekerja dengan dua jenis *slash* tersebut, namun sayangnya, *backslash* memiliki arti penting bagi R, dan untuk mendapatkan *single backslash* di dalam *path*, Anda harus menuliskan dua *backslash*. Hal ini membuat bingung, jadi disarankan untuk selalu menggunakan *path* dengan style Mac/Linux dengan *forward slash*.
2. *Absolute path* (contohnya *path* yang menunjukkan lokasi yang sama tanpa menghiraukan *working directory* Anda) terlihat berbeda. Di Windows dimulai dengan huruf *drive* (misalnya C:) atau dua *backslash* (e.g. `\servername`) dan di Mac/Linux dimulai dengan *slash* (misalnya `/user/hadley`). Jangan

pernah gunakan *absolute path*, karena tidak ada orang yang memiliki lokasi *directory* yang sama persis dengan kita.

3. Perbedaan selanjutnya adalah penggunaan tanda ~. Tanda ~ digunakan sebagai shortcut lokasi *home directory* kita. Windows tidak memiliki istilah *home directory*, sehingga tanda ~ akan mengarahkan ke lokasi Documents/My Documents Anda.

### Menggunakan menu Project di dalam RStudio

Seorang pengguna R yang *expert* selalu menyimpan semua *file* yang terkait dengan sebuah *project* dalam satu kesatuan: input data, R script, hasil analisis, dan gambar plot atau *figures*. Semua hal ini telah diakomodir oleh R lewat fitur Project.

Sekarang, mari kita buat Project pertama kita dengan mengklik menu **File-->New Project**, kemudian lihat Gambar @ref(fig:MakeProject1):

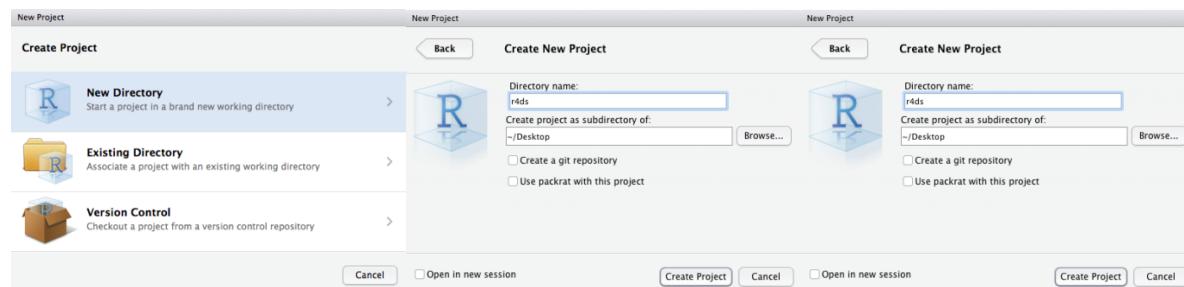


Figure 5: Membuat Project di R

Silahkan menuliskan directory names sebagai **r4ds** dan di kolom “Create project as subdirectory of:” sebagai lokasi di mana Anda akan menyimpan project Anda. Anda dapat mengecek directory aktif yang sedang berjalan dengan menggunakan fungsi `getwd()`.

### Jenis-jenis data yang umum digunakan

Seperti berbagai jenis bahasa lain, pada R kita juga mengenal tipe data untuk membantu kita dalam pengolahan data. Beberapa tipe data yang akan sering kita jumpai dalam R di antaranya:

#### Tipe dasar (atomic vector)

Ini merupakan tipe yang mendasari semua objek yang ada pada R. Pada praktiknya, yang akan banyak kita temui adalah **logical**, **numeric**, dan **character**. Tipe **integer** sebenarnya hanya bagian dari **numeric** yang dioptimalkan dalam penggunaan memori komputer. Sedangkan untuk **complex** dan **raw** akan sangat jarang kita jumpai dalam kasus sehari-hari.

Table 1: Berbagai tipe data yang sering digunakan di R

Tipe	Contoh
logical	TRUE, FALSE
numeric	12.3, 4, 15.09
character	satu, dua
integer	5L, 2L, 11L
complex	2+, 3i
raw	48, 65, 6c, 6c, 6f, (hex)

## Vectors

Vector merupakan sebuah tipe data gabungan yang berisi beberapa elemen atomic berjenis sama. Untuk membuat sebuah vector, digunakan perintah kombinasi `c()`. Contoh pembuatan vector adalah sebagai berikut:

```
a <- c(1,2,5.3,6,-2,4) # numeric vector
b <- c("one","two","three") # character vector
c <- c(TRUE,TRUE,TRUE,FALSE,TRUE,FALSE) #logical vector
```

Untuk mengakses elemen dalam vector, kita menggunakan kurung siku `[]` disertai perintah gabungan `c()` seperti contoh berikut:

```
# mengakses elemen nomer 2 dari vector a
print (a[c(3)])
```

## Lists

List merupakan tipe data gabungan yang mirip dengan Vector, namun di sini memungkinkan penggunaan elemen-elemen dengan tipe data berbeda. Sebagai contoh sebagai berikut:

```
w <- list(name="Fred", age=25, height=159.7)
x <- list("saya",5.4,1,TRUE)
```

Pada contoh di atas, terdapat dua cara dalam mendefinisikan List, yaitu dengan menyebutkan `key` seperti `name`, `age`, `height` atau secara langsung menyebutkan elemennya. Kemudian untuk mengakses elemen dalam List, kita dapat menggunakan dua cara berikut:

```
print(x[1]) # mengakses melalui urutan indeks
print(w['name']) # mengakses melalui key
```

## Data Frame

Data Frame merupakan tipe data yang akan sering kita jumpai dalam pengolahan data terstruktur. Pada dasarnya, *Dataframe* dapat kita anggap sebagai sebuah tabel yang sering kita gunakan. *Dataframe* secara sederhana dapat kita buat dengan menggabungkan beberapa Vector yang kita gunakan sebagai kolom. Contoh pembuatan *dataframe*:

```
id <- c(1,2,3,4) #vector kolom id
name <- c("tom", "jerry", "dora", "emon") #vector kolom name
score <- c(85.4,78.3,88.9,90) #vector kolom score

# membuat data frame dari kolom vector
mydata <- data.frame(id,name,score)

# menambahkan nama kolom
names(mydata) <- c("ID","Nama","Nilai")

print(mydata)
```

Pengolahan *dataframe* dalam R sangat banyak jenisnya, hal ini akan kita pelajari melalui contoh kasus. Namun perintah dasar untuk mengakses data dalam *dataframe* dapat kita amati pada contoh berikut:

```
# mengambil kolom 1 sampai 3
mydata[1:3]

# mengambil kolom dengan nama "ID" dan "Nilai"
mydata[c("ID","Nilai")]
```

```
# mengambil satu kolom dengan nama "Nilai"  
mydata$Nilai
```

## Fungsi-fungsi Dasar

### Fungsi `paste`

Terdapat dua fungsi `paste`, yaitu `paste()` dan `paste0()`. Pertama, `paste()` adalah cara untuk menggabungkan string dan menyesuaikan dengan pembatas (*delimiter*). Sementara `paste0()` tidak memberikan spasi pada string yang digabungkan. Cobalah jalankan satu-persatu baris beberapa contoh berikut dan amati perbedaan hasilnya.

```
teks1 <- "aku adalah anak gembala"  
teks2 <- "selalu riang serta gembira"  
  
teks12 <- paste(teks1, teks2, sep = " ")  
teks12  
  
teks0 <- paste0(teks1, teks2)  
teks0  
  
teks <- paste0(teks1, " ", teks2)  
teks
```

### Fungsi `if` dan `else`

Fungsi ini digunakan untuk memilih output sesuai dengan kondisi yang sudah ditentukan. Untuk lebih jelasnya mari perhatikan ilustrasi berikut:

Jika Nilai Tim A lebih dari Tim B, maka Tim A Menang

Dalam contoh di atas, nilai tim merupakan kondisi, sementara kalimat **Tim A Menang** merupakan hasil yang diinginkan jika kondisinya terpenuhi atau biasa disebut **statement**. Hal tersebut dapat dituliskan dalam skrip sebagai berikut:

```
Tim_A = 10  
Tim_B = 6  
  
if (Tim_A > Tim_B) {  
  print("Tim A Menang")  
}
```

Contoh pertama digunakan hanya dengan satu statement. Bagaimana jika kondisi yang terjadi justru sebaliknya, yaitu nilai Tim B lebih besar dari nilai Tim A. Artinya, justru Tim B yang menang. Dalam konteks ini, berarti kita memiliki dua kondisi, yaitu kondisi Tim A menang, atau sebaliknya Tim B Menang.

```
Tim_A = 7  
Tim_B = 9  
  
if (Tim_A > Tim_B) {  
  print("Tim A Menang")  
} else {  
  print("Tim B Menang")  
}
```

Bagaimana jika kondisinya berimbang atau Nilai Tim A dan B sama. Artinya, ada tiga kondisi yang harus menjadi pertimbangan, yaitu Tim A lebih dari Tim B, atau sebaliknya dan Nilai Tim A sama dengan Tim B.

Di dalam skrip tiga kondisi tersebut dapat direpresentasikan dengan menambahkan fungsi `else if` pada dua kondisi yang sudah dipelajari sebelumnya. Contohnya adalah sebagai berikut.

```
Tim_A = 8
Tim_B = 8

if (Tim_A > Tim_B) {
  print("Tim A Menang")
} else if(Tim_A == Tim_B) {
  print("Seri")
} else {
  print("Tim B Menang")
}
```

### Fungsi `for`-Loop

Walaupun di R sebenarnya ada fungsi lain yang lebih sederhana yang dapat digunakan sebagai pengganti `for` loop. Tapi kita juga perlu memahami konsepnya seperti dapat dilihat pada skrip di bawah ini.

```
# Create a vector filled with random normal values
u1 <- rnorm(30)
print("This loop calculates the square of the first 10 elements of vector u1")

# Initialize `usq`
usq <- 0

for(i in 1:10) {
  # i-th element of `u1` squared into `i`-th position of `usq`
  usq[i] <- u1[i] * u1[i]
  print(usq[i])
}

print(i)
```

### Fungsi `apply` family

Secara umum fungsi `apply()` digunakan untuk menghindari banyak pengulangan penulisan skrip atau fungsi untuk melakukan hal yang sama. Dalam manual R fungsi `apply()` dinyatakan sebagai berikut: `apply(X, MARGIN, FUN, ...)`, di mana: `X` is an array or a matrix if the dimension of the array is 2;

`MARGIN` is a variable defining how the function is applied: when `MARGIN=1`, it applies over rows, whereas with `MARGIN=2`, it works over columns. Note that when you use the construct `MARGIN=c(1,2)`, it applies to both rows and columns; and

`FUN`, which is the function that you want to apply to the data. It can be any R function, including a User Defined Function (UDF).

Hal di atas mungkin masih sedikit membingungkan. Sekarang coba bayangkan kita memiliki matrix  $5 \times 6$  dan ingin melihat jumlah tiap barisnya. Maka hal tersebut dapat dilakukan dengan skrip berikut.

```
X <- matrix(rnorm(30), nrow=5, ncol=6) # Construct a 5x6 matrix
apply(X, 2, sum) # Sum the values of each column with `apply()`

## [1] 1.8681962 -0.7881402 3.8308811 2.4977790 -2.1291879 3.9352520
```

`apply` dapat digunakan untuk matrix dan list, sementara `lapply()` juga memiliki fungsi yang mirip tapi juga dapat digunakan untuk data frame.

```

A <- c(1:5)
B <- c(6:20)
C <- c(7:15)

# Create a list of matrices
MyList <- list(A,B,C)

# Extract the 2nd column from `MyList` with the selection operator `[]` with `lapply()`
lapply(X = MyList, FUN = "[", 2)

# Extract the 1st row from `MyList`
lapply(MyList, "[", 1)

lapply(airquality, class) # return classes of each column in 'airquality' in a list
sapply(airquality, class) # return classes of each column in 'airquality'

```

Baris terakhir menggunakan `sapply()` yang digunakan untuk mengecek dengan fungsi `class()` pada setiap kolom data frame `airquality`.

#### Fungsi `str()`, `class()` dan `summary()`

Fungsi-fungsi di atas digunakan untuk melihat struktur, kelas, dan rangkuman data. `str()` bisa dibaca struktur, `class()` atau kelas digunakan untuk mengetahui jenis data yang ada dalam data, apakah ia data frame, list, matrix atau lainnya. Sementara `summary` akan lebih banyak digunakan saat melakukan eksplorasi data.

## Membuat Fungsi

Fungsi digunakan untuk menyederhanakan skrip yang ingin kita gunakan agar kita tidak perlu mengulang sintak yang sama untuk mengerjakan hal yang sama. Walaupun tidak ada pedoman umum yang dapat dijadikan patokan kapan kita perlu membuat fungsi. Namun secara umum, jika kita mengulang lebih dari tiga kali sintak yang sama maka sebaiknya membuatnya menjadi sebuah fungsi. Contohnya bisa dilihat dalam dua skenario operasi matematika berikut.

```

# Skenario dengan menggunakan fungsi
a <- 3
b <- 4
c <- 10
d <- 6

penambahan <- function(data1, data2, data3, data4) {
  A_tambah_B <- data1 + data2 + data3 / data4
  return(A_tambah_B)
}

tambah1 <- penambahan(a, b, c, d)
tambah2 <- penambahan(d, b, c, a)

# Skenario tanpa menggunakan fungsi
tambah3 <- a + b + c / d
tambah4 <- d + b + c / a

```

Mungkin, di dalam dua skenario yang dibuat kita belum terlalu banyak melihat kebermanfaatannya. Namun, bayangkan jika kita harus melakukannya lebih dari sepuluh kali. Peluang kita untuk membuat kesalahan penggunaan sintaks menjadi semakin besar. Kesalahan bisa terjadi misalnya salah menggunakan tanda yang

seharusnya + tapi malah - atau yang lainnya. Oleh karena itu, dengan membuat fungsi setidaknya kita telah meminimalisasi peluang terjadinya kesalahan yang disebabkan oleh kekurang telitian.

Selain itu, dengan membuat operasi matematika dalam contoh menjadi sebuah fungsi skrip yang kita buat menjadi lebih rapi dan mudah dibaca. Tentunya hal ini akan sangat bermanfaat saat kita telah menulis skrip yang banyak dan melibatkan operasi sintaks yang lebih kompleks namun konsisten.

Di dalam membuat fungsi terdapat tiga hal yang perlu dipehatikan seperti dapat dilihat pada Gambar @ref(fig:fungsi1), yaitu: (1) Argument; (2) Statement; dan (3) Return.

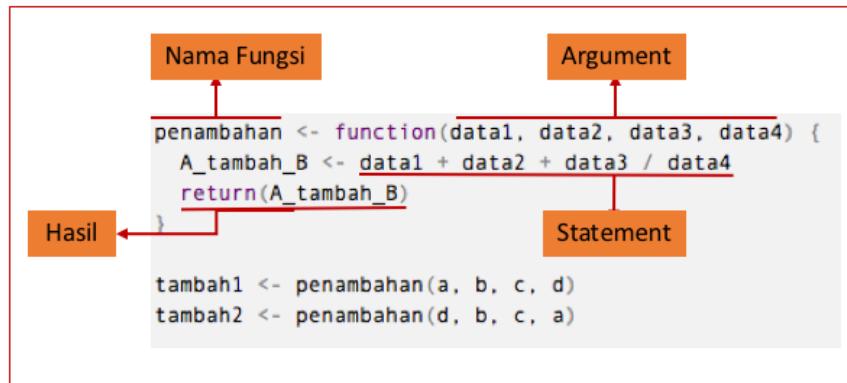


Figure 6: Anatomi sebuah fungsi dalam R

Di dalam fungsi pada Gambar @ref(fig:fungsi1) kita memiliki tiga bagian. Bagian pertama disebut **argument** yang berisi tentang hal apa saja yang akan di eksekusi dalam bagian berikutnya yang disebut **body** atau **statement**. Sebagaimana dapat dilihat, pada Gambar @ref(fig:fungsi1) disini kita memiliki empat argument yang perlu dipenuhi. Artinya, pada saat kita memanggil/menggunakan fungsi keempatnya harus dipenuhi.

Pada bagian **statement**, kita sebenarnya hanya menempatkan skrip R biasa untuk mengeksekusi tindakan yang akan dilakukan pada argument yang diberikan. Dalam konteks ini, kita ingin membagi hasil penjumlahan tiga variabel pertama dengan variabel keempat atau terakhir. Lalu, pada bagian terakhir, merupakan hasil yang ingin kita dapat dari fungsi yang dibuat.

Oleh karena itu, di dalam membuat fungsi kita sebaiknya juga mencobanya terlebih dahulu sebagai skrip biasa. Setelah bisa dipastikan skrip bisa berjalan tanpa error baru kemudian dijadikan sebuah fungsi.

## *Chapter 2*

### Teknik Pengambilan Data

Bagian ini akan membahas tentang dua teknik utama dalam pengambilan data dari internet. berfokus pada media sosial dan website, kita akan belajar cara mendapatkan data memanfaatkan API dan memahami metode *scraping* sederhana untuk laman statis. Setelah mengikuti materi ini peserta diharapkan mampu untuk mendapatkan dari media sosial, khususnya Twitter dan dari Website untuk kebutuhan analisis.

#### Pengambilan data dengan API

Secara teknis API (Application Programming Interface) adalah sebuah protokol komunikasi antara dua entitas. Sebagai ilustrasi, saat kita menyimpan sebuah lemari, kemudian lemarinya dikunci dengan sebuah gembok berkode. Maka hanya orang yang telah diberitahu dan mengetahui kombinasi kodennya saja yang bisa mengambil dan menggunakan benda dalam lemari tersebut. Prinsip yang kurang lebih sama diterapkan oleh berbagai perusahaan pengembang teknologi agar para developer dapat membuat aplikasi dengan memanfaatkan data yang mereka simpan di server.

Media sosial, seperti Facebook, Youtube, Twitter, dan portal-portal berita biasanya menyediakan API agar publik dapat mengakses apa yang mereka miliki. Tapi API di sini ada yang bersifat gratis, ada yang berbayar. Contohnya Twitter yang menyediakan beberapa tingkat API dengan fasilitas yang berbeda. Untuk lebih lengkapnya, API Twitter dapat dilihat di sini: <https://developer.twitter.com/en/pricing>.

Di sini kita akan mencoba menggunakan API BASIC dari Twitter dengan tujuan mendapatkan data aktivitas dan konten yang dilakukan oleh para pengguna Twitter. Namun sebelum berharap lebih jauh, API BASIC hanya bisa menyediakan data:

1. Sejauh 7 hari ke belakang dihitung dari saat penggunaan
2. Tidak bisa mengambil data dalam jumlah banyak
3. Memiliki beberapa keterbatasan fitur aktivitas yang dapat diambil.

Walaupun demikian, karena kemudahan penggunaannya penggunaan API BASIC merupakan salah satu hal yang bisa bermanfaat. Manfaat tersebut khususnya saat kita hendak melakukan observasi terhadap data di Twitter sebelum melakukan penelitian yang lebih mendalam.

#### Konsep

API atau Application Programming Interfaces sebenarnya bukan merupakan konsep baru. Konsep ini telah ada sejak tahun 1960an. Namun memang dalam beberapa waktu terakhir API menjadi salah satu topik yang banyak dibicarakan karena banyaknya aplikasi dan layanan berbasis data dan web seperti Facebook, Twitter, dan social media lainnya. Untuk memahaminya secara detail mari kita bedah singkatan API.

**A = Application.** Anggap saja kita adalah sebuah nasabah bank dan ingin mengambil uang melalui ATM. ATM di sini merupakan sebuah aplikasi. Di mana saat kita ke ATM kita berharap untuk dapat masuk ke akun bank dan melakukan transaksi seperti transfer atau mengambil uang dari tabungan. Seperti sebuah ATM, aplikasi juga menyediakan fungsi tertentu, tapi tidak melakukannya sendiri. Aplikasi butuh berkomunikasi, baik dengan pengguna (nasabah bank), maupun dengan bank yang diaksesnya. Secara sederhana, aplikasi menangani input dan output.

**P = Programming.** Sebuah aplikasi memungkinkan ATM untuk berkomunikasi dengan bank. Programming merupakan bagian dari aplikasi yang berfungsi untuk menerjemahkan input menjadi output. Dengan kata lain, pada saat kita memasukan input untuk menarik sejumlah uang dari ATM, program dalam ATM menerjemahkan permintaan kita dan menyampaikannya/mencocokannya dengan data yang ada dan dimiliki oleh bank terkait dengan nasabah tertentu.

**I = Interface.** Tampilan memastikan bagaimana kita berinteraksi dengan aplikasi. Dalam kasus ATM, tampilan berupa layar, keyboard, dan laci uang. Di mana melalui tampilan tersebut, setelah memasukkan kartu ATM, kita perlu memasukkan nomor pin (*key*), memilih layanan yang ingin digunakan, serta perlu

menginput hal lainnya sesuai dengan fitur yang ingin di manfaatkan. Dalam konteks APIs, hal yang sama juga diperlukan, yang diganti hanya pengguna menjadi perangkat lunak (*software*).

Sederhananya, API memungkinkan dua perangkat lunak untuk saling berkomunikasi berdasarkan ketentuan dan syarat yang telah ditentukan oleh pembuatnya.

### Membuat API Twitter

Untuk membuat API twitter kita harus memiliki akun Twitter. Selanjutnya menuju ke halaman <https://developer.twitter.com/en/apps> yang kemudian akan mengarahkan kita ke sebuah laman seperti tampak pada gambar @ref(fig:api1). Selanjutnya klik tombol `create an app` di sebelah kanan.

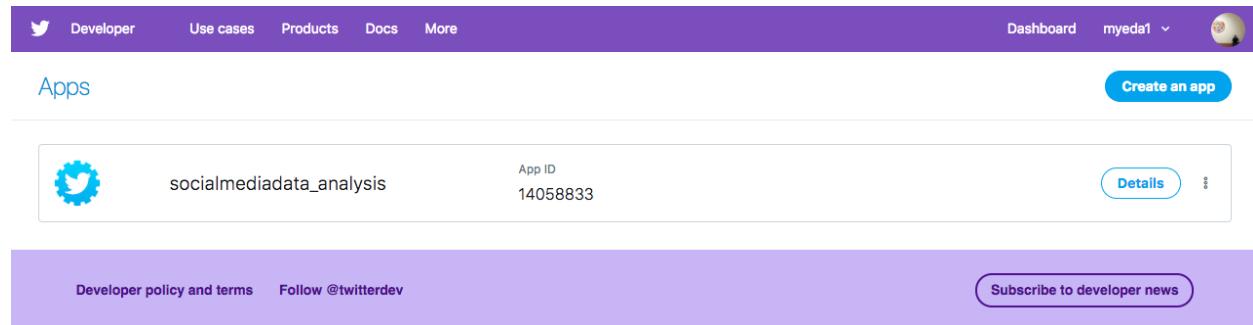


Figure 1: Halaman untuk membuat API Twitter

Pada bagian selanjutnya kita harus mengisi semua kolom yang diwajibkan seperti nama aplikasi, tujuan penggunaan dan lain-lain. Selain kolom yang diwajibkan, pada bagian ini juga kita perlu mengisi kolom **Callback URLs (required)** (lihat gambar @ref(fig:api2)) dengan <http://127.0.0.1:1410>.

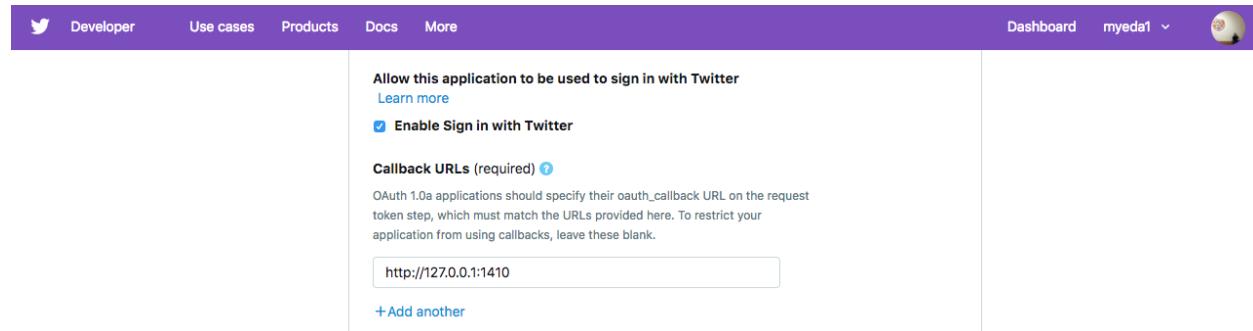


Figure 2: Pengisian kolom Callback URLs

Khusus pada bagian kolom **Tell us how this app will be used (required)** kita wajib mengisi tujuan pembuatan aplikasi secara detil. Contohnya adalah: *I am going to use this app to explore social movement on the Internet using Twitter data. In addition, I will also use this application to teach data mining in class.* Setelah semuanya terisi lalu tekan tombol `create` di bagian bawah dan akan memunculkan kota dialog baru seperti gambar @ref(fig:api3).

Sampai di sini, kita telah bisa membuat sebuah API Twitter basic. Selanjutnya untuk mendapatkan data dari Twitter kita perlu mendapatkan **Keys and tokens** dengan menekan tombolnya pada halaman baru yang terbuka. Bagian pertama seperti dapat dilihat pada gambar @ref(fig:api4).

Pada gambar @ref(fig:api4) terdapat dua bagian yaitu **Consumer API keys** dan **Access token & access token secret**. Namun pada saat pertama halaman ini terbuka bagian kedua masing kosong. Di sana ada tombol `create` untuk mendapatkan nomor yang dimaksud. Keempat nomor tersebutlah yang selanjutnya akan digunakan setiap kita mau mengambil data dari Twitter dengan menggunakan API BASIC.

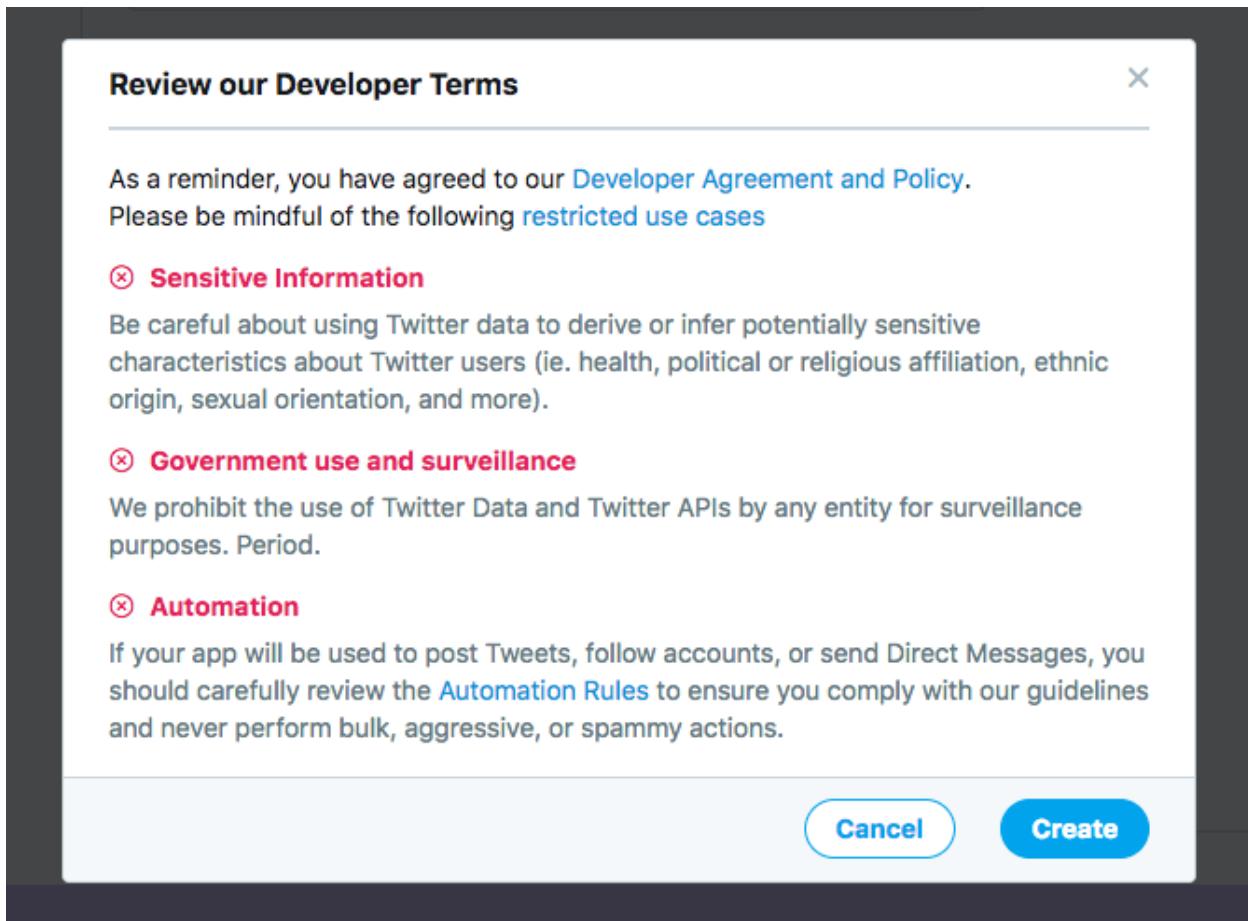


Figure 3: Tombol create API Twitter

The screenshot shows the "Keys and tokens" section of the Twitter developer dashboard. It displays Consumer API keys and Access token & access token secret, along with "Regenerate" and "Revoke" buttons.

**Keys and tokens**  
Keys, secret keys and access tokens management.

**Consumer API keys**  
7gNLk4bTdAnM5nMTBrMRF0Kvo (API key)  
eWeZpmndDkH2T9mEqSxYxGlpzk7qb22iFXAcEW1nlMv29biMT4d (API secret key)

**Access token & access token secret**  
73705532-7bfqUaHZNinLaVxxdfw04y8g7BuG5ZMGJD1z9HWw (Access token)  
OouvQVrAxdoN3gTbqUytKajUQiv4d2cuJE6k6gPqnAG0 (Access token secret)  
Read and write (Access level)

**Actions**  
Regenerate (for keys)  
Revoke (for tokens)  
Regenerate (for tokens)

Figure 4: Mendapatkan Keys and tokens Twitter

## Teknis dan Contoh

Pada bagian sebelumnya telah dibahas tentang cara mendapatkan API Basic Twitter, di mana kita sudah bisa mendapatkan:

1. API key;
2. API secret key;
3. Access token; dan
4. Access token secret.

Selain itu, kita juga sudah memahami bagaimana cara kerja APIs (lihat Gambar @ref(fig:skemaapi01)), sehingga bisa membantu kita (sebagai “programmer”) dalam membuat aplikasi atau sebagai analis untuk membuat sebuah analisis. Maka hal selanjutnya adalah mempraktikan pengetahuan tersebut untuk mendapatkan data dan mencapai target yang dituju. Mengingat data yang bisa dikumpulkan dengan menggunakan API Basic Twitter paling lama dari 10 hari terakhir, maka target yang mungkin adalah mengetahui twit dalam seminggu terakhir.

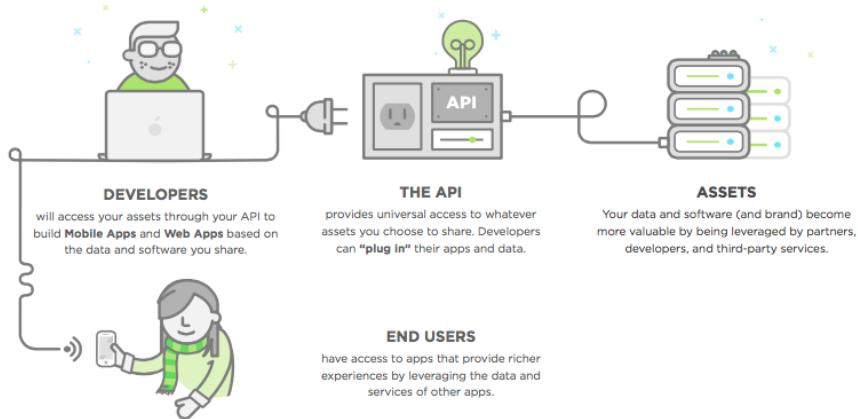


Figure 5: Cara kerja APIs

Untuk mendapatkan data dari Twitter dengan memanfaatkan API Basic di R, kita akan menggunakan paket `twitteR` [@twitteR]. Di mana melalui paket tersebut kita bisa dengan mudah mendapatkan data dari Twitter. Syaratnya, kita telah berhasil membuat API-nya. Jika belum silahkan ulangi proses pada bagian @ref(getdataapi) tentang membuat API. Terdapat beberapa hal yang perlu disiapkan untuk menggunakan dan mendapatkan data dari Twitter, diantaranya adalah:

1. Setting environment API;
2. Menentukan parameter;
3. Menentukan waktu data; dan
4. Jenis data yang diambil.

## Mendapatkan Data Twitter

Setelah berhasil membuat api **basic**, selanjutnya kita sudah bisa mengambil data dari Twitter secara langsung. Pada kesempatan ini kita akan menggunakan paket `rtweet` yang dibuat oleh Michael W. Kearney [-@rtweet-package]. Skrip berikut merupakan langkah pertamanya.

```
# loading the package
library(rtweet)
# create access between R and Twitter
create_token()
```

```

app = "myresearchapi",
consumer_key = "7gNLk4bTdAnM5nMTBrMRF0Kvo",
consumer_secret = "eWeZpmDkH2T9mEqSxYxGlpzk7qb22iFXAceW1nlMv29biMT4d",
access_token = "73705532-7bfqUaHizNinLaVxxdfw04y8g7BuG5ZMGJD1z9HWw",
access_secret = "OouvQVrAxdjoN3gTbqUytKajUQiv4d2cujE6k6gPqnAGO")

```

Skrip di atas digunakan untuk memanggil library `rtweet` dan mengatur hubungan dengan API Twitter. Hubungan dengan Twitter diatur dengan menggunakan aplikasi API yang dibuat serta beberapa nomor kuncinya. Nomor kunci tersebut hanya dibuat sekali, namun setiap kali pengambilan data harus dijalankan terlebih dahulu. Selanjutnya, setelah hal tersebut dilakukan kita dapat segera mengambil data dengan menggunakan beberapa fungsi dari `rtweet`.

### Mendapatkan twit berdasarkan kata kunci

Kata kunci yang dimaksud di sini bisa berupa kata, tagar, atau pun term lain yang terdapat dalam *body* twit (bagian teks dalam sebuah twit). Contohnya skrip berikut digunakan untuk mendapatkan twit yang bukan retweet yang mengandung tagar **tolak212** selama tujuh hari terakhir.

```

# trying to get 18000 tweets containing #Tolak212 hashtag
twitterdata1 <- search_tweets("#Tolak212", n = 18000, include_rts = FALSE)

# write_rds(twitterdata1, "Data/twitterData1.rds")

```

Fungsi pertama yang digunakan dalam skrip di atas adalah: `search_tweets` yang berguna untuk mencari twit. `n` menunjukkan jumlah twit yang mencoba didapatkan. Sementara `include_rts` merupakan fungsi logical (FALSE/TRUE) yang jika di atur FALSE tidak akan mengambil retweet dan sebaliknya. Di dalam pencarian twit dengan fungsi `search_tweets` juga terdapat beberapa parameter lain yang dapat diterapkan. Misalnya, `type` yang bisa di isi "recent", "mixed" dan "popular" yang menunjukkan jenis twit yang akan diambil bersifat yang terbaru, atau yang paling populer, atau gabungan.

Tantangan: setelah menjalankan skrip untuk mendapatkan data dari twitter, pada bagian environment akan menyimpan data yang bisa didapat. Coba eksplorasi data yang didapat dengan menggunakan pengetahuan yang didapat dari bagian @ref(fungsidasar), yaitu `str()`, `class()`, dan `summary()`.

### Mendapatkan timeline

Terkadang kita ingin mendapatkan *timeline* dari sebuah atau beberapa akun tertentu. Untuk mendapatkannya kita bisa menggunakan fungsi `get_timelines()` seperti dalam contoh skrip berikut.

```
tm1s <- get_timelines(c("cnn", "BBCWorld", "foxnews"), n = 3200)
```

Skrip di atas akan mencari twit yang dikirim oleh tiga akun (cnn, BBCWorld, foxnews) dalam 7 hari terakhir. Di sini kita atur untuk mencoba mendapatkan 3200 twit dari masing-masing akun. Hasil yang didapatkan kemudian bisa langsung di visualisasikan menggunakan skrip berikut. Tapi untuk pembahasan visualisasi lebih lengkapnya akan dibahas pada bagian @ref(vis).

```

tm1s %>%
  # dplyr::filter(created_at > "2017-10-29") %>%
  dplyr::group_by(screen_name) %>%
  ts_plot("days", trim = 1L) +
  ggplot2::geom_point() +
  ggplot2::theme_minimal() +
  ggplot2::theme(
    legend.title = ggplot2::element_blank(),
    legend.position = "bottom",
    plot.title = ggplot2::element_text(face = "bold"))

```

```

ggplot2::labs(
  x = NULL, y = NULL,
  title = "Frequency of Twitter statuses posted by news organization",
  subtitle = "Twitter status (tweet) counts aggregated by day from October/December 2018",
  caption = "\nSource: Data collected from Twitter's REST API via rtweet"
)

```

## Mendapatkan Data User

Selanjutnya, setelah kita berhasil mendapatkan data dengan dua metode sebelumnya. Kita juga mungkin ingin mendapatkan data tentang siapa sebenarnya user-user yang mengirim twit, misalnya tentang tagar tolak212. Untuk mendapatkan hal ini kita bisa menggunakan fungsi `lookup_user()` seperti contoh berikut.

```
userT212 <- lookup_users(twitterdata1$user_id)
```

Skrip di atas digunakan untuk mendapatkan data tentang semua akun yang menjadi pengirim. Di sini kita menggunakan kolom `user_id` dari data `twitterdata1` yang menghasilkan 442 obs (observasi), dan 88 kolom (variabel). Data ini selanjutnya dapat digunakan untuk berbagai hal. Misalnya untuk mengetahui jenis perangkat yang digunakan oleh pengguna Twitter, tempat, dan kita bisa menggunakannya untuk membuat klasifikasi sebuah akun bertindak seperti bot atau bukan.

## Keterbatasan

Di satu sisi menggunakan API BASIC sangat memudahkan kita mendapatkan data dari Twitter baik untuk penelitian maupun lainnya. Di sisi lain terdapat beberapa keterbatasan seperti telah dijelaskan sebelumnya. Selain itu, dengan menggunakan API BASIC kita juga memiliki keterbatasan dalam hal mendapatkan twit dalam jumlah tertentu lebih dari 18000. Jumlah tersebut merupakan jumlah maksimal yang bisa didapat dalam 15 menit pertama. Untuk mendapatkan lebih dari itu, kita bisa menambahkan parameter `retryonratelimit = TRUE`. Contohnya adalah sebagai berikut.

```

# how many total follows does cnn have?
cnn <- lookup_users("cnn")
# get them all (this would take a little over 5 days)
cnn_flw <- get_followers("cnn", n = cnn$followers_count, retryonratelimit = TRUE)
# get 130000 tweets containing Reuni212monas hashtag
twitterdata2 <- search_tweets("#Reuni212monas", n = 130000, include_rts = FALSE,
                                retryonratelimit = TRUE)

```

Pertama, skrip di atas mencari tentang jumlah akun yang mengikuti (follower) akun @CNN (pada saat skrip ini dibuat ada 40.862.969 followers). Skrip kedua kemudian digunakan untuk mendapatkan ID masing-masing pengikut akun @CNN. Berdasarkan keterangan dalam website resmi paket `rtweet` untuk mendapatkan seluruh data tentang follower akun @CNN tersebut setidaknya membutuhkan waktu 5 hari tanpa terputus. Sementara skrip yang terakhir mencoba mendapatkan 130 ribu tweet yang mengandung tagar Reuni212monas dalam waktu 7 hari terakhir.

Selain fungsi-fungsi di atas, `rtweet` juga memiliki beberapa fungsi lain yang bisa dilihat di sini<sup>1</sup>. Berikut adalah salah contoh fungsi lain yang dapat digunakan untuk mendapatkan teman dari sebuah akun.

Skrip di bawah ini menggunakan fungsi `get_friends()` dari `rtweet`. Coba jalankan dan lihat apa yang bisa didapat.

```
canggih_fds <- get_friends("canggihpw")
canggih_fds_data <- lookup_users(canggih_fds$user_id)
```

---

<sup>1</sup>Semua fungsi dari `rtweet`: <https://rtweet.info/reference/index.html>

# **Chapter 3**

## ***Akuisisi Data***

Dengan maraknya perdagangan online, banyak sekali pelanggan yang melakukan pencarian produk secara online. Sangat berbeda dengan pasar offline, melalui pasar online calon pembeli dapat membandingkan produk yang tersedia di tempat berbeda secara real time. Oleh karena itu, penetapan harga kompetitif adalah sesuatu yang telah menjadi bagian terpenting dari strategi bisnis.

Untuk menjaga harga produk tetap kompetitif dan menarik, Anda perlu melacak harga yang ditetapkan oleh pesaing Anda sehingga Anda dapat menyelaraskan strategi penetapan harga Anda. Oleh karena itu, pemantauan harga menjadi bagian hal penting dari menjalankan proses bisnis online. Anda mungkin bertanya tanya, bagaimana cara mendapatkan data dan membandingkan harganya.

Pada kondisi dilapangan, kita dapat mendapatkan data melalui dua cara, yakni, melalui API dan web scraping. Web scraping merupakan salah satu cara paling kuat dan andal untuk mendapatkan data dari internet. Hal ini paling dimungkinkan karena sangat dimungkinkan Anda tidak mempunyai akses terhadap API (pilihan pertama) dalam akuisisi data.

Dalam materi ini, kita akan belajar cara mengikis nama dan harga produk dari Amazon di semua kategori, di bawah merek tertentu.

### ***Konsep dan Teknis***

Scraping merupakan salah satu cara untuk mengumpulkan data dari sebuah laman diinternet. Sebagai gambaran, saat kita melihat sebuah laman yang terdapat beberapa informasi penting dalam jumlah yang sangat besar kita ingin mengkopip dan memasukan dalam file csv atau dalam jenis file lain. Akan tetapi, jika kita melakukan secara manual, kita akan kesulitan mengcopy file satu persatu. Konsep scraping sebenarnya juga kurang lebih seperti itu. Perbedaannya, kita tidak melakukan secara manual akan tetapi melakukan secara otomatis.

Pada saat kita mengkopip sebuah data yang berasal dari situs umumnya kita mengebloknya terlebih dahulu supaya semua data yang dibutuhkan terkopip. Namun dengan hanya mengkopip kode html kita akan mengetahui penyusun sebuah website. Namun, sebelum mengambil sebuah data yang terdapat pada sebuah website kita harus mengetahui susunan sebuah website, oleh sebab itu, secara tidak langsung kita juga dituntut untuk mengetahui bahasa lain, yaitu, html, css, bahkan kadangkala kita juga membutuhkan pengetahuan mengenai java.

### ***Kebutuhan untuk Scraping***

Secara umum, scraping hanya membutuhkan dua langkah, yaitu : (1) Mengambil halaman website, (2) Menyalin sebuah data spesifik. Oleh karena itu, tindakan scraping bisa diartikan sebuah tindakan **copy** dan **paste** yang sebagian besar dilakukan orang. namun dalam hal ini kita melakukannya secara otomatis.

Pada R Studio, kita membutuhkan beberapa langkah untuk melakukan scraping, berikut penjelasannya:

1. Mengunduh halaman html
2. Mengambil nodes html tertentu
3. Mengekstrak konten nodes html tertentu.

Dengan mengikuti tiga langkah di atas, kita dapat menyimpan data yang ada dalam sebuah website ke dalam format yang kita inginkan tanpa kemudian kita harus mengerjakannya secara manual. Untuk mengunduh 'html' kita dapat menggunakan fungsi 'read\_html()' yang ada dalam package 'readr'. Contohnya sebagai berikut:

```
# library(rvest)
# page <- read_html("teks url")
```

Setelah mendapatkan nodes dari sebuah laman. Bagian selanjutnya merupakan bagian yang paling rumit, dan kita membutuhkan latihan untuk menguasainya. Pada bagian ini kita akan belajar untuk membaca laman dan mendapatkan informasi yang dibutuhkan dengan nodes spesifik.

Untuk mengetahui nodes dalam sebuah laman, kita membutuhkan bantuan browser dan menggunakan alat bantu lain. Sekarang, coba buka browser anda dan ketik sebuah website favorite anda, setelah itu silahkan anda lakukan klik kanan dan pilih inspect elemen. Laman browser Anda akan terbagi menjadi dua bagian, bagian pertama merupakan bagian dari user interface dari website itu sendiri sedangkan bagian lainnya merupakan bagian dari nodes website. Selanjutnya anda dapat memilih nodes sesuai dengan kebutuhan anda.

Namun, sebelum ke tahap selanjutnya, terdapat alat yang dapat mempermudah kita untuk mengenai bagian-bagian dari sebuah laman website, yakni, selectorgadget yang bisa digunakan untuk menelusuri element website dalam keperluan pengambilan data. Namun, dalam beberapa kasus yang kompleks, kita harus menelusuri element website secara manual.

selectorgadget dapat diunduh dan diterapkan melalui Chrome melalui tautan berikut ini: <https://bitly/selgedget>. Untuk menggunakan, kita bisa memulai dengan membuka laman yang menjadi target. Lalu tekan klik pada logo selector gadget dan arahkan pada gambar terkait.

## ***Package yang dibutuhkan***

### ***Menggunakan paket Rvest***

Hadley wickham menulis paket **rvest** untuk melakukan web scraping dalam R. Paket ini berfungsi untuk mengekstrak informasi yang kita butuhkan dalam sebuah website. Selain paket di atas, kita juga harus menginstall paket **selectr** dan **xml2**.

```
{ r package1, echo=TRUE, warning=FALSE, message=FALSE, fig.align='center'} install.packages("selectr")
install.packages("xml2") install.packages("rvest")
```

Dalam paket **rvest** terdapat beberapa fungsi yang dapat membantu kita untuk mengekstrak informasi dari situs web.

- **read\_html()**: melakukan scraping terhadap konten HTML dari URL yang diberikan.
- **html\_nodes()**: berguna untuk melakukan identifikasi terhadap HTML.
- **'html\_nodes(".class")'** : berguna untuk memanggil nodes berdasarkan kelas CSS.
- **'html\_nodes("#id")'** : memanggil berdasarkan id
- **'html\_nodes(xpath = "xpath")'** : memanggil simpul berdasarkan xpath (kami akan membahasnya nanti).
- **'html\_attrs()'**: mengidentifikasi atribut (berguna untuk debugging).
- **'html\_table()'**: mengubah tabel HTML menjadi bingkai data.
- **'html\_text()'**: strip tag HTML dan mengekstrak teks.

### ***Menggunakan paket stringr***

Setelah mendapatkan data menggunakan R, kita dapat menggunakan **stringr** untuk melakukan pembersihan dan persiapan data. Terdapat empat fungsi mendasar yang penting dalam **stringr**.

- Dengan menggunakan **stringr**, memungkinkan Anda untuk bekerja di data bertipe vektor atau pun karakter.
- terdapat *whitespace tools* yang berguna untuk melakukan manipulasi data.
- Terdapat beberapa pola yang dapat digunakan untuk mengenali dekripsi pada teks, salah satunya adalah regular ekspresi.

```
library("stringr")
```

## **Menggunakan Jsonlite**

Dalam melakukan web scraping, JSON sangat bermanfaat dikarenakan JSON memang dibuat untuk dioptimalkan untuk website. Dengan menggunakan JASON, kita dapat melakukan konversi antara objek R dan JSON tanpa kehilangan jenis atau informasi.

```
library("jsonlite")
```

Sebelum kita mulai, mari kita berikan penjelasan cara kerjasanya.

Pada dasarnya setiap situs website berbeda, hal ini dikarenakan pengkodean yang digunakan pada setiap website juga berbeda. Scraping website merupakan teknik mengidentifikasi pola pada setiap website yang kemudian digunakan untuk mengekstrak data yang dibutuhkan. Scraping HTML hanyalah penguraian informasi yang tersedia dari HTML.

Scraping website memiliki serangkaian proses yang berfungsi sebagai berikut:

1. Mengakses halaman menggunakan R
2. Instruksi melalui R untuk melihat pada halaman yang akan diekstrak informasinya.
3. Konversi data dalam format yang dapat digunakan dalam R menggunakan **rvest**.

## **Web Scraping**

Dalam kondisi tertentu, pada toko penjualan daring, pelanggan membuat keputusan pembelian berdasarkan harga. Singkatnya, harga merupakan faktor pendorong calon konsumen untuk memutuskan proses transaksi dalam toko online. Oleh karena itu, banyak sekali calon konsumen yang membandingkan harga antara produk satu dengan produk lainnya. Hal ini dikarenakan calon konsumen sangat mudah membandingkan harga pada seluruh toko yang tersedia. Satu-satunya tantangan dalam hal ini adalah memperbarui data secara real time.

Cara ini tentunya tidak akan mudah dikarenakan dalam data web terdapat bahasa seperti HTML, XML, dan JSON untuk mendistribusikan konten dalam website. Dengan demikian, untuk mendapatkan data yang Anda butuhkan, anda harus memahami sedikit bahasa di atas dan pemahaman mendalam mengenai R sebelum Anda memulai.

### **Implementasi pada 1 laman**

Mari kita implementasikan sehingga kita bisa memahami bagaimana paket ini bekerja. Kita akan mengambil data Amazon untuk membandingkan produk *One Plus 6*.

1. Jalankan paket yang kita butuhkan, seperti **rvest**, **xml2**, dan **stringr**.

```
library(rvest)
library(xml2)
library(stringr)
```

2. Membaca HTML konten dari Amazon

```
Amazon <- read_html("https://www.amazon.in/OnePlus-Mirror-Black-64GB-Memory/dp/B0756Z43QS?tag=googinhydr-0")
```

3. Scrape produk lebih detil dari Amazon

Sekarang, kita akan mengekstrak beberapa informasi lain dari situs web Amazon, seperti judul, harga, deskripsi, peringkat, ukuran, dan warna.

```
knitr::include_graphics("Gambar/Scraping/Amazon1.png")
```

Selanjutnya, kita akan menggunakan HTML untuk mengekstraksi data dengan membuka elemen dari laman amazon.

Untuk mengakses kelas tag HTML, gunakan langkah-langkah berikut:

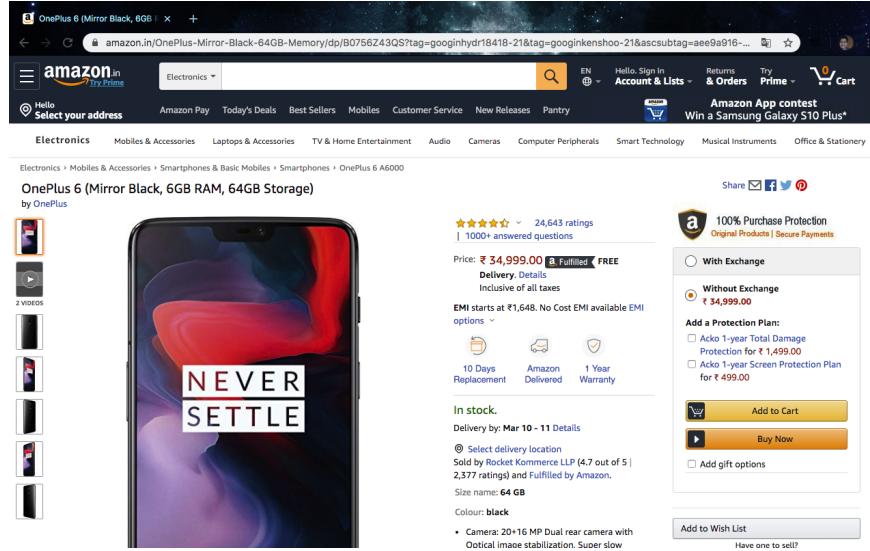


Figure 1: Amazon web page

- Silahkan laman browser anda, dan langsung menuju URL dari laman Amazon yang akan kita ekstrak informasinya.
- Selanjutnya, klik kanan, dan pilih *inspect element*.

*Langkah ini hanya berlaku jika Anda menggunakan browser Chrome*

```
knitr:::include_graphics("Gambar/Scraping/Amazon2.png")
```

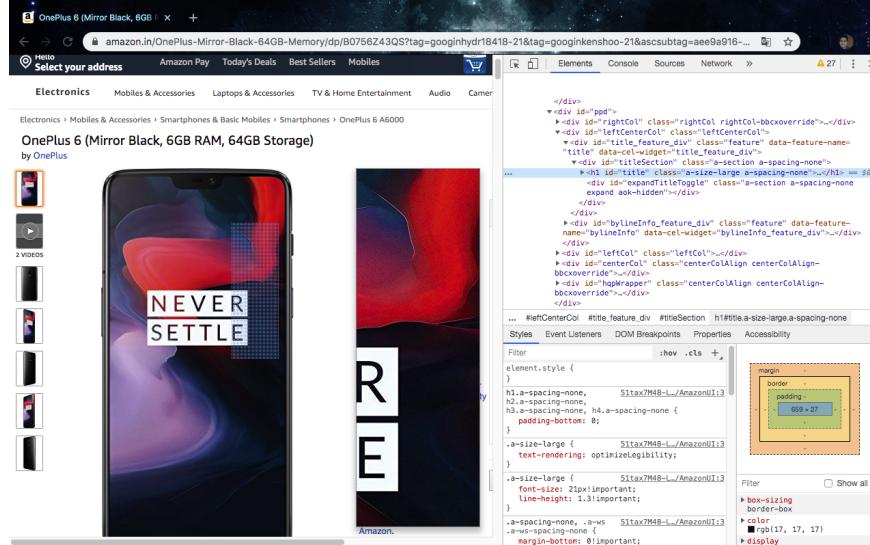


Figure 2: Amazon web page setelah inspect element

Gambar di atas merupakan kondisi ketika laman amazon telah kita inspect element. Pada tahap selanjutnya, kita akan mengekstrak informasi dengan bantuan `html_nodes()` pada halaman web yang menyimpan informasi konten. Kita juga dapat mendapatkan teks judul dengan menggunakan `html_text` dan mencetaknya dengan bantuan fungsi `head()`.



```
# Preprocessing
desc <- str_replace_all(desc, "[\r\n\t]", " ")
desc <- str_trim(desc)
head(desc)
```

```
## [1] "Size name:64 GB           |   Colour:black"
```

The OnePlus 6 comes with a 19:9 Full HD+ display.

#### *Rating Produk*

Fungsi di bawah ini bertujuan untuk mengambil nilai/rating dari laman Amazon.

```
# Selecting nodes
rate_html <- html_nodes(x = page, css = "span#acrPopover")
```

```
# Text extraction
rate <- html_text(rate_html)
head(rate)
```

```
## [1] "\n\n\n4.6 out of 5 stars\n\n\n"
## [2] "\n\n\n4.6 out of 5 stars\n\n\n"
```

```
# Remove Space, new lines, & tabs
rate <- str_replace_all(rate, "[\r\n]", "")
rate <- str_trim(rate)
head(rate)
```

```
## [1] "4.6 out of 5 stars" "4.6 out of 5 stars"
```

*Ukuran Produk* Sedangkan fungsi di bawah digunakan untuk mencari ukuran (size) dari produk, dalam konteks kasus yang akan kita coba adalah ukuran RAM.

```
# Selecting nodes
size_html <- html_nodes(x = page, css = "div#variation_size_name")
size_html <- html_nodes(size_html, "span.selection")
```

```
# Text extraction
size <- html_text(size_html)
head(size)
```

```
## [1] "64 GB"
```

#### *Warna Produk*

```
# Selecting nodes
color_html <- html_nodes(x = page, css = "div#variation_color_name")
color_html <- html_nodes(color_html, "span.selection")
```

```
# Text extraction
color <- html_text(color_html)
head(color)
```

```
## [1] "black"
```

- Setelah berhasil melakukan ekstraksi data dari semua bidang yang dapat digunakan untuk membandingkan informasi produk dari situs lain. Mari kita mengkompilasi dan menggabungkannya untuk menyusun kerangka datanya.

```
product_data <- data.frame(Title = lines, Description = desc, Rating = rate, Size = size, Color = color)
```

- Menyimpan data dalam format JSON, ketika data dikumpulkan, kita dapat membandingkan tugas yang berbeda seperti membandingkan, menganalisis dan lain sebagainya. Data akan disimpan lebih

lanjut dalam bentuk JSON supaya mempermudah untuk proses lebih lanjut.

```
library(jsonlite)

# Convert data
json_data <- toJSON(product_data)

# Print output
cat(json_data)

## [{"Title":"OePlus 6 (Mirror Black, 6GB RAM, 64GB Storage)","Description":"Size name:64 GB
```

Seperti yang Anda lihat, R dapat memberi Anda pengaruh besar dalam mengumpulkan data dari situs web yang berbeda. Dengan ilustrasi praktis tentang bagaimana R dapat digunakan, Anda sekarang dapat menjelajahinya sendiri dan mengekstrak data produk dari Amazon atau situs web e-commerce lainnya.

### ***Implementasi pada laman bercabang***

Seperti yang dijelaskan pada laman sebelumnya, bahwa ‘rvest’ memiliki beberapa kemampuan untuk membuat scraping menjadi lebih mudah. Beberapa fungsi dapat membantu kita untuk mengambil data dari sebuah website. Pada contoh sebelumnya, kita sudah mencoba menggunakan ‘rvest’ dan selectorGadget untuk scraping website secara sederhana. Pada implementasi selanjutnya, coba bayangkan jika website memiliki beberapa cabang laman.

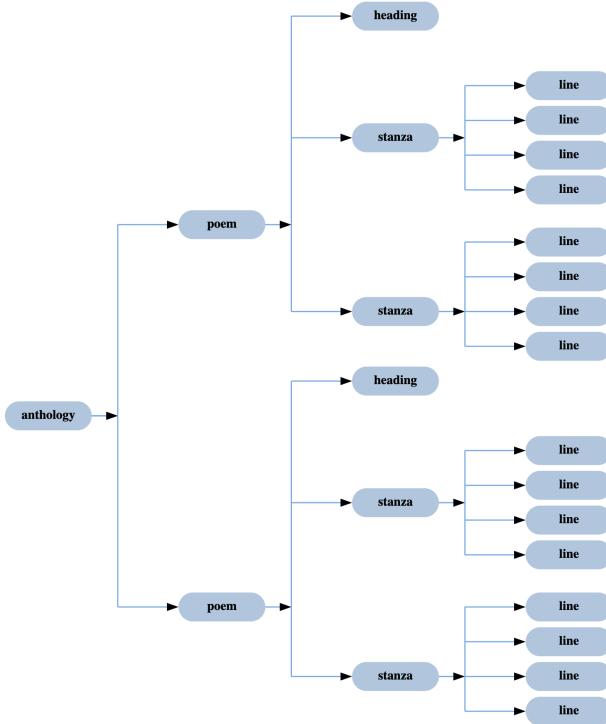


Figure 3: Analogi XML Website

Sehingga kadangkala kita membutuhkan data yang berasal dari sub sebuah laman, maka ketika kita ingin mengakses sebuah sub laman, maka nodes yang diperlukan adalah Laman 1 -> laman 2 -> laman 3 atau dalam xpath dalam dinyatakan sebagai laman 1/laman 2/laman 3. Sedangkan pada CSS dapat dinyatakan sebagai laman 1 > laman 2 > laman 3 atau bisa digantikan dengan skema 'laman 1=div[1]'. Jika diaplikasikan

dalam sebuah website dapat dilihat sebagai berikut (Teknis Xpath).

```
library(rvest)
library(tidyverse)

page <- read_html("https://setkab.go.id/category/transkrip-pidato/")

# tanggal
html_nodes(page, xpath = "//article/div[1]/div[1]") %>%
  html_text(trim = TRUE)

# Url's
html_nodes(page, xpath = "//article/div[1]/a") %>%
  html_attr("href")

# Judul
html_nodes(page, xpath = "//article/div[1]/a/h2") %>%
  html_text(trim = TRUE)

# Headline
html_nodes(page, xpath = "//article/div[1]/div[2]/p") %>%
  html_text(trim = TRUE)
```

Sedangkan untuk teknis CSS, mari kita coba lakukan inspect elemen dalam laman <https://setkab.go.id/category/transkrip-pidato/>, maka kita akan mengetahui bahwa teks tanggal berada pada kelas date. Artinya, karena dia merupakan kelas maka dibutuhkan tanda(.) diikuti kelasnya. Perhatikan skrip dibawah ini.

```
page <- read_html("https://setkab.go.id/category/transkrip-pidato/")

# tanggal
html_nodes(page, css = ".date") %>%
  html_text(trim = TRUE)

# Url's
html_nodes(page, css = ".text a") %>%
  html_attr("href")

# Judul
html_nodes(page, css = ".text a") %>%
  html_text(trim = TRUE)

# Headline
html_nodes(page, css = ".desc p") %>%
  html_text(trim = TRUE)
```

Pertanyaannya adalah, kapan kita akan menggunakan CSS atau Xpath, untuk menjawab pertanyaan tersebut kita tidak bisa memilih salah satu. Kadangkala kita dapat menggunakan salah satu dan kadangkala kita harus menggunakan keduanya. Hal ini dikarenakan, kadang menggunakan CSS lebih cepat dibandingkan Xpath kadang sebaliknya, oleh sebab itu, jika memungkinkan bahkan kita merancang terlebih dahulu.

## Latihan Scraping Website

Untuk latihan saat ini, kita akan menggunakan website <https://setkab.go.id/category/transkrip-pidato/> sebagai contoh. Laman tersebut dikategorikan laman statis dan dibuat menggunakan html. Namun, website tersebut terdiri dari laman 1 hingga 115. Untuk laman dinamis, atau laman yang lebih kompleks dan menggunakan html, database, dan atau java. 'rvest' sebenarnya memiliki kemampuan untuk melakukannya,

akan tetapi cukup rumit sehingga harus menggunakan pekat lain, yakni, ‘Rselenium’. Perhatikan skrip di bawah ini:

```
library(rvest)
library(tidyverse)

page <- read_html("https://setkab.go.id/category/transkrip-pidato/")

scrap_pidato <- function(page) {
  hasil_scrap <- data_frame(
    tanggal = html_nodes(page, css = ".date") %>%
      html_text(trim = TRUE),
   Urls = html_nodes(page, css = ".text a") %>%
      html_attr("href"),
    Judul = html_nodes(page, css = ".text a") %>%
      html_text(trim = TRUE),
    Headline = html_nodes(page, css = ".desc p") %>%
      html_text(trim = TRUE)
  )
}

data_hasil <- scrap_pidato(page = page)
```

Namun, skrip di atas hanya mengambil data dari satu laman sedangkan alaman terdapat 115 buah. Dengan skema tersebut, kita harus membuat fungsi paste dengan membuat nomor 1-115.

```
url_dasar <- "https://setkab.go.id/category/transkrip-pidato/page/"

no_url <- paste0(seq_along(1:115), "/")

data_halaman <- data_frame(URLs = paste0(url_dasar, no_url))
data_halaman <- data_halaman[1:5,]
```

Menggunakan skrip di atas kita sudah memiliki 115 url yang berbeda pada laman website yang mencatut transkrip pidato presiden. Dalam setiap laman, kurang lebih kita akan mendapatkan kurang lebih 10 pidato, sehingga kurang lebih akan mendapatkan 1150 pidato. Untuk mengeksekusinya dalam R kita bisa menggunakan fungsi for loop atau lapply untuk melakukan iterasi.

```
datalist = list()

for(i in seq_along(data_halaman$URLs)) {
  page <- read_html(paste0(data_halaman$URLs[i]))
  data_hasil <- scrap_pidato(page = page)
  datalist[[i]] <- data_hasil
}
```

Pekerjaan yang dilakukan skrip di atas adalah membuat url dari data secara otomatis dari 1 hingga 115 yang kemudian digunakan untuk mengambil data secara otomatis.

## Chapter 4

### ***Mencari Ide dalam Data Text***

Teks mining atau yang dikenal juga sebagai analisis teks merupakan sebuah proses mengubah data teks yang tidak terstruktur menjadi informasi yang bermakna dan dapat ditindaklanjuti. Penambangan teks sangat berguna bagi perusahaan untuk menambah wawasan terkait perusahaan serta memungkinkan perusahaan untuk membuat keputusan berbasis data. Dalam bisnis, data text dihasilkan setiap harinya yang dapat membantu perusahaan mendapatkan wawasan cerdas tentang pendapat orang atas suatu produk atau pun layanan. Sekarang, coba bayangkan semua ide yang potensial yang dapat Anda dapatkan ketika menganalisis email, ulasan produk, postingan media sosial, masukan pelanggan, dan lain sebagainya. Di sisi lain, ada dilema bagaimana memproses semua data ini. Disitulah, analisis text memainkan peran utamanya.

#### ***Apa itu Text Mining***

Text mining merupakan proses otomatis yang menggunakan pemrosesan bahasa alami untuk mengekstrak wawasan berharga dari teks yang tidak terstruktur. Dengan mengubah data menjadi informasi yang dapat dipahami mesin, text mining mengotomasi proses pengelompokan teks dan topik yang dimaksud.

#### ***Perbedaan Antara Text Mining, Text Analisis, dan Text Analytics?***

Text mining atau pun text analysis sering dianggap sebagai sinonim atau hal yang dianggap sama. Namun konsep text analytics mempunyai pendekatan yang berbeda dengan 2 konsep lainnya. Singkatnya:

- text mining dan text analysis mempunyai fokus pada analisis data teks mentah dengan menggunakan metode tertentu sehingga menghasilkan temuan kualitatif.
- Sedangkan text analytics mempunyai fokus pada tren atau pola yang ada diseluruh rangkaian kata yang banyak dan menghasilkan kuantitatif.

Text mining menggabungkan gagasan statistik, linguistik, dan pembelajaran mesin untuk membuat model belajar dari pembelajaran sebelumnya. Anda dapat memilih pendekatan sesuai data yang tersedia serta kebutuhan analisis. Dalam kebanyakan kasu, kedua pendekatan ini digabungkan untuk hasil yang lebih menarik.

#### ***Metode dan Teknik***

Pada dasarnya, terdapat beberapa metode dan teknik yang dapat digunakan dalam analisis teks, namun, pada tahapan awal ini kita akan mempelajari beberapa metode yang sering digunakan untuk mengekplorasi ide dalam analisis teks mining, berikut penjelasannya:

- *Word frequency*, frekuensi kata dapat digunakan untuk mengidentifikasi istilah atau konsep yang paling sering digunakan dalam satu set data. Menemukan kata-kata yang paling banyak disebutkan dalam teks yang tidak terstruktur sangat berguna saat melakukan analisis terhadap pelanggan, percakapan media sosial, atau pun respon pelanggan. Sebagai contoh, jika kata mahal yang berlebihan muncul pada ulasan pelanggan Anda, mungkin, Anda perlu melakukan penyesuaian terhadap harga produk Anda di pasar.
- *Kolokasi*, kolokasi mengacu pada urutan kata yang biasanya munculnya berdekatan. Jenis kolokasi paling umum yang kita gunakan adalah bigram (kombinasi dua kata) dan trigram (kombinasi tiga kata). Mengidentifikasi kolokasi dan menghitungnya sebagai satu kata untuk meningkatkan granulitas teks, memungkinkan pemahaman yang lebih baik terkait konteks pembicaraan sehingga dapat mengarahkan analisis teks lebih akurat.

Pada chapter kali ini, kita akan mempelajari dua metode di atas untuk mencari ide dari analisis teks. Namun sebelumnya, pada tahapan lebih lanjut, kita akan belajar sedikit mengenai regular ekspresion yang sangat penting dalam melakukan analisis teks.

## *Pre Processing dengan regex*

Regular ekspresi atau yang biasa disebut regex merupakan alat ringkas dan fleksibel untuk menggambarkan pola dalam dalam analisis teks. Regular ekspresi merupakan pattern engine yang terdapat pada stringr. Dengan demikian, kita anda menggunakan fungsi pencocokan pada stringr artinya Anda membungkusnya dalam panggilan regex:

- Basic Regex

```
library(stringr)
x <- c("apple", "banana", "pear")
str_extract(x, "an")

## [1] NA    "an" NA
# [1] NA    "an" NA
```

Namun, dalam beberapa kondisi anda dapat menggunakan `ignore_case` TRUE dikarenakan perbedaan huruf besar kecil dalam teks.

```
bananas <- c("banana", "Bananas", "BANANA")
str_detect(bananas, "banana")

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

str_detect(bananas, regex("banana", ignore_case = TRUE))

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

Skrip di atas akan sangat bermanfaat untuk melakukan deteksi pada data teks yang tidak terstruktur dan melakukan pre prosesing di dalamnya. Selanjutnya, kita akan belajar salah satu fungsi dari regex, yakni, `gsub()` yang sangat bermanfaat dalam melakukan pre prosesing text. Pada latihan kali ini, kita akan menggunakan data twitter terkait penerbangan di Amerika Serikat dan langsung memprosesnya menjadi word cloud.

```
library(tidyverse)
library(dplyr)
library(stringr)
library(textclean)

# Import data to environment
data_airlines <- read.csv2("Tweets.csv", sep = ", ")

# Select column before tokenization
data_airlines <- data_airlines %>%
  select(text)
# Cleaning text
data_airlines$text <- replace_html(data_airlines$text)
data_airlines$text <- replace_hash(data_airlines$text)
data_airlines$text <- str_replace_all(data_airlines$text, "@[[alnum:]_]*", "")
# Remove sign
data_airlines$text <- gsub("[[:punct:]]+[:blank:]", " ", data_airlines$text)
# to lower text
data_airlines$text <- tolower(data_airlines$text)
```

```

# 1 what said
# 2 plus you ve added commercials to the experience tacky
# 3 i didn t today must mean i need to take another trip
# 4 it s really aggressive to blast obnoxious entertainment in your guests faces they have little recou
# 5 and it s a really big bad thing about it

```

Pada awalnya, data airlines yang tersedia sangat kotor, artinya, terdapat berbagai macam kode, simbol dari emot icon, dan link html yang menjadi bagian dari data text di atas. Oleh sebab itu, kita harus menghilangkannya dalam proses data menggunakan fungsi di atas. Pada tahap selanjutnya, kita harus menghilangkan stopwords yang terdapat dalam teks.

```

library(tm)
library(wordcloud)

Text <- data_airlines %>%
  rename("Text" = 'text' ) %>%
  select("Text")

# vector
Text <- Corpus(VectorSource(Text))

# continue cleaning the text
Text <- tm_map(Text, removeWords, stopwords("english"))

# Remove punctuations
Text <- tm_map(Text, removePunctuation)

# Eliminate extra white spaces
Text <- tm_map(Text, stripWhitespace)

dtm <- TermDocumentMatrix(Text)
m <- as.matrix(dtm)
v <- sort(rowSums(m),decreasing=TRUE)
d <- data.frame(word = names(v),freq=v)
head(d, 10)

##          word freq
## flight      flight 3923
## can         can 1650
## get         get 1340
## http        http 1136
## thanks      thanks 1077
## cancelled   cancelled 1056
## now         now 1037
## just        just  972
## service     service  965
## help        help  852

```

### \* Visualisasi Menggunakan ggplot2/wordcloud\*

Selanjutnya, kita akan memvisualisasikannya menggunakan ggplot untuk mengetahui frekuensi kata yang paling sering muncul dalam data airlines.

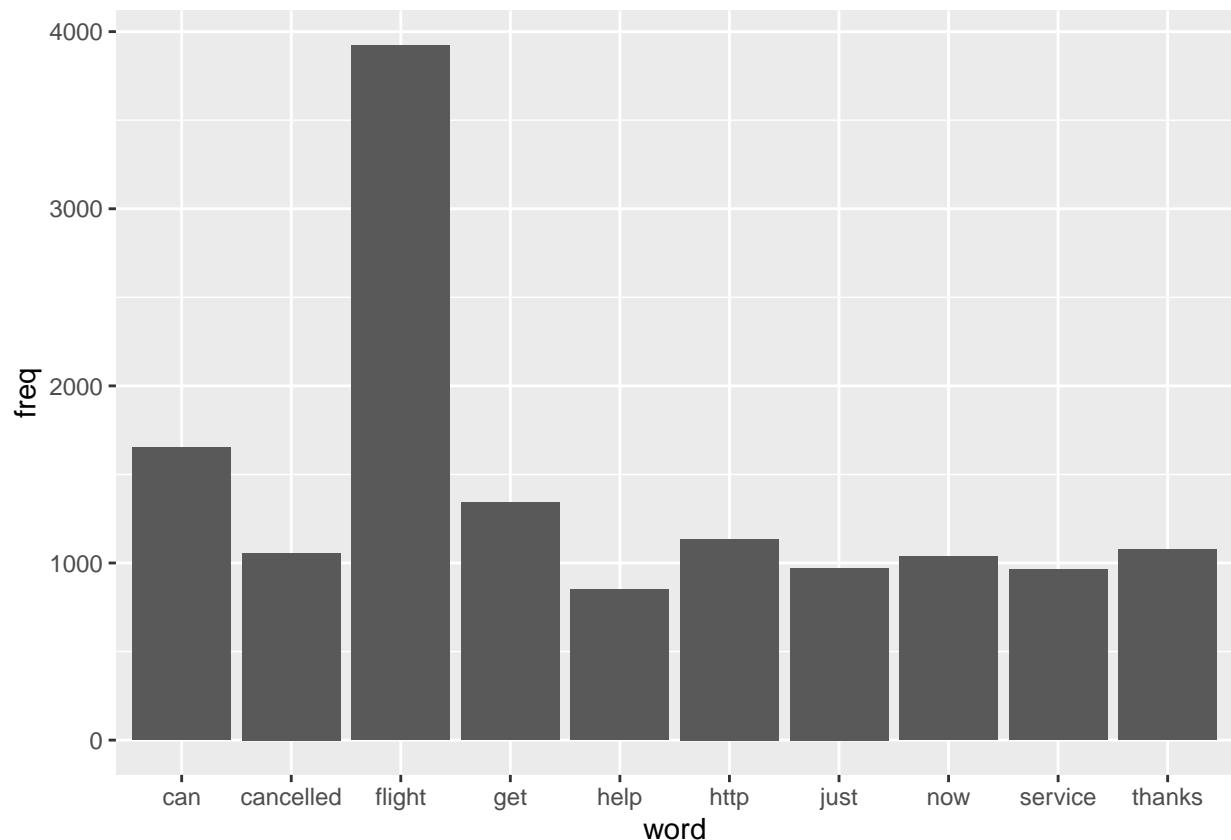
```
library(ggplot2)
```

```

# select top 10
d_10 <- d %>%
  head(10)

# Visualization with ggplot
ggplot(data = d_10, mapping = aes(x = word, y = freq)) +
  geom_bar(stat = "identity")

```



Dari visualisasi di atas, kita dapat menerka apa yang menjadi percakapan dalam ribuan teks/data yang kita analisi. Sebagai contoh, dari visualisasi di atas percakapan umumnya menyebut soal servis, pembatalan, dan pertolongan dalam penerbangan. Artinya, percakapan di atas umumnya menyinggung soal servis pesawat terbang yang kaitannya dengan pembatalan pesawat. Selanjutnya, kita juga dapat memvisualisasikannya dengan menggunakan wordcloud agar mendapatkan visualisasi yang lebih bagus.

```

wordcloud(words = d$word, freq = d$freq, min.freq = 1,
          max.words=200, random.order=FALSE, rot.per=0.34,
          colors=brewer.pal(8, "Dark2"))

```



Dengan menggunakan fungsi di atas, kita dapat memvisualisasikan seluruh kata-kata yang terdapat pada data kita. Kata dengan frekuensi terbanyak akan divisualisasikan dengan warna dan ukuran yang menonjol/berbeda. Sedangkan frekuensi yang lebih rendah akan divisualisasikan dengan ukuran lebih kecil dan warna yang tidak cukup menonjol.

## *Visualisasi Menggunakan Bigram untuk Mencari Konteks*

Setelah kita memvisualisasikan kata yang sering muncul menggunakan bar chart atau pun wordcloud, Anda hanya akan sebatas tau kata yang paling sering digunakan tanpa mengetahui konteks isu yang berkembang. Sebagai contoh, dari visualisasi di atas, penerbangan/flight menjadi kata dengan frekuensi paling banyak disebutkan akan tetapi kita tidak mengetahui konteks apa yang kemudian berhubungan atau berkorelasi dengan kata penerbangan/flight di atas. Oleh sebab itu, kita membutuhkan visualisasi menggunakan bigram.

### *Tokenization dengan ngrams*

Pada tahap pertama, kita harus melakukan tokenisasi menggunakan fungsi `unnest_token`. Tokenisasi bertujuan untuk memberikan token pada setiap kata yang menyusun sebuah kalimat.

```
library(dplyr)
library(tidytext)
library(tidyverse)
library(tidyr)

Semantic <- data_airlines %>%
  select(`text`)

# Tokenisasi by ngrams
Semantic <- Semantic %>%
  unnest_tokens(bigram, `text`, token = "ngrams", n = 2)

# Counting and filtering n-grams
Semantic %>%
```

```

count(bigram, sort = TRUE)

## # A tibble: 103,372 x 2
##   bigram          n
##   <chr>       <int>
## 1 t co        1211
## 2 http t      1155
## 3 i m         713
## 4 thank you    567
## 5 customer service 562
## 6 on the       548
## 7 can t        540
## 8 on hold      517
## 9 cancelled flightled 506
## 10 it s        506
## # ... with 103,362 more rows

# t co 1211
# http t 1155
# i m 713
# thank you 567
# customer service 562
# on the 548
# can t 540
# on hold 517
# cancelled flightled 508
# it s 506

```

Pada tahap selanjutnya, kita akan membagi/memisah kata yang telah anda tokenisasi menjadi dua kolom berbeda menggunakan fungsi **separate**.

```

Semantic_separated <- Semantic %>%
  separate(bigram, c("word1", "word2"), sep = " ")

```

Coba Anda perhatikan hasil dari pemisahan data hasil tokenisasi, kita masih dapat menemukan beberapa kata hubung yang masih terdapat pada hasil tokenisasi. Kata hubung yang masih terdapat dalam hasil tokenisasi harus kita hilangkan karena akan mengaburkan pemaknaan konteks. Dengan demikian, tahap selanjutnya kita akan menggunakan kamus **stop\_words** untuk menghilangkan kata hubung pada data.

```

Semantic_filtered <- Semantic_separated %>%
  filter(!word1 %in% stop_words$word) %>%
  filter(!word2 %in% stop_words$word)

```

Setelah menghapus stopwords/kata hubung pada data, kita harus menghitung frekuensi kata yang saling berhubungan untuk melihat pusat dari konteks pembicaraan dan menggabungkannya kembali dalam satu kolom.

```

# new bigram count
Semantic_filtered_count <- Semantic_filtered %>%
  count(word1, word2, sort = TRUE)

Semantic_united <- Semantic_filtered %>%
  unite(bigram, word1, word2, sep = " ")

```

Terakhir, kita akan melakukan visualisasi menggunakan igraph yang akan membantuk kita untuk melihat konteks dari pembicaraan yang dilakukan/dari data teks yang kita analisis.

```

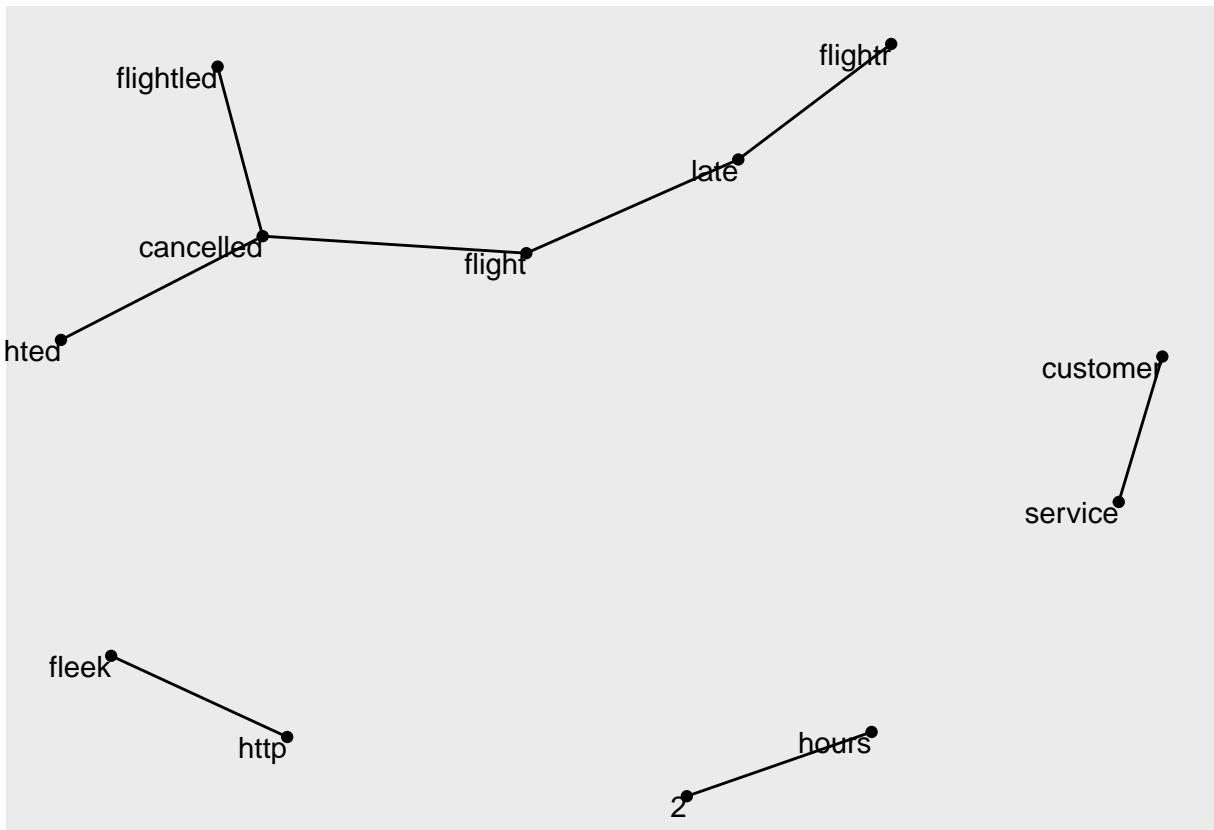
library(igraph)

Semantic_graph <- Semantic_filtered_count %>%
  filter(n > 100) %>%
  graph_from_data_frame()

library(ggraph)
set.seed(2019)

ggraph(Semantic_graph, layout = "fr") +
  geom_edge_link() +
  geom_node_point() +
  geom_node_text(aes(label = name), vjust = 1, hjust = 1)

```



## Chapter 5

### *Sentiment Analysis dengan tidy data*

Pada bab sebelumnya, kita telah mengekplor apa itu text dengan *tidy format* yang digunakan untuk mencari frekuensi kata dalam sebuah dokumen. Hal ini juga berguna, untuk melihat kata-kata yang sering digunakan dalam sebuah dokumen atau membandingkan antar dokumen, tapi mari kita coba untuk hal yang berbeda. Marri kita coba untuk melakukan analisis terhadap pendapat atau sentimen. Ketika seoarang manusia membaca sebuah dokumen, ia akan melakukan pemahaman dan mengekstrak informasi dari dokumen tersebut yang kemudian berimplikasi terhadap emosional pembaca. Perhatikan gambar di bawah ini.

```
library(knitr)
knitr::include_graphics("Gambar_1.png")
```

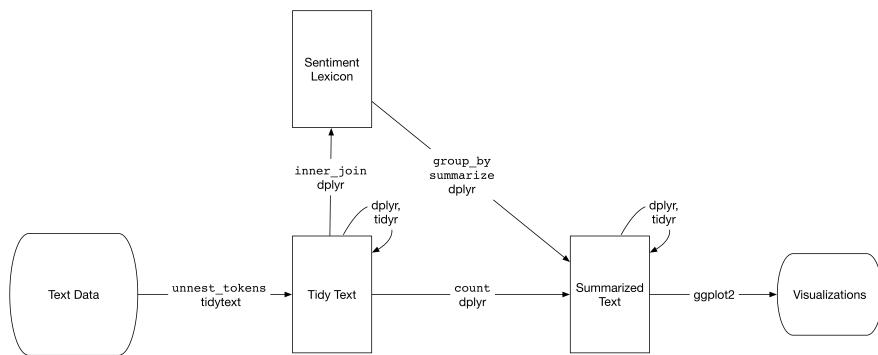


Figure 1: Flowchart text analysist menggunakan tidytext

Salah satu cara untuk menganalisis sentimen dalam sebuah dokumen adalah dengan memposisikan sebuah teks merupakan kombinasi dari beberapa kata dan sentimen merupakan jumlah dari nilai masing masing kata. Faktanya, ini bukan satu-satunya cara untuk melakukan teks analisis, namun, metode ini merupakan pendekatan yang paling sering digunakan.

### *Sentiment Dataset*

Seperti yang telah disinggung di atas, ada berbagai metode dan kamus yang dapat mengevaluasi pendapat atau emosi dalam teks. Paket `tidytext` berisi beberapa leksikon sentimen. Tiga leksikon tersebut adalah:

1. AFINN dari Finn Arup Nielsen
2. Bing dari Bing Lie Kolaborator
3. NRC dari Saif Mohammad dan Peter Turney

Ketiga leksikon ini berdasarkan pada unigram atau kata tunggal. Kamus ini mengandung banyak kata-kata berbahasa inggris yang kemudian kata-kata tersebut diberikan nilai positif/negatif, dan juga emosi seperti kegembiraan, kemarahan, kesedihan, dan sebagainya. Leksikon NRC mengkategorikan kata-kata dalam mode biner (positif/negatif). Kamus Bing juga mengategorikan kata-kata dalam dua kelompok yang berbeda, yakni, positif dan negatif. Sedangkan untuk AFINN, kamus ini memberikan nilai berkisar antara -5 dan 5, dengan skor negatif menunjukkan sentimen negatif dan skor positif menunjukkan sentimen positif. Semua informasi ini ditabulasikan dalam dataset `sentiment` dan dapat kita akses dengan fungsi `get_sentiment()`.

Mungkin beberapa dari pembaca menanyakan, bagaimana leksikon sentimen di atas di satukan dan di validasi?, pada kenyataannya, leksikon ini disusun melalui **crowdsourcing** atau tenaga penulis yang kemudian divalidasi melalui beberapa kombinasi **crowdsourcing**. Khusus pada kasus Bahasa Indonesia, kita belum memiliki kamus sentimen yang memadai sehingga kita harus mengembangkannya secara mandiri atau kemudian melakukan **crowdsourcing**.

### *Sentiment Analysis dengan Inner Join*

Dengan menggunakan format data yang rapi, kita akan lebih mudah melakukan analisis terhadap teks. Umumnya, kita harus menghilangkan kata-kata hubung sebelum melakukan analisis sentimen dalam sebuah dokumen.

Mari kita praktikan, pertama, kita perlu mengambil teks dari sebuah novel dan mengkonversikannya menjadi format yang lebih rapi menggunakan `unnest_token()`, selain itu, mari kita menyiapkan kolom yang berfungsi untuk mengidentifikasi asal baris dan buku menggunakan fungsi `group_by()`.

```
library(tidytext)
library(janeaustenr)
library(dplyr)
library(stringr)

tidy_books <- austen_books() %>%
  group_by() %>%
  mutate(linenumber = row_number(),
        chapter = cumsum(str_detect(text, regex("^chapter [\\divxlc]",
                                                ignore_case = TRUE)))) %>%
  ungroup() %>%
  unnest_tokens(word, text)

head(get_sentiments("bing"), 15)

## # A tibble: 15 x 2
##   word      sentiment
##   <chr>     <chr>
## 1 2-faces   negative
## 2 abnormal   negative
## 3 abolish    negative
## 4 abominable negative
## 5 abominably negative
## 6 abominate   negative
## 7 abomination negative
## 8 abort      negative
## 9 aborted    negative
## 10 aborts    negative
## 11 abound     positive
## 12 abounds    positive
## 13 abrade     negative
## 14 abrasive   negative
## 15 abrupt     negative

# 2-faces negative
# abnormal negative
# abolish negative
# abominable negative
# abominably negative
# abominate negative
# abomination negative
# abort negative
# aborted negative
# aborts negative
```

Perhatikan bahwa kita menggunakan kata `word` sebagai nama kolom dari output `unnest_tokens()`. Hal

ini dikarenakan, leksikon sentiment dan himpunan kata juga memiliki kolom dengan kata yang sama, yakni *word*. Dengan demikian, kita lebih mudah ketika melakukan `inner_join()` dan `anti_join()`.

Setelah teks dalam format yang rapi, kita siap melakukan analisis sentiment. Pada analisis ini kita akan menggunakan bing lexicon dan fungsi `filter` untuk menggabungkan kata.

```
positive_sentiment <- get_sentiments("bing")  
  
positive_sentiment <- positive_sentiment %>%  
  filter(sentiment == "positive")
```

Selanjutnya, kita melakukan penyaringan terhadap data buku, kita hanya menggunakan buku berjudul Emma dalam analisis sentiment kali ini. Pada tahap ini, kita akan menyaring buku berjudul Emma dan memasukannya ke dalam data `positive_sentiment`.

```
head(tidy_books %>%  
  filter(book == "Emma") %>%  
  semi_join(positive_sentiment) %>%  
  count(word, sort = TRUE, 10))
```

```
## # A tibble: 6 x 3  
##   word     `10`     n  
##   <chr>    <dbl> <int>  
## 1 well      10     401  
## 2 good      10     359  
## 3 great     10     264  
## 4 like      10     200  
## 5 better     10     173  
## 6 enough     10     129  
  
# well 10 401  
# good 10 359  
# great 10 264  
# like 10 200  
# better 10 173  
# enough 10 129
```

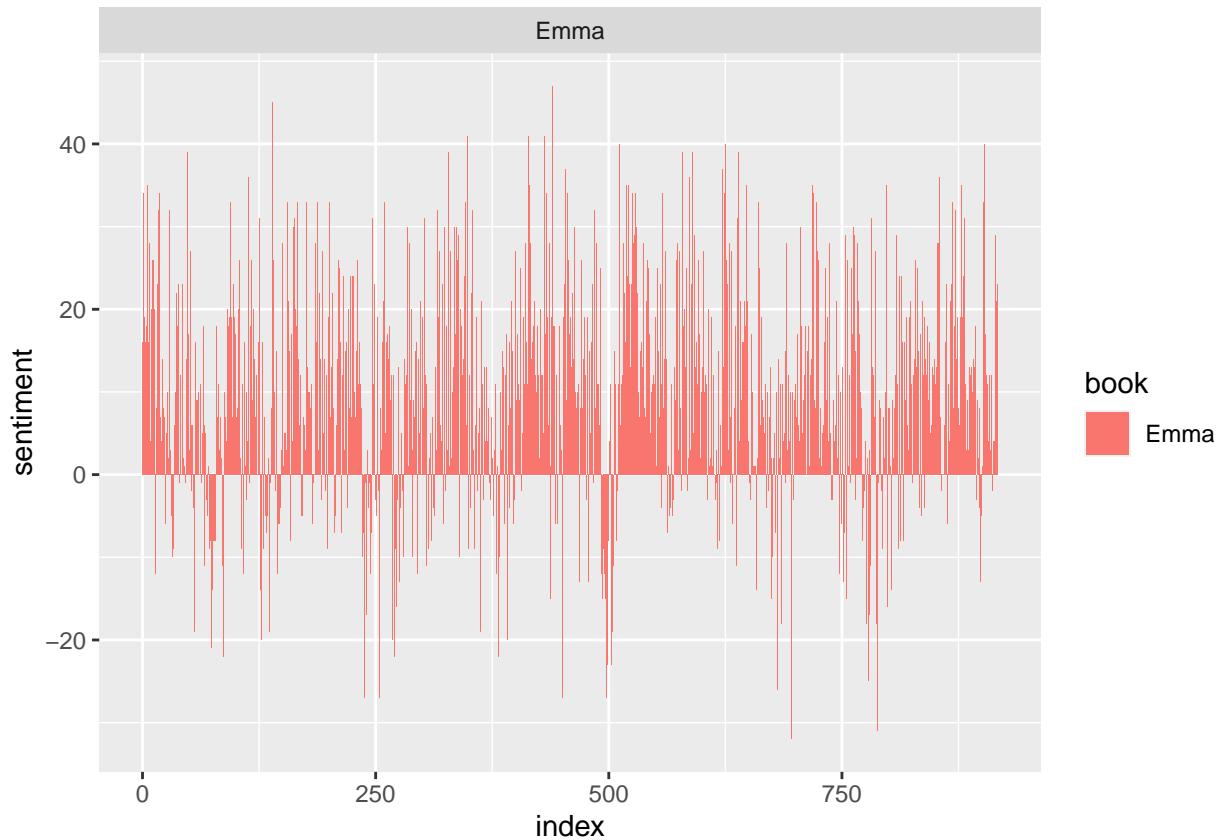
Pada tahap selanjutnya, kita akan membuat kelompok yang terdiri dari 80 kata gabungan dari leksikon bing dan data sentiment. Dari proses di bawah ini kita akan mengetahui sentiment dari setiap index dari buku Emma.

```
library(tidyr)  
  
bing <- get_sentiments("bing") # memanggil leksikon bing  
  
emma_sentiment <- tidy_books %>%  
  inner_join(bing)  
  
emma_sentiment <- emma_sentiment %>%  
  count(book = "Emma", index = linenumbers %/% 80, sentiment) %>%  
  spread(sentiment, n, fill = 0) %>% # memisahkan positif dan negatif sentiment  
  mutate(sentiment = positive - negative) # mengkalkulasi total sentiment
```

Pada tahap selanjutnya, kita akan memvisualisasikan data dari analisis sentiment di atas. Kita akan membagi berdasarkan indeks dari buku Emma.

```
library(ggplot2)  
ggplot(emma_sentiment, aes(index, sentiment, fill = book)) +
```

```
geom_bar(stat = "identity", show.legend = TRUE) +
facet_wrap(~book, ncol = 2, scales = "free_x")
```



Gambar di atas menunjukkan perbandingan sentimen positif dan negatif. Grafik ke atas menunjukkan sentimen positif sedangkan grafik kebawah menunjukkan sentimen negatif. Akan tetapi, dengan pewarnaan di atas, grafik kurang menarik. Hal ini dikarenakan grafik menggunakan tone warna yang sama. Mari kita coba untuk membedakan warna antara sentimen negatif dan positif.

Sekarang kita coba memetakan komen positif/negatif yang terdapat pada buku Emma dan memvisualisasikannya. Langkah pertama, kita harus membedakan dan menghitung jumlah sentimen negatif dan positif pada buku austin

```
counting_words <- tidy_books %>%
  inner_join(bing) %>%
  count(word, sentiment, sort = TRUE)

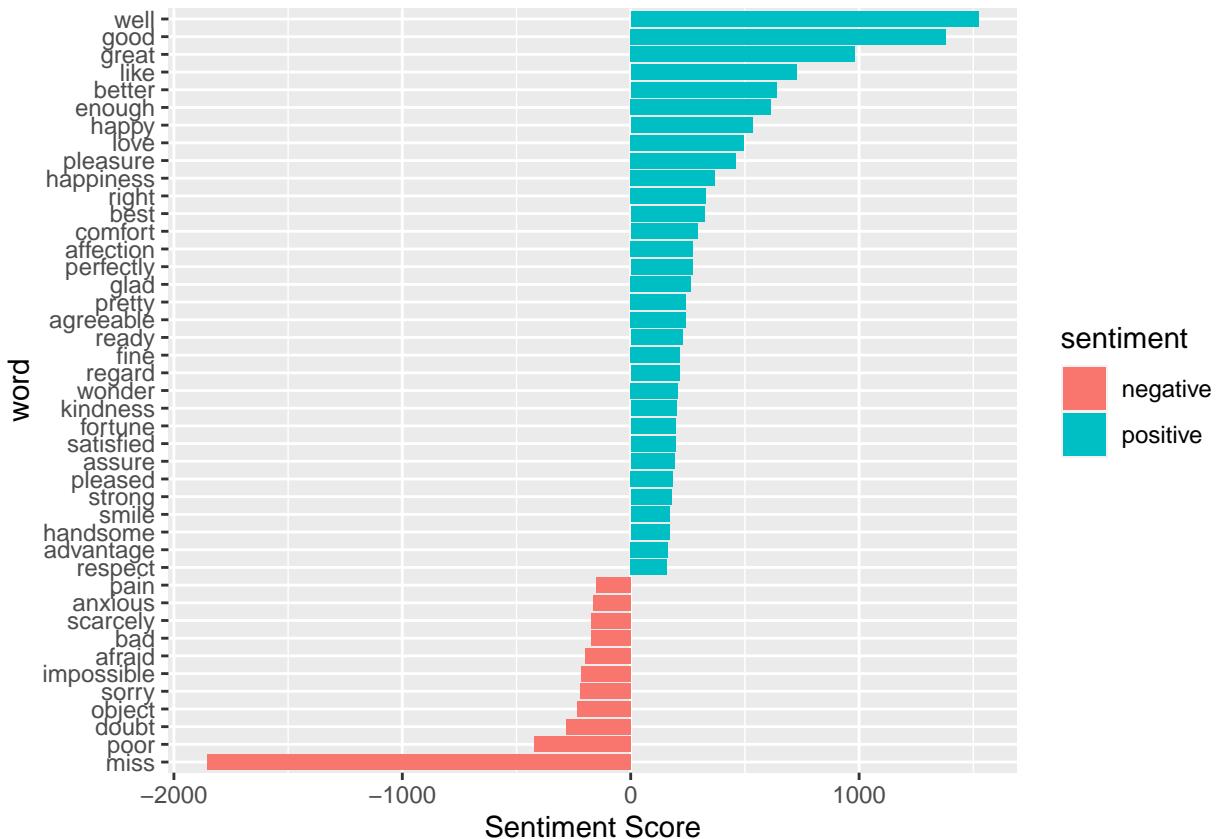
head(counting_words) # 6 nomiasi teratas
```

```
## # A tibble: 6 x 3
##   word    sentiment     n
##   <chr>    <chr>     <int>
## 1 miss    negative  1855
## 2 well    positive   1523
## 3 good    positive   1380
## 4 great   positive   981
## 5 like    positive   725
## 6 better   positive   639
```

```
# miss negative      1855
# well  positive    1523
# good  positive    1380
# great positive    981
# like   positive    725
# better  positive    639
```

Pada tahap selanjutnya, kita akan mencoba melakukan visualisasi menggunakan ‘ggplot’ dengan membandingkan antara sentimen negatif dan positif. Dengan catatan, sentimen negatif dan positif menggunakan warna yang berbeda.

```
counting_words %>%
  filter(n > 150) %>%
  mutate(n = ifelse(sentiment == "negative", -n, n)) %>%
  mutate(word = reorder(word, n)) %>%
  ggplot(aes(word, n, fill = sentiment)) +
  geom_col() +
  coord_flip() +
  labs(y = "Sentiment Score")
```



Dari visualisasi bar chart di atas, kita dapat mengetahui frekuensi kata dari mengandung sentimen positif data data buku austin yang coba kita analisis di atas.

Selain menggunakan visualisasi bar di atas, kita juga dapat memvisualisasikan dalam bentuk lain, seperti, sentimen dalam bentuk *WordCloud*. Mari kita siapkan beberapa library seperti ‘reshape’ dan ‘wordcloud’.

```
library(reshape2)
library(wordcloud)
```

```

tidy_books %>%
  inner_join(bing) %>%
  count(word, sentiment, sort = TRUE) %>%
  acast(word~sentiment, value.var = "n", fill = 0) %>%
  comparison.cloud(colors = c("red", "dark green"),
                    max.words = 100)

```

# negative



# positive

Dari gambar di atas kita dapat mengetahui, kumpulan kata yang sering digunakan dalam tiap-tiap kelompok sentimen. Cara membacanya, semakin besar sebuah kata maka kata tersebut semakin sering digunakan dalam kelompok sentimen.

## Chapter 6

### **Topic Modeling dengan Latent Dirichlet Allocation (LDA)**

Dalam pengolahan *text*, kita seringkali mengumpulkan beberapa jenis dokumen seperti postingan blog, berita, artikel, yang ingin kita petakan dan memahaminya secara terpisah. *Topic Modeling* merupakan sebuah metode untuk *unsupervised classification* untuk dokumen, mirip seperti klusterisasi pada data angka yang menemukan pengelompokan secara alami.

*Latent Dirichlet Allocation (LDA)* merupakan sebuah metode yang sangat umum digunakan untuk *topic modeling*. Caranya, kita akan memperlakukan setiap dokumen sebagai campuran dari berbagai topik yang setiap topik terdiri dari campuran kata-kata. Kondisi ini memungkinkan dokumen untuk tumpang tindih satu sama lain dalam hal konten. Dengan demikian dibutuhkan sebuah solusi untuk mencerminkan bahasa alami/khas dari setiap dokumen/konten.

```
library(knitr)
knitr::include_graphics("Topic_Modeling.png")
```

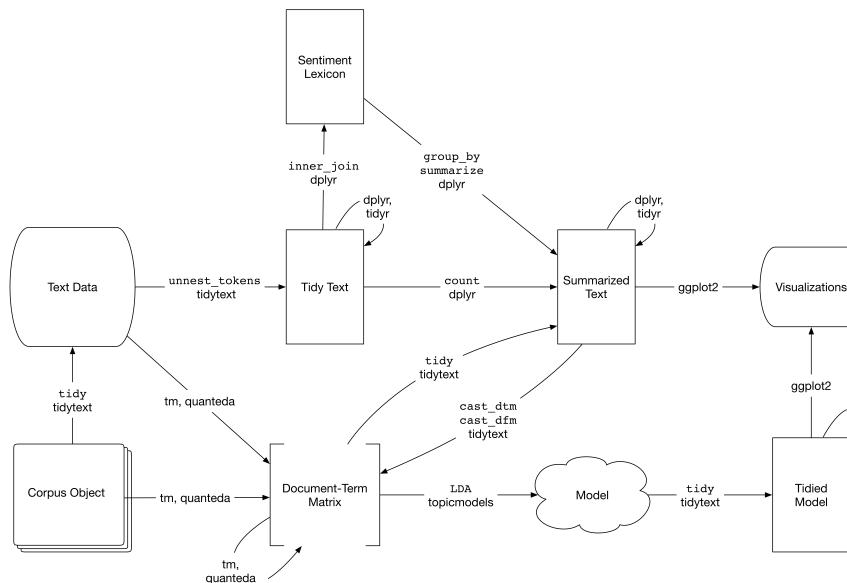


Figure 1: Flowchart topic modeling

Pada gambar 1.1 menunjukkan bahwa kita dapat menggunakan pendekatan *tidy text* untuk melakukan pemodelan topik. Pendekatan tersebut akan dibantu oleh paket-paket yang tersedia seperti **ggplot2** dan **dplyr** sebelum kita melakukan analisis *topic modeling*.

#### **Latent Dirichlet Allocation**

*Latent Dirichlet Allocation* merupakan algoritma yang sangat umum digunakan untuk melakukan analisis *topic modeling*. Tanpa mengetahui prinsip matematika secara lebih mendalam, kita dapat memahami jika mengikuti dua prinsip dibawah ini.

1. *Setiap dokumen merupakan campuran berbagai macam topik*, kita dapat membayangkan bahwa setiap dokumen terdiri dari berbagai macam kata dari beberapa topik dalam proporsi tertentu. Sebagai contoh, kita mempunyai dua dokumen, yaitu, dokumen A dan dokumen B. Dalam dokumen A, 90% merupakan topik politik dan 10% sisanya merupakan ekonomi. Sedangkan pada dokumen B, 30% merupakan topik politik dan 70% sisanya merupakan topik ekonomi.

2. Setiap topik merupakan campuran berbagai kata, sebagai contoh, kita dapat membayangkan model topik dari berita Indonesia, yakni, topik politik dan ekonomi. Kata-kata yang umum digunakan dalam politik umumnya adalah *Presiden*, *DPR*, dan *Pemerintah*, sedangkan dalam topik ekonomi terdiri dari kata *Inflasi*, *Pertumbuhan*, dan *Ekonomi* itu sendiri.

*LDA* merupakan metode matematika yang dapat menemukan campuran berbagai macam kata yang terkait dengan sebuah topik, disisi lain juga menentukan campuran topik yang dapat menggambarkan setiap dokumen. Pada tahapan selanjutnya, kita akan berlatih dengan data **AssociatedPress** yang disediakan oleh paket **topicmodels**. Data tersebut merupakan kumpulan artikel berita Amerika Serikat.

```
library(topicmodels)
data("AssociatedPress")
AssociatedPress

## <<DocumentTermMatrix (documents: 2246, terms: 10473)>>
## Non-/sparse entries: 302031/23220327
## Sparsity           : 99%
## Maximal term length: 18
## Weighting          : term frequency (tf)
```

Kita dapat menggunakan fungsi **LDA()** dari paket **topicmodels** dengan pengaturan **k=2** untuk membuat model **LDA** dua topik. Fungsi ini dapat membantu kita untuk mengetahui bagaimana berbagai macam kata dikaitkan dengan topik dan topik-topik dikaitkan dengan dokumen.

```
LDA_1 <- LDA(AssociatedPress, k = 2, control = list(seed = 1234))
LDA_1
```

```
## A LDA_VEM topic model with 2 topics.
```

### Word Topic Probabilities

Pada sub-bab ini, kita akan menggunakan sebuah packages bernama **tidytext** yang berasal dari **broom package** untuk merapikan objek model. Paket ini memungkinkan kita untuk mengekstraksi probabilitas topik atau kata dari setiap dokumen.

```
library(tidytext)
topic_1 <- tidy(LDA_1, matrix = "beta")
topic_1

## # A tibble: 20,946 x 3
##   topic term      beta
##   <int> <chr>    <dbl>
## 1     1 aaron 1.69e-12
## 2     2 aaron 3.90e- 5
## 3     1 abandon 2.65e- 5
## 4     2 abandon 3.99e- 5
## 5     1 abandoned 1.39e- 4
## 6     2 abandoned 5.88e- 5
## 7     1 abandoning 2.45e-33
## 8     2 abandoning 2.34e- 5
## 9     1 abbott 2.13e- 6
## 10    2 abbott 2.97e- 5
## # ... with 20,936 more rows
```

Pada setiap kombinasi, fungsi akan menghitung probabilitas dari istilah yang dihasilkan dari topik yang ada sehingga kita dapat mengetahui pembagian tiap topik dan probabilitasnya.

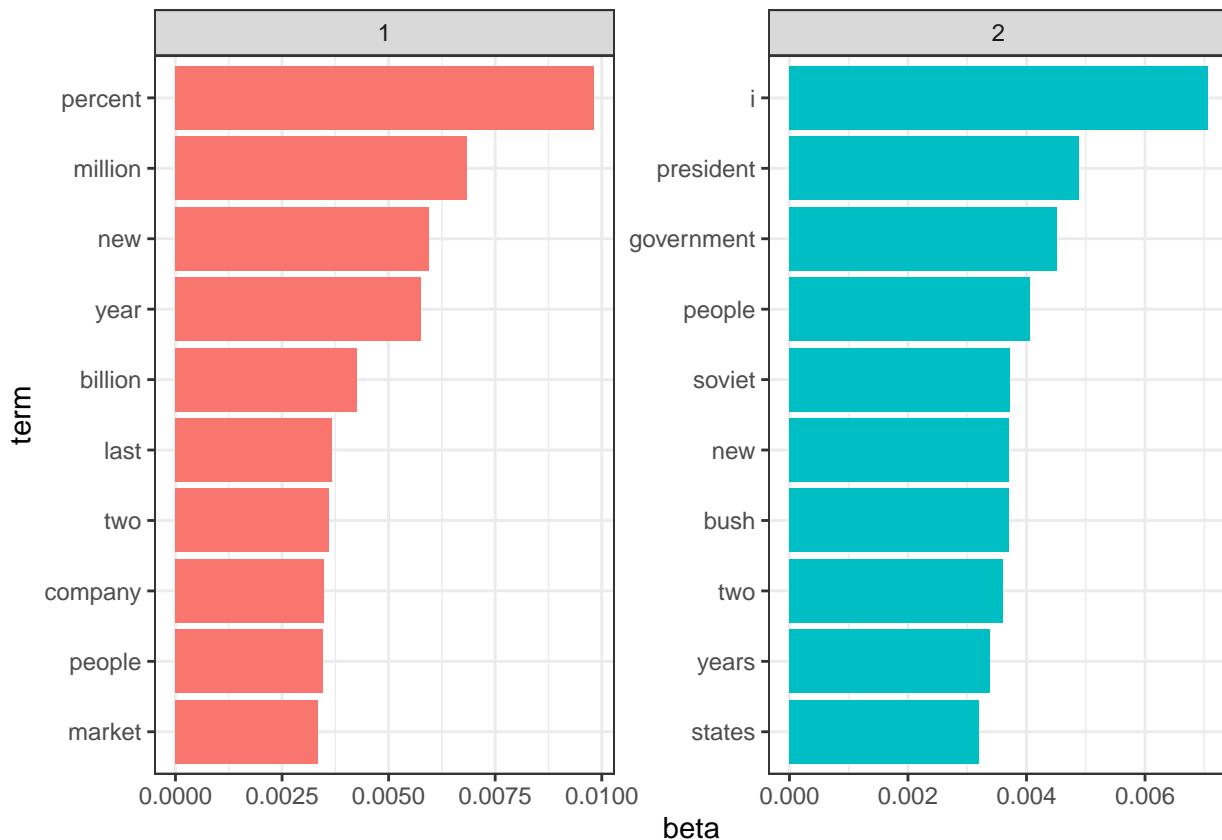
Pada tahap selanjutnya, kita dapat menggunakan fungsi **top\_n()** yang terdapat pada **dplyr** untuk menemukan 10 kata yang paling umum digunakan dalam setiap topik. Sebagai visualisasinya, kita akan

menggunakan **ggplot2**.

```
library(dplyr)
library(ggplot2)
library(tidytext)

# Mencari 10 kata yang paling umum digunakan dalam setiap topik
top_terms <- topic_1 %>%
  group_by(topic) %>%
  top_n(10, beta) %>%
  ungroup() %>%
  arrange(topic, -beta)

# Melakukan visualisasi terhadap 10 kata yang paling sering digunakan dalam topik
top_terms %>%
  mutate(term = reorder_within(term, beta, topic)) %>%
  ggplot(aes(term, beta, fill = factor(topic))) +
  geom_col(show.legend = FALSE) +
  facet_wrap(~topic, scales = "free") +
  coord_flip() +
  scale_x_reordered() +
  theme_bw()
```



Visualisasi di atas memungkinkan kita memahami dua topik yang diambil dari artikel. Kata-kata paling umum dalam topik pertama adalah *persen*, *juta*, *miliar*, dan *persen* yang menunjukkan bahwa topik pertama mewakili berita bisnis atau keuangan. Sedangkan pada topik kedua terdapat kata *presiden*, *pemerintah* dan *soviet* menunjukkan bahwa topik kedua mewakili berita politik. Namun, jika kita mengamati lebih rinci kita akan menemukan beberapa kata yang lebih umum seperti *baru* dan *orang*. Kata-kata dalam bahasa

keseharian akan memiliki kemungkinan lebih tinggi untuk saling tumpang tindih dalam beberapa topik.

Sebagai alternatif, kita dapat mempertimbangkan istilah yang memiliki perbedaan terbesar dalam rasio *beta* antara topik pertama dan kedua.

```
library(tidyr)
library(dplyr)
library(tidytext)

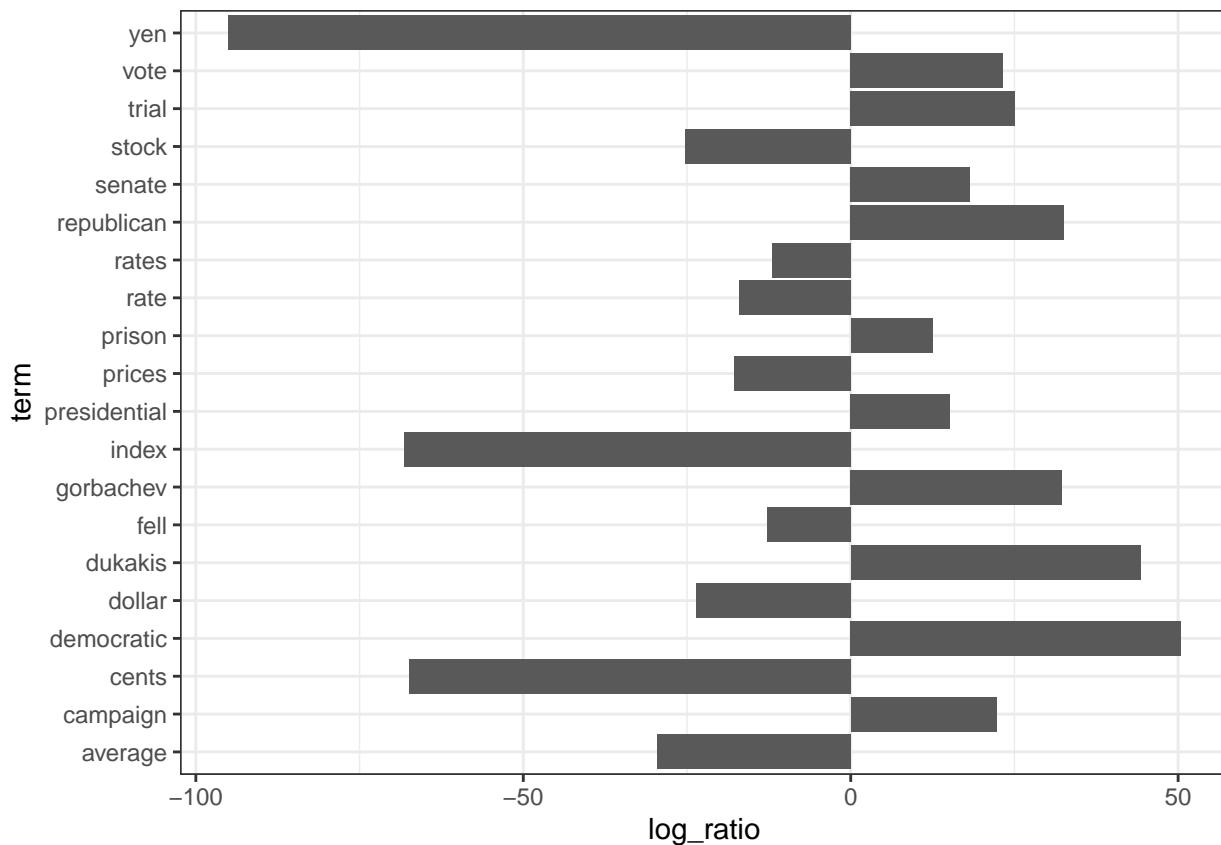
topic_spread <- topic_1 %>%
  mutate(topic = paste0("topic", topic)) %>%
  spread(topic, beta) %>%
  filter(topic1 > .001 | topic2 > .001) %>%
  mutate(log_ratio = log2 (topic2/topic1)) %>%
  top_n(10, log_ratio)

topic_spread_2 <- topic_1 %>%
  mutate(topic = paste0("topic", topic)) %>%
  spread(topic, beta) %>%
  filter(topic1 > .001 | topic2 > .001) %>%
  mutate(log_ratio = log2 (topic2/topic1)) %>%
  top_n(-10, log_ratio)

# Merge file with same column
topic_spread_all <- merge(x = topic_spread, y = topic_spread_2, all = TRUE)
```

Setelah melakukan preprosesing di atas, kita akan mencoba memvisualisasikan **log\_ratio** dari data *topic\_spread*.

```
library(ggplot2)
topic_spread_all %>%
  ggplot(aes(term, log_ratio)) +
  geom_col(show.legend = FALSE) +
  coord_flip() +
  scale_x_reordered() +
  theme_bw()
```



### Document-topic probabilities

Selain memperkirakan setiap topik sebagai campuran kata, LDA juga dapat memodelkan setiap dokumen sebagai campuran topik. Kita dapat memeriksa probabilitas dokumen setiap topik yang disebut sebagai gamma  $Y$ .

```
ap_documents <- tidy(LDA_1, matrix = "gamma")
ap_documents

## # A tibble: 4,492 x 3
##   document topic    gamma
##       <int> <int>    <dbl>
## 1         1     1  0.248
## 2         2     1  0.362
## 3         3     1  0.527
## 4         4     1  0.357
## 5         5     1  0.181
## 6         6     1  0.000588
## 7         7     1  0.773
## 8         8     1  0.00445
## 9         9     1  0.967
## 10        10    1  0.147
## # ... with 4,482 more rows
```

Masing-masing nilai di atas merupakan taksiran proporsi kata dari topik-topik yang berada dalam dokumen. Sebagai contoh, model di atas memperkirakan bahwa hanya terdapat sekitar 25% kata-kata dalam dokumen yang mengandung topik 1.

Kita dapat melihat bahwa beberapa dokumen dicampur dari dua topik, tapi dalam dokumen 6 hampir

keseluruhannya membahas topik dua. Untuk mengecek jawaban tersebut, kita dapat menggunakan `tidy()` pada dokumen dan mengetahui kata yang paling sering muncul dalam dokumen.

```
tidy(AssociatedPress) %>%  
  filter(document == 6) %>%  
  arrange(desc(count))
```

```
## # A tibble: 287 x 3  
##   document term      count  
##       <int> <chr>     <dbl>  
## 1         6 noriega     16  
## 2         6 panama      12  
## 3         6 jackson      6  
## 4         6 powell       6  
## 5         6 administration 5  
## 6         6 economic      5  
## 7         6 general       5  
## 8         6 i             5  
## 9         6 panamanian    5  
## 10        6 american      4  
## # ... with 277 more rows
```

Berdasarkan kata yang sering muncul dalam artikel, temuan ini mengeindikasikan bahwa artikel yang kita analisis membahas mengenai hubungan antara Pemerintah Amerika dengan Diktator Panama Manuel Noriega. Artinya, algoritma yang dirancang sebelumnya sudah tepat menempatkan artikel ini ke dalam topik politik/berita nasional.

### *Example: the great library heist*

Selanjutnya kita akan mencoba menggunakan metode statistik yang telah dibuat sebelumnya untuk menguji kasus yang sebenarnya kita telah mengetahui jawabannya. Misalnya, kita mengumpulkan satu set dokumen yang sudah pasti terkait dengan empat topik terpisah, kemudian kita melakukan pemodelan topik untuk melihat apakah algoritma dapat membedakan empat topik dengan benar. Tahap ini memungkinkan kita untuk memeriksa apakah metode yang sudah kita pelajari memang berguna dan tidak salah. Kita akan mencoba dengan beberapa data dari literatur lama.

Misalkan, ada seorang pencuri yang masuk ke dalam rumah Anda dan mengambil beberapa buku yang Anda miliki, seperti *Great Expectation* by Charles Dickens, *The War of Worlds* by H.G. Wells, *Twenty Thousand Leagues Under the Sea* by Jules Verne, *Pride and Prejudice* by Jane Austen.

```
titles <- c("Twenty Thousand Leagues under the Sea", "The War of the Worlds", "Pride and Prejudice", "Gr  
  
library(gutenbergr)  
books <- gutenberg_works(title %in% titles) %>%  
  gutenberg_download(meta_fields = "title")
```

Sebagai bagian dari preprosesing, kita akan membaginya menjadi beberapa bagian menggunakan `tidytext unest_token()` untuk memisahkannya menjadi kata-kata dan menghapus `stop_words`. Pada tahapan ini, kita akan memberlakukan setiap bab sebagai dokumen yang terpisah, masing-masing dengan nama *Great Expectations\_1* dan *prejudice\_11*.

```
library(stringr)  
  
# Membagi ke dalam beberapa dokumen, masing-masing satu bab.  
by_chapter <- books %>%  
  group_by(title) %>%  
  mutate(chapter = cumsum(str_detect(text, regex("^chapter ", ignore_case = TRUE)))) %>%
```

```

ungroup() %>%
  filter(chapter > 0) %>%
  unite(document, title, chapter)

# Membagi kedalam beberapa kata
by_chapter_word <- by_chapter %>%
  unnest_tokens(word, text)

# Menghitung kata pada dokumen
word_count <- by_chapter_word %>%
  anti_join(stop_words) %>%
  count(document, word, sort = TRUE) %>%
  ungroup()

word_count

## # A tibble: 68,023 x 3
##   document                 word     n
##   <chr>                   <chr>   <int>
## 1 The War of the Worlds_16 brother    50
## 2 Pride and Prejudice_43 elizabeth  36
## 3 Twenty Thousand Leagues under the Sea_26 captain  34
## 4 Twenty Thousand Leagues under the Sea_36 nautilus 34
## 5 Pride and Prejudice_18 darcy      33
## 6 Twenty Thousand Leagues under the Sea_27 sea       33
## 7 Twenty Thousand Leagues under the Sea_21 captain  32
## 8 Twenty Thousand Leagues under the Sea_29 captain  32
## 9 Twenty Thousand Leagues under the Sea_39 nautilus 31
## 10 Twenty Thousand Leagues under the Sea_44 nautilus 29
## # ... with 68,013 more rows

```

### LDA on chapters

Setelah mengetahui `word_counts` dari data yang telah rapi, kita belum dapat menggunakan paket `topicmodels` karena paket tersebut membutuhkan *Document Term Matrix*. Seperti yang dijelaskan di bab sebelumnya, kita dapat memasukan tabel token pada tiap baris ke dalam *Document Term Matrix* dengan fungsi `t_dtm` atau `cast_dtm()`

```

chapters_dtm <- word_count %>%
  cast_dtm(document, word, n)

chapters_dtm

## <<DocumentTermMatrix (documents: 134, terms: 14149)>>
## Non-/sparse entries: 68023/1827943
## Sparsity           : 96%
## Maximal term length: 17
## Weighting          : term frequency (tf)

```

Tahap selanjutnya, kita akan menggunakan fungsi LDA untuk membuat model empat topik. Dalam hal ini kami mencari empat topik karena hanya terdapat empat buku, mungkin dalam masalah nyata kita akan menemukan beberapa nilai yang berbeda.

```

chapters_lda <- LDA(chapters_dtm, k = 4, control = list(seed = 1234))

chapters_lda

```

```

## A LDA_VEM topic model with 4 topics.

Seperti yang telah kita lakukan pada data Associated Press, kita akan memeriksa probabilitas topik pada tiap kata.

chapter_topic <- tidy(chapters_lda, matrix = "beta")

chapter_topic

## # A tibble: 56,596 x 3
##   topic term      beta
##   <int> <chr>    <dbl>
## 1     1 brother  3.48e-118
## 2     2 brother  8.60e- 5
## 3     3 brother  1.77e- 3
## 4     4 brother  7.30e- 3
## 5     1 elizabeth 1.11e-116
## 6     2 elizabeth 5.28e- 19
## 7     3 elizabeth 1.60e- 2
## 8     4 elizabeth 1.90e-100
## 9     1 captain   1.59e- 2
## 10    2 captain   3.39e- 32
## # ... with 56,586 more rows

```

Perhatikan hasil di atas, kita telah mengubah model menjadi format per topik. Pada setiap baris, model akan menghitung probabilitas istilah. Misalnya, istilah *brother* memiliki kemungkinan hampir 0% pada topik 1.

Kita dapat menggunakan fungsi **top\_n** dalam **dplyr()** untuk menemukan 5 istilah teratas dalam setiap topik.

```

top_terms <- chapter_topic %>%
  group_by(topic) %>%
  top_n(5, beta) %>%
  ungroup() %>%
  arrange(topic, -beta)

```

```

top_terms

## # A tibble: 20 x 3
##   topic term      beta
##   <int> <chr>    <dbl>
## 1     1 captain  0.0159
## 2     1 nautilus 0.0136
## 3     1 nemo     0.00908
## 4     1 sea      0.00907
## 5     1 ned      0.00837
## 6     2 martians 0.00738
## 7     2 pit      0.00662
## 8     2 people   0.00621
## 9     2 time     0.00609
## 10    2 black    0.00561
## 11    3 elizabeth 0.0160
## 12    3 darcy    0.0100
## 13    3 bennet   0.00790
## 14    3 miss     0.00761

```

```

## 15    3 jane      0.00707
## 16    4 people    0.00747
## 17    4 brother   0.00730
## 18    4 martians  0.00615
## 19    4 black     0.00509
## 20    4 night     0.00505

```

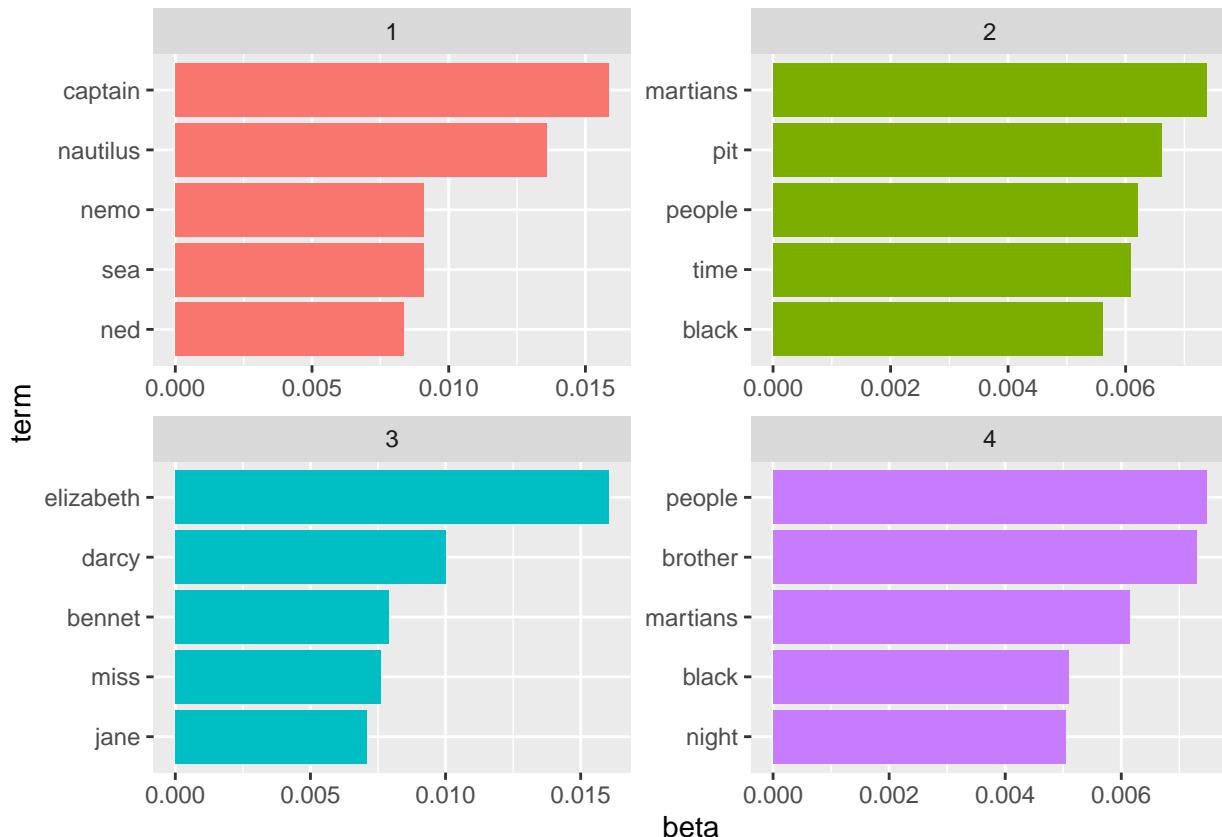
Tahapan selanjutnya, kita akan membuat visualisasi dari luaran data yang telah kita analisis sebelumnya.

```

library(ggplot2)

top_terms %>%
  mutate(term = reorder_within(term, beta, topic)) %>%
  ggplot(aes(term, beta, fill = factor(topic))) +
  geom_col(show.legend = FALSE) +
  facet_wrap(~topic, scales = "free") +
  coord_flip() +
  scale_x_reordered()

```



Dari visualisasi di atas, kita mengetahui bahwa istilah Captain memiliki probabilitas pada term 1 dan tidak pada topik lainnya, sedangkan martians memiliki probabilitas pada term 2. Melalui analisis topik modeling, kita dapat memetakan probabilitas kata dalam sebuah topik yang berbeda.

## Chapter 7

### ***Hubungan Antar Kata/Terms: n-grams and hubungan***

Pada dasarnya, kata atau diksi merupakan sebuah unit terkecil yang memiliki keterkaitan dengan sentimen atau sebuah dokumen. Dalam chapter 7, kita akan mengeksplorasi beberapa metode yang ditawarkan oleh **tidytext** untuk menghitung dan memvisualisasikan hubungan antara kata-kata pada sebuah data. Pada bab ini, kita akan belajar melakukan teknisasi menggunakan **ngrams**. Selain itu, pada chapter ini kita juga akan mengenalkan dua paket baru, yakni, **ggraph** dan **widyr**.

#### ***Tokenisasi Menggunakan n-gram***

Pada chapter sebelumnya, kita telah menggunakan fungsi **unnest\_token** yang berfungsi untuk tokenisasi kata atau bahkan kalimat ketika menganalisis sentimen dan frekuensi. Tetapi, lebih jauh lagi kita dapat menggunakan fungsi **unnest\_token** untuk menandai urutan kata yang kita sebut sebagai **n-gram**. Melalui fungsi tersebut, kita akan melihat seberapa sering sebuah kata diikuti oleh kata lainnya yang kemudian membangun model hubungan diantara mereka.

```
library(dplyr)
library(tidytext)
library(janeaustenr)
library(tidyverse)

austen_bigram <- austen_books()
austen_bigram <- austen_bigram %>%
  unnest_tokens(austen_bigram, text, token = "ngrams", n = 2)

austen_bigram

## # A tibble: 725,049 x 2
##   book           austen_bigram
##   <fct>          <chr>
## 1 Sense & Sensibility sense and
## 2 Sense & Sensibility and sensibility
## 3 Sense & Sensibility sensibility by
## 4 Sense & Sensibility by jane
## 5 Sense & Sensibility jane austen
## 6 Sense & Sensibility austen 1811
## 7 Sense & Sensibility 1811 chapter
## 8 Sense & Sensibility chapter 1
## 9 Sense & Sensibility 1 the
## 10 Sense & Sensibility the family
## # ... with 725,039 more rows
```

Struktur data di atas merupakan hasil dari format teks yang sebenarnya sudah rapi. Semua data berasal dari data buku yang sama, hanya saja tokenisasi dilakukan ber baris dengan metdata tambahan, yakni, nama buku.

#### ***Counting dan filtering n-grams***

Terdapat metode yang umum digunakan dalam analisis data yang dapat digunakan untuk merapikan data **n-grams**. Kita akan menggunakan salah satu fungsi dari **dplyr()**, yakni, **count()**:

```
austen_bigram %>%
  count(austen_bigram, sort = TRUE)
```

```

## # A tibble: 211,236 x 2
##   austen_bigram     n
##   <chr>           <int>
## 1 of the          3017
## 2 to be           2787
## 3 in the          2368
## 4 it was          1781
## 5 i am            1545
## 6 she had         1472
## 7 of her           1445
## 8 to the           1387
## 9 she was          1377
## 10 had been        1299
## # ... with 211,226 more rows

```

Seperti yang kita duga, banyak bigrams yang merupakan gabungan kata-kata umum seperti kata hubung dan tidak menjadi esensi dalam sebuah kalimat. Pada tahapan ini, kita akan menggunakan fungsi `tidyr` untuk memisahkan antara kata pertama dan kedua dan menghapus jika kedua kolom tersebut merupakan kata hubung dalam sebuah kalimat.

```

library(tidyr)

bigram_separated <- austen_bigram %>%
  separate(austen_bigram, c("Word1", "Word2"), sep = " ")

bigram_filtered <- bigram_separated %>%
  filter(!Word1 %in% stop_words$word) %>%
  filter(!Word2 %in% stop_words$word)

# new bigram count
bigram_count <- bigram_filtered %>%
  count(Word1, Word2, sort = TRUE)

bigram_count

## # A tibble: 33,421 x 3
##   Word1    Word2     n
##   <chr>    <chr>   <int>
## 1 sir      thomas    287
## 2 miss     crawford  215
## 3 captain  wentworth 170
## 4 miss     woodhouse 162
## 5 frank    churchill 132
## 6 lady     russell   118
## 7 lady     bertram   114
## 8 sir      walter    113
## 9 miss     fairfax   109
## 10 colonel brandon   108
## # ... with 33,411 more rows

```

Dari fungsi di atas, kita dapat melihat bahwa pasangan yang paling umum dalam bigram berasal dari Jane Austen. Dalam kondisi tertentu, kita mungkin juga akan mengkombinasikan ulang kata-kata di atas. Fungsi `tidy unite()` adalah kebalikan dari `separate()`. Mari kita menyatukan dua kata yang berbeda menggunakan fungsi `unite()`.

```

bigram_united <- bigram_filtered %>%
  unite(bigram, Word1, Word2, sep = " ")

bigram_united

## # A tibble: 44,784 x 2
##   book           bigram
##   <fct>          <chr>
## 1 Sense & Sensibility jane austen
## 2 Sense & Sensibility austen 1811
## 3 Sense & Sensibility 1811 chapter
## 4 Sense & Sensibility chapter 1
## 5 Sense & Sensibility norland park
## 6 Sense & Sensibility surrounding acquaintance
## 7 Sense & Sensibility late owner
## 8 Sense & Sensibility advanced age
## 9 Sense & Sensibility constant companion
## 10 Sense & Sensibility happened ten
## # ... with 44,774 more rows

```

Bahkan, dalam kondisi tertentu, anda tertarik menggunakan analisis trigram dan meninggalkan bigram. Anda ingin mengurutkan 3 kata berurutan dan menetapkan nilai n sebesar 3.

```

austen_books() %>%
  unnest_tokens(trigram, text, token = "ngrams", n = 3) %>%
  separate(trigram, c("word1", "word2", "word3"), sep = " ") %>%
  filter(!word1 %in% stop_words$word,
         !word2 %in% stop_words$word,
         !word3 %in% stop_words$word) %>%
  count(word1, word2, word3, sort = TRUE)

```

```

## # A tibble: 8,757 x 4
##   word1    word2    word3      n
##   <chr>    <chr>    <chr>    <int>
## 1 dear     miss     woodhouse  23
## 2 miss     de       bourgh   18
## 3 lady     catherine de      14
## 4 catherine de     bourgh   13
## 5 poor     miss     taylor   11
## 6 sir      walter   elliot   11
## 7 ten      thousand pounds  11
## 8 dear     sir      thomas   10
## 9 twenty   thousand pounds  8
## 10 replied  miss     crawford 7
## # ... with 8,747 more rows

```

### *Analyzing Bigram*

Format analisis bigram satu baris merupakan format yang bermanfaat untuk melakukan analisis eksplorasi teks. Sebagai contoh, kita mungkin tertarik pada nama seseorang yang paling umum disebutkan dalam sebuah buku.

```

bigram_filtered %>%
  filter(Word2 == "elinor") %>%
  count(book, Word1, sort = TRUE)

```

```

## # A tibble: 161 x 3
##   book           Word1     n
##   <fct>         <chr>    <int>
## 1 Sense & Sensibility replied  26
## 2 Sense & Sensibility cried   7
## 3 Sense & Sensibility answered 6
## 4 Sense & Sensibility dear   6
## 5 Sense & Sensibility continued 5
## 6 Sense & Sensibility returned 4
## 7 Sense & Sensibility poor   3
## 8 Sense & Sensibility subject 3
## 9 Sense & Sensibility word   3
## 10 Sense & Sensibility world  3
## # ... with 151 more rows

```

Bigram juga dapat digunakan pada sebuah dokumen dan difungsikan pada kata yang berdiri sendiri. Sebagai contoh, kita telah menggunakan tf-idf pada bab sebelumnya pada Austen Novel. Sekarang, kita akan menggunakan bigram pada tf-idf.

```

bigram_tf_idf <- bigram_united %>%
  count(book, bigram) %>%
  bind_tf_idf(bigram, book, n) %>%
  arrange(desc(tf_idf)) %>%
  group_by(book)

```

`bigram_tf_idf`

```

## # A tibble: 36,217 x 6
## # Groups:   book [6]
##   book           bigram      n      tf     idf tf_idf
##   <fct>         <chr>     <int>  <dbl>  <dbl>  <dbl>
## 1 Persuasion     captain wentworth 170 0.0299  1.79 0.0535
## 2 Mansfield Park sir thomas    287 0.0287  1.79 0.0515
## 3 Mansfield Park miss crawford 215 0.0215  1.79 0.0386
## 4 Persuasion     lady russell 118 0.0207  1.79 0.0371
## 5 Persuasion     sir walter 113 0.0198  1.79 0.0356
## 6 Emma           miss woodhouse 162 0.0170  1.79 0.0305
## 7 Northanger Abbey miss tilney  82 0.0159  1.79 0.0286
## 8 Sense & Sensibility colonel brandon 108 0.0150  1.79 0.0269
## 9 Emma           frank churchill 132 0.0139  1.79 0.0248
## 10 Pride & Prejudice   lady catherine 100 0.0138  1.79 0.0247
## # ... with 36,207 more rows

```

### *Menggunakan Bigram untuk Melihat Konteks dalam Sentiment Analyst*

Dalam analisis sentimen yang sudah kita bahas sebelumnya, kita mengelompokkan kalimat negatif dan positif dalam kelompok berbeda menggunakan lexicon. Salah satu permasalahan yang akan kita temui dalam melihat analisis sentimen adalah melihat konteks di dalamnya. Menggunakan bigram, kita akan melihat konteks dari analisis sentimen.

```

bigram_separated %>%
  filter(Word1 == "not") %>%
  count(Word1, Word2, sort = TRUE)

```

```

## # A tibble: 1,246 x 3
##   Word1 Word2     n
##   <chr> <chr> <int>
## 1 not   not    1246
## 2 not   and    1246
## 3 not   or     1246
## 4 not   but    1246
## 5 not   for    1246
## 6 not   because 1246
## 7 not   since  1246
## 8 not   if     1246
## 9 not   than   1246
## 10 not  unless 1246
## 11 not  while  1246
## 12 not  when  1246
## 13 not  as    1246
## 14 not  that   1246
## 15 not  who   1246
## 16 not  which  1246
## 17 not  where  1246
## 18 not  why   1246
## 19 not  how   1246
## 20 not  who's  1246
## 21 not  what's 1246
## 22 not  what'll 1246
## 23 not  what'd  1246
## 24 not  what're 1246
## 25 not  what've 1246
## 26 not  what'm  1246
## 27 not  what'v  1246
## 28 not  what'w  1246
## 29 not  what'z  1246
## 30 not  what'x  1246
## 31 not  what'c  1246
## 32 not  what'p  1246
## 33 not  what'q  1246
## 34 not  what'r  1246
## 35 not  what'g  1246
## 36 not  what'k  1246
## 37 not  what'j  1246
## 38 not  what'z  1246
## 39 not  what'x  1246
## 40 not  what'c  1246
## 41 not  what'p  1246
## 42 not  what'q  1246
## 43 not  what'r  1246
## 44 not  what'g  1246
## 45 not  what'k  1246
## 46 not  what'j  1246
## 47 not  what'z  1246
## 48 not  what'x  1246
## 49 not  what'c  1246
## 50 not  what'p  1246
## 51 not  what'q  1246
## 52 not  what'r  1246
## 53 not  what'g  1246
## 54 not  what'k  1246
## 55 not  what'j  1246
## 56 not  what'z  1246
## 57 not  what'x  1246
## 58 not  what'c  1246
## 59 not  what'p  1246
## 60 not  what'q  1246
## 61 not  what'r  1246
## 62 not  what'g  1246
## 63 not  what'k  1246
## 64 not  what'j  1246
## 65 not  what'z  1246
## 66 not  what'x  1246
## 67 not  what'c  1246
## 68 not  what'p  1246
## 69 not  what'q  1246
## 70 not  what'r  1246
## 71 not  what'g  1246
## 72 not  what'k  1246
## 73 not  what'j  1246
## 74 not  what'z  1246
## 75 not  what'x  1246
## 76 not  what'c  1246
## 77 not  what'p  1246
## 78 not  what'q  1246
## 79 not  what'r  1246
## 80 not  what'g  1246
## 81 not  what'k  1246
## 82 not  what'j  1246
## 83 not  what'z  1246
## 84 not  what'x  1246
## 85 not  what'c  1246
## 86 not  what'p  1246
## 87 not  what'q  1246
## 88 not  what'r  1246
## 89 not  what'g  1246
## 90 not  what'k  1246
## 91 not  what'j  1246
## 92 not  what'z  1246
## 93 not  what'x  1246
## 94 not  what'c  1246
## 95 not  what'p  1246
## 96 not  what'q  1246
## 97 not  what'r  1246
## 98 not  what'g  1246
## 99 not  what'k  1246
## 100 not  what'j  1246
## 101 not  what'z  1246
## 102 not  what'x  1246
## 103 not  what'c  1246
## 104 not  what'p  1246
## 105 not  what'q  1246
## 106 not  what'r  1246
## 107 not  what'g  1246
## 108 not  what'k  1246
## 109 not  what'j  1246
## 110 not  what'z  1246
## 111 not  what'x  1246
## 112 not  what'c  1246
## 113 not  what'p  1246
## 114 not  what'q  1246
## 115 not  what'r  1246
## 116 not  what'g  1246
## 117 not  what'k  1246
## 118 not  what'j  1246
## 119 not  what'z  1246
## 120 not  what'x  1246
## 121 not  what'c  1246
## 122 not  what'p  1246
## 123 not  what'q  1246
## 124 not  what'r  1246
## 125 not  what'g  1246
## 126 not  what'k  1246
## 127 not  what'j  1246
## 128 not  what'z  1246
## 129 not  what'x  1246
## 130 not  what'c  1246
## 131 not  what'p  1246
## 132 not  what'q  1246
## 133 not  what'r  1246
## 134 not  what'g  1246
## 135 not  what'k  1246
## 136 not  what'j  1246
## 137 not  what'z  1246
## 138 not  what'x  1246
## 139 not  what'c  1246
## 140 not  what'p  1246
## 141 not  what'q  1246
## 142 not  what'r  1246
## 143 not  what'g  1246
## 144 not  what'k  1246
## 145 not  what'j  1246
## 146 not  what'z  1246
## 147 not  what'x  1246
## 148 not  what'c  1246
## 149 not  what'p  1246
## 150 not  what'q  1246
## 151 not  what'r  1246
## 152 not  what'g  1246
## 153 not  what'k  1246
## 154 not  what'j  1246
## 155 not  what'z  1246
## 156 not  what'x  1246
## 157 not  what'c  1246
## 158 not  what'p  1246
## 159 not  what'q  1246
## 160 not  what'r  1246
## 161 not  what'g  1246
## 162 not  what'k  1246
## 163 not  what'j  1246
## 164 not  what'z  1246
## 165 not  what'x  1246
## 166 not  what'c  1246
## 167 not  what'p  1246
## 168 not  what'q  1246
## 169 not  what'r  1246
## 170 not  what'g  1246
## 171 not  what'k  1246
## 172 not  what'j  1246
## 173 not  what'z  1246
## 174 not  what'x  1246
## 175 not  what'c  1246
## 176 not  what'p  1246
## 177 not  what'q  1246
## 178 not  what'r  1246
## 179 not  what'g  1246
## 180 not  what'k  1246
## 181 not  what'j  1246
## 182 not  what'z  1246
## 183 not  what'x  1246
## 184 not  what'c  1246
## 185 not  what'p  1246
## 186 not  what'q  1246
## 187 not  what'r  1246
## 188 not  what'g  1246
## 189 not  what'k  1246
## 190 not  what'j  1246
## 191 not  what'z  1246
## 192 not  what'x  1246
## 193 not  what'c  1246
## 194 not  what'p  1246
## 195 not  what'q  1246
## 196 not  what'r  1246
## 197 not  what'g  1246
## 198 not  what'k  1246
## 199 not  what'j  1246
## 200 not  what'z  1246
## 201 not  what'x  1246
## 202 not  what'c  1246
## 203 not  what'p  1246
## 204 not  what'q  1246
## 205 not  what'r  1246
## 206 not  what'g  1246
## 207 not  what'k  1246
## 208 not  what'j  1246
## 209 not  what'z  1246
## 210 not  what'x  1246
## 211 not  what'c  1246
## 212 not  what'p  1246
## 213 not  what'q  1246
## 214 not  what'r  1246
## 215 not  what'g  1246
## 216 not  what'k  1246
## 217 not  what'j  1246
## 218 not  what'z  1246
## 219 not  what'x  1246
## 220 not  what'c  1246
## 221 not  what'p  1246
## 222 not  what'q  1246
## 223 not  what'r  1246
## 224 not  what'g  1246
## 225 not  what'k  1246
## 226 not  what'j  1246
## 227 not  what'z  1246
## 228 not  what'x  1246
## 229 not  what'c  1246
## 230 not  what'p  1246
## 231 not  what'q  1246
## 232 not  what'r  1246
## 233 not  what'g  1246
## 234 not  what'k  1246
## 235 not  what'j  1246
## 236 not  what'z  1246
## 237 not  what'x  1246
## 238 not  what'c  1246
## 239 not  what'p  1246
## 240 not  what'q  1246
## 241 not  what'r  1246
## 242 not  what'g  1246
## 243 not  what'k  1246
## 244 not  what'j  1246
## 245 not  what'z  1246
## 246 not  what'x  1246
## 247 not  what'c  1246
## 248 not  what'p  1246
## 249 not  what'q  1246
## 250 not  what'r  1246
## 251 not  what'g  1246
## 252 not  what'k  1246
## 253 not  what'j  1246
## 254 not  what'z  1246
## 255 not  what'x  1246
## 256 not  what'c  1246
## 257 not  what'p  1246
## 258 not  what'q  1246
## 259 not  what'r  1246
## 260 not  what'g  1246
## 261 not  what'k  1246
## 262 not  what'j  1246
## 263 not  what'z  1246
## 264 not  what'x  1246
## 265 not  what'c  1246
## 266 not  what'p  1246
## 267 not  what'q  1246
## 268 not  what'r  1246
## 269 not  what'g  1246
## 270 not  what'k  1246
## 271 not  what'j  1246
## 272 not  what'z  1246
## 273 not  what'x  1246
## 274 not  what'c  1246
## 275 not  what'p  1246
## 276 not  what'q  1246
## 277 not  what'r  1246
## 278 not  what'g  1246
## 279 not  what'k  1246
## 280 not  what'j  1246
## 281 not  what'z  1246
## 282 not  what'x  1246
## 283 not  what'c  1246
## 284 not  what'p  1246
## 285 not  what'q  1246
## 286 not  what'r  1246
## 287 not  what'g  1246
## 288 not  what'k  1246
## 289 not  what'j  1246
## 290 not  what'z  1246
## 291 not  what'x  1246
## 292 not  what'c  1246
## 293 not  what'p  1246
## 294 not  what'q  1246
## 295 not  what'r  1246
## 296 not  what'g  1246
## 297 not  what'k  1246
## 298 not  what'j  1246
## 299 not  what'z  1246
## 300 not  what'x  1246
## 301 not  what'c  1246
## 302 not  what'p  1246
## 303 not  what'q  1246
## 304 not  what'r  1246
## 305 not  what'g  1246
## 306 not  what'k  1246
## 307 not  what'j  1246
## 308 not  what'z  1246
## 309 not  what'x  1246
## 310 not  what'c  1246
## 311 not  what'p  1246
## 312 not  what'q  1246
## 313 not  what'r  1246
## 314 not  what'g  1246
## 315 not  what'k  1246
## 316 not  what'j  1246
## 317 not  what'z  1246
## 318 not  what'x  1246
## 319 not  what'c  1246
## 320 not  what'p  1246
## 321 not  what'q  1246
## 322 not  what'r  1246
## 323 not  what'g  1246
## 324 not  what'k  1246
## 325 not  what'j  1246
## 326 not  what'z  1246
## 327 not  what'x  1246
## 328 not  what'c  1246
## 329 not  what'p  1246
## 330 not  what'q  1246
## 331 not  what'r  1246
## 332 not  what'g  1246
## 333 not  what'k  1246
## 334 not  what'j  1246
## 335 not  what'z  1246
## 336 not  what'x  1246
## 337 not  what'c  1246
## 338 not  what'p  1246
## 339 not  what'q  1246
## 340 not  what'r  1246
## 341 not  what'g  1246
## 342 not  what'k  1246
## 343 not  what'j  1246
## 344 not  what'z  1246
## 345 not  what'x  1246
## 346 not  what'c  1246
## 347 not  what'p  1246
## 348 not  what'q  1246
## 349 not  what'r  1246
## 350 not  what'g  1246
## 351 not  what'k  1246
## 352 not  what'j  1246
## 353 not  what'z  1246
## 354 not  what'x  1246
## 355 not  what'c  1246
## 356 not  what'p  1246
## 357 not  what'q  1246
## 358 not  what'r  1246
## 359 not  what'g  1246
## 360 not  what'k  1246
## 361 not  what'j  1246
## 362 not  what'z  1246
## 363 not  what'x  1246
## 364 not  what'c  1246
## 365 not  what'p  1246
## 366 not  what'q  1246
## 367 not  what'r  1246
## 368 not  what'g  1246
## 369 not  what'k  1246
## 370 not  what'j  1246
## 371 not  what'z  1246
## 372 not  what'x  1246
## 373 not  what'c  1246
## 374 not  what'p  1246
## 375 not  what'q  1246
## 376 not  what'r  1246
## 377 not  what'g  1246
## 378 not  what'k  1246
## 379 not  what'j  1246
## 380 not  what'z  1246
## 381 not  what'x  1246
## 382 not  what'c  1246
## 383 not  what'p  1246
## 384 not  what'q  1246
## 385 not  what'r  1246
## 386 not  what'g  1246
## 387 not  what'k  1246
## 388 not  what'j  1246
## 389 not  what'z  1246
## 390 not  what'x  1246
## 391 not  what'c  1246
## 392 not  what'p  1246
## 393 not  what'q  1246
## 394 not  what'r  1246
## 395 not  what'g  1246
## 396 not  what'k  1246
## 397 not  what'j  1246
## 398 not  what'z  1246
## 399 not  what'x  1246
## 400 not  what'c  1246
## 401 not  what'p  1246
## 402 not  what'q  1246
## 403 not  what'r  1246
## 404 not  what'g  1246
## 405 not  what'k  1246
## 406 not  what'j  1246
## 407 not  what'z  1246
## 408 not  what'x  1246
## 409 not  what'c  1246
## 410 not  what'p  1246
## 411 not  what'q  1246
## 412 not  what'r  1246
## 413 not  what'g  1246
## 414 not  what'k  1246
## 415 not  what'j  1246
## 416 not  what'z  1246
## 417 not  what'x  1246
## 418 not  what'c  1246
## 419 not  what'p  1246
## 420 not  what'q  1246
## 421 not  what'r  1246
## 422 not  what'g  1246
## 423 not  what'k  1246
## 424 not  what'j  1246
## 425 not  what'z  1246
## 426 not  what'x  1246
## 427 not  what'c  1246
## 428 not  what'p  1246
## 429 not  what'q  1246
## 430 not  what'r  1246
## 431 not  what'g  1246
## 432 not  what'k  1246
## 433 not  what'j  1246
## 434 not  what'z  1246
## 435 not  what'x  1246
## 436 not  what'c  1246
## 437 not  what'p  1246
## 438 not  what'q  1246
## 439 not  what'r  1246
## 440 not  what'g  1246
## 441 not  what'k  1246
## 442 not  what'j  1246
## 443 not  what'z  1246
## 444 not  what'x  1246
## 445 not  what'c  1246
## 446 not  what'p  1246
## 447 not  what'q  1246
## 448 not  what'r  1246
## 449 not  what'g  1246
## 450 not  what'k  1246
## 451 not  what'j  1246
## 452 not  what'z  1246
## 453 not  what'x  1246
## 454 not  what'c  1246
## 455 not  what'p  1246
## 456 not  what'q  1246
## 457 not  what'r  1246
## 458 not  what'g  1246
## 459 not  what'k  1246
## 460 not  what'j  1246
## 461 not  what'z  1246
## 462 not  what'x  1246
## 463 not  what'c  1246
## 464 not  what'p  1246
## 465 not  what'q  1246
## 466 not  what'r  1246
## 467 not  what'g  1246
## 468 not  what'k  1246
## 469 not  what'j  1246
## 470 not  what'z  1246
## 471 not  what'x  1246
## 472 not  what'c  1246
## 473 not  what'p  1246
## 474 not  what'q  1246
## 475 not  what'r  1246
## 476 not  what'g  1246
## 477 not  what'k  1246
## 478 not  what'j  1246
## 479 not  what'z  1246
## 480 not  what'x  1246
## 481 not  what'c  1246
## 482 not  what'p  1246
## 483 not  what'q  1246
## 484 not  what'r  1246
## 485 not  what'g  1246
## 486 not  what'k  1246
## 487 not  what'j  1246
## 488 not  what'z  1246
## 489 not  what'x  1246
## 490 not  what'c  1246
## 491 not  what'p  1246
## 492 not  what'q  1246
## 493 not  what'r  1246
## 494 not  what'g  1246
## 495 not  what'k  1246
## 496 not  what'j  1246
## 497 not  what'z  1246
## 498 not  what'x  1246
## 499 not  what'c  1246
## 500 not  what'p  1246
## 501 not  what'q  1246
## 502 not  what'r  1246
## 503 not  what'g  1246
## 504 not  what'k  1246
## 505 not  what'j  1246
## 506 not  what'z  1246
## 507 not  what'x  1246
## 508 not  what'c  1246
## 509 not  what'p  1246
## 510 not  what'q  1246
## 511 not  what'r  1246
## 512 not  what'g  1246
## 513 not  what'k  1246
## 514 not  what'j  1246
## 515 not  what'z  1246
## 516 not  what'x  1246
## 517 not  what'c  1246
## 518 not  what'p  1246
## 519 not  what'q  1246
## 520 not  what'r  1246
## 521 not  what'g  1246
## 522 not  what'k  1246
## 523 not  what'j  1246
## 524 not  what'z  1246
## 525 not  what'x  1246
## 526 not  what'c  1246
## 527 not  what'p  1246
## 528 not  what'q  1246
## 529 not  what'r  1246
## 530 not  what'g  1246
## 531 not  what'k  1246
## 532 not  what'j  1246
## 533 not  what'z  1246
## 534 not  what'x  1246
## 535 not  what'c  1246
## 536 not  what'p  1246
## 537 not  what'q  1246
## 538 not  what'r  1246
## 539 not  what'g  1246
## 540 not  what'k  1246
## 541 not  what'j  1246
## 542 not  what'z  1246
## 543 not  what'x  1246
## 544 not  what'c  1246
## 545 not  what'p  1246
## 546 not  what'q  1246
## 547 not  what'r  1246
## 548 not  what'g  1246
## 549 not  what'k  1246
## 550 not  what'j  1246
## 551 not  what'z  1246
## 552 not  what'x  1246
## 553 not  what'c  1246
## 554 not  what'p  1246
## 555 not  what'q  1246
## 556 not  what'r  1246
## 557 not  what'g  1246
## 558 not  what'k  1246
## 559 not  what'j  1246
## 560 not  what'z  1246
## 561 not  what'x  1246
## 562 not  what'c  1246
## 563 not  what'p  1246
## 564 not  what'q  1246
## 565 not  what'r  1246
## 566 not  what'g  1246
## 567 not  what'k  1246
## 568 not  what'j  1246
## 569 not  what'z  1246
## 570 not  what'x  1246
## 571 not  what'c  1246
## 572 not  what'p  1246
## 573 not  what'q  1246
## 574 not  what'r  1246
## 575 not  what'g  1246
## 576 not  what'k  1246
## 577 not  what'j  1246
## 578 not  what'z  1246
## 579 not  what'x  1246
## 580 not  what'c  1246
## 581 not  what'p  1246
## 582 not  what'q  1246
## 583 not  what'r  1246
## 584 not  what'g  1246
## 585 not  what'k  1246
## 586 not  what'j  1246
## 587 not  what'z  1246
## 588 not  what'x  1246
## 589 not  what'c  1246
## 590 not  what'p  1246
## 591 not  what'q  1246
## 592 not  what'r  1246
## 593 not  what'g  1246
## 594 not  what'k  1246
## 595 not  what'j  1246
## 596 not  what'z  1246
## 597 not  what'x  1246
## 598 not  what'c  1246
## 599 not  what'p  1246
## 600 not  what'q  1246
## 601 not  what'r  1246
## 602 not  what'g  1246
## 603 not  what'k  1246
## 604 not  what'j  1246
## 605 not  what'z  1246
## 606 not  what'x  1246
## 607 not  what'c  1246
## 608 not  what'p  1246
## 609 not  what'q  1246
## 6
```

```

##      <chr> <chr> <int>
## 1 not    be      610
## 2 not    to      355
## 3 not    have   327
## 4 not    know   252
## 5 not    a      189
## 6 not    think  176
## 7 not    been   160
## 8 not    the    147
## 9 not    at     129
## 10 not   in     118
## # ... with 1,236 more rows

```

Dengan melakukan analisis dengan bigram pada data sentimen, kita dapat memeriksa hubungan antar kata yang didahului dengan kata “tidak” atau kata-kata lainnya. Kita akan menggunakan leksikon AFFIN untuk melakukan analisis sentimen yang mungkin Anda ingat dan memberikan nilai sentimen pada tiap kata.

```

library(readr)
library(textdata)
AFFIN <- read.delim("AFINN/AFINN-111.txt")
AFFIN <- AFFIN %>%
  tibble::rownames_to_column("word")

```

Setelah itu kita memeriksa kata-kata yang sering didahului dengan kata “tidak” dan dikaitkan dengan sentimen.

```

not_words <- bigram_separated %>%
  filter(Word1 == "not") %>%
  inner_join(AFFIN, by = c(Word2 = "word")) %>%
  count(Word2, value, sort = TRUE)

not_words

## # A tibble: 245 x 3
##       Word2   value     n
##       <chr>   <int> <int>
## 1 like        2     99
## 2 help        2     82
## 3 want        1     45
## 4 wish        1     39
## 5 allow       1     36
## 6 care        2     23
## 7 sorry       -1    21
## 8 leave       -1    18
## 9 pretend     -1    18
## 10 worth      2     17
## # ... with 235 more rows

```

Misalnya, kata yang berhubungan dengan sentimen paling umum mengikuti kata “tidak” adalah “seperti” yang biasanya memiliki skor 2.

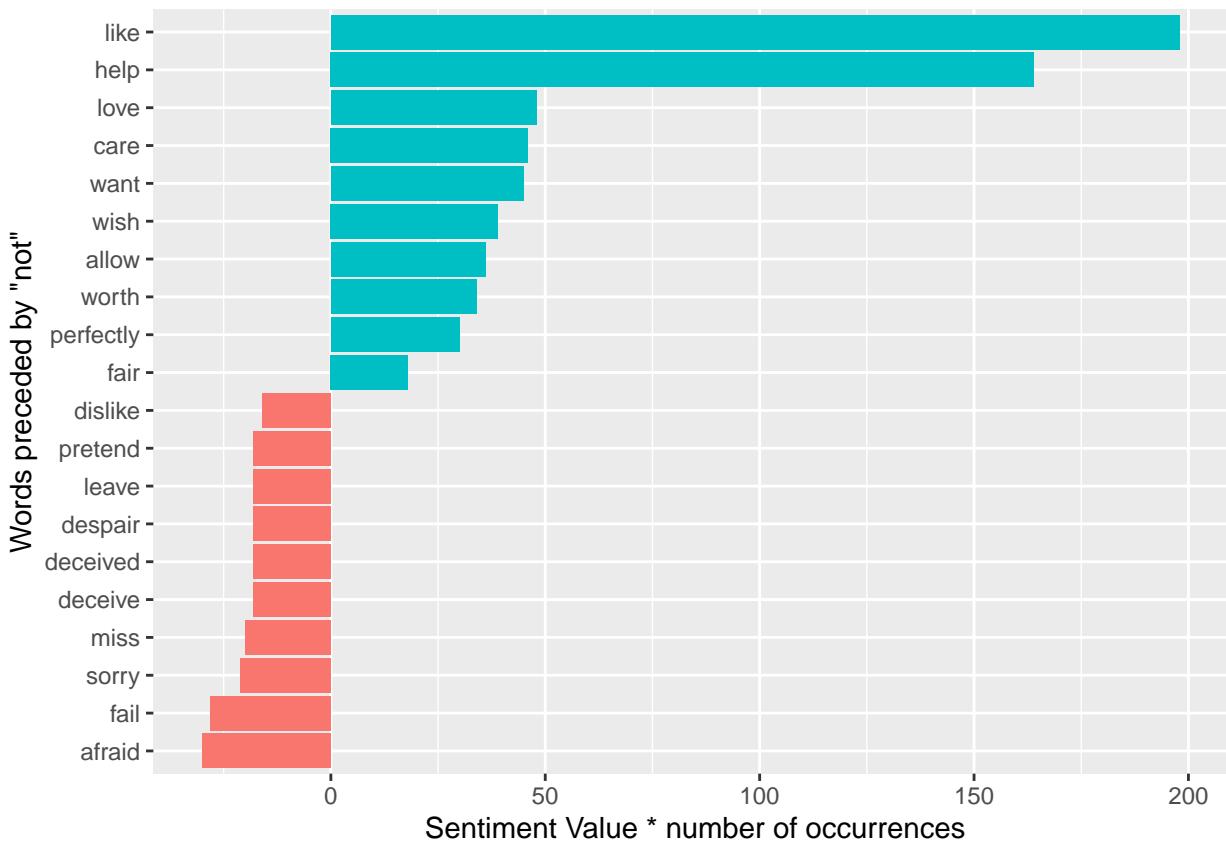
Sebaiknya, kita melakukan observasi sederhana terlebih dahulu terkait kata-kata mana yang memberikan kontribusi ke arah yang tidak tepat. Untuk menghitungnya, kita dapat mengalikan nilai dengan jumlah kemunculan (sehingga kata dengan nilai 3 yang muncul 10 kali memiliki nilai yang sama dengan kata nilai 1 dan muncul 30 kali) Kami akan mencoba memvisualisasikan dengan `geom_bar`.

```

library(ggplot2)

not_words %>%
  mutate(contribution = n * value) %>%
  arrange(desc(abs(contribution))) %>%
  head(20) %>%
  mutate(Word2 = reorder(Word2, contribution)) %>%
  ggplot(aes(Word2, n * value, fill = n * value > 0)) +
  geom_col(show.legend = FALSE) +
  xlab("Words preceded by \"not\"") +
  ylab(" Sentiment Value * number of occurrences") +
  coord_flip()

```



### *Visualizing a Network of bigrams with ggraph*

Kami bisa saja memvisualisasi semua hubungan kata-kata secara bersamaan dan mengaturnya dalam sebuah jaringan atau grafik. Sebelum membuat grafik, terdapat beberapa objek yang wajib dalam sebuah grafik jariangan, yakni, node yang menunjukkan asal, node yang dituju, dan nilai numerik dari setiap sisi.

Paket igraph memiliki beberapa fungsi yang kuat untuk melakukan manipulasi dan menganalisis jaringan. Salah satu fungsi yang mempermudah dalam visualisasi adalah `graph_from_data_frame()`.

```

library(igraph)

# Original count
bigram_count

## # A tibble: 33,421 x 3

```

```

##      Word1    Word2      n
##      <chr>    <chr>    <int>
## 1 sir      thomas     287
## 2 miss     crawford   215
## 3 captain  wentworth  170
## 4 miss     woodhouse  162
## 5 frank    churchill  132
## 6 lady     russell    118
## 7 lady     bertram    114
## 8 sir      walter     113
## 9 miss     fairfax    109
## 10 colonel brandon    108
## # ... with 33,411 more rows
# Filter for only relatively common combinations
bigram_graph <- bigram_count %>%
  filter(n > 20) %>%
  graph_from_data_frame()

bigram_graph

## IGRAPH 9b493e7 DN-- 91 77 --
## + attr: name (v/c), n (e/n)
## + edges from 9b493e7 (vertex names):
## [1] sir      ->thomas    miss     ->crawford  captain  ->wentworth
## [4] miss     ->woodhouse frank    ->churchill lady     ->russell
## [7] lady     ->bertram   sir      ->walter    miss     ->fairfax
## [10] colonel ->brandon   miss     ->bates    lady     ->catherine
## [13] sir      ->john      jane     ->fairfax   miss     ->tilney
## [16] lady     ->middleton miss     ->bingley   thousand ->pounds
## [19] miss     ->dashwood   miss     ->bennet    john     ->knightley
## [22] miss     ->morland   captain ->benwick  dear     ->miss
## + ... omitted several edges

```

Pada dasarnya, `igraph` mempunyai fungsi plot bawaan akan tetapi fungsi tersebut bukan fungsi utama dari paket ini. Dengan demikin, banyak paket lain yang mengembangkan metode visualisasi untuk objek grafik.

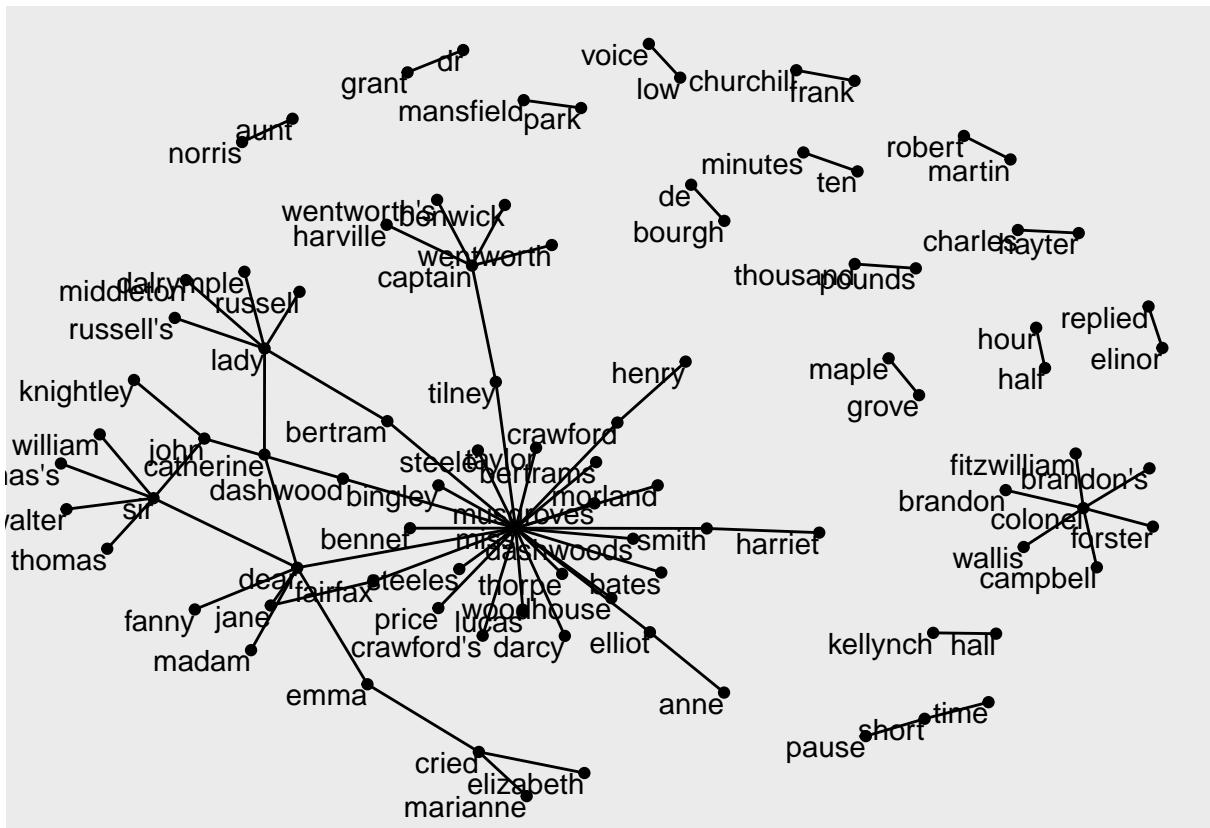
Kita dapat mengubah objek `igraph` menjadi `ggraph` dengan fungsi terdapat pada `ggraph`, setelah itu kita menambahkan layer yang ada layaknya di `ggplot2`.

```

library(ggraph)
set.seed(2017)

ggraph(bigram_graph, layout = "fr") +
  geom_edge_link() +
  geom_node_point() +
  geom_node_text(aes(label = name), vjust = 1, hjust = 1)

```



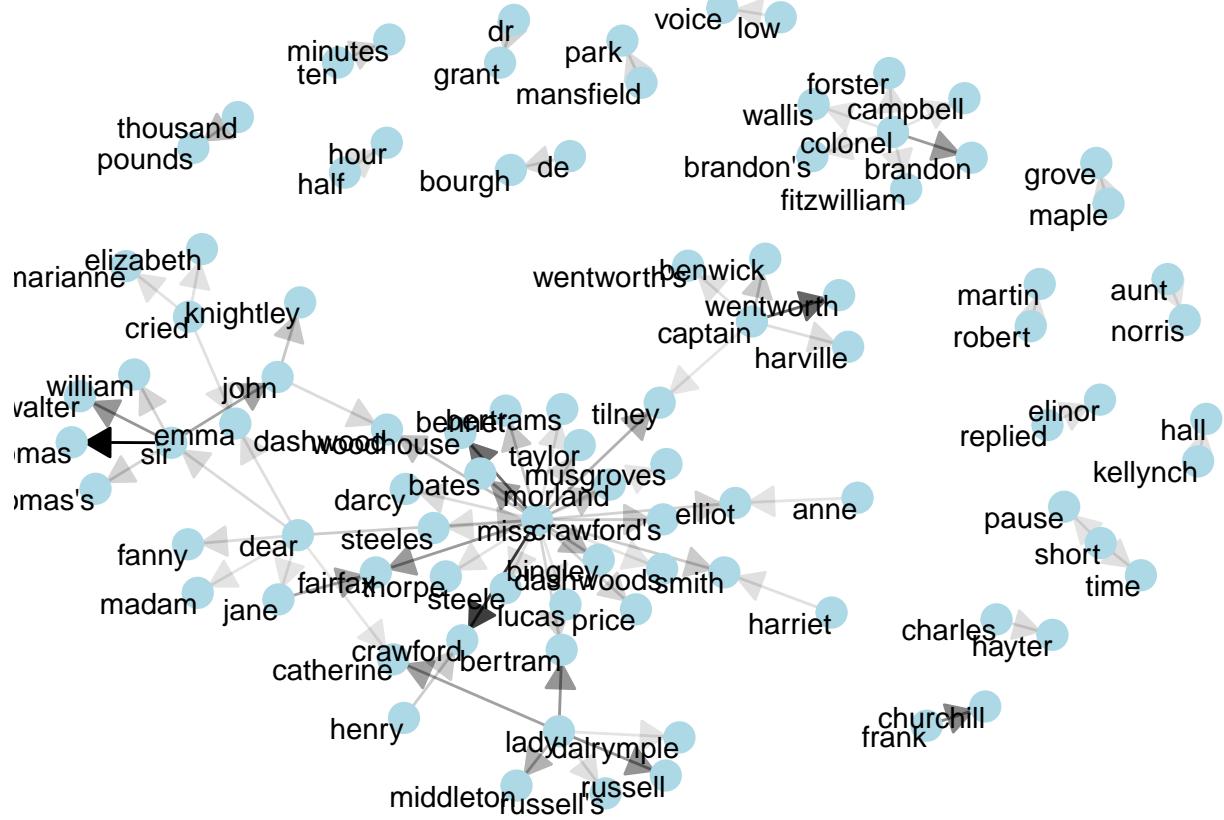
Pada gambar di atas, kita dapat memperhatikan detil struktur teks, seperti “miss”, “lady”, “sir”, “and” dan lain sebagainya. Kita juga dapat melihat pasangan atau kembar tiga di sepanjang bagian luar yang membentuk frase pendek umum.

Pada grafik lebih lanjut, kita akan menambahkan arah untuk mempermudah kita melihat hubungan dari dan menuju kemana sebuah kata. Selain itu, kita juga akan menggunakan tema lain agar visualisasi lebih menarik.

```
set.seed(2016)
```

```
a <- grid::arrow(type = "closed", length = unit(.15, "inches"))

ggraph(bigram_graph, layout = "fr") +
  geom_edge_link(aes(edge_alpha = n), show.legend = FALSE,
                 arrow = a, end_cap = circle(.07, 'inches')) +
  geom_node_point(color = "lightblue", size = 5) +
  geom_node_text(aes(label = name), vjust = 1, hjust = 1) +
  theme_void()
```



Dalam bab ini, Anda telah belajar bagaimana pendekatan teks berguna untuk melakukan analisis data secara individual dan melakukan eksplorasi hubungan antara kata-kata. Visualisasi jaringan ini merupakan alat yang fleksibel untuk kita gunakan untuk melakukan eksplorasi terhadap hubungan dan bermanfaat besar dalam analisis teks.