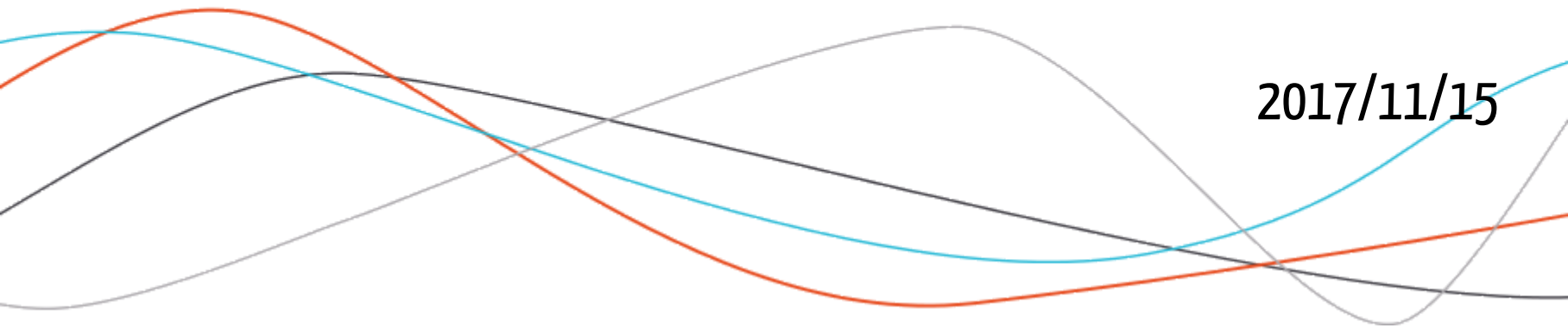
# Visualising text data with ggplot2

Colin FAY - ThinkR

2017/11/15



# $ whoami

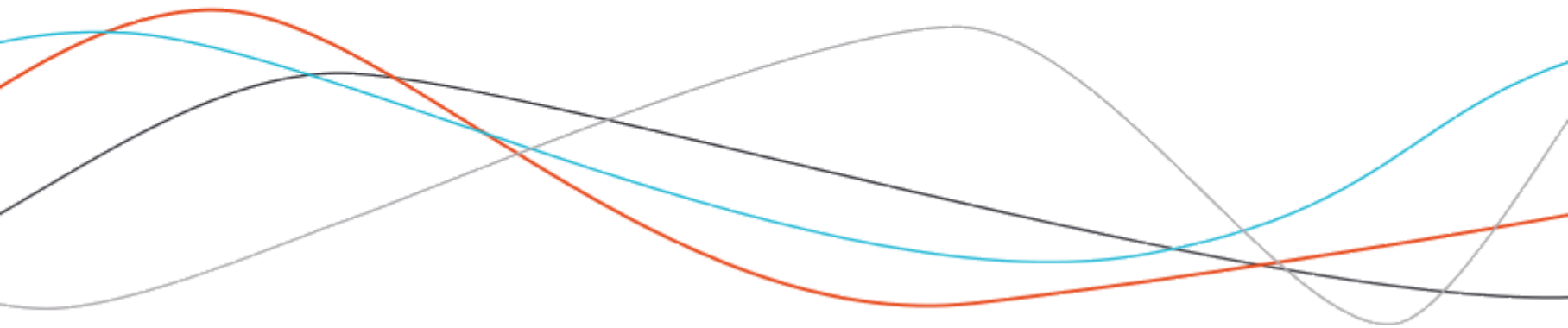
## Colin FAY

I'm a Data Analyst, R trainer and Social Media Expert at ThinkR, a French agency focused on everything R-related.

- http://thinkr.fr
- http://twitter.com/thinkr_fr
- http://twitter.com/_colinfay
- http://github.com/colinfay

# Visualising text data with ggplot2

# But before, a quick poll...



# Who's this guy ?

# {ggplot2}

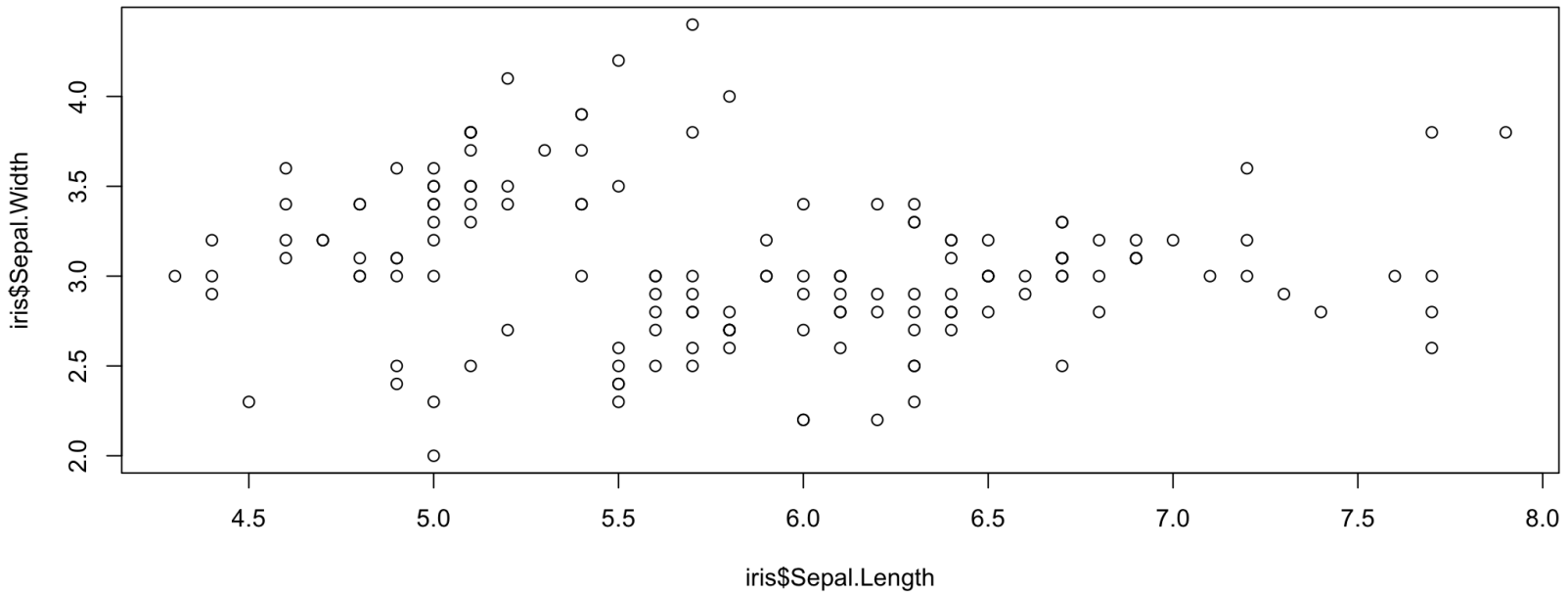
{ggplot2} is a package which has been developped by Hadley Wickham.

It's a plotting system for R that relies on the grammar on graphics and

> "which tries to take the good parts of base and lattice graphics and none of the bad parts. It takes care of many of the fiddly details that make plotting a hassle (like drawing legends) as well as providing a powerful model of graphics that makes it easy to produce complex multi-layered graphics."

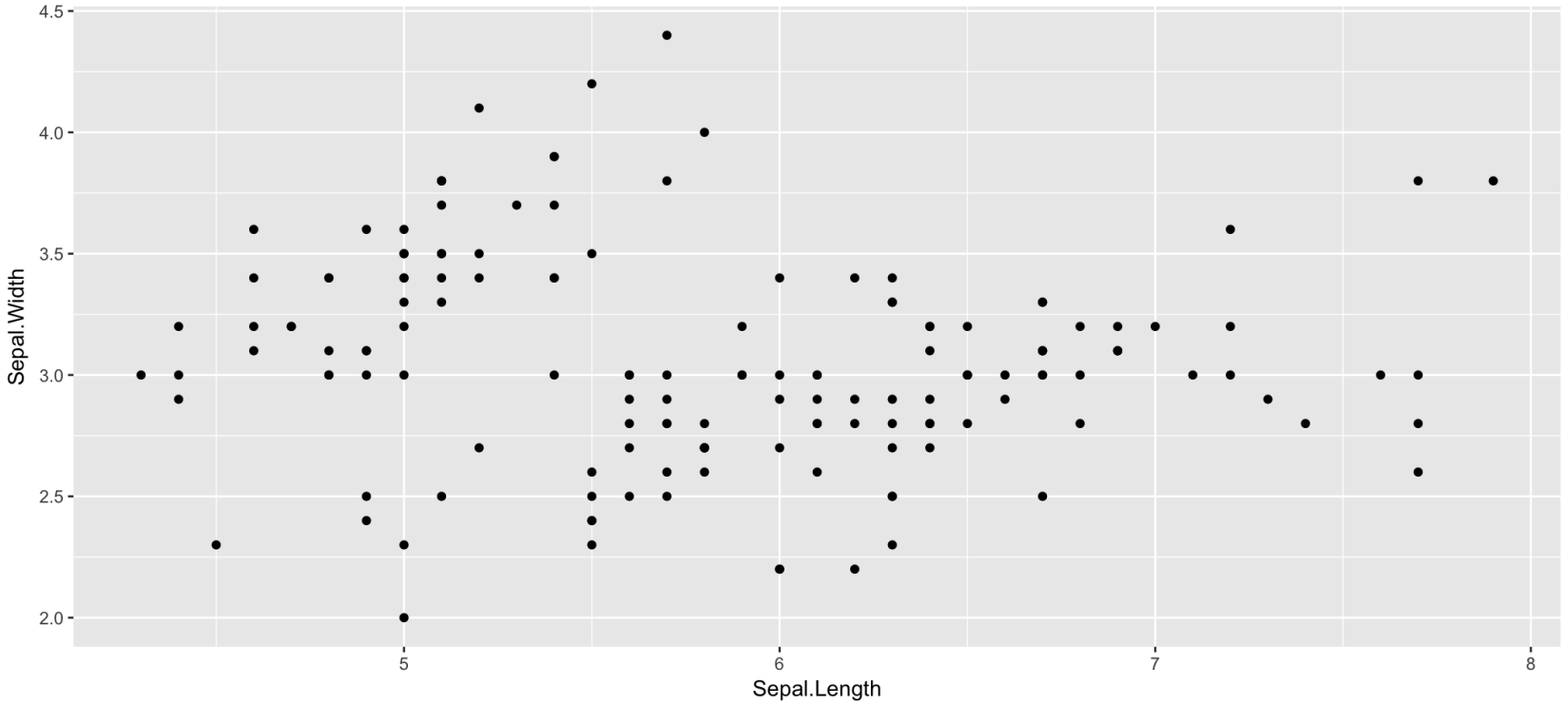
# From {base}...

```
plot(iris$Sepal.Length, iris$Sepal.Width)
```



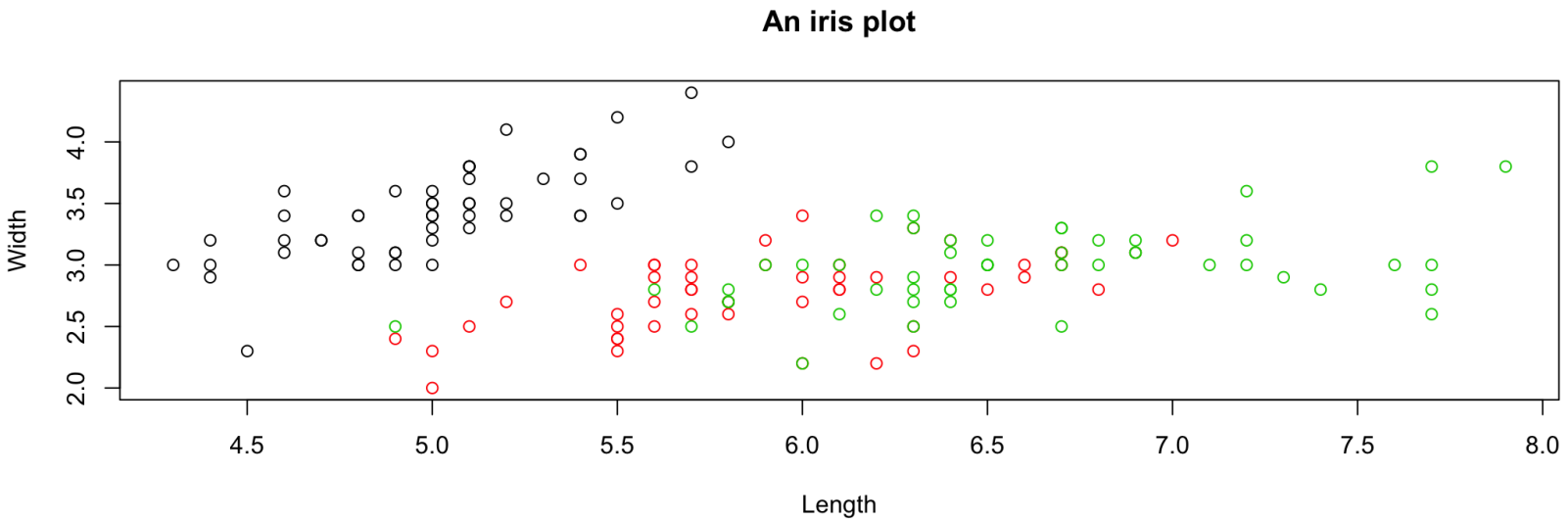
# ...to {ggplot2}

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



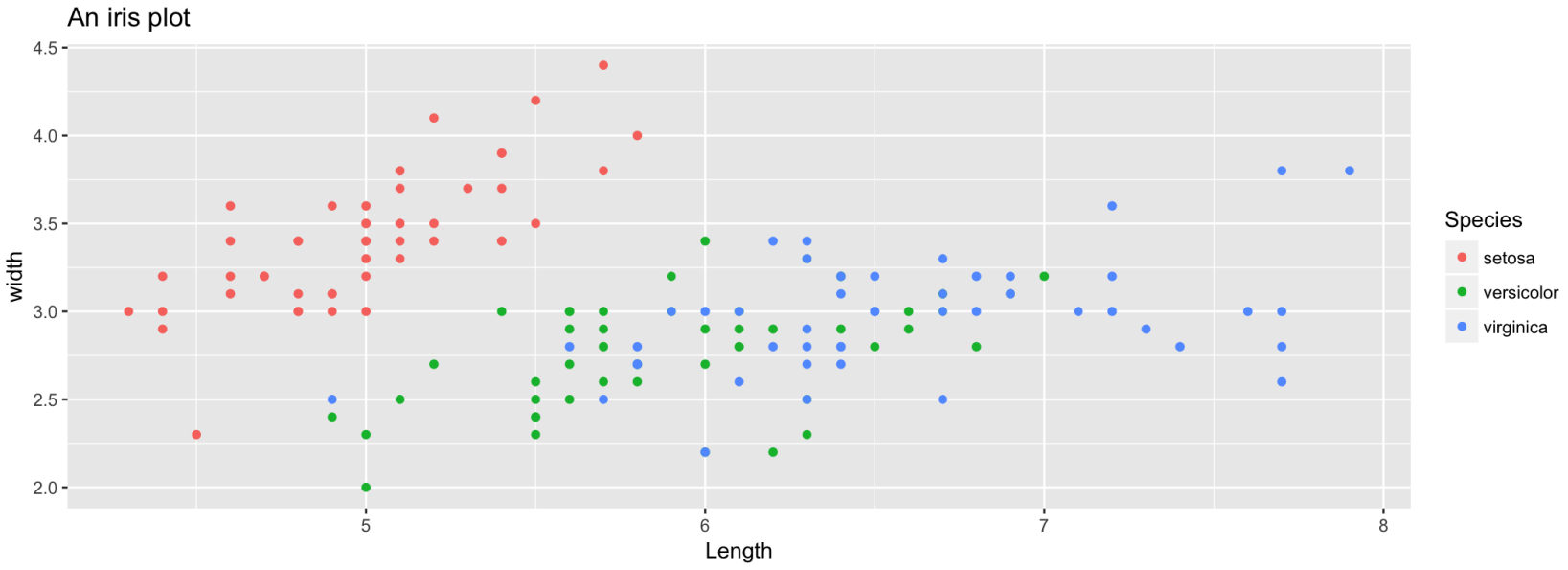
# From {base}...

```
plot(iris$Sepal.Length, iris$Sepal.Width,
     col = iris$Species,
     xlab = "Length", ylab = "Width", main = "An iris plot")
```



# ...to {ggplot2}

```
ggplot(iris) +
  aes(Sepal.Length, Sepal.Width, color = Species) +
  geom_point() +
  labs(x = "Length", y = "width", title = "An iris plot")
```

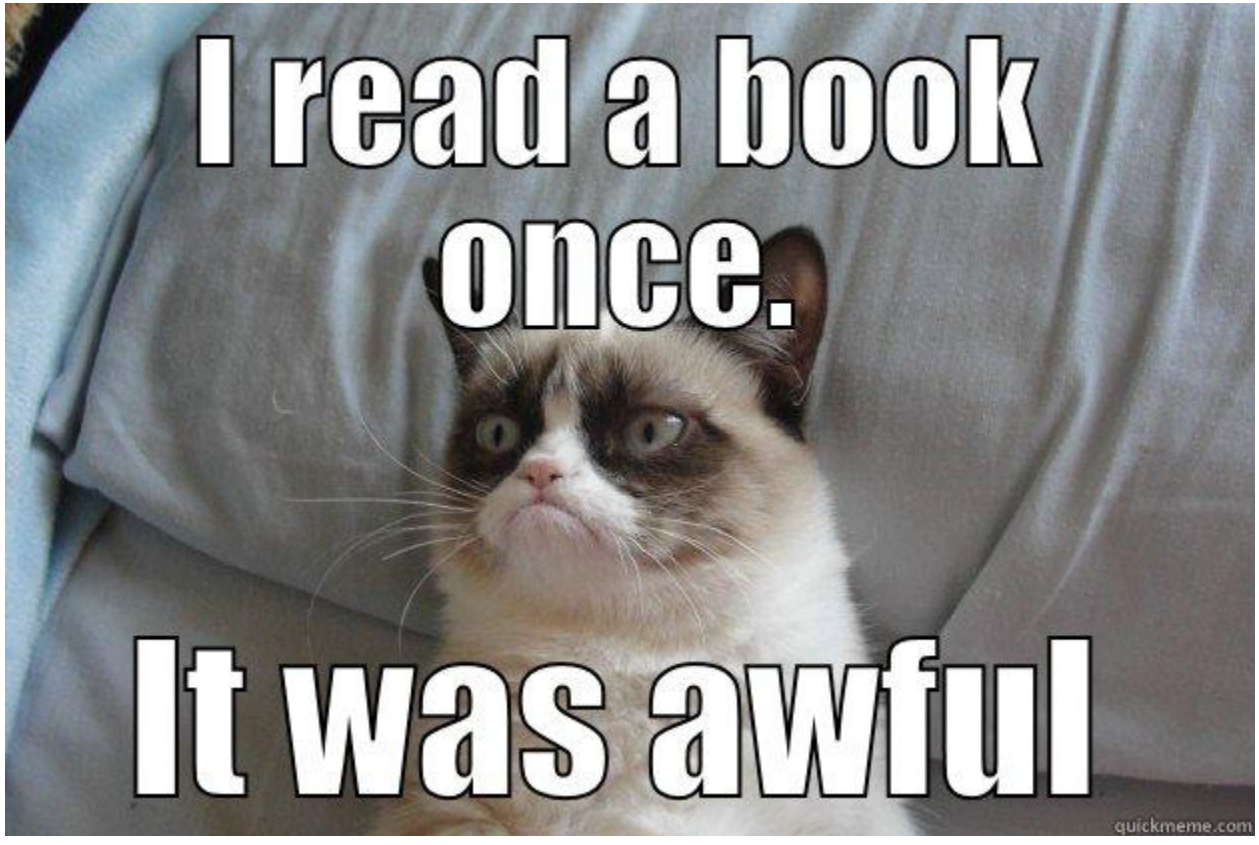


# Building a ggplot

With {ggplot2}, you're building your plot "layer by layer", adding them with a + :

- data
- aesthetics
- geometries: geom_XXX
- facets: facet_XXX (ex : `facet_grid()`, `facet_wrap()`)
- statistics: stat_XXX (ex: `stat_smooth()`...)
- coordinates: scale_XXX, coord_XXX
- theme : theme_ (ex: `theme_mininal()`)

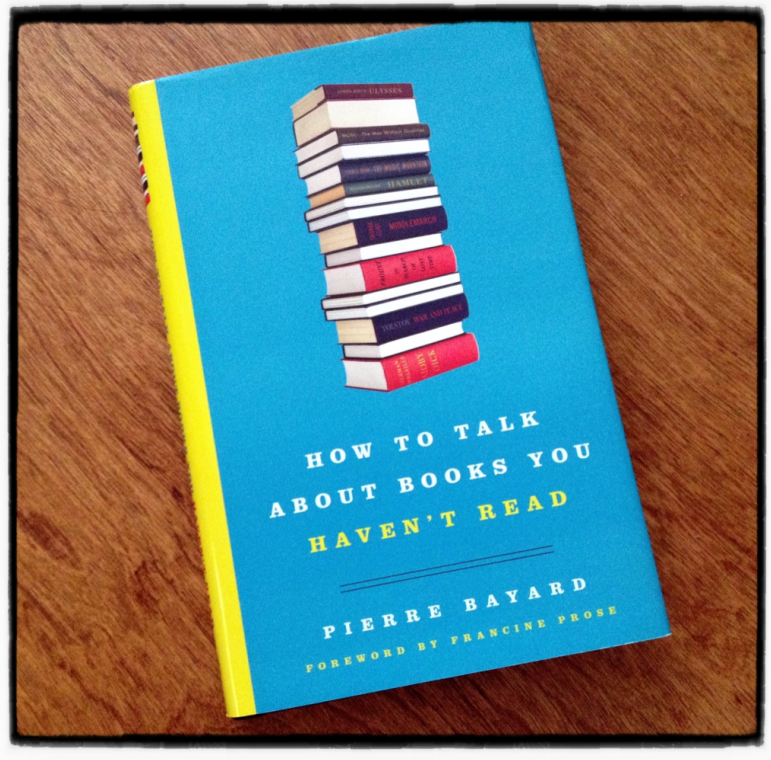
# Visualising textual data with {ggplot2}



# Visualising textual data with {ggplot2}

Why?

- You don't have as many time as you'd wish to read Proust.
- There's so many tweets and so few hours in a day.
- Who still reads customers reviews in 2017?
- You need to get a quick insight into a corpus without spending time looking into it.



# Preparing a dataset

The biggest headache when you need to visualise your text data is the data preparation (i.e turning a raw text into a data.frame).

The good news is that if you plan is to use {ggplot2} to visualise your text data, the tools are already there.

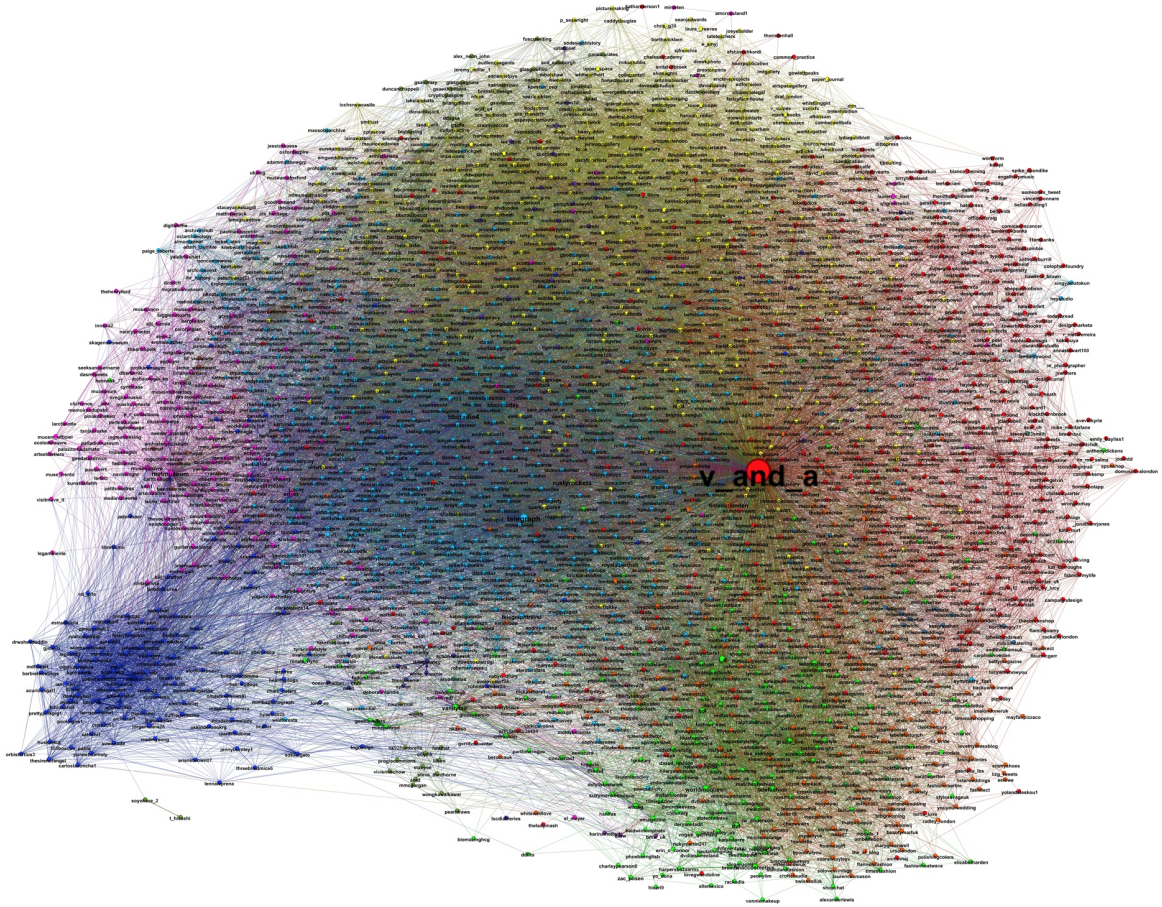
```
library(dplyr) # for data munging
library(tidytext) # for tidy text preparation
library(proustr) # a package for NLP in French, provides some additionnal functions for stemming
```

# First rule of text visualisation

~~Don't talk about text visualisation~~

Don't try to show **everything** in one plot.

# Can we read that?



# Using {ggplot2} and tidy tools to focus on the essential

When you're doing text visualisation, what you want is getting key insights.

Like:

- What are the most common n-grams used?
- How are words related to each other?
- What are the main sentiments we can find in our dataset?
- Can we spot patterns in our corpus?

# From raw text to {tidytext}

```
# Search made today at 12.00 sharp
tweets <- rtweet::search_tweets("#RStats", n = 2000, include_rts = FALSE)
```

We have a corpus of 1981 with the hashtag #RStats (without RT).

Let's face it, we can read them all.

```
glimpse(tweets)
```

```
#> Observations: 1,981
#> Variables: 35
#> $ screen_name                  <chr> "DavidZumbach", "dalejbarr", "n...
#> $ user_id                      <chr> "3143396517", "191232431", "318...
#> $ created_at                   <dttm> 2017-11-15 10:54:41, 2017-11-1...
#> $ status_id                    <chr> "930750897243787264", "93074662...
#> $ text                         <chr> "That moment when you realize y...
#> $ retweet_count                <int> 0, 0, 0, 1, 0, 1, 2, 0, 0, 1, 1...
#> $ favorite_count               <int> 0, 0, 0, 0, 1, 1, 6, 0, 0, 5, 3...
#> $ is_quote_status              <lgl> FALSE, FALSE, FALSE, TRUE, FALS...
#> $ quote_status_id              <chr> NA, NA, NA, "930684293583724545...
#> $ is_retweet                   <lgl> FALSE, FALSE, FALSE, FALSE, FAL...
#> $ retweet_status_id            <chr> NA, NA, NA, NA, NA, NA, NA, NA,...
```

# Turning into a one-token-per-row format

{ggplot2}, as all the packages in the tidyverse, needs a dataframe to work with. We've already got a df with our tweets, but we need to turn the `text` column into a *one-token-per-row* format.

We'll do this with the {tidytext} package.

```
otpr <- tweets %>%
  unnest_tokens(output = word, input = text)
select(otpr, screen_name, word) %>% slice(1:5)
```

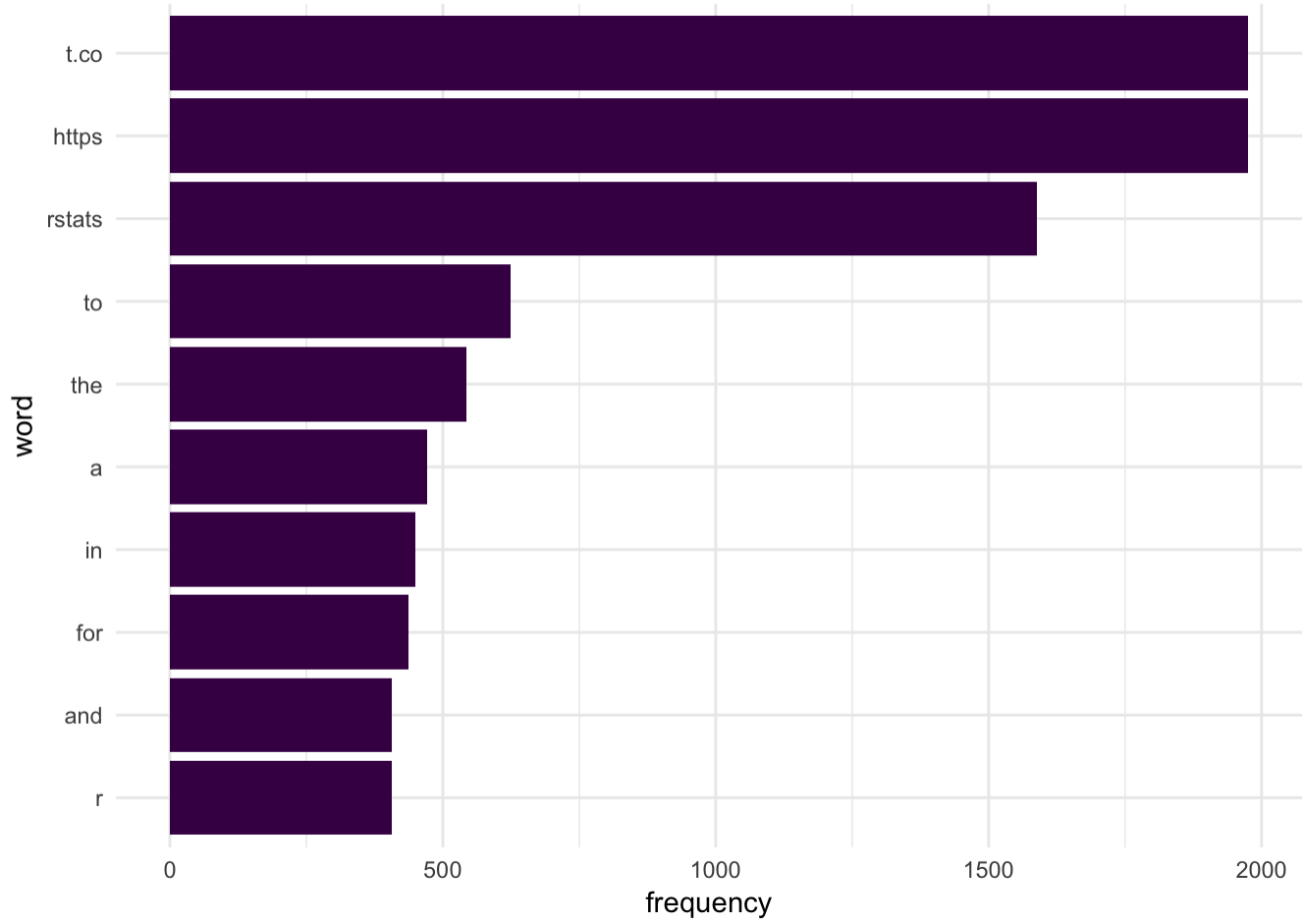
```
#> # A tibble: 5 x 2
#>    screen_name    word
#>          <chr>   <chr>
#> 1 DavidZumbach    that
#> 2 DavidZumbach  moment
#> 3 DavidZumbach    when
#> 4 DavidZumbach     you
#> 5 DavidZumbach realize
```

# Plotting the most common words

Let's try without any filter:

```
otpr %>%
  count(word) %>%
  top_n(10, n) %>%
  ggplot() +
  aes(reorder(word, n), n) +
  geom_col(fill = viridis::viridis(10)[1]) +
  coord_flip() +
  labs(x = "word",
       y = "frequency") +
  theme_minimal()
```

# Plotting the most common words

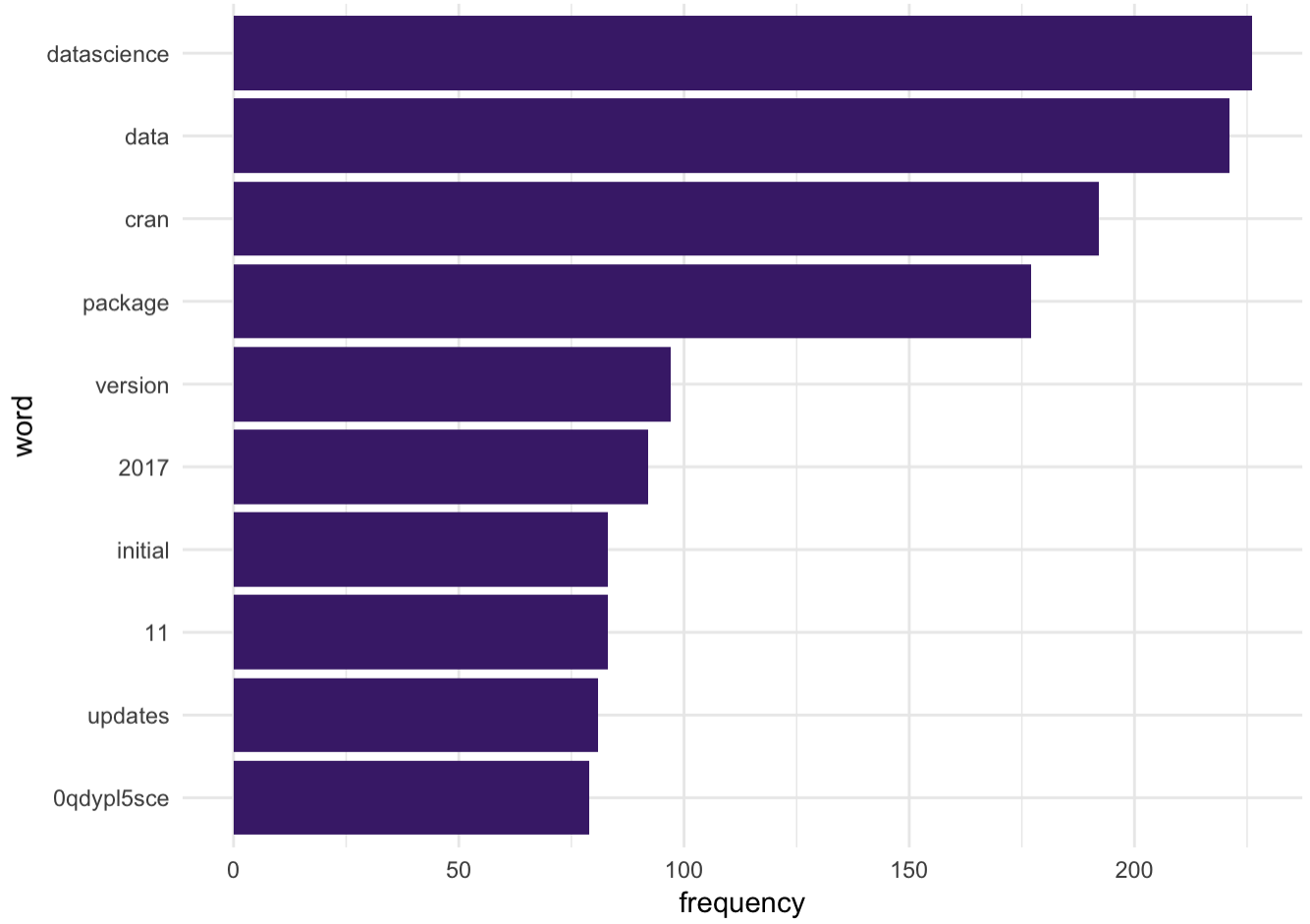


# Removing stopwords

{tidytext} has a dataframe of english stopwords you can anti_join to your dataframe. If you're working with french, you can also use the {proustr} package.

```
otpr %>%
  count(word) %>%
  # Dataset of stopwords from tidytext
  anti_join(stop_words) %>%
  # Filter on custom stopwords
  filter(! word %in% c("amp","https", "t.co", "rstats") ) %>%
  top_n(10, n) %>%
  ggplot() +
  aes(reorder(word, n), n) +
  geom_col(fill = viridis::viridis(1)) +
  coord_flip() +
  labs(x = "word",
       y = "frequency") +
  theme_minimal()
```

# Removing stopwords



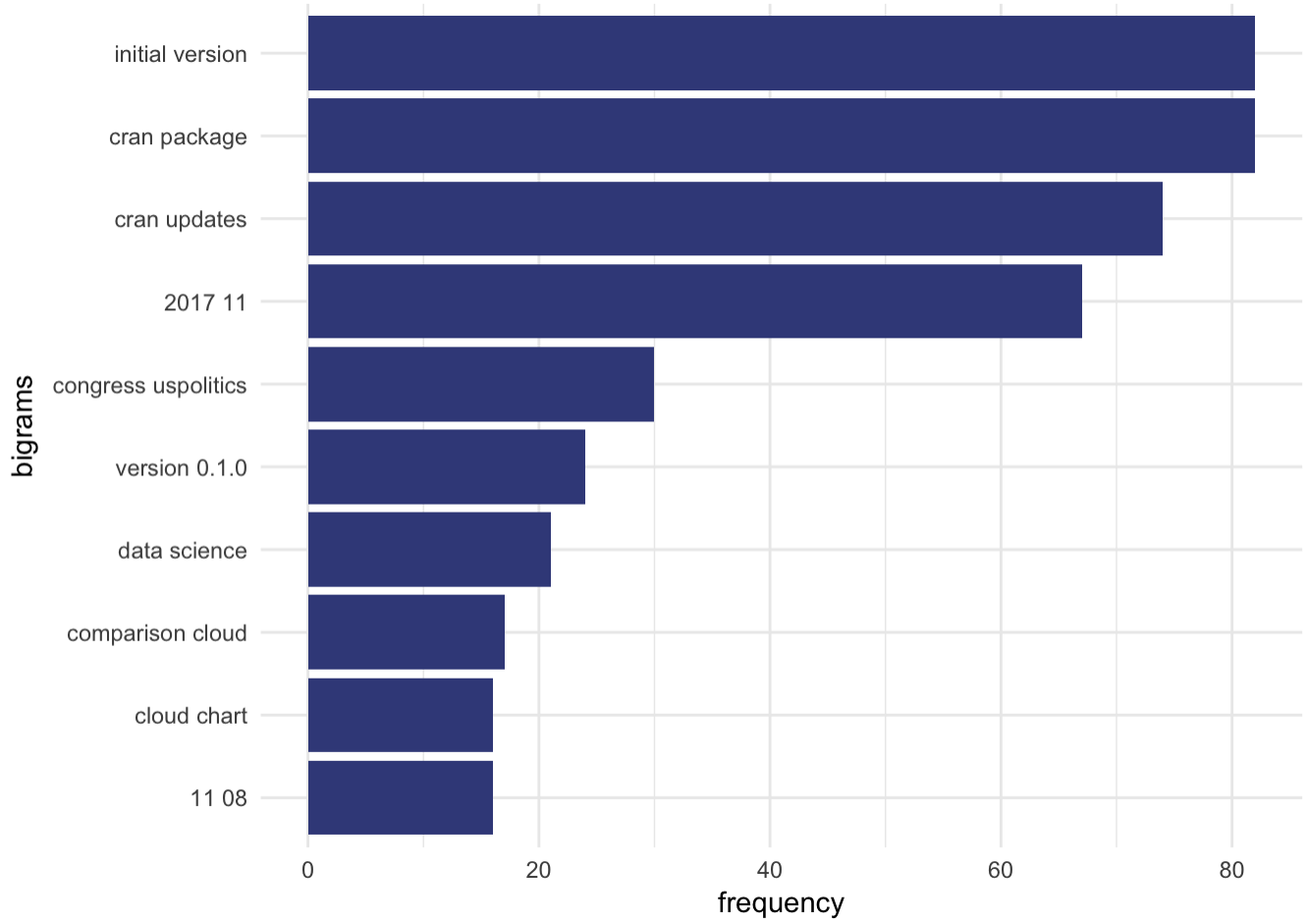
# Do the same with bigrams

```
library(tidyr)

# Creating a custom stop words list
custom_sw_list <- c(stop_words$word, c("amp","https", "t.co", "rstats"))

bigrams <- tweets %>%
  unnest_tokens(bigrams, text, token = "ngrams", n = 2) %>%
  separate(bigrams, into = c("word1", "word2"), sep = " ") %>%
  filter(! word1 %in% custom_sw_list) %>%
  filter(! word2 %in% custom_sw_list) %>%
  unite(bigrams, word1, word2, sep = " ")
```

# Plotting the most common bigrams

```
bigrams %>%
  count(bigrams) %>%
  top_n(10, n) %>%
  ggplot() +
  aes(reorder(bigrams, n), n) +
  geom_col(fill = viridis::viridis(10)[3]) +
  coord_flip() +
  labs(x = "bigrams",
       y = "frequency") +
  theme_minimal()
```

# Plotting the most common bigrams



# Basic sentiment analysis

You can do basic sentiment analysis with the `get_sentiments()` function from {tidytext}, and a simple `inner_join()` from {dplyr}:

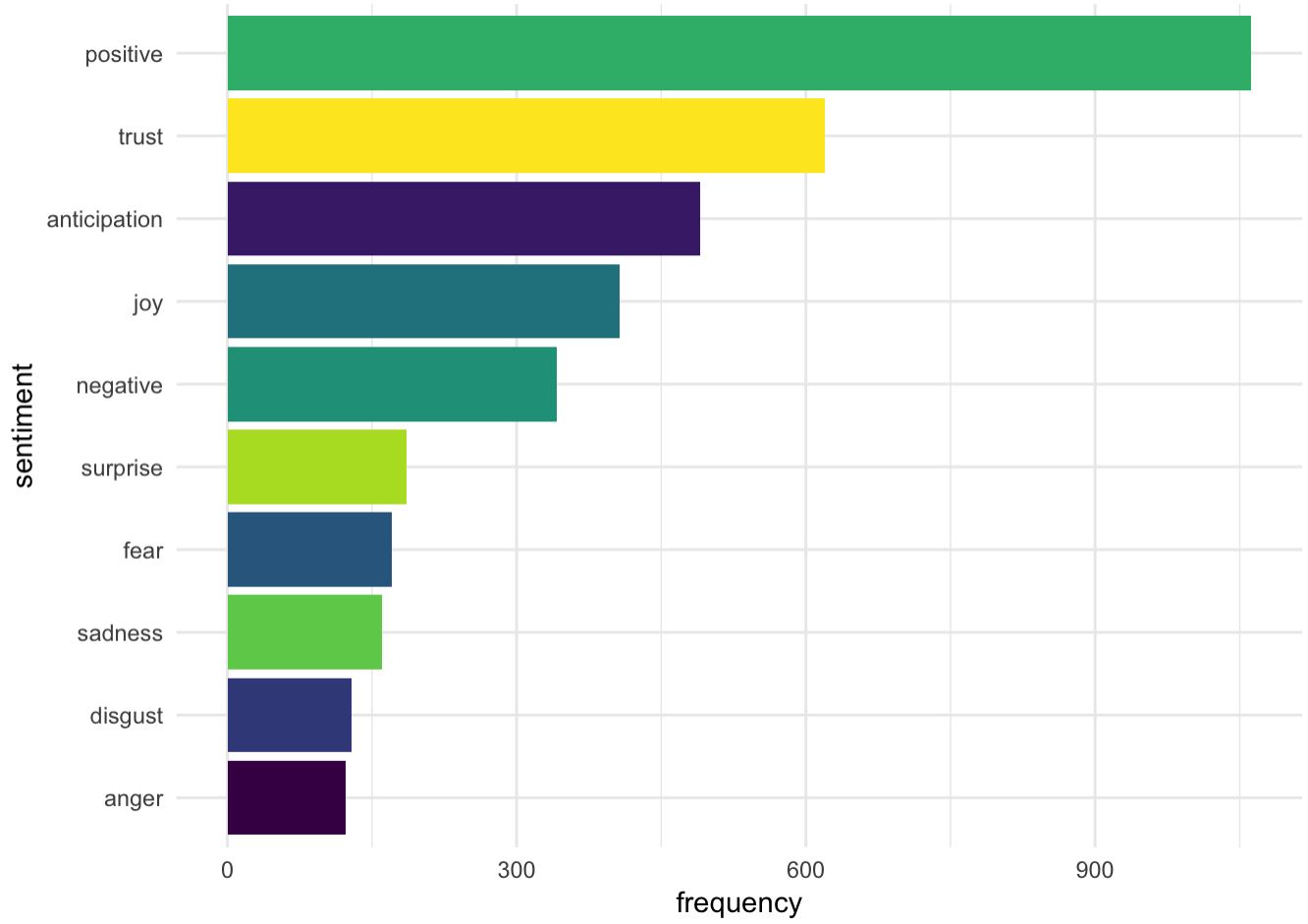
```
get_sentiments("nrc")
```

```
#> # A tibble: 13,901 x 2
#>           word sentiment
#>          <chr>     <chr>
#>  1      abacus     trust
#>  2     abandon      fear
#>  3     abandon  negative
#>  4     abandon   sadness
#>  5   abandoned     anger
#>  6   abandoned      fear
#>  7   abandoned  negative
#>  8   abandoned   sadness
#>  9 abandonment     anger
#> 10 abandonment      fear
#> # ... with 13,891 more rows
```

# Basic sentiment analysis

```
otpr %>%
  inner_join(get_sentiments("nrc")) %>%
  count(sentiment) %>%
  ggplot() +
  aes(reorder(sentiment, n), n, fill = sentiment) +
  geom_col() +
  coord_flip() +
  scale_fill_viridis_d() +
  theme_minimal() +
  labs(x = "sentiment",
       y = "frequency") +
  theme(legend.position = 'none')
```

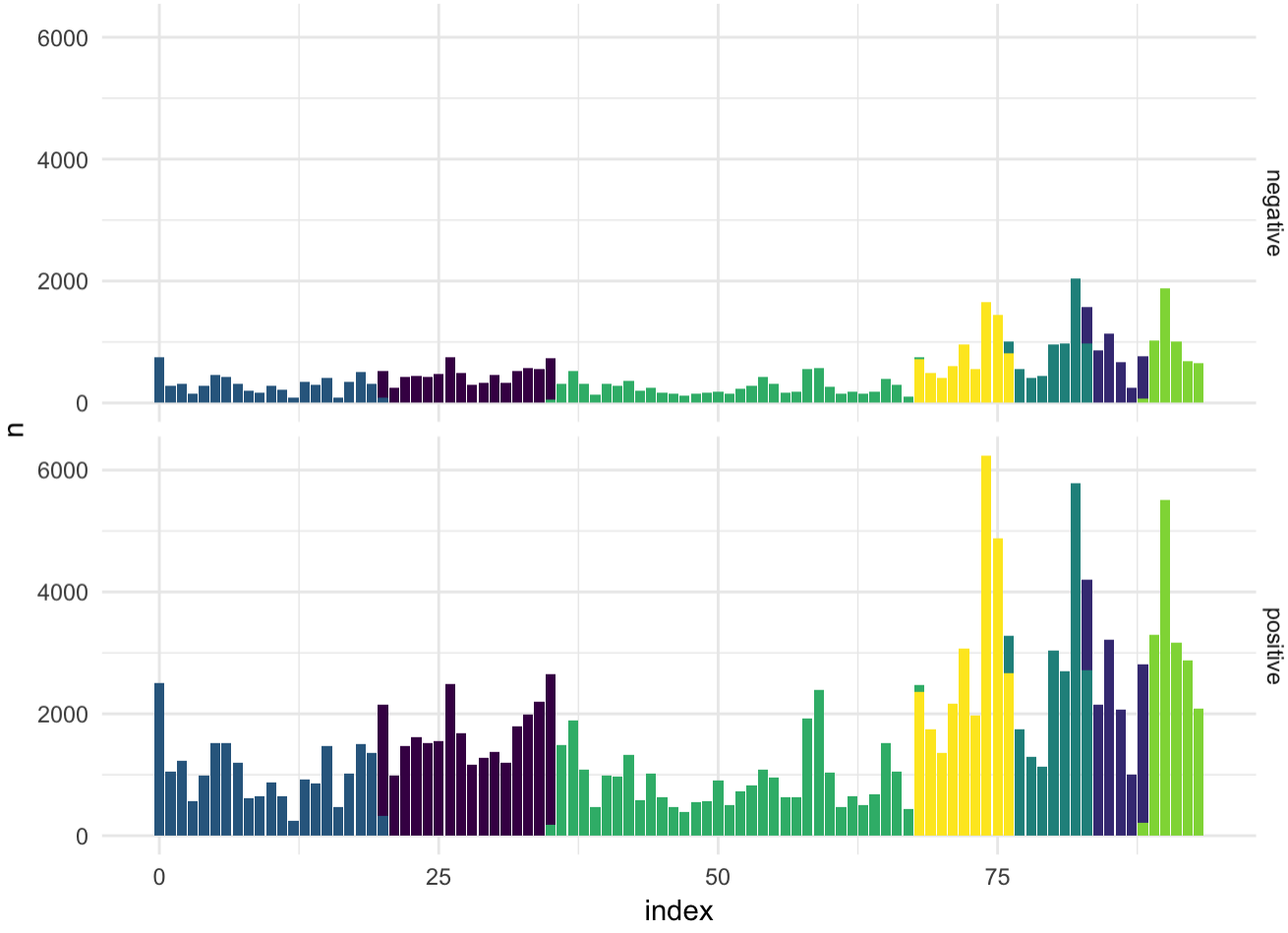
# Basic sentiment analysis



# Temporal sentiment analysis

```
proust_books() %>%
  tibble::rownames_to_column() %>%
  mutate(index = as.numeric(rowname) %/% 50) %>%
  unnest_tokens(word, text) %>%
  inner_join(proust_sentiments("polarity")) %>%
  count(index, book, polarity) %>%
  ggplot() +
  aes(index, n, fill = book) +
  geom_col() +
  facet_grid(polarity ~ .) +
  scale_fill_viridis_d() +
  theme_minimal() +
  theme(legend.position = 'none')
```

# Temporal sentiment analysis



# Making things to ggraph

## About {ggraph}

This package has been designed to work flawflessly with {ggplot2}.

If you're already familiar with the {ggplot2} grammar, you'll be able to render {ggraph} plots in a matter of minutes.

If you want to know more about ggraph, come to my talk tomorrow ;)

# Back to our Twitter data

Let's map with a function who's talking about what in our twitter corpus.

```
# First, a corpus
gr_tweets <- tweets %>%
  unnest_tokens(word, text) %>%
  select(screen_name, word) %>%
  filter(! word %in% custom_sw_list) %>%
  pr_stem_words(word, language = "english")
```

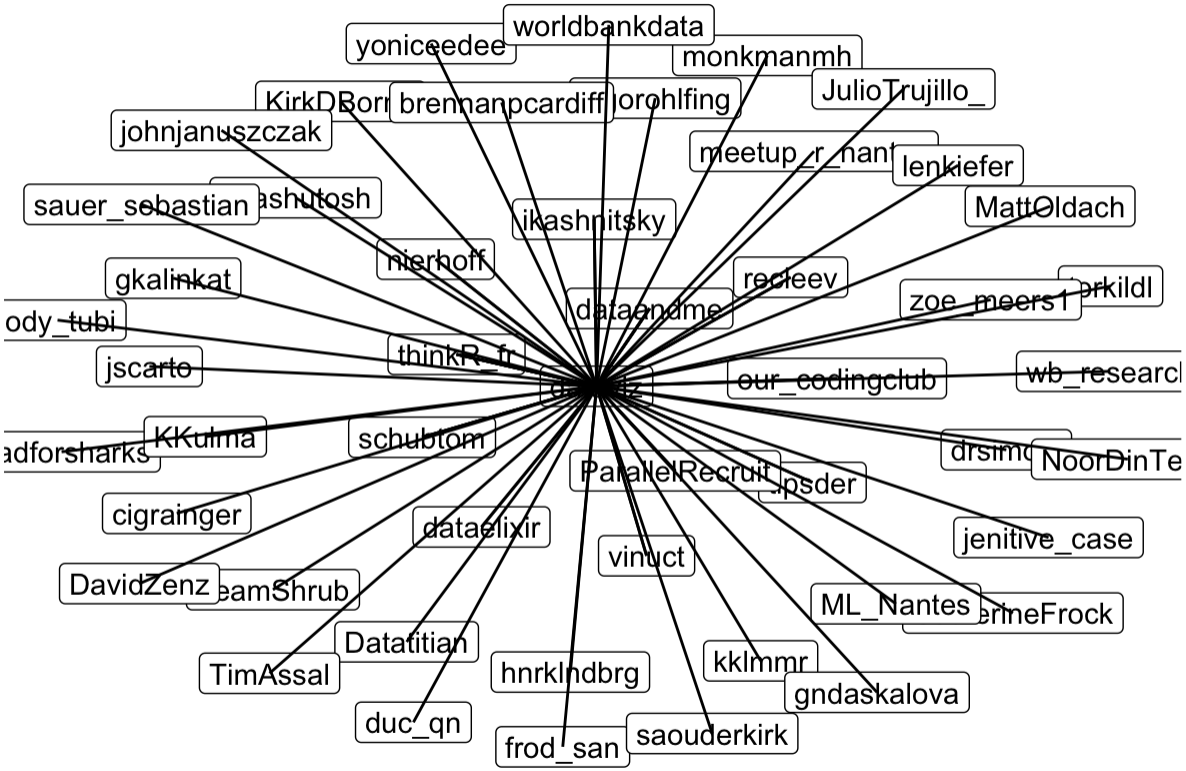
# Back to our Twitter data

Let's map with a function who's talking about what in our twitter corpus.

```
#Then the function
library(ggraph)
who_says_that <- function(what){
  filter(gr_tweets, word == what) %>%
    graph_from_data_frame() %>%
    ggraph() +
    geom_node_label(aes(label = name)) +
    geom_edge_link() +
    theme_graph()
}
```

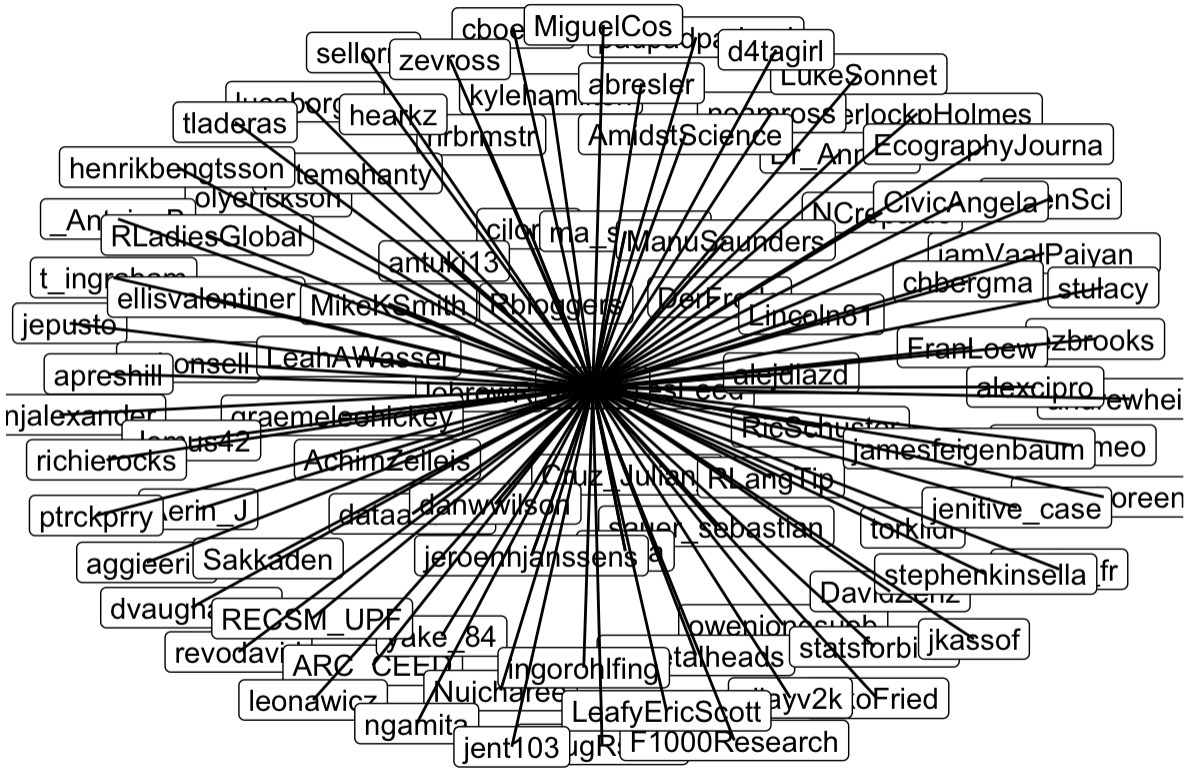
# Run the function

```
who_says_that("dataviz")
```



# Run the function

```
who_says_that("packag")
```



# Shiny App

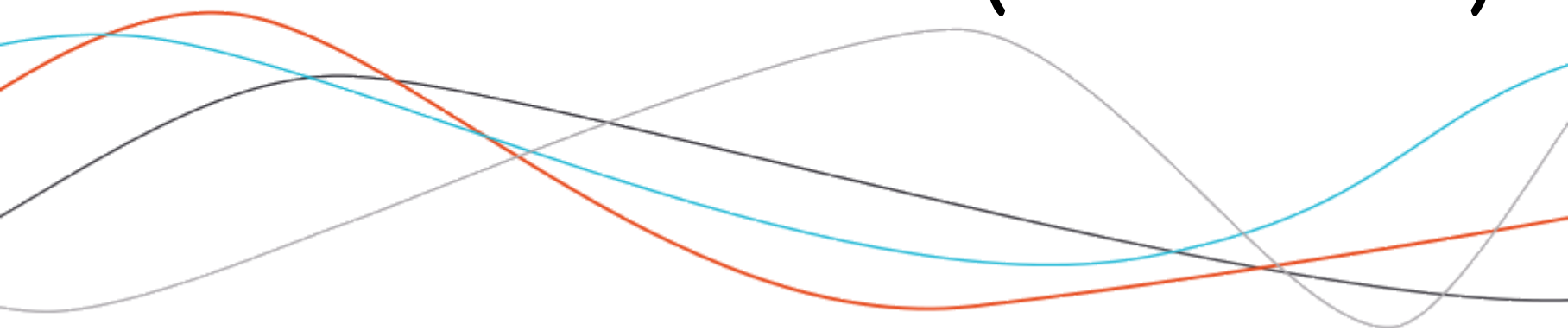
```
library(shiny)
ui <- fluidPage(
  titlePanel("Who says what on #RStats"),
  sidebarLayout(
    sidebarPanel(
      selectInput(inputId = "word", multiple = TRUE,
                  label = "Word to filter on",
                  choices = gr_tweets$word,
                  selected = "packag")
    ),
    mainPanel(
      h3("Network"), plotOutput("network"),
      h3("Who says that?"), DT::dataTableOutput("table")
    )
  )
)
```

# Shiny App

```
server <- function(input, output) {
   output$network <- renderPlot({
     who_says_that(input$word)
   })
   output$table <- DT::renderDataTable({
     DT::datatable(filter(gr_tweets, word == input$word))
   })
}
shinyApp(ui = ui, server = server)
```

# Quick demo

## (If I have time...)



# Thanks !

## Any questions ?

**Find me in the web:**

- colin@thinkr.fr
- http://twitter.com/_colinfay
- http://twitter.com/thinkr_fr
- https://github.com/ColinFay

**And also:**

- https://thinkr.fr/
- http://colinfay.me/