MaRtin's R CouRse 5th edition.

Fall 2019: An introduction to R programming and text mining.

Dr. Martín Lozano.
martin.lozano@udem.edu

octubre 03, 2019. 13:25:27

Abstract

This is the course material for the fifth edition of the R course by MaRtin, designed for UDEM students. In case I edit this course material (extensions, new sections, amended codes and even corrections for English grammar) then the most updated version will be located at https://github.com/mlozanoqf/CouRse5ed. This material is mostly adapted, extended and/or taken from past editions of this course; https://www.tidytextmining.com/; the Summer School on Data Science for Studying Science, Technology and Innovation 2019 (Strathclyde Business School, Glasgow, United Kingdom); Juan Bosco Mendoza Vega work; and many other freely available Internet resources (see the last section of this document).

Contents

---

1 Introduction.

Welcome to the the fifth edition of MaRtin's R course. Every semester I try to incorporate a different approach and new topics you can learn in R. This semester I am pleased to introduce text mining. Text mining is a relatively new area that has been experiencing an interesting development recently thanks to the new available technology to implement statistical tools to the analysis of big amount of data in text format. The main objective of text mining is to extract information from big and sometimes unstructured data to understand its contents, classify, compare and evaluate intrinsic characteristics such as sentiments, writing styles, and contents that could not be easily identified by the human eye and we make them evident with text mining tools.

In the area of business, there are many kind of applications of text mining we can exploit. For some of these applications the evidence is still growing because this is still considered as a new field, and there are also some new and unexplored applications. For example, it is common to analyze speeches of politicians to better understand his or her views about economics or about any other specific relevant topic for the voter. Central banks also publish information in form of text to support their decisions about monetary policy that can be compared to identify regime changes or to formally find out the main concerns of the policymakers. People have also been interested to analyze successful investors interviews to try to extract the hidden secrets of a good investment. The social media like Facebook, Twitter and recently WhatsApp are also sources of text data that can be accessed with R to analyze relationships between tweets and changes in financial instrument prices (to mention just an example). We also have rich information in the firm's annual reports, CEO and CFO declarations to the media, and many other sources of text data that we could incorporate in quantitative analysis to better understand economics and finance.

The objective of this introductory course is to explain the basics of computer programming in R and an introduction to text mining by using diverse examples from different fields that we can all understand. For example, finance, economics, English literature, physics texts comparisons, an analysis of news articles from a well known American news agency, and finally a tweets analysis of Mexican presidential candidates. Hopefully, you can take this examples to analyze other kind of relevant data-bases in your work and even in your PEF.

1.1 Why R?

You may know that just a few years ago, those economic agents who had privileged access to information had a clear comparative advantage in business and overall decision taking. Today, information and data are pretty much available to everyone thanks to technology, and consequently the possibility to achieve an advantage simply due to information access has vanished or at least it is currently vanishing at a high speed.

Now that data availability per se is no longer a comparative advantage in business, knowledge has become a critical aspect in business. Nowadays, it is not about having access to information; it is about knowing what to do with it to create value in business. Manipulating and transforming data into business tools and decisions has become a required skill for business professionals.

There are many ways in which you can learn basic data science, or how to transform information into valuable business intelligence, but I consider learning to code is one of them. This is because coding allows you to train your brain to think more efficiently and more productively to solve complex problems and come up with innovative solutions. In this class, you will learn how to use R, a free (yes, absolutely free) computer language which allows branching and looping as well as doing modular programming using functions. Nowadays, the Finance job market is increasingly demanding for candidates with some knowledge in the field of data science or computational finance, mainly because this boost creative problem-solving skills and data analysis abilities. The free (open source) R programming language is considered as one of the most important tools for companies such as Google, Facebook, The New York Times, Twitter, Pfizer, Merck, Bank of America, and LinkedIn among others.

According to my view, English is the leading language for doing research and business; math and statistics are the languages used to understand nature; and R is one of the most popular languages to communicate directly to a computer. Given that computers are and will undoubtedly be part of our lives, we better learn how to communicate with them not only as a plain and boring user level but as a programmer level. As young professionals, it is important to differentiate yourselves with the rest of your colleagues and be prepared for the changing job market conditions in the area of finance. My view is that you are expected to be as proficient as possible in these three ways to interact with our environment regardless of your own professional expertise: English, math, and coding.

During your undergraduate studies, you will be expected to learn a bunch of good commercials (and unfortunately expensive) software such as Microsoft Excel, SPSS, STATA, E-Views and many others. I truly encourage you to learn them as good as you can; however, you will have to be aware that these programs are fully controlled by private firms, so there is no guarantee that their associated file formats could be readable in the future, or even exist in the future.

Commercial software products as the ones listed above are important in the job market, but you also must realize that the main interaction with these programs is by using the mouse to click on pre-defined, limited and sometimes inflexible menus. This kind of user-interaction is most of the times ephemeral and unrecorded, so that many of the choices made during a full quantitative procedure are frequently undocumented and this turns out to be highly problematic because there is no trace about how an analysis was conducted, and also because it becomes hard to propose an extension to the analysis in phases or replication in different contexts. Learning how to code is equivalent as writing a cooking recipe and every time you click run you get the dish done. Although, chefs must pay for ovens, kitchen items and even ingredients, while in finance most of our inputs are free data and the technology is also free as R is an open source software. So, by learning how to code, you can share, expand, reproduce and innovate by your own to the point of producing original empirical results that are important inputs for research outputs as your own PEF.

Other commercial products like Microsoft Excel, SPSS, STATA, E-Views and many others, have high licensing fees and rely on mysterious black boxes. These black boxes are problematic because the data comes in and the result comes out as magic, showing no details about the procedure followed to produce the results, and the user could sadly get the wrong illusion that he or she understands statistics. This might be convenient in some specific and limited cases but in others you miss the fun that represents having access to all the details of the computation and limit the extent to which you can customize or extend to innovative and create new improved applications. The general alternative to using a point-and-click program is to familiarize with languages like R which allows writing scripts to program algorithms for financial analysis and visualizations.

My courses here at UDEM openly incorporates data science and specifically computational finance with R to implement and learn the main subject. R is only one computer language, there are many and very good. We are going to use R because I consider it is still the best choice for the area of business. Also, by learning one language you are actually learning others without noticing. The barriers to getting started with R are currently very low because there are plenty of resources available in Internet, you only have to allocate the right amount of time and effort to it.

1.2 About this document: Rmarkdown.

This document has been made (or compiled) with R Markdown. This mean that the original source is a "Rmd" document produced in R Studio.

The process of producing scientific documents has two parts: the content and the format, and we care about both of them. The content depends fully on the author or researcher; and the format depends on the software or tool we choose to write the scientific documents. Traditionally, we have two types of software, those who rely on the principle of "what you see is what you get" or WYSIWYG, and those based on LaTeX code, which when compiled will produce a document. The most well known WYSIWYG text is MS-Word processor. And probably one of the most flexible and powerful platforms to produce LaTeX documents (and more) is R markdown. An additional feature is that R Studio is a free and open-source integrated development environment (IDE) for R, whereas most of the WYSIWYG alternatives are expensive and we have to rely on one private enterprise to get updates.

1.3 My teaching philosophy.

In 100 words:

The time and the brain are resources and constrains as both restrict the amount of time available to learn. We understand this at the university, so we use these resources efficiently. Here, we stimulate our mental capacity to create and innovate by doing research; and we teach how to think and learn in a systematic and scientific manner. Here, we promote the most fundamental human values to build a prosperous society. I love teaching almost as much as doing research because every semester represents a new opportunity to further enhance our lives by learning the most given our time constraints.

More specifically, with respect to R, my view is that you can learn as long a you practice. Courses like this are useful because you can get familiar with the computational environment, main functions, recommendations, sources and references to learn more, but the true is that you cannot learn R in one course. You have to practice. The good news is that the data you need to practice is most of the times free, the technology is your own computer, and the software is free, so you only have to spend time and effort.

2 Computer programming.

Computer programming is the process of writing instructions that get executed by computers. The instructions, also known as code, are written in a programming language which the computer can understand and use to perform a task or solve a problem.

Operations as in a calculator.

```
5+5
```

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

```
13-8
```

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

```
1/7
```

```
## [1] 0.1428571
```

```
9*8
```

```
## [1] 72
```

```
exp(5)
```

```
## [1] 148.4132
```

```r
log(2)
```

```
## [1] 0.6931472
```

```r
1/0
```

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

```r
5*5*5*5*5
```

```
## [1] 3125
```

Store numerical values in one variable.

```r
a=5+5
b=13-8
c=exp(5)
d=1/0
a
```

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

```r
b
```

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

```r
c
```

```
## [1] 148.4132
```

```r
d
```

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

We can use previously created variables to conduct new operations.

```r
e=a^2+b
e
```

```
## [1] 105
```

```r
a+b
```

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

```r
c/d
```

```
## [1] 0
```

Variables can represent single numerical value, vector, matrix.

```r
(r=seq(0,1,0.1))
```

```
##  [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
```

```r
matrix(1,2,3)
```

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

Variables can represent even functions.

```r
VP = function(VF,r,Tiempo){
  VP = VF/(1+r)^Tiempo
```

```
  VP
}
```

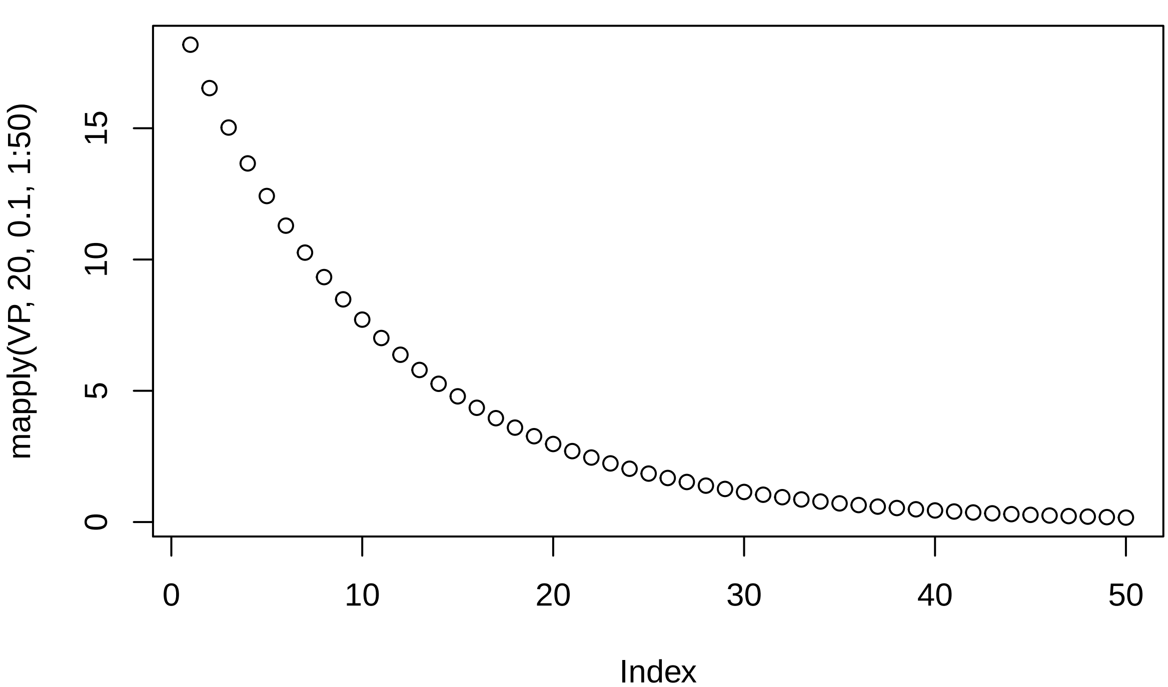
And we can evaluate functions quite easily.

```r
mapply(VP,20,0.1,1:50)
```

```
##  [1] 18.1818182 16.5289256 15.0262960 13.6602691 12.4184265 11.2894786
##  [7] 10.2631624  9.3301476  8.4819524  7.7108658  7.0098780  6.3726164
## [13]  5.7932876  5.2666251  4.7878410  4.3525827  3.9568934  3.5971758
## [19]  3.2701598  2.9728726  2.7026114  2.4569195  2.2335632  2.0305120
## [25]  1.8459200  1.6781091  1.5255537  1.3868670  1.2607882  1.1461711
## [31]  1.0419737  0.9472488  0.8611353  0.7828503  0.7116821  0.6469837
## [37]  0.5881670  0.5346973  0.4860884  0.4418986  0.4017260  0.3652054
## [43]  0.3320049  0.3018227  0.2743842  0.2494402  0.2267638  0.2061489
## [49]  0.1874081  0.1703710
```

Even plot it.

```r
plot(mapply(VP,20,0.1,1:50))
```



Nice.

3 Economic data and basic econometrics.

We can download US data directly from FRED (Federal Reserve Economic Data) and then manipulate it. In this case, we download unemployment rate and consumer confidence.

```r
library(alfred)
```

```
## Warning: package 'alfred' was built under R version 3.5.3
```

```r
startdate="1980-01-01"
enddate="2019-10-01"
dfunrate=get_fred_series("UNRATE","unrate",
                         observation_start=startdate,
                         observation_end=enddate)
```

```
## Warning: package 'bindrcpp' was built under R version 3.5.2
```

```r
dfumcsent=get_fred_series("UMCSENT","umcsent",
                          observation_start=startdate,
                          observation_end = enddate)
dfall=cbind(dfunrate,dfumcsent)
```

Let's view the data.

```r
head(dfall)
```

```
##         date unrate       date umcsent
## 1 1980-01-01    6.3 1980-01-01    67.0
## 2 1980-02-01    6.3 1980-02-01    66.9
## 3 1980-03-01    6.3 1980-03-01    56.5
## 4 1980-04-01    6.9 1980-04-01    52.7
## 5 1980-05-01    7.5 1980-05-01    51.7
## 6 1980-06-01    7.6 1980-06-01    58.7
```

```r
tail(dfall)
```

```
##           date unrate       date umcsent
## 471 2019-03-01    3.8 2019-03-01    98.4
## 472 2019-04-01    3.6 2019-04-01    97.2
## 473 2019-05-01    3.6 2019-05-01   100.0
## 474 2019-06-01    3.7 2019-06-01    98.2
## 475 2019-07-01    3.7 2019-07-01    98.4
## 476 2019-08-01    3.7 2019-08-01    89.8
```

Let's clean the data because the date appears twice in our database.

```r
dfall=dfall[,c(1,2,4)]
head(dfall)
```

```
##         date unrate umcsent
## 1 1980-01-01    6.3    67.0
## 2 1980-02-01    6.3    66.9
## 3 1980-03-01    6.3    56.5
## 4 1980-04-01    6.9    52.7
## 5 1980-05-01    7.5    51.7
## 6 1980-06-01    7.6    58.7
```

How many observations?

```r
count=nrow(dfall)
count
```

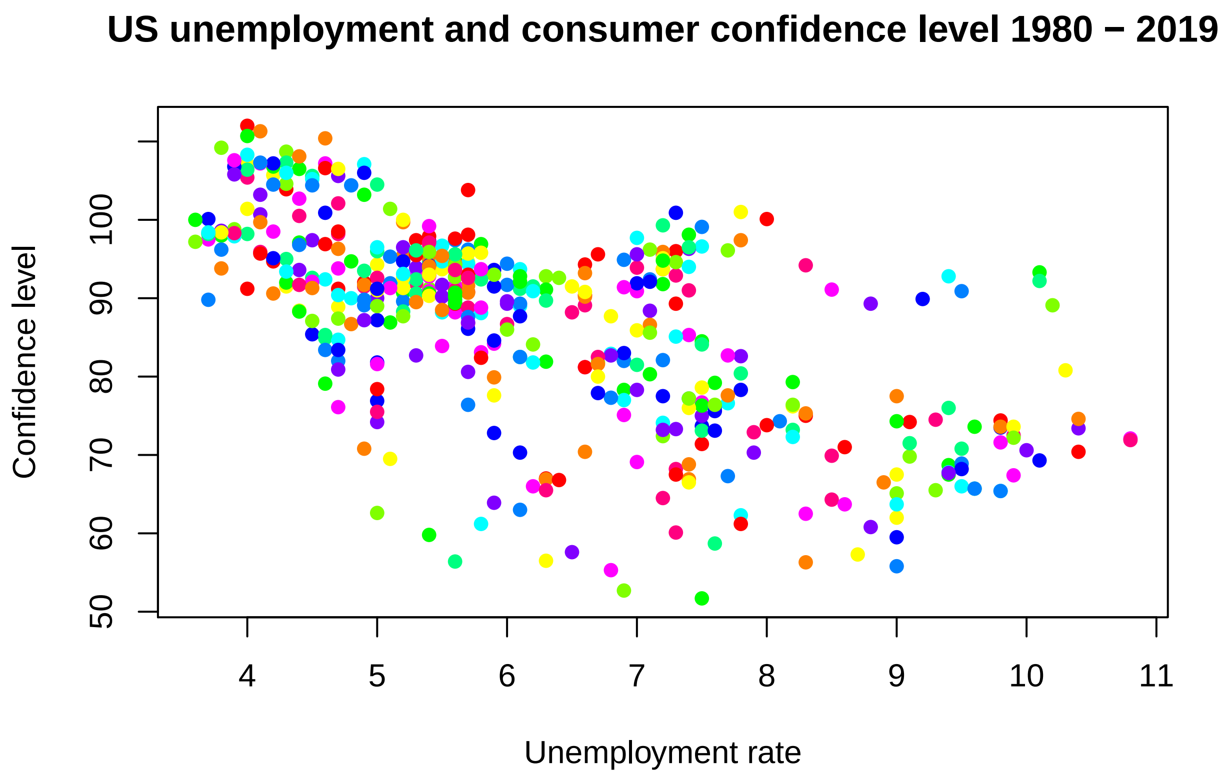
```
## [1] 476
```

We tell R that the date column represent a date.

```r
dfall$date=as.Date(dfall$date)
tail(dfall$date)
```

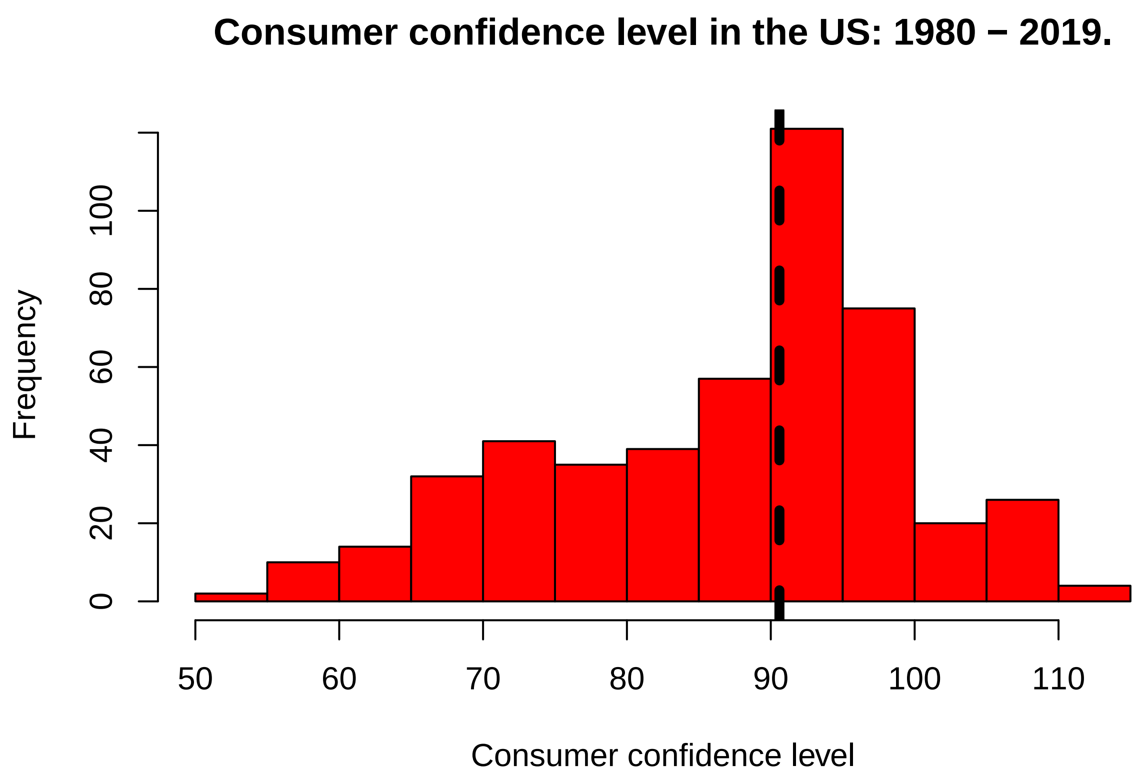
```
## [1] "2019-03-01" "2019-04-01" "2019-05-01" "2019-06-01" "2019-07-01"
## [6] "2019-08-01"
```

```r
plot(dfall$unrate,dfall$umcsent,main = "US unemployment and consumer confidence level 1980 - 2019",xlab=
     ylab="Confidence level",col=rainbow(12),cex=1,pch=16)
```

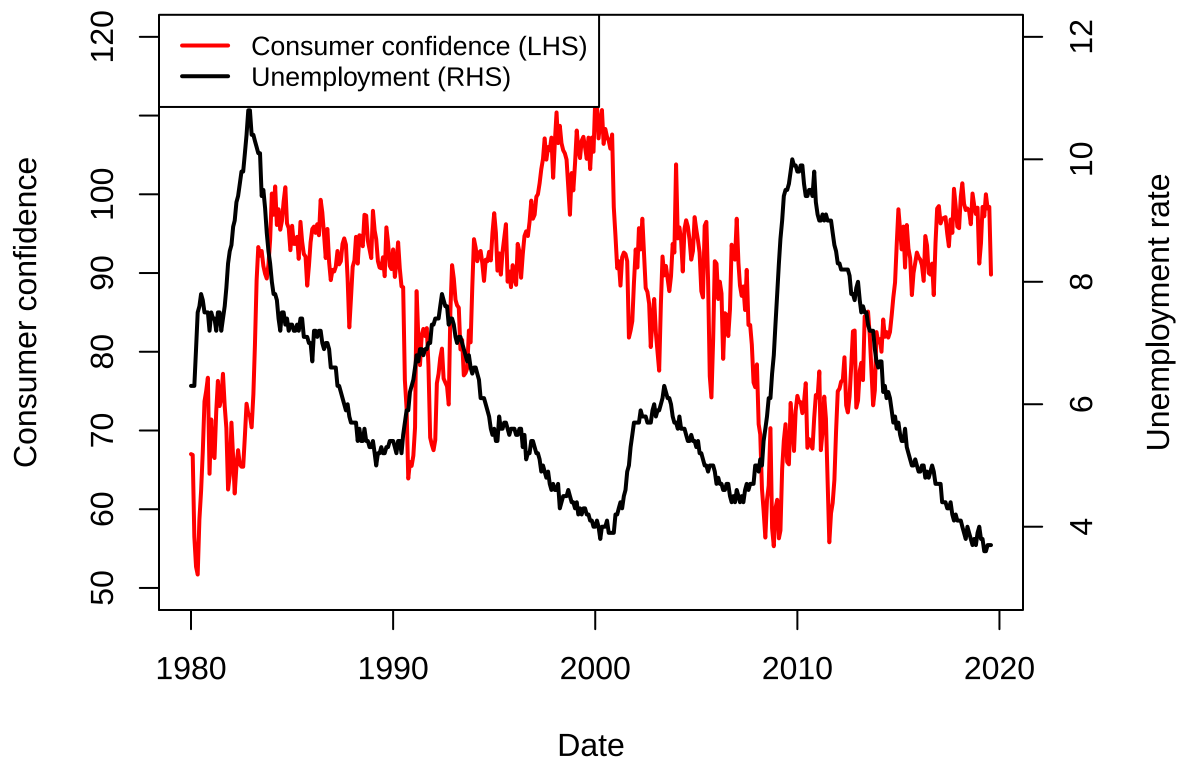



```r
hist(dfall$unrate,50,col=rainbow(5),main = "US unemployment rate histogram: 1980 - 2019.",xlab="Unemplo
     ylab = "Frequency")
abline(v=median(dfall$unrate),lwd=5,lty=2)
```


```
hist(dfall$umcsent,20,col="red",
     main = "Consumer confidence level in the US: 1980 - 2019.",
     xlab="Consumer confidence level", ylab = "Frequency")
abline(v=median(dfall$umcsent),lwd=5,lty=2)
```




```
par(mar = c(5,5,2,5))
with(dfall, plot(dfall$date,dfall$umcsent,type="l",col="red",
             ylab="Consumer confidence",ylim=c(50,120),
             lwd=2,xlab="Date"))
par(new=T)
with(dfall, plot(dfall$date,dfall$unrate,axes=F,xlab=NA,
                 ylab=NA,type="l",ylim=c(3,12),lwd=2))
axis(side=4)
mtext(side=4,line=3,"Unemployment rate")
legend("topleft",
       legend=c("Consumer confidence (LHS)", "Unemployment (RHS)"),
       lwd=c(2,2),col=c("red","black"),bg="white",cex=0.8)
```

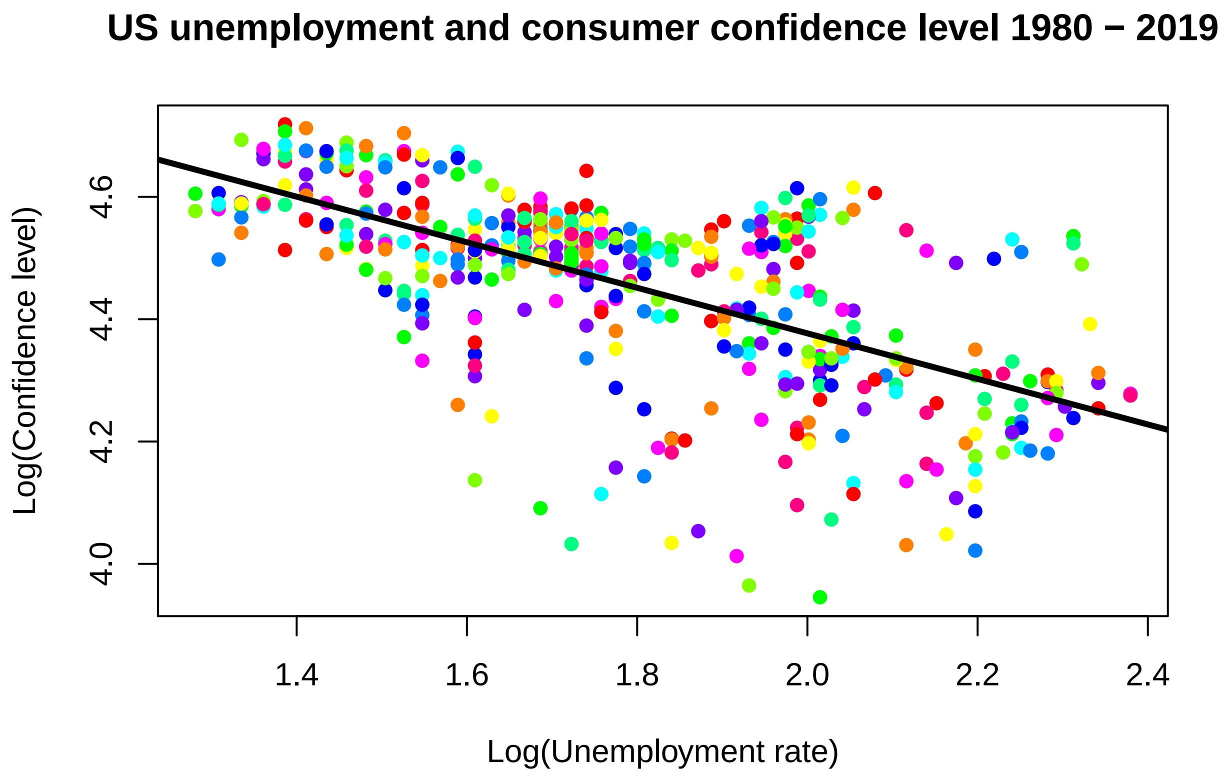


Regression analysis.

```r
dffit=lm(log(umcsent) ~ log(unrate),data=dfall)
summary(dffit)
```

```
##
## Call:
## lm(formula = log(umcsent) ~ log(unrate), data = dfall)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.44758 -0.05079  0.01068  0.06971  0.27536
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  5.12151    0.03737  137.04   <2e-16 ***
## log(unrate) -0.37234    0.02062  -18.06   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1177 on 474 degrees of freedom
## Multiple R-squared:  0.4075, Adjusted R-squared:  0.4063
## F-statistic:   326 on 1 and 474 DF,  p-value: < 2.2e-16
```

```r
plot(log(dfall$unrate),log(dfall$umcsent),main = "US unemployment and consumer confidence level 1980 - 2
     ylab="Log(Confidence level)",col=rainbow(12),cex=1,pch=16)
abline(dffit,lwd=3)
```



4 Financial data and basic econometrics.

We download data from Internet.

```r
library(tidyquant)
```

```
## Warning: package 'tidyquant' was built under R version 3.5.2

## Loading required package: lubridate

## Warning: package 'lubridate' was built under R version 3.5.2

##
## Attaching package: 'lubridate'

## The following object is masked from 'package:base':
##
##     date

## Loading required package: PerformanceAnalytics

## Warning: package 'PerformanceAnalytics' was built under R version 3.5.2

## Loading required package: xts

## Warning: package 'xts' was built under R version 3.5.2

## Loading required package: zoo

##
## Attaching package: 'zoo'
```

```
## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
##
## Attaching package: 'PerformanceAnalytics'
## The following object is masked from 'package:graphics':
##
##     legend
## Loading required package: quantmod
## Warning: package 'quantmod' was built under R version 3.5.2
## Loading required package: TTR
## Warning: package 'TTR' was built under R version 3.5.2
## Version 0.4-0 included new data defaults. See ?getSymbols.
## Loading required package: tidyverse
## Warning: package 'tidyverse' was built under R version 3.5.2
## -- Attaching packages --------------------------------------------------- tidyverse 1.2.1 --
## v ggplot2 3.1.0     v purrr   0.2.5
## v tibble  2.0.1     v dplyr   0.7.8
## v tidyr   0.8.2     v stringr 1.3.1
## v readr   1.3.1     v forcats 0.3.0
## Warning: package 'ggplot2' was built under R version 3.5.2
## Warning: package 'tibble' was built under R version 3.5.2
## Warning: package 'tidyr' was built under R version 3.5.2
## Warning: package 'readr' was built under R version 3.5.2
## Warning: package 'purrr' was built under R version 3.5.2
## Warning: package 'dplyr' was built under R version 3.5.2
## Warning: package 'stringr' was built under R version 3.5.2
## Warning: package 'forcats' was built under R version 3.5.2
## -- Conflicts ------------------------------------------------------ tidyverse_conflicts() --
## x lubridate::as.difftime() masks base::as.difftime()
## x lubridate::date()        masks base::date()
## x dplyr::filter()          masks stats::filter()
## x dplyr::first()           masks xts::first()
## x lubridate::intersect()   masks base::intersect()
## x dplyr::lag()             masks stats::lag()
## x dplyr::last()            masks xts::last()
## x lubridate::setdiff()     masks base::setdiff()
## x lubridate::union()       masks base::union()
```

```r
library(ggplot2)
library(GGally)
```

```
## Warning: package 'GGally' was built under R version 3.5.3
```

```
##
## Attaching package: 'GGally'

## The following object is masked from 'package:dplyr':
##
##     nasa
```

```
S=tq_get("^GSPC", get = "stock.prices",
         from="2010-01-03",to="2019-10-01")
y=tq_get("TYO", get = "stock.prices",
         from="2010-01-03",to="2019-10-01")
a=tq_get("AAPL",get="stock.prices",
         from="2010-01-03",to="2019-10-01")
h=tq_get("HSY",get="stock.prices",
         from="2010-01-03",to="2019-10-01")

p=data.frame(S$adjusted,y$adjusted,a$adjusted,h$adjusted)
date.p=c(S$date)
tail(p)
```

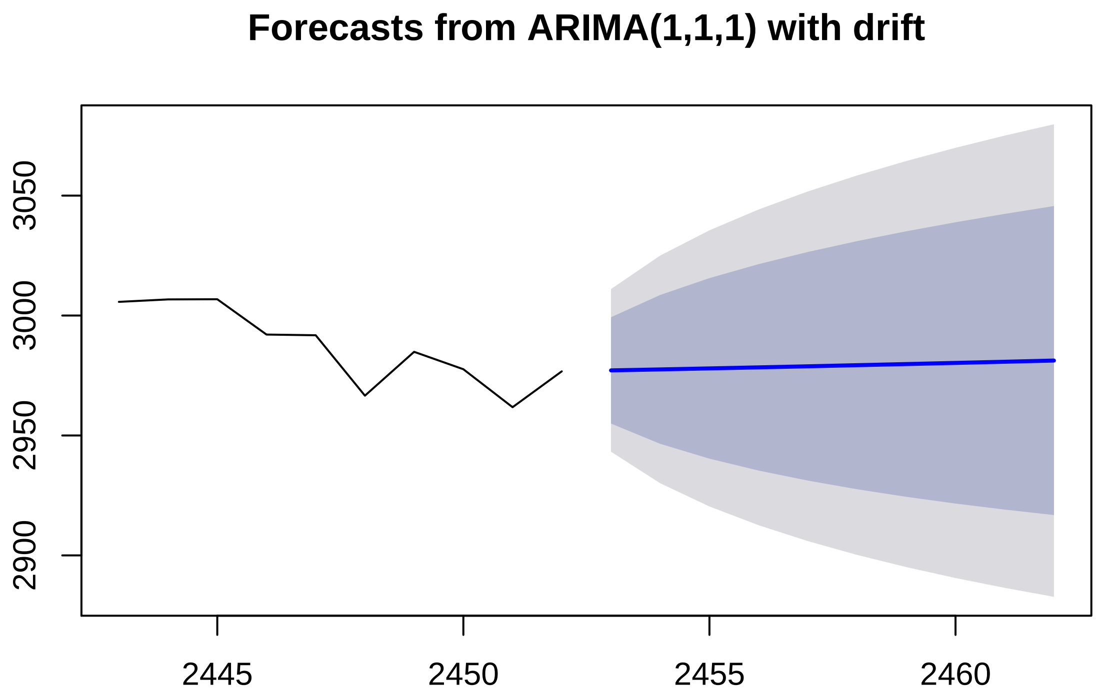
```
##      S.adjusted y.adjusted a.adjusted h.adjusted
## 2447    2991.78     10.713     218.72     153.31
## 2448    2966.60     10.540     217.68     155.66
## 2449    2984.87     10.740     221.03     153.37
## 2450    2977.62     10.680     219.89     154.28
## 2451    2961.79     10.630     218.82     153.78
## 2452    2976.74     10.610     223.97     154.99
```

Basic forecast. I need to improve this, it turn out very naive.

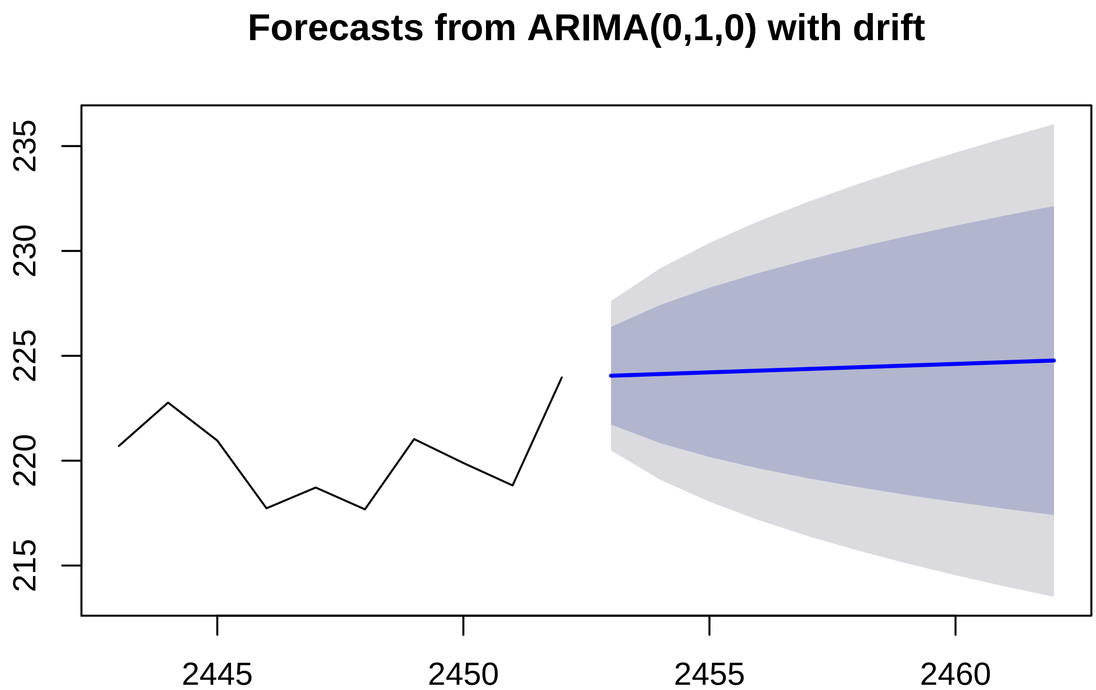
```
library(forecast)
```

```
## Warning: package 'forecast' was built under R version 3.5.2
```

```
plot(forecast(auto.arima(p$S.adjusted)),10)
```

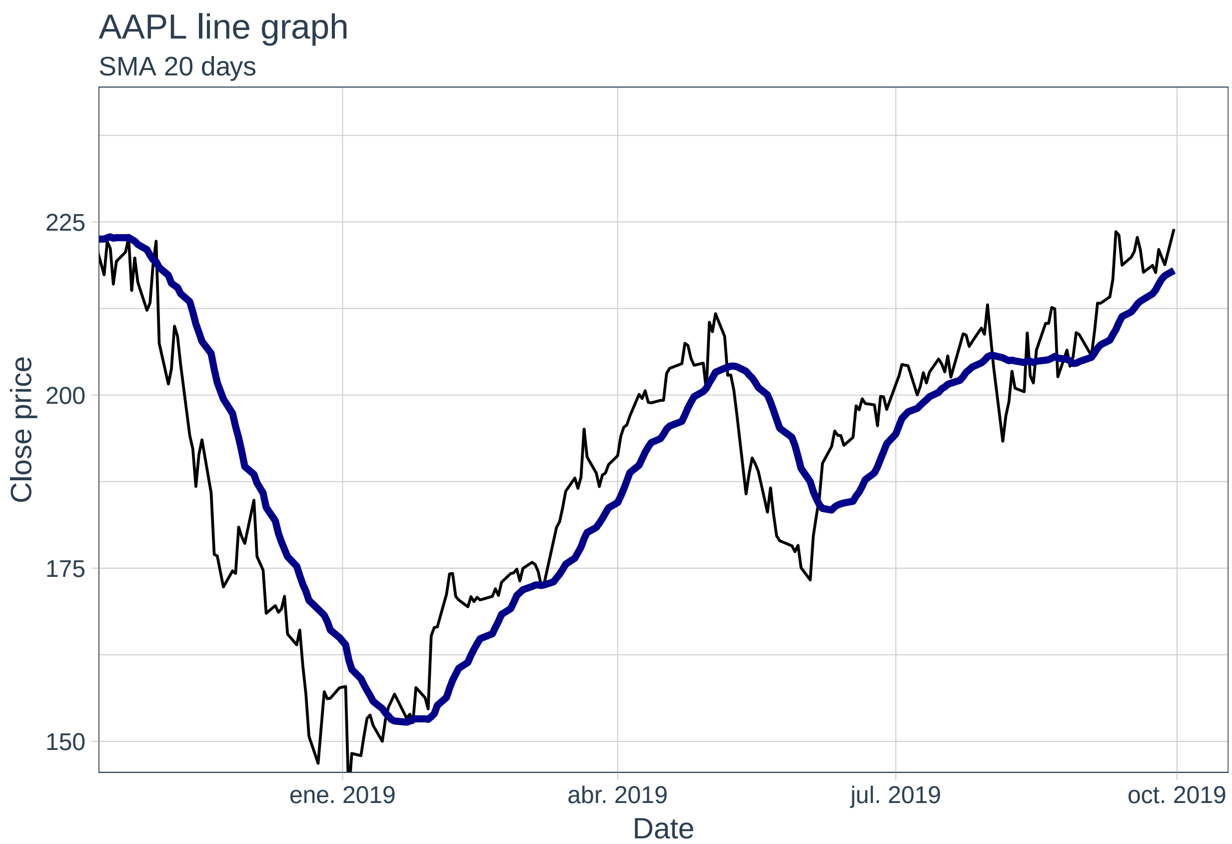



```r
plot(forecast(auto.arima(p$a.adjusted)),10)
```



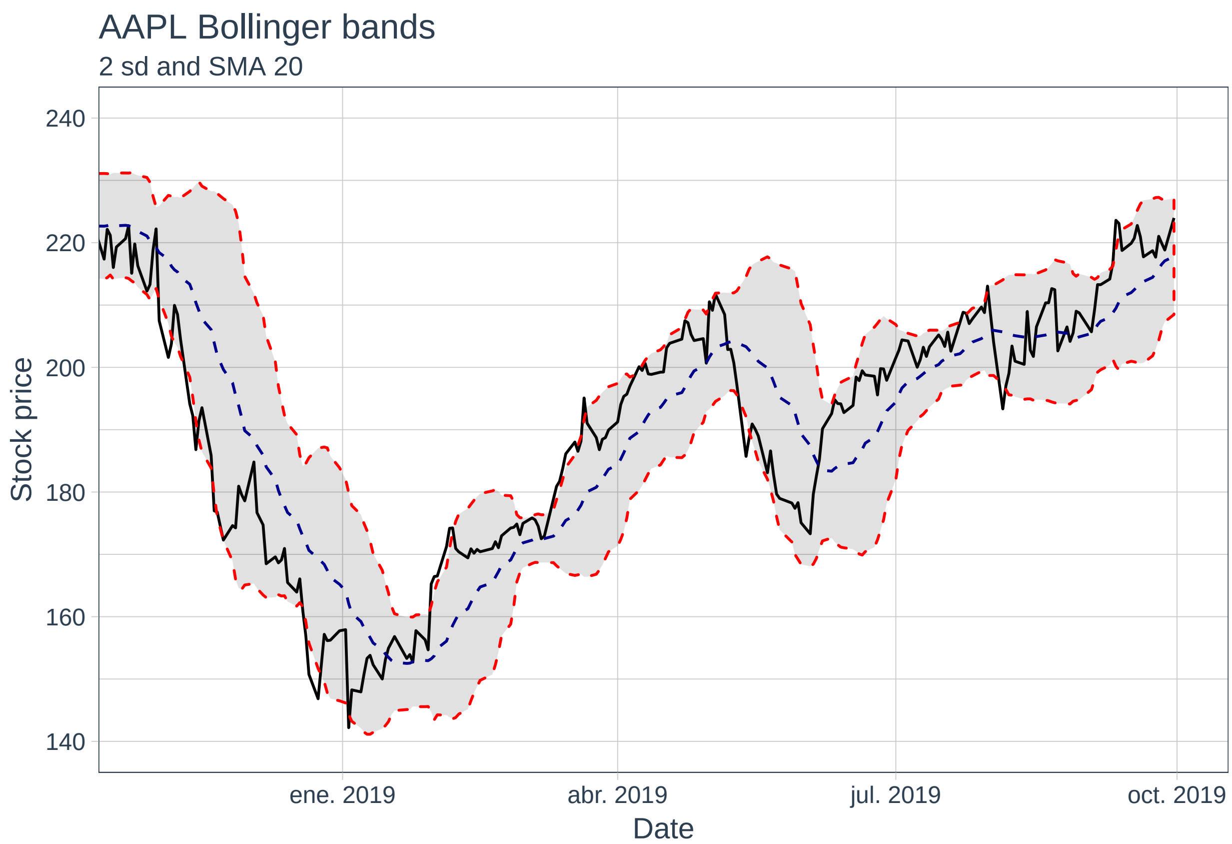
Technical analysis. The simple moving average.

```
end = as_date("2019-10-01")
a %>%
  ggplot(aes(x = date, y = close)) +
  geom_line() +
  geom_ma(ma_fun = SMA, n = 20, linetype = 1, size = 1.25) +
  labs(title = "AAPL line graph",
       subtitle = "SMA 20 days",
       y= "Close price", x = "Date") +
  coord_x_date(xlim = c(end - weeks(48), end), ylim = c(150, 240)) +
  theme_tq()
```



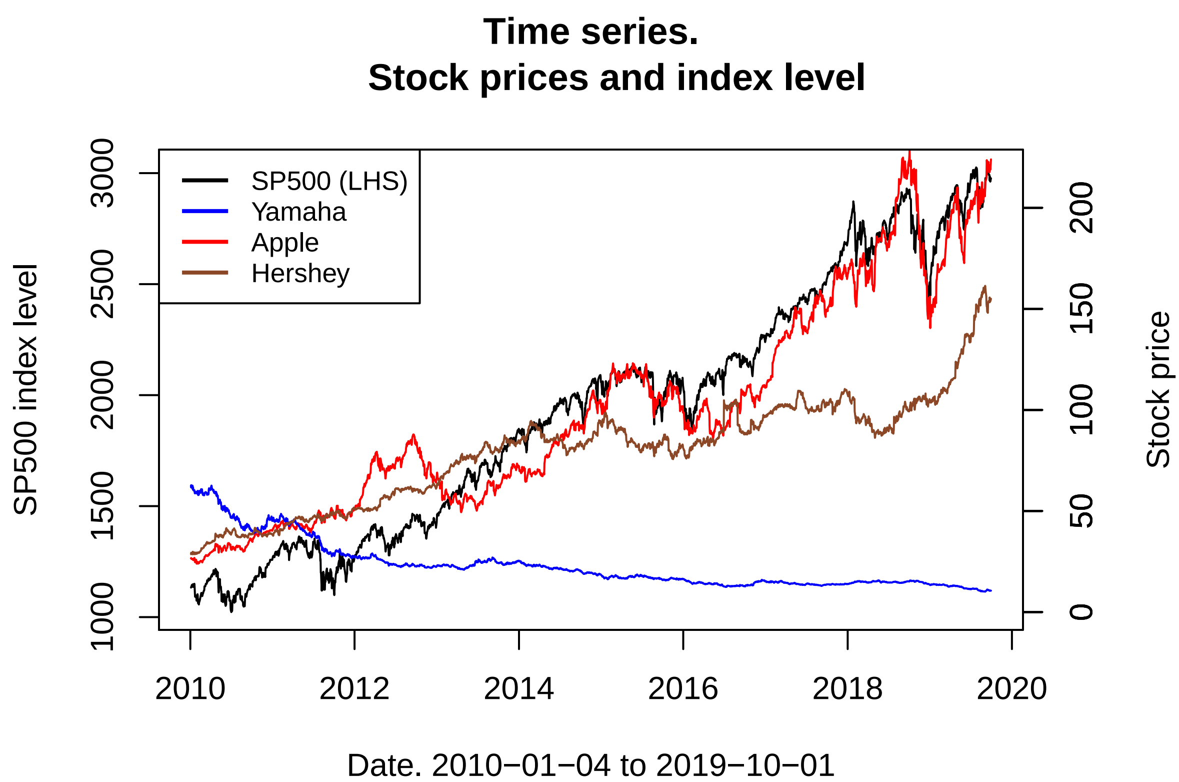
Bollinger bands.

```
a %>%
  ggplot(aes(x = date, y = close,
             open = open, high = high,
             low = low, close = close)) +
  geom_line() +
  geom_bbands(ma_fun=SMA,sd=2, n = 20) +
  labs(title="AAPL Bollinger bands",
       subtitle = "2 sd and SMA 20",
       y= "Stock price", x = "Date") +
  coord_x_date(xlim = c(end - weeks(48), end), ylim = c(140, 240)) +
  theme_tq()
```



Prices.

```
par(mar = c(5, 5, 5, 5))
plot(date.p,p$S.adjusted, type ="l", ylab = "SP500 index level",
     main = "Time series.
     Stock prices and index level",
     xlab = "Date. 2010-01-04 to 2019-10-01")
par(new = TRUE)
plot(date.p,p$a.adjusted, type = "l", xaxt = "n", yaxt = "n",
     ylab = "", xlab = "", col = "red",ylim = c(0,220))
axis(side = 4)
par(new = TRUE)
lines(date.p,p$y.adjusted, xaxt = "n", yaxt = "n",
      ylab = "", xlab = "", col = "blue")
axis(side = 4)
par(new = TRUE)
lines(date.p,p$h.adjusted, xaxt = "n", yaxt = "n",
      ylab = "", xlab = "", col = "sienna4")
axis(side = 4)
par(new = TRUE)
axis(side = 4)
mtext("Stock price", side = 4, line = 3)
legend("topleft", c("SP500 (LHS)", "Yamaha", "Apple", "Hershey"),
       col = c("black","blue","red", "sienna4"),lwd=c(2,2,2,2),cex=0.8)
```



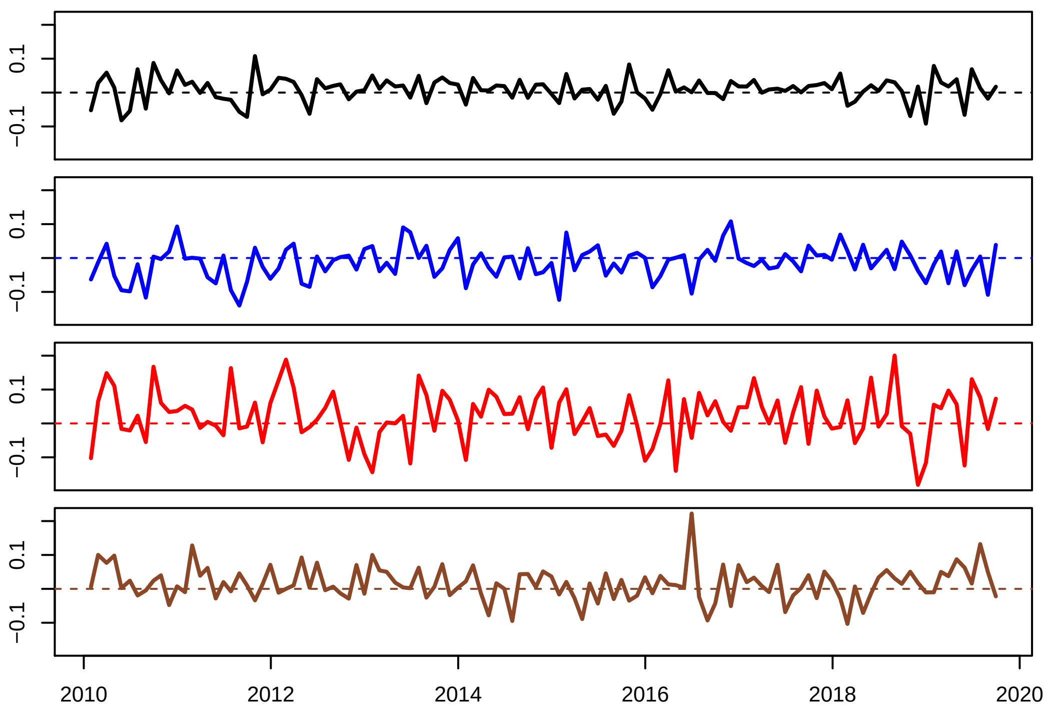
Returns.

```
n=length(p$y.adjusted)
r.y=(p$y.adjusted[2:n]-p$y.adjusted[1:(n-1)])/p$y.adjusted[1:(n-1)]

stock_returns_monthly = c("^GSPC","TYO","AAPL","HSY") %>%
  tq_get(get  = "stock.prices",
         from = "2010-01-03",
         to   = "2019-10-01") %>%
  group_by(symbol) %>%
  tq_transmute(select     = adjusted,
               mutate_fun = periodReturn,
               period     = "monthly",
               col_rename = "Ra")

date.r=data.frame(filter(stock_returns_monthly,symbol=="^GSPC"))[,2]
S=data.frame(filter(stock_returns_monthly,symbol=="^GSPC"))[,3]
y=data.frame(filter(stock_returns_monthly,symbol=="TYO"))[,3]
a=data.frame(filter(stock_returns_monthly,symbol=="AAPL"))[,3]
h=data.frame(filter(stock_returns_monthly,symbol=="HSY"))[,3]

r=data.frame(S,y,a,h)
min=min(r)
max=max(r)

head(r)
```

```
##             S           y           a           h
## 1 -0.05218051 -0.06327318 -0.10256479 0.005798019
## 2  0.02851369 -0.01304134  0.06539588 0.100268078
## 3  0.05879643  0.04213797  0.14847048 0.076709749
## 4  0.01475923 -0.05279443  0.11102103 0.098107979
## 5 -0.08197584 -0.09513168 -0.01612487 0.002412095
## 6 -0.05388244 -0.09847909 -0.02082659 0.024145441
```

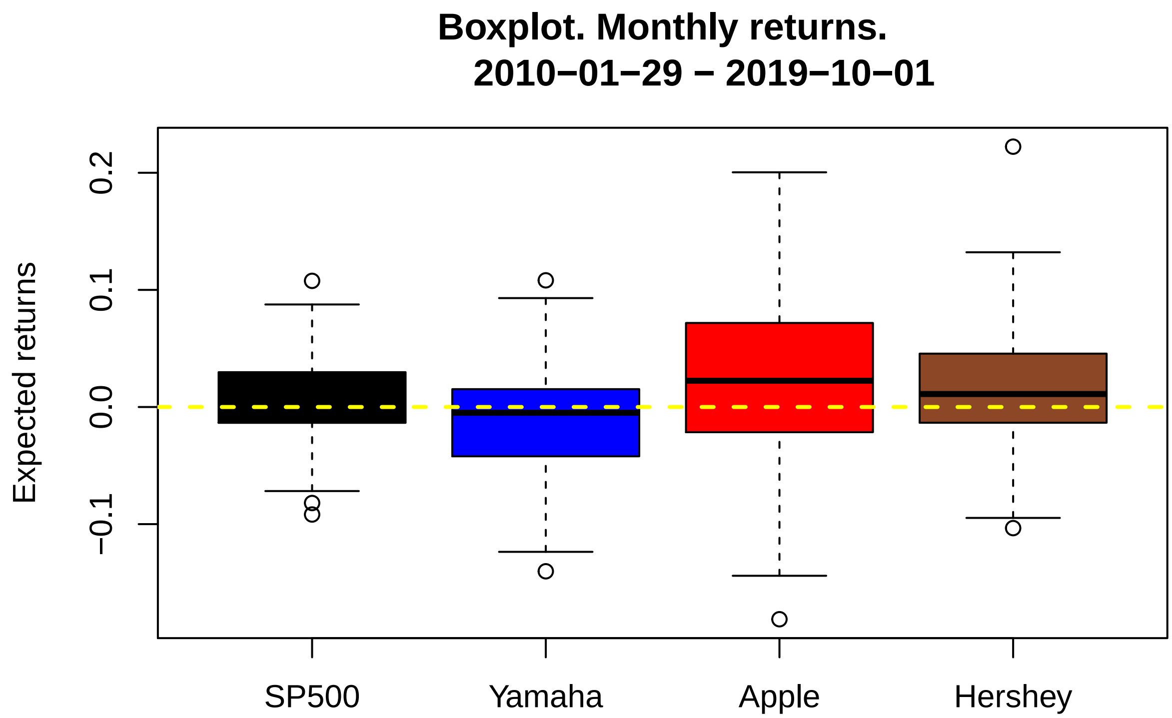
Time series of monthly returns.

```
par(mfcol=c(4,1)) # divide the plotting space into 4 plots
par(mar=c(0.5, 0.5, 0.2, 0.2),
    oma = c(4, 6, 4, 4))
plot(date.r,r$S,type="l",xaxt="n",xlab="",lwd=2,
     ylim=c(min,max))
abline(0,0,lty=2)
plot(date.r,r$y,type="l",xaxt="n",xlab="",col="blue",lwd=2,
     ylim=c(min,max))
abline(0,0,lty=2,col="blue")
plot(date.r,r$a,type="l",xaxt ="n",xlab="",col="red",lwd=2,
     ylim=c(min,max))
abline(0,0,lty=2,col="red")
plot(date.r,r$h,type="l",xlab = "Date",col="sienna4",lwd=2,
     ylim=c(min,max))
abline(0,0,lty=2,col="sienna4")
```



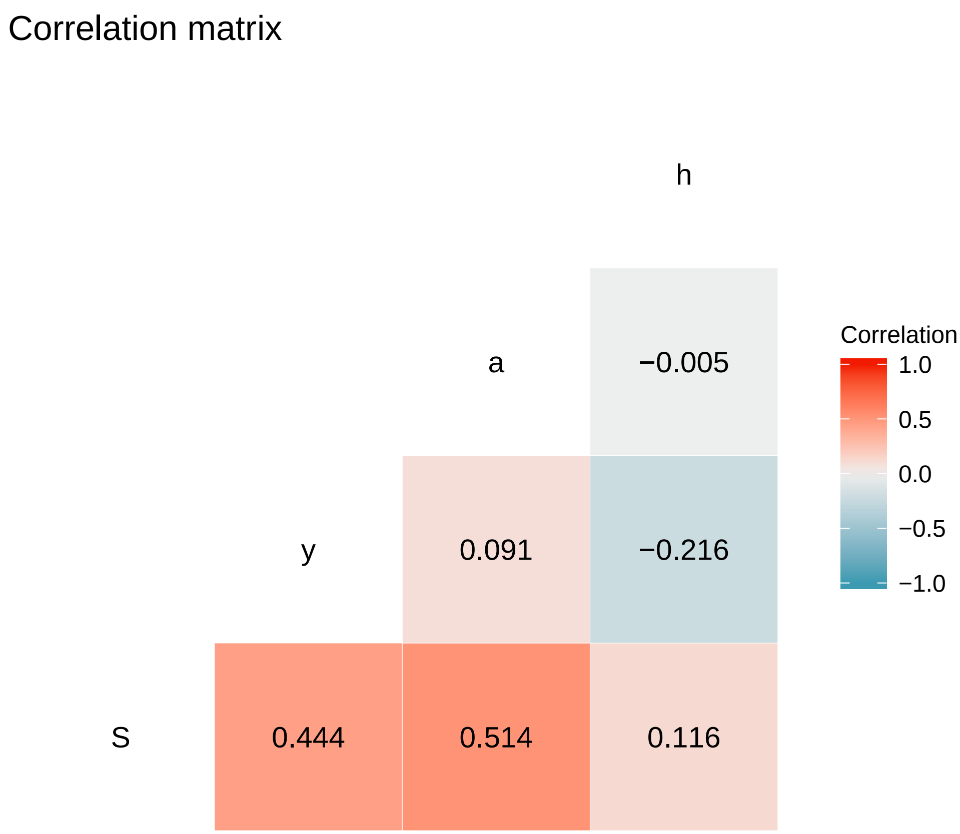
Return distribution.

```r
boxplot(r,col=c("black","blue","red", "sienna4"),
        names = c("SP500","Yamaha","Apple","Hershey"),
        main="Boxplot. Monthly returns.
        2010-01-29 - 2019-10-01",
        ylab="Expected returns")
abline(h=0,lty=2,col="yellow",lwd=2)
```



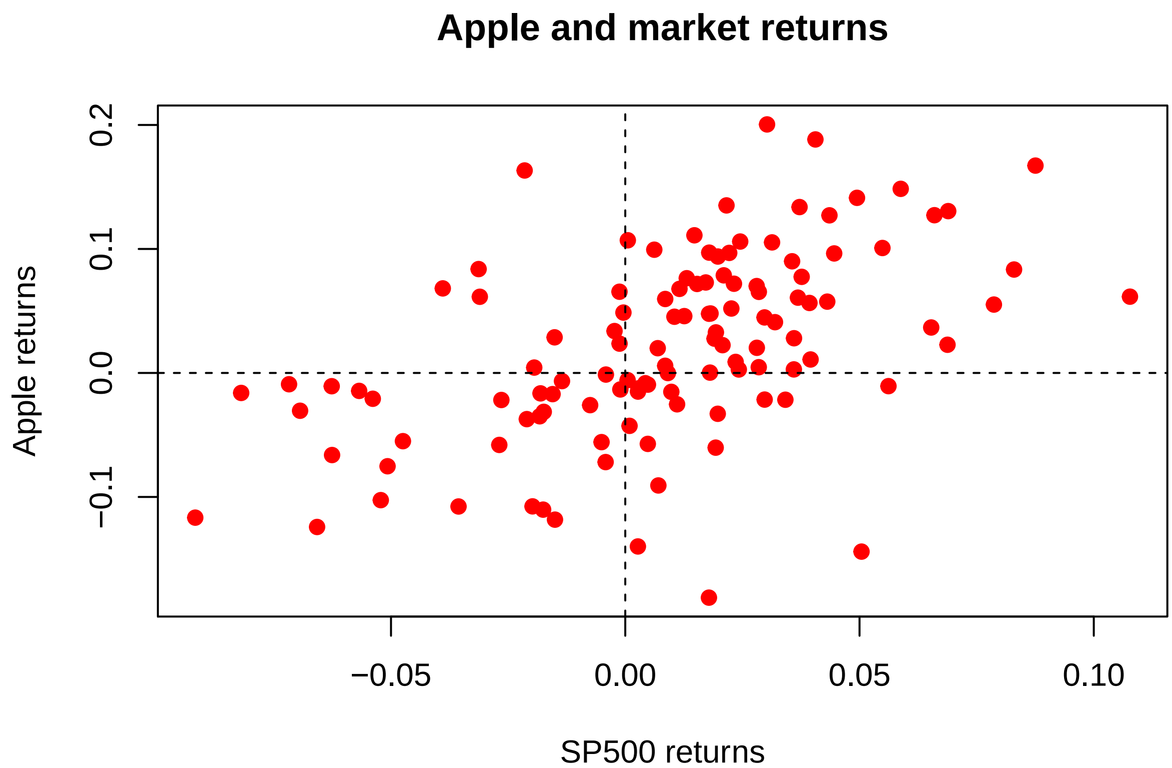
Correlation.

```r
ggcorr(r,label = TRUE,label_round = 3,
       name="Correlation",geom = "tile")+
  labs(title = "Correlation matrix")
```



Returns in a scatter plot.

```
plot(r$S,r$a,pch=19,col="red",
     main = "Apple and market returns",
     ylab = "Apple returns",xlab = "SP500 returns")
abline(v=0,lty=2)
abline(h=0, lty=2)
```



Apple as a function of the stock index.

```r
apple=lm(a ~ S,data=r)
summary(apple)
```

```
##
## Call:
## lm(formula = a ~ S, data = r)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.211736 -0.034484 -0.003948  0.043886  0.174336
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.01166    0.00615   1.895   0.0606 .
## S            1.05734    0.16434   6.434 2.97e-09 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.06459 on 115 degrees of freedom
## Multiple R-squared:  0.2647, Adjusted R-squared:  0.2583
## F-statistic: 41.39 on 1 and 115 DF,  p-value: 2.965e-09
```

Get rid of heteroscedasticiy.

```r
library(car)
```

```
## Warning: package 'car' was built under R version 3.5.2
```

```
## Loading required package: carData

## Warning: package 'carData' was built under R version 3.5.2

##
## Attaching package: 'car'

## The following object is masked from 'package:dplyr':
##
##     recode

## The following object is masked from 'package:purrr':
##
##     some
```

```
library(lmtest)
```

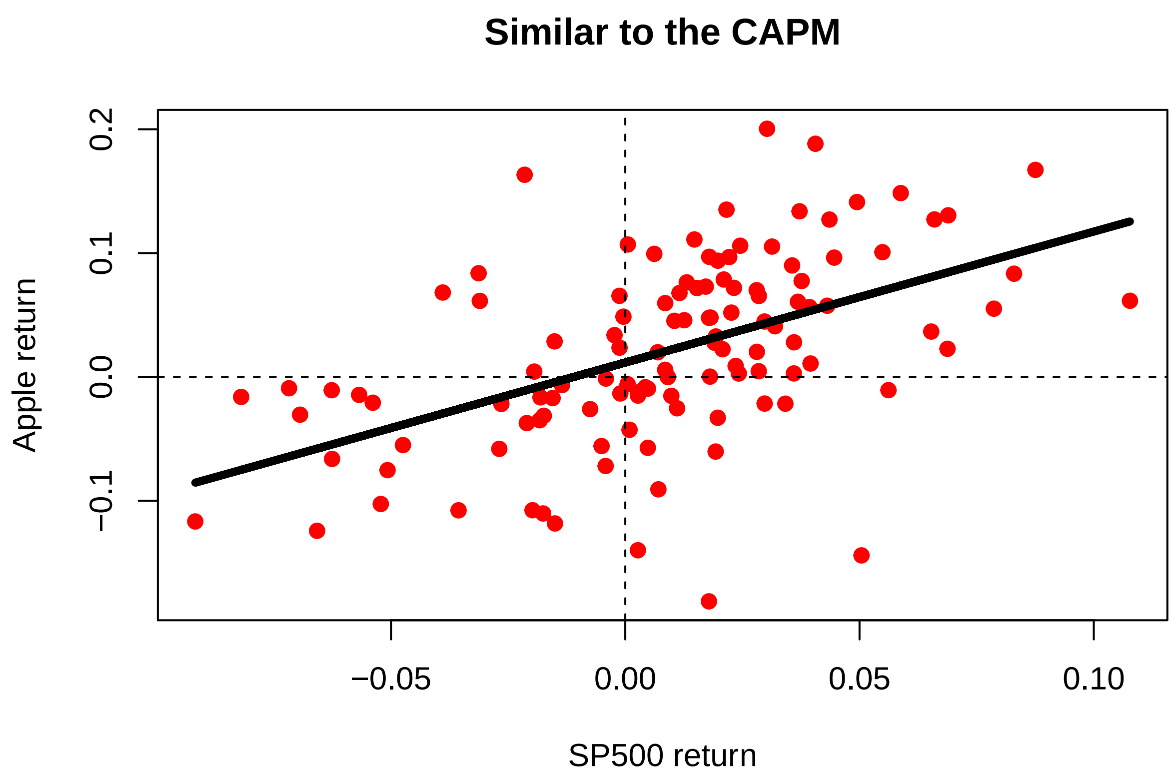
```
## Warning: package 'lmtest' was built under R version 3.5.2
```

```
cov = hccm(apple, type="hc1")
apple.h = coeftest(apple, vcov.=cov)
apple.h
```

```
##
## t test of coefficients:
##
##              Estimate Std. Error t value  Pr(>|t|)
## (Intercept) 0.0116548  0.0059878  1.9464   0.05405 .
## S           1.0573450  0.1516453  6.9725 2.094e-10 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Something like the Capital Asset Pricing Model.

```
plot(r$S,r$a,pch=19,col="red",
     main = "Similar to the CAPM",
     ylab = "Apple return",xlab = "SP500 return")
lines(r$S,fitted(apple),lwd=4)
abline(v=0,lty=2)
abline(h=0,lty=2)
```



5 Text mining.

The following material and examples are about text mining.

6 Susan B. Anthony.

This section show the main point of text mining techniques. In particular, how to go from a given text to the main subject of study which is called token. And then, how to deal with tokens.

Susan B. Anthony was an American social reformer and women's rights activist who played a pivotal role in the women's suffrage movement.

Consider the following speech extract by Susan B. Anthony - 1873. This is a typical character vector that we might want to analyze. The whole speech it is available in Internet, you can find it as "Is it a Crime for a Citizen of the United States to Vote?" It is a great speech by the way.

```
rm(list=ls())
# Here, R is asked to store the text in the variable called "text".
```

```r
text <- c("Friends and fellow citizens: I stand",
          "before you tonight under indictment for",
          "the alleged crime of having voted at the",
          "last presidential election, without having",
          "a lawful right to vote. It shall be my work",
          "this evening to prove to you that in thus",
          "voting, I not only committed no crime,",
          "but, instead, simply exercised my",
          "citizen's rights, guaranteed to me and all",
          "United States citizens by the National Constitution,",
          "beyond the power of any state to deny.")
# By typing the name of the variable, we reproduce its contents.
text
```

```
##  [1] "Friends and fellow citizens: I stand"
##  [2] "before you tonight under indictment for"
##  [3] "the alleged crime of having voted at the"
##  [4] "last presidential election, without having"
##  [5] "a lawful right to vote. It shall be my work"
##  [6] "this evening to prove to you that in thus"
##  [7] "voting, I not only committed no crime,"
##  [8] "but, instead, simply exercised my"
##  [9] "citizen's rights, guaranteed to me and all"
## [10] "United States citizens by the National Constitution,"
## [11] "beyond the power of any state to deny."
```

We have 11 lines of text in the variable "text". Although this is the regular format in common printed documents, it is not the best way to analyze it in R. In order to turn it into a tidy text data-set. A tidy text format has the following principles: each variable is a column, each observation is a row, and each type of observational unit is a table. We thus define the tidy text format as being a table with one-token-per-row. In order to do this, we first need to take the original text and put it into a data frame.

```r
library(dplyr) # This package provides a set of tools for efficiently manipulating datasets.
# We transform the varaible "text" into a data frame and store it in text_df.
    text_df <- data_frame(line = 1:11, text = text)
```

```
## Warning: `data_frame()` is deprecated, use `tibble()`.
## This warning is displayed once per session.
```

```r
text_df # We reproduce the result.
```

```
## # A tibble: 11 x 2
##     line text
##    <int> <chr>
##  1     1 Friends and fellow citizens: I stand
##  2     2 before you tonight under indictment for
##  3     3 the alleged crime of having voted at the
##  4     4 last presidential election, without having
##  5     5 a lawful right to vote. It shall be my work
##  6     6 this evening to prove to you that in thus
##  7     7 voting, I not only committed no crime,
##  8     8 but, instead, simply exercised my
##  9     9 citizen's rights, guaranteed to me and all
## 10    10 United States citizens by the National Constitution,
## 11    11 beyond the power of any state to deny.
```

This is still not ready to be analyzed by R. Let's transform this data frame in one token per row using unnest_tokens() method.

```
library(tidytext) # One of the most useful tools to do text mining with R.
```

```
## Warning: package 'tidytext' was built under R version 3.5.3
```

```
  text_df %>%
    unnest_tokens(word, text)
```

```
## # A tibble: 78 x 2
##     line word
##    <int> <chr>
##  1     1 friends
##  2     1 and
##  3     1 fellow
##  4     1 citizens
##  5     1 i
##  6     1 stand
##  7     2 before
##  8     2 you
##  9     2 tonight
## 10     2 under
## # ... with 68 more rows
```

```
# You can always learn about R functions by (1) typing ?unnest_tokens in the R console,
  #(2) Look in the Help menu, (3) Google it.
```

Now we have a tidy text and every row is one word. However, this is still not ready to be analyzed since we have extremely frequent words and this does not add any valuable meaning to the text analysis. For example:

```
  tokens=text_df %>%
    unnest_tokens(word, text)
  tokens %>% count(word, sort = TRUE)
```

```
## # A tibble: 63 x 2
##    word          n
##    <chr>     <int>
##  1 to            5
##  2 the           4
##  3 and           2
##  4 citizens      2
##  5 crime         2
##  6 having        2
##  7 i             2
##  8 my            2
##  9 of            2
## 10 you           2
## # ... with 53 more rows
```

The word "to" repeats 5 times and "the" repeats 4 times. Normally, we are not interested in these kind of words as this adds no value to understand Susan B. Anthony speech. This is how we can get rid of those words:

```
 library(stringr)
data(stop_words)
    tokens2 <- tokens %>%
      mutate(word = str_extract(word, "[a-z']+")) %>%
```

```
    anti_join(stop_words)
```

```
## Joining, by = "word"
```

```
  tokens2
```

```
## # A tibble: 29 x 2
##     line word
##    <int> <chr>
##  1     1 friends
##  2     1 fellow
##  3     1 citizens
##  4     1 stand
##  5     2 tonight
##  6     2 indictment
##  7     3 alleged
##  8     3 crime
##  9     3 voted
## 10     4 presidential
## # ... with 19 more rows
```

Now we have 29 instead of 63 words. The new frequency list of words is:

```
tokens2 %>% count(word, sort = TRUE)
```

```
## # A tibble: 27 x 2
##    word             n
##    <chr>        <int>
##  1 citizens         2
##  2 crime            2
##  3 alleged          1
##  4 citizen's        1
##  5 committed        1
##  6 constitution     1
##  7 deny             1
##  8 election         1
##  9 evening          1
## 10 exercised        1
## # ... with 17 more rows
```

Now we can understand better the text. Normally, we analyze a significantly bigger amount of text data.

7 Jane Austen.

Let's use the text of Jane Austen's six completed, published novels from the janeaustenr R package (Silge 2016), and transform them into a tidy format. These novels explore the dependence of women on marriage in the pursuit of favourable social standing and economic security, and are fully available in R for free.

We use mutate() to annotate a line number quantity to keep track of lines in the original format, and a chapter (using a regex) to find where all the chapters are.

```
library(janeaustenr)
```

```
## Warning: package 'janeaustenr' was built under R version 3.5.3
```

```
original_books <- austen_books() %>%
  group_by(book) %>%
  mutate(linenumber = row_number(),
```

```r
        chapter = cumsum(str_detect(text, regex("^chapter [\\divxlc]",
                    ignore_case = TRUE)))) %>%
  ungroup()
original_books
```

```
## # A tibble: 73,422 x 4
##    text                  book                linenumber chapter
##    <chr>                 <fct>                    <int>   <int>
##  1 SENSE AND SENSIBILITY Sense & Sensibility          1       0
##  2 ""                    Sense & Sensibility          2       0
##  3 by Jane Austen        Sense & Sensibility          3       0
##  4 ""                    Sense & Sensibility          4       0
##  5 (1811)                Sense & Sensibility          5       0
##  6 ""                    Sense & Sensibility          6       0
##  7 ""                    Sense & Sensibility          7       0
##  8 ""                    Sense & Sensibility          8       0
##  9 ""                    Sense & Sensibility          9       0
## 10 CHAPTER 1             Sense & Sensibility         10       1
## # ... with 73,412 more rows
```

You can browse the book contents directly in R Studio. According to the previous results we have a total of 73,422 lines, not words, but lines.

To have a visual idea of the amount of text that we analyze, here are the six novels by Jane Austen:

And here is how we extract the names of these books in R.

```r
original_books %>% count(book)
```

```
## # A tibble: 6 x 2
##   book                    n
##   <fct>               <int>
## 1 Sense & Sensibility 12624
## 2 Pride & Prejudice   13030
## 3 Mansfield Park      15349
## 4 Emma                16235
## 5 Northanger Abbey     7856
## 6 Persuasion           8328
```

The variable n in the table above represent the number of text lines in each book. In total, we have $12,624 + 13,030 + 15,349 + 16,235 + 7,856 + 8,328 = 73,422$ lines of text in the six books. If we would like to read the first 15 lines of the first book we can do:

```r
original_books$text[1:15]
```

```
##  [1] "SENSE AND SENSIBILITY"
##  [2] ""
##  [3] "by Jane Austen"
##  [4] ""
##  [5] "(1811)"
##  [6] ""
##  [7] ""
##  [8] ""
##  [9] ""
## [10] "CHAPTER 1"
## [11] ""
## [12] ""
## [13] "The family of Dashwood had long been settled in Sussex.  Their estate"
## [14] "was large, and their residence was at Norland Park, in the centre of"
## [15] "their property, where, for many generations, they had lived in so"
```

And the last 10 lines of Sense & Sensibility:

```r
original_books$text[12614:12624]
```

```
##  [1] ""
##  [2] "Between Barton and Delaford, there was that constant communication"
##  [3] "which strong family affection would naturally dictate;--and among the"
##  [4] "merits and the happiness of Elinor and Marianne, let it not be ranked"
##  [5] "as the least considerable, that though sisters, and living almost"
##  [6] "within sight of each other, they could live without disagreement"
##  [7] "between themselves, or producing coolness between their husbands."
##  [8] ""
##  [9] ""
## [10] ""
## [11] "THE END"
```

7.1 Most and least common words.

As in regular statistic procedures, we are interested to know which are the most common words used in a given material. The least common words are also informative because these might represent the distinctive approach of the material or the differentiated characteristic with respect to other materials. In order to extract the most and least common words, we need to restructure the data in the one-token-per-row format. This basically mean that instead of lines per row as in the example above, we now need one word per row.

```r
library(tidytext)
    tidy_books <- original_books %>%
      unnest_tokens(word, text)
    tidy_books
```

```
## # A tibble: 725,055 x 4
##    book                linenumber chapter word
##    <fct>                    <int>   <int> <chr>
##  1 Sense & Sensibility          1       0 sense
##  2 Sense & Sensibility          1       0 and
##  3 Sense & Sensibility          1       0 sensibility
##  4 Sense & Sensibility          3       0 by
##  5 Sense & Sensibility          3       0 jane
##  6 Sense & Sensibility          3       0 austen
```

```
##   7 Sense & Sensibility          5       0 1811
##   8 Sense & Sensibility         10       1 chapter
##   9 Sense & Sensibility         10       1 1
##  10 Sense & Sensibility         13       1 the
## # ... with 725,045 more rows
```

We have $725,055$ rows or words in the six books including stop words. This makes sense because remember we have $73,422$ lines, so we have an average of $725,055/73,422 = 9.9$ words per-line. Now it is time to remove stop words. We can remove stop words (kept in the tidytext data-set stop_words) with an anti_join(). We also avoid UTF-8 encoded text with underscores recording the underscores as words with mutate(word = str_extract(word, "[a-z']+")). In practical terms, we can do so with the following chunk of code:

```r
data(stop_words)
    tidy_books <- tidy_books %>%
      mutate(word = str_extract(word, "[a-z']+")) %>%
      anti_join(stop_words)
```

```
## Joining, by = "word"
```

```r
    tidy_books
```

```
## # A tibble: 216,385 x 4
##    book                linenumber chapter word
##    <fct>                    <int>   <int> <chr>
##  1 Sense & Sensibility          1       0 sense
##  2 Sense & Sensibility          1       0 sensibility
##  3 Sense & Sensibility          3       0 jane
##  4 Sense & Sensibility          3       0 austen
##  5 Sense & Sensibility          5       0 <NA>
##  6 Sense & Sensibility         10       1 chapter
##  7 Sense & Sensibility         10       1 <NA>
##  8 Sense & Sensibility         13       1 family
##  9 Sense & Sensibility         13       1 dashwood
## 10 Sense & Sensibility         13       1 settled
## # ... with 216,375 more rows
```

We end up with $216,385$ rows or words used in the six books once we get rid of stop words. If you do the math, we started with $725,055$, so the amount of stop words that we eliminate is $725,055 - 216,385 = 508,670$, that is around 70% of the original amount of words.

These $216,385$ words are not unique, some of them are repeated in the six novels. Let's use count() to group the $216,385$ and find the new set of most common words.

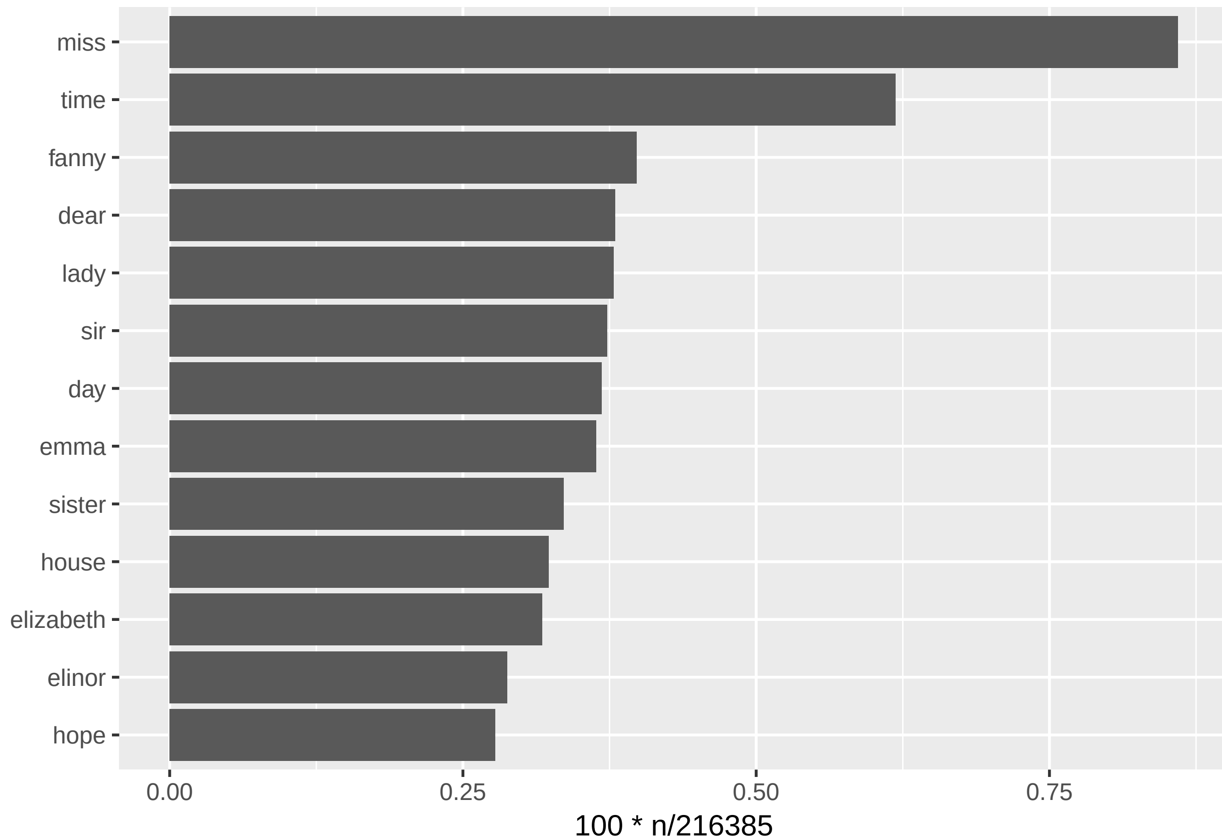
```r
tidy_books %>% count(word, sort = TRUE)
```

```
## # A tibble: 13,464 x 2
##    word       n
##    <chr>  <int>
##  1 miss    1860
##  2 time    1339
##  3 fanny    862
##  4 dear     822
##  5 lady     819
##  6 sir      807
##  7 day      797
##  8 emma     787
##  9 sister   727
## 10 house    699
```

```
## # ... with 13,454 more rows
```

So we have 216, 385 words and 13, 464 of them are unique. Let's visualize the most common words in all the six books.

```
library(ggplot2)
tidy_books %>%
count(word, sort = TRUE) %>% filter(n >600) %>%
mutate(word = reorder(word, 100*n/216385)) %>%
  ggplot(aes(word, 100*n/216385)) +
geom_col() +
xlab(NULL) +
coord_flip()
```



Note that the most common word (miss, 1, 860) represent 0.85% of the total (216, 385).

The words that are used once in the whole collection can be extracted as:

```
tidy_books %>% count(word, sort = TRUE) %>% filter(n ==1)
```

```
## # A tibble: 4,470 x 2
##    word             n
##    <chr>        <int>
##  1 a'n't            1
##  2 abandoned        1
##  3 abashed          1
##  4 abating          1
##  5 abbeyland        1
##  6 abbots           1
```

```
##   7 abbreviation     1
##   8 abdication       1
##   9 abdy's           1
##  10 abiding          1
## # ... with 4,460 more rows
```

All the words used once sum 4, 470, or about 2% of the total (216, 385).

Text mining is also about classify. The previous analysis take the six books as a total. Now, we are interested to analyze the number of words per book. This requires a slightly different arrangement.

```
book_words <- austen_books() %>% unnest_tokens(word, text) %>%
  count(book, word, sort = TRUE) %>% ungroup()
    total_words <- book_words %>%
      group_by(book) %>%
      summarize(total = sum(n))
    book_words <- left_join(book_words, total_words)
```

```
## Joining, by = "book"
```

```
book_words
```

```
## # A tibble: 40,379 x 4
##    book               word      n  total
##    <fct>              <chr> <int>  <int>
##  1 Mansfield Park     the    6206 160460
##  2 Mansfield Park     to     5475 160460
##  3 Mansfield Park     and    5438 160460
##  4 Emma               to     5239 160996
##  5 Emma               the    5201 160996
##  6 Emma               and    4896 160996
##  7 Mansfield Park     of     4778 160460
##  8 Pride & Prejudice the    4331 122204
##  9 Emma               of     4291 160996
## 10 Pride & Prejudice to     4162 122204
## # ... with 40,369 more rows
```

Now we remove the stop words.

```
data(stop_words)
    book_words <- book_words %>%
      mutate(word = str_extract(word, "[a-z']+")) %>%
      anti_join(stop_words)
```

```
## Joining, by = "word"
```

```
    book_words
```

```
## # A tibble: 36,904 x 4
##    book                  word          n  total
##    <fct>                 <chr>     <int>  <int>
##  1 Mansfield Park        fanny       816 160460
##  2 Emma                  emma        786 160996
##  3 Sense & Sensibility elinor      623 119957
##  4 Emma                  miss        599 160996
##  5 Pride & Prejudice   elizabeth   597 122204
##  6 Mansfield Park        crawford    493 160460
##  7 Sense & Sensibility marianne    492 119957
##  8 Persuasion            anne        447  83658
```

```
##  9 Mansfield Park      miss        432 160460
## 10 Northanger Abbey    catherine   428  77780
## # ... with 36,894 more rows
```

The word "fanny" is the most common word as Fanny Price is the heroine in Jane Austen's 1814 novel, Mansfield Park. In fact, most of these frequent words corresponds to the names of the main characters in each novel. Note that we can learn more about the contents by doing a simple classification of the six novels.

Given the analysis in the previous section (without classification per novel), we know that the word "miss" very frequently in the six books. And here is how we can know how this word is distributed among the six books:

```
book_words %>% filter(word=="miss")
```

```
## # A tibble: 9 x 4
##   book                word      n  total
##   <fct>               <chr> <int>  <int>
## 1 Emma                miss    599 160996
## 2 Mansfield Park      miss    432 160460
## 3 Pride & Prejudice   miss    283 122204
## 4 Sense & Sensibility miss    210 119957
## 5 Northanger Abbey    miss    206  77780
## 6 Persuasion          miss    125  83658
## 7 Emma                miss      3 160996
## 8 Mansfield Park      miss      1 160460
## 9 Mansfield Park      miss      1 160460
```

It is inevitable to get some fun here:

```
book_words %>% filter(word=="martin")
```

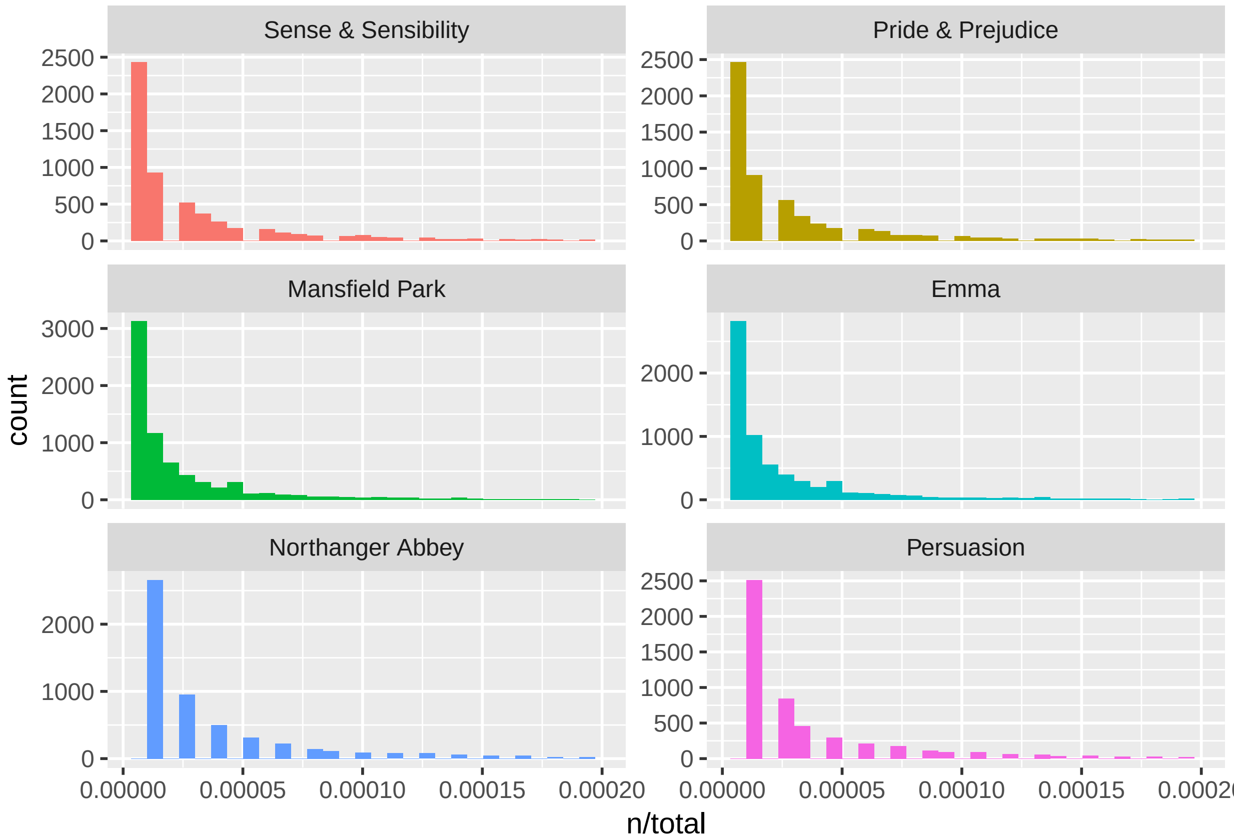
```
## # A tibble: 1 x 4
##   book  word       n  total
##   <fct> <chr>  <int>  <int>
## 1 Emma  martin    72 160996
```

Robert Martin is a character in Jane Austen's Emma. He is a farmer, not a gentleman, Mr. Martin is not considered handsome, Harriet Smith at first thought him quite plain. Later, she changed her mind as she was charmed by his personality and genuine friendliness. (Good for you Martin).

Enough fun. Let's look at the distribution of term frequency (i.e. n/total) for each novel.

```
library(ggplot2)
ggplot(book_words, aes(n/total, fill = book)) +
  geom_histogram(show.legend = FALSE) + xlim(NA, 0.0002) +
  facet_wrap(~book, ncol = 2, scales = "free_y")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.

## Warning: Removed 1644 rows containing non-finite values (stat_bin).

## Warning: Removed 6 rows containing missing values (geom_bar).
```



7.2 Zipf's Law.

Zipf's law is an empirical law, formulated using mathematical statistics, named after the linguist George Kingsley Zipf, who first proposed it.

Zipf's law states that given a large sample of words used, the frequency of any word is inversely proportional to its rank in the frequency table. So word number n has a frequency proportional to $1/n$.

Thus the most frequent word will occur about twice as often as the second most frequent word, three times as often as the third most frequent word, etc. For example, in one sample of words in the English language, the most frequently occurring word, "the", accounts for nearly 7% of all the words (69,971 out of slightly over 1 million). True to Zipf's Law, the second-place word "of" accounts for slightly over 3.5% of words (36,411 occurrences), followed by "and" (28,852). Only about 135 words are needed to account for half the sample of words in a large sample.

Let's examine Zipf's law for Jane Austen's novels. This value is in the last column "term frequency" which measures how frequently a term occurs in a document. Later we name it simply as "tf".

```
freq_by_rank <- book_words %>%
      group_by(book) %>%
      mutate(rank = row_number(),
             `term frequency` = n/total)
    freq_by_rank
```

```
## # A tibble: 36,904 x 6
## # Groups:   book [6]
##    book                 word          n  total  rank `term frequency`
##    <fct>                <chr>     <int>  <int> <int>            <dbl>
```

```
##   1 Mansfield Park      fanny       816 160460     1           0.00509
##   2 Emma                emma        786 160996     1           0.00488
##   3 Sense & Sensibility elinor      623 119957     1           0.00519
##   4 Emma                miss        599 160996     2           0.00372
##   5 Pride & Prejudice   elizabeth   597 122204     1           0.00489
##   6 Mansfield Park      crawford    493 160460     2           0.00307
##   7 Sense & Sensibility marianne    492 119957     2           0.00410
##   8 Persuasion          anne        447  83658     1           0.00534
##   9 Mansfield Park      miss        432 160460     3           0.00269
##  10 Northanger Abbey    catherine   428  77780     1           0.00550
## # ... with 36,894 more rows
```

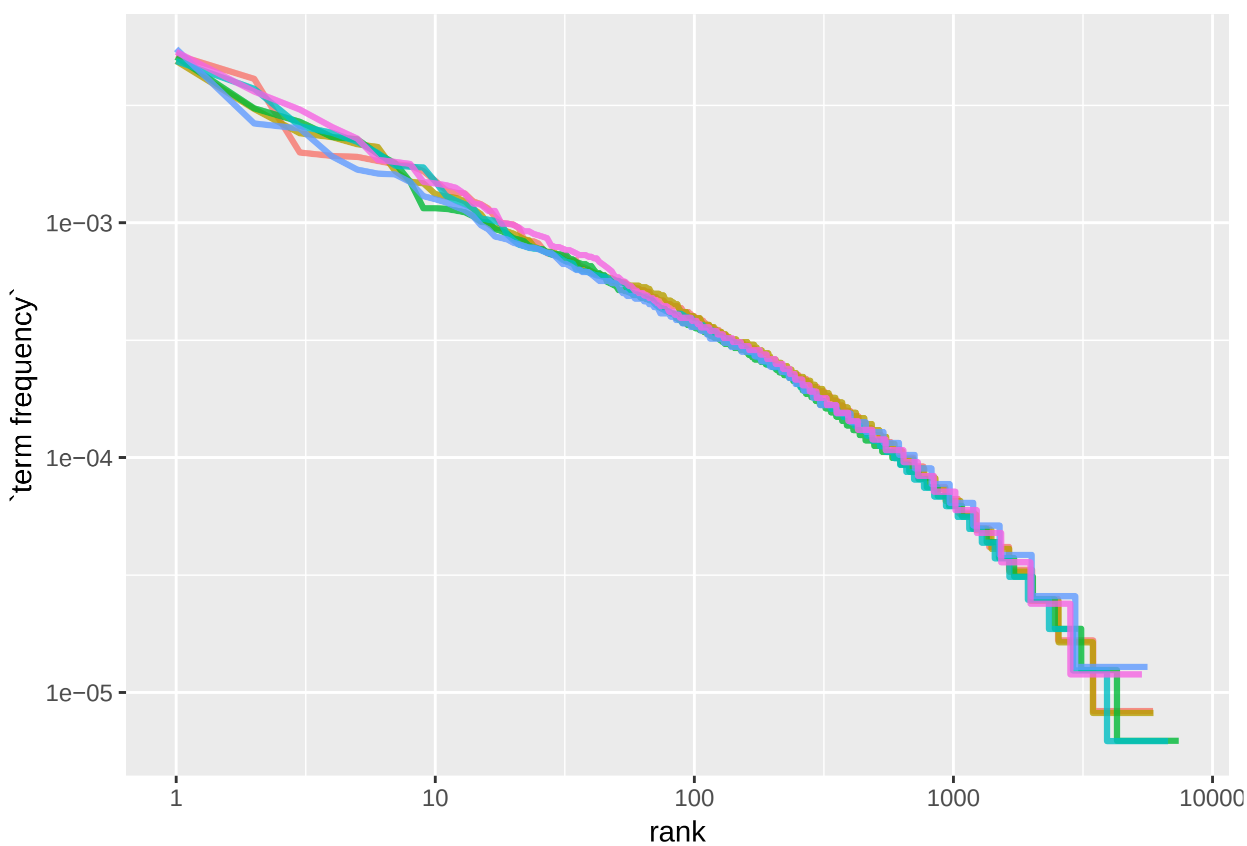
```
freq_by_rank %>% filter(word=="the")
```

```
## # A tibble: 0 x 6
## # Groups:   book [0]
## # ... with 6 variables: book <fct>, word <chr>, n <int>, total <int>,
## #   rank <int>, `term frequency` <dbl>
```

Here, the word “the” accounts for 3.63% of all the words in the six novels.

Zipf ’s law is often visualized by plotting rank on the x-axis and term frequency on the y-axis, on logarithmic scales.

```
freq_by_rank %>%
ggplot(aes(rank, `term frequency`, color = book)) +
  geom_line(size = 1.1, alpha = 0.8, show.legend = FALSE) +
  scale_x_log10() + scale_y_log10()
```

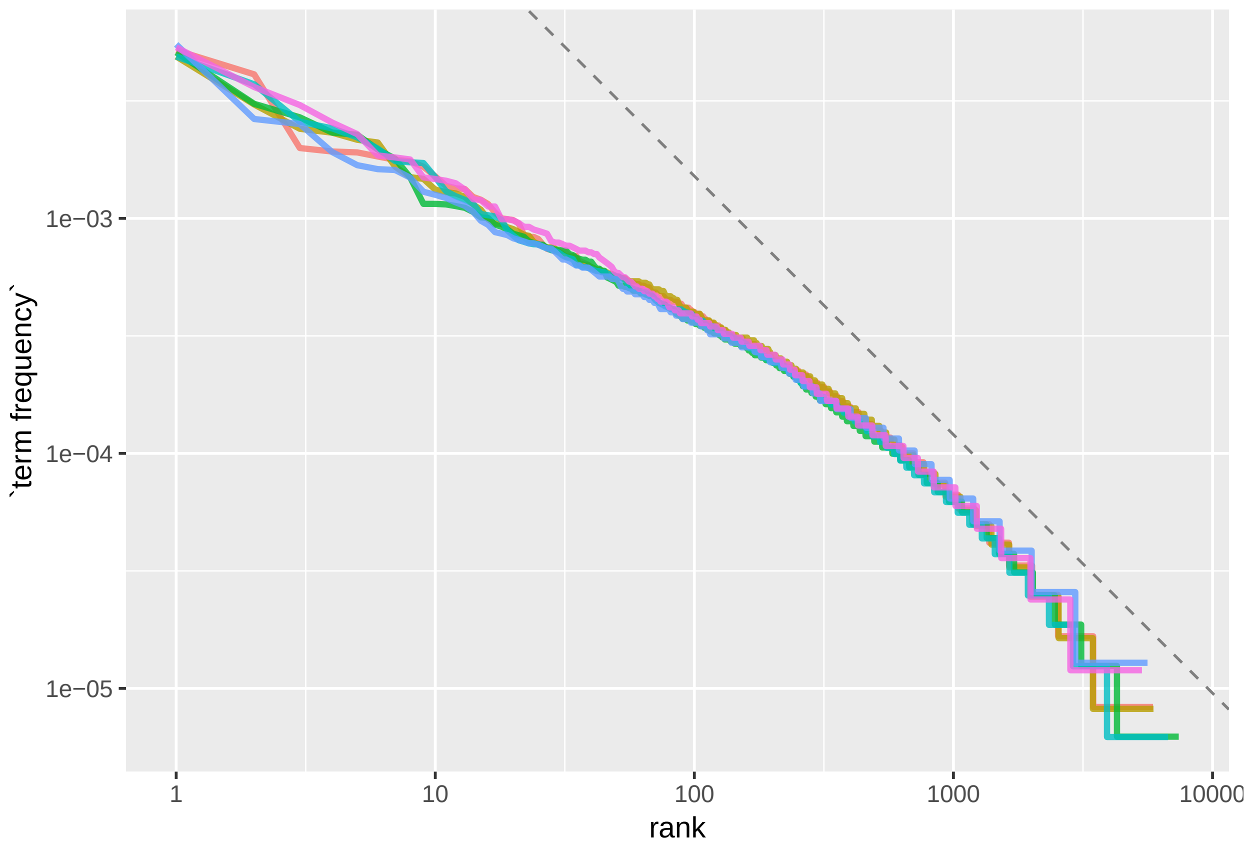


We could view this as a broken power law with three sections. Let's see what the exponent of the power law is for the middle section of the rank range.

```
rank_subset <- freq_by_rank %>%
      filter(rank < 500,
             rank > 10)
    lm(log10(`term frequency`) ~ log10(rank), data = rank_subset)
```

```
##
## Call:
## lm(formula = log10(`term frequency`) ~ log10(rank), data = rank_subset)
##
## Coefficients:
## (Intercept)  log10(rank)
##     -2.1693      -0.6327
```

```
freq_by_rank %>%
ggplot(aes(rank, `term frequency`, color = book)) +
geom_abline(intercept = -0.62, slope = -1.1, color = "gray50", linetype = 2) + geom_line(size = 1.1, alp
scale_x_log10() +
scale_y_log10()
```



Let's calculate the tf-idf of terms in Jane Austen's works.

Consider a document containing 100 words wherein the word cat appears 3 times. The term frequency (i.e., tf) for cat is then $3/100 = 0.03$.

Now, assume we have 10 million documents and the word cat appears in one thousand of these. Then, the inverse document frequency (i.e., idf) is calculated as $ln(10,000,000/1,000) = 4$. The inverse document frequency is important because it measures how important a term is. While computing tf (term frequency), all terms are considered equally important. However, it is known that certain terms, such as "is", "of", and "that", may appear a lot of times but have little importance. Thus we need to weigh down the frequent terms while scale up the rare ones.

Finally, the Tf-idf weight is the product of these quantities: $(0.03)(4) = 0.12$.

```
book_words <- book_words %>%
  bind_tf_idf(word, book, n)
```

```
## Warning in bind_tf_idf.data.frame(., word, book, n): A value for tf_idf is negative:
## Input should have exactly one row per document-term combination.
```

```
book_words
```

```
## # A tibble: 36,904 x 7
##    book                word          n  total      tf    idf   tf_idf
##    <fct>               <chr>     <int>  <int>   <dbl>  <dbl>    <dbl>
##  1 Mansfield Park      fanny       816 160460 0.0172   0.693  0.0119
##  2 Emma                emma        786 160996 0.0169   1.10   0.0186
##  3 Sense & Sensibility elinor      623 119957 0.0172   1.79   0.0307
##  4 Emma                miss        599 160996 0.0129  -0.405 -0.00522
##  5 Pride & Prejudice   elizabeth   597 122204 0.0162   0.693  0.0112
##  6 Mansfield Park      crawford    493 160460 0.0104   1.79   0.0186
##  7 Sense & Sensibility marianne    492 119957 0.0135   1.79   0.0243
##  8 Persuasion          anne        447  83658 0.0175   0.182  0.00320
##  9 Mansfield Park      miss        432 160460 0.00911 -0.405 -0.00370
## 10 Northanger Abbey    catherine   428  77780 0.0180   0.693  0.0125
## # ... with 36,894 more rows
```

```
book_words %>% filter(word=="elinor" | word=="miss")
```

```
## # A tibble: 10 x 7
##    book                word       n  total        tf    idf      tf_idf
##    <fct>               <chr>  <int>  <int>     <dbl>  <dbl>       <dbl>
##  1 Sense & Sensibility elinor   623 119957 0.0172     1.79   0.0307
##  2 Emma                miss     599 160996 0.0129    -0.405 -0.00522
##  3 Mansfield Park      miss     432 160460 0.00911   -0.405 -0.00370
##  4 Pride & Prejudice   miss     283 122204 0.00767   -0.405 -0.00311
##  5 Sense & Sensibility miss     210 119957 0.00578   -0.405 -0.00234
##  6 Northanger Abbey    miss     206  77780 0.00865   -0.405 -0.00351
##  7 Persuasion          miss     125  83658 0.00490   -0.405 -0.00199
##  8 Emma                miss       3 160996 0.0000645 -0.405 -0.0000262
##  9 Mansfield Park      miss       1 160460 0.0000211 -0.405 -0.00000855
## 10 Mansfield Park      miss       1 160460 0.0000211 -0.405 -0.00000855
```

Let's look at terms with high tf-idf in Jane Austen's works. Some of the values for idf are the same for different terms because there are six documents in this corpus and we are seeing the numerical value for $ln(6/1)$, $ln(6/2)$, etc.

```
book_words %>%
      select(-total) %>%
      arrange(desc(tf_idf))
```

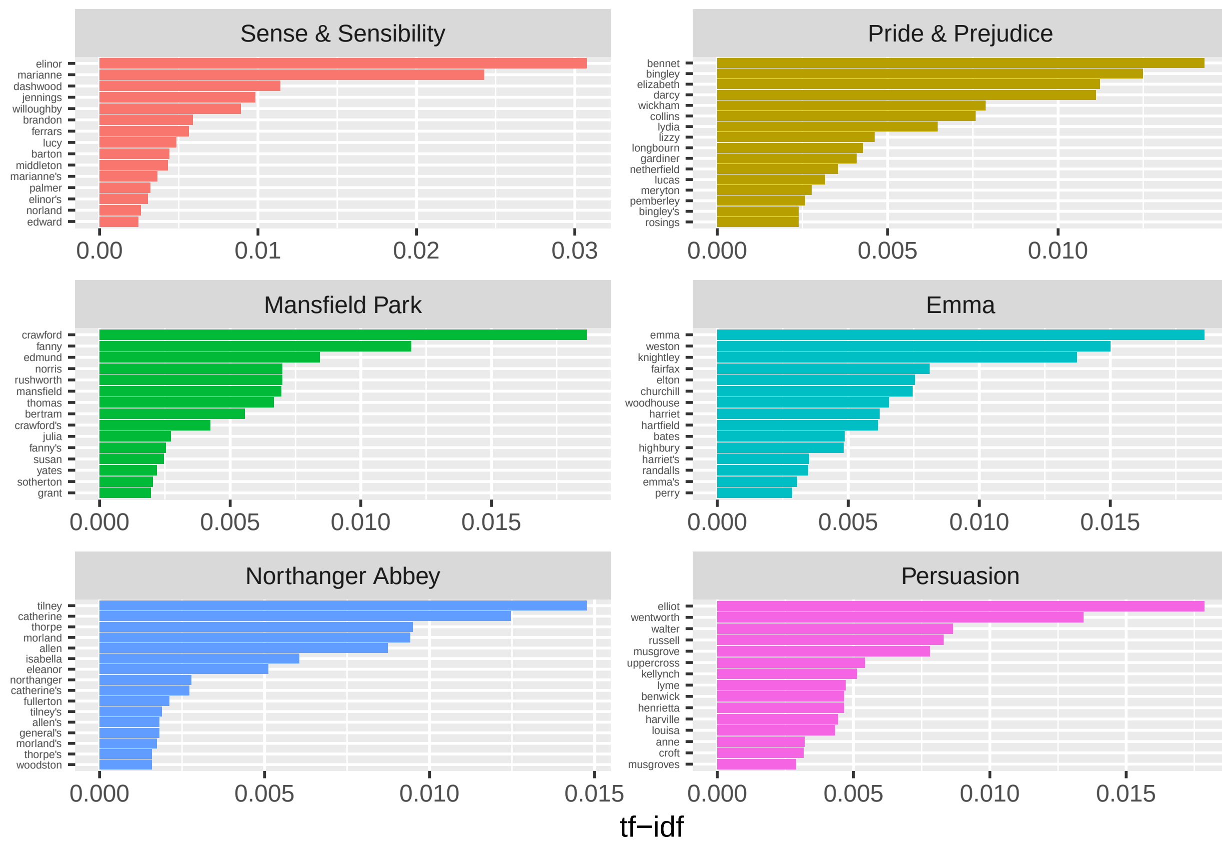
```
## # A tibble: 36,904 x 6
##    book                word          n     tf   idf tf_idf
##    <fct>               <chr>     <int>  <dbl> <dbl>  <dbl>
##  1 Sense & Sensibility elinor      623 0.0172  1.79 0.0307
```

```
##  2 Sense & Sensibility marianne    492 0.0135    1.79 0.0243
##  3 Mansfield Park      crawford    493 0.0104    1.79 0.0186
##  4 Emma                emma        786 0.0169    1.10 0.0186
##  5 Persuasion          elliot      254 0.00997   1.79 0.0179
##  6 Emma                weston      389 0.00837   1.79 0.0150
##  7 Northanger Abbey    tilney      196 0.00823   1.79 0.0148
##  8 Pride & Prejudice   bennet      294 0.00797   1.79 0.0143
##  9 Emma                knightley   356 0.00766   1.79 0.0137
## 10 Persuasion          wentworth   191 0.00749   1.79 0.0134
## # ... with 36,894 more rows
```

Let's visualize this!

```
book_words %>% arrange(desc(tf_idf)) %>%
  mutate(word = factor(word, levels = rev(unique(word)))) %>%
  group_by(book) %>% top_n(15) %>% ungroup %>%
  ggplot(aes(word, tf_idf, fill = book)) +
  geom_col(show.legend = FALSE) +
   theme(axis.text.y=element_text(size=rel(0.5)))+
  labs(x = NULL, y = "tf-idf") +
  facet_wrap(~book, ncol = 2, scales = "free") + coord_flip()
```

```
## Selecting by tf_idf
```

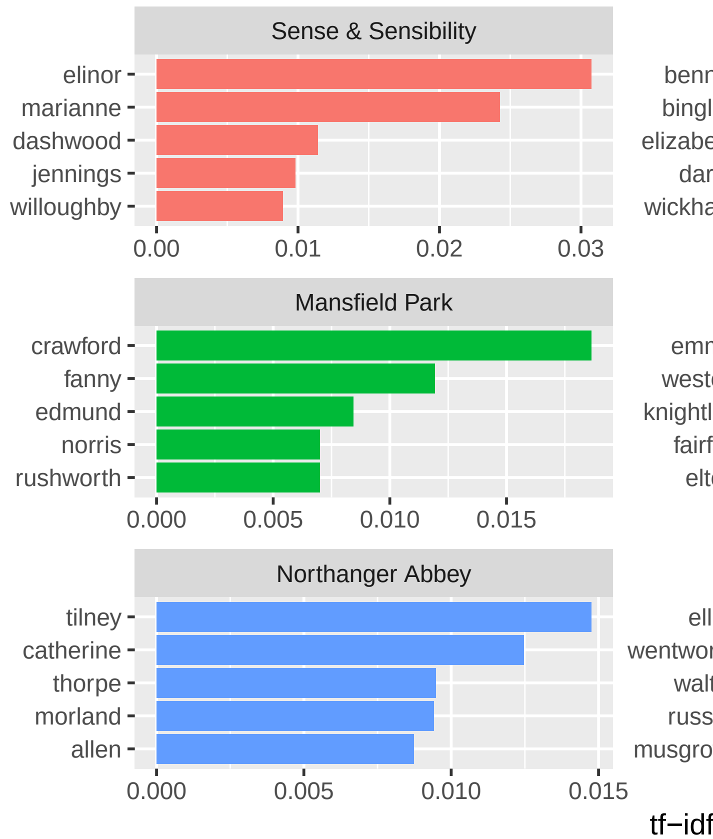


A closer look, only the top 5 tf_idf per novel.

```
book_words %>% arrange(desc(tf_idf)) %>%
  mutate(word = factor(word, levels = rev(unique(word)))) %>%
```

```r
  group_by(book) %>% top_n(5) %>% ungroup %>%
  ggplot(aes(word, tf_idf, fill = book)) +
  geom_col(show.legend = FALSE) +
  labs(x = NULL, y = "tf-idf") +
  facet_wrap(~book, ncol = 2, scales = "free") + coord_flip()
```

```
## Selecting by tf_idf
```



7.3 Sentiment analysis.

When human readers approach a text, we use our understanding of the emotional intent of words to infer whether a section of text is positive or negative, or perhaps characterized by some other more nuanced emotion like surprise or disgust.

There are a variety of methods and dictionaries that exist for evaluating the opinion or emotion in text.

```r
get_sentiments("afinn")
```

```
## # A tibble: 2,477 x 2
##    word       value
##    <chr>      <dbl>
##  1 abandon       -2
##  2 abandoned     -2
##  3 abandons      -2
##  4 abducted      -2
##  5 abduction     -2
##  6 abductions    -2
```

```
## 7 abhor        -3
## 8 abhorred     -3
## 9 abhorrent    -3
## 10 abhors      -3
## # ... with 2,467 more rows
```

```
get_sentiments("bing")
```

```
## # A tibble: 6,786 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
## # ... with 6,776 more rows
```

```
get_sentiments("loughran")
```

```
## # A tibble: 4,150 x 2
##    word         sentiment
##    <chr>        <chr>
##  1 abandon      negative
##  2 abandoned    negative
##  3 abandoning   negative
##  4 abandonment  negative
##  5 abandonments negative
##  6 abandons     negative
##  7 abdicated    negative
##  8 abdicates    negative
##  9 abdicating   negative
## 10 abdication   negative
## # ... with 4,140 more rows
```

All three of these lexicons are based on unigrams, i.e., single words. These lexicons contain many English words and the words are assigned scores for positive/negative sentiment, and also possibly emotions like joy, anger, sadness, and so forth.

Let's set up the data-base again:

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

Now, we apply the lexicon bing for each novel. We do it in blocks of 80 lines.

```r
library(tidyr)
jane_austen_sentiment <- tidy_books %>%
inner_join(get_sentiments("bing")) %>%
count(book, index = linenumber %/% 80, sentiment) %>%
spread(sentiment, n, fill = 0) %>%
mutate(sentiment = positive - negative)
```

```
## Joining, by = "word"
```

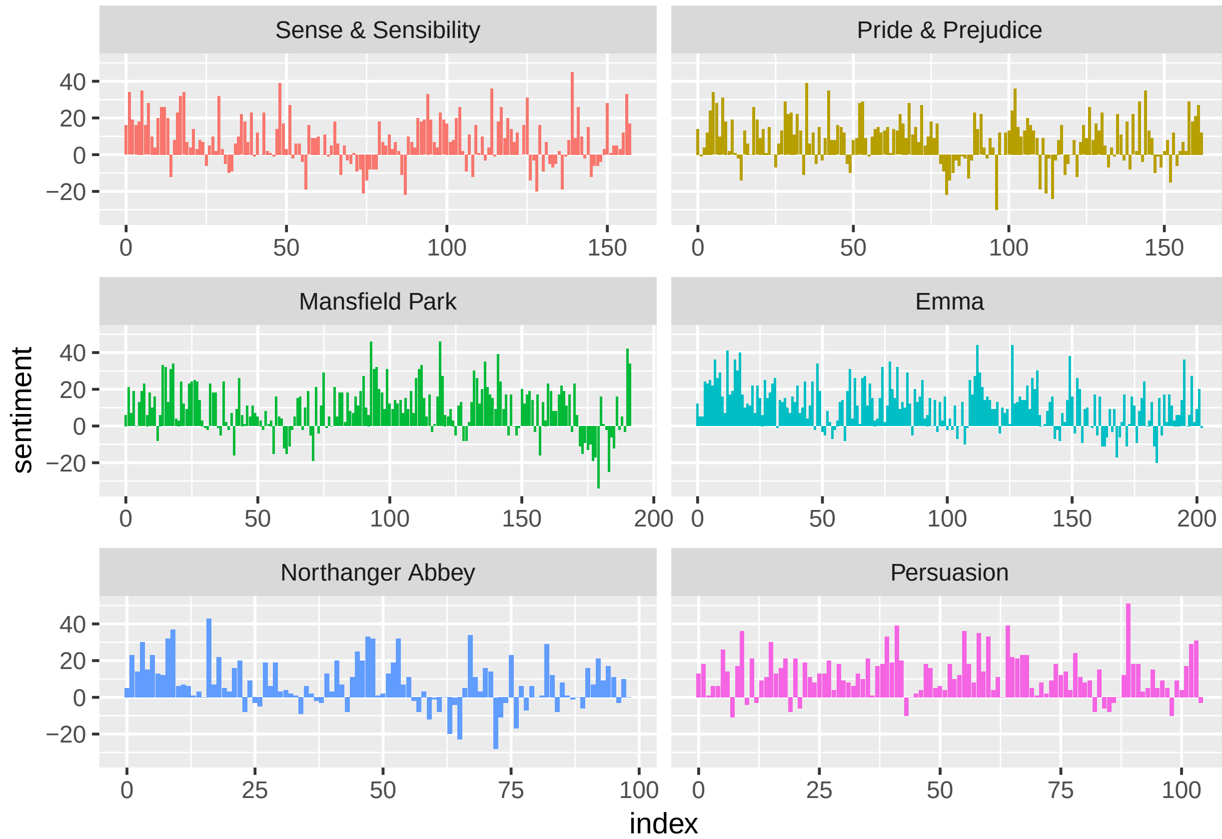
```r
jane_austen_sentiment
```

```
## # A tibble: 920 x 5
##    book                index negative positive sentiment
##    <fct>               <dbl>    <dbl>    <dbl>     <dbl>
##  1 Sense & Sensibility     0       16       32        16
##  2 Sense & Sensibility     1       19       53        34
##  3 Sense & Sensibility     2       12       31        19
##  4 Sense & Sensibility     3       15       31        16
##  5 Sense & Sensibility     4       16       34        18
##  6 Sense & Sensibility     5       16       51        35
##  7 Sense & Sensibility     6       24       40        16
##  8 Sense & Sensibility     7       23       51        28
##  9 Sense & Sensibility     8       30       40        10
## 10 Sense & Sensibility     9       15       19         4
## # ... with 910 more rows
```

The index for Sense & Sensibility goes from 1 to 157. Each index stands for 80 lines, thus $(157)(80) = 12,560$ which coincide with the total number of lines in Sense & Sensibility, which is $12,624$. Then, we have a score of each index (80 lines) based on the lexicon bing for negative and positive. The last column sentiment is simply positive-negative score.

We can show the results in a plot.

```r
ggplot(jane_austen_sentiment, aes(index, sentiment, fill = book)) +
geom_col(show.legend = FALSE) +
facet_wrap(~book, ncol = 2, scales = "free_x")
```



The most frequent words with a high sentiment value are:

```
bing_word_counts <- tidy_books %>%
  inner_join(get_sentiments("bing")) %>%
  count(word, sentiment, sort = TRUE) %>%
  ungroup()
```

```
## Joining, by = "word"
```

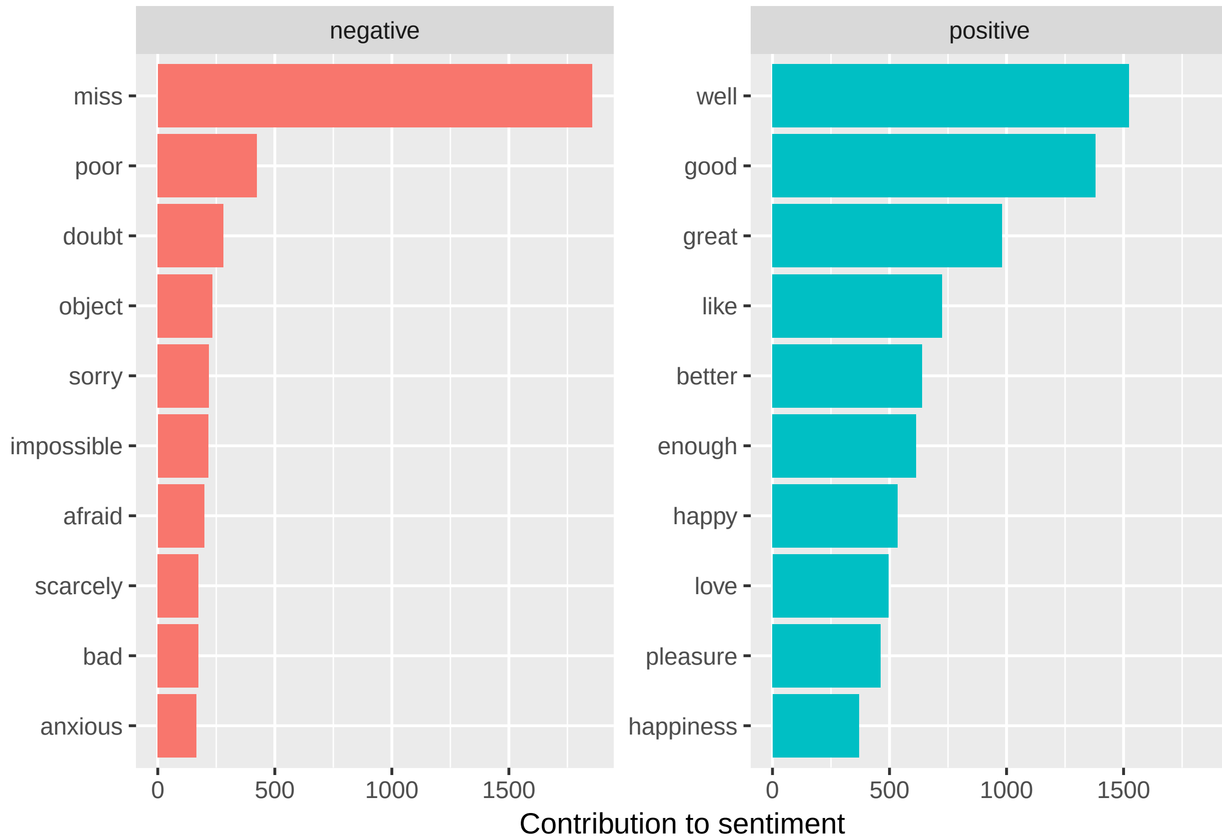
```
bing_word_counts
```

```
## # A tibble: 2,585 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
##  7 enough   positive    613
##  8 happy    positive    534
##  9 love     positive    495
## 10 pleasure positive    462
## # ... with 2,575 more rows
```

We can visualize the top 10 positive and negative words.

```r
bing_word_counts %>%
  group_by(sentiment) %>%
  top_n(10) %>%
  ungroup() %>%
  mutate(word = reorder(word, n)) %>%
  ggplot(aes(word, n, fill = sentiment)) +
  geom_col(show.legend = FALSE) +
  facet_wrap(~sentiment, scales = "free_y") +
  labs(y = "Contribution to sentiment",
       x = NULL) +
  coord_flip()
```

```
## Selecting by n
```



We spot an anomaly in the sentiment analysis. The word "miss" is coded as negative but it is used as a title for young, unmarried women in Jane Austen's works. If it were appropriate for our purposes, we could easily add "miss" to a custom stop-words list using bind_rows(). We could implement that with a strategy such as this.

```r
custom_stop_words <- bind_rows(tibble(word = c("miss"),
                                      lexicon = c("custom")),
                               stop_words)

custom_stop_words
```

```
## # A tibble: 1,150 x 2
##    word        lexicon
```

```
##     <chr>       <chr>
##   1 miss        custom
##   2 a           SMART
##   3 a's         SMART
##   4 able        SMART
##   5 about       SMART
##   6 above       SMART
##   7 according   SMART
##   8 accordingly SMART
##   9 across      SMART
##  10 actually    SMART
## # ... with 1,140 more rows
```

Now, we can remove the word "miss".

```r
data(stop_words)
    bing_word_counts <- bing_word_counts %>%
      mutate(word = str_extract(word, "[a-z']+")) %>%
      anti_join(custom_stop_words)
```

```
## Joining, by = "word"
```

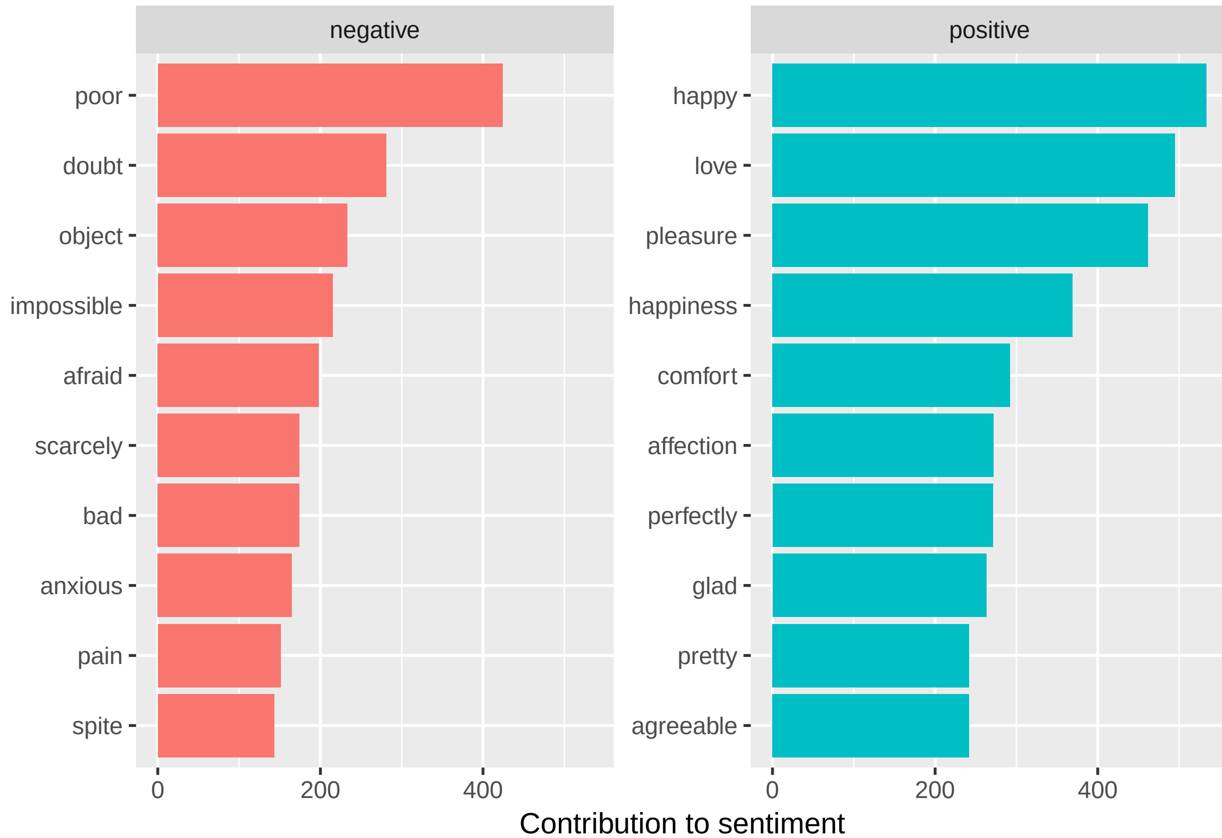
```r
    bing_word_counts
```

```
## # A tibble: 2,554 x 3
##     word      sentiment     n
##     <chr>     <chr>     <int>
##   1 happy     positive    534
##   2 love      positive    495
##   3 pleasure  positive    462
##   4 poor      negative    424
##   5 happiness positive    369
##   6 comfort   positive    292
##   7 doubt     negative    281
##   8 affection positive    272
##   9 perfectly positive    271
##  10 glad      positive    263
## # ... with 2,544 more rows
```

And we can repeat the same figure without the word "miss".

```r
bing_word_counts %>%
  group_by(sentiment) %>%
  top_n(10) %>%
  ungroup() %>%
  mutate(word = reorder(word, n)) %>%
  ggplot(aes(word, n, fill = sentiment)) +
  geom_col(show.legend = FALSE) +
  facet_wrap(~sentiment, scales = "free_y") +
  labs(y = "Contribution to sentiment",
       x = NULL) +
  coord_flip()
```

```
## Selecting by n
```



7.4 Wordclouds.

Let's look at the most common words in Jane Austen's works as a whole again, but this time as a wordcloud.

```
library(wordcloud)
```

```
## Warning: package 'wordcloud' was built under R version 3.5.3

## Loading required package: RColorBrewer

## Warning: package 'RColorBrewer' was built under R version 3.5.2

##
## Attaching package: 'wordcloud'

## The following object is masked from 'package:PerformanceAnalytics':
##
##     textplot
```

```
tidy_books %>%
    anti_join(stop_words) %>%
  count(word) %>%
  with(wordcloud(word, n, max.words = 100,scale=c(4,.2)))
```

```
## Joining, by = "word"
```


```r
library(wordcloud)

tidy_books %>%
    anti_join(custom_stop_words) %>%
  count(word) %>%
  with(wordcloud(word, n, max.words = 100,scale=c(3,.2)))
```

```
## Joining, by = "word"
```


```
library(reshape2)
```

```
## Warning: package 'reshape2' was built under R version 3.5.2

##
## Attaching package: 'reshape2'

## The following object is masked from 'package:tidyr':
##
##     smiths
```

```
tidy_books %>%
  inner_join(get_sentiments("bing")) %>%
  count(word, sentiment, sort = TRUE) %>%
  acast(word ~ sentiment, value.var = "n", fill = 0) %>%
  comparison.cloud(colors = c("gray20", "gray80"),
                   max.words = 100)
```

```
## Joining, by = "word"
```

7.5 Bigrams and trigrams.

We have been using the unnest_tokens function to tokenize by word, or sometimes by sentence, which is useful for the kinds of sentiment and frequency analyses. But we can also use the function to tokenize into consecutive sequences of words, called n-grams. By seeing how often word X is followed by word Y, we can then build a model of the relationships between them.

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

```
## # A tibble: 725,049 x 2
##    book                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
```

Let's examine the most common bigrams using dplyr's count().

```r
austen_bigrams %>%
  count(bigram, sort = TRUE)
```

```
## # A tibble: 211,236 x 2
##    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
```

let's separate bigram into two columns, "word1" and "word2," at which point we can remove cases where either is a stop word. To do so, we use tidyr's separate(), which splits a column into multiple columns based on a delimiter.

```r
    bigrams_separated <- austen_bigrams %>%
      separate(bigram, c("word1", "word2"), sep = " ")
    bigrams_filtered <- bigrams_separated %>%
      filter(!word1 %in% stop_words$word) %>%
      filter(!word2 %in% stop_words$word)
    # new bigram counts:
bigram_counts <- bigrams_filtered %>% count(word1, word2, sort = TRUE)
    bigram_counts
```

```
## # 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
```

Let's recombine the columns into one. To do so, we use tidyr's unite() function.

```r
bigrams_united <- bigrams_filtered %>%
      unite(bigram, word1, word2, sep = " ")
bigrams_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
```

Same process for the most common trigrams.

```r
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
```

What are the most common “streets” mentioned in each book?

```r
bigrams_filtered %>% filter(word2 == "street") %>%
  count(book, word1, sort = TRUE)
```

```
## # A tibble: 34 x 3
##    book                word1           n
##    <fct>               <chr>       <int>
##  1 Sense & Sensibility berkeley       16
##  2 Sense & Sensibility harley         16
##  3 Northanger Abbey    pulteney       14
##  4 Northanger Abbey    milsom         11
##  5 Mansfield Park      wimpole        10
##  6 Pride & Prejudice   gracechurch     9
##  7 Sense & Sensibility conduit         6
##  8 Sense & Sensibility bond            5
##  9 Persuasion          milsom          5
## 10 Persuasion          rivers          4
## # ... with 24 more rows
```

Let’s look at the tf-idf of bigrams across Austen novels.

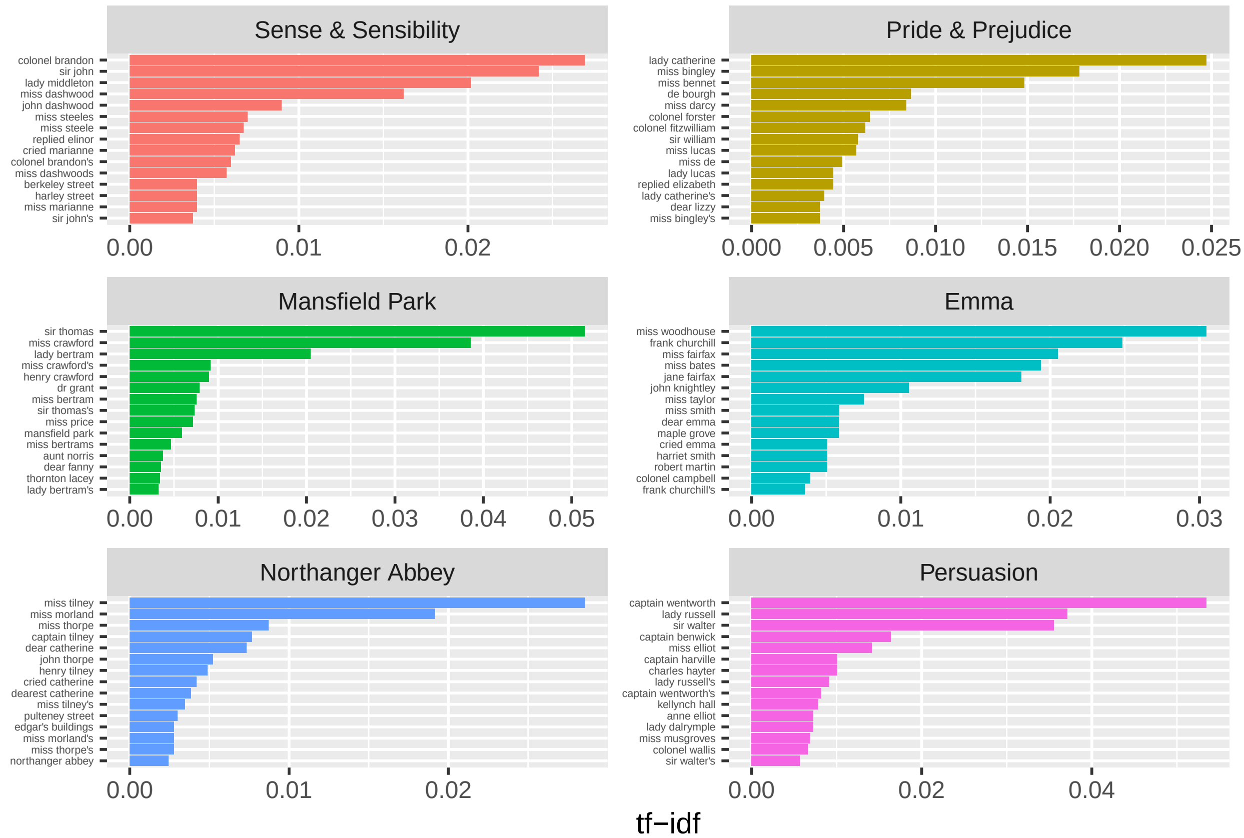
```r
bigram_tf_idf <- bigrams_united %>%
      count(book, bigram) %>%
      bind_tf_idf(bigram, book, n) %>%
      arrange(desc(tf_idf))
    bigram_tf_idf
```

```
## # A tibble: 36,217 x 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
```

Let's visualize the bigrams with the highest tf-idf across Austen novels.

```r
library(ggplot2)
bigram_tf_idf %>%
arrange(desc(tf_idf)) %>%
mutate(bigram = factor(bigram, levels = rev(unique(bigram)))) %>% group_by(book) %>%
top_n(15) %>%
ungroup %>%
ggplot(aes(bigram, tf_idf, fill = book)) + geom_col(show.legend = FALSE) +
labs(x = NULL, y = "tf-idf") +
  theme(axis.text.y=element_text(size=rel(0.5)))+
facet_wrap(~book, ncol = 2, scales = "free") + coord_flip()
```

```
## Selecting by tf_idf
```



7.6 Correlation analysis.

Counting and Correlating pairs of Words with the widyr package. Consider the book "Pride and Prejudice" divided into 10-line sections, as we did (with larger sections) for sentiment analysis. We may be interested in what words tend to appear within the same section.

Let's see what words tend to appear within the same section.

```
austen_section_words <- austen_books() %>%
     filter(book == "Pride & Prejudice") %>%
     mutate(section = row_number() %/% 10) %>%
     filter(section > 0) %>%
     unnest_tokens(word, text) %>%
     filter(!word %in% stop_words$word)
austen_section_words
```

```
## # A tibble: 37,240 x 3
##    book              section word
##    <fct>               <dbl> <chr>
##  1 Pride & Prejudice       1 truth
##  2 Pride & Prejudice       1 universally
##  3 Pride & Prejudice       1 acknowledged
##  4 Pride & Prejudice       1 single
##  5 Pride & Prejudice       1 possession
##  6 Pride & Prejudice       1 fortune
##  7 Pride & Prejudice       1 wife
##  8 Pride & Prejudice       1 feelings
```

```
##  9 Pride & Prejudice       1 views
## 10 Pride & Prejudice       1 entering
## # ... with 37,230 more rows
```

let's count common pairs of words co-appearing within the same section.

```r
library(widyr)
```

```
## Warning: package 'widyr' was built under R version 3.5.3
```

```r
    # count words co-occuring within sections
word_pairs <- austen_section_words %>% pairwise_count(word, section, sort = TRUE)
word_pairs
```

```
## # A tibble: 796,008 x 3
##    item1     item2         n
##    <chr>     <chr>     <dbl>
##  1 darcy     elizabeth   144
##  2 elizabeth darcy       144
##  3 miss      elizabeth   110
##  4 elizabeth miss        110
##  5 elizabeth jane        106
##  6 jane      elizabeth   106
##  7 miss      darcy        92
##  8 darcy     miss         92
##  9 elizabeth bingley      91
## 10 bingley   elizabeth    91
## # ... with 795,998 more rows
```

Let's find the words that most often occur with Darcy.

```r
word_pairs %>%
      filter(item1 == "darcy")
```

```
## # A tibble: 2,930 x 3
##    item1 item2         n
##    <chr> <chr>     <dbl>
##  1 darcy elizabeth   144
##  2 darcy miss         92
##  3 darcy bingley      86
##  4 darcy jane         46
##  5 darcy bennet       45
##  6 darcy sister       45
##  7 darcy time         41
##  8 darcy lady         38
##  9 darcy friend       37
## 10 darcy wickham      37
## # ... with 2,920 more rows
```

7.7 Examining pairwise correlation.

Let's find the phi coefficient between words based on how often they appear in the same section.The focus of the phi coefficient is how much more likely it is that either both word X and Y appear, or neither do, than that one appears without the other.

```r
word_cors <- austen_section_words %>% group_by(word) %>%
filter(n() >= 20) %>% pairwise_cor(word, section, sort = TRUE)
word_cors
```

```
## # A tibble: 154,842 x 3
##    item1     item2     correlation
##    <chr>     <chr>           <dbl>
##  1 bourgh    de              0.951
##  2 de        bourgh          0.951
##  3 pounds    thousand        0.701
##  4 thousand  pounds          0.701
##  5 william   sir             0.664
##  6 sir       william         0.664
##  7 catherine lady            0.663
##  8 lady      catherine       0.663
##  9 forster   colonel         0.622
## 10 colonel   forster         0.622
## # ... with 154,832 more rows
```

We can find the words most correlated with a word like "pounds" using a filter operation.

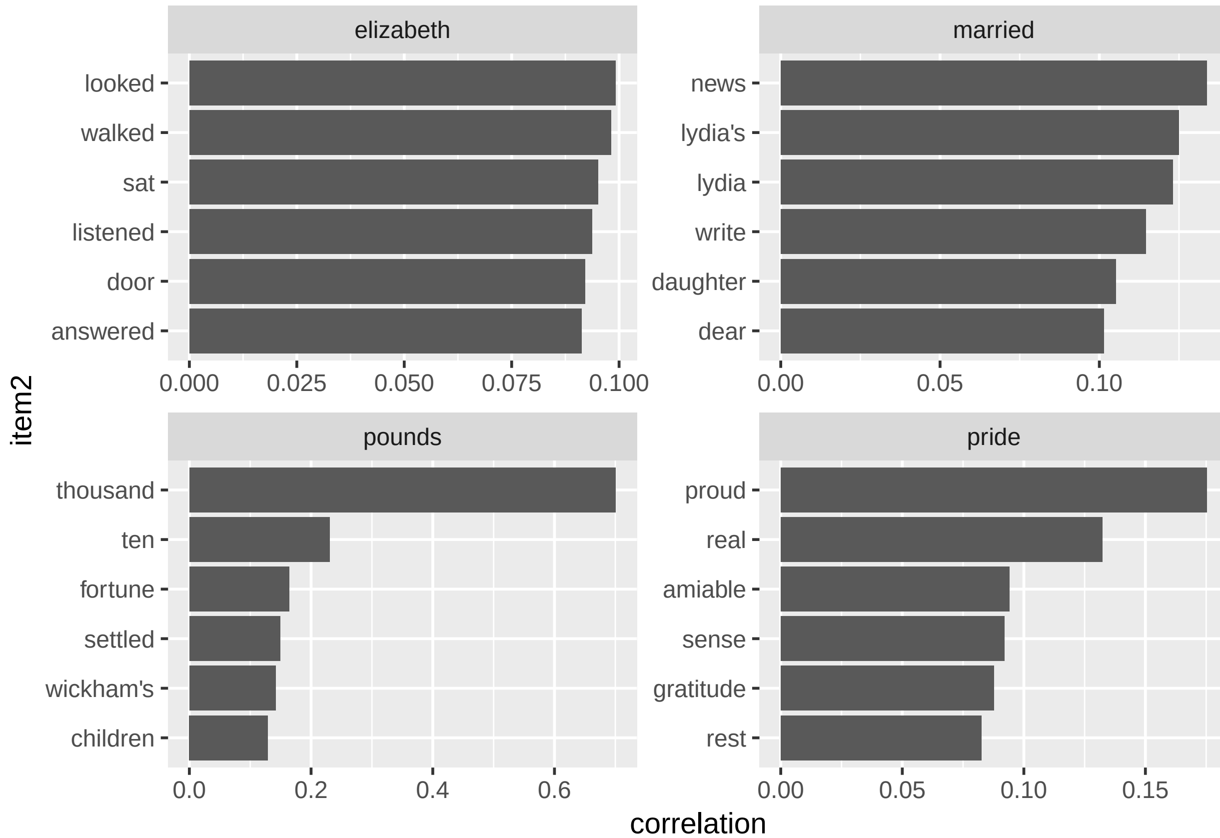
```
word_cors %>%
      filter(item1 == "pounds")
```

```
## # A tibble: 393 x 3
##    item1  item2     correlation
##    <chr>  <chr>           <dbl>
##  1 pounds thousand       0.701
##  2 pounds ten            0.231
##  3 pounds fortune        0.164
##  4 pounds settled        0.149
##  5 pounds wickham's      0.142
##  6 pounds children       0.129
##  7 pounds mother's       0.119
##  8 pounds believed       0.0932
##  9 pounds estate         0.0890
## 10 pounds ready          0.0860
## # ... with 383 more rows
```

Let's pick particular interesting words and find the other words most associated with them.

```
word_cors %>%
      filter(item1 %in% c("elizabeth", "pounds", "married", "pride")) %>%
      group_by(item1) %>%
      top_n(6) %>%
      ungroup() %>%
      mutate(item2 = reorder(item2, correlation)) %>%
      ggplot(aes(item2, correlation)) +
      geom_bar(stat = "identity") +
      facet_wrap(~ item1, scales = "free") +
      coord_flip()
```

```
## Selecting by correlation
```



8 Jane Austen, H.G. Wells, Brontë sisters.

The gutenbergr package provides access to the public domain works from the Project Gutenberg collection. We can access these works using gutenberg_down load() and the Project Gutenberg ID numbers for each novel. Here we extract works by H.G. Wells.

We need some preliminary code.

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

```r
tidy_books <- original_books %>%
  unnest_tokens(word, text)
data(stop_words)
tidy_books <- tidy_books %>%
  anti_join(stop_words)
```

```
## Joining, by = "word"
```

```r
library(gutenbergr)
```

```
## Warning: package 'gutenbergr' was built under R version 3.5.3
```

```r
hgwells <- gutenberg_download(c(35, 36, 5230, 159))
```

```
## Determining mirror for Project Gutenberg from http://www.gutenberg.org/robot/harvest
```

```
## Using mirror http://aleph.gutenberg.org
```

```r
tidy_hgwells <- hgwells %>%
  unnest_tokens(word, text) %>%
  anti_join(stop_words)
```

```
## Joining, by = "word"
```

8.1 Common words of different authors.

What are the most common words in these novels of H.G. Wells?

```r
  tidy_hgwells %>%
  count(word, sort = TRUE)
```

```
## # A tibble: 11,769 x 2
##    word       n
##    <chr>  <int>
##  1 time     454
##  2 people   302
##  3 door     260
##  4 heard    249
##  5 black    232
##  6 stood    229
##  7 white    222
##  8 hand     218
##  9 kemp     213
## 10 eyes     210
## # ... with 11,759 more rows
```

What are the most common words in these novels of the Brontë sisters?

```
bronte <- gutenberg_download(c(1260, 768, 969, 9182, 767))
tidy_bronte <- bronte %>%
  unnest_tokens(word, text) %>%
  anti_join(stop_words)
```

```
## Joining, by = "word"
```

```
tidy_bronte %>%
  count(word, sort = TRUE)
```

```
## # A tibble: 23,050 x 2
##    word       n
##    <chr>  <int>
##  1 time    1065
##  2 miss     855
##  3 day      827
##  4 hand     768
##  5 eyes     713
##  6 night    647
##  7 heart    638
##  8 looked   601
##  9 door     592
## 10 half     586
## # ... with 23,040 more rows
```

Let's calculate the frequency for each word in the works of Jane Austen, the Brontë's sisters, and H.G. Wells by binding the data frames together.

We use str_extract() here because the UTF-8 encoded texts from Project Gutenberg have some examples of words with underscores around them to indicate emphasis (like italics).

```
frequency <- bind_rows(mutate(tidy_bronte, author = "Brontë Sisters"),
                       mutate(tidy_hgwells, author = "H.G. Wells"),
                       mutate(tidy_books, author = "Jane Austen")) %>%
  mutate(word = str_extract(word, "[a-z']+")) %>%
  count(author, word) %>%
  group_by(author) %>%
  mutate(proportion = n / sum(n)) %>%
  select(-n) %>%
  spread(author, proportion) %>%
  gather(author, proportion, `Brontë Sisters`:`H.G. Wells`)
```

8.2 Visualize patterns.

Words that are close to the line in these plots have similar frequencies in both sets of texts. Words that are far from the line are words that are found more in one set of texts than another. These characteristics indicate that Austen and the Brontë sisters use more similar words than Austen and H.G. Wells. Also, we see that not all the words are found in all three sets of texts and there are fewer data points in the panel for Austen and H.G. Wells.

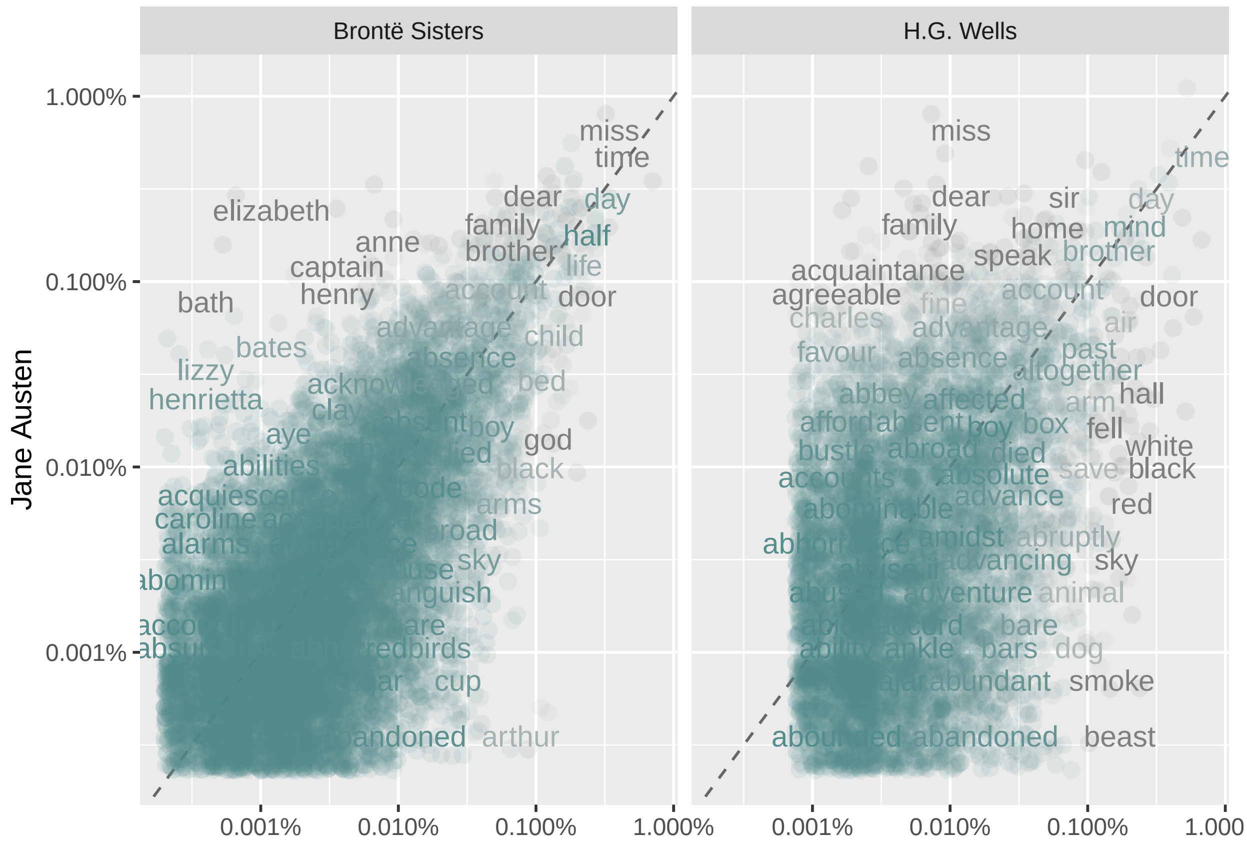
```
library(scales)
```

```
## Warning: package 'scales' was built under R version 3.5.2
##
## Attaching package: 'scales'
## The following object is masked from 'package:purrr':
```

```
##
##     discard

## The following object is masked from 'package:readr':
##
##     col_factor
```

```
# expect a warning about rows with missing values being removed
ggplot(frequency, aes(x = proportion, y = `Jane Austen`, color = abs(`Jane Austen` - proportion))) +
  geom_abline(color = "gray40", lty = 2) +
  geom_jitter(alpha = 0.1, size = 2.5, width = 0.3, height = 0.3) +
  geom_text(aes(label = word), check_overlap = TRUE, vjust = 1.5) +
  scale_x_log10(labels = percent_format()) +
  scale_y_log10(labels = percent_format()) +
  scale_color_gradient(limits = c(0, 0.001), low = "darkslategray4", high = "gray75") +
  facet_wrap(~author, ncol = 2) +
  theme(legend.position="none") +
  labs(y = "Jane Austen", x = NULL)
```

```
## Warning: Removed 41357 rows containing missing values (geom_point).

## Warning: Removed 41359 rows containing missing values (geom_text).
```



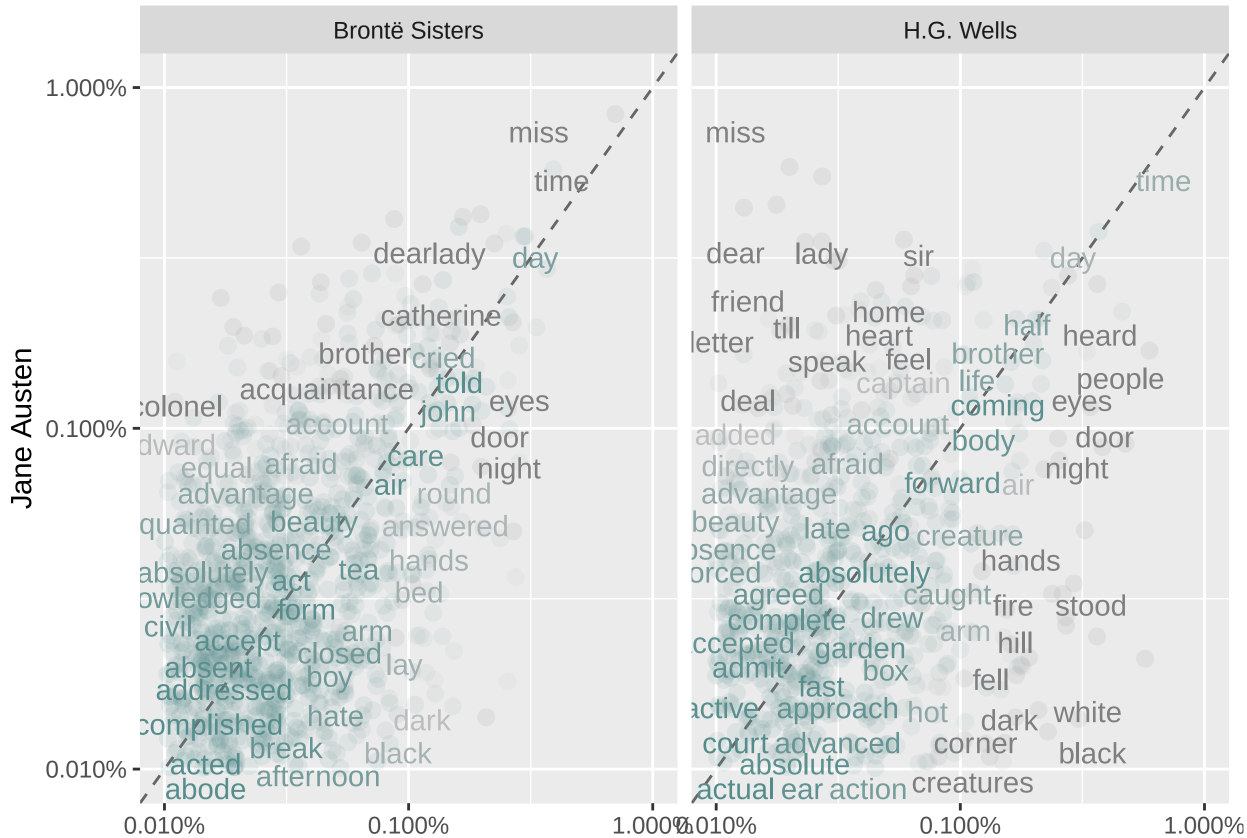
A closer look.

```
    ggplot(frequency, aes(x = proportion, y = `Jane Austen`,
                          color = abs(`Jane Austen` - proportion))) +
geom_abline(color = "gray40", lty = 2) +
geom_jitter(alpha = 0.1, size = 2.5, width = 0.3, height = 0.3) + geom_text(aes(label = word), check_ove
```

```r
scale_y_log10(labels = percent_format(), limits=c(0.0001,.01)) + scale_color_gradient(limits = c(0, 0.00
                                   low = "darkslategray4", high = "gray75") +
      facet_wrap(~author, ncol = 2) +
theme(legend.position="none") + labs(y = "Jane Austen", x = NULL)
```

```
## Warning: Removed 56090 rows containing missing values (geom_point).
```

```
## Warning: Removed 55747 rows containing missing values (geom_text).
```



8.3 Quantify patterns.

Let's quantify how similar and different these sets of word frequencies are using a correlation test.

```r
cor.test(data = frequency[frequency$author == "Brontë Sisters",],
        ~ proportion + `Jane Austen`)
```

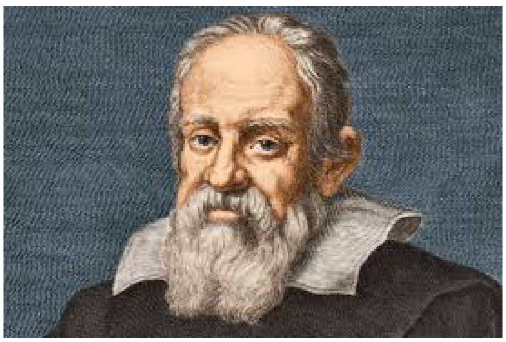
```
##
##  Pearson's product-moment correlation
##
## data:  proportion and Jane Austen
## t = 119.65, df = 10404, p-value < 2.2e-16
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.7527869 0.7689642
## sample estimates:
##       cor
## 0.7609938
```

```
cor.test(data = frequency[frequency$author == "H.G. Wells",],
         ~ proportion + `Jane Austen`)
```

```
##
##  Pearson's product-moment correlation
##
## data:  proportion and Jane Austen
## t = 36.441, df = 6053, p-value < 2.2e-16
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.4032800 0.4445987
## sample estimates:
##       cor
## 0.4241601
```

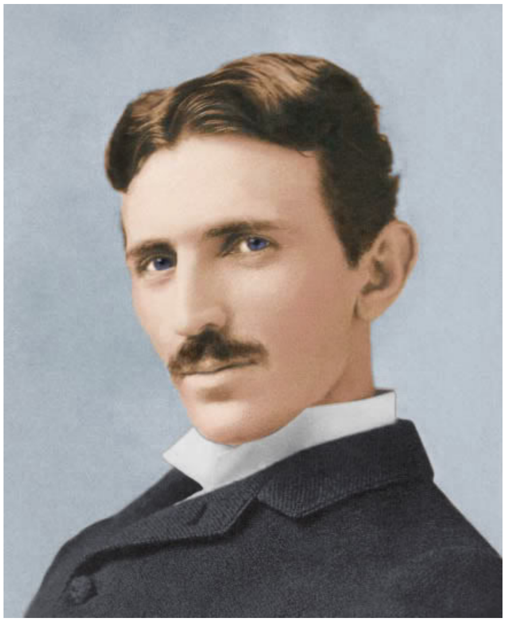
9 Galileo Galilei, Christiaan Huygens, Nikola Tesla.

Let's start by downloading three books, by Galileo Galilei:



Christiaan Huygens:

Nikola Tesla:



```r
library(gutenbergr)
    physics <- gutenberg_download(c(37729, 14725, 13476),
meta_fields = "author")
```

The names of the books are: Discourse on Floating Bodies (Galileo Galilei 1612), Treatise on Light (Christiaan Huygens 1690), Experiments with Alternate Currents of High Potential and High Frequency (Nikola Tesla 1892).

let's use unnest_tokens() and count() to find out how many times each word is used in each text.

```r
physics_words <- physics %>%
  unnest_tokens(word, text) %>%
  count(author, word, sort = TRUE) %>% ungroup()
    physics_words
```

```
## # A tibble: 9,673 x 3
##    author             word      n
##    <chr>              <chr> <int>
##  1 Galilei, Galileo   the    3760
##  2 Tesla, Nikola      the    3604
##  3 Huygens, Christiaan the    3553
##  4 Galilei, Galileo   of     2049
##  5 Tesla, Nikola      of     1737
##  6 Huygens, Christiaan of     1708
##  7 Huygens, Christiaan to     1207
##  8 Tesla, Nikola      a      1176
##  9 Galilei, Galileo   and    1148
## 10 Galilei, Galileo   to     1133
## # ... with 9,663 more rows
```

```r
data(stop_words)
    physics_words <- physics_words %>% mutate(word = str_extract(word, "[a-z']+")) %>%
      anti_join(stop_words)
```

```
## Joining, by = "word"
```

```r
physics_words
```

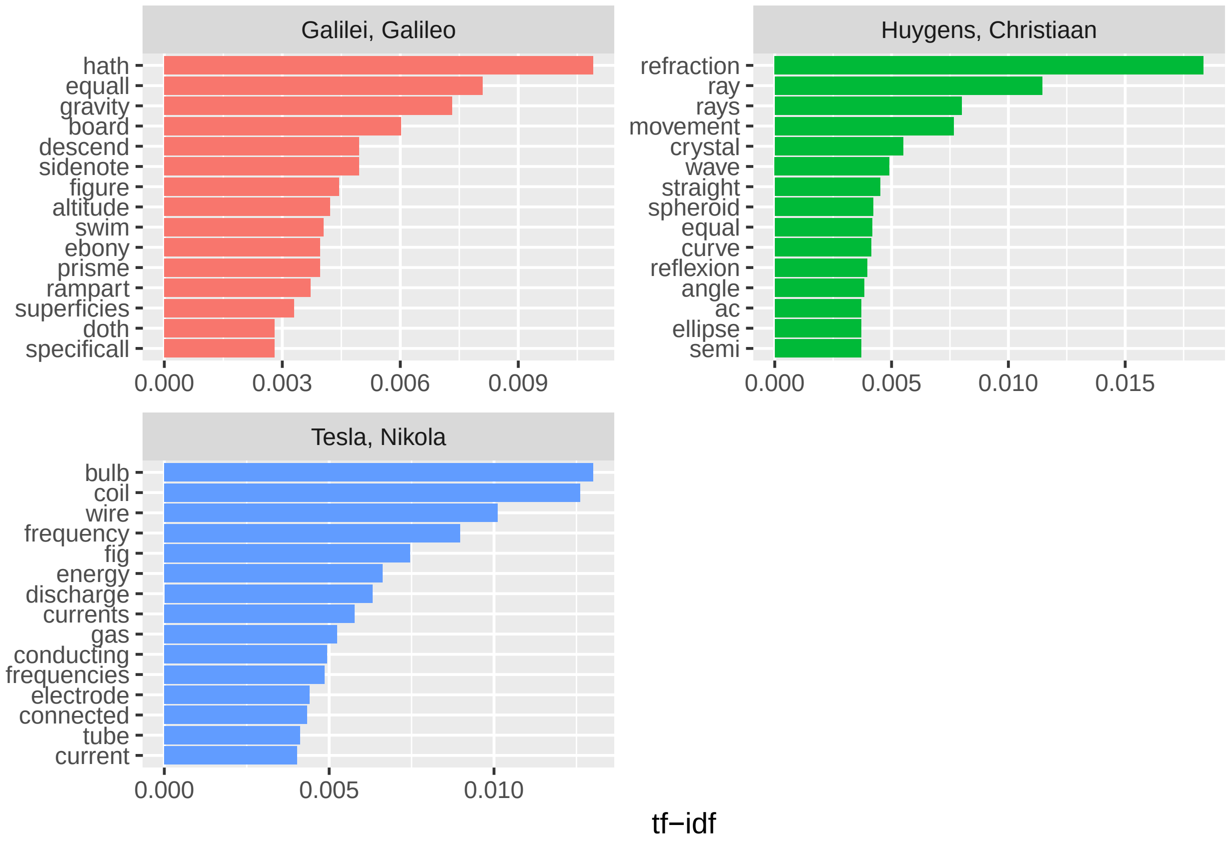
```
## # A tibble: 8,209 x 3
##    author              word           n
##    <chr>               <chr>      <int>
##  1 Galilei, Galileo    water        828
##  2 Galilei, Galileo    gravity      240
##  3 Huygens, Christiaan refraction   218
##  4 Galilei, Galileo    air          211
##  5 Galilei, Galileo    mass         208
##  6 Huygens, Christiaan light        201
##  7 Huygens, Christiaan line         198
##  8 Galilei, Galileo    solid        192
##  9 Huygens, Christiaan crystal      177
## 10 Tesla, Nikola       bulb         171
## # ... with 8,199 more rows
```

Let's go ahead and calculate tf-idf, then visualize the high tf-idf words.

```r
plot_physics <- physics_words %>%
      bind_tf_idf(word, author, n) %>%
      arrange(desc(tf_idf)) %>%
mutate(word = factor(word, levels = rev(unique(word)))) %>%
  mutate(author = factor(author, levels = c("Galilei, Galileo",
                                            "Huygens, Christiaan",
                                            "Tesla, Nikola")))
```

```
## Warning in bind_tf_idf.data.frame(., word, author, n): A value for tf_idf is negative:
## Input should have exactly one row per document-term combination.
```

```r
plot_physics %>%
group_by(author) %>%
top_n(15, tf_idf) %>%
ungroup() %>%
mutate(word = reorder(word, tf_idf)) %>% ggplot(aes(word, tf_idf, fill = author)) + geom_col(show.legend
labs(x = NULL, y = "tf-idf") + facet_wrap(~author, ncol = 2, scales = "free") + coord_flip()
```

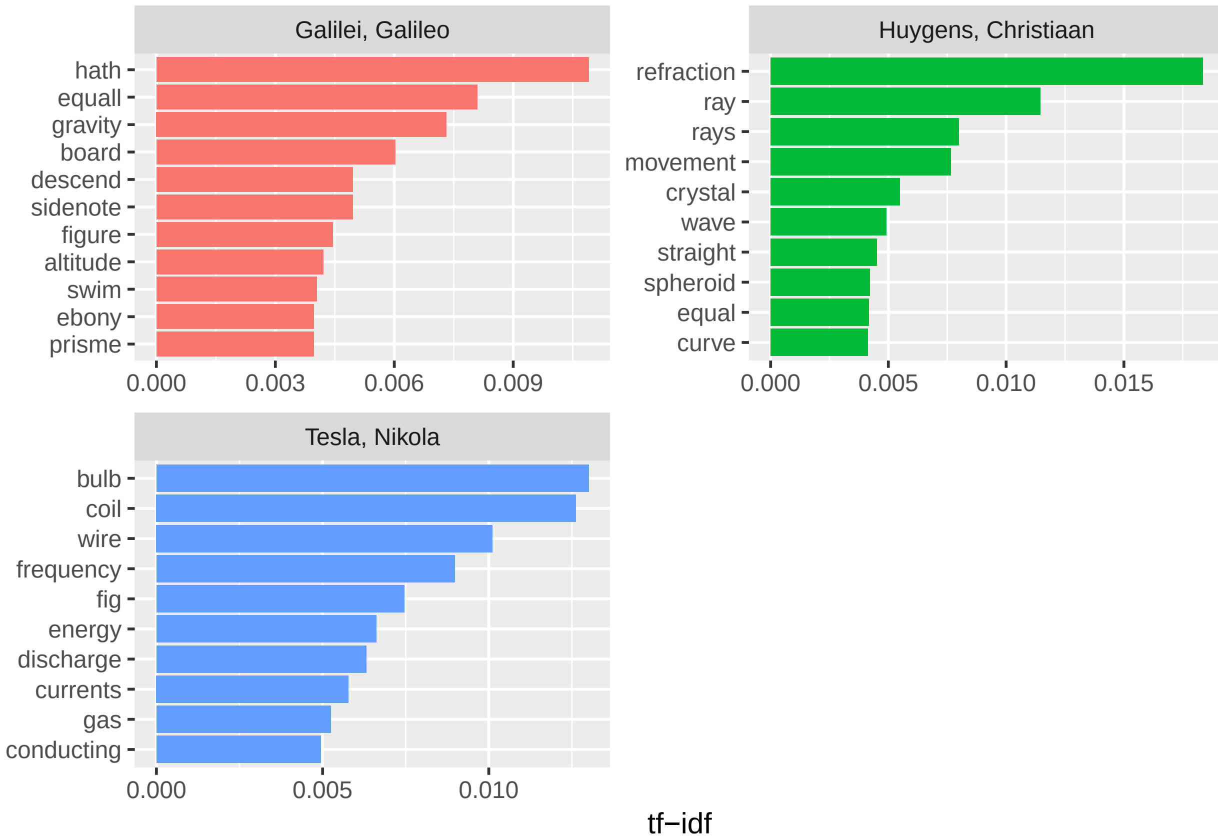


Closer look. Top 10.

```
plot_physics <- physics_words %>%
      bind_tf_idf(word, author, n) %>%
      arrange(desc(tf_idf)) %>%
mutate(word = factor(word, levels = rev(unique(word)))) %>%
  mutate(author = factor(author, levels = c("Galilei, Galileo",
                                            "Huygens, Christiaan",
                                            "Tesla, Nikola")))
```

```
## Warning in bind_tf_idf.data.frame(., word, author, n): A value for tf_idf is negative:
## Input should have exactly one row per document-term combination.
```

```
plot_physics %>%
group_by(author) %>%
top_n(10, tf_idf) %>%
ungroup() %>%
mutate(word = reorder(word, tf_idf)) %>% ggplot(aes(word, tf_idf, fill = author)) + geom_col(show.legen
labs(x = NULL, y = "tf-idf") + facet_wrap(~author, ncol = 2, scales = "free") + coord_flip()
```



"AK", "AB" "RC" and so forth are names of rays, circles, angles, and so on for Huygens.

```
library(stringr)
physics %>%
      filter(str_detect(text, "AK")) %>%
      select(text)
```

```
## # A tibble: 34 x 1
##    text
##    <chr>
##  1 Now let us assume that the ray has come from A to C along AK, KC; the
##  2 be equal to the time along KMN. But the time along AK is longer than
##  3 that along AL: hence the time along AKN is longer than that along ABC.
##  4 And KC being longer than KN, the time along AKC will exceed, by as
##  5 line which is comprised between the perpendiculars AK, BL. Then it
##  6 ordinary refraction. Now it appears that AK and BL dip down toward the
##  7 side where the air is less easy to penetrate: for AK being longer than
##  8 than do AK, BL. And this suffices to show that the ray will continue
##  9 surface AB at the points AK_k_B. Then instead of the hemispherical
## 10 along AL, LB, and along AK, KB, are always represented by the line AH,
## # ... with 24 more rows
```

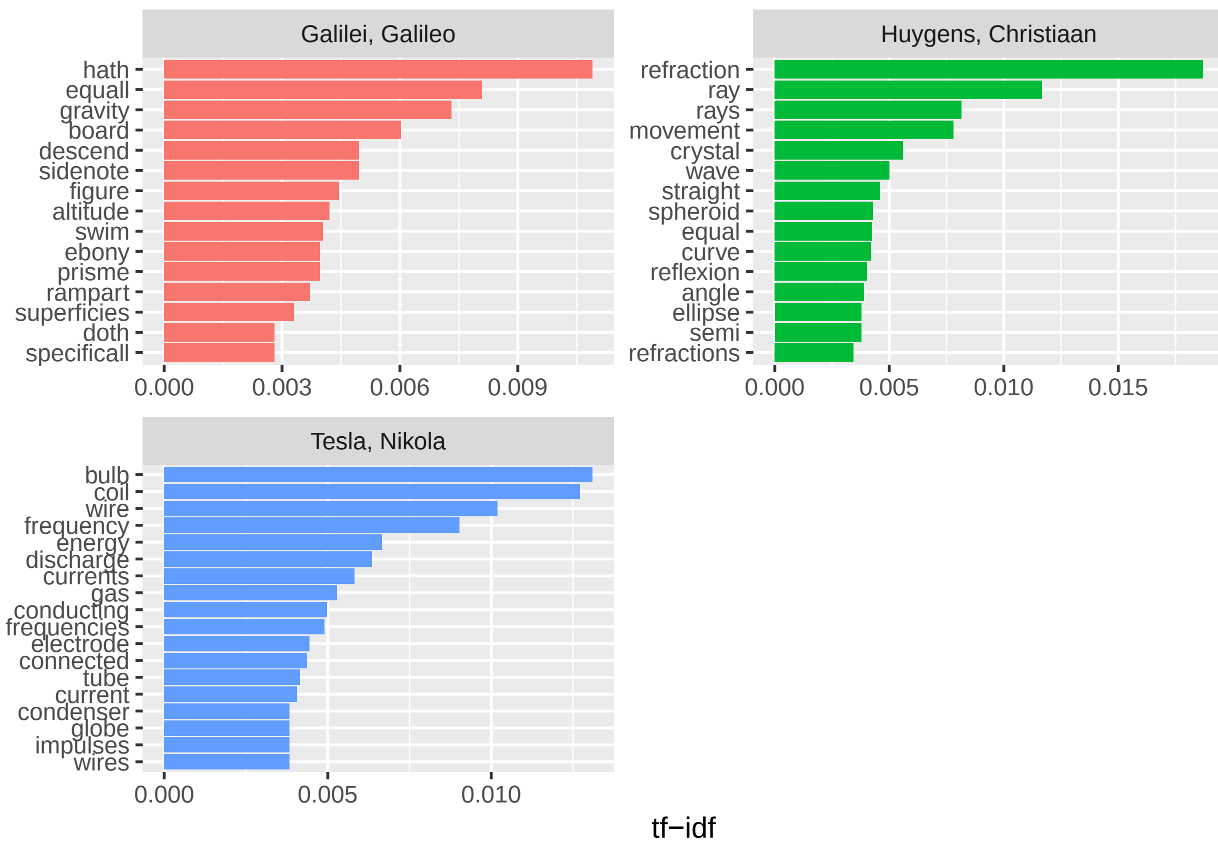
Let's remove some of these less meaningful words to make a better, more meaningful plot.

```
mystopwords <- data_frame(word = c("co", "rc", "ac", "ak", "bn",
                                       "fig", "file", "cg", "cb", "cm"))
    physics_words <- anti_join(physics_words, mystopwords, by = "word")
    plot_physics <- physics_words %>%
```

```
      bind_tf_idf(word, author, n) %>%
      arrange(desc(tf_idf)) %>%
      mutate(word = factor(word, levels = rev(unique(word)))) %>%
      group_by(author) %>%
      top_n(15, tf_idf) %>%
      ungroup %>%
      mutate(author = factor(author, levels = c("Galilei, Galileo",
                                                "Huygens, Christiaan",
                                                "Tesla, Nikola")))
```

```
## Warning in bind_tf_idf.data.frame(., word, author, n): A value for tf_idf is negative:
## Input should have exactly one row per document-term combination.
```

```
ggplot(plot_physics, aes(word, tf_idf, fill = author)) + geom_col(show.legend = FALSE) +
labs(x = NULL, y = "tf-idf") +
facet_wrap(~author, ncol = 2, scales = "free") + coord_flip()
```



10 The AssociatedPress dataset.

Let's load the AssociatedPress data-set, provided by the topicmodels package, as an example of a Document-TermMatrix. This is a collection of 2,246 news articles from an American news agency, mostly published around 1988.

```
library(topicmodels)
```

```
## Warning: package 'topicmodels' was built under R version 3.5.3
```

```r
    data("AssociatedPress")
    AssociatedPress
```

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

Le's use the LDA() function from the topicmodels package, setting k = 2, to create a two-topic LDA model. This function returns an object containing the full details of the model fit, such as how words are associated with topics and how topics are associated with documents.

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

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

10.1 Word-topic probabilities.

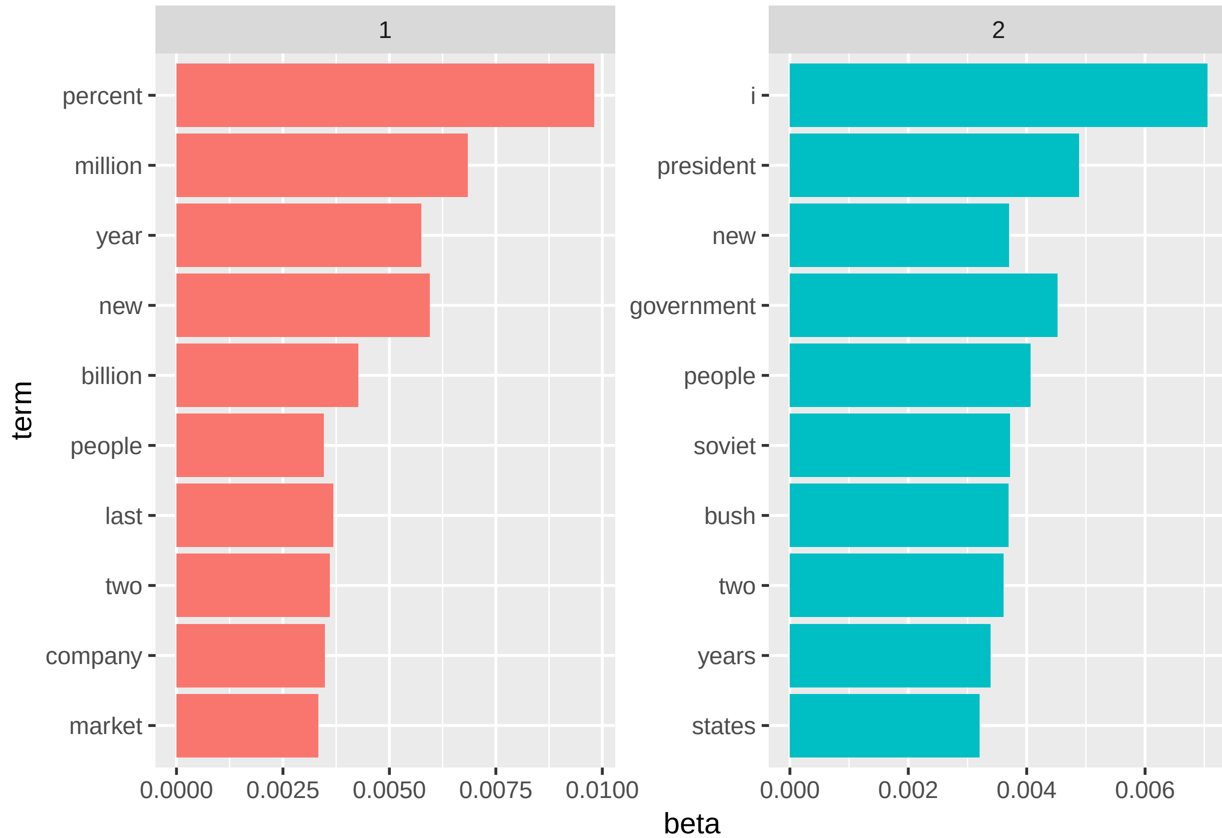
Let's use tidy() method for extracting the per-topic-per-word probabilities, called β, from the model.

```r
library(tidytext)
ap_topics <- tidy(ap_lda, matrix = "beta")
ap_topics
```

```
## # 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
```

Let's use dplyr's top_n() to find the 10 terms that are most common within each topic.

```r
library(ggplot2)
    library(dplyr)
    ap_top_terms <- ap_topics %>%
      group_by(topic) %>%
      top_n(10, beta) %>%
      ungroup() %>%
      arrange(topic, -beta)
ap_top_terms %>%
mutate(term = reorder(term, beta)) %>% ggplot(aes(term, beta, fill = factor(topic))) + geom_col(show.leg
facet_wrap(~ topic, scales = "free") + coord_flip()
```



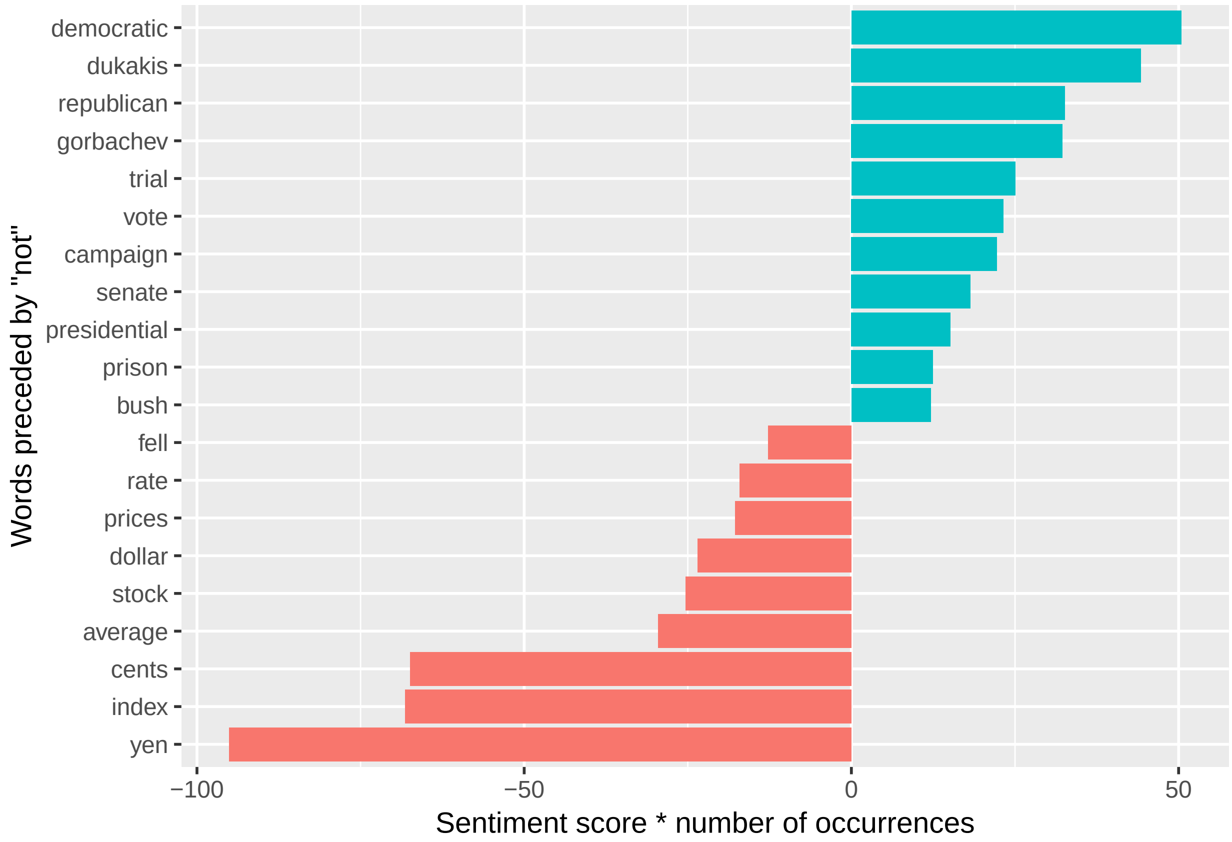
To constrain it to a set of especially relevant words, let's filter for relatively common words, such as those that have a β greater than $1/1000$ in at least one topic.

```
beta_spread <- ap_topics %>%
  mutate(topic = paste0("topic", topic)) %>%
  spread(topic, beta) %>%
  filter(topic1 > .001 | topic2 > .001) %>%
  mutate(log_ratio = log2(topic2 / topic1))
beta_spread
```

```
## # A tibble: 198 x 4
##    term              topic1      topic2 log_ratio
##    <chr>              <dbl>       <dbl>     <dbl>
##  1 administration 0.000431  0.00138         1.68
##  2 ago            0.00107   0.000842       -0.339
##  3 agreement      0.000671  0.00104         0.630
##  4 aid            0.0000476 0.00105         4.46
##  5 air            0.00214   0.000297       -2.85
##  6 american       0.00203   0.00168        -0.270
##  7 analysts       0.00109   0.000000578   -10.9
##  8 area           0.00137   0.000231       -2.57
##  9 army           0.000262  0.00105         2.00
## 10 asked          0.000189  0.00156         3.05
## # ... with 188 more rows
```

Let's visualize the words with the greatest differences between the two topics.

```r
beta_spread %>%
mutate(contribution = log_ratio) %>% arrange(desc(abs(contribution))) %>%
head(20) %>%
mutate(term = reorder(term, contribution)) %>%
  ggplot(aes(term, log_ratio, fill = log_ratio > 0)) + geom_col(show.legend = FALSE) +
xlab("Words preceded by \"not\"") +
ylab("Sentiment score * number of occurrences") + coord_flip()
```



Let's use tidy() method for extracting the per-topic-per-word probabilities, called γ, from the model.

```r
ap_documents <- tidy(ap_lda, 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
```

We can see that document 6 is drawn almost entirely from topic 2, having a γ from topic 1 close to zero.

To check this answer, we could tidy() the document-term matrix and check what the most common words in that document are.

```
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
```

11 Unsupervised learning example.

Example: The Great Library Heist - know the "right answer"!!

It can be useful to try it on a very simple case where you know the "right answer". Let's collect a set of documents that definitely relate to four separate topics, then perform topic modeling to see whether the algorithm can correctly distinguish the four groups.

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

Let's divide these into chapters, use tidytext's unnest_tokens() to separate them into words, then remove stop_words. We're treating every chapter as a separate "document"!!

```
library(stringr)
    # divide into documents, each representing one chapter
reg <- regex("^chapter ", ignore_case = TRUE)
by_chapter <- books %>%
      group_by(title) %>%
      mutate(chapter = cumsum(str_detect(text, reg))) %>%
      ungroup() %>%
      filter(chapter > 0) %>%
      unite(document, title, chapter)
    # split into words
    by_chapter_word <- by_chapter %>%
      unnest_tokens(word, text)
    # find document-word counts
word_counts <- by_chapter_word %>% anti_join(stop_words) %>% count(document, word, sort = TRUE) %>% ung
```

```
## Joining, by = "word"
```

```r
word_counts
```

```
## # A tibble: 104,722 x 3
##    document                 word        n
##    <chr>                    <chr>   <int>
##  1 Great Expectations_57    joe        88
##  2 Great Expectations_7     joe        70
##  3 Great Expectations_17    biddy      63
##  4 Great Expectations_27    joe        58
##  5 Great Expectations_38    estella    58
##  6 Great Expectations_2     joe        56
##  7 Great Expectations_23    pocket     53
##  8 Great Expectations_15    joe        50
##  9 Great Expectations_18    joe        50
## 10 The War of the Worlds_16 brother    50
## # ... with 104,712 more rows
```

The topicmodels package requires a DocumentTermMatrix. So, let's cast a one-token-per-row table into a DocumentTermMatrix with tidytext's cast_dtm().

```r
chapters_dtm <- word_counts %>%
      cast_dtm(document, word, n)
    chapters_dtm
```

```
## <<DocumentTermMatrix (documents: 193, terms: 18215)>>
## Non-/sparse entries: 104722/3410773
## Sparsity           : 97%
## Maximal term length: 19
## Weighting          : term frequency (tf)
```

Let's then use the LDA() function to create a four-topic model.

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

```
## A LDA_VEM topic model with 4 topics.
```

Let's examine per-topic-per-word probabilities.

```r
chapter_topics <- tidy(chapters_lda, matrix = "beta")
    chapter_topics
```

```
## # A tibble: 72,860 x 3
##    topic term        beta
##    <int> <chr>      <dbl>
##  1     1 joe     1.44e-17
##  2     2 joe     5.96e-61
##  3     3 joe     9.88e-25
##  4     4 joe     1.45e- 2
##  5     1 biddy   5.14e-28
##  6     2 biddy   5.02e-73
##  7     3 biddy   4.31e-48
##  8     4 biddy   4.78e- 3
##  9     1 estella 2.43e- 6
## 10     2 estella 4.32e-68
## # ... with 72,850 more rows
```

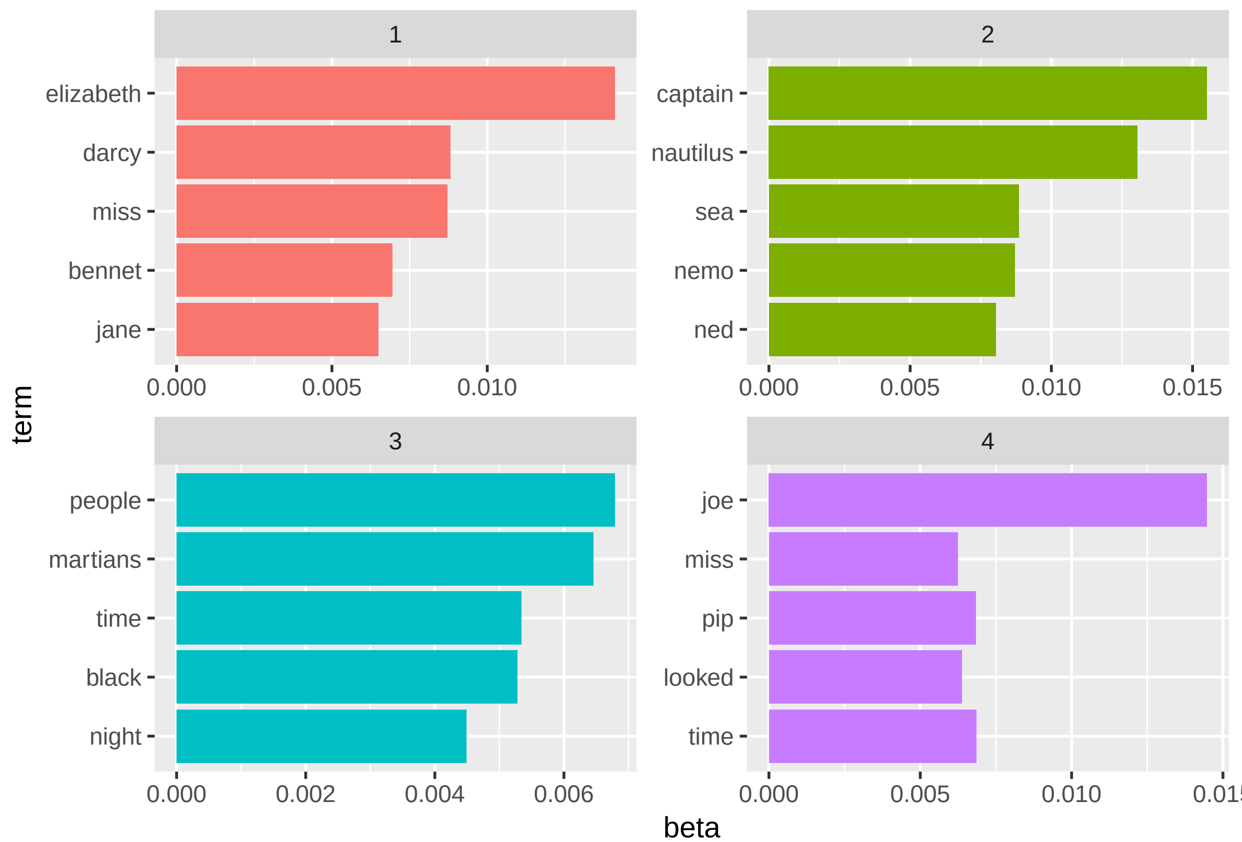
Let's use dplyr's top_n() to find the top five terms within each topic.

```
top_terms <- chapter_topics %>%
      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 elizabeth 0.0141
##  2     1 darcy     0.00881
##  3     1 miss      0.00871
##  4     1 bennet    0.00694
##  5     1 jane      0.00649
##  6     2 captain   0.0155
##  7     2 nautilus  0.0131
##  8     2 sea       0.00884
##  9     2 nemo      0.00871
## 10     2 ned       0.00803
## 11     3 people    0.00679
## 12     3 martians  0.00646
## 13     3 time      0.00534
## 14     3 black     0.00528
## 15     3 night     0.00449
## 16     4 joe       0.0145
## 17     4 time      0.00685
## 18     4 pip       0.00683
## 19     4 looked    0.00637
## 20     4 miss      0.00623
```

Let's visualize this using a ggplot2.

```
library(ggplot2)
top_terms %>%
mutate(term = reorder(term, beta)) %>% ggplot(aes(term, beta, fill = factor(topic))) + geom_col(show.leg
facet_wrap(~ topic, scales = "free") + coord_flip()
```



11.1 Per-document classification.

We want to know which topics are associated with each document. We can find this by examining the per-document-per-topic probabilities, γ.

```
chapters_gamma <- tidy(chapters_lda, matrix = "gamma")
    chapters_gamma
```

```
## # A tibble: 772 x 3
##    document                 topic     gamma
##    <chr>                    <int>     <dbl>
##  1 Great Expectations_57        1 0.0000134
##  2 Great Expectations_7         1 0.0000146
##  3 Great Expectations_17        1 0.0000210
##  4 Great Expectations_27        1 0.0000190
##  5 Great Expectations_38        1 0.355
##  6 Great Expectations_2         1 0.0000171
##  7 Great Expectations_23        1 0.547
##  8 Great Expectations_15        1 0.0124
##  9 Great Expectations_18        1 0.0000126
## 10 The War of the Worlds_16     1 0.0000107
## # ... with 762 more rows
```

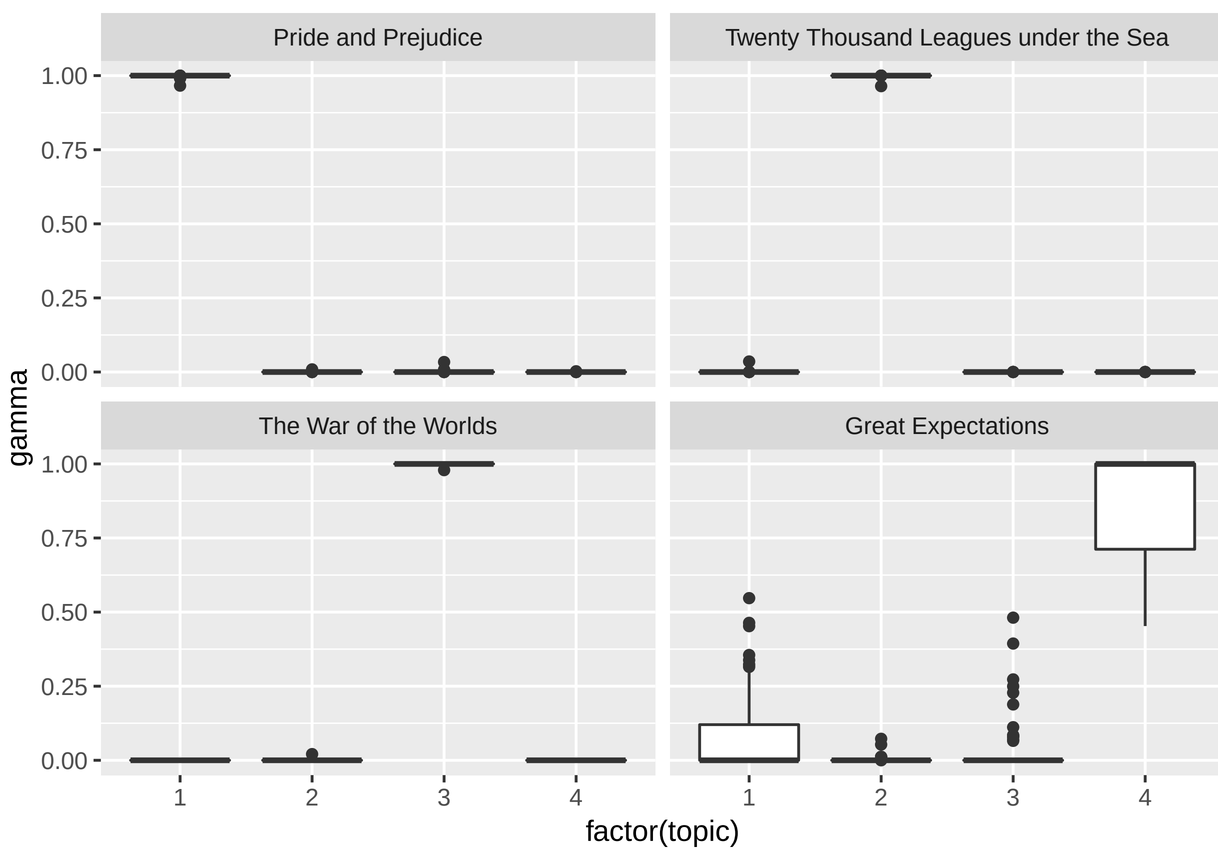
Let's see how well our unsupervised learning did at distinguishing the four books. To do so, first we re-separate the document name into title and chapter, after which we can visualize the per-document-per-topic probability for each.

```r
chapters_gamma <- chapters_gamma %>%
separate(document, c("title", "chapter"), sep = "_", convert = TRUE)
    chapters_gamma
```

```
## # A tibble: 772 x 4
##    title                 chapter topic     gamma
##    <chr>                   <int> <int>     <dbl>
##  1 Great Expectations         57     1 0.0000134
##  2 Great Expectations          7     1 0.0000146
##  3 Great Expectations         17     1 0.0000210
##  4 Great Expectations         27     1 0.0000190
##  5 Great Expectations         38     1 0.355
##  6 Great Expectations          2     1 0.0000171
##  7 Great Expectations         23     1 0.547
##  8 Great Expectations         15     1 0.0124
##  9 Great Expectations         18     1 0.0000126
## 10 The War of the Worlds      16     1 0.0000107
## # ... with 762 more rows
```

Let's visualize this!

```r
# reorder titles in order of topic 1, topic 2, etc. before plotting
chapters_gamma %>%
  mutate(title = reorder(title, gamma * topic)) %>%
  ggplot(aes(factor(topic), gamma)) +
  geom_boxplot() +
  facet_wrap(~ title)
```



Now we want to know if there are any cases where the topic most associated with a chapter belonged to another book? To do so, first we find the topic that was most associated with each chapter using top_n(), which is effectively the "classification" of that chapter.

```
chapter_classifications <- chapters_gamma %>%
      group_by(title, chapter) %>%
      top_n(1, gamma) %>%
ungroup()
chapter_classifications
```

```
## # A tibble: 193 x 4
##    title                 chapter topic gamma
##    <chr>                   <int> <int> <dbl>
##  1 Great Expectations         23     1 0.547
##  2 Pride and Prejudice        43     1 1.000
##  3 Pride and Prejudice        18     1 1.000
##  4 Pride and Prejudice        45     1 1.000
##  5 Pride and Prejudice        16     1 1.000
##  6 Pride and Prejudice        29     1 1.000
##  7 Pride and Prejudice        10     1 1.000
##  8 Pride and Prejudice         8     1 1.000
##  9 Pride and Prejudice        56     1 1.000
## 10 Pride and Prejudice        47     1 1.000
## # ... with 183 more rows
```

We can then compare each to the "consensus" topic for each book (the most common topic among its chapters), and see which were most often misidentified.

```
book_topics <- chapter_classifications %>%
      count(title, topic) %>%
      group_by(title) %>%
      top_n(1, n) %>%
      ungroup() %>%
      transmute(consensus = title, topic)
    chapter_classifications %>%
      inner_join(book_topics, by = "topic") %>%
      filter(title != consensus)
```

```
## # A tibble: 2 x 5
##   title               chapter topic gamma consensus
##   <chr>                 <int> <int> <dbl> <chr>
## 1 Great Expectations       23     1 0.547 Pride and Prejudice
## 2 Great Expectations       54     3 0.481 The War of the Worlds
```

11.2 By-word assignments: augment.

We want to take the original document-word pairs and find which words in each document were assigned to which topic.

```
assignments <- augment(chapters_lda, data = chapters_dtm)
    assignments
```

```
## # A tibble: 104,722 x 4
##    document              term  count .topic
##    <chr>                 <chr> <dbl>  <dbl>
##  1 Great Expectations_57 joe      88      4
##  2 Great Expectations_7  joe      70      4
```

```
##   3 Great Expectations_17 joe       5      4
##   4 Great Expectations_27 joe      58      4
##   5 Great Expectations_2  joe      56      4
##   6 Great Expectations_23 joe       1      4
##   7 Great Expectations_15 joe      50      4
##   8 Great Expectations_18 joe      50      4
##   9 Great Expectations_9  joe      44      4
##  10 Great Expectations_13 joe      40      4
## # ... with 104,712 more rows
```

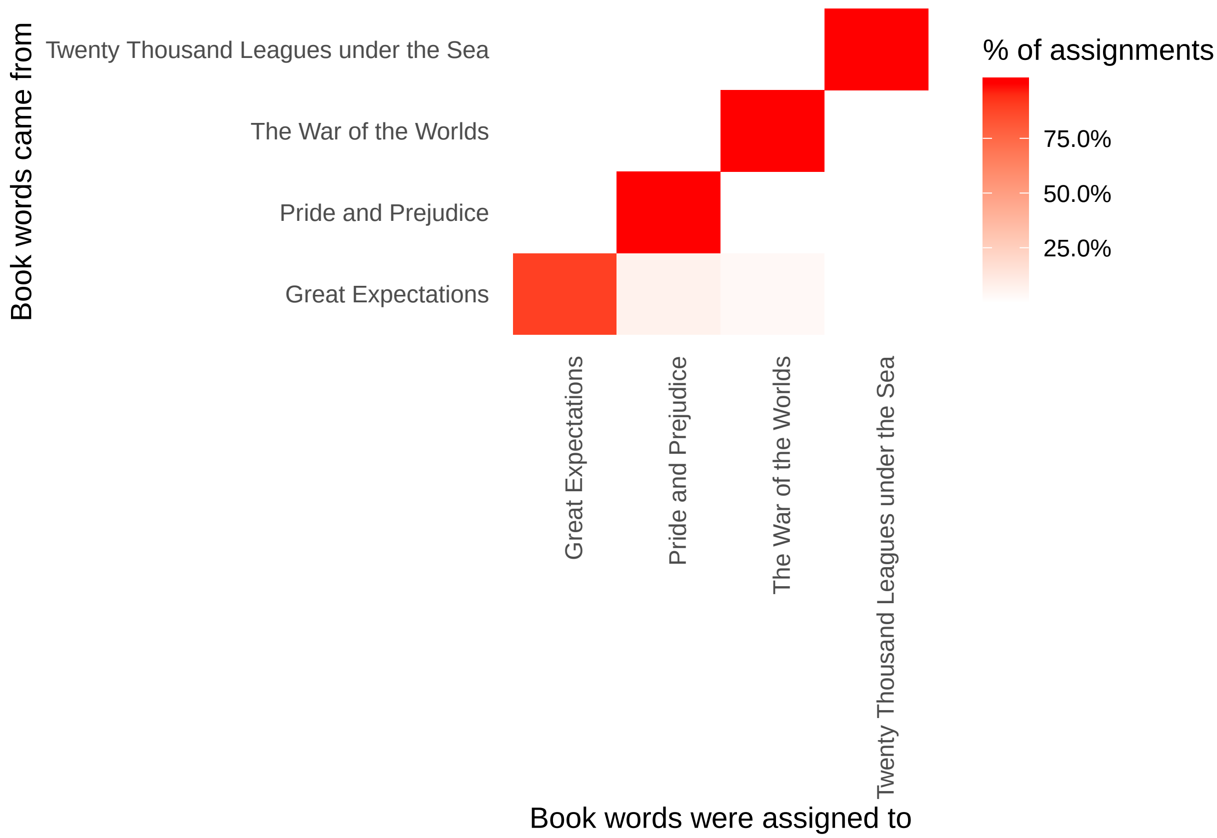
Let's combine this assignments table with the consensus book titles to find which words were incorrectly classified.

```
assignments <- assignments %>%
separate(document, c("title", "chapter"), sep = "_", convert = TRUE) %>% inner_join(book_topics, by = c
assignments
```

```
## # A tibble: 104,722 x 6
##    title              chapter term  count .topic consensus
##    <chr>                <int> <chr> <dbl>  <dbl> <chr>
##  1 Great Expectations      57 joe      88      4 Great Expectations
##  2 Great Expectations       7 joe      70      4 Great Expectations
##  3 Great Expectations      17 joe       5      4 Great Expectations
##  4 Great Expectations      27 joe      58      4 Great Expectations
##  5 Great Expectations       2 joe      56      4 Great Expectations
##  6 Great Expectations      23 joe       1      4 Great Expectations
##  7 Great Expectations      15 joe      50      4 Great Expectations
##  8 Great Expectations      18 joe      50      4 Great Expectations
##  9 Great Expectations       9 joe      44      4 Great Expectations
## 10 Great Expectations      13 joe      40      4 Great Expectations
## # ... with 104,712 more rows
```

Let's visualize a confusion matrix, showing how often words from one book were assigned to another, using dplyr's count() and ggplot2's geom_tile.

```
assignments %>%
      count(title, consensus, wt = count) %>%
      group_by(title) %>%
      mutate(percent = n / sum(n)) %>%
      ggplot(aes(consensus, title, fill = percent)) +
      geom_tile() +
      scale_fill_gradient2(high = "red", label = scales::percent_format()) +
      theme_minimal() +
      theme(axis.text.x = element_text(angle = 90, hjust = 1),
            panel.grid = element_blank()) +
      labs(x = "Book words were assigned to",
           y = "Book words came from",
           fill = "% of assignments")
```



What were the most commonly mistaken words?

```
wrong_words <- assignments %>%
      filter(title != consensus)
wrong_words
```

```
## # A tibble: 4,617 x 6
##    title                 chapter term   count .topic consensus
##    <chr>                   <int> <chr>  <dbl>  <dbl> <chr>
##  1 Great Expectations         38 broth~     2      1 Pride and Prejudice
##  2 Great Expectations         22 broth~     4      1 Pride and Prejudice
##  3 Great Expectations         23 miss       2      1 Pride and Prejudice
##  4 Great Expectations         22 miss      23      1 Pride and Prejudice
##  5 Twenty Thousand Leag~       8 miss       1      1 Pride and Prejudice
##  6 Great Expectations         31 miss       1      1 Pride and Prejudice
##  7 Great Expectations          5 serge~    37      1 Pride and Prejudice
##  8 Great Expectations         46 capta~     1      2 Twenty Thousand Leagu~
##  9 Great Expectations         32 capta~     1      2 Twenty Thousand Leagu~
## 10 The War of the Worlds      17 capta~     5      2 Twenty Thousand Leagu~
## # ... with 4,607 more rows
```

```
wrong_words %>%
  count(title, consensus, term, wt = count) %>%
  ungroup() %>%
  arrange(desc(n))
```

```
## # A tibble: 3,551 x 4
##    title                consensus                 term          n
```

```
##    <chr>              <chr>                  <chr>    <dbl>
##  1 Great Expectations Pride and Prejudice   love        44
##  2 Great Expectations Pride and Prejudice   sergeant    37
##  3 Great Expectations Pride and Prejudice   lady        32
##  4 Great Expectations Pride and Prejudice   miss        26
##  5 Great Expectations The War of the Worlds boat        25
##  6 Great Expectations The War of the Worlds tide        20
##  7 Great Expectations The War of the Worlds water       20
##  8 Great Expectations Pride and Prejudice   father      19
##  9 Great Expectations Pride and Prejudice   baby        18
## 10 Great Expectations Pride and Prejudice   flopson     18
## # ... with 3,541 more rows
```

We can confirm "flopson" appears only in Great Expectations, even though it's assigned to the Pride and Prejudice cluster. This shows that the LDA algorithm is stochastic, and it can accidentally land on a topic that spans multiple books.

```
word_counts %>%
      filter(word == "flopson")
```

```
## # A tibble: 3 x 3
##   document              word        n
##   <chr>                 <chr>   <int>
## 1 Great Expectations_22 flopson    10
## 2 Great Expectations_23 flopson     7
## 3 Great Expectations_33 flopson     1
```

12 Tweets de candidatos presidenciales (en español).

Código adaptado de Análisis de sentimientos con R - Léxico Afinn, por Juan Bosco Mendoza Vega. https://rpubs.com/jboscomendoza/

```
library(tidyverse)
library(tidytext)
library(tm)
```

```
## Warning: package 'tm' was built under R version 3.5.3

## Loading required package: NLP

## Warning: package 'NLP' was built under R version 3.5.2

##
## Attaching package: 'NLP'

## The following object is masked from 'package:ggplot2':
##
##     annotate
```

```
library(lubridate)
library(zoo)
library(scales)
```

Definimos un tema para facilitar la visualización de nuestros resultados.

```
tema_graf <-
  theme_minimal() +
  theme(text = element_text(family = "serif"),
        panel.grid.minor = element_blank(),
```

```
        strip.background = element_rect(fill = "#EBEBEB", colour = NA),
        legend.position = "none",
        legend.box.background = element_rect(fill = "#EBEBEB", colour = NA))
```

```
tuits <- read.csv("tuits_candidatos.csv",
                  stringsAsFactors = F, fileEncoding = "latin1") %>%
  tbl_df()
```

Nuestros datos lucen así:

```
tuits
```

```
## # A tibble: 2,660 x 4
##    status_id created_at    screen_name  text
##        <dbl> <chr>         <chr>        <chr>
##  1   7.30e17 09/05/2016 0~ lopezobrado~ Proceso habla de vilezas de EPN-Ca~
##  2   7.30e17 10/05/2016 0~ lopezobrado~ "MORENA llegó como bendición justo~
##  3   7.30e17 10/05/2016 1~ lopezobrado~ Muchas felicidades a las madres, t~
##  4   7.31e17 12/05/2016 0~ lopezobrado~ "En Chihuahua, Javier Félix Muñoz,~
##  5   7.31e17 13/05/2016 0~ lopezobrado~ Están desatados priístas, panistas~
##  6   7.31e17 13/05/2016 1~ lopezobrado~ No sé a ustedes, pero a mí lo que ~
##  7   7.32e17 15/05/2016 0~ lopezobrado~ Luego de la explosión en Pajaritos~
##  8   7.32e17 16/05/2016 0~ lopezobrado~ Hoy 15 de mayo nuestro sincero res~
##  9   7.32e17 17/05/2016 0~ lopezobrado~ "El periódico Financial Times ning~
## 10   7.33e17 18/05/2016 1~ lopezobrado~ Sandra Ávila Beltrán, conocida com~
## # ... with 2,650 more rows
```

El número de twits (2016 al 2018) por candidato se extrae de la siguiente manera:

```
tuits %>% count(screen_name, sort = TRUE)
```

```
## # A tibble: 5 x 2
##   screen_name       n
##   <chr>         <int>
## 1 lopezobrador_   598
## 2 Mzavalagc       593
## 3 RicardoAnayaC   578
## 4 JoseAMeadeK     562
## 5 JaimeRdzNL      329
```

Para este análisis de sentimiento usaremos el léxico Afinn. Este es un conjunto de palabras, puntuadas de acuerdo a qué tan positivamente o negativamente son percibidas. Las palabras que son percibidas de manera positiva tienen puntuaciones de -4 a -1; y las positivas de 1 a 4.

La versión que usaremos es una traducción automática, de inglés a español, de la versión del léxico presente en el conjunto de datos sentiments de tidytext, con algunas correcciones manuales. Por supuesto, esto quiere decir que este léxico tendrá algunos defectos, pero será suficiente para nuestro análisis.

```
afinn <- read.csv("lexico_afinn.en.es.csv",
                  stringsAsFactors = F, fileEncoding = "latin1") %>%
  tbl_df()
```

Este léxico luce así:

```
afinn
```

```
## # A tibble: 2,476 x 3
##    Palabra     Puntuacion Word
##    <chr>            <int> <chr>
```

```
##   1 a bordo              1 aboard
##   2 abandona            -2 abandons
##   3 abandonado          -2 abandoned
##   4 abandonar           -2 abandon
##   5 abatido             -2 dejected
##   6 abatido             -3 despondent
##   7 aborrece            -3 abhors
##   8 aborrecer           -3 abhor
##   9 aborrecible         -3 abhorrent
##  10 aborrecido          -3 abhorred
## # ... with 2,466 more rows
```

Tenemos tres columnas. Una con palabras en español, su puntuación y una tercera columna con la misma palabra, en inglés.

Hora de preparar nuestros datos para análisis.

12.1 Peparando los datos

Fechas

Lo primero que necesitamos es filtrar el objeto tuits para limitar nuestros datos sólo a los del 2018. Manipulamos la columna created_at con la función separate() de tidyr. Separamos esta columna en una fecha y hora del día, y después separaremos la fecha en día, mes y año. Usamos la función ymd() de lubridate para convertir la nueva columna Fecha a tipo de dato fecha.

Por último, usamos filter() de dplyr para seleccionar sólo los tuits hechos en el 2018.

```
tuits <-
  tuits %>%
  separate(created_at, into = c("Fecha", "Hora"), sep = " ") %>%
  separate(Fecha, into = c("Dia", "Mes", "Periodo"), sep = "/",
           remove = FALSE) %>%
  mutate(Fecha = dmy(Fecha),
         Semana = week(Fecha) %>% as.factor(),
         text = tolower(text)) %>%
  filter(Periodo == 2018)
```

Nuestros datos lucen así:

```
tuits
```

```
## # A tibble: 1,607 x 9
##    status_id Fecha      Dia   Mes   Periodo Hora  screen_name  text  Semana
##        <dbl> <date>     <chr> <chr> <chr>   <chr> <chr>        <chr> <fct>
##  1   9.48e17 2018-01-01 01    01    2018    01:42 JoseAMeadeK  les ~ 1
##  2   9.48e17 2018-01-01 01    01    2018    15:28 lopezobrado~ dese~ 1
##  3   9.48e17 2018-01-01 01    01    2018    22:46 JoseAMeadeK  en d~ 1
##  4   9.48e17 2018-01-02 02    01    2018    13:36 JoseAMeadeK  feli~ 1
##  5   9.48e17 2018-01-02 02    01    2018    14:01 JoseAMeadeK  vamo~ 1
##  6   9.48e17 2018-01-02 02    01    2018    17:38 JoseAMeadeK  esto~ 1
##  7   9.48e17 2018-01-02 02    01    2018    17:57 JoseAMeadeK  inic~ 1
##  8   9.48e17 2018-01-02 02    01    2018    18:44 Mzavalagc    rt @~ 1
##  9   9.48e17 2018-01-02 02    01    2018    19:07 JoseAMeadeK  zaca~ 1
## 10   9.48e17 2018-01-02 02    01    2018    20:27 lopezobrado~ nues~ 1
## # ... with 1,597 more rows
```

El número de twits (2018) por candidato se extrae de la siguiente manera:

```
tuits %>% count(screen_name, sort = TRUE)
```

```
## # A tibble: 5 x 2
##   screen_name        n
##   <chr>          <int>
## 1 Mzavalagc        529
## 2 JoseAMeadeK      352
## 3 JaimeRdzNL       329
## 4 RicardoAnayaC    291
## 5 lopezobrador_    106
```

12.2 Convirtiendo tuits en palabras

Necesitamos separar cada tuit en palabras, para así asignarle a cada palabra relevante una puntuación de sentimiento usando el léxico Afinn. Usamos la función unnest_token() de tidytext, que tomara los tuits en la columna text y los separá en una nueva columna llamada Palabra Hecho esto, usamos left_join() de dplyr, para unir los objetos tuits y afinn, a partir del contenido de la columna Palabra. De este modo, obtendremos un data frame que contiene sólo los tuits con palabras presentes en el léxico Afinn.

Además, aprovechamos para crear una columna con mutate() de dplyr a las palabras como Positiva o Negativa. Llamaremos esta columna Tipo y cambiamos el nombre de la columna screen_name a Candidato.

```
tuits_afinn <-
  tuits %>%
  unnest_tokens(input = "text", output = "Palabra") %>%
  inner_join(afinn, ., by = "Palabra") %>%
  mutate(Tipo = ifelse(Puntuacion > 0, "Positiva", "Negativa")) %>%
  rename("Candidato" = screen_name)
```

Obtenemos también una puntuación por tuit, usando group_by() y summarise() de dplyr, y la agregamos tuits para usarla después. Tambien asignamos a los tuits sin puntuación positiva o negativa un valor de 0, que indica neutralidad. Por último cambiamos el nombre de la columna screen_name a Candidato

```
tuits <-
  tuits_afinn %>%
  group_by(status_id) %>%
  summarise(Puntuacion_tuit = mean(Puntuacion)) %>%
  left_join(tuits, ., by = "status_id") %>%
  mutate(Puntuacion_tuit = ifelse(is.na(Puntuacion_tuit), 0, Puntuacion_tuit)) %>%
  rename("Candidato" = screen_name)
```

```
tuits_afinn
```

```
## # A tibble: 2,696 x 12
##    Palabra Puntuacion Word  status_id Fecha      Dia   Mes   Periodo Hora
##    <chr>        <int> <chr>     <dbl> <date>     <chr> <chr> <chr>   <chr>
##  1 abando~         -2 aban~   9.81e17 2018-04-03 03    04    2018    21:31
##  2 abando~         -2 aban~   9.79e17 2018-03-27 27    03    2018    18:59
##  3 abrazo           2 hug     9.58e17 2018-01-30 30    01    2018    13:27
##  4 abrazo           2 hug     9.58e17 2018-01-30 30    01    2018    13:27
##  5 abrazo           2 hug     9.58e17 2018-01-30 30    01    2018    15:09
##  6 abrazo           2 hug     9.59e17 2018-02-01 01    02    2018    03:16
##  7 abrazo           2 hug     9.69e17 2018-03-02 02    03    2018    02:20
##  8 abrazo           2 hug     9.79e17 2018-03-29 29    03    2018    16:26
##  9 abrazo           2 hug     9.80e17 2018-03-30 30    03    2018    20:20
## 10 abrazo           2 hug     9.83e17 2018-04-09 09    04    2018    02:19
```

```
## # ... with 2,686 more rows, and 3 more variables: Candidato <chr>,
## #   Semana <fct>, Tipo <chr>
```

Con esto estamos listos para empezar.

12.3 Explorando los datos, medias por día

Empecemos revisando cuántas palabras en total y cuantas palabras únicas ha usado cada candidato con count(), group_by() y distinct() de dplyr.

```
# Total
tuits_afinn %>%
  count(Candidato)
```

```
## # A tibble: 5 x 2
##   Candidato         n
##   <chr>         <int>
## 1 JaimeRdzNL      525
## 2 JoseAMeadeK     533
## 3 lopezobrador_   183
## 4 Mzavalagc       838
## 5 RicardoAnayaC   617
```

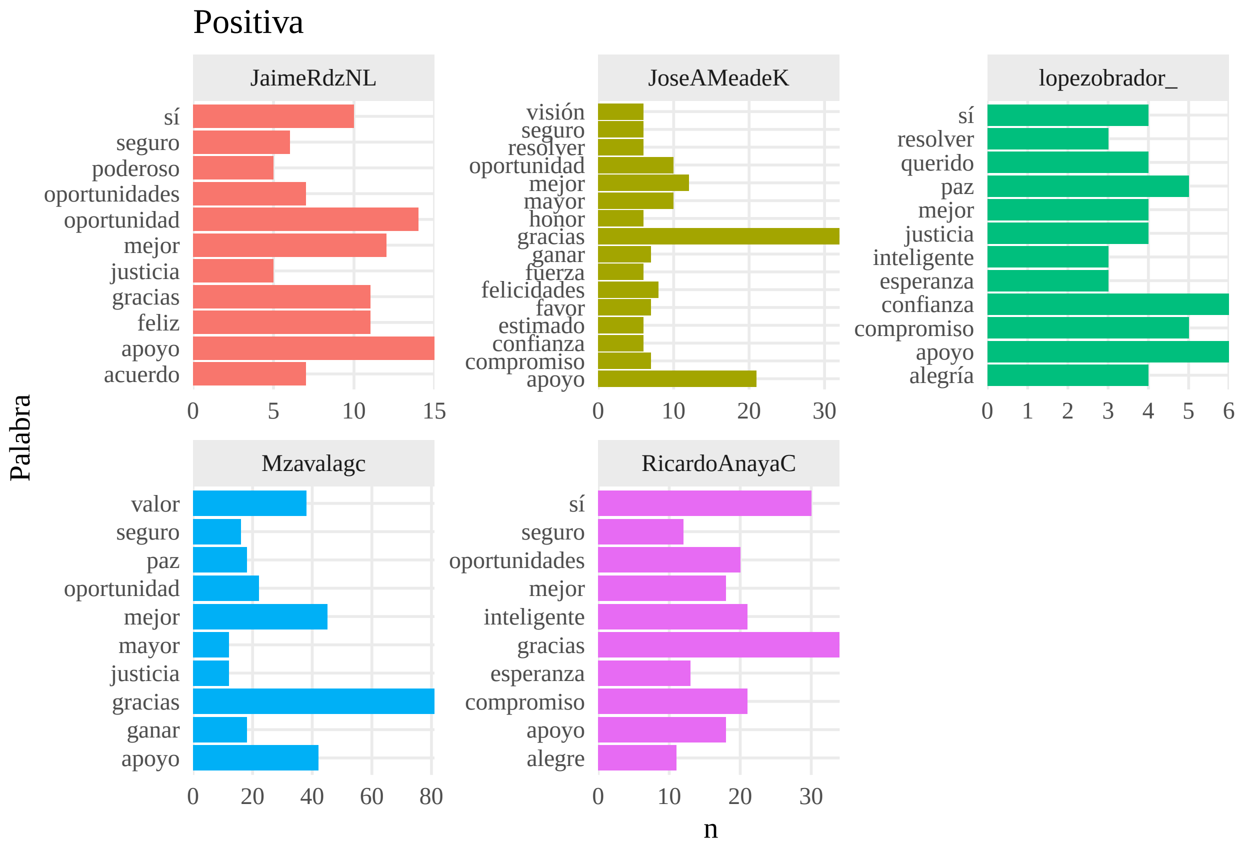
```
# Únicas
tuits_afinn %>%
  group_by(Candidato) %>%
  distinct(Palabra) %>%
  count()
```

```
## # A tibble: 5 x 2
## # Groups:   Candidato [5]
##   Candidato         n
##   <chr>         <int>
## 1 JaimeRdzNL      117
## 2 JoseAMeadeK     183
## 3 lopezobrador_    73
## 4 Mzavalagc       190
## 5 RicardoAnayaC   153
```

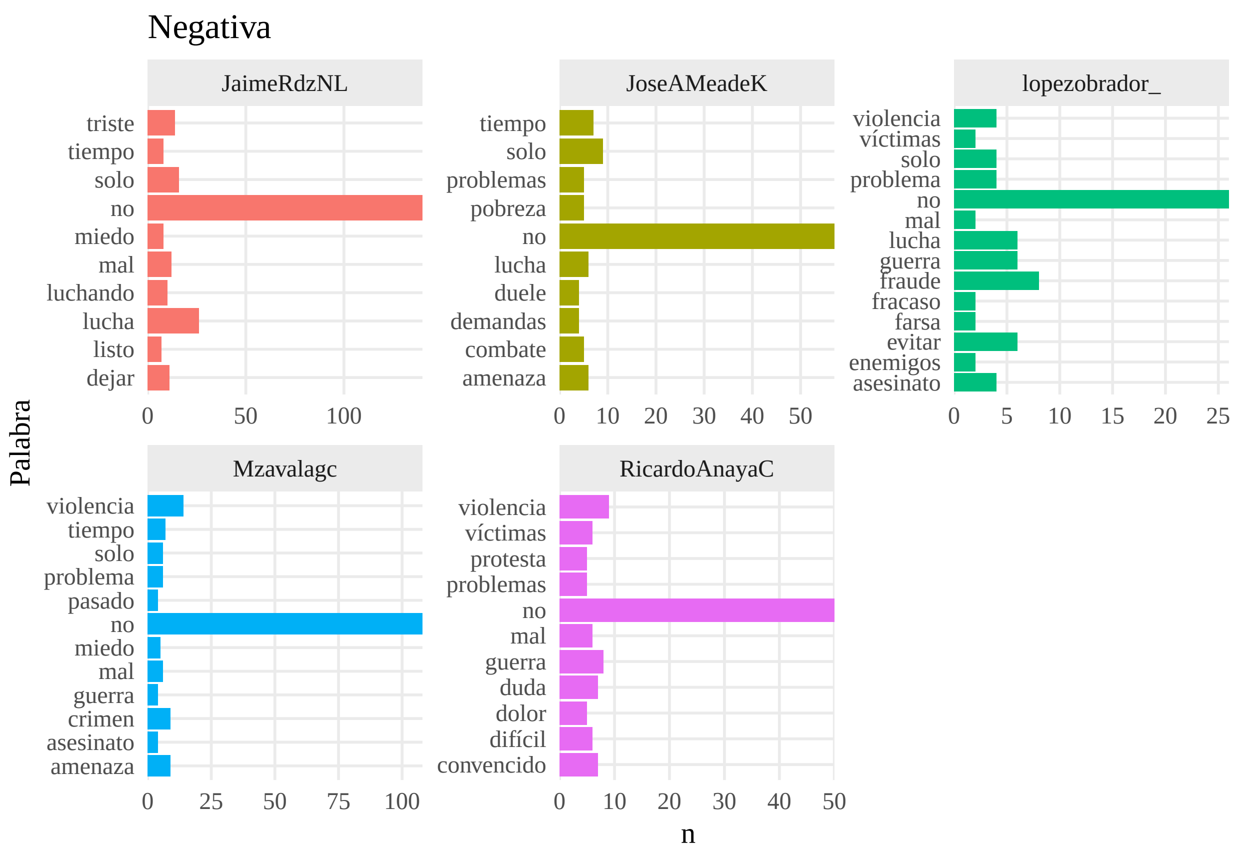
Y veamos también las palabras positivas y negativas más usadas por cada uno de ellos, usando map() de purr, top_n() de dplyr() y ggplot.

```
map(c("Positiva", "Negativa"), function(sentimiento) {
  tuits_afinn %>%
    filter(Tipo ==  sentimiento) %>%
    group_by(Candidato) %>%
    count(Palabra, sort = T) %>%
    top_n(n = 10, wt = n) %>%
    ggplot() +
    aes(Palabra, n, fill = Candidato) +
    geom_col() +
    facet_wrap("Candidato", scales = "free") +
    scale_y_continuous(expand = c(0, 0)) +
    coord_flip() +
    labs(title = sentimiento) +
    tema_graf
})
```

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


Positiva
JaimeRdzNL
sí
seguro
poderoso
oportunidades
oportunidad
mejor
justicia
gracias
feliz
apoyo
acuerdo
0 5 10 15
JoseAMeadeK
visión
seguro
resolver
oportunidad
mejor
mayor
honor
gracias
ganar
fuerza
felicidades
favor
estimado
confianza
compromiso
apoyo
0 10 20 30
lopezobrador_
sí
resolver
querido
paz
mejor
justicia
inteligente
esperanza
confianza
compromiso
apoyo
alegría
0 1 2 3 4 5 6
Palabra
Mzavalagc
valor
seguro
paz
oportunidad
mejor
mayor
justicia
gracias
ganar
apoyo
0 20 40 60 80
RicardoAnayaC
sí
seguro
oportunidades
mejor
inteligente
gracias
esperanza
compromiso
apoyo
alegre
0 10 20 30
n

```
##
## [[2]]
```



Aunque hay similitudes en las palabras usadas, también observamos una diferencia considerable en la cantidad de palabras usadas por el candidato con menos palabras (157, 72 únicas de lopezobrador_) y la candidata con más (730, 189 únicas de Mzavalagc).

Si calculamos el sentimiento de los candidatos, haciendo una suma de puntuaciones, aquellos con más palabras podrían tener puntuaciones más altas, lo cual sesgaría nuestra interpretación de la magnitud de los resultados. En un caso como este, nos conviene pensar en una medida resumen como la media para hacer una mejor interpretación de nuestros datos.

Quitamos "no" de nuestras palabras. Es una palabra muy comun en español que no necesariamente implica un sentimiento negativo. Es la palabra negativa más frecuente entre los candidatos, por lo que podría sesgar nuestros resultados.

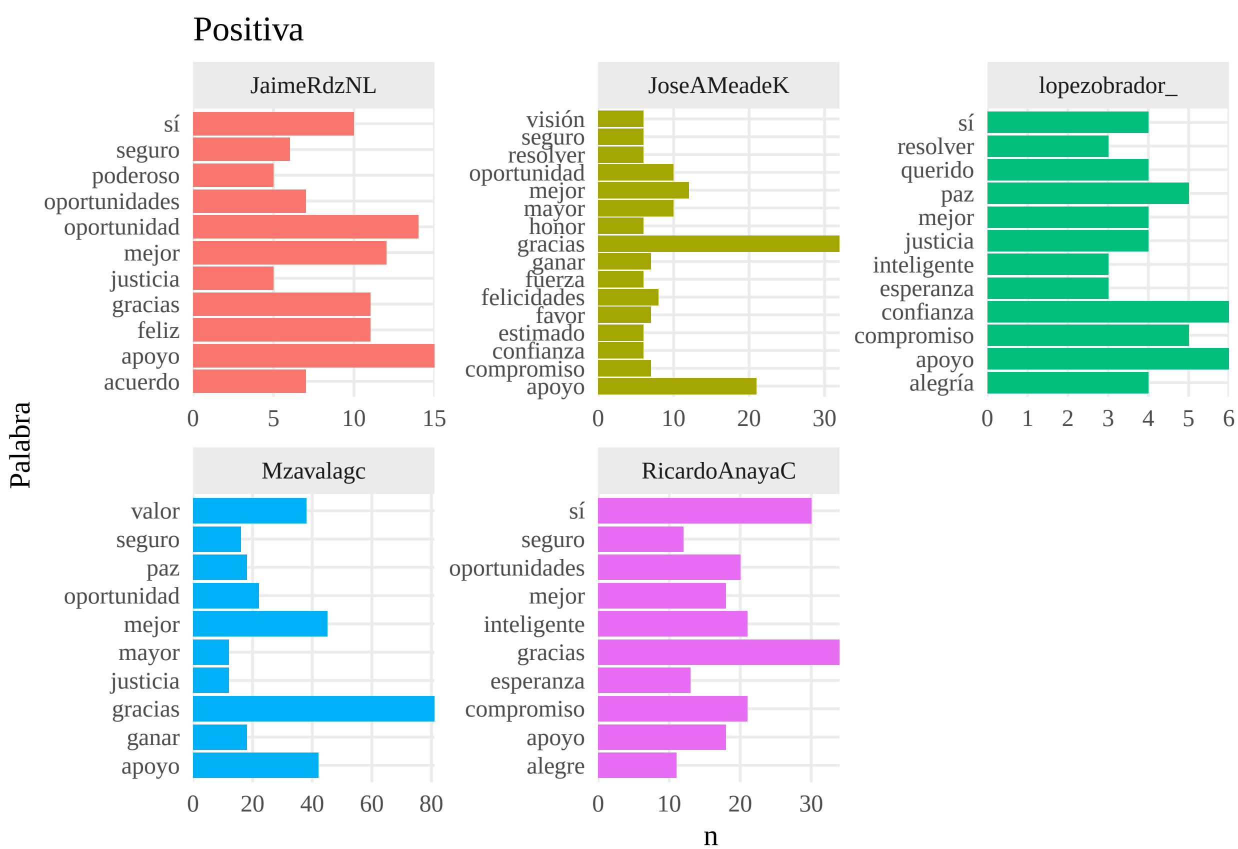
```r
tuits_afinn <-
  tuits_afinn %>%
  filter(Palabra != "no")
```

Repetimos el análisis.

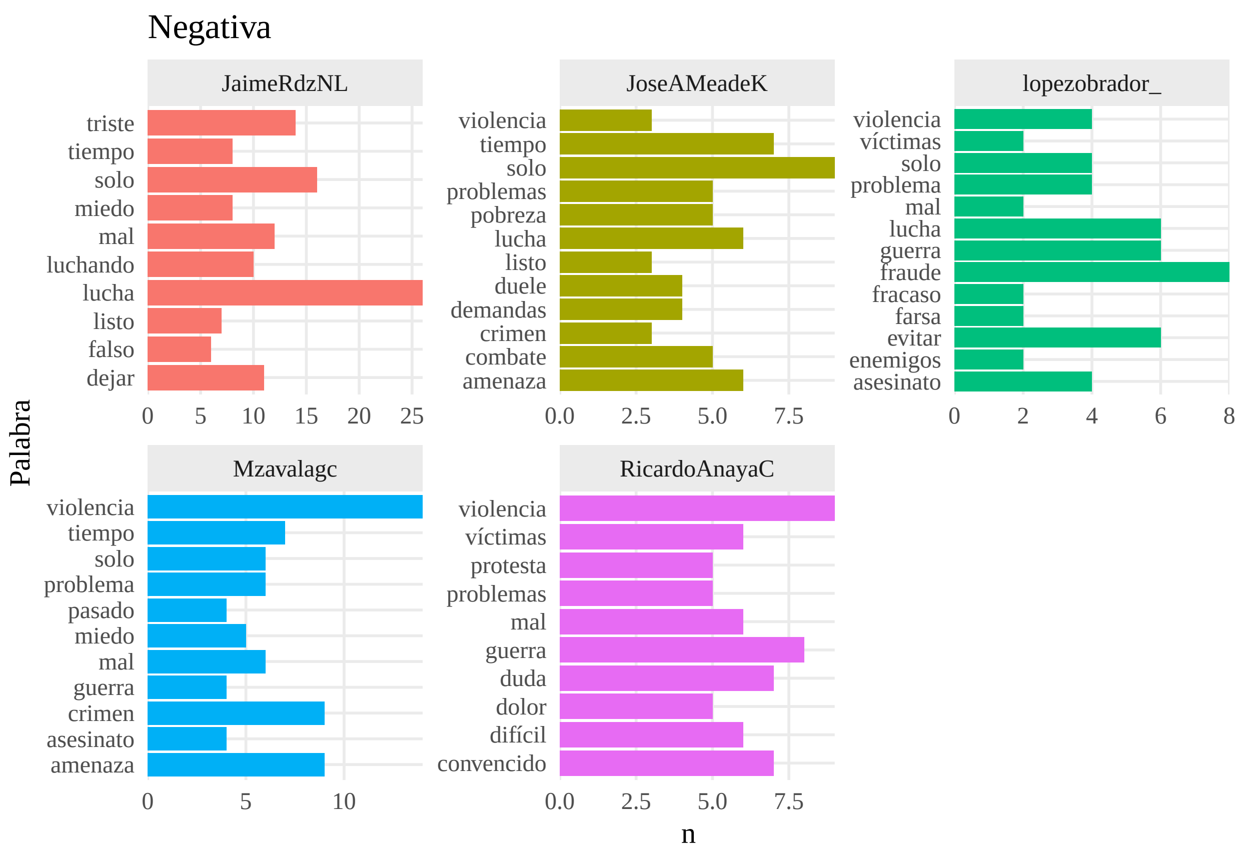
```r
map(c("Positiva", "Negativa"), function(sentimiento) {
  tuits_afinn %>%
    filter(Tipo ==  sentimiento) %>%
    group_by(Candidato) %>%
    count(Palabra, sort = T) %>%
    top_n(n = 10, wt = n) %>%
    ggplot() +
    aes(Palabra, n, fill = Candidato) +
    geom_col() +
```

```
    facet_wrap("Candidato", scales = "free") +
    scale_y_continuous(expand = c(0, 0)) +
    coord_flip() +
    labs(title = sentimiento) +
    tema_graf
})
```

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




```
##
## [[2]]
```

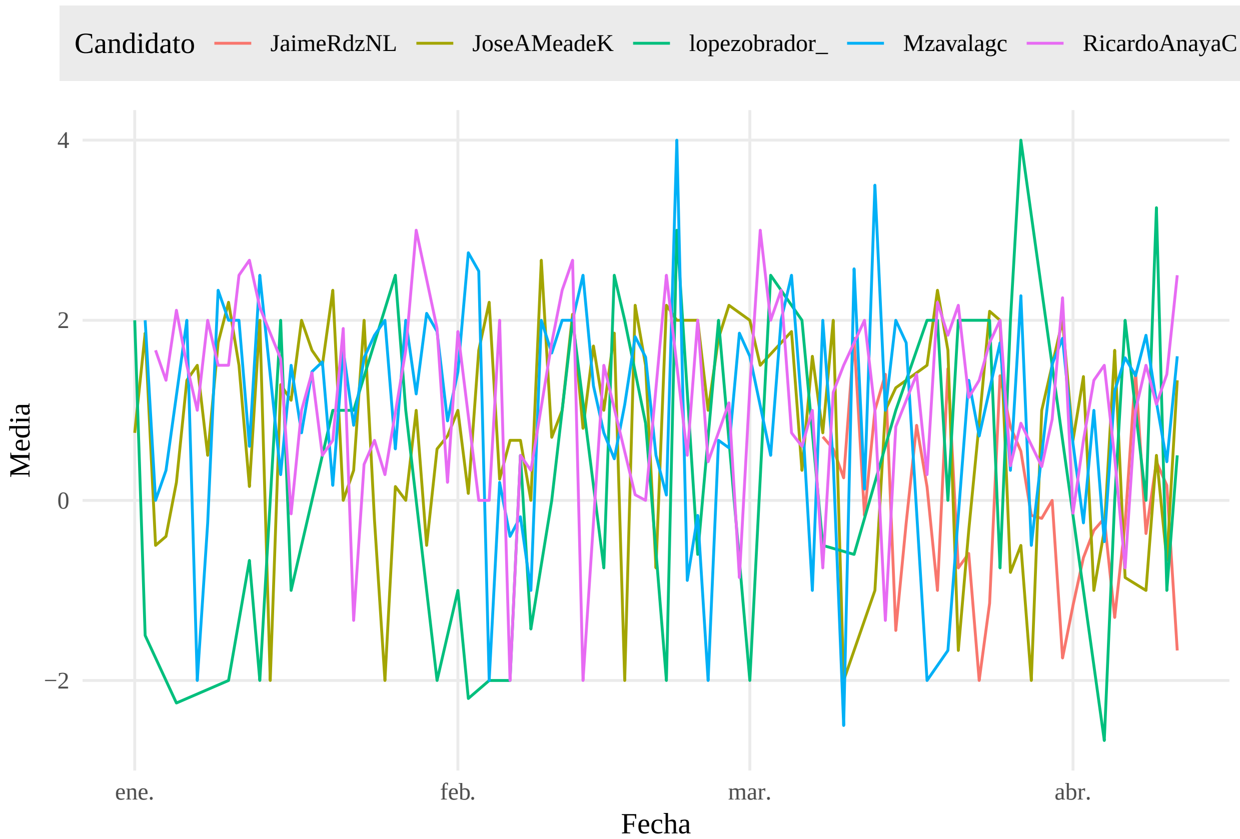


Como deseamos observar tendencias, vamos a obtener la media de sentimientos por día, usando group_by() y summarise() y asignamos los resultados a tuits_afinn_fecha

```
tuits_afinn_fecha <-
  tuits_afinn %>%
  group_by(status_id) %>%
  mutate(Suma = mean(Puntuacion)) %>%
  group_by(Candidato, Fecha) %>%
  summarise(Media = mean(Puntuacion))
```

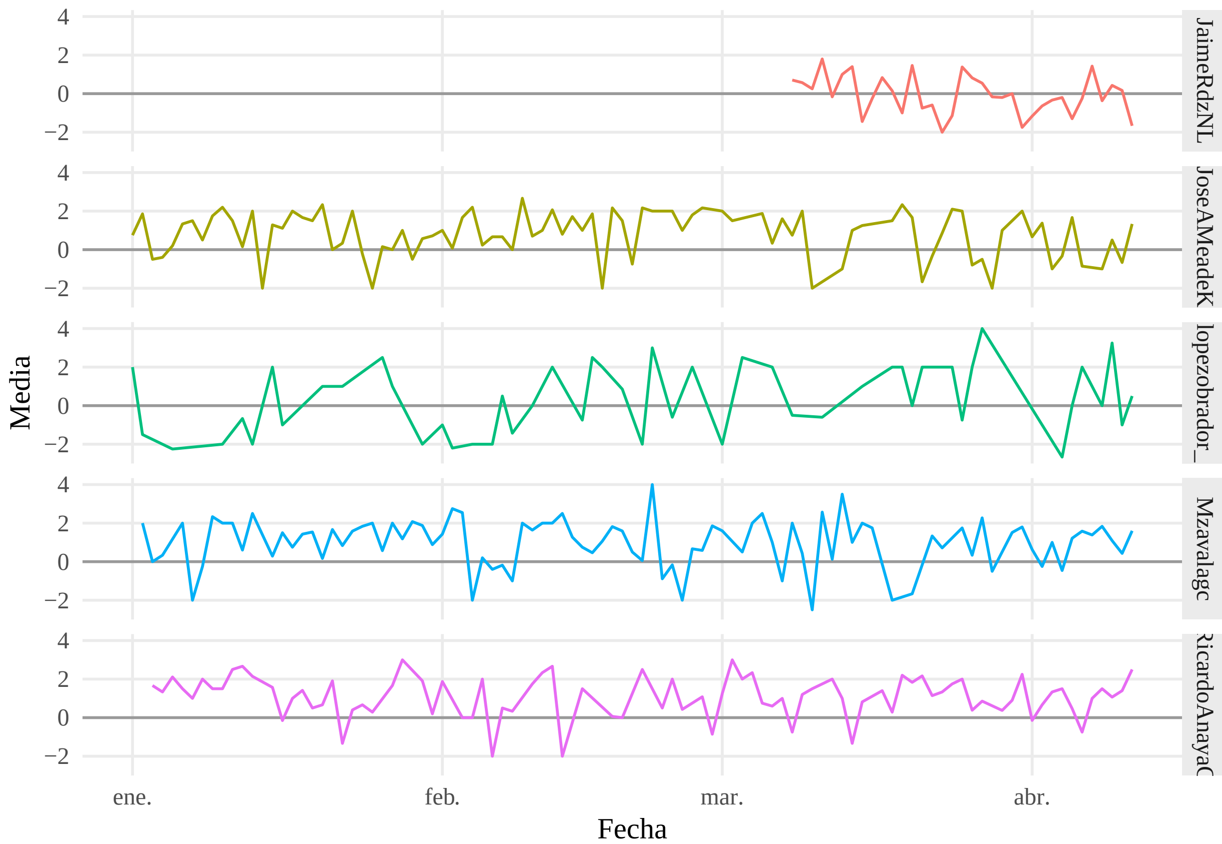
Veamos nuestros resultados con ggplot()

```
tuits_afinn_fecha %>%
  ggplot() +
  aes(Fecha, Media, color = Candidato) +
  geom_line() +
  tema_graf +
  theme(legend.position = "top")
```



No nos dice mucho. Sin embargo, si separamos las líneas por candidato, usando facet_wrap(), será más fácil observar el las tendencias de los Candidatos.

```
tuits_afinn_fecha %>%
  ggplot() +
  aes(Fecha, Media, color = Candidato) +
  geom_hline(yintercept = 0, alpha = .35) +
  geom_line() +
  facet_grid(Candidato~.) +
  tema_graf +
  theme(legend.position = "none")
```



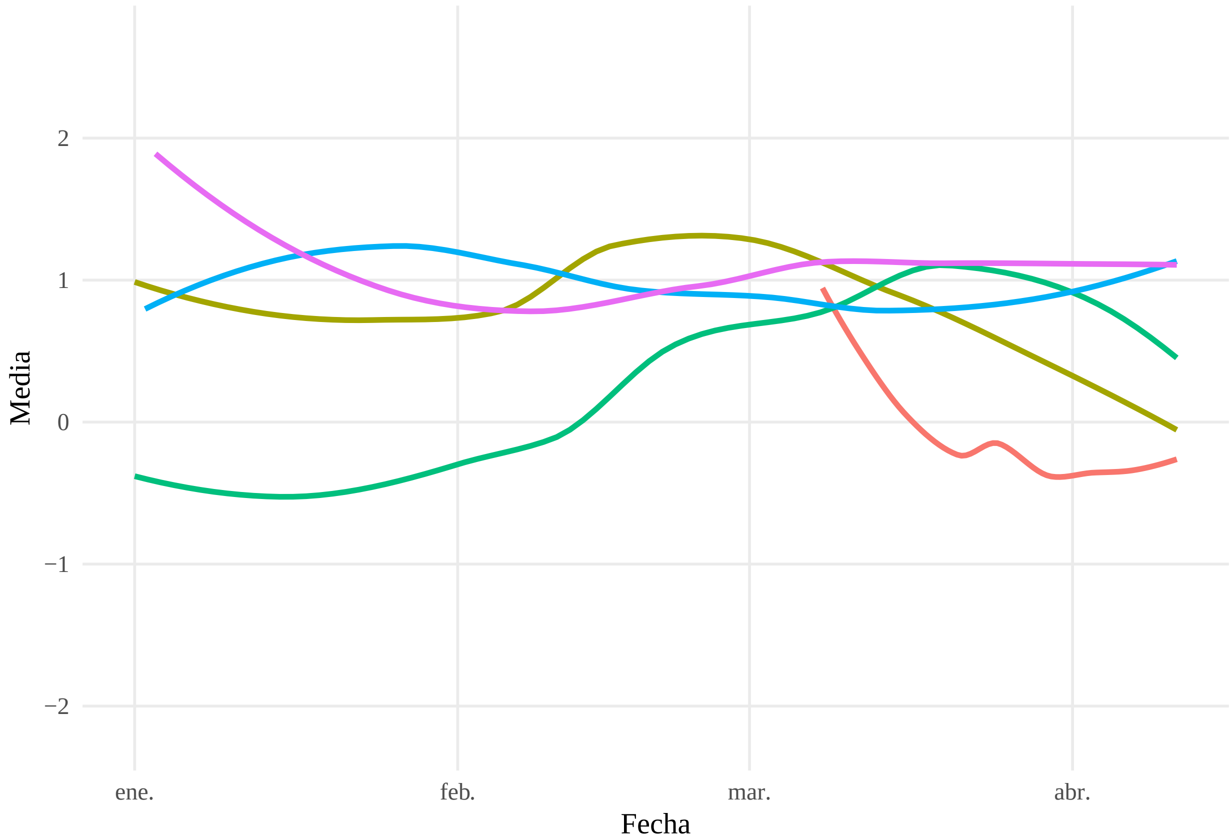
12.4 Usando LOESS (regression local)

Una manera en que podemos extraer tendencias es usar el algoritmo de regresión local LOESS. Con este algoritmo trazaremos una línea que intenta ajustarse a los datos contiguos. Como sólo tenemos una observación por día, quitaremos el sombreado que indica el error estándar.

Una explicación más completa de LOESS se encuentra aquí:

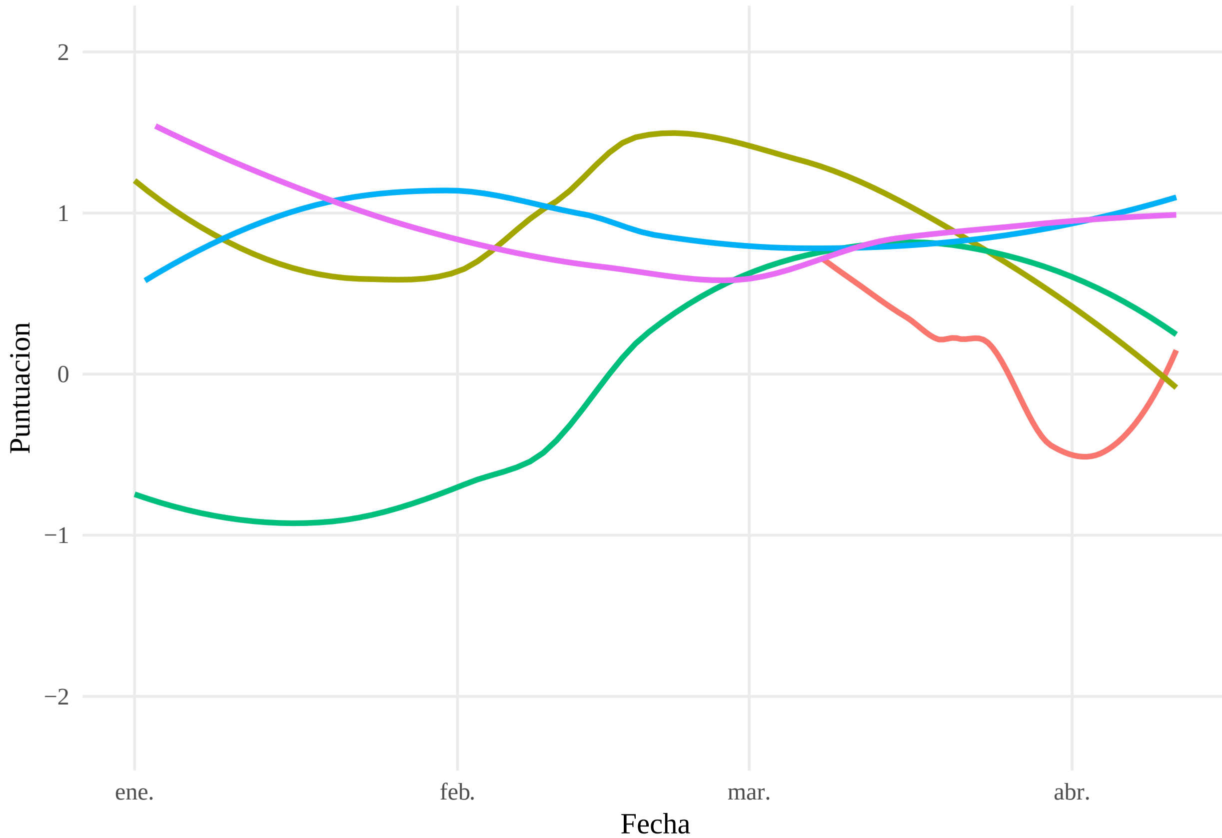
https://www.itl.nist.gov/div898/handbook/pmd/section1/pmd144.htm Usamos la función geom_smooth() de ggplot2, con el argumento method = "loess" para calcular y graficar una regresión local a partir de las medias por día.

```
tuits_afinn_fecha %>%
  ggplot() +
  aes(Fecha, Media, color = Candidato) +
  geom_smooth(method = "loess", fill = NA) +
  tema_graf
```



En realidad, podemos obtener líneas muy similares directamente de las puntuaciones.

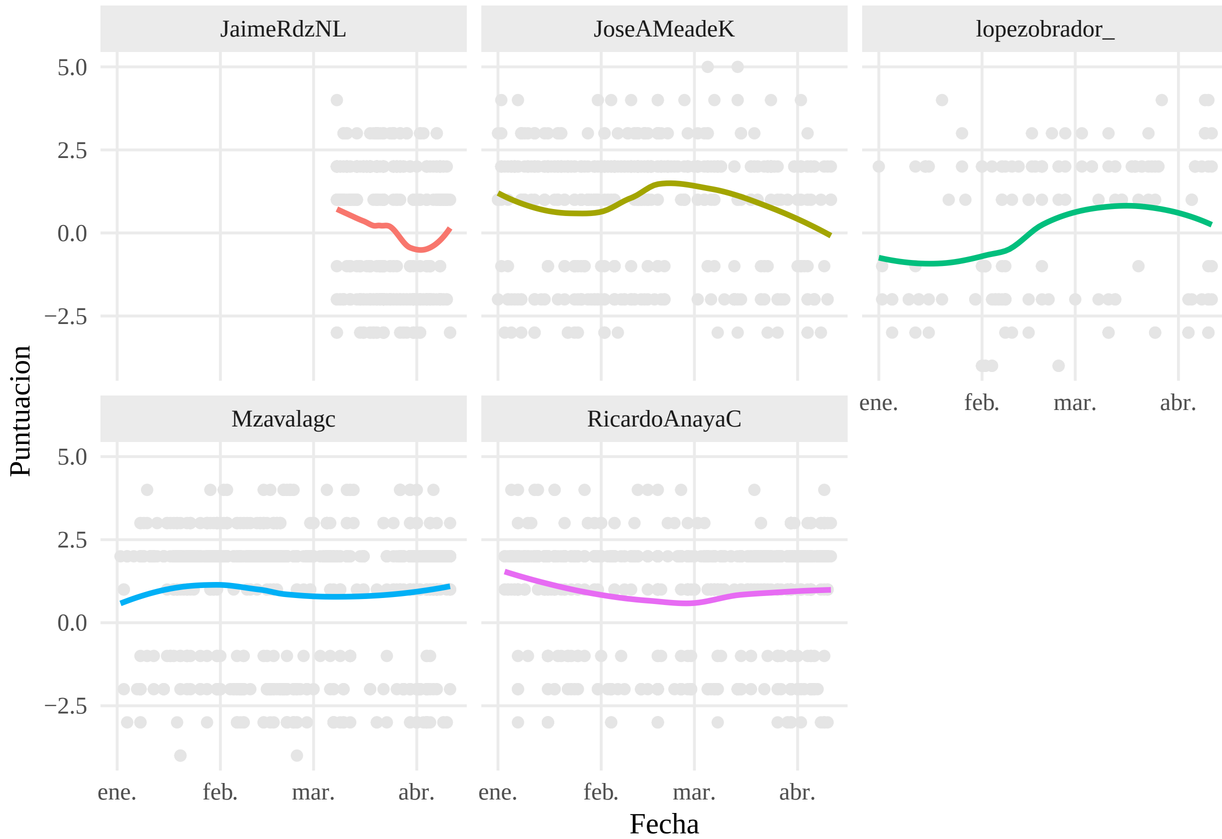
```
tuits_afinn %>%
  ggplot() +
  aes(Fecha, Puntuacion, color = Candidato) +
  geom_smooth(method = "loess", fill = NA) +
  tema_graf
```



la manera en que el algoritmo LOESS llega a sus resultados. También es manera de observar que este algoritmo no nos permite obtener una formula de regresión, de la misma manera que lo haríamos

Si separamos las lineas por candidato y mostramos los puntos a partir de los cuales se obtienen las líneas de regresión, podemos observar con más claridad la manera en que el algoritmo LOESS llega a sus resultado. Haremos esto con facet_wrap() y geom_point.

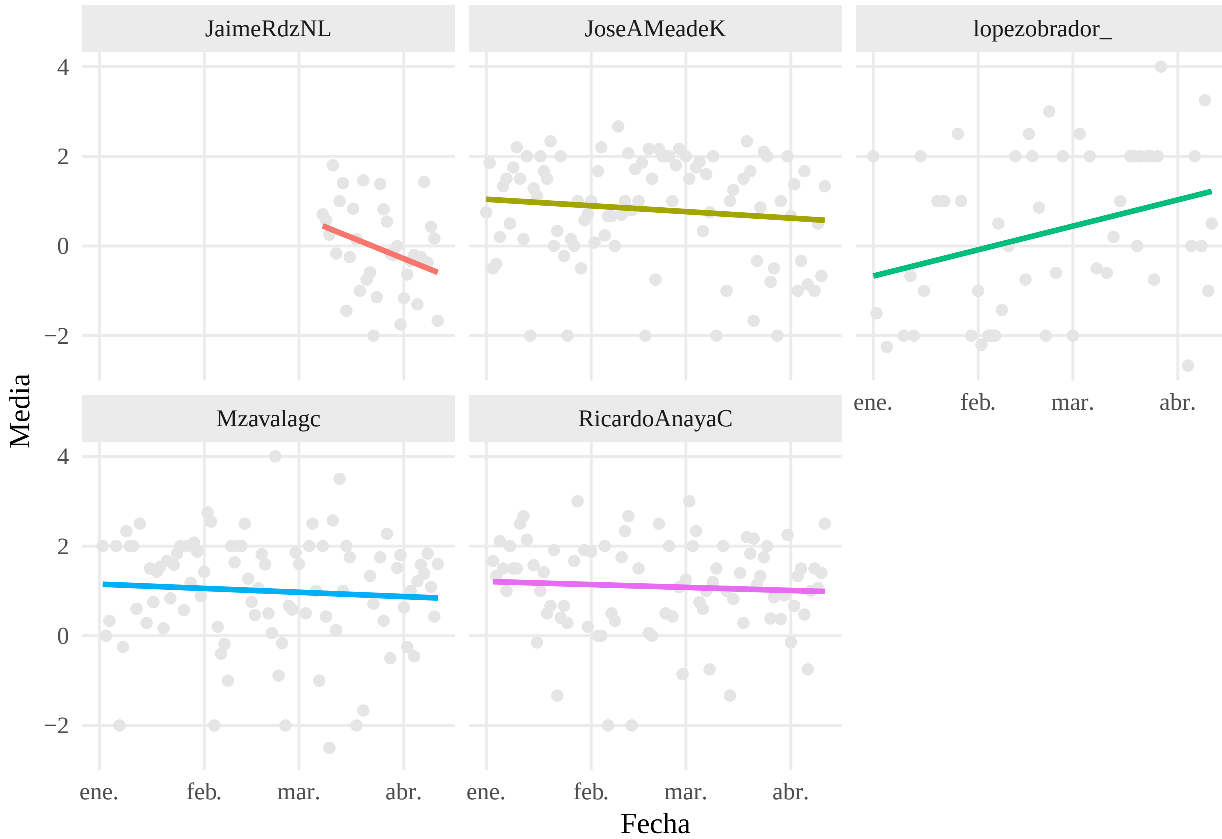
```
tuits_afinn %>%
  ggplot() +
  aes(Fecha, Puntuacion, color = Candidato) +
  geom_point(color = "#E5E5E5") +
  geom_smooth(method = "loess", fill = NA) +
  facet_wrap(~Candidato) +
  tema_graf
```



Esto es conveniente, pues podemos identificar tendencias de datos que en apariencia no tienen ninguna. Al mismo tiempo, esto es una desventaja, pues podemos llegar a sobre ajustar la línea de regresión y, al interpretarla, llegar a conclusiones que no siempre son precisas.

Comparemos los resultados de al algoritmo LOESS con los resultados de una Regresión Lineal ordinaria, que intentará ajustar una recta.

```
tuits_afinn_fecha %>%
  ggplot() +
  aes(Fecha, Media, color = Candidato) +
  geom_point(color = "#E5E5E5") +
  geom_smooth(method = "lm", fill = NA) +
  facet_wrap(~Candidato) +
  tema_graf
```



observar una tendencia, pero en la mayoría de los casos no es tan "clara" como parecería usando LOESS. También podemos ver cómo es que pocos datos, es posible que valores extremos cambien notablemente la forma de una línea trazada con LOESS, de manera similar a cómo cambian la pendiente de una Regresión Lineal ordinaria. Esto es osbervable con los datos de lopezobrador_.

Para nuestros fines, LOESS es suficiente para darnos un panorama general en cuanto a la tendencia de sentimientos en los candidatos. No obstante, es importante ser cuidadosos con las interpretaciones que hagamos.

12.5 Usando la media móvil

La media móvil se obtiene a partir de subconjuntos de datos que se encuentran ordenados. En nuestro ejemplo, tenemos nuestros datos ordenados por fecha, por lo que podemos crear subconjuntos de fechas consecutivas y obtener medias de ellos. En lugar de obtener una media de puntuación de todas las fechas en nuestros datos, obtenemos una media de los días 1 al 3, después de los días 2 al 4, después del 3 al 5, y así sucesivamente hasta llegar al final de nuestras fechas.

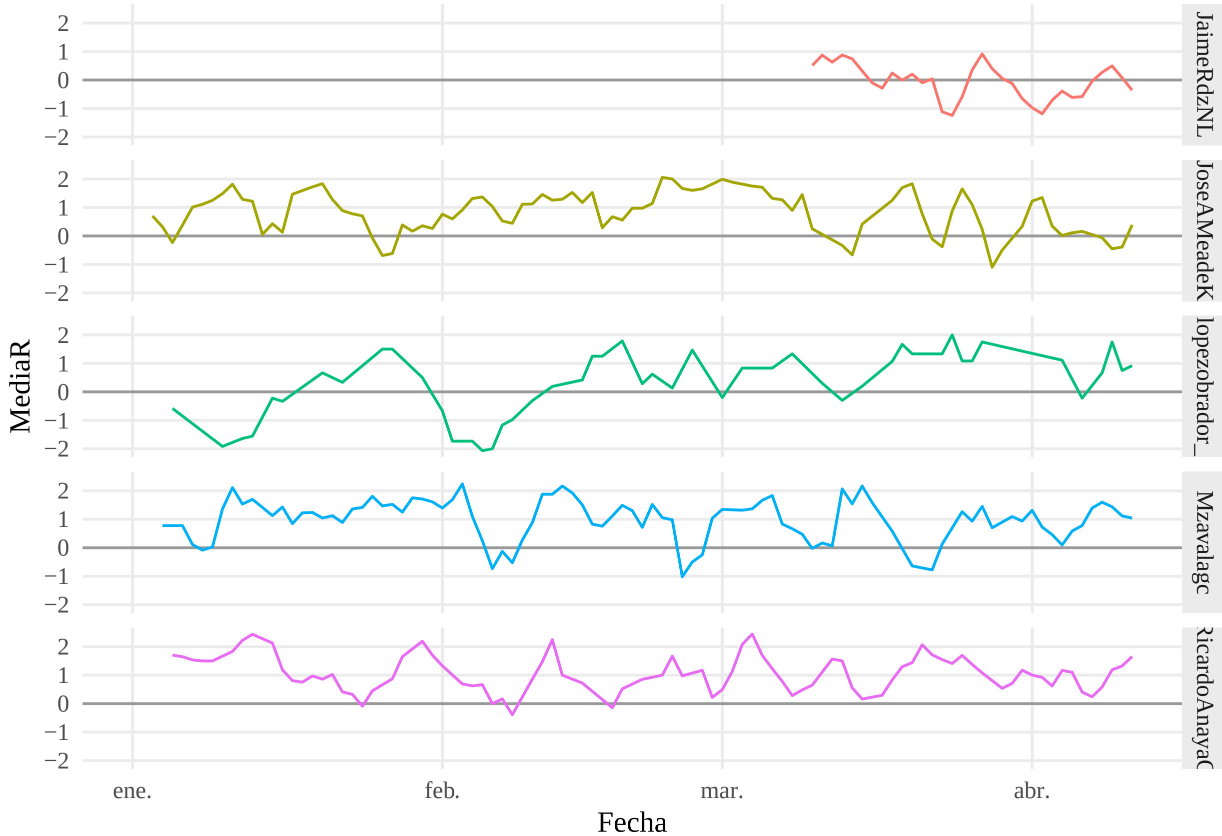
Lo que obtendríamos con esto son todos los agregados de tres días consecutivos, que en teoría debería ser menos fluctuantes que de los días individuales, es decir, más estables y probablemente más apropiados para identificar tendencias.

Crearemos medias móviles usando rollmean() de zoo. Con esta función calculamos la media de cada tres días y la graficamos con ggplot.

```
tuits_afinn_fecha %>%
  group_by(Candidato) %>%
  mutate(MediaR = rollmean(Media, k = 3, align = "right", na.pad = TRUE)) %>%
  ggplot() +
  aes(Fecha, MediaR, color = Candidato) +
```

```
  geom_hline(yintercept = 0, alpha = .35) +
  geom_line() +
  facet_grid(Candidato~.) +
  tema_graf
```

```
## Warning: Removed 10 rows containing missing values (geom_path).
```

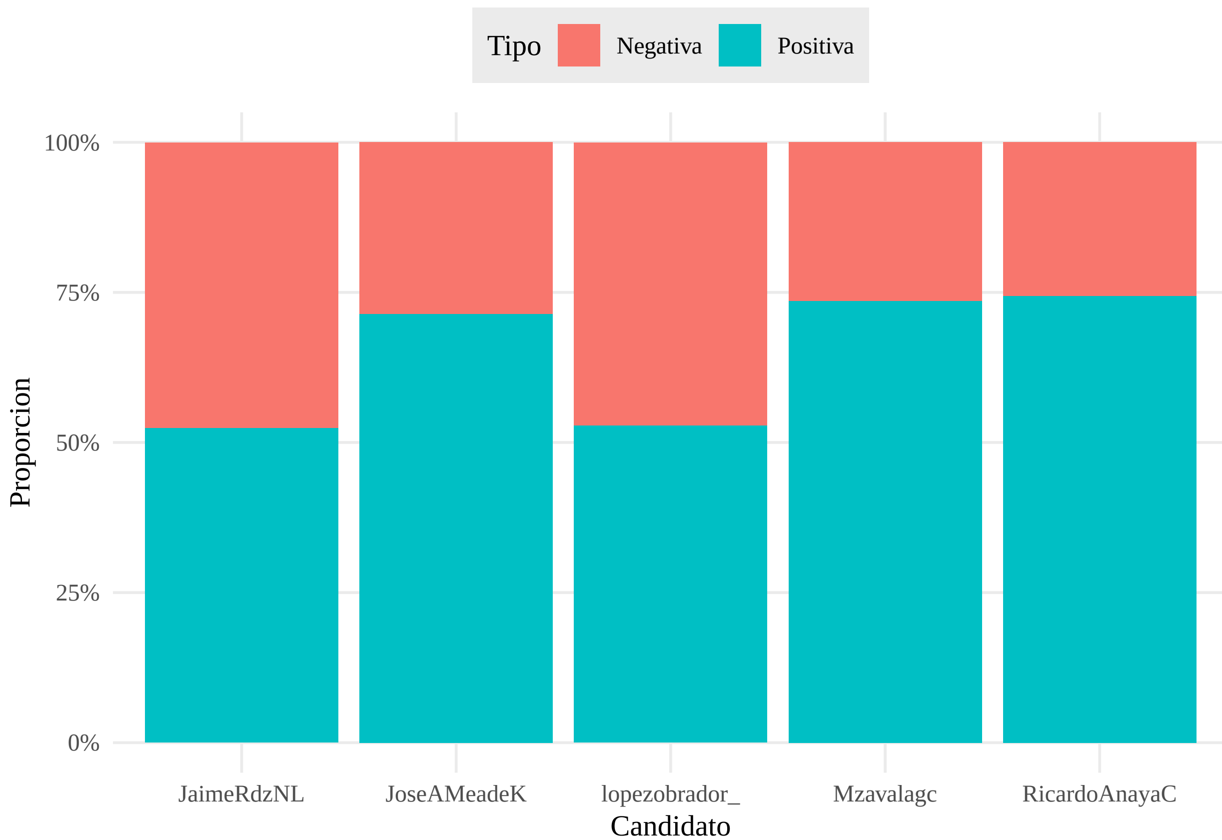


Si comparamos con la gráfica que obtuvimos a partir de las medias por día, esta es menos "ruidosa" y nos permite observar más fácilmente las tendencias.

Comparando sentimientos positivos y negativos Es posible que no nos interen las puntuaciones de sentimiento de cada día, sólo si la tendencia ha sido positiva o negativa. Como ya etiquetamos la puntuación de nuestros tuits como "Positiva" y "Negativa", sólo tenemos que obtener proporciones y graficar.

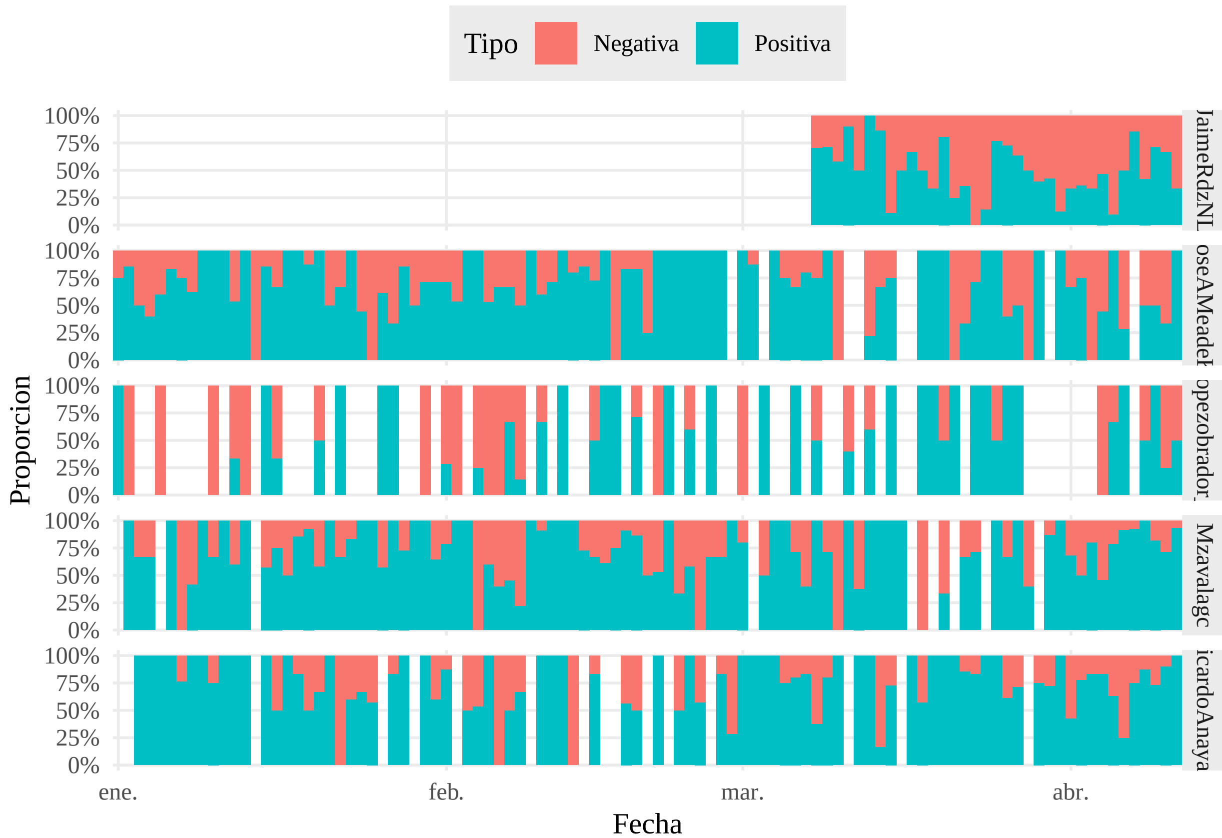
Primero, veamos que proporción de tuits fueron positivos y negativos, para todo el 2018 y para cada Candidato. Usamos geom_col() de ggplot2 para elegir el tipo de gráfica y la función percent_format() de scales para dar formato de porcentaje al eje y.

```
tuits_afinn %>%
  count(Candidato, Tipo) %>%
  group_by(Candidato) %>%
  mutate(Proporcion = n / sum(n)) %>%
  ggplot() +
  aes(Candidato, Proporcion, fill = Tipo) +
  geom_col() +
  scale_y_continuous(labels = percent_format()) +
  tema_graf +
  theme(legend.position = "top")
```



Si obtenemos la proporción de positiva y negativa por día, podemos obsrvar cómo cambia con el paso del tiempo. Usamos el argumento width = 1 de geom_col() para quitar el espacio entre barras individuales y el argumento expand = c(0, 0) de scale_x_date() para quitar el espacio en blanco en los extremos del eje x de nuestra gráfica (intenta crear esta gráfica sin este argumento para ver la diferencia).

```
tuits_afinn %>%
  group_by(Candidato, Fecha) %>%
  count(Tipo) %>%
  mutate(Proporcion = n / sum(n)) %>%
  ggplot() +
  aes(Fecha, Proporcion, fill = Tipo) +
  geom_col(width = 1) +
  facet_grid(Candidato~.) +
  scale_y_continuous(labels = percent_format()) +
  scale_x_date(expand = c(0, 0)) +
  tema_graf +
  theme(legend.position = "top")
```



En este ejemplo, como los candidatos no tuitearon todos los días, tenemos algunos huecos en nuestra gráfica. De todos modos es posible observar la tendencia general de la mayoría de ellos.

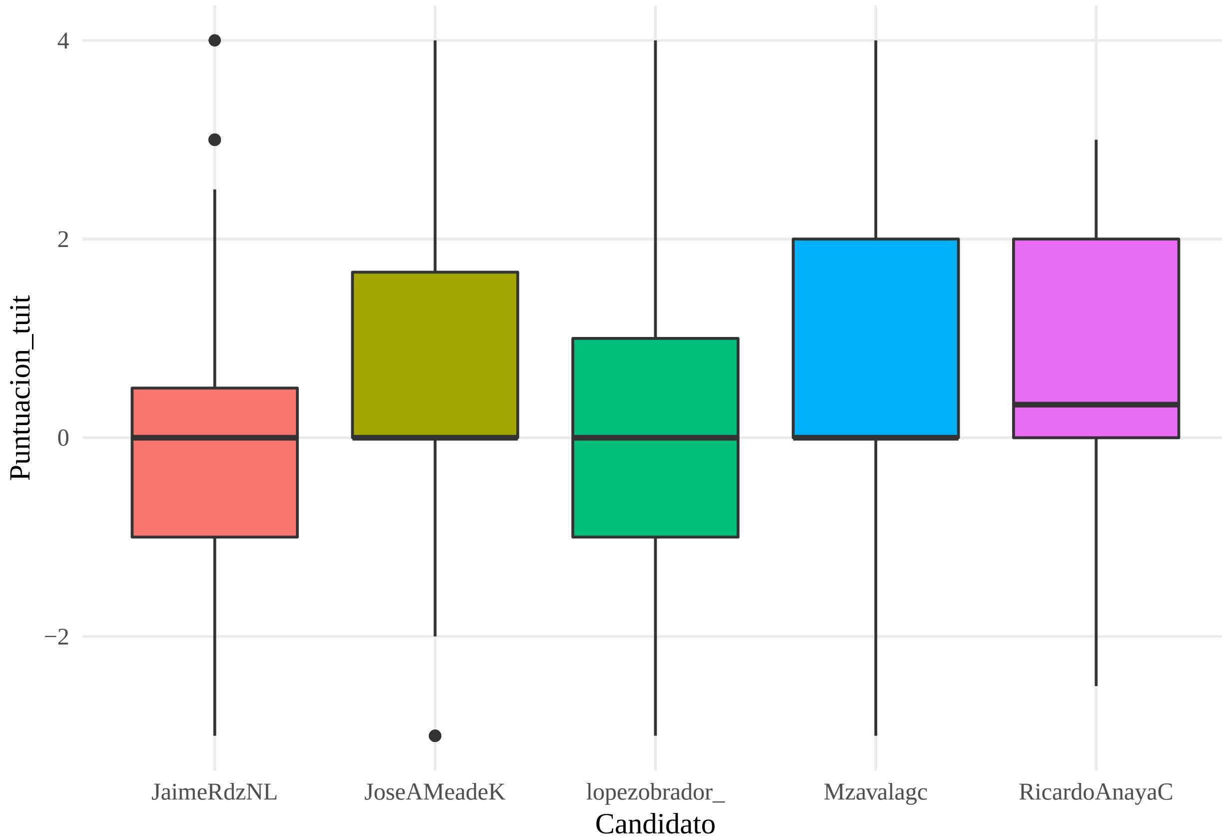
12.6 Bloxplots (diagrama caja y bigotes)

Una manera más en la que podemos visualizar la puntuación sentimientos es usando boxplots. En nuestro análisis quizás no es la manera ideal de presentar los resultados dado que tenemos una cantidad relativamente baja de casos por Candidato. Sin embargo, vale la pena echar un vistazo, pues es una herramienta muy útil cuando tenemos una cantidad considerable de casos por analizar.

En este tipo de gráficos, la caja representa el 50% de los datos, su base se ubica en el primer cuartil (25% de los datos debajo) y su tope en el tercer cuartil (75% de los datos debajo). La línea dentro de la caja representa la mediana o secundo cuartil (50% de los datos debajo). Los bigotes se extienden hasta abarcar un largo de 1.5 veces el alto de la caja, o hasta abarcar todos los datos, lo que ocurra primero. Los puntos son los outliers, datos extremos que salen del rango de los bigotes. Por todo lo anterior, esta visualización es preferible cuando tenemos datos con distribuciones similares a una normal.

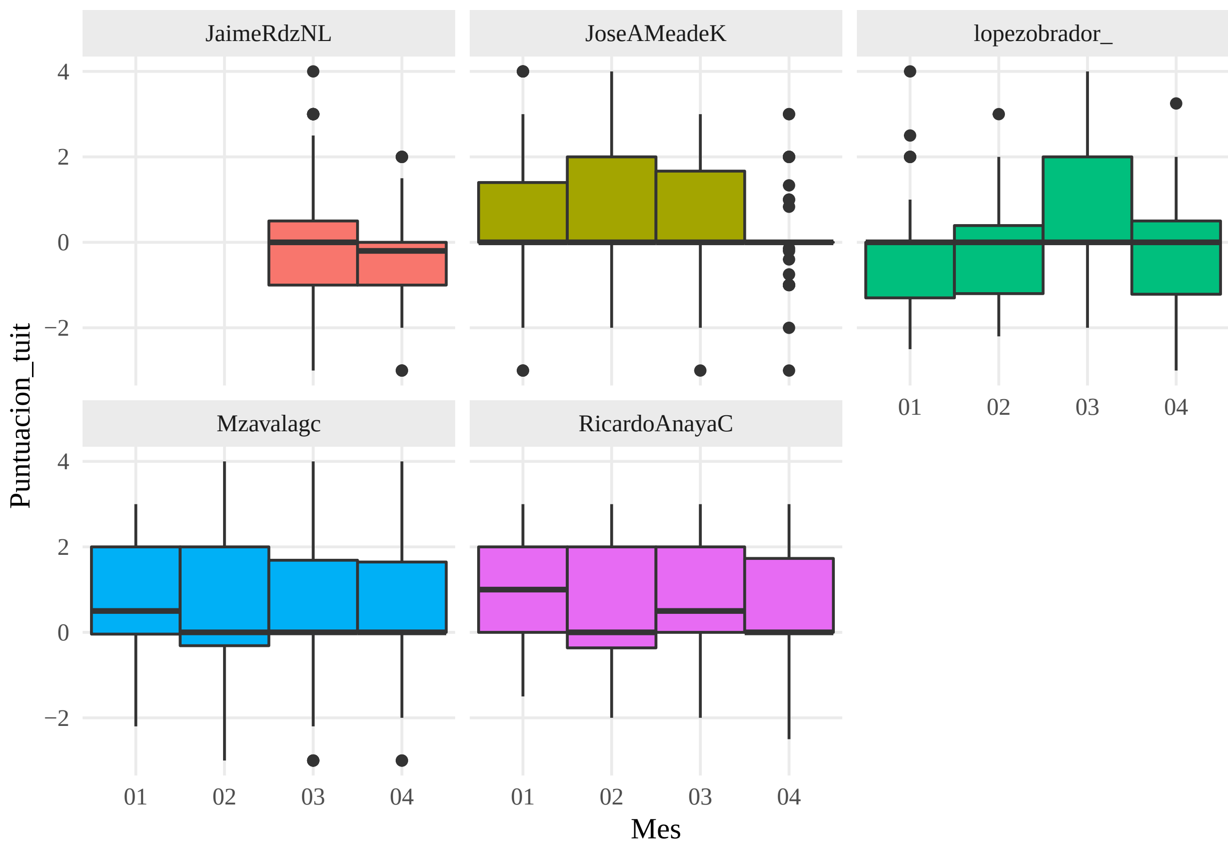
Usamos la función geom_boxplot() de ggplot2 para elegir el tipo de gráfica. Creamos un boxplot por candidato.

```
tuits %>%
  ggplot() +
  aes(Candidato, Puntuacion_tuit, fill = Candidato) +
  geom_boxplot() +
  tema_graf
```



También podemos crear boxplots para ver cambios a través del tiempo, sólo tenemos que agrupar nuestros datos. Como nuestros datos ya tienen una columna para el mes del año, usaremos esa como variable de agrupación. Nota que usamos factor() dentro de mutate() para cambiar el tipo de dato de Mes, en R los boxplots necesitan una variable discreta en el eje x para mostrarse correctamente.

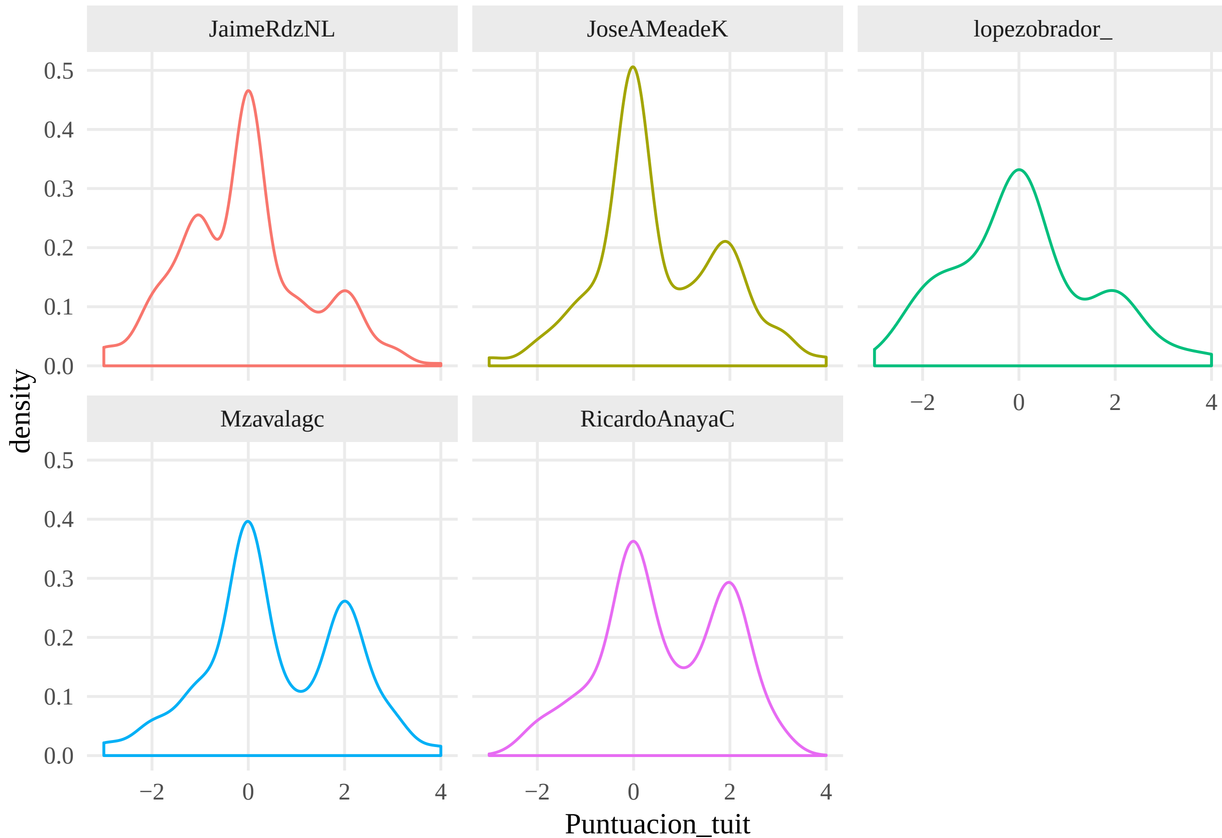
```
tuits %>%
  mutate(Mes = factor(Mes)) %>%
  ggplot() +
  aes(Mes, Puntuacion_tuit, fill = Candidato) +
  geom_boxplot(width = 1) +
  facet_wrap(~Candidato) +
  tema_graf +
  theme(legend.position = "none")
```



12.7 Usando densidades

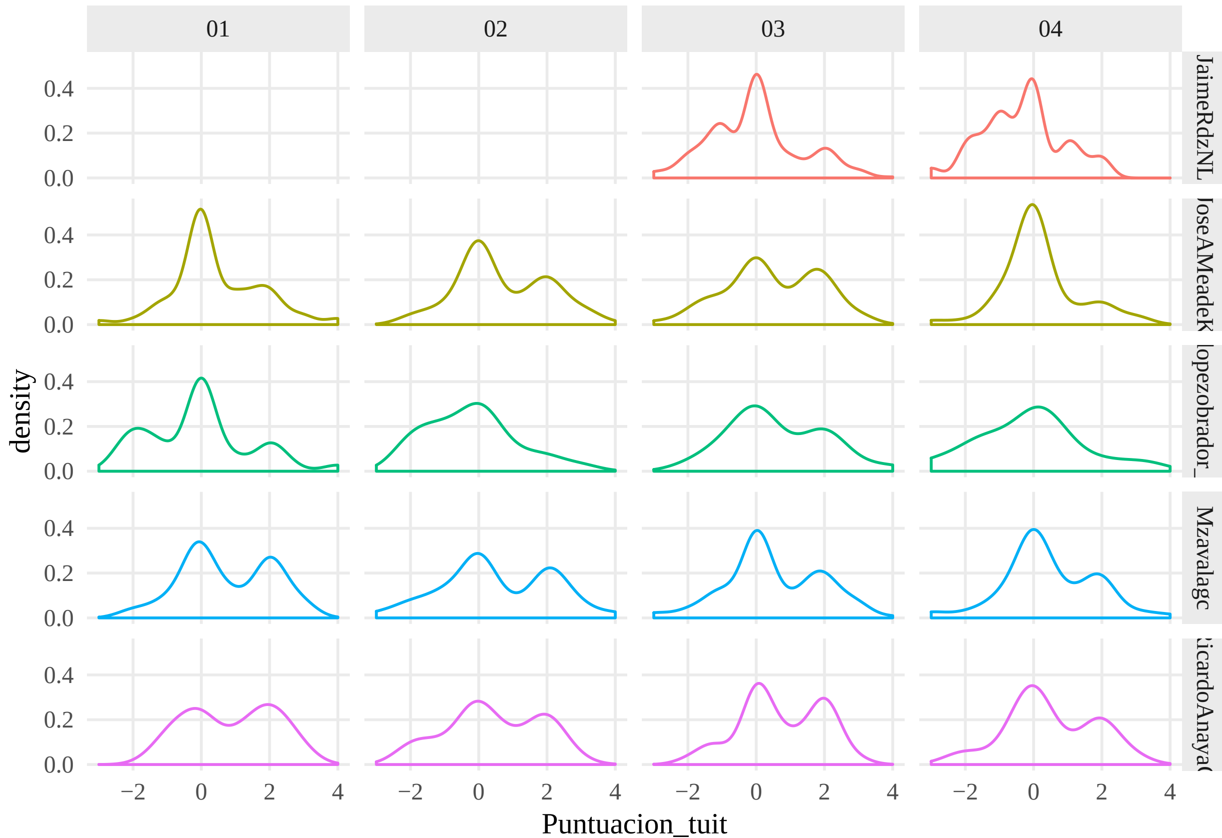
Por último, podemos analizar las tendencias de sentimientos usando las funciones de densidad de las puntuaciones. ggplot2 tiene la función geom_density() que hace muy fácil crear y graficar estas funciones.

```
tuits %>%
  ggplot() +
  aes(Puntuacion_tuit, color = Candidato) +
  geom_density() +
  facet_wrap(~Candidato) +
  tema_graf
```



Por supuesto, también podemos observar las tendencias a través del tiempo usando facet_grid() para crear una cuadrícula de gráficas, con los candidatos en el eje x y los meses en el eje y.

```
tuits %>%
  ggplot() +
  aes(Puntuacion_tuit, color = Candidato) +
  geom_density() +
  facet_grid(Candidato~Mes) +
  tema_graf
```



Para concluir En este artículo revisamos algunas de las estrategias principales para analizar sentimientos con R, usando el léxico Afinn. Este léxico le asigna una puntuación a las palabras, de acuerdo a su contenido, que puede ser positivo o negativo.

En realidad, que la puntuación sea de tipo numérico es lo nos abre una amplia gama de posibilidades para analizar sentimientos usando el léxico Afinn. Con conjuntos de datos más grandes que el que usamos en este ejemplo, es incluso plausible pensar en análisis más complejos, por ejemplo, establer correlaciones y crear conglomerados.

Aunque no nos adentramos al análisis de los resultados que obtuvimos con nuestros datos, algunas tendencias se hicieron evidentes rápidamente. Por ejemplo, la mayoría de los candidatos ha tendido a tuitear de manera positiva. Con un poco de conocimiento del tema, sin duda podríamos encontrar información útil e interesante.

13 Games.

This has to be done only once:

```
#install.packages("devtools")
#devtools::install_github('RLesur/Rcade')
```

And do this to play:

```
#library(Rcade)
#Rcade::games$Pacman
```

14 R references.

The following are a constant growing list of unsorted online R references.

R is a free software environment for statistical computing and graphics https://www.r-project.org/

R - Install R and R Studio on Windows 10 for PC: https://youtu.be/9-RrkJQQYqY MAC: https://youtu.be/GLLZhc_5enQ

http://stat.slu.edu/~speegle/_book/index.html#installing-r

R Studio provides popular open source and enterprise-ready professional software for the R statistical computing environment https://www.rstudio.com/

R Programming for Data Science https://bookdown.org/rdpeng/rprogdatascience/

R for Data Science https://r4ds.had.co.nz/

R-ladies is a world-wide organization to promote gender diversity in the R community https://rladies.org/

Learn Data Science Online https://www.datacamp.com/

Social Science Data and Statistics Resources https://researchguides.library.tufts.edu/data/r

Daily news about using open source R for big data analysis, predictive modelling, data science, and visualization since 2008 https://blog.revolutionanalytics.com/

Stack Overflow is the largest online community for programmers to learn, share their knowledge and build their careers https://stackoverflow.com

R-Bloggers is about empowering bloggers to empower other R users https://www.r-bloggers.com

Swirl teaches you R programming and data science interactively, at your own pace, and right in the R console! https://swirlstats.com/

Foundations of Statistics with R https://mathstat.slu.edu/~speegle/_book/

Introduction to Econometrics with R https://www.econometrics-with-r.org/

The technology learning platform https://www.pluralsight.com

This website presents documents, examples, tutorials and resources on R and data mining http://www.rdatamining.com

rstats (as a hashtag in Twitter)

DataScience (as a hashtag in Twitter)

https://www.pluralsight.com/search?q=R

http://wdb.ugr.es/~bioestad/

Social Science Data and Statistics Resources http://researchguides.library.tufts.edu/data/r

Milestones in AI, Machine Learning, Data Science, and visualization with R and Python since 2008 http://blog.revolutionanalytics.com/

http://www.datasciencecentral.com/profiles/blogs/list-of-companies-using-r