Folgend: Interessante NLP-Packages

Montag, 30. März 2020 18:15

Praktische Beispiele sind immer in den einzelnen Abschnitten:

koRpus: Bekommen von Statistiken zu Wordcounts, etc. Readability, lexical.diversity

wordcloud: Visualisieren von Wörtern Anhand ihrer Häufigkeit

hunspell - High Performance Stemmer: Checkt Rechtschreibung, parsed text, schlägt richtige Wörter vor, sucht Wortstamm von Wörtern

tidytext: Hauptsächlich Sammlung von Hilfen bei der Datenverarbeitung + Sammlung von Stopwörtern, Sentiment-Wörtern und Verstärker/Abschwächer-Worten, z.B. zum Aufbau einer eigenen Sentiment-Analyse

udpipe: Generell ist es nicht das, was wir brauchen, **aber es hat super Textverarbeitungsmöglichkeiten!**

koRpus

Dienstag, 3. März 2020 14:39

koRpus (https://cran.r-project.org/web/packages/koRpus/index.html)

- Für Infos zum richtigen installieren: https://reaktanz.de/?c=hacking&s=koRpus
- Doku: https://reaktanz.de/R/pckg/koRpus/koRpus.pdf
- Benötigt für vieles: TreeTagger - https://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/
- Problem: Schaut recht kompliziert aus, hinsichtlich des Formates, was die Daten haben müssen
  - Kann sehr aufwendig sein, bis wir uns da reingearbeitet haben, wie das richtig funktioniert!
- Vorteil: die Readability-Packages sehen sehr mächtig aus - siehe z.B. Doku Seite 45
- Interessante Funktionen
  - lexical diversity-Funktionen: wie oft werden Wörter wiederholt, etc.
    -> für bessere Beschreibung: https://textinspector.com/help/lexical-diversity/
    -> gibt einen Haufen Indexe, da müssten wir uns noch näher reinlesen!
    - $lex.div$

```
C characteristics:
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.8255  0.8733  0.8857  0.8952  0.9279  1.0000
   SD
 0.0404


Guiraud's R
   R: 5.2

R characteristics:
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  2.236   5.138   5.391   5.127   5.683   6.155
   SD
 0.9539


Carroll's CTTR
   CTTR: 3.68

CTTR characteristics:
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 1.581   3.633   3.812   3.625   4.019   4.352
   SD
 0.6745


Uber Index
```

  - Readability-Funktionen z.B. Komplexität der Sätze/Wörter, Reading Ease, etc.
    -> müssten wir uns auch reinlesen
    - $readability$

	index	flavour	raw	grade	age
1	ARI			10.7	
2	Coleman-Liau		72	3.29	
3	Danielson-Bryan DB1		7.29		
4	Danielson-Bryan DB2		52.13	7-8	
5	Dickes-Steiwer		63.19		
6	ELF		2.8		
7	Farr-Jenkins-Paterson		81.8	6	
8	Flesch en (Flesch)		81.67	6	
9	Flesch-Kincaid			9.87	14.9
10	FOG			13.13	
11	FORCAST			6.3	11.3
12	Fucks		109.6	10.47	
13	Linsear-Write			16.3	
14	LIX		38.41	6	
15	nWS1			2.45	
16	nWS2			3.61	
17	nWS3			5.2	
18	nWS4			7.03	
19	RIX		2	6	
20	SMOG			5.68	10.7
21	Strain		10.56		
22	TRI		2.25		
23	Tuldava		3.8		
24	Wheeler-Smith		28	4	

- $flesch$
- $flesch - kincaid$
- $forcast$

- $guess\_lang$ -> Funktion, was die Sprache erkennt
- $freq.analysis$

	freq
sentences	5.000000
avg.sentence.length	32.200000
words	161.000000
avg.word.length	3.403727
all.characters	762.000000
letters	548.000000
lemmata	1.000000
questions	5.000000
exclamations	0.000000
semicolon	0.000000
colon	0.000000

- $textFeatures$

```
> textFeatures(test2)
  uniqWd    complx sntCt sntLen   syllCt charCt lttrCt      FOG   flesch
1     69 0.4099379     5   32.2 1.093168    762    548 13.12845 81.67001
```

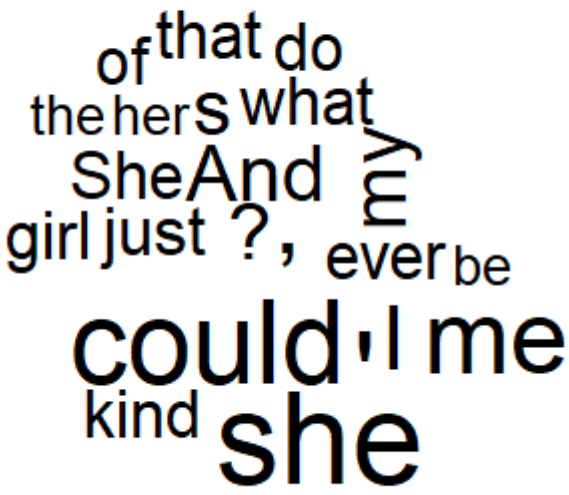
- $treeTag$ - *damit werden Daten vorbereitet*

wordcloud

Montag, 30. März 2020 18:14

Interessante Funktionen:

- *wordcloud* - visualisiert Wörter Anhand ihrer Häufigkeit



hunspell - High Performance Stemmer

Montag, 30. März 2020 18:44

Interessante Funktionen:

- *hunspell*(*text*) - testet auf Rechtschreibfehler im Englischen und gibt Fehler zurück
  -
```
> test$song[1]
[1] "Ahe's My Kind Of Girl"
> hunspell(test$song[1], format = "text")
[[1]]
[1] "Ahe's"
```
- *hunspell_parse*(*text*) - entfernt Satzzeichen
  -
```
> hunspell_parse(test$text[1])
[[1]]
  [1] "Look"      "at"        "her"       "face"      "it's"      "a"         "wonderful" "face"      "And"       "it"        "means"     "something"
 [13] "special"   "to"        "me"        "Look"      "at"        "the"       "way"       "that"      "she"       "smiles"    "when"      "she"
 [25] "sees"      "me"        "How"       "lucky"     "can"       "one"       "fellow"    "be"        "She's"     "just"      "my"        "kind"
 [37] "of"        "girl"      "she"       "makes"     "me"        "feel"      "fine"      "who"       "could"     "ever"      "believe"   "that"
 [49] "she"       "could"     "be"        "mine"      "She's"     "just"      "my"        "kind"      "of"        "girl"      "without"   "her"
 [61] "I'm"       "blue"      "And"       "if"        "she"       "ever"      "leaves"    "me"        "what"      "could"     "I"         "do"
 [73] "what"      "could"     "I"         "do"        "And"       "when"      "we"        "go"        "for"       "a"         "walk"      "in"
 [85] "the"       "park"      "And"       "she"       "holds"     "me"        "and"       "squeezes"  "my"        "hand"      "We'll"     "go"
 [97] "on"        "walking"   "for"       "hours"     "and"       "talking"   "About"     "all"       "the"       "things"    "that"      "we"
[109] "plan"      "She's"     "just"      "my"        "kind"      "of"        "girl"      "she"       "makes"     "me"        "feel"      "fine"
[121] "who"       "could"     "ever"      "believe"   "that"      "she"       "could"     "be"        "mine"      "She's"     "just"      "my"
[133] "kind"      "of"        "girl"      "without"   "her"       "I'm"       "blue"      "And"       "if"        "she"       "ever"      "leaves"
[145] "me"        "what"      "could"     "I"         "do"        "what"      "could"     "I"         "do"
```
- *hunspell_check* - testet darauf ob einzelne Wörter falsch geschrieben sind
  -
```
> hunspell_parse(test$song[1])
[[1]]
[1] "Ahe's" "My"    "Kind"  "Of"    "Girl"

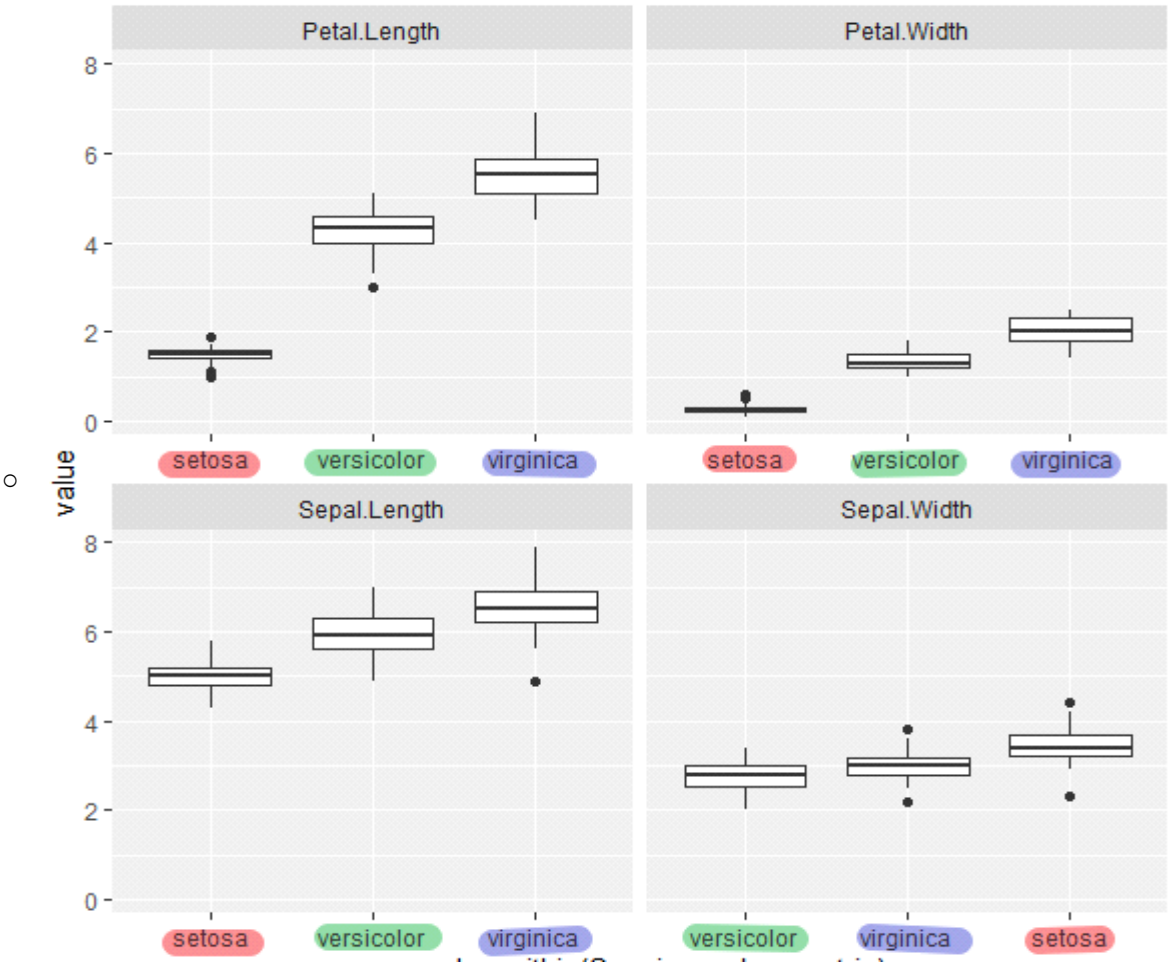
> hunspell_check(unlist(hunspell_parse(test$song[1])))
[1] FALSE  TRUE  TRUE  TRUE  TRUE
```
- *hunspell_suggest* - schlägt Wörter für falsche Wörter vor
  -
```
> unlist(hunspell_parse(test$song[1]))[!hunspell_check(unlist(hunspell_parse(test$song[1])))]
[1] "Ahe's"
> hunspell_suggest(unlist(hunspell_parse(test$song[1]))[!hunspell_check(unlist(hunspell_parse(test$song[1])))])
[[1]]
 [1] "He's"   "Ashe's" "She's"  "Abe's"  "Ave's"  "Che's"  "A he's" "Age's"  "Ache's" "Are's"  "Ale's"  "Ace's"  "Ape's"  "Aye's"  "Awe's"
```
- *hunspell_stem* - gibt Wortstamm für ein Wort aus, damit es in einer Wortwolke zusammengefasst werden kann (z.B. she und she's zusammenfassen) - funktioniert aber nicht Perfekt, siehe "her" -> "h" beim stem
  -
```
> unlist(hunspell_parse(test$text[1]))
  [1] "Look"      "at"        "her"       "face"      "it's"      "a"         "wonderful" "face"      "And"       "it"        "means"     "something"
 [13] "special"   "to"        "me"        "Look"      "at"        "the"       "way"       "that"      "she"       "smiles"    "when"      "she"
 [25] "sees"      "me"        "How"       "lucky"     "can"       "one"       "fellow"    "be"        "She's"     "just"      "my"        "kind"
 [37] "of"        "girl"      "she"       "makes"     "me"        "feel"      "fine"      "who"       "could"     "ever"      "believe"   "that"
 [49] "she"       "could"     "be"        "mine"      "She's"     "just"      "my"        "kind"      "of"        "girl"      "without"   "her"
 [61] "I'm"       "blue"      "And"       "if"        "she"       "ever"      "leaves"    "me"        "what"      "could"     "I"         "do"
 [73] "what"      "could"     "I"         "do"        "And"       "when"      "we"        "go"        "for"       "a"         "walk"      "in"
 [85] "the"       "park"      "And"       "she"       "holds"     "me"        "and"       "squeezes"  "my"        "hand"      "We'll"     "go"
 [97] "on"        "walking"   "for"       "hours"     "and"       "talking"   "About"     "all"       "the"       "things"    "that"      "we"
[109] "plan"      "She's"     "just"      "my"        "kind"      "of"        "girl"      "she"       "makes"     "me"        "feel"      "fine"
[121] "who"       "could"     "ever"      "believe"   "that"      "she"       "could"     "be"        "mine"      "She's"     "just"      "my"
[133] "kind"      "of"        "girl"      "without"   "her"       "I'm"       "blue"      "And"       "if"        "she"       "ever"      "leaves"
[145] "me"        "what"      "could"     "I"         "do"        "what"      "could"     "I"         "do"
> unlist(hunspell_stem(unlist(hunspell_parse(test$text[1]))))
  [1] "look"      "at"        "h"         "face"      "it"        "a"         "wonderful" "face"      "and"       "it"        "mean"      "something"
 [13] "special"   "to"        "me"        "look"      "at"        "the"       "way"       "that"      "she"       "smile"     "when"      "she"
 [25] "see"       "me"        "how"       "lucky"     "can"       "one"       "fellow"    "be"        "she"       "just"      "my"        "kind"
 [37] "of"        "girl"      "she"       "make"      "me"        "feel"      "fine"      "who"       "who"       "could"     "ever"      "believe"
 [49] "that"      "she"       "could"     "be"        "mine"      "she"       "just"      "my"        "kind"      "of"        "girl"      "without"
 [61] "h"         "I'm"       "blue"      "and"       "if"        "she"       "ever"      "leave"     "me"        "what"      "could"     "i"
 [73] "I"         "do"        "what"      "could"     "i"         "I"         "do"        "and"       "when"      "we"        "go"        "for"
```

tidytext

Montag, 30. März 2020 21:02

Doku: https://cran.r-project.org/web/packages/tidytext/tidytext.pdf

Interessante Funktionen:

- Reinlesen in: *pivot_wider* und in *pivot_longer* - ähnlich wie melt und dcast aber besser
- *reorder_within - sortiert die Daten bei facet_grid nach der Größe der Werte*
  - 

- *get_sentiment* - 4 Arten:
  - ```
    get_sentiments("nrc")
    A tibble: 13,901 x 2
     word        sentiment
     <chr>       <chr>
    1 abacus      trust
    2 abandon     fear
    3 abandon     negative
    4 abandon     sadness
    5 abandoned   anger
    6 abandoned   fear
    7 abandoned   negative
    8 abandoned   sadness
    9 abandonment anger
    0 abandonment fear
    ```
    ```

```
get_sentiments("bing")
A tibble: 6,786 x 2
 word        sentiment
 <chr>       <chr>
 2-faces     negative
 abnormal    negative
 abolish     negative
 abominable  negative
 abominably  negative
 abominate   negative
 abomination negative
 abort       negative
 aborted     negative
 aborts      negative
... with 6,776 more rows
```

```
get_sentiments("loughran")
A tibble: 4,150 x 2
 word         sentiment
 <chr>        <chr>
 abandon      negative
 abandoned    negative
 abandoning   negative
 abandonment  negative
 abandonments negative
 abandons     negative
 abdicated    negative
 abdicates    negative
 abdicating   negative
 abdication   negative
... with 4,140 more rows
```

```
> get_sentiments(lexicon = "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
```

- *sentiments - entspricht: get_sentiments("bing")*
- $get\_stopwords(language = "en, source = snowball)$

```
get_stopwords(source = "smart")
A tibble: 571 x 2
 word        lexicon
 <chr>       <chr>
 a           smart
 a's         smart
 able        smart
 about       smart
 above       smart
 according   smart
 accordingly smart
 across      smart
 actually    smart
 after       smart
```

```
get_stopwords(source = "snowball")
A tibble: 175 x 2
 word      lexicon
 <chr>     <chr>
 i         snowball
 me        snowball
 my        snowball
 myself    snowball
 we        snowball
 our       snowball
 ours      snowball
 ourselves snowball
 you       snowball
 your      snowball
... with 165 more rows
```

```
get_stopwords(source = "stopwords-iso")
A tibble: 1,298 x 2
 word      lexicon
 <chr>     <chr>
 'll       stopwords-iso
 'tis      stopwords-iso
 'twas     stopwords-iso
 've       stopwords-iso
 10        stopwords-iso
 39        stopwords-iso
 a         stopwords-iso
 a's       stopwords-iso
 able      stopwords-iso
 ableabout stopwords-iso
... with 1,288 more rows
```

- *nma_words* - Englische Negatoren, Modalverben oder Adverben

```
nma_words
A tibble: 44 x 2
 word      modifier
 <chr>     <chr>
 cannot    negator
 could not negator
 did not   negator
 does not  negator
 had no    negator
 have no   negator
 may not   negator
 never     negator
 no        negator
 not       negator
... with 34 more rows
```

udpipe

Dienstag, 31. März 2020 11:15

Doku: https://cran.r-project.org/web/packages/udpipe/udpipe.pdf

Interessante Funktionen:

- *paste.data.frame* - paste, was auch auf Data-Frames anwendbar ist -> kann z.B. Gruppen zusammenfügen

```
y = paste.data.frame(data = x, term = "feedback", group = c("listing_id", "language"))
glimpse(y)
bservations: 1,184
ariables: 3
 listing_id <int> 1291276, 1274584, 1991750, 2576349, 1866754, 5247223, 7925019, 4442255, 2863621, 3117760, 4384392, 346290, 2736998, 3403812, 4213027,...
 language   <chr> "es", "es", "es", "es", "es", "es", "es", "es", "es", "es", "es", "es", "es", "es", "es", "es", "es", "es", "es", "es", "es", "es", "...
 feedback   <chr> "Gwen fue una magnifica anfitriona El motivo de mi viaje a Bruselas era la busqueda de un apartamento y Gwen me ayudo en todo momento...
```

- *strsplit.data.frame* - teilt z.B. Sätze in einem DF an speziellen Ausdrücken

```
> glimpse(brussels_reviews)
observations: 1,500
variables: 4
$ id         <int> 32198807, 12919832, 23786310, 20048068, 17571798, 28394425, 46322841, 27719650, 14512388, 37675819, 25495201, 45083625, 37550220, 417...
$ listing_id <int> 1291276, 1274584, 1991750, 2576349, 1866754, 5247223, 7925019, 4442255, 2863621, 3117760, 4384392, 346290, 2736998, 3403812, 4213027,...
$ feedback   <chr> "Gwen fue una magnifica anfitriona. El motivo de mi viaje a Bruselas era la busqueda de un apartamento y Gwen me ayudo en todo moment...
$ language   <chr> "es", "es", "es", "es", "es", "es", "es", "es", "es", "es", "es", "es", "es", "es", "es", "es", "es", "es", "es", "es", "es", "es", "...
> x <- strsplit.data.frame(brussels_reviews, term = "feedback", group = "id")
> head(x)
        id   feedback
1 32198807       Gwen
2 32198807        fue
3 32198807        una
4 32198807  magnifica
5 32198807 anfitriona
6 32198807         El
>
```

- *txt_collapse* - ähnlich wie Paste, nur mit NA-Entfernung

```
> txt_collapse(c(NA, "hello", "world", NA))
[1] "hello world"
```

- *txt_contains* - sucht explizit nach den enthaltenen pattern als regex

```
x <- c("The cats are eating catfood","Our cat is eating the catfood","the dog eats catfood, he likes it")
txt_contains(x, patterns = c("cat", "dog"))
L] TRUE TRUE TRUE
txt_contains(x, patterns = c("cat", "dog"), value = T)
L] "The cats are eating catfood"      "Our cat is eating the catfood"     "the dog eats catfood, he likes it"
txt_contains(x, patterns = c("^The"), value = T)
L] "The cats are eating catfood"      "the dog eats catfood, he likes it"
```

- *txt_freq* - gruppiert Wörter nach ihrer Frequenz. Ist die Variable ein Faktor, wird nach der Faktorreihenfolge geordnet.

```
x <- sample(LETTERS, 1000, replace = TRUE)
x
 [1] "F" "S" "J" "Z" "Z" "S" "X" "O" "I" "J" "T" "E" "M" "J" "A" "V" "Y" "T" "B" "R" "
[38] "S" "T" "O" "E" "E" "O" "S" "C" "U" "P" "D" "M" "V" "Y" "O" "N" "E" "L" "I" "C" "
[75] "R" "U" "I" "N" "C" "S" "N" "N" "J" "R" "P" "J" "G" "U" "A" "Q" "I" "X" "V" "U" "
[112] "E" "N" "M" "I" "A" "P" "H" "Y" "K" "L" "Y" "I" "B" "E" "I" "L" "T" "L" "D" "Q" "

txt_freq(x)
 key freq freq_pct
   Z   53      5.3
   S   47      4.7
   Y   45      4.5
   L   45      4.5
   X   44      4.4
   E   44      4.4
   Q   43      4.3
   O   42      4.2

txt_freq(x, order = FALSE)
 key freq freq_pct
   A   37      3.7
   B   36      3.6
   C   42      4.2
   D   27      2.7
   E   44      4.4
   F   38      3.8
   G   34      3.4
   H   40      4.0
   I   39      3.9
```

- *txt_highlight* - hebt Wörter in Sätzen mit |*wort*| heraus

```
> x <- "I like milk and sugar in my coffee."
> txt_highlight(x, terms = "sugar")
[1] "I like milk and |sugar| in my coffee."
> txt_highlight(x, terms = c("milk", "my"))
[1] "I like |milk| and sugar in |my| coffee."
```

- *txt_next* & *txt_nextogram:* Entfernt die $n - ten$ Worte davor oder bindet n Wörter zusammen

```
x
[1] "A1"  "B2"  "C3"  "D4"
26] "Z26"
txt_next(x, n = 1)
[1] "B2"  "C3"  "D4"  "E5"

txt_nextgram(x, n = 2)
1] "A1 B2"   "B2 C3"   "C3 D4"   "D4 E5"
```

  - *txt_previous* und *txt_previousgram* ist Umkehrung von *text_next*

- *txt_recode_ngram:* Wenn man Wörter trennt, wo man weiß, dass sie zusammengehören (z.B. Städte-Namen) kann man einen Vektor eingeben, wie die Städte wirklich heißen würden und diese werden wieder zusammengefügt.
  Achtung: Groß- und Kleinschreibung wichtig
  Achtung 2: *ngram* ist der Parameter der angiebt, wie viele aufeinanderfolgende Indizes berücksichtigt werden sollen - siehe Beispiel.

```
x <- c("I", "went", "to", "New", "York", "City", "and", "Salt", "Lake", "City", "on", "holiday", ".")

> txt_recode_ngram(x, compound = c("New York City", "Salt Lake City"), ngram = 2, sep = " ")
 [1] "I"       "went"    "to"      "New"     "York"    "City"    "and"     "Salt"    "Lake"    "City"    "on"      "holiday" "."
> txt_recode_ngram(x, compound = c("New York City", "Salt Lake City"), ngram = 3, sep = " ")
 [1] "I"             "went"          "to"            "New York City" NA              NA              "and"           "Salt Lake City" NA
[10] NA              "on"            "holiday"       "."
```

- *txt_sentiment*: Führt Wörterbuch-basierende Sentiment-Analysen an einem tokenisierten Data-Frame durch.
  Identifiziert positive/negative Bedeutungen und checkt auf *amplifiers/deamplifiers*
  (z.B. Sehr gut vs nicht sehr gut)
  Will man ein supervised model bauen, um sentiment zu scoren: ***ruimtchol* R-package**

- Der Input ist genau das, was von der Funktion *udpipe* zurückgegeben wird
- Problem: Muss alles händisch eingegeben werden - sehr aufwendig!
  - Könnte aber super mit *get_sentiment* von tidytext kombiniert werden
- Eher nicht verwenden, auch wenn ganz cool

➢ *udpipe*: Tokenising, Lemmatising (=Wortstammsuche), Tagging and Dependency Parsing of raw text in TIF format

- Beispiel:

```
x <- c("I do not like whatsoever when an R package has soo many dependencies.",
       "Making other people install java is annoying,
       as it is a really painful experience in classrooms.")
```

```
udpipe(x, "english-gum")
```

doc_id	paragraph_id	sentence_id	sentence	start	end	term_id	token_id	token	lemma	upos	xpos
doc1	1	1	I do not like whatsoever when an R package has soo ma...	1	1	1	1	I	I	PRON	PRP
doc1	1	1	I do not like whatsoever when an R package has soo ma...	3	4	2	2	do	do	AUX	VBP
doc1	1	1	I do not like whatsoever when an R package has soo ma...	6	8	3	3	not	not	PART	RB
doc1	1	1	I do not like whatsoever when an R package has soo ma...	10	13	4	4	like	like	VERB	VB
doc1	1	1	I do not like whatsoever when an R package has soo ma...	15	24	5	5	whatsoever	whatsoever	ADV	RB
doc1	1	1	I do not like whatsoever when an R package has soo ma...	26	29	6	6	when	when	SCONJ	WRB
doc1	1	1	I do not like whatsoever when an R package has soo ma...	31	32	7	7	an	an	DET	DT
doc1	1	1	I do not like whatsoever when an R package has soo ma...	34	34	8	8	R	r	ADJ	JJ
doc1	1	1	I do not like whatsoever when an R package has soo ma...	36	42	9	9	package	package	NOUN	NN
doc1	1	1	I do not like whatsoever when an R package has soo ma...	44	46	10	10	has	have	VERB	VBZ
doc1	1	1	I do not like whatsoever when an R package has soo ma...	48	50	11	11	soo	soo	ADV	RB
doc1	1	1	I do not like whatsoever when an R package has soo ma...	52	55	12	12	many	many	ADJ	JJ
doc1	1	1	I do not like whatsoever when an R package has soo ma...	57	68	13	13	dependencies	dependency	NOUN	NNS
doc1	1	1	I do not like whatsoever when an R package has soo ma...	69	69	14	14	.	.	PUNCT	.
doc2	1	1	Making other people install java is annoying, as it is a re...	1	6	1	1	Making	make	VERB	VBG
doc2	1	1	Making other people install java is annoying, as it is a re...	8	12	2	2	other	other	ADJ	JJ

feats	head_token_id	dep_rel	deps	misc
Case=Nom|Number=Sing|Person=1|PronType=Prs	4	nsubj	NA	NA
Mood=Ind|Tense=Pres|VerbForm=Fin	4	aux	NA	NA
Polarity=Neg	4	advmod	NA	NA
VerbForm=Inf	0	root	NA	NA
NA	4	advmod	NA	NA
PronType=Int	10	mark	NA	NA
Definite=Ind|PronType=Art	9	det	NA	NA
Degree=Pos	9	amod	NA	NA
Number=Sing	10	nsubj	NA	NA
Mood=Ind|Number=Sing|Person=3|Tense=Pres|VerbFor...	4	advcl	NA	NA
NA	12	advmod	NA	NA
Degree=Pos	13	amod	NA	NA
Number=Plur	10	obj	NA	SpaceAfter=No
NA	4	punct	NA	SpacesAfter=\n
VerbForm=Ger	0	root	NA	NA
Degree=Pos	3	amod	NA	NA
Number=Plur	1	obj	NA	NA
NA	5	det	NA	NA
Number=Sing	7	nsubj	NA	NA

- **Bedeutung der Variablen:**
  - doc_id: The document identifier.
  - paragraph_id: The paragraph identifier which is unique within each document.
  - sentence_id: The sentence identifier which is unique within each document.
  - sentence: The text of the sentence of the sentence_id.
  - start: Integer index indicating in the original text where the token starts. Missing in case of tokens part of multi-word tokens which are not in the text.
  - end: Integer index indicating in the original text where the token ends. Missing in case of tokens part of multi-word tokens which are not in the text.
  - term_id: A row identifier which is unique within the doc_id identifier.
  - token_id: Token index, integer starting at 1 for each new sentence. May be a range for multi-word tokens or a decimal number for empty nodes.
  - token: The token.
  - lemma: The lemma of the token.
  - upos: The universal parts of speech tag of the token. See http://universaldependencies.org/format.html
  - xpos: The treebank-specific parts of speech tag of the token. See http://universaldependencies.org/format.html
  - feats: The morphological features of the token, separated by |. See http://universaldependencies.org/format.html
  - head_token_id: Indicating what is the token_id of the head of the token, indicating to which other token in the sentence it is related. See http://universaldependencies.org/format.html
  - dep_rel: The type of relation the token has with the head_token_id. See http://universaldependencies.org/format.html
  - deps: Enhanced dependency graph in the form of a list of head-deprel pairs. See http://universaldependencies.org/format.html
  - misc: SpacesBefore/SpacesAfter/SpacesInToken spaces before/after/inside the token. Used to reconstruct the original text. See http://ufal.mff.cuni.cz/udpipe/users-manual

wordnet

Samstag, 25. April 2020 12:52

Ist ein Interface für WordNet von der Princeton University
https://wordnet.princeton.edu/

- Ist eine riesige lexicale Datenbank für Englisch, in der Nouns, verbs, adjectives und adverbs in ein Set von kognitiven Synonymen gruppiert werden ($synsets$), die alle ein unterschiedliche Konzept ansprechen.
- Sind verbunden durch konzeptuelle Semantic und lexicale Relationen.
- Gutes Tool für NLP und computational linguistics.
- Funktioniert ähnlich wie ein Thesaurus:
  - Wörter werden anhand ihrer Bedeutung gruppiert
    -> nicht nur nach Wortform, sondern auch nach dem Sinn der Worte
  - Wörter, die im Wortnetz nah beieinander liegen, sind semantisch ähnlich.
  - Diese semantischen Zusammenhänge werden von WordNet gelabelt.
- Struktur:
  - Hauptrelationen sind Synonyme, die das selbe Konzept ansprechen, aber austauschbar sind, z.B. car und automobile
  - Diese werden in ungeoordneten sets $synsets$ gespeichert
    -> WordNet umfasst 117 000 Synsets, die durch konzeptuelle Relationen verbunden sind
  - Hat ein Wort mehrere Bedeutungen, kommt es in ebensovielen synsets vor!
- Mehr zur Funktion auf https://wordnet.princeton.edu/

Muss vorher gedownloaded werden über:
http://wordnetcode.princeton.edu/3.0/WNdb-3.0.tar.gz

Aus <https://cran.r-project.org/web/packages/wordnet/wordnet.pdf>

Dann in R, nach laden des packages $wordnet$ mit setDict() auf den Pfad, wo es entpackt wurde, verwiesen werden

$Beispiel$:
- Pacman::p_load(wordnet)
- setDict("C:/Users/Michi/Google Drive/Data Science/6. Semester/Advanced Data Challenge/WordNet/dict")

Dann kann damit gearbeitet werden.

Parameter der Funktionen werden beschrieben in: http://mfwallace.googlepages.com/jawbone

Funktionen, die für uns relevant sein können:
- Bei $wordnet$ interagieren viele Funktionen, deshalb keine einzelnen zusammengefasst, sondern eher volle Funktion, aber die einzelnen Komponenten machen:
  - $getTermFilter(type, word, ignoreCase)$: erstellt einen Term-Filter für andere Funktionen
    - Type: Typ des Filters. Siehe hier
    - Word: Wort, welches gematched werden soll
    - ignoreCase: Soll zwischen Groß- und Kleinschreibung unterschieden werden?
  - $getIndexTerms(pos, maxLimit, filter)$: wirft eine Liste von index Terms aus
    - Pos: part of speech, z.B. NOUN, ADJECTIVE, usw.
    - maxLimit: Wie viele Results möchte ich haben
    - Filter: ist ein $getTermFilter()$-Objekt
  - $getLemma(indexterm)$
    - Ein Objekt, welches von $getIndexTerms$ erstellt wurde
      -> zieht Wort raus
    - Anwendungsbeispiel:

```
filter <- getTermFilter("StartsWithFilter", "car", TRUE)   kreeirt Filter nach Wörtern, die mit "car" starten, Schreibung wird ignoriert
terms = getIndexTerms("ADJECTIVE", 10, filter)  zieht alle Adjektive raus, die mit Car starten - n = 10
sapply(terms, getLemma)    zieht die Wörter aus den Term-Indizes raus

sapply(terms, getLemma)
[1] "caramel"        "caramel brown" "carangid"      "carbocyclic"   "carbolated"    "carbonaceous"  "carbonated"    "carbonic"      "carboniferous"
[10] "carbonous"
```

  - $getWord(synset)$ - zieht die Wörter aus einem synset raus
    - Anwendung: siehe untere Beispiele
  - $getSynsets(indexterm)$
    - Ein Objekt, welches von $getIndexTerms$ erstellt wurde
    - Gibt Synsets zurück, in denen das Wort vorkommt
    - Anwendungsbeispiel: (Anmerkung: $terms[[1]]$ sind immer die Synsets, in denen das Wort liegt bzw. u denen es ähnlich ist

```
filter <- getTermFilter("ExactMatchFilter", "red", TRUE)        matched "red" exact
terms <- getIndexTerms("ADJECTIVE", 5, filter)     5 Adjective, wo "red" exakt vorkommt
synsets <- getSynsets(terms[[1]])  nimmt nur die Terme selbst, bzw. holt sich die synsets für die Terme
sapply(synsets, getWord)    wirft die Worte aus den einzelnen Synsets aus

sapply(synsets, getWord)
[1]]
[1] "red"         "reddish"     "ruddy"       "blood-red"   "carmine"     "cerise"      "cherry"      "cherry-red"  "crimson"     "ruby"        "ruby-red"
[12] "scarlet"

[2]]
[1] "crimson" "red"     "violent"

[3]]
[1] "crimson"   "red"       "reddened"  "red-faced" "flushed"
```

  - $getRelatedSynsets(synset, pointerSymbol)$
    - Synset: Ein Objekt, welches von $getSynsets$ erstellt wurde
    - pointerSymbol: Sollen Relationen zwischen den einzelnen Wörtern angeben. Kurzzusammenfassung hier, längere auf https://wordnet.princeton.edu/documentation/wninput5wn
    - Anwendungsbeispiel:

```
filter <- getTermFilter("ExactMatchFilter", "hot", TRUE)     Filter
terms <- getIndexTerms("ADJECTIVE", 5, filter)  Spezifikation, dass es ein adjektiv ist
synsets <- getSynsets(terms[[1]])   holt Synsets
related <- getRelatedSynsets(synsets[[1]], "!")   holt das antagonistische Synset mit "!"
sapply(related, getWord)   zieht die Wörter aus dem antagonistischen Synset raus

sapply(related, getWord)
[1] "cold"
```

  - $getSynonyms(indexterm)$ - wirft Synonyme eines Wortes aus
    - Indexterm: Ein Objekt, welches von $getIndexTerms()$ erstellt wurde

- Anwendungsbeispiel:

```
filter <- getTermFilter("ExactMatchFilter", "company", TRUE)      kreiirt Filter für das Wort "company"
terms <- getIndexTerms("NOUN", 5, filter)  wirft die IndexTerms für Nomen aus
getSynonyms(terms[[1]])    wirft die Synonyme für diese Nomen aus
```

```
getSynonyms(terms[[1]])
L] "caller"          "companionship"  "company"        "fellowship"     "party"          "ship's company" "society"        "troupe"
```

- $synonymes(word, pos)$ - gibt Synonyme für ein eingegebenes Wort aus
  - Word: Das Wort, für welches wir Synonyme haben wollen
  - pos: part of speech (NOUN, VERB, etc.)

Filtertypen, die in $getTermFilter()$ verwendet werden können:

- **ContainsFilter** - Matches terms that contain the argument passed in the constructor: ContainsFilter(String word, boolean ignoreCase)
- **EndsWithFilter** - Matches terms that end with the argument passed in the constructor: EndsWithFilter(String word, boolean ignoreCase)
- **ExactMatchFilter** - Matches terms that are exactly the same as the argument passed in the constructor: ExactMatchFilter(String word, boolean ignoreCase)
- **RegexFilter** - Matches terms that match the regular expression string passed in the constructor: RegexFilter(String regex, boolean ignoreCase)
- **SimilarFilter** - Matches terms that are similar to the term passed in the constructor, with a maximum distance also specified in the constructor: SimilarFilter(String word, boolean ignoreCase, int maxDistance) (this class uses the Levenshtein algorithm to compute the distance)
- **SoundFilter** - Matches terms that sound like the term passed in the constructor: SoundFilter(String word, boolean ignoreCase)
- **StartsWithFilter** - Matches terms that start with the argument passed in the constructor: StartsWithFilter(String word, boolean ignoreCase)
- **WildcardFilter** - Matches terms that match the wildcard pattern passed in the constructor: WildcardFilter(String word, boolean ignoreCase) (this class uses the Wildcard code; see that page for more info)

$pointer - types$

The *pointer_symbol* s for nouns are:

!	Antonym
@	Hypernym
@i	Instance Hypernym
~	Hyponym
~i	Instance Hyponym
#m	Member holonym
#s	Substance holonym
#p	Part holonym
%m	Member meronym
%s	Substance meronym
%p	Part meronym
=	Attribute
+	Derivationally related form
;c	Domain of synset - TOPIC
-c	Member of this domain - TOPIC
;r	Domain of synset - REGION
-r	Member of this domain - REGION
;u	Domain of synset - USAGE
-u	Member of this domain - USAGE

The *pointer_symbol* s for verbs are:

!	Antonym
@	Hypernym
~	Hyponym
*	Entailment
>	Cause
^	Also see
$	Verb Group
+	Derivationally related form
;c	Domain of synset - TOPIC
;r	Domain of synset - REGION
;u	Domain of synset - USAGE

The *pointer_symbol* s for adjectives are:

!	Antonym
&	Similar to
<	Participle of verb
\	Pertainym (pertains to noun)
=	Attribute
^	Also see
;c	Domain of synset - TOPIC
;r	Domain of synset - REGION
;u	Domain of synset - USAGE

The *pointer_symbol* s for adverbs are:

> **!** Antonym
> **** Derived from adjective
> **;c** Domain of synset - TOPIC
> **;r** Domain of synset - REGION
> **;u** Domain of synset - USAGE

RKEA

Samstag, 25. April 2020 13:17

http://community.nzdl.org/kea/Download/Kea-5.0-Readme.txt

- Ist ein tool, um automatisch keyphrases von Text-Dokumenten rauszuziehen
- Könnte sehr spannend sein, zu versuchen die Themen von Liedern über RKEA zu trainieren und automatisch auf Keyphrases klassifizieren zu lassen - also welches Thema behandelt das Lied, nur von den Lyrics aus - wäre sicher eine gute Variable für's Modell, aber könnte sehr viel Aufwand sein
    - Wäre aber eher eine eigene Fragestellung, ist glaub ich zu viel aufwand, aber ich wollts mal aufschreiben, weil ich das Prinzip ganz cool finde

Nicht geeignete Packages

Montag, 30. März 2020 18:39

Package	Wieso nicht geeignet?
tau	Ungeeignete Funktionen
languageR	Zu detailierte Informationen zu Lyrik und nicht für normale Texte
zipfR	Auch zu spezifisch für Wortanalysen, nicht für texte geeignet
Mscstexta4r - R-Client für Microsoft Cognitive Services Text Analytics	Wäre super für Sentiment-Analysen, aber nur 5000 Anfragen wären gratis und wir haben deutlich mehr für alle Lieder
openNLP	Funktionen dienen eher zum summarisen von Sätzen oder zum Tokenizen - bringt uns für unsere Fragestellungen ziemlich garnix OpenNLP selbst dürfte extrem leistungsstark sein, aber die Packages, die bereits existieren haben für uns keine sinnvollen Funktionen
monkeylearn	Gleiches Problem wie Mscstexta4r - wäre super, aber nicht kostenlos -> Aber: es erklärt Sachen gut! siehe: https://monkeylearn.com/sentiment-analysis/ -> haben auch noch mehr Theorie
gsubfn	Bietet die Möglichkeit pattern in einem Text zu extrahieren und durch das Ergebnis einer selbst eingegebenen Funktion zu extrahieren -> nicht passend für unsere Fragestelllung
textreuse	Gibt die Möglichkeit zu vergleichen, ob Texte sehr ähnlich sind **könnte relevant sein, kommt drauf an, was wir genau machen wollen** - wenn wir sagen wollen "populäre Lieder haben ähnliche Texte" könnt es nützlich sein - wäre aber eine riesen-Matrix und halte ich für nicht sinnvoll, mit jedem Lied einen Ähnlichkeitsindex zu bestimmen

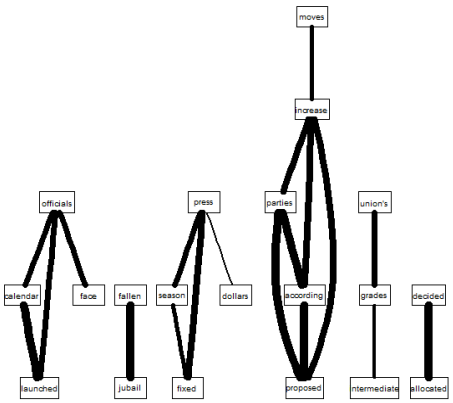
tm - Text Mining Package

Samstag, 25. April 2020 13:52

Ist ein Framework für Text Mining-Anwendungen in R

https://cran.r-project.org/web/packages/tm/tm.pdf

Interessante Funktionen:

- Parallellisiertes lapply
  - $tm\_parLapply(X, FUN, \dots)$
    - *X:* Ein Vektor (atomic oder list), der für die verwendete Engine funktioniert
    - FUN: Funktion, die auf jedes Element von X angewandt werden soll
  - $tm\_parLapply\_engine(new)$
    - *new:* Object, welches von $makeCluster()$ vom package *parallel* erstellt wurde
    - Verwendet, um die Parallelisierungsengine zu bekommen oder zu setzen
    - Wird dies nicht definiert, dann führt tm_parLapply einfach nur lapply aus
  - ```
    cores = parallel::detectCores()
    cl = parallel::makePSOCKcluster(names = cores-1)
    tm_parLapply_engine(cl)
    parallel::stopCluster(cl)
    ```
    Hier eine beliebige tm_parLaplly(X,FUN) einfügen
- $inspect(x)$
  - Gibt Informationen über die Dokumente in einem Pcorpus, Vcorpus, TermDocumentMatrix oder TextDocument aus
  - ```
    > inspect(test)
    <<TermDocumentMatrix (terms: 1266, documents: 20)>>
    Non-/sparse entries: 2255/23065
    Sparsity           : 91%
    Maximal term length: 17
    Weighting          : term frequency (tf)
    Sample             :
            Docs
    Terms    144 236 237 242 246 248 273 489 502 704
      and      9   7  11   3   9   6   5   5   6   5
      for      5   4   3   1   6   2   4   4   5   3
      its      6   8   3   0   3   2   0   2   2   1
      mln      4   4   1   0   0   3   9   2   2   0
      oil     11   7   3   3   4   9   5   4   4   3
      opec    10   6   1   2   1   6   5   0   0   0
      prices   3   2   0   1   0   7   4   2   2   2
      said     9   6   0   3   4   5   5   2   2   3
      that    10   4   1   0   2   2   0   1   1   3
      the     17  15  30   6  18  27  21   8  13  21
    ```
- $meta(x)$ - gibt Metadaten von einem Corpus oder Textdokument aus
  -> Anmerkung: In Corpi sind metadaten in Listen gespeichert, also immer mit $data[[1]]$
  - ```
    meta(crude[[1]])
    author       : character(0)
    datetimestamp: 1987-02-26 17:00:56
    description  :
    heading      : DIAMOND SHAMROCK (DIA) CUTS CRUDE PRICES
    id           : 127
    language     : en
    origin       : Reuters-21578 XML
    topics       : YES
    lewissplit   : TRAIN
    cgisplit     : TRAINING-SET
    oldid        : 5670
    places       : usa
    people       : character(0)
    orgs         : character(0)
    exchanges    : character(0)
    ```
  -
- $plot(x)$ - wird verwendet um TermDocumentMatrices zu plotten (braucht Package $Rgraphviz$)
  - ```
    plot(test, corThreshold = .2, weighting = T)
    ```
  - 

- $removeNumbers()$
  - Entfernt Zahlen aus einem Textdokument
- $removePunctuation()$
  - Entfernt Satzzeichen aus einem Textdokument
- $stemDocument()$
  - Führt Wortstemming auf einen Character Vektor oder ein Text-Dokument durch

```
> data$text[1]
[1] "Look at her face, it's a wonderful face  \nAnd it means something special to me  \nLook at the way that she smiles when she sees me  \nHow lucky can one fellow
 be?  \n  \nShe's just my kind of girl, she makes me feel fine  \nWho could ever believe that she could be mine?  \nShe's just my kind of girl, without her I'm blue
  \nAnd if she ever leaves me what could I do, what could I do?  \n  \nAnd when we go for a walk in the park  \nAnd she holds me and squeezes my hand  \nWe'll go on
 walking for hours and talking  \nAbout all the things that we plan  \n  \nShe's just my kind of girl, she makes me feel fine  \nWho could ever believe that she cou
ld be mine?  \nShe's just my kind of girl, without her I'm blue  \nAnd if she ever leaves me what could I do, what could I do?\n\n"
> stemDocument(data$text[1])
[1] "Look at her face, it a wonder face And it mean someth special to me Look at the way that she smile when she see me How lucki can one fellow be? She just my kin
d of girl, she make me feel fine Who could ever believ that she could be mine? She just my kind of girl, without her I'm blue And if she ever leav me what could I d
o, what could I do? And when we go for a walk in the park And she hold me and squeez my hand We'll go on walk for hour and talk About all the thing that we plan She
 just my kind of girl, she make me feel fine Who could ever believ that she could be mine? She just my kind of girl, without her I'm blue And if she ever leav me wh
at could I do, what could I do?"
>
```

Bin echt kein Fan von stemming bei Liedtexten ^^

- $stopwords(kind = \text{en})$
  - Gibt verschiedene Stoppwörter aus dem englischen zurück
- $stripWhitespace()$
  - Entfernt Lleerzeichen aus einem Textdokument oder Charakter-Vektor

1]]

Sonntag, 26. April 2020 10:56

quanteda

Sonntag, 26. April 2020 11:44

Problem: Wir hantieren an Lyrics, da ist die Punctuation oft einfach nicht gegeben

$nsentence(x)$ - gibt aus, wie viele Sätze ein Textdokument hat.

- Bei unseren Lyrics problematisch, da keine Punkte - Fragezeichen ja, rest nein

$nsyllable(x)$ - wie viele Silben hat ein Satz? (Ist ein Wort unbekannt, werden Vokalcluster gezählt und eine Schätzung abgelegt)

```
> data$text[1]
[1] "Look at her face, it's a wonderful face  \nAnd it means something special to me  \nLook at the way that she smiles when she sees me  \nHow lucky can one fell
ow be?  \n  \nShe's just my kind of girl, she makes me feel fine  \nWho could ever believe that she could be mine?  \nShe's just my kind of girl, without her I'm
blue  \nAnd if she ever leaves me what could I do, what could I do?  \n  \nAnd when we go for a walk in the park  \nAnd she holds me and squeezes my hand  \nWe'll
 go on walking for hours and talking  \nAbout all the things that we plan  \n  \nShe's just my kind of girl, she makes me feel fine  \nWho could ever believe that
 she could be mine?  \nShe's just my kind of girl, without her I'm blue  \nAnd if she ever leaves me what could I do, what could I do?\n\n"
> nsyllable(data$text[1])
[1] 186
> data$text[2]
[1] "Take it easy with me, please  \nTouch me gently like a summer evening breeze  \nTake your time, make it slow  \nAndante, Andante  \nJust let the feeling grow
  \n  \nMake your fingers soft and light  \nLet your body be the velvet of the night  \nTouch my soul, you know how  \nAndante, Andante  \nGo slowly with me now
\n  \nI'm your music  \n(I am your music and I am your song)  \nI'm your song  \n(I am your music and I am your song)  \nPlay me time and time again and make me s
trong  \n(Play me again 'cause you're making me strong)  \nMake me sing, make me sound  \n(You make me sing and you make me)  \nAndante, Andante  \nTread lightly
on my ground  \nAndante, Andante  \nOh please don't let me down  \n  \nThere's a shimmer in your eyes  \nLike the feeling of a thousand butterflies  \nPlease don'
t talk, go on, play  \nAndante, Andante  \nAnd let me float away  \n  \nI'm your music  \n(I am your music and I am your song)  \nI'm your song  \n(I am your musi
c and I am your song)  \nPlay me time and time again and make me strong  \n(Play me again 'cause you're making me strong)  \nMake me sing, make me sound  \n(You m
ake me sing and you make me)  \nAndante, Andante  \nTread lightly on my ground  \nAndante, Andante  \nOh please don't let me down  \n  \nMake me sing, make me sou
nd  \n(You make me sing and you make me)  \nAndante, Andante  \nTread lightly on my ground  \nAndante, Andante  \nOh please don't let me down  \nAndante, Andante
 \nOh please don't let me down\n\n"
> nsyllable(data$text[2])
[1] 368
```

$textstat\_readability(x, measure = \text{Flesch})$

- Berechnet einen readability-Index oder reading-ease Maßnahmen, je nach measure für die eingegebenen Charakter-Vektoren
- Für die verschiedenen Measures siehe https://cran.r-project.org/web/packages/quanteda/quanteda.pdf Seite 98
- Beispiele:
  - Nw = n_w = number of words
  - Nc = n_c = number of characters
  - Nst = n_{st} = number of sentences
  - Nsy = n_{sy} = number of syllables
  - Nwf = n_{wf} = number of words matching the Dale-Chall List of 3000 "familiar words"
  - ASL = Average Sentence Length: number of words / number of sentences
  - AWL = Average Word Length: number of characters / number of words
  - AFW = Average Familiar Words: count of words matching the Dale-Chall list of 3000 "familiar words" / number of all words
  - Nwd = n_{wd} = number of "difficult" words not matching the Dale-Chall list of "familiar" words
  - "Flesch": Flesch's Reading Ease Score (Flesch 1948).

$$206.835 - (1.015 \times ASL) - (84.6 \times \frac{n_{sy}}{n_w})$$

    - Je niedriger, desto schwieriger zu lesen
    - Achtung: Müssen aber vorher gecleaned werden, da Punktuation hier eine große Rolle spielt (da ASL - Average Sentence Length!)

```
> textstat_readability(data$text)
   document       Flesch
1     text1   80.6701176
2     text2 -164.4419231
3     text3 -209.0873077
4     text4 -148.1942857
5     text5 -274.4110249
6     text6 -208.0315789
7     text7 -202.1875000
8     text8  -99.9662871
9     text9  -73.6286585
10   text10   79.4450204
11   text11 -130.3837931
```

reading-ease score für die Lyrics in unserem File

  - "Flesch.Kincaid": Flesch-Kincaid Readability Score (Flesch and Kincaid 1975).

$$0.39 \times ASL + 11.8 \times \frac{n_{sy}}{n_w} - 15.59$$

    - Je höher, desto schwieriger zu lesen (= mehr Bildungsjahre nötig zum verstehen)

```
textstat_readability(data$text[1:11], measure = "Flesch.Kincaid")
 document Flesch.Kincaid
    text1       9.609359
    text2     100.786923
    text3     119.932308
    text4      94.794286
    text5     141.216620
    text6     117.797632
    text7     116.982500
    text8      77.385050
    text9      64.271220
0   text10      9.482122
1   text11     89.080517
```

FK-readability score für unsere Lyrics

Qdap

Sonntag, 26. April 2020 12:09

https://cran.r-project.org/web/packages/qdap/qdap.pdf

$add\_incomplete(text.var, endmarks, silent = F)$

- Würde fehlende Satzendmarks ergänzen
- Problem: Er findet sie nicht in den Lyrics, da es ja keine vollständigen Sätze sind

```
> add_incomplete(stripWhitespace(data$text[1]))
[1] "Look at her face, it's a wonderful face And it means something special to me Look at the way that she smiles when she sees me How lucky can one fellow be? s
e's just my kind of girl, she makes me feel fine Who could ever believe that she could be mine? She's just my kind of girl, without her I'm blue And if she ever
eaves me what could I do, what could I do? And when we go for a walk in the park And she holds me and squeezes my hand We'll go on walking for hours and talking
bout all the things that we plan She's just my kind of girl, she makes me feel fine Who could ever believe that she could be mine? She's just my kind of girl, wi
hout her I'm blue And if she ever leaves me what could I do, what could I do?"
```

```
==========================================================================================
The following elements were missing endmarks (`|` added):


character(0)
```

$automated\_readability\_index(text.var, grouping.var)$

- *Alternativ:*
  - automated_readability_index - Apply Automated Readability Index to transcript(s) by zero or more grouping variable(s).
  - coleman_liau - Apply Coleman Liau Index to transcript(s) by zero or more grouping variable(s).
  - SMOG - Apply SMOG Readability to transcript(s) by zero or more grouping variable(s).
  - flesch_kincaid - Flesch-Kincaid Readability to transcript(s) by zero or more grouping variable(s).
  - fry - Apply Fry Readability to transcript(s) by zero or more grouping variable(s).
  - linsear_write - Apply Linsear Write Readability to transcript(s) by zero or more grouping variable(s).
- Führt automatisch Readability auf eine textvariable aus, gruppiert nach einer optionalen grouping-Variable.
- Beispiel:

```
> automated_readability_index(data$text[1:5], grouping.var = data$song_name[1:5])
                1:5 word.count sentence.count character.count Automated_Readability_Index
1      Andante, Andante        260              1            1030                     127.229
2        As Good As New        312              1            1067                     150.678
3      Bang-A-Boomerang        198              1             924                      99.550
4             Cassandra        361              1            1527                     178.993
5 She's My Kind Of Girl        153              1             556                      72.186
```

$sentSplit(df, text.var)$

- Splitted Sätze an ihren endmarks
- Müsste aber bei unseren lyrics schon gecleaned sein, damit das funktioniert
  - In Kombination mit $add\_incomplete()$ vielleicht möglich
  - $sent\_detect()$ könnte auch hilfreich sein (aber man sieht, ohne richtige Satzzeichen problematisch.)

```
> sent_detect(text.var = data$text[1])
[1] "Look at her face, it's a wonderful face And it means something special to me Look at the way that she smiles when she sees me How lucky can one fellow be?"

[2] "She's just my kind of girl, she makes me feel fine Who could ever believe that she could be mine?"

[3] "She's just my kind of girl, without her I'm blue And if she ever leaves me what could I do, what could I do?"

[4] "And when we go for a walk in the park And she holds me and squeezes my hand We'll go on walking for hours and talking About all the things that we plan She
 just my kind of girl, she makes me feel fine Who could ever believe that she could be mine?"
[5] "She's just my kind of girl, without her I'm blue And if she ever leaves me what could I do, what could I do?"
```

bag_o_words(text.var) - verwandelt einen Satz in einen Bag-of-words-Vektor

- Macht aber das gleiche, wie die $hunspell\_parse()$-Funktion

```
> bag_o_words(data$text[1])
  [1] "look"     "at"       "her"      "face"     "it's"     "a"        "wonderful" "face"     "and"      "it"       "means"    "something"
 [13] "special"  "to"       "me"       "look"     "at"       "the"      "way"       "that"     "she"      "smiles"   "when"     "she"
 [25] "sees"     "me"       "how"      "lucky"    "can"      "one"      "fellow"    "be"       "she's"    "just"     "my"       "kind"
 [37] "of"       "girl"     "she"      "makes"    "me"       "feel"     "fine"      "who"      "could"    "ever"     "believe"  "that"
 [49] "she"      "could"    "be"       "mine"     "she's"    "just"     "my"        "kind"     "of"       "girl"     "without"  "her"
 [61] "i'm"      "blue"     "and"      "if"       "she"      "ever"     "leaves"    "me"       "what"     "could"    "i"        "do"
 [73] "what"     "could"    "i"        "do"       "and"      "when"     "we"        "go"       "for"      "a"        "walk"     "in"
 [85] "the"      "park"     "and"      "she"      "holds"    "me"       "and"       "squeezes" "my"       "hand"     "we'll"    "go"
 [97] "on"       "walking"  "for"      "hours"    "and"      "talking"  "about"     "all"      "the"      "things"   "that"     "we"
[109] "plan"     "she's"    "just"     "my"       "kind"     "of"       "girl"      "she"      "makes"    "me"       "feel"     "fine"
[121] "who"      "could"    "ever"     "believe"  "that"     "she"      "could"     "be"       "mine"     "she's"    "just"     "my"
[133] "kind"     "of"       "girl"     "without"  "her"      "i'm"      "blue"      "and"      "if"       "she"      "ever"     "leaves"
[145] "me"       "what"     "could"    "i"        "do"       "what"     "could"     "i"        "do"
```

$blank2NA(dataframe, msissing = NA)$

- Wandelt blanks zu NAs um

$bracketX(text.var)$

- Entfernt alle Klammern aus einem Text

```
> bracketX("Das ist (ein Test)")
[1] "Das ist"
```

$genX(text.var, left, right)$

- Generalisierte Form von bracketX
- Left: Vektor von Charakteren oder numerischen Symbolen als Anker, ab wo gestartet werden soll etwas zu entfernen
- Right: Vektor von Charakteren oder numerischen Symbolen als Anker, wo der Endpunkt liegt, vor dem entfernt werden soll

$check\_spelling(text.var, range = 2, n.suggests = 8, \dots)$

- Checkt das Spelling für einen Vektor von Strings
  - Werden geparsed
  - In einem Wörterbuch nachgesehen
  - Wurde ein Wort nicht gefunden, wird nach Ersetzungen gesucht
    - Sehr ähnliche Worte werden gesucht
    - Mit stringdist wird verglichen, welche die ähnlichsten Worte sind
    - Die $n.suggests$-Werte werden als Ersetzungen vorgeschlagen

$check\_spelling\_interactive()$

- Möglichkeit, um die replacements selbst interaktiv auszuwählen.

$check\_text(text.var)$ - wirft aus, wo potenzielle Probleme in einem Text auftreten können und wie man damit umgehen kann

- Problem: Manche Probleme nicht behebbar bei unseren Lyrics (z.B. das Sätze nicht vollständig sind und $add\_incomplete$ nicht weiß, wo die Punktuation hingehören würde)

```
> summary(check_text(data$text[1]))
                          Length Class  Mode
non_character             0      -none- NULL
missing_ending_punctuation 1      -none- numeric
empty                     0      -none- NULL
double_punctuation        1      -none- numeric
non_space_after_comma     0      -none- NULL
no_alpha                  0      -none- NULL
non_ascii                 0      -none- NULL
missing_value             0      -none- NULL
containing_escaped        0      -none- NULL
containing_digits         0      -none- NULL
indicating_incomplete     0      -none- NULL
potentially_misspelled    0      -none- NULL
```

chunker(*text.var*, *grouping.var*, *n.words*, *n.chunks*)

- Teilt einzelne Textspalten Anhand einer (optionalen) grouping-variable in Textlisten auf, in denen Punktuation entfernt wurde
- Anwendungsbeispiel:

```
> chunker(text.var = data$text[1], grouping.var = data$song_name[1], n.chunks = 1)
$`She's My Kind Of Girl`
$`She's My Kind Of Girl`$`1`
[1] "look at her face it's a wonderful face and it means something special to me look at the way that she smiles when she sees me how lucky can one fellow be she'
s just my kind of girl she makes me feel fine who could ever believe that she could be mine she's just my kind of girl without her i'm blue and if she ever leaves
 me what could i do what could i do and when we go for a walk in the park and she holds me and squeezes my hand we'll go on walking for hours and talking about al
l the things that we plan she's just my kind of girl she makes me feel fine who could ever believe that she could be mine she's just my kind of girl without her i
'm blue and if she ever leaves me what could i do what could i do"
```

clean(*text.var*)

- Entfernt escape-Charaktere aus einem Text
- Beispiel:

```
> data$text[1]
[1] "Look at her face, it's a wonderful face  \nAnd it means something special to me  \nLook at the way that she smiles when she sees me  \nHow lucky can one fell
ow be?  \n  \nShe's just my kind of girl, she makes me feel fine  \nWho could ever believe that she could be mine?  \nShe's just my kind of girl, without her I'm
blue  \nAnd if she ever leaves me what could I do, what could I do?  \n  \nAnd when we go for a walk in the park  \nAnd she holds me and squeezes my hand  \nWe'll
 go on walking for hours and talking  \nAbout all the things that we plan  \n  \nShe's just my kind of girl, she makes me feel fine  \nWho could ever believe that
 she could be mine?  \nShe's just my kind of girl, without her I'm blue  \nAnd if she ever leaves me what could I do, what could I do?\n\n"
> clean(data$text[1])
[1] "Look at her face, it's a wonderful face And it means something special to me Look at the way that she smiles when she sees me How lucky can one fellow be? Sh
e's just my kind of girl, she makes me feel fine Who could ever believe that she could be mine? She's just my kind of girl, without her I'm blue And if she ever l
eaves me what could I do, what could I do? And when we go for a walk in the park And she holds me and squeezes my hand We'll go on walking for hours and talking A
bout all the things that we plan She's just my kind of girl, she makes me feel fine Who could ever believe that she could be mine? She's just my kind of girl, wit
hout her I'm blue And if she ever leaves me what could I do, what could I do? "
```

comma_spacer(*text.var*)

- Fügt ein Leerzeichen nach einem Komma ein, damit bei Strip-Funktionen nicht Wörter wie *one,two zu onetwo anstatt one two* gemacht werden

condense(*dataframe*, *sep* = ,)

- Kondensiert Dataframe-Spalten, die aus Listen von Vektoren bestehen in einen einzelnen String-Vektor
- Fügt also alle Spalten, die Lists in Vektoren sind in eine Spalte und hängt alle hintereinander

counts(*x*)

- Greift auf die count dataframes von *qdap*-Outputs zu
- Beispiel:

```
DATA.SPLIT
     person  tot TOT sex adult code                    state               stem.text
        sam  1.1   1   m     0   K1          Computer is fun.          Comput is fun.
        sam  1.2   1   m     0   K1               Not too fun.            Not too fun.
       greg  2.1   2   m     0   K2     No it's not, it's dumb.       No it not it dumb.
    teacher  3.1   3   m     1   K3          What should we do?       What should we do?
        sam  4.1   4   m     0   K4         You liar, it stinks!       You liar it stink!
       greg  5.1   5   m     0   K5     I am telling the truth!     I am tell the truth!
      sally  6.1   6   f     0   K6      How can we be certain?   How can we be certain?
       greg  7.1   7   m     0   K7           There is no way.         There is no way.
        sam  8.1   8   m     0   K8            I distrust you.          I distrust you.
      sally  9.1   9   f     0   K9 What are you talking about? What are you talk about?
 researcher 10.1  10   f     1  K10           Shall we move on?        Shall we move on?
 researcher 10.2  10   f     1  K10                 Good then.               Good then.
       greg 11.1  11   m     0  K11                I'm hungry.              I'm hungri.
       greg 11.2  11   m     0  K11                 Let's eat.                 Let eat.
       greg 11.3  11   m     0  K11                You already?              You alreadi?
poldat <- with(DATA.SPLIT, polarity(state, person))
poldat
     person total.sentences total.words ave.polarity sd.polarity stan.mean.polarity
       greg               6          20       -0.075       0.183             -0.408
 researcher               2           6        0.354       0.500              0.707
      sally               2          10        0.000       0.000                NaN
        sam               4          13       -0.394       0.678             -0.582
    teacher               1           4        0.000          NA                 NA
counts(poldat)
     person wc polarity pos.words    neg.words                   text.var
        sam  3    0.577       fun            -           Computer is fun.
        sam  3   -0.577       fun            -                Not too fun.
       greg  5   -0.447         -         dumb     No it's not, it's dumb.
    teacher  4    0.000         -            -          What should we do?
        sam  4   -1.000         - liar, stinks        You liar, it stinks!
       greg  5    0.000         -            -     I am telling the truth!
      sally  5    0.000         -            -      How can we be certain?
       greg  4    0.000         -            -           There is no way.
        sam  3   -0.577         -     distrust            I distrust you.
      sally  5    0.000         -            - What are you talking about?
 researcher  4    0.000         -            -           Shall we move on?
 researcher  2    0.707      good            -                 Good then.
       greg  2    0.000         -            -                I'm hungry.
       greg  2    0.000         -            -                 Let's eat.
       greg  2    0.000         -            -               You already?
```

formality(*text.var*, *grouping.var*)

- Gibt aus, wie formal ein Text geschrieben ist
- Anwendungsbeispiel:

```
> formality(data$text[1])
  all word.count formality
1 all        161     38.51
```

freq_terms(*text.var*, *top*)

- Gibt die häufigsten Worte in einem Textvektor aus

htruncdf(*df*)

- Printed einen data.frame, der Spalten hat, wo sehr viel Text drin ist, in einer schönen Form
- Gibt es auch für andere Datenformate oder ganze Datensets truncdf, ltruncdf, usw.
- Beispiel an unseren Daten: Man merke, wie schön der Text abgeschnitten ist :P

```
htruncdf(data)
 artist       song  song_name    song_id       text  artist_id popularity duration_m year artist_gen artist_pop duration_s
   ABBA Ahe's My K She's My K 5Ca2lylmEM Look at he OLcJLqbBma         19     164586 1973 europop, s         79        165
   ABBA Andante, A Andante, A 1YnBV85uTT Take it ea OLcJLqbBma          4     279000 1999 europop, s         79        279
   ABBA As Good As As Good As 2bOU6wQfK5 I'll never OLcJLqbBma         33     204693 1979 europop, s         79        205
   ABBA Bang-A-Boo Bang-A-Boo 6di98QRVO1 Making som OLcJLqbBma          2     184933 1999 europop, s         79        185
   ABBA  Cassandra  Cassandra 45PCC9Q70n Down in th OLcJLqbBma         20     296773 1981 europop, s         79        297
   ABBA Chiquitita Chiquitita 4g5LUzHFRr Chiquitita OLcJLqbBma         56     327040 1979 europop, s         79        327
   ABBA Crazy Worl Crazy Worl 58646QOETW I was out  OLcJLqbBma         33     226320 1975 europop, s         79        226
   ABBA Dancing Qu Dancing Qu OGjEhVFGZW You can da OLcJLqbBma         75     230400 1976 europop, s         79        230
   ABBA Disillusio Disillusio 5kphQdMWF5 Changing,  OLcJLqbBma         21     183306 1973 europop, s         79        183
0  ABBA Does Your  Does Your  OeTqauier0 You're so  OLcJLqbBma         19     249733 1986 europop, s         79        250
```

- ***qview***$(df, \dots)$ ist die noch schönere Alternative!

```
> qview(data)
======================================================================
nrow =  36718          ncol =  12          data
======================================================================
   artist       song  song_name    song_id       text  artist_id popularity duration_m year artist_gen artist_pop duration_s
1    ABBA Ahe's My K She's My K 5Ca2lylmEM Look at he OLcJLqbBma         19     164586 1973 europop, s         79        165
2    ABBA Andante, A Andante, A 1YnBV85uTT Take it ea OLcJLqbBma          4     279000 1999 europop, s         79        279
3    ABBA As Good As As Good As 2bOU6wQfK5 I'll never OLcJLqbBma         33     204693 1979 europop, s         79        205
4    ABBA Bang-A-Boo Bang-A-Boo 6di98QRVO1 Making som OLcJLqbBma          2     184933 1999 europop, s         79        185
5    ABBA  Cassandra  Cassandra 45PCC9Q70n Down in th OLcJLqbBma         20     296773 1981 europop, s         79        297
6    ABBA Chiquitita Chiquitita 4g5LUzHFRr Chiquitita OLcJLqbBma         56     327040 1979 europop, s         79        327
7    ABBA Crazy Worl Crazy Worl 58646QOETW I was out  OLcJLqbBma         33     226320 1975 europop, s         79        226
8    ABBA Dancing Qu Dancing Qu OGjEhVFGZW You can da OLcJLqbBma         75     230400 1976 europop, s         79        230
9    ABBA Disillusio Disillusio 5kphQdMWF5 Changing,  OLcJLqbBma         21     183306 1973 europop, s         79        183
10   ABBA Does Your  Does Your  OeTqauier0 You're so  OLcJLqbBma         19     249733 1986 europop, s         79        250
>
```

lexical_classification$(text.var, grouping.var)$
> klassifiziert Wörter als "content" (Nomen, Verben, Adjektive, Adverben) oder "functional"
(mehr die Wörter, die Sätze zusammenhalten) (grouping.var könnten wieder song_namesein)
> gibt dann einen Prozentwert aus, wie viele der Worte "content" sind und welche das sind
>

```
lexical_classification(sentSplit(a, "text")$text, grouping.var = sentSplit(a, "text")$song_name)
          song_name word.count ave.content.rate   SE n.content n.functional content                    functional
She's My Kind Of Girl        153            28.8% 3.67        44          109 look, face, wonderful,... at, her, it's, a, and,...
```

(Anmerkung: Vielleicht kann man über den Content dann eine Topic-Analyse laufen lassen, was genau behandelt wird?)

multigsub()
> funktioniert wie *gsub*, nur mit mehreren Pattern und Replacements gleichzeitig

NAer$(x, replace = 0)$
> ersetzt Nas in einem Dataframe mit den replace-Value

name2sex()
> versucht das Geschlecht von Personen über den Vornamen zu erkennen

Network()
> wird verwendet, um plots für gewisse Funktionen (formality, lexical_classification, polarity) zu erstellen (siehe Seite 174 - 178 in https://cran.r-project.org/web/packages/qdap/qdap.pdf)

qdap_df$(dataframe, text.var)$
> qdap_df ist eine Möglichkeit um in einem Dataframe zu spezifizieren, welche Variable die text.var ist
> dann kann mit dem Pipe-Operator %&% (vgl. Dplyr) immer die text.var weggelassen werden
Beispiel:

```
dat <- qdap_df(DATA, state)
dat %&% trans_cloud(grouping.var=person)
```

qprep$(text.var, \dots)$

- Wrapper-Funktion für *bracketX*, *replace_number*, *replace_symbol*, *replace_abbreviation* & *scrubber* um schnell Daten für die Analyse aufzubereiten

```
> data$text[1]
[1] "Look at her face, it's a wonderful face  \nAnd it means something special to me  \nLook at the way that she smiles when she sees me  \nHow lucky can one fell
ow be?  \n  \nShe's just my kind of girl, she makes me feel fine  \nWho could ever believe that she could be mine?  \nShe's just my kind of girl, without her I'm
blue  \nAnd if she ever leaves me what could I do, what could I do?  \n  \nAnd when we go for a walk in the park  \nAnd she holds me and squeezes my hand  \nWe'll
 go on walking for hours and talking  \nAbout all the things that we plan  \n  \nShe's just my kind of girl, she makes me feel fine  \nWho could ever believe that
 she could be mine?  \nShe's just my kind of girl, without her I'm blue  \nAnd if she ever leaves me what could I do, what could I do?\n\n"
> qprep(data$text[1])
[1] "Look at her face, it's a wonderful face And it means something special to me Look at the way that she smiles when she sees me How lucky can one fellow be? Sh
e's just my kind of girl, she makes me feel fine Who could ever believe that she could be mine? She's just my kind of girl, without her I'm blue And if she ever l
eaves me what could I do, what could I do? And when we go for a walk in the park And she holds me and squeezes my hand We'll go on walking for hours and talking A
bout all the things that we plan She's just my kind of girl, she makes me feel fine Who could ever believe that she could be mine? She's just my kind of girl, wit
hout her I'm blue And if she ever leaves me what could I do, what could I do?"
```

replace_abbreviation$(text.var)$

- Abkürzungen werden zu ganzen Wörtern

```
> replace_abbreviation("dr.")
[1] "Doctor."
```

-> gibt es auch für contraction (z.B. why's wird zu why is), numbers (Zahlen werden zu Worten), ordinal (z.B. 1st wird zu first) und Symbolen (z.B. € wird Euro, etc.)

rm_stopwords$(text.var)$

- Entfernt Stopworte

scores(x)

- Extrahiert aus einem qdap-Element, wie *automoted_readability_index* die Score-Werte

strip(x)

- Entfernt nicht gewollte Charaktere aus dem Text
- Zahlen, Beistriche, etc.

word_count$(text.var)$

- Zählt die Anzahl an Worten in einem Textvektor

```
> word_count(data$text[1])
[1] 153
```

character_count$(text.var)$

- Zählt Anzahl an Buchstaben in einem Text-Vektor

```
> character_count(data$text[1])
[1] 556
```

word_list$(text.var, grouping.var)$

- Macht für jede Gruppe einen Frequency table mit den Worten und ihrer Anzahl

```
$`Bang-A-Boomerang`
              WORD FREQ
1                a   19
2               is   11
3    boomaboomerang   10
4             love   10
5             bang    8
```

```
$`Bang-A-Boomerang`
               WORD FREQ
1                 a   19
2                is   11
3    boomaboomerang   10
4              love   10
5              bang    8
6               you    7
7               and    5
8                so    5
9                it    4
10           return    4
```

word_stats(*text.var*, *grouping.var*, ...)

- Kann über andere Parameter noch eingestellt werden, welche man genau bekommen möchte

```
word_stats(text.var = data$text[1:10], grouping.var = data$song_name[1:10])
 1:10                  n.sent n.words n.char n.syl n.poly wps  cps sps psps   cpw   spw pspw p.hapax pspw n.hapax n.dis grow.rate prop.dis
 Cassandra                  1     361   1527   476     15 361 1527 476   15 4.230 1.319 .042      89 .042       89      8       .247      .022
 As Good As New             1     312   1067   363      9 312 1067 363    9 3.420 1.163 .029      48 .029       48     33       .154      .106
 Chiquitita                 1     304   1181   380     16 304 1181 380   16 3.885 1.250 .053      43 .053       43     36       .141      .118
 Crazy World                1     304   1103   353      5 304 1103 353    5 3.628 1.161 .016      85 .016       85     19       .280      .062
 Does Your Mother Know      1     294   1104   321      0 294 1104 321    0 3.755 1.092 .000      41 .000       41     16       .139      .054
 Andante, Andante           1     260   1030   246     21 260 1030 246   21 3.962  .946 .081      29 .081       29     11       .112      .042
 Dancing Queen              1     202    834   244      8 202  834 244    8 4.129 1.208 .040      38 .040       38     17       .188      .084
 Bang-A-Boomerang           1     198    924   305     22 198  924 305   22 4.667 1.540 .111      53 .111       53     18       .268      .091
 Disillusion                1     164    706   218     11 164  706 218   11 4.305 1.329 .067      46 .067       46     19       .280      .116
0 She's My Kind Of Girl     1     153    556   170      1 153  556 170    1 3.634 1.111 .007      30 .007       30     19       .196      .124

 sent.num word.count character.count syllable.count polysyllable.count char2word.ratio syl2word.ratio polysyl2word.ratio end.mark sent.type
        1        153             556            170                 1           3.634           1.111           0.007        \n        NA
        2        260            1030            246                21           3.962           0.946           0.081        \n        NA
        3        312            1067            363                 9           3.420           1.163           0.029        \n        NA
        4        198             924            305                22           4.667           1.540           0.111        \n        NA
        5        361            1527            476                15           4.230           1.319           0.042        \n        NA
        6        304            1181            380                16           3.885           1.250           0.053        \n        NA
        7        304            1103            353                 5           3.628           1.161           0.016        \n        NA
        8        202             834            244                 8           4.129           1.208           0.040        \n        NA
        9        164             706            218                11           4.305           1.329           0.067        \n        NA
0      10        294            1104            321                 0           3.755           1.092           0.000        \n        NA
```

```
word_stats(text.var = data$text[1:10], grouping.var = data$song_name[1:10])$sent.elem
 n.hapax  pspw p.hapax pspw.1
      89 0.042      89  0.042
      48 0.029      48  0.029
      43 0.053      43  0.053
      85 0.016      85  0.016
      41 0.000      41  0.000
      29 0.081      29  0.081
      38 0.040      38  0.040
      53 0.111      53  0.111
      46 0.067      46  0.067
)     30 0.007      30  0.007
```

Returns a list of three descriptive word statistics:

ts	A data frame of descriptive word statistics by row
gts	A data frame of word/sentence statistics per grouping variable: • n.tot - number of turns of talk • n.sent - number of sentences • n.words - number of words • n.char - number of characters • n.syl - number of syllables • n.poly - number of polysyllables • sptot - syllables per turn of talk • wptot - words per turn of talk • wps - words per sentence • cps - characters per sentence • sps - syllables per sentence • psps - poly-syllables per sentence • cpw - characters per word • spw - syllables per word • n.state - number of statements • n.quest - number of questions • n.exclm - number of exclamations • n.incom - number of incomplete statements • p.state - proportion of statements • p.quest - proportion of questions • p.exclm - proportion of exclamations • p.incom - proportion of incomplete statements • n.hapax - number of hapax legomenon • n.dis - number of dis legomenon • grow.rate - proportion of hapax legomenon to words • prop.dis - proportion of dis legomenon to words
mpun	An account of sentences with an improper/missing end mark
word.elem	A data frame with word element columns from gts
sent.elem	A data frame with sentence element columns from gts
omit	Counter of omitted sentences for internal use (only included if some rows contained missing values)
percent	The value of percent used for plotting purposes.
zero.replace	The value of zero.replace used for plotting purposes.
digits	integer value od number of digits to display; mostly internal use

Warning

It is assumed the user has run `sentSplit` on their data, otherwise some counts may not be accurate.