Regular expressions and text mining

String manipulations and regex

Working with strings

- In R, a piece of text is represented as a sequence of characters (letters, numbers, and symbols).
- The data type R provides for storing sequences of characters is character. Formally, the mode of an object that holds character strings in R is "character".
- You express character strings by surrounding text within double quotes:

```
x <- "This is R!"
x
```

```
## [1] "This is R!"
```

```
class(x)
```

```
## [1] "character"
```

Working with strings

You can use single quotes as well

```
x <- 'This is R!'
x
```

```
## [1] "This is R!"
```

```
class(x)
```

```
## [1] "character"
```

When writing strings, you can insert single quotes in a string with double quotes, and vice versa:

```
x <- "Why call langauge 'python' ? "
x
```

```
## [1] "Why call langauge 'python' ? "
```

```
x <- 'Why call langauge "python" ? '
x
```

```
## [1] "Why call langauge \"python\" ? "
```

If you try to include single quotes within single quotes, or double quotes within double quotes, it will not work

```
x <- "Why call langauge "python" ? "
```

```
x <- 'Why call langauge 'python' ? '
```

But if by some reason, you desperately want to do it, use escaping character (we will talk about this later)

Working with strings

Use \ as an escape before quotation mark

```
x <- 'Why call langauge \'python\' ? '
x
```

```
## [1] "Why call langauge 'python' ? "
```

```
class(x)
```

```
## [1] "character"
```

```
is.character(x)
```

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

Check if x is a character

Working with strings

- The function **paste()** is perhaps one of the most important functions that you can use to create and build strings.
- **paste()** takes one or more R objects, converts them to "character", and then it concatenates (pastes) them to form one or several character strings. Its usage has the following form:

```
PI <- paste("Life of", pi)
PI

## [1] "Life of 3.14159265358979"
```

Life of Pi

Working with strings

- Element wise concatenation of two vectors
- By default **space** is the seperator

```
x <- 1:10
a <- letters[1:10]
x
```

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

```
a
```

```
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j"
```

```
k <- paste(x,a)
k
```

```
##  [1] "1 a"  "2 b"  "3 c"  "4 d"  "5 e"  "6 f"  "7 g"  "8 h"  "9 i"  "10 j"
```

Working with strings

Define your own seperator

```
k <- paste(x,a, sep="-")
k
```

```
##  [1] "1-a"  "2-b"  "3-c"  "4-d"  "5-e"  "6-f"  "7-g"  "8-h"  "9-i"  "10-j"
```

```
length(k)
```

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

The argument collapse is an optional string to indicate if you want all the terms to be collapsed into a single string

```
k <- paste(x,a, sep="-", collapse = ',')
k
```

```
## [1] "1-a,2-b,3-c,4-d,5-e,6-f,7-g,8-h,9-i,10-j"
```

```
length(k)
```

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

Basic string manipulations

Function	Description
`nchar()`	number of characters
`tolower()`	convert to lower case
`toupper()`	convert to upper case
`casefold()`	case folding
`chartr()`	character translation
`abbreviate()`	abbreviation
`substring()`	substrings of a character vector
`substr()`	substrings of a character vector

nchar() counts the number of characters in the string

```
x <- 'This is R!'
nchar(x)
```

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

If you apply it to the character vector, you will get back a vector with number of characters for each element

Built in vector with month names

```
month.name
```

```
##  [1] "January"   "February"  "March"     "April"     "May"
##  [6] "June"      "July"      "August"    "September" "October"
## [11] "November"  "December"
```

```
nchar(month.name)
```

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

upper and lower cases

```
toupper(x)
```

```
## [1] "THIS IS R!"
```

```
tolower(x)
```

```
## [1] "this is r!"
```

```
casefold(x, upper=T)
```

```
## [1] "THIS IS R!"
```

```
casefold(x, upper=F)
```

```
## [1] "this is r!"
```

Working with strings

Abbreviations are handy especially when you need to use them as labels/legends for the plots

```
summer <- read.csv('summer.csv', stringsAsFactors = F)
discipline <- unique(summer$Discipline)
discipline[1:10]
```

```
##  [1] "Swimming"        "Athletics"       "Cycling Road"
##  [4] "Cycling Track"   "Fencing"         "Artistic G."
##  [7] "Shooting"        "Tennis"          "Weightlifting"
## [10] "Wrestling Gre-R"
```

```
disc_abbr <- abbreviate(discipline, minlength = 5)
disc_abbr[1:10]
```

```
##        Swimming       Athletics    Cycling Road   Cycling Track
##         "Swmmn"         "Athlt"         "CyclR"         "CyclT"
##         Fencing     Artistic G.        Shooting          Tennis
##         "Fncng"         "ArtG."         "Shtng"         "Tenns"
##   Weightlifting Wrestling Gre-R
##         "Wghtl"         "WGr-R"
```

Working with strings

One common operation when working with strings is the extraction and replacement of some characters. There a various ways in which characters can be replaced. If the replacement is based on the positions that characters occupy in the string, you can use the functions **substr()** and **substring()**

substr() extracts or replaces substrings in a character vector. Its usage has the following form:

```
x <- "ABCDEF"
substr(x, start=3,stop =5)
```

```
## [1] "CDE"
```

Working with strings

You can replace the portion of the string with other string

```
y <- c("may", "the", "force", "be", "with", "you")
substr(y, 2, 2) <- "#"
y
```

```
## [1] "m#y"   "t#e"   "f#rce" "b#"    "w#th"  "y#u"
```

```
y <- c("may", "the", "force", "be", "with", "you")
substr(y, 2, 3) <- ":)"
y
```

```
## [1] "m:)"   "t:)"   "f:)ce" "b:"    "w:)h"  "y:)"
```

Star Wars

Working with strings

- **stringr**
- Powerful but easy to learn
- Built on stringi

- Concise and consistent
    - All functions start with str_
    - All functions take a vector of strings as the first argument

```
install.packages('stringr')
```

Works the same way as substr() but allows for negative subsetting as well

```
library(stringr)
str_sub("ABCDEF", start=1, end=3)
```

```
## [1] "ABC"
```

Getting the last two characters

```
str_sub("ABCDEF", start=-2, end=-1)
```

```
## [1] "EF"
```

str_detect() detects the presence or absence of a pattern and returns a logical vector

```
geners <- c("Action, Adventure, Comedy","Comedy",
"Comedy, Drama, Drama, Romance",
"Crime, Drama, History")
str_detect(string = geners,  pattern = "Drama")
```

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

str_subset() Keep strings matching a pattern

```
str_subset(string = geners,  pattern = "Drama")
```

```
## [1] "Comedy, Drama, Drama, Romance" "Crime, Drama, History"
```

str_count() Count the number of matches in a string.

```
str_count(string = geners, pattern = "Drama")
```

```
## [1] 0 0 2 1
```

Working with strings

How many romcom movies are there ?
Romcom is the abbreviation for Romantic Comedy

```
movies <- read.csv('movies3.csv', stringsAsFactors = F)
movies$Comedy <- str_detect(string = movies$Genre,  pattern = "Comedy")
movies$Romance <- str_detect(string = movies$Genre,  pattern = "Romance")
table(Romance= movies$Romance, Comedy=movies$Comedy)
```

```
##         Comedy
## Romance FALSE TRUE
##   FALSE  1478  886
##   TRUE    202  319
```

str_split() Split up a string into pieces, the result is a list of vectors for each string

```
str_split(geners, ",")
```

```
## [[1]]
## [1] "Action"      " Adventure" " Comedy"
##
## [[2]]
## [1] "Comedy"
##
## [[3]]
## [1] "Comedy"    " Drama"    " Drama"    " Romance"
##
## [[4]]
## [1] "Crime"     " Drama"    " History"
```

Use simplify=T to get back a character matrix

```
gen_m <- str_split(geners, ",", simplify = T)
gen_m
```

```
##      [,1]     [,2]          [,3]       [,4]
## [1,] "Action" " Adventure" " Comedy"  ""
## [2,] "Comedy" ""           ""         ""
## [3,] "Comedy" " Drama"     " Drama"   " Romance"
## [4,] "Crime"  " Drama"     " History" ""
```

str_replace() Replace matched patterns in a string.

```
str_replace(geners, pattern = ",", replacement = " &")
```

```
## [1] "Action & Adventure, Comedy"     "Comedy"
## [3] "Comedy & Drama, Drama, Romance" "Crime & Drama, History"
```

Only the first match is replaced

For all matches use str_replace_all

```
str_replace_all(geners, pattern = ",", replacement = " &")
```

```
## [1] "Action & Adventure & Comedy"      "Comedy"
## [3] "Comedy & Drama & Drama & Romance" "Crime & Drama & History"
```

Working with strings

- stre_extract() extracts the pattern from the text
- Return NA if the pattern is not matched

- **str_extract()** returns the first match, use **str_extract_all()** for all matches

```
str <-c("123abd", "ab567cd", "abc5.00")
str_extract(str, pattern="123")
```

```
## [1] "123" NA    NA
```

Regular Expressions

Regular expressions

A **regular expression**, **regex** is, in theoretical computer science and formal language theory, a sequence of characters that define a search pattern. Usually this pattern is then used by string searching algorithms for "find" or "find and replace" operations on strings, or for input validation.

A **regular expression** is a specific pattern in a set of strings. For instance, one could have the following pattern : 2 digits, 2 letters and 4 digits.

The most basic type of regular expressions are the literal characters which are characters that match themselves.

However, not all characters match themselves. Any character that is not a literal character is a metacharacter. This type of characters have a special meaning and they allow you to transform literal characters in very powerful ways.

Metacharacters

```
. \ | ( ) [ ] { } $ - ^ * + ?
```

If you want to match the metacharacter you need to use \ (escape) before the metacharacter

The first metacharacter you should learn about is the dot or period **"."**, better known as the wild metacharacter. This metacharacter is used to match ANY character except for a new line.

To actually match the dot character, what you need to do is escape the metacharacter. In most languages, the way to escape a metacharacter is by adding a backslash character in front of the metacharacter: "\.". When you use a backslash in front of a metacharacter you are “escaping” the character, this means that the character no longer has a special meaning, and it will match itself.

However, R is a bit different. Instead of using a backslash you have to use two backslashes: "5\\.00". This is because the backslash "\", which is another metacharacter, has a special meaning in R.

All strings in the vector are matches

```
str <-c("123abd", "ab567cd", "abc5.00")
str_subset(string = str,  pattern = ".")
```

```
## [1] "123abd"  "ab567cd" "abc5.00"
```

Use escape

```
str_subset(string = str,  pattern = "\\.")
```

```
## [1] "abc5.00"
```

Escape syntax for metacharacters

Metacharacter	Literal Meaning	Escape Syntax
.	period or dot	\\.
$	dollar sign	\\$
*	asterisk	*
+	plus sign	\\+
?	question mark	\\?
|	vertical bar	\\\|
\\	double backslash	\\\\
^	caret	\\^
[	square bracket	\\[
{	curly brace	\\{
(	parenthesis	\\(

*adapted from *Handling and Processing Strings in R* (Sanchez, 2013)

Other metacharacters:

^ indicates the beginning of the string
The following regex will match all the strings that start with “ab” characters

```
str <-c("123abd", "ab567cd", "abc5.00")
str_subset(string = str,  pattern = "^ab")
```

```
## [1] "ab567cd" "abc5.00"
```

- The $ sign indicates the end of the string
- The following regex will match all the strings ending with “cd”

```
str <-c("123abd", "ab567cd", "abc5.00")
str_subset(string = str,  pattern = "cd$")
```

```
## [1] "ab567cd"
```

Character ranges give you convenient shortcut based on the dash metacharacter "-" to indicate a range of characters. A character range consists of a character set with two characters separated by a dash or minus "-" sign.

For example: you want to match a string that has numbers from 1-4

```
str <-c("123abd", "ab567cd", "abc5.00")
str_subset(string = str,  pattern = "[1-4]")
```

```
## [1] "123abd"
```

- The following regex will match all the strings containing lower letters from f to z
- As there are no such strings, the result is empty vector

```
str <-c("123abd", "ab567cd", "abc5.00")
str_subset(string = str,  pattern = "[f-z]")

## character(0)
```

Character classes

To match one of several characters in a specified set we can enclose the characters of concern with square brackets [ ]. In addition, to match any characters **not** in a specified character set we can include the caret ^ at the beginning of the set within the brackets. The following displays the general syntax for common character classes but these can be altered easily as shown in the examples that follow:

Closely related with character sets and character ranges, regular expressions provide another useful construct called **character classes** which, as their name indicates, are used to match a certain class of characters. The most common character classes in most regex engines are:

Character	Matches	Same as
`\\d`	any digit	`[0-9]`
`\\D`	any nondigit	`[^0-9]`
`\\w`	any character considered part of a word	`[a-zA-Z0-9_]`
`\\W`	any character not considered part of a word	`[^a-zA-Z0-9_]`
`\\s`	any whitespace character	`[\f\n\r\t\v]`
`\\S`	any nonwhitespace character	`[^\f\n\r\t\v]`

POSIX character classes

Class	Description	Same as
`[:alnum:]`	any letter or digit	`[a-zA-Z0-9]`
`[:alpha:]`	any letter	`[a-zA-Z]`
`[:digit:]`	any digit	`[0-9]`
`[:lower:]`	any lower case letter	`[a-z]`
`[:upper:]`	any upper case letter	`[A-Z]`
`[:space:]`	any whitespace inluding space	`[\f\n\r\t\v ]`
`[:punct:]`	any punctuation symbol	
`[:print:]`	any printable character	
`[:graph:]`	any printable character excluding space	
`[:xdigit:]`	any hexadecimal digit	`[a-fA-F0-9]`
`[:cntrl:]`	ASCII control characters	

Regular expressions

- The Awards column contains text information about awards won and nominations
- Look at the first 20 records. The text is standard for all the movies
- Can you create a column that will show how many wins and nominations each movies has ?

```
movies$Awards[1:20]
```

```
##  [1] "4 wins & 8 nominations."
##  [2] "7 wins & 17 nominations."
##  [3] "1 win & 2 nominations."
##  [4] "9 wins & 28 nominations."
##  [5] "2 wins & 67 nominations."
##  [6] "1 nomination."
##  [7] "Won 1 Oscar. Another 87 wins & 171 nominations."
##  [8] "2 wins & 3 nominations."
##  [9] "2 wins & 4 nominations."
## [10] "Nominated for 1 Golden Globe. Another 5 wins & 7 nominations."
## [11] "1 win & 1 nomination."
## [12] "1 win & 3 nominations."
## [13] "Nominated for 1 Oscar. Another 1 win & 3 nominations."
## [14] "1 win & 2 nominations."
## [15] "Nominated for 2 Oscars. Another 11 wins & 5 nominations."
## [16] "Nominated for 1 Golden Globe. Another 3 wins & 31 nominations."
## [17] "2 wins & 2 nominations."
## [18] "2 wins & 2 nominations."
## [19] "1 win & 3 nominations."
## [20] "1 win & 5 nominations."
```

Regular expressions

- You can use str_replace_all, with character class [a-zA-Z] to delete all alphabetical letters from the string first. You replace the match with nothing
- We are left with numbers and punctuation marks
- Youy can use another regex to get rid off ounctuations

```
movies$awards_num <- str_replace_all(movies$Awards, pattern="[a-zA-Z]",
                                     replacement = "")
movies$awards_num[1:20]
```

```
##  [1] "4  & 8 ."          "7  & 17 ."         "1  & 2 ."
##  [4] "9  & 28 ."         "2  & 67 ."         "1 ."
##  [7] " 1 .  87  & 171 ." "2  & 3 ."          "2  & 4 ."
## [10] "  1  .  5  & 7 ."  "1  & 1 ."          "1  & 3 ."
## [13] "  1 .  1  & 3 ."   "1  & 2 ."          "  2 .  11  & 5 ."
## [16] "  1  .  3  & 31 ." "2  & 2 ."          "2  & 2 ."
## [19] "1  & 3 ."          "1  & 5 ."
```

Regular expressions

- Or you can use pattern [^0-9] which says match everything that is not number (^ is the negation)
- Delete everything that is not a number

```
movies$awards_num <- str_replace_all(movies$Awards, pattern="[^0-9]",
                                     replacement = " ")
movies$awards_num[1:20]
```

```
##  [1] "4         8              "
##  [2] "7         17              "
##  [3] "1        2              "
##  [4] "9         28              "
##  [5] "2         67              "
##  [6] "1             "
##  [7] "    1                87         171              "
##  [8] "2         3              "
##  [9] "2         4              "
## [10] "                1                          5         7              "
## [11] "1        1              "
## [12] "1        3              "
## [13] "                1                1        3              "
## [14] "1        2              "
## [15] "                2                  11         5              "
## [16] "                1                          3         31              "
## [17] "2         2              "
## [18] "2         2              "
## [19] "1        3              "
## [20] "1        5              "
```

- We are left with lot of white space that need to be deleted
- The new vector need to have the numbers separated by a single space
- The regex we use here is \\s+, where \\s is the metacharacter for space and + matches at least 1 times.
- Next we need to trim white space

```
movies$awards_num <- str_replace_all(movies$awards_num, pattern="\\s+",
                                     replacement = " ")
movies$awards_num[1:20]
```

```
##  [1] "4 8 "       "7 17 "      "1 2 "       "9 28 "      "2 67 "
##  [6] "1 "         " 1 87 171 " "2 3 "       "2 4 "       " 1 5 7 "
## [11] "1 1 "       "1 3 "       " 1 1 3 "    "1 2 "       " 2 11 5 "
## [16] " 1 3 31 "   "2 2 "       "2 2 "       "1 3 "       "1 5 "
```

Regular expressions

- Trim white spaces
- After we will split the string using space as a pattern

```
movies$awards_num <- trimws(movies$awards_num)
movies$awards_num[1:20]
```

```
##  [1] "4 8"       "7 17"      "1 2"       "9 28"      "2 67"      "1"
##  [7] "1 87 171" "2 3"       "2 4"       "1 5 7"     "1 1"       "1 3"
## [13] "1 1 3"     "1 2"       "2 11 5"    "1 3 31"    "2 2"       "2 2"
## [19] "1 3"       "1 5"
```

Regular expressions

- Result is a matrix but still character
- Use apply to make it numeric

```
x1 <- str_split(movies$awards_num, pattern = " ",simplify = T)
head(x1)
```

```
##      [,1] [,2] [,3]
## [1,] "4"  "8"  ""
## [2,] "7"  "17" ""
## [3,] "1"  "2"  ""
## [4,] "9"  "28" ""
## [5,] "2"  "67" ""
## [6,] "1"  ""   ""
```

- We got numeric matrix using apply function
- Next we can use rowSums() to sum all wins and nominations

```
x1 <- apply(x1, 2, as.numeric)
head(x1)
```

```
##      [,1] [,2] [,3]
## [1,]    4    8   NA
## [2,]    7   17   NA
## [3,]    1    2   NA
## [4,]    9   28   NA
## [5,]    2   67   NA
## [6,]    1   NA   NA
```

- As our data contains NA, use na.rm = T, which stands for remove.na
- Then assign x1 to the column awards_num in the dataframe movies

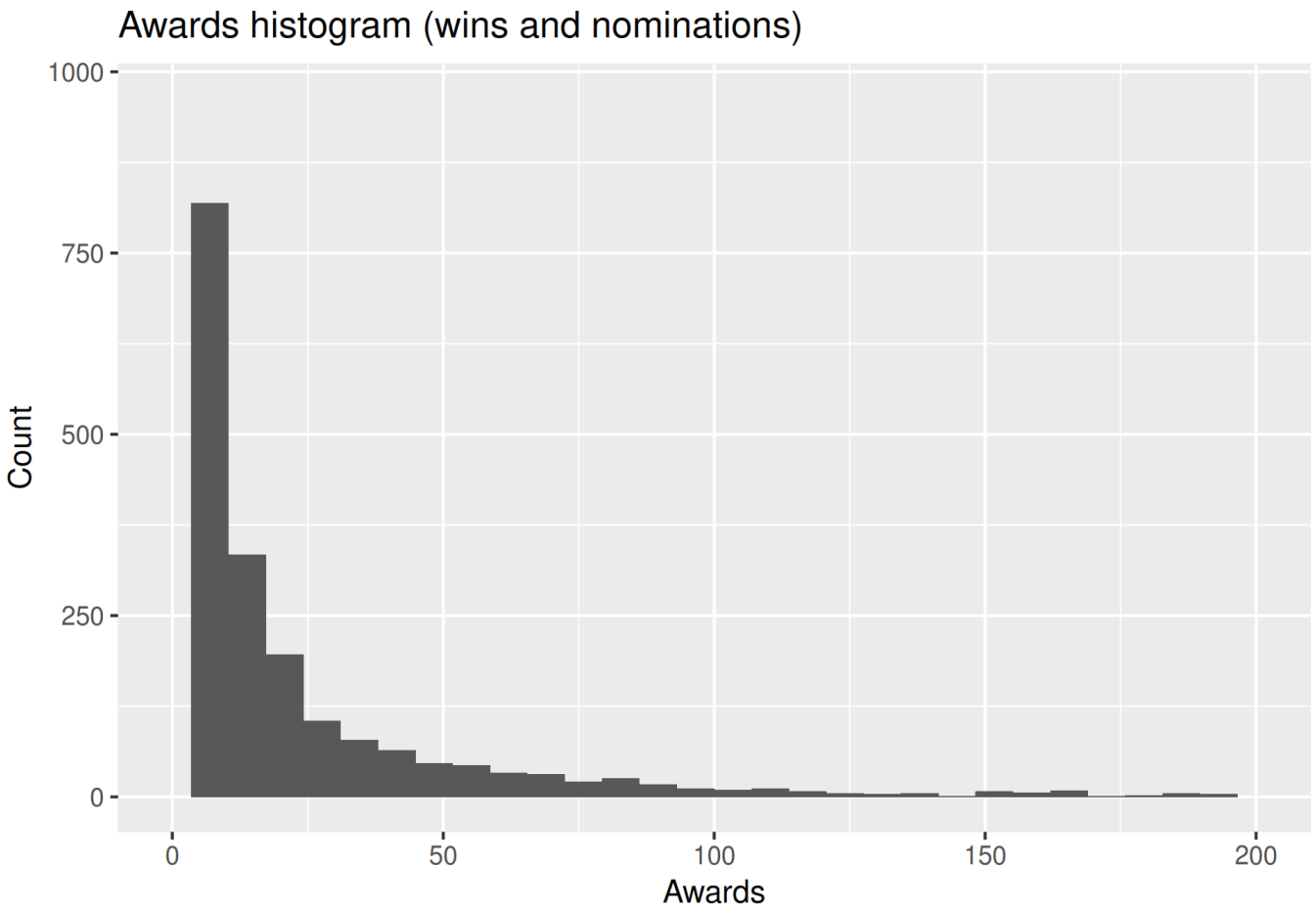
```
x1 <- rowSums(x1, na.rm=T)
x1[1:20]
```

```
##  [1]  12  24   3  37  69   1 259   5   6  13   2   4   5   3  18  35   4
## [18]   4   4   6
```

```
movies$awards_num <- x1
```

Create the histogram of awards

```
ggplot(movies, aes(x=awards_num))+geom_histogram()+
  xlim(c(0,200))+labs(title="Awards histogram (wins and nominations)",
                      x="Awards", y="Count")
```



Regular expressions

- Now lets try to count wins only
- Count all wins except for golden globe and Oscar

Possible strings

2 wins & 3 nominations.

1 win & 1 nomination.

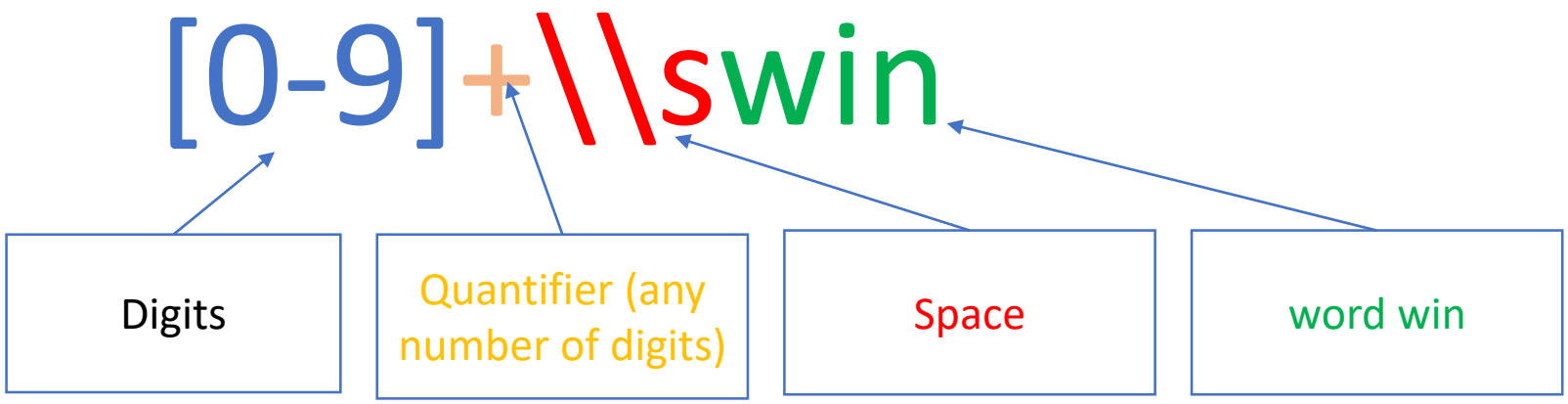
Nominated for 1 Oscar. Another 1 win & 3 nominations.

Nominated for 2 Oscars. Another 11 wins & 5 nominations.

Snapshot of the awards variable

```
##  [1] "4 wins & 8 nominations."
##  [2] "7 wins & 17 nominations."
##  [3] "1 win & 2 nominations."
##  [4] "9 wins & 28 nominations."
##  [5] "2 wins & 67 nominations."
##  [6] "1 nomination."
##  [7] "Won 1 Oscar. Another 87 wins & 171 nominations."
##  [8] "2 wins & 3 nominations."
##  [9] "2 wins & 4 nominations."
## [10] "Nominated for 1 Golden Globe. Another 5 wins & 7 nominations."
## [11] "1 win & 1 nomination."
## [12] "1 win & 3 nominations."
## [13] "Nominated for 1 Oscar. Another 1 win & 3 nominations."
## [14] "1 win & 2 nominations."
## [15] "Nominated for 2 Oscars. Another 11 wins & 5 nominations."
## [16] "Nominated for 1 Golden Globe. Another 3 wins & 31 nominations."
## [17] "2 wins & 2 nominations."
## [18] "2 wins & 2 nominations."
## [19] "1 win & 3 nominations."
```

Lets look at the regex pattern



+ Quantifier: The preceding item will be matched 1 or more times

```
m1 <- str_extract_all(movies$Awards, pattern="[0-9]*\\swin", simplify=T)
m1[1:20]
```

```
##  [1] "4 win"  "7 win"  "1 win"  "9 win"  "2 win"  ""       "87 win"
##  [8] "2 win"  "2 win"  "5 win"  "1 win"  "1 win"  "1 win"  "1 win"
## [15] "11 win" "3 win"  "2 win"  "2 win"  "1 win"  "1 win"
```

Quantifiers

When we want to match a **certain number** of characters that meet a certain criteria we can apply quantifiers to our pattern searches. The quantifiers we can use are:

Quantifier	Description
?	the preceding item is optional and will be matched at most once
*	the preceding item will be matched zero or more times
+	the preceding item will be matched one or more times
{n}	the preceding item is matched exactly n times
{n,}	the preceding item is matched n or more times
{n,m}	the preceding item is matched at least n times, but not more than m times

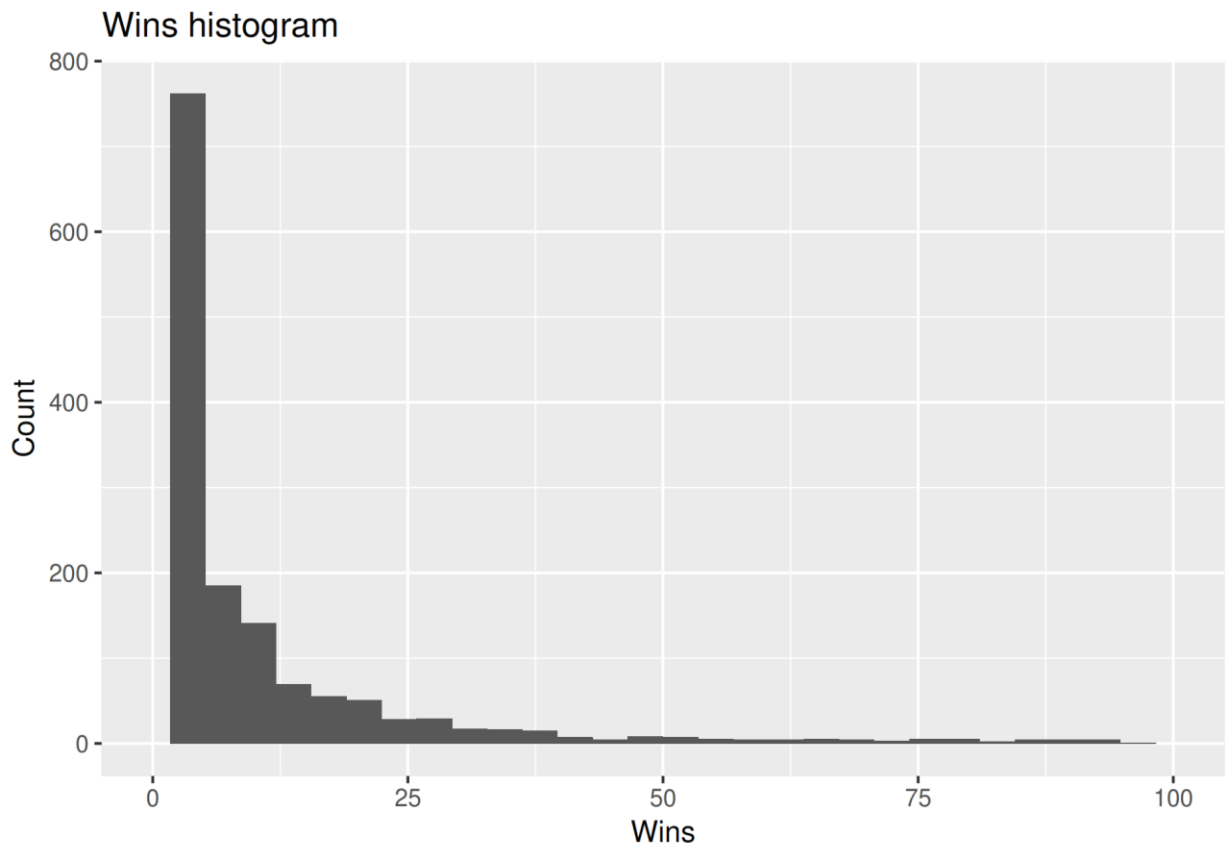
*adapted from *Handling and Processing Strings in R* (Sanchez, 2013)

str_remove_all() is a wrapper for str_replace_all() where replacement=“”

```
m1 <- str_remove_all(m1, pattern="[a-zA-Z]")
# No need to trim the white spaces
m1 <- as.numeric(m1)
```

Add the variable to the movies data frame and create the histogram

```
movies$Wins <- m1
ggplot(movies, aes(x=Wins))+geom_histogram()+
labs(title="Wins histogram", x="Wins", y="Count")+
  xlim(c(0,100))
```



Can you extract the number of the Oscar wins ?

Use dplyr to answer to the question: which movies won the biggest number of Oscars. Get the dataframe with the name of the movie and number of Oscars – Use **select()** to select the variables, **arrange()** and head()

Try different plots to visualize the new variable

Regular expressions

```
osc <- str_extract_all(movies$Awards, pattern="Won\\s[0-9]\\sOscar", simplify=T)
movies$Oscar <- as.numeric(str_remove_all(osc, pattern="[^\\d]"))
```

```
movies %>%
  select(Oscar, title) %>%
  arrange(desc(Oscar)) %>%
  head()
```

```
##   Oscar                title
## 1     9 The English Patient
## 2     8   On the Waterfront
## 3     8         My Fair Lady
## 4     8  Gone with the Wind
## 5     8             Amadeus
## 6     7           The Sting
```

Regular expressions: extracting the phone numbers

- Extract the phone numbers in one column and the names in another:
- Combine them in the dataframe

```
phones <- c("Anna 077-789663", "Hagopik 99656565",
            "Serozh2 099-65-6569 MALYAR")
```

These are random names and random numbers, don't call them

Regular expressions: extracting the phone numbers

```
phones <- c("Anna 077-789663", "Hagopik 99656565",
            "Serozh2 099-65-6569 MALYAR")
```

Extracting numbers

```
numbers<-str_extract_all(phones,pattern = "\\s{1}[0-9,-]*\\s?", simplify = T)
numbers<-trimws(numbers)
```

Extracting names

```
names<-str_replace_all(phones,pattern="\\s{1}[0-9,-]*\\s?"," ")
names<-trimws(names)
```

Dataframe

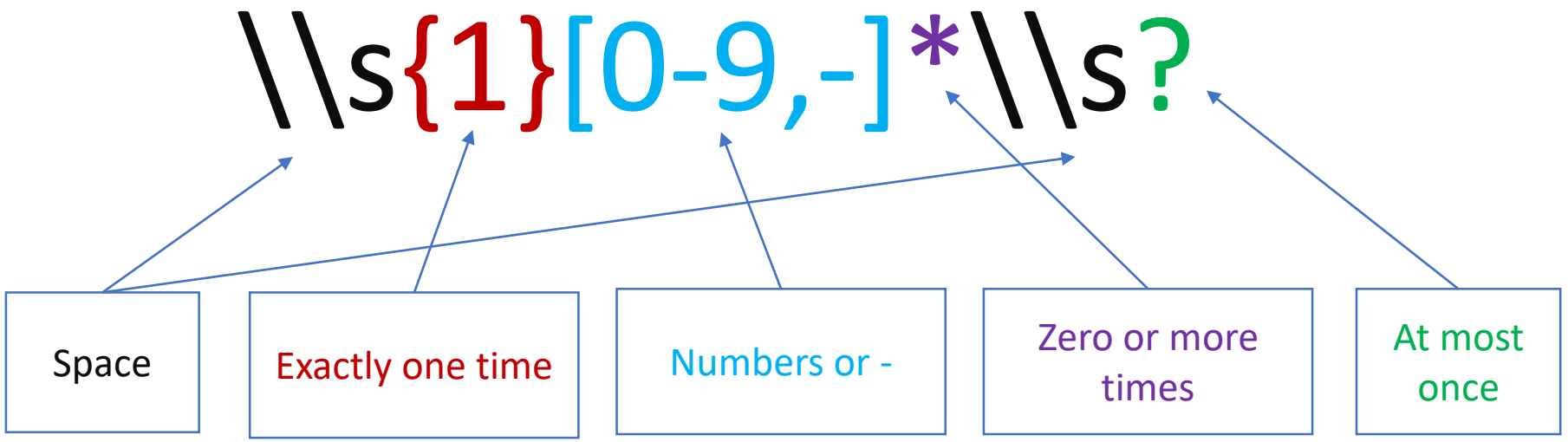
```
phone_book<-data.frame(names,numbers)
head(phone_book)
```

```
##            names     numbers
## 1           Anna  077-789663
## 2        Hagopik    99656565
## 3 Serozh2 MALYAR 099-65-6569
```

Regular expressions: extracting the phone numbers

```
numbers<-str_extract_all(phones,pattern = "\\s{1}[0-9,-]*\\s?", simplify = T)
numbers<-trimws(numbers)
```

Regex pattern



Text mining

- The main structure for managing documents in tm is a so-called Corpus, representing a collection of text documents. A corpus is an abstract concept, and there can exist several implementations in parallel.

- *Corpora* are collections of documents containing (natural language) text.

- First you need to create a VectorSource - **A vector source interprets each element of the vector x as a document.**

Create a corpus from the vector text

```
library(tm)
```

```
text <- c( "Dogs are the best pets.",
           "Are these dogs yours?",
           "My dogs are the best dogs."
)
vs <- VectorSource(text)
corpus <- VCorpus(vs)
corpus
```

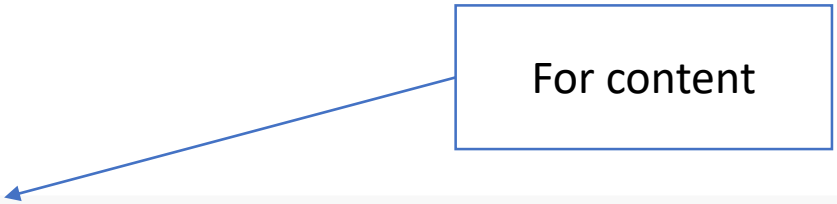
```
## <<VCorpus>>
## Metadata:  corpus specific: 0, document level (indexed): 0
## Content:  documents: 3
```

use [[to access single element in the corpus

```
inspect(corpus[[2]])
```

```
## <<PlainTextDocument>>
## Metadata:  7
## Content:  chars: 21
##
## Are these dogs yours?
```

Access content directly for the second document




```
corpus[[2]][1]
```

```
## $content
## [1] "Are these dogs yours?"
```

Access metadata

```
corpus[[2]][2]
```

```
## $meta
##   author       : character(0)
##   datetimestamp: 2018-07-02 12:37:16
##   description  : character(0)
##   heading      : character(0)
##   id           : 2
##   language     : en
##   origin       : character(0)
```

Metadata is used to annotate text documents or whole corpora with additional information. The easiest way to accomplish this with tm is to use the **meta()** function.

- Metadata is used to annotate text documents or whole corpora with additional information. The easiest way to accomplish this with tm is to use the **meta()** function.

- Corpora in tm have two types of metadata: one is the metadata on the corpus level (corpus), the other is the metadata related to the individual documents (indexed) in form of a data frame.

- The latter is often done for performance reasons (hence the named indexed for indexing) or because the metadata has an own entity but still relates directly to individual text documents.

- Look at the individual level document meta
- Specify the tag you want to get

```
meta(corpus, tag = 'language')
```

```
## $`1`
## [1] "en"
##
## $`2`
## [1] "en"
##
## $`3`
## [1] "en"
```

Create new metadata tag “class” for the document N1 and assign CSE252 to it

```
meta(corpus[[1]], tag="class") <- "CSE252"
corpus[[1]][2]
```

```
## $meta
##   author       : character(0)
##   datetimestamp: 2018-07-02 17:11:46
##   description  : character(0)
##   heading      : character(0)
##   id           : 1
##   language     : en
##   origin       : character(0)
##   class        : CSE252
```

A **document-term matrix** or **term-document matrix** is a mathematical matrix that describes the frequency of terms that occur in a collection of documents. In a document-term matrix, rows correspond to documents in the collection and columns correspond to terms.

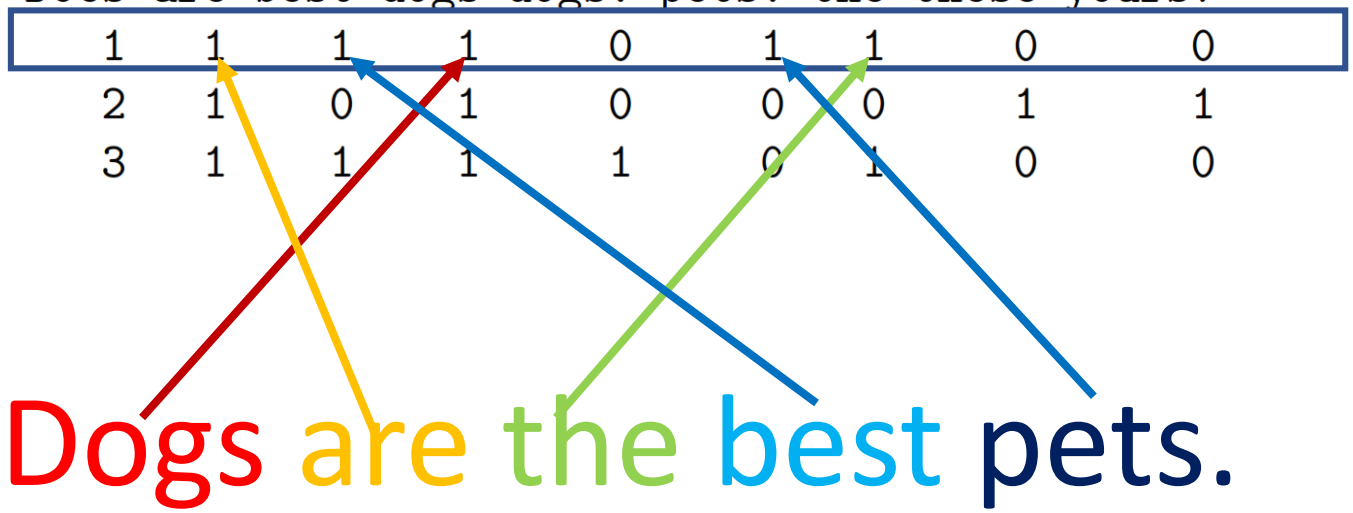
```
dtm <- DocumentTermMatrix(corpus)
inspect(dtm)
```

```
## <<DocumentTermMatrix (documents: 3, terms: 8)>>
## Non-/sparse entries: 14/10
## Sparsity           : 42%
## Maximal term length: 6
## Weighting          : term frequency (tf)
## Sample             :
##     Terms
## Docs are best dogs dogs. pets. the these yours?
##    1   1    1    1     0     1   1     0      0
##    2   1    0    1     0     0   0     1      1
##    3   1    1    1     1     0   1     0      0
```

"Dogs are the best pets."

```
dtm <- DocumentTermMatrix(corpus)
inspect(dtm)
```

```
## <<DocumentTermMatrix (documents: 3, terms: 8)>>
## Non-/sparse entries: 14/10
## Sparsity           : 42%
## Maximal term length: 6
## Weighting          : term frequency (tf)
## Sample             :
##     Terms
## Docs are best dogs dogs. pets. the these yours?
##    1   1    1    1     0     1   1     0      0
##    2   1    0    1     0     0   0     1      1
##    3   1    1    1     1     0   1     0      0
```



Term document matrix is the transposed document term matrix

```
tdm <- TermDocumentMatrix(corpus)
inspect(tdm)
```

```
## <<TermDocumentMatrix (terms: 8, documents: 3)>>
## Non-/sparse entries: 14/10
## Sparsity           : 42%
## Maximal term length: 6
## Weighting          : term frequency (tf)
## Sample             :
##         Docs
## Terms    1 2 3
##   are    1 1 1
##   best   1 0 1
##   dogs   1 1 1
##   dogs.  0 0 1
##   pets.  1 0 0
##   the    1 0 1
##   these  0 1 0
##   yours? 0 1 0
```

Sparcity is defined as a percentage of element with value 0 in total number of elements in document term matrix

```
dtm <- DocumentTermMatrix(corpus)
inspect(dtm)
```

```
## <<DocumentTermMatrix (documents: 3, terms: 8)>>
## Non-/sparse entries: 14/10
## Sparsity           : 42%
```

Calculate it by hand

```
mm <- as.matrix(dtm)
# Sparsity
sum(mm == 0)/(3*8)
```

```
## [1] 0.4166667
```

Text mining

```
dtm <- DocumentTermMatrix(corpus)
inspect(dtm)
```

```
## <<DocumentTermMatrix (documents: 3, terms: 8)>>
## Non-/sparse entries: 14/10
## Sparsity           : 42%
## Maximal term length: 6
## Weighting          : term frequency (tf)
## Sample             :
##     Terms
## Docs are best dogs dogs. pets. the these yours?
##    1   1    1    1     0     1   1     0      0
##    2   1    0    1     0     0   0     1      1
##    3   1    1    1     1     0   1     0      0
```

Problem !

Do some cleaning
One way would be to remove all punctuations before creating corpus, using regex

```
text <- c( "Dogs are the best pets.",
           "Are these dogs yours?",
           "My dogs are the best dogs."
)
library(stringr)
text <- str_remove_all(text, pattern = "[:punct:]")
text
```

```
## [1] "Dogs are the best pets"    "Are these dogs yours"
## [3] "My dogs are the best dogs"
```

regex for punctuations

Or alternatively use tm_map()

```
text <- c( "Dogs are the best pets.",
           "Are these dogs yours?",
           "My dogs are the best dogs.")
vs <- VectorSource(text)
corpus <- VCorpus(vs)
corpus <- tm_map(corpus, removePunctuation)
dtm <- DocumentTermMatrix(corpus)
inspect(dtm)
```

tm_map() takes corpus and applies on it a function, in this case removePunctuation.

Text mining

- In information retrieval, **tf–idf** or **TFIDF**, short for **term frequency–inverse document frequency**, is a numerical statistic that is intended to reflect how important a word is to a document tin a collection or corpus. It is often used as a weighting factor in searches of information retrieval, text mining, and user modeling.

- The tf-idf value increases proportionally to the number of times a word appears in the document and is offset by the frequency of the word in the corpus, which helps to adjust for the fact that some words appear more frequently in general.

Typically, the tf-idf weight is composed by two terms: the first computes the normalized Term Frequency (TF). the second term is the Inverse Document Frequency (IDF), computed as the logarithm of the number of the documents in the corpus divided by the number of documents where the specific term appears.

- TF: Term Frequency, which measures how frequently a term occurs in a document. Since every document is different in length, it is possible that a term would appear much more times in long documents than shorter ones. Thus, the term frequency is often divided by the document length (the total number of terms in the document) as a way of normalization:

$$TF(t) = \frac{Number\ of\ times\ term\ t\ appears\ in\ a\ document}{Total\ number\ of\ terms\ in\ the\ document}$$

- IDF: Inverse Document Frequency, which measures how important a term is. While computing TF, 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, by computing the following:

$$IDF(t) = \log_2(\frac{Total\ number\ of\ documents}{Number\ of\ documents\ with\ term\ t\ in\ it})$$

$$\boldsymbol{tf - idf = TF(t) * IDF(t)}$$

Text mining

```
dtm1 <- DocumentTermMatrix(corpus, control = list(weighting = weightTfIdf))
inspect(dtm1)
```

```
## <<DocumentTermMatrix (documents: 3, terms: 7)>>
## Non-/sparse entries: 7/14
## Sparsity           : 67%
## Maximal term length: 5
## Weighting          : term frequency - inverse document frequency (normalized) (tf-idf)
## Sample             :
##     Terms
## Docs are      best dogs      pets       the     these     yours
##    1   0 0.1169925    0 0.3169925 0.1169925 0.0000000 0.0000000
##    2   0 0.0000000    0 0.0000000 0.0000000 0.3962406 0.3962406
##    3   0 0.1169925    0 0.0000000 0.1169925 0.0000000 0.0000000
```

Pets, Doc1

TF = 1/5
IDF = $\log_2 3/1$
tf-idf

```
(1/5)*log(3/1, base=2)
```

```
## [1] 0.3169925
```

Dogs

TF = 1/5
Idf = $\log_2 3/3$

```
1/5 * log(3/3, base=2)
```

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

lyrics.csv contains information on about 300,000 songs and their lyrics

```
lyrics <- read.csv("lyrics.csv", stringsAsFactors = F)
summary(lyrics)
```

```
##      index            song                year         artist
##  Min.   :     0   Length:339277      Min.   :  67   Length:339277
##  1st Qu.: 84819   Class :character   1st Qu.:2006   Class :character
##  Median :169638   Mode  :character   Median :2008   Mode  :character
##  Mean   :169638                      Mean   :2009
##  3rd Qu.:254457                      3rd Qu.:2014
##  Max.   :339276                      Max.   :2038
##     genre              lyrics
##  Length:339277      Length:339277
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
```

There is a problem with the variable year

```
summary(lyrics$year)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##      67    2006    2008    2009    2014    2038
```

```
lyrics %>%
  group_by(year) %>%
  summarise(count = n()) %>%
  arrange(desc(year))
```

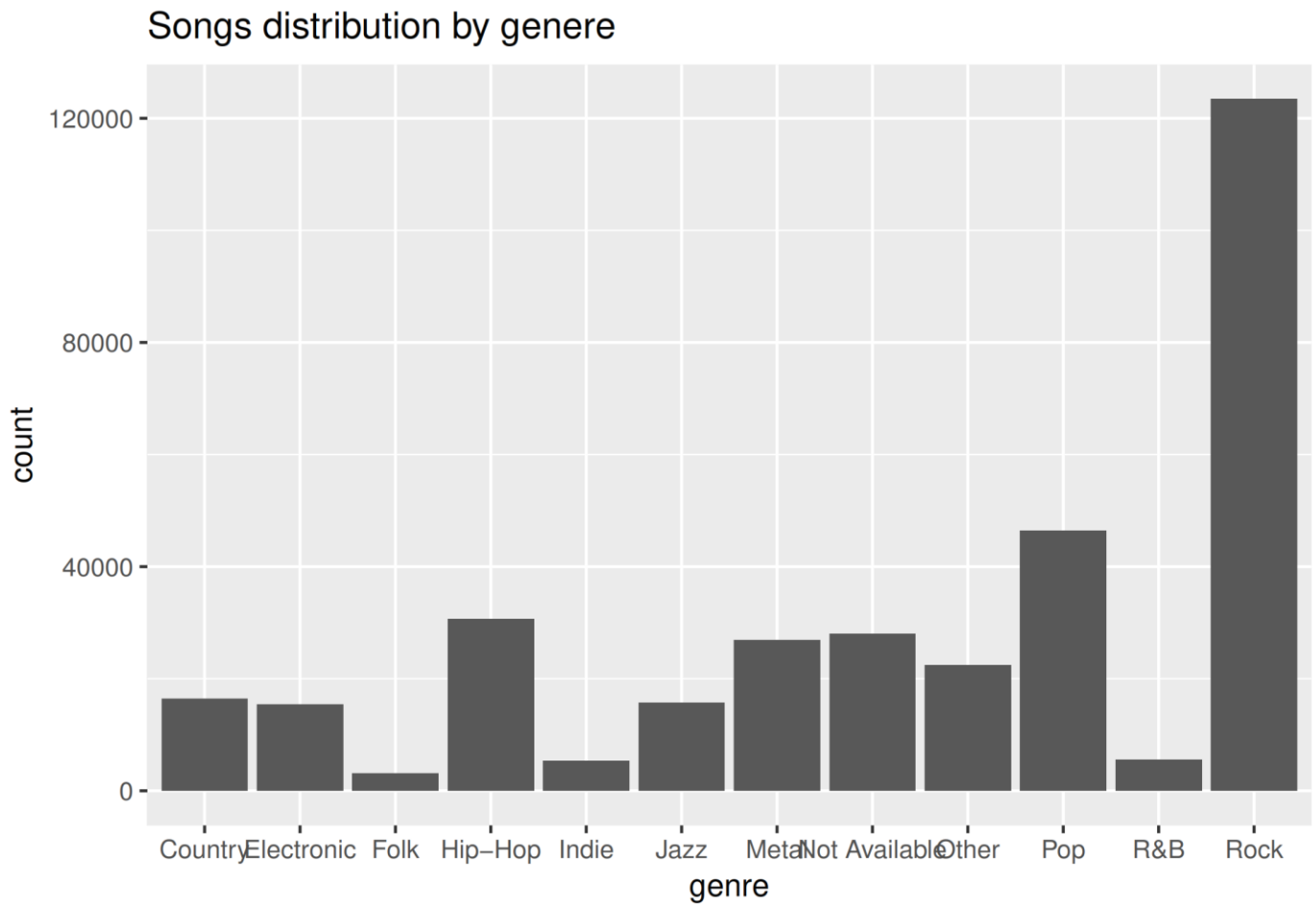
```
## # A tibble: 52 x 2
##     year count
##    <int> <int>
##  1  2038    10
##  2  2016 35042
##  3  2015 27159
##  4  2014 26393
##  5  2013 16245
##  6  2012 14581
##  7  2011 11387
##  8  2010 11374
##  9  2009 11333
## 10  2008 20375
## # ... with 42 more rows
```

```
lyrics %>%
  group_by(year) %>%
  summarise(count = n()) %>%
  arrange(year)
```

```
## # A tibble: 52 x 2
##     year count
##    <int> <int>
##  1    67     1
##  2   112     4
##  3   702     1
##  4  1968     1
##  5  1970   421
##  6  1971   480
##  7  1972   496
##  8  1973   504
##  9  1974   395
## 10  1975   321
## # ... with 42 more rows
```

Looking at the genre

```
ggplot(data=lyrics, aes(x=genre))+geom_bar()+
  ggtitle("Songs distribution by genere")
```



Quick look at the most frequent artists

```
lyrics %>%
  group_by(artist) %>%
  summarise(n_songs=n()) %>%
  arrange(desc(n_songs))
```

```
## # A tibble: 17,088 x 2
##    artist            n_songs
##    <chr>               <int>
##  1 dolly-parton          755
##  2 american-idol         700
##  3 elton-john            680
##  4 b-b-king              667
##  5 chris-brown           655
##  6 eddy-arnold           628
##  7 barbra-streisand      624
##  8 ella-fitzgerald       623
##  9 bob-dylan             614
## 10 bee-gees              599
## # ... with 17,078 more rows
```

Lyrics: spotting missing values

```
lyrics$char_num <- nchar(lyrics$lyrics)
summary(lyrics$char_num)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     0.0     0.0   744.0   844.1  1212.0 42155.0
```

Cleaning the data

```
lyrics <- lyrics %>%
  filter(char_num>100 & genre != "Not Available" & year > 1968 & year <= 2016)
dim(lyrics)
```

```
## [1] 221669      7
```

Cleaning the data

- It is possible that your text will have non standard/non Unicode entries.
- It is not necessarily a problem and there are several ways to deal with it:
  - easiest one: delete them

Function iconv() will delete all non ASCII characters

```
b <- "[Intro: BeyoncÃfÂ©]"
iconv(b, to="ASCII", sub="")
```

```
## [1] "[Intro: Beyonc]"
```

```
lyrics$lyrics <- iconv(lyrics$lyrics, to="ASCII", sub="")
```

Lets analyze lyrics of Beyonce first

```
beyonce <- lyrics[lyrics$artist =='beyonce-knowles',]
beyonce_vs <- VectorSource(beyonce$lyrics)
beyonce_corpus <- VCorpus(beyonce_vs)
```

Often there are words that are frequent but provide little information. So you may want to remove these so-called stop words. Some common English stop words include "I", "she'll", "the", etc. In the tm package, there are 174 stop words on this common list.

```
stopwords("english")
```

```
##   [1] "i"          "me"         "my"         "myself"     "we"
##   [6] "our"        "ours"       "ourselves"  "you"        "your"
##  [11] "yours"      "yourself"   "yourselves" "he"         "him"
##  [16] "his"        "himself"    "she"        "her"        "hers"
##  [21] "herself"    "it"         "its"        "itself"     "they"
##  [26] "them"       "their"      "theirs"     "themselves" "what"
##  [31] "which"      "who"        "whom"       "this"       "that"
##  [36] "these"      "those"      "am"         "is"         "are"
##  [41] "was"        "were"       "be"         "been"       "being"
##  [46] "have"       "has"        "had"        "having"     "do"
##  [51] "does"       "did"        "doing"      "would"      "should"
##  [56] "could"      "ought"      "i'm"        "you're"     "he's"
##  [61] "she's"      "it's"       "we're"      "they're"    "i've"
##  [66] "you've"     "we've"      "they've"    "i'd"        "you'd"
##  [71] "he'd"       "she'd"      "we'd"       "they'd"     "i'll"
##  [76] "you'll"     "he'll"      "she'll"     "we'll"      "they'll"
```

A stemming algorithm is a process of linguistic normalisation, in which the variant forms of a word are reduced to a common form, for example,

connection
connections
connective ---> connect
connected
connecting

```
love <- c("love", "loving", "lovingly", "loved", "lover", "lovely", "love",
          "game", "gaming", "gamification")
stemDocument(love)
```

```
##  [1] "love"  "love"  "love"  "love"  "lover" "love"  "love"  "game"
##  [9] "game"  "gamif"
```

- create TermDocumentMatrix for Beyonce's songs.
- The text transformation/cleaning parameters are given in control()
  - remove all numbers
  - remove all punctuations
  - remove stopwords
  - stem the documents

```
beyonce_dtm <- TermDocumentMatrix(beyonce_corpus,
                          control=list(removeNumbers=T,removePunctuation=T,
                                       stopwords=T, stemming=T))
```

```
beyonce_dtm
```

```
## <<TermDocumentMatrix (terms: 3833, documents: 245)>>
## Non-/sparse entries: 20049/919036
## Sparsity           : 98%
## Maximal term length: 22
## Weighting          : term frequency (tf)
```

What are the most frequent words used by Beyonce?

```
dtm_mat <- as.matrix(beyonce_dtm)
freqs <- rowSums(dtm_mat)
df_freq <- data.frame(terms=rownames(dtm_mat),
                      freq = freqs, stringsAsFactors = F)
```

- Sort the dataframe in decreasing order

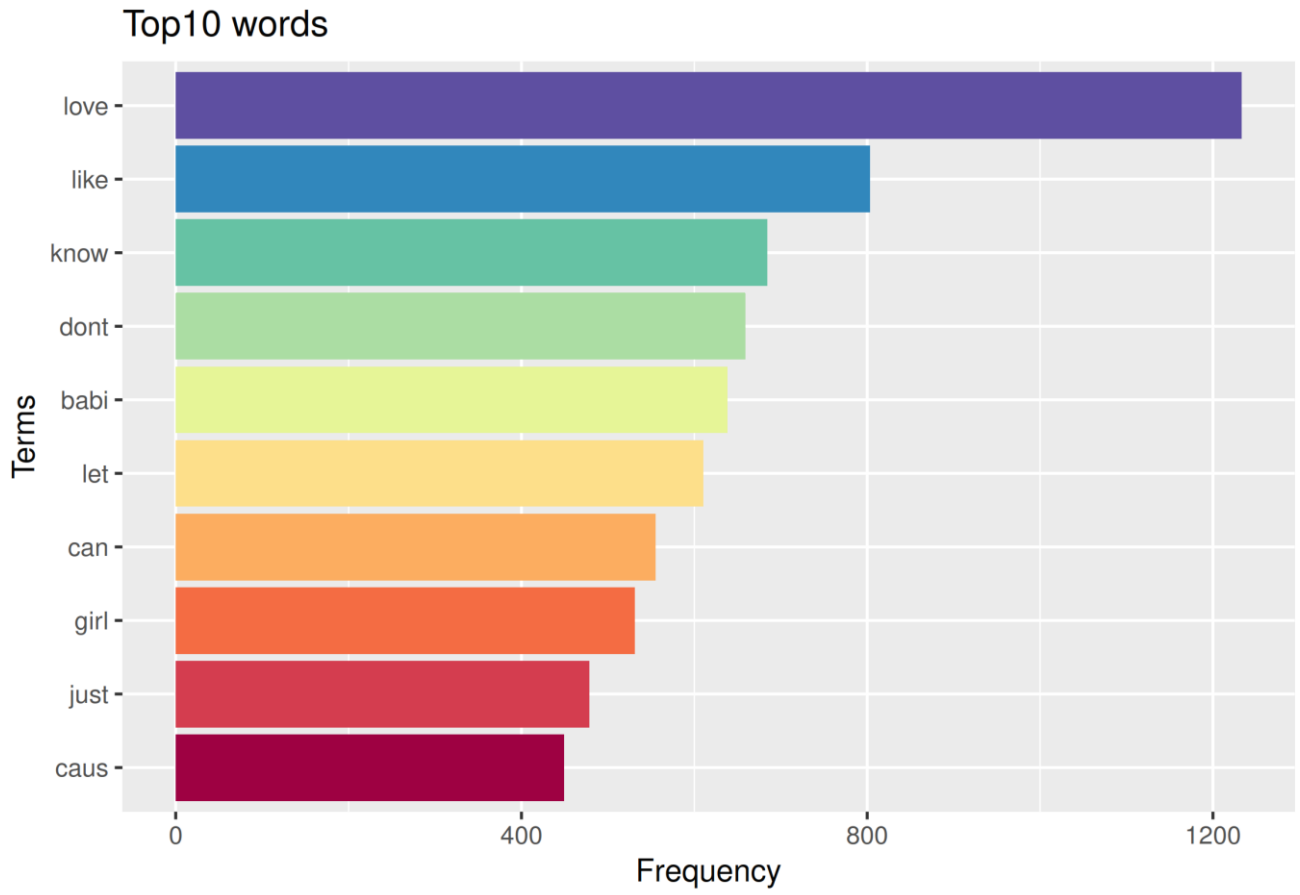
```
df_freq <- df_freq[order(df_freq$freq, decreasing = T),]
head(df_freq)
```

```
##       terms freq
## love   love 1233
## like   like  803
## know   know  684
## dont   dont  659
## babi   babi  638
## let     let  610
```

Text mining

- Visaulize the most frequent terms.
- we use reorder() on x axis variable, to get the most frequent terms first

```
ggplot(df_top10, aes(x=reorder(terms, freq), y=freq))+
  geom_bar(stat='identity', fill=brewer.pal(n=10,name='Spectral'))+
  coord_flip()+labs(x='Terms', y='Frequency', title='Top10 words')
```



Wordclouds are used to visualize the frequency of the words used in the corpus

```
set.seed(1)
wordcloud(words = df_freq$terms, freq = df_freq$freq, min.freq = 10,
          max.words=200, random.order=FALSE,
          colors=brewer.pal(8, "Dark2"))
```

Arguments:
words - the terms
freq – frequencies
min.freq – the minimum frequency the word need to have to appear on wordcloud
max.word – maximum number of words to show
random.order = F, if set to false the words will be plotted in decreasing frequency.

Text mining

```r
set.seed(1)
wordcloud(words = df_freq$terms, freq = df_freq$freq, min.freq = 10,
          max.words=200, random.order=FALSE,
          colors=brewer.pal(8, "Dark2"))
```

Its all about love babi girl

Lets try weighted tdm by tf-idf

```
beyonce_tdm <- TermDocumentMatrix(beyonce_corpus,
               control=list(removeNumbers=T, removePunctuation=T,
               stopwords=T, stemming=T, weighting = weightTfIdf))
```

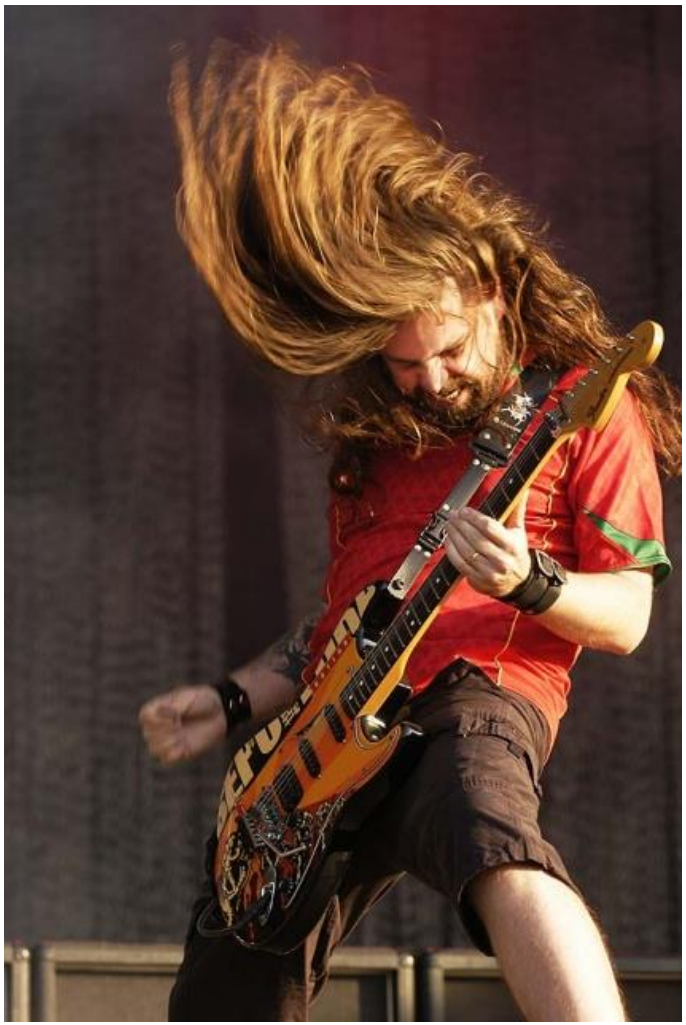
Construct the frequency dataframe

```
tdm_mat <- as.matrix(beyonce_tdm)
freqs <- rowSums(tdm_mat)
df_freq <- data.frame(terms=rownames(tdm_mat),
                      freq = freqs)
df_freq <- df_freq[order(df_freq$freq, decreasing = T),]
head(df_freq)
```

```
##       terms     freq
## halo   halo 5.295742
## love   love 4.083825
## girl   girl 3.326538
## run     run 3.293408
## let     let 3.269829
## babi   babi 2.828966
```

Text mining

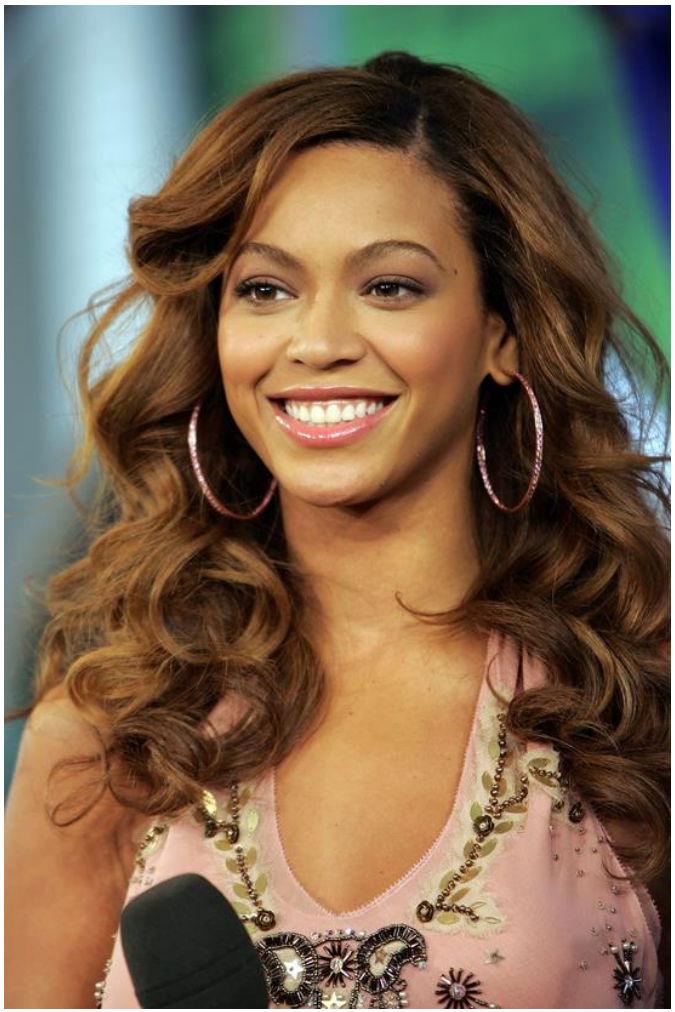
```
set.seed(1)
wordcloud(words = df_freq$terms, freq = df_freq$freq, min.freq = 1,
          max.words = 90,
          random.order=FALSE,
          colors=brewer.pal(8, "Dark2"))
```



Metal

VS

Pop



To make everything quicker, take the songs after 2012

```
metal <- lyrics[lyrics$genre=='Metal' & lyrics$year >2012,]
pop <- lyrics[lyrics$genre=="Pop"& lyrics$year >2012,]

metal_vc <- VCorpus(VectorSource(metal$lyrics))
pop_vc <- VCorpus(VectorSource(pop$lyrics))
```

```
metal_tdm <- TermDocumentMatrix(metal_vc,
                          control=list(removeNumbers=T, removePunctuation=T,
              stopwords=T, stemming=T))

pop_tdm <- TermDocumentMatrix(pop_vc,
                          control=list(removeNumbers=T, removePunctuation=T,
              stopwords=T, stemming=T))
```

Text mining

```
inspect(metal_tdm)
```

```
## <<TermDocumentMatrix (terms: 14563, documents: 1959)>>
## Non-/sparse entries: 129952/28398965
## Sparsity           : 100%
## Maximal term length: 67
## Weighting          : term frequency (tf)
## Sample             :
##        Docs
## Terms   1039 1254 1343 1363 1659 1660 1773 856 860 862
##   dont     4    1    0    1    5   12    2   9   3   1
##   just     6    2    2   44   11    6    0   0   3   0
##   know     2    0    2    2    8    8    2   3   2   7
##   like     1    7    0    8    4    4    3   5   2   4
##   never    3    0    1    1    0    2    6  12   1   0
##   now      2    0    2    1    2   13    1   2  13   4
##   one      2    1    0    2    7    1    2   1   5   3
##   see      6    0    0    0    5    3    3   0   2   2
##   time     3    0    1    0    1    1    0   4   0   5
##   will    11    0    1    0    0    0    0   1   0   0
```

```
inspect(pop_tdm)
```

```
## <<TermDocumentMatrix (terms: 57340, documents: 7938)>>
## Non-/sparse entries: 512486/454652434
## Sparsity           : 100%
## Maximal term length: 32
## Weighting          : term frequency (tf)
## Sample             :
##       Docs
## Terms  1820 1880 2057 4370 5570 5897 6002 606 6671 7642
##   can     0    0    0    4    0   17    0   2    0    1
##   dont   12    5    1    0    0   10   11   3   11    0
##   get    11    1    0   25    0    3   12   0    6    5
##   just    5   18    0   10    0    2   26   6    3    0
##   know    0    0    0    0    0    1    5   6    5    0
##   let     0   21    0   40    0   15   21   0    0    4
##   like    7    0    5    6    0   13    1   7    7    6
##   love    6    0    0    3    0   22    5   9    2    7
##   now     6    9    0    4    0   16    6   3    3    3
##   your    8    2    1    1    0    4    2  18    0    8
```

- In the function **removeSparseTerms()**, the argument sparse = x tells R to "remove all terms whose sparsity is greater than the threshold (x)".
  removeSparseTerms(my_dtm, sparse = 0.95) means remove all terms in the corpus whose sparsity is greater than 95%.

- Lets say, a term that appears just 5 times in a corpus of size 1000, will have a frequency of appearance of 0.005 =5/1000.

- This term's sparsity will be (1000-5)/1000 = 1- 0.005 = 0.995 = 99.5%.

- Therefore if sparsity threshold is set to sparse = 0.90, this term will be removed as its sparsity (0.995) is greater than the threshold(0.90).

- However, if sparsity threshold is set to sparse = 0.999, this term will not be removed as its sparsity (0.995) is lower than the threshold (0.999)

```
pop_tdm <- removeSparseTerms(pop_tdm, 0.99)
inspect(pop_tdm)
```

```
## <<TermDocumentMatrix (terms: 903, documents: 7938)>>
## Non-/sparse entries: 312040/6855974
## Sparsity           : 96%
## Maximal term length: 10
## Weighting          : term frequency (tf)
## Sample             :
##       Docs
## Terms  1811 1822 1827 1880  2 5891 5897 6002 6671 7113
##   can     0    0    0    0  1    1   17    0    0    3
##   dont   22   22   22    5  7    7   10   11   11    2
##   get     0    0    0    1  7    7    3   12    6    0
##   just    4    4    4   18  9    9    2   26    3    2
##   know    0    0    0    0 14   11    1    5    5    1
##   let    24   24   24   21  5    5   15   21    0    8
##   like    0    0    0    0  4    4   13    1    7    2
##   love   13   13   13    0  0    0   22    5    2    1
##   now    12   12   12    9  3    3   16    6    3    2
##   your   18   18   19    2 26   27    4    2    0    6
```

Make the frequency dataframe

```
pop_m <- as.matrix(pop_tdm)
pop_freq <- data.frame(terms = rownames(pop_m),
                       freq = rowSums(pop_m))
```

Text mining

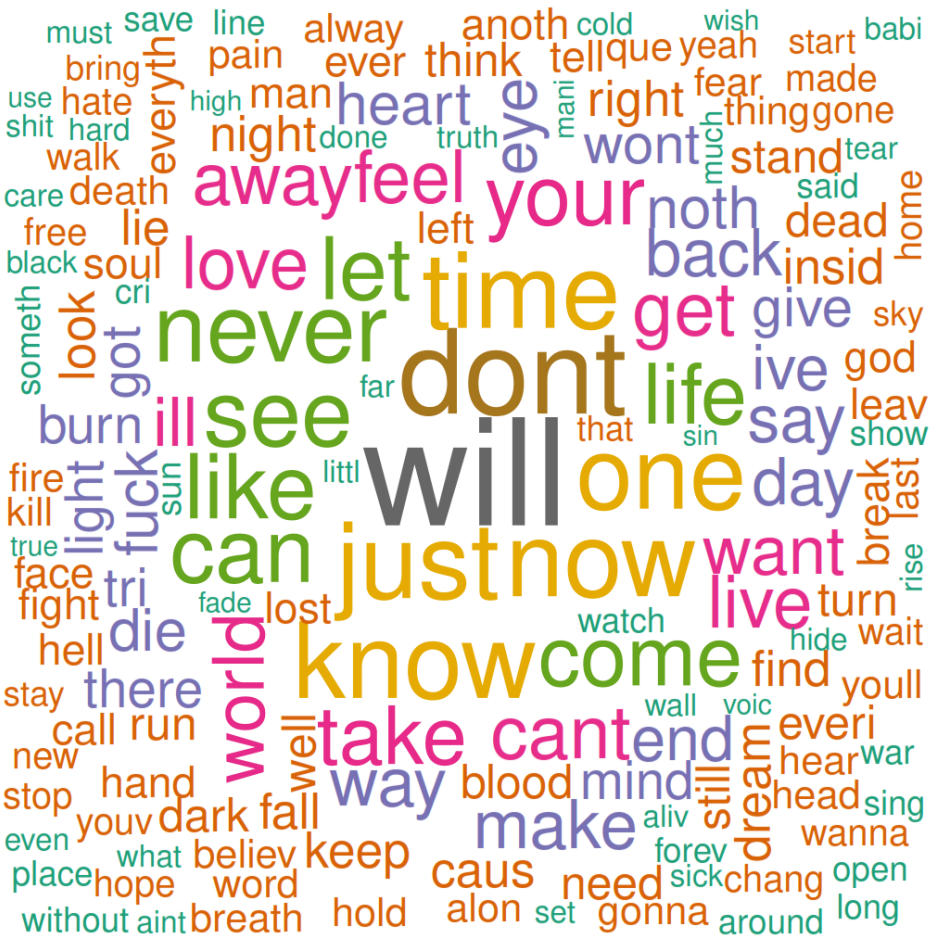
```
wordcloud(words = pop_freq$terms, freq = pop_freq$freq, min.freq = 10,
          max.words=200, random.order=FALSE,
          colors=brewer.pal(8, "Dark2"))
```

Metal music word frequencies

```
metal_tdm <- removeSparseTerms(metal_tdm, 0.99)
metal_m <- as.matrix(metal_tdm)
metal_freq <- data.frame(terms = rownames(metal_m),
                         freq = rowSums(metal_m))
```

Text mining

```
set.seed(1)
wordcloud(words = metal_freq$terms, freq = metal_freq$freq, min.freq = 10,
          max.words=200, random.order=FALSE,
          colors=brewer.pal(8, "Dark2"))
```



Another way to compare lyrics from two geners

- Combine all songs together for each genere into one document (use paste)
- Then create a corpus with two documents inside-one for pop one for metal

```
metal1 <- paste(metal$lyrics, collapse="")
pop1 <- paste(pop$lyrics, collapse="")
metal_pop <- c(metal1, pop1)
metal_pop_vc<-VCorpus(VectorSource(metal_pop))
metal_pop_tdm <- TermDocumentMatrix(metal_pop_vc,
  control=list(removeNumbers=T, removePunctuation=T,
               stopwords=T, stemming=T))
inspect(metal_pop_tdm)
```

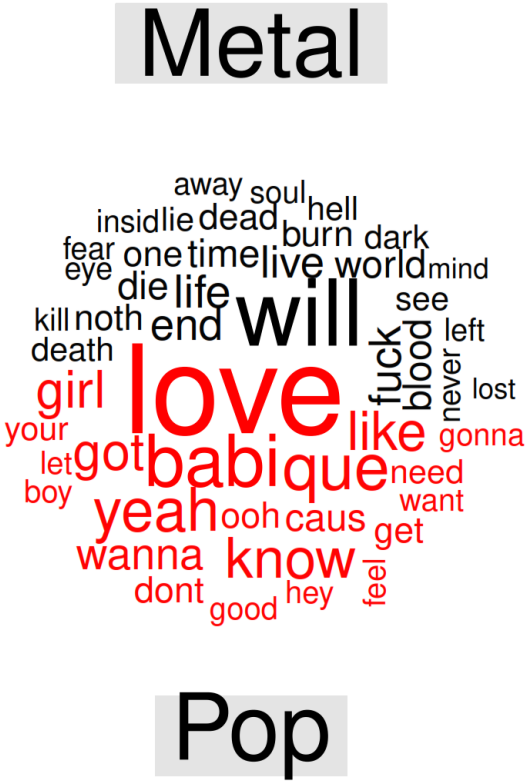
```
## <<TermDocumentMatrix (terms: 70584, documents: 2)>>
## Non-/sparse entries: 79801/61367
## Sparsity           : 43%
## Maximal term length: 67
## Weighting          : term frequency (tf)
## Sample             :
##       Docs
## Terms     1     2
##   can  1518  8428
##   dont 2101 12333
##   get  1171  7728
##   just 1836 10029
##   know 1789 12927
##   let  1412  8532
##   like 1528 11512
##   love 1112 16969
##   now  1789  7316
##   your 1336  8388
```

Convert to martix

```
metal_pop_m <- as.matrix(metal_pop_tdm)
colnames(metal_pop_m) <- c("Metal", "Pop")
```

comparison.cloud() compares terms frequencies for two documents

```
set.seed(1)
comparison.cloud(metal_pop_m, max.words=50,
                 colors = c("black", "red"),
                 random.order=F
                 )
```



Lets analyse the common words

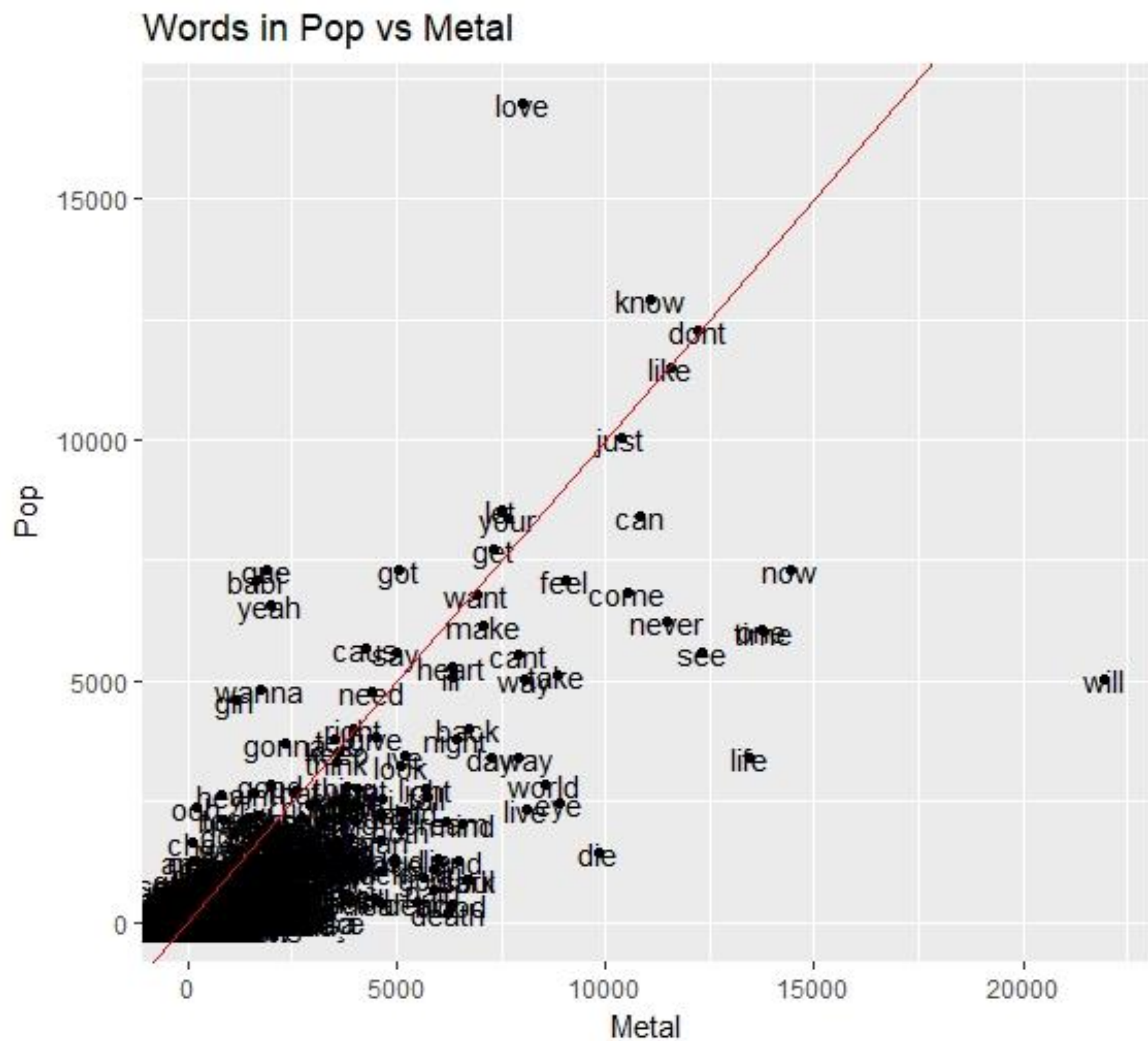
```
common <- data.frame(terms = rownames(metal_pop_m),metal_pop_m) %>%
  filter(Metal>0 & Pop>0)
head(common)
```

```
##     terms Metal Pop
## 1 aaaaaah     3   2
## 2  aaaaah     1   3
## 3   aaaah     3   3
## 4    aaah     6  11
## 5     aah     1  72
## 6     aap     1   9
```

Plotting

```
ggplot(common, aes(x=Metal, y=Pop))+geom_point()+geom_text(aes(label=terms))+
  geom_abline(intercept = 0, slope = 1, col="red")+
  ggtitle("Words in Pop vs Metal")
```

Text mining


Words in Pop vs Metal
Pop
Metal
0
5000
10000
15000
20000
love
know
dont
like
just
can
your
get
got
feel
come
now
want
make
never
time
see
yeah
cause
say
heart
cant
take
way
will
wanna
need
back
night
day
life
world
live
eye
die
death

Sentiment Analysis with R

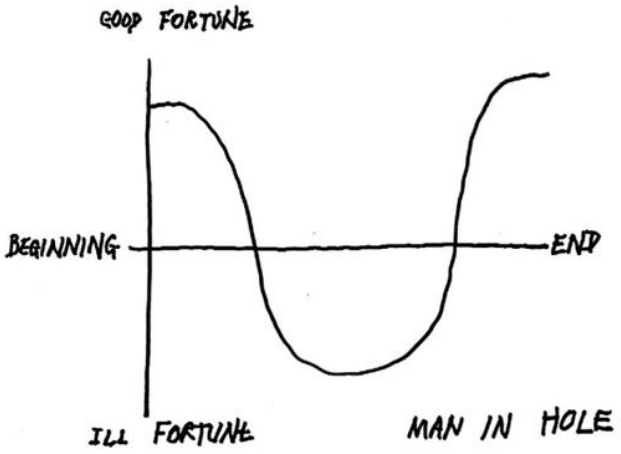
Sentiment Analysis

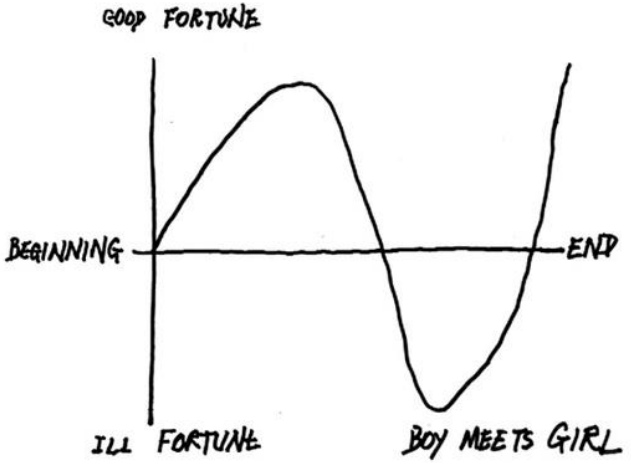
Kurt Vonnegut

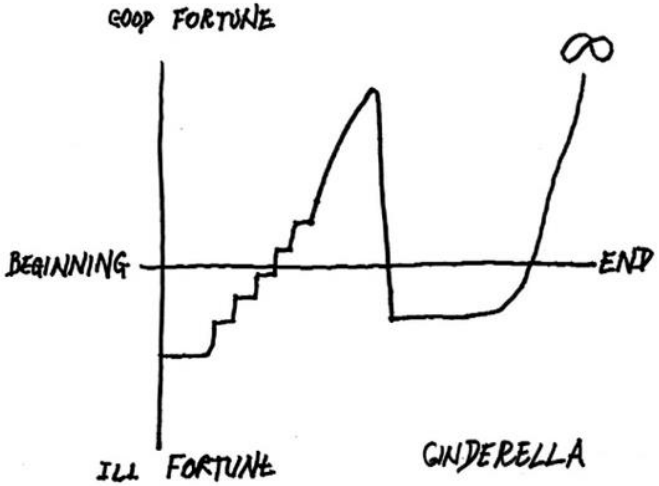
https://en.wikipedia.org/wiki/Kurt_Vonnegut

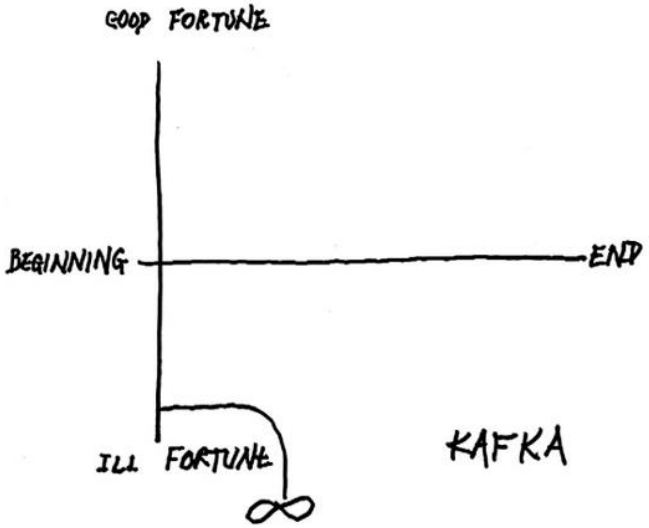
In general, there are only 4 types of stories

Sentiment Analysis





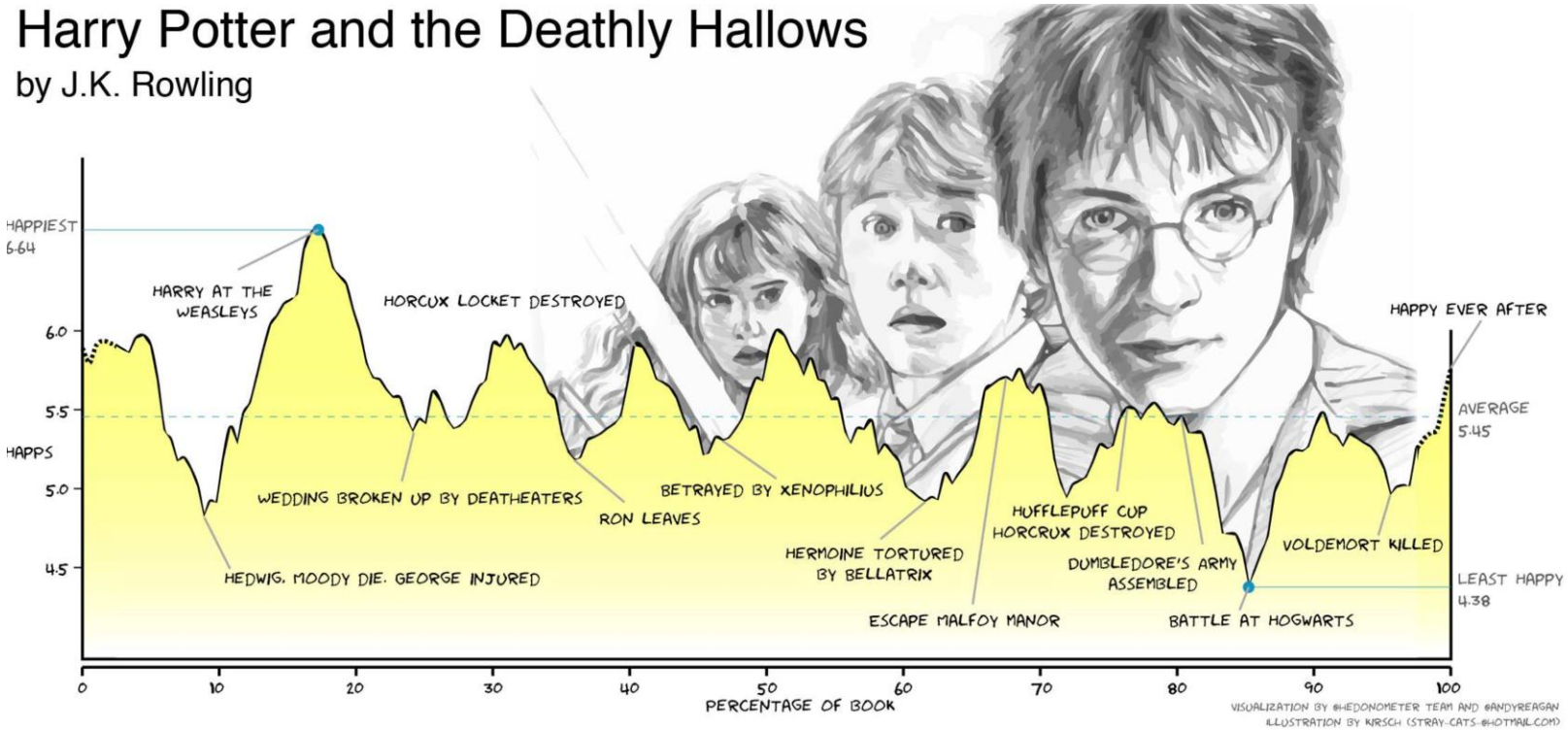




Sentiment Analysis

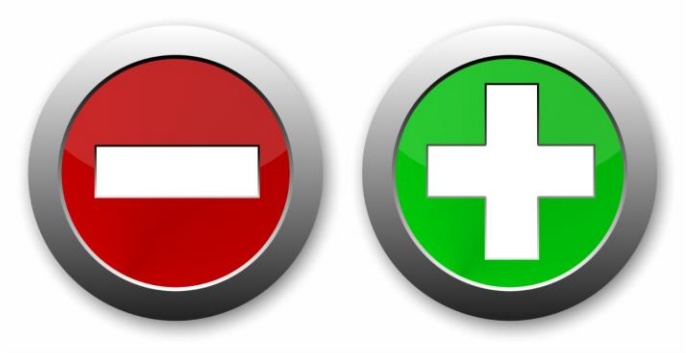
- In 2016 a group of scientists proved that Vonnegut was right
  The emotional arcs of stories are dominated by six basic shapes

- Analyzing 1,327 stories, with the usage of text mining, natural language processing and some machine learning techniques, found out that there are 6 basic shapes of stories.



Sentiment Analysis

Sentiment analysis is *the process of extracting an author's emotional intent from text*



Sentiment Analysis

Four sentences, how will you grade sentiment there (negative, positive ?)

1. unbelievably disappointing
2. Full of zany characters and richly applied satire, and some great plot twists
3. this is the greatest screwball comedy ever filmed
4. It was pathetic. The worst part about it was the boxing scenes.

Why to do sentiment analysis?

- **Movie:** is this review positive or negative?
- **Products:** what do people think about the new iPhone?
- **Public sentiment:** how is consumer confidence? Is despair increasing?
- **Politics:** what do people think about this candidate or issue?
- **Prediction:** predict election outcomes or market trends from sentiment

Other names for sentiment analysis

- Opinion extraction
- Opinion mining
- Sentiment mining
- Subjectivity analysis

R library qdap

Qualitative Data and Quantitative Analysis

```
install.packages("qdap")
library(qdap)
```

Sentiment Analysis

qdap::polarity function

Counts polarity score for a piece of text

- **Negative** value means there is a negative emotion in the text

- **Positive** value means there is a positive emotion in the text

polarity also allows to look for the context as well

Polarized Term

associated with
positive/negative

NeutralTerm
no emotional context

Negator
words that invert polarized meaning e.g. "not good"

Context cluster glossary

Valence Shifters

words that effect the emotional context

Amplifiers

words that increase emotional intent

De-Amplifiers

words that decrease emotional intent

Sentiment Analysis

```
deamplification.words
```

```
##  [1] "barely"       "faintly"      "few"          "hardly"
##  [5] "little"       "only"         "rarely"       "seldom"
##  [9] "slightly"     "sparsely"     "sporadically" "very few"
## [13] "very little"
```

```
amplification.words
```

```
##  [1] "acute"          "acutely"       "certain"       "certainly"
##  [5] "colossal"       "colossally"    "deep"          "deeply"
##  [9] "definite"       "definitely"    "enormous"      "enormously"
## [13] "extreme"        "extremely"     "great"         "greatly"
## [17] "heavily"        "heavy"         "high"          "highly"
## [21] "huge"           "hugely"        "immense"       "immensely"
## [25] "incalculable"   "incalculably"  "massive"       "massively"
## [29] "more"           "particular"    "particularly"  "purpose"
## [33] "purposely"      "quite"         "real"          "really"
## [37] "serious"        "seriously"     "severe"        "severely"
## [41] "significant"    "significantly" "sure"          "surely"
## [45] "true"           "truly"         "vast"          "vastly"
## [49] "very"
```

Sentiment Analysis

```
negation.words
```

```
##  [1] "ain't"     "aren't"    "can't"     "couldn't"  "didn't"
##  [6] "doesn't"   "don't"     "hasn't"    "isn't"     "mightn't"
## [11] "mustn't"   "neither"   "never"     "no"        "nobody"
## [16] "nor"       "not"       "shan't"    "shouldn't" "wasn't"
## [21] "weren't"   "won't"     "wouldn't"
```

Sentiment Analysis

```
length(negative.words)
```

```
## [1] 4776
```

```
negative.words[1:20]
```

```
##  [1] "abnormal"      "abolish"       "abominable"    "abominably"
##  [5] "abominate"     "abomination"   "abort"         "aborted"
##  [9] "aborts"        "abrade"        "abrasive"      "abrupt"
## [13] "abruptly"      "abscond"       "absence"       "absent minded"
## [17] "absentee"      "absurd"        "absurdity"     "absurdly"
```

```
length(positive.words)
```

```
## [1] 2003
```

```
positive.words[1:20]
```

```
##  [1] "a plus"          "abound"          "abounds"
##  [4] "abundance"       "abundant"        "accessable"
##  [7] "accessible"      "acclaim"         "acclaimed"
## [10] "acclamation"     "accolade"        "accolades"
## [13] "accommodative"   "accomodative"    "accomplish"
## [16] "accomplished"    "accomplishment"  "accomplishments"
## [19] "accurate"        "accurately"
```

Sentiment Analysis

- Formula for polarity score

$$(-1 * number\_of\_negative\_words + 1 * number\_of\_positive\_words \\ 0.8 * ammplifier)/sqrt(number\ of\ words)$$

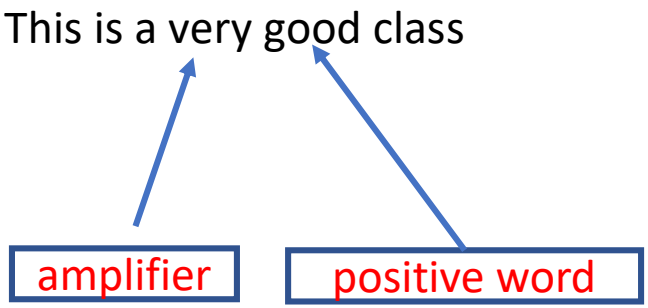
- by default the polarity() function looks for amplifiers for 4 words before and 2 words after the given text

- If there is an amplifier its weight (0.8 by default) is added to the words negative/positive score

"This is a very good class"

"I hate coming early in the morning“

"I deeply love the course , but i hate Sports"



score = $\frac{1+0.8}{\sqrt{6}} = 0.735$

You define the ***grouping variable*** here to show that the texts come from different authors

```
str<- c("This is a very good class", "I hate coming early in the morning",
        "I deeply love the course , but i hate Sports")
str_df <- data.frame(str, author=c(1:3))
library(qdap)
polarity(text.var = str_df$str, grouping.var = str_df$author )
```

```
##   author total.sentences total.words ave.polarity sd.polarity stan.mean.polarity
## 1      1               1           6        0.735          NA                 NA
## 2      2               1           7       -0.378          NA                 NA
## 3      3               1           9        0.267          NA                 NA
```

Sentiment Analysis

What will be the polaorty score of the sentence?

"Don't worry be happy"

Is Beyoncé whinny ?

To save time on processing, we will analysie only 50 songs from Beyoncé

```
beyonce <- lyrics[lyrics$artist =='beyonce-knowles',]
beyonce <- beyonce[1:50,]
beyonce_sent <- polarity(text.var = beyonce$lyrics, grouping.var = beyonce$song)
beyonce_scores <- scores(beyonce_sent)
head(beyonce_scores)
```

```
##                              song total.sentences total.words ave.polarity sd.polarity stan.me
## 1              a-girl-with-no-name               1         223        0.000          NA
## 2           all-i-could-do-was-cry               1         144       -0.500          NA
## 3                       amor-gitano               1         286       -0.237          NA
## 4                             angel               1         339        3.965          NA
## 5                    beautiful-liar               1         328        0.000          NA
```

Most positive songs

```
beyonce_scores %>%
  arrange(desc(ave.polarity)) %>%
  select(song, ave.polarity) %>%
  head(n=5)
```

```
##                      song ave.polarity
## 1                   angel     3.964816
## 2              ego-remix      1.931889
## 3   once-in-a-lifetime      1.606934
## 4         the-first-day      1.510400
## 5 what-s-good-with-you      1.400000
```

```
beyonce_scores %>%
  arrange(ave.polarity) %>%
  select(song, ave.polarity) %>%
  head(n=5)
```

```
##                          song ave.polarity
## 1                      poison   -1.0850595
## 2               my-first-time   -0.6562050
## 3 keep-givin-your-love-to-me   -0.6305926
## 4     all-i-could-do-was-cry   -0.5000000
## 5     no-broken-hearted-girl   -0.4636364
```

Most negative songs

```
beyonce_scores %>%
  arrange(ave.polarity) %>%
  select(song, ave.polarity) %>%
  head(n=5)
```

```
##                          song ave.polarity
## 1                      poison   -1.0850595
## 2               my-first-time   -0.6562050
## 3 keep-givin-your-love-to-me   -0.6305926
## 4     all-i-could-do-was-cry   -0.5000000
## 5     no-broken-hearted-girl   -0.4636364
```

How many negative and positive songs are there ?

```
# Positive songs
sum(beyonce_scores$ave.polarity>0)
```

```
## [1] 25
```

```
# Negative songs
sum(beyonce_scores$ave.polarity<0)
```

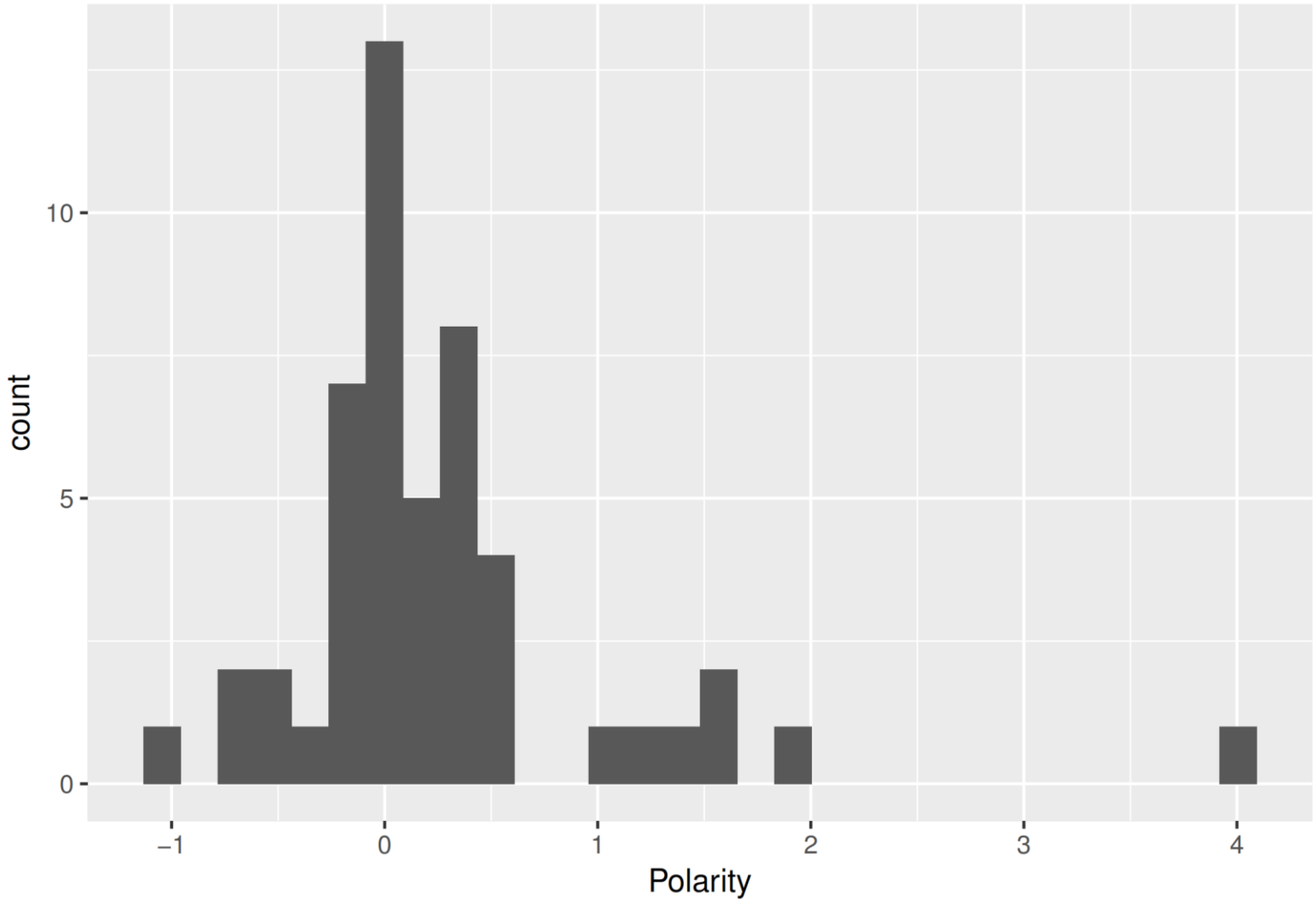
```
## [1] 21
```

```
# Neutral
sum(beyonce_scores$ave.polarity==0)
```

```
## [1] 4
```

Sentiment Analysis

```
ggplot(beyonce_scores, aes(x=ave.polarity))+geom_histogram()+
  labs(x="Polarity","Polarity distribution for Beyonce's 50 songs")
```



Sentiment Analysis

David Bowie

Bob Dylan

Sentiment Analysis

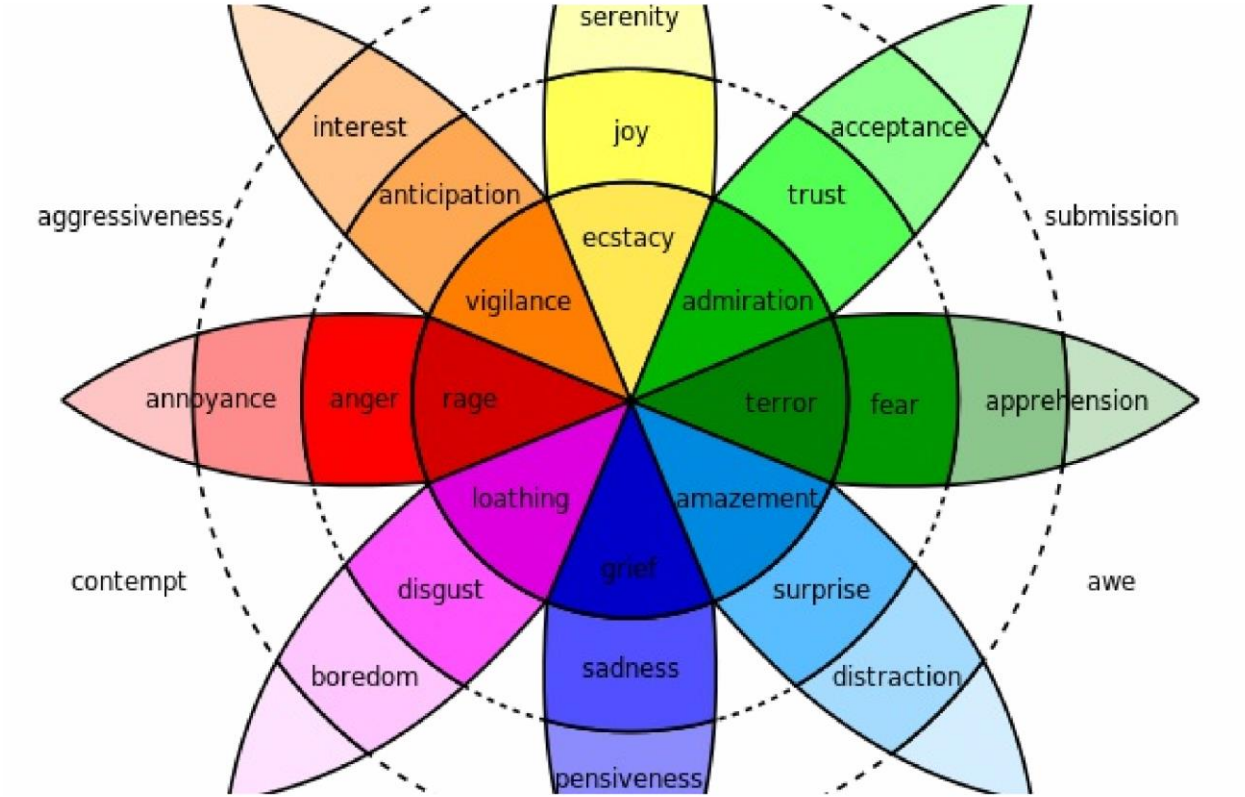
Grouped by artist

```
df_n <- lyrics[lyrics$artist %in% c('david-bowie','bob-dylan'),]
df_n_sent <- polarity(text.var = df_n$lyrics, grouping.var = df_n$artist)
df_n_scores <- scores(df_n_sent)
df_n_scores
```

```
##        artist total.sentences total.words ave.polarity sd.polarity stan.mean.polarity
## 1   bob-dylan             585      151537        0.030       0.533              0.056
## 2 david-bowie             549      112962        0.024       0.792              0.030
```

Sentiment Analysis

There are numerous other sentiment analysis tools and libraries



Plutchik's wheel of emotions