# Data Science: Visualisations in R - Assignment 6

Github of CCS Amsterdam

*dr. Mariken A.C.G. van der Velden*

*November 29, 2019*

## Instructions for the tutorial

Read the *entire* document that describes the assignment for this tutorial. Follow the indicated steps. Each assignment in the tutorial contains several exercises, divided into numbers and letters (1a, 1b, 1c, 2a, 2b, etc.).

Write down your answers with **the same number-letter combination** in a separate document (e.g., Google Doc), and submit this document to Canvas in a PDF format.

*Note*: this is the first time that this course has ever been taught. Because of rapidly changing techniques, the course tries to keep up/ reflect those changes. As a result, the assignments may contain errors or ambiguities. If in doubt, ask for clarification during the hall lectures or tutorials. All feedback on the assignments is greatly appreciated!

## Text Analysis with Quanteda

In this tutorial you will learn how to perform text analysis using the quanteda package. The quanteda package is an extensive text analysis suite for R. It covers everything you need to perform a variety of automatic text analysis techniques, and features clear and extensive documentation. Here we'll focus on the main preparatory steps for text analysis, and on learning how to browse the quanteda documentation. The documentation for each function can also be found here.

For a more detailed explanation of the steps discussed here, you can read the paper Text Analysis in R (Welbers, van Atteveldt & Benoit, 2017).

```r
library(quanteda)
library(tidyverse)
```

**Step 1: Importing text and creating a quanteda corpus**: The first step is getting text into R in a proper format. Text can be stored in a variety of formats, from plain text and CSV files to HTML and PDF, and with different 'encodings'. There are various packages for reading these file formats, and there is also the convenient readtext that is specialized for reading texts from a variety of formats. For this tutorial, we will be importing text from a csv. For convenience, we're using a csv that's available online, but the process is the same for a csv file on your own computer. The data consists of the State of the Union speeches of US presidents, with each document (i.e. row in the csv) being a paragraph. The data will be imported as a `data.frame`.

```r
library(readr)
url <- 'https://bit.ly/2QoqUQS'
(d <- read_csv(url))
```

```
## # A tibble: 23,469 x 5
##    paragraph date       President     Party text
##        <dbl> <date>     <chr>         <chr> <chr>
##  1         1 1790-01-08 George Washi~ Other I embrace with great satisfact~
##  2         2 1790-01-08 George Washi~ Other In resuming your consultations~
##  3         3 1790-01-08 George Washi~ Other Among the many interesting obj~
##  4         4 1790-01-08 George Washi~ Other A free people ought not only t~
##  5         5 1790-01-08 George Washi~ Other The proper establishment of th~
##  6         6 1790-01-08 George Washi~ Other There was reason to hope that ~
##  7         7 1790-01-08 George Washi~ Other The interests of the United St~
##  8         8 1790-01-08 George Washi~ Other Various considerations also re~
##  9         9 1790-01-08 George Washi~ Other Uniformity in the currency, we~
## 10        10 1790-01-08 George Washi~ Other The advancement of agriculture~
## # ... with 23,459 more rows
```

We can now create a quanteda corpus with the `corpus()` function. If you want to learn more about this function, recall that you can use the question mark to look at the documentation.

```
?corpus
```

Here you see that for a data.frame, we need to specify which column contains the text field. Also, the text column must be a character vector.

```
(corp <- corpus(d, text_field = 'text'))  ## create the corpus
```

```
## Corpus consisting of 23,469 documents and 4 docvars.
```

**Step 2: Creating the DTM (or DFM)**: Often, text analysis techniques only use the frequencies of words in documents. This is also called the bag-of-words assumption, because texts are then treated as bags of individual words. Despite ignoring much relevant information in the order of words and syntax, this approach has proven to be very powerfull and efficient.

The standard format for representing a bag-of-words is as a `document-term matrix` (DTM). This is a matrix in which rows are documents, columns are terms, and cells indicate how often each term occured in each document. We'll first create a small example DTM from a few lines of text. Here we use quanteda's `dfm()` function, which stands for `document-feature matrix` (DFM), which is a more general form of a DTM.

```
text <-  c(d1 = "Cats are awesome!",
           d2 = "We need more cats!",
           d3 = "This is a soliloquy about a cat.")

(dtm <- dfm(text, tolower=F))
```

```
## Document-feature matrix of: 3 documents, 15 features (64.4% sparse).
## 3 x 15 sparse Matrix of class "dfm"
##     features
## docs Cats are awesome ! We need more cats This is a soliloquy about cat .
##   d1    1   1       1 1  0    0    0    0    0  0 0         0     0   0 0
##   d2    0   0       0 1  1    1    1    1    0  0 0         0     0   0 0
##   d3    0   0       0 0  0    0    0    0    1  1 2         1     1   1 1
```

Here you see, for instance, that the observation (word) `soliloquy` only occurs in the third document. In this matrix format, we can perform calculations with texts, like analyzing different sentiments of frames regarding cats, or the computing the similarity between the third sentence and the first two sentences.

However, directly converting a text to a DTM is a bit crude. Note, for instance, that the words `Cats`, `cats`, and `cat` are given different columns. In this DTM, "Cats" and "awesome" are as different as "Cats" and "cats", but for many types of analysis we would be more interested in the fact that both texts are about felines, and not about the specific word that is used. Also, for performance it can be useful (or even necessary) to use fewer columns, and to ignore less interesting words such as `is` or very rare words such as `soliloquy`.

This can be achieved by using additional *preprocessing* steps. In the next example, we'll again create the DTM, but this time we make all text lowercase, ignore stopwords and punctuation, and perform *stemming*. Simply put, stemming removes some parts at the ends of words to ignore different forms of the same word, such as singular versus plural ("gun" or "gun-s") and different verb forms ("walk","walk-ing","walk-s")

```
(dtm <- dfm(text, tolower=T, remove = stopwords('en'), stem = T, remove_punct=T))
```

```
## Document-feature matrix of: 3 documents, 4 features (50.0% sparse).
## 3 x 4 sparse Matrix of class "dfm"
##     features
## docs cat awesom need soliloquy
##   d1   1      1    0         0
##   d2   1      0    1         0
##   d3   1      0    0         1
```

The `tolower` argument determines whether texts are (TRUE) or aren't (FALSE) converted to lowercase. `stem` determines whether stemming is (TRUE) or isn't (FALSE) used. The remove argument is a bit more tricky. If you look at the documentation for the dfm function (`?dfm`) you'll see that `remove` can be used to give "a pattern of user-supplied features to ignore". In this case, we actually used another function, `stopwords()`, to get a list of english stopwords. You can see for yourself.

```
stopwords('en')
```

```
##   [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"
##  [81] "isn't"      "aren't"     "wasn't"     "weren't"    "hasn't"
##  [86] "haven't"    "hadn't"     "doesn't"    "don't"      "didn't"
##  [91] "won't"      "wouldn't"   "shan't"     "shouldn't"  "can't"
##  [96] "cannot"     "couldn't"   "mustn't"    "let's"      "that's"
## [101] "who's"      "what's"     "here's"     "there's"    "when's"
## [106] "where's"    "why's"      "how's"      "a"          "an"
## [111] "the"        "and"        "but"        "if"         "or"
## [116] "because"    "as"         "until"      "while"      "of"
## [121] "at"         "by"         "for"        "with"       "about"
## [126] "against"    "between"    "into"       "through"    "during"
## [131] "before"     "after"      "above"      "below"      "to"
## [136] "from"       "up"         "down"       "in"         "out"
## [141] "on"         "off"        "over"       "under"      "again"
## [146] "further"    "then"       "once"       "here"       "there"
## [151] "when"       "where"      "why"        "how"        "all"
## [156] "any"        "both"       "each"       "few"        "more"
## [161] "most"       "other"      "some"       "such"       "no"
## [166] "nor"        "not"        "only"       "own"        "same"
## [171] "so"         "than"       "too"        "very"       "will"
```

This list of words is thus passed to the `remove` argument in the `dfm()` to ignore these words. If you are using texts in another language, make sure to specify the language, such as `stopwords('nl')` for Dutch or `stopwords('de')` for German.

There are various alternative preprocessing techniques, including more advanced techniques that are not implemented in quanteda. Whether, when and how to use these techniques is a broad topic that we won't cover today.

For this tutorial, we'll use the State of the Union speeches. We already created the corpus above. We can now pass this corpus to the `dfm()` function and set the preprocessing parameters.

```
(dtm <- dfm(corp, tolower=T, stem=T, remove=stopwords('en'), remove_punct=T))
```

```
## Document-feature matrix of: 23,469 documents, 20,431 features (99.8% sparse).
```

This dtm has 23,469 documents and 20,429 features (i.e. terms), and no longer shows the actual matrix because it simply wouldn't fit. Depending on the type of analysis that you want to conduct, we might not need this many words, or might actually run into computational limitations.

Luckily, many of these 20K features are not that informative. The distribution of term frequencies tends to have a very long tail, with many words occuring only once or a few times in our corpus. For many types of bag-of-words analysis it would not harm to remove these words, and it might actually improve results.

We can use the `dfm_trim` function to remove columns based on criteria specified in the arguments. Here we say that we want to remove all terms for which the frequency (i.e. the sum value of the column in the DTM) is below 10.

```
(dtm  <- dfm_trim(dtm, min_termfreq = 10))
```

```
## Document-feature matrix of: 23,469 documents, 5,293 features (99.4% sparse).
```

Now we have about 5000 features left. See `?dfm_trim` for more options.

In many cases, especially for languages with a richer morphology than English, it can be useful to use tools from computational linguistics to preprocess the data. In particular, *lemmatization* often works better for stemming, and *Part of Speech tagging* can be a great way to select e.g. only the names or verbs in a document.

`udpipe` is an R package that can do many preprocessing steps for a variety of languages including English, French, German and Dutch. If you call it for a language you have not previously used, it will automatically download the language model.

For this example, we will use a very short text, as it can take (very) long to process large amounts of text.

```
small_text <- c("Pelosi says Trump is welcome to testify in impeachment inquiry, if he chooses", "House
small_corpus <- corpus(small_text)
```

small_text <- c("Pelosi says Trump is welcome to testify in impeachment inquiry, if he chooses", "House speaker pushes back against president's accusations that process is stacked against him as Schumer echoes her suggestion")

Now, let's lemmatize and tag this corpus:

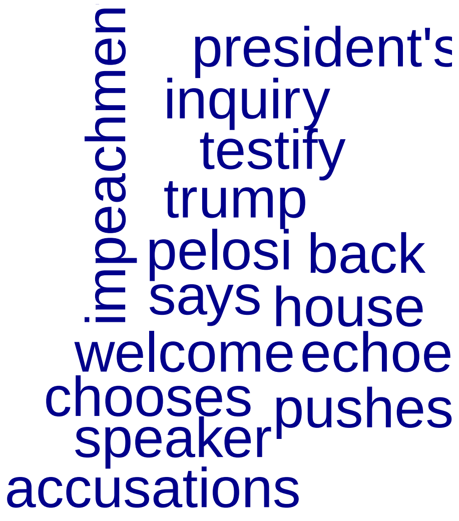
```
library(udpipe)
tokens <- udpipe(texts(small_corpus), "english", parser="none")
tokens %>% as_tibble() %>% select(token_id:xpos)
```

```
## # A tibble: 33 x 5
##    token_id token       lemma       upos  xpos
##    <chr>    <chr>       <chr>       <chr> <chr>
##  1 1        Pelosi      Pelosi      PROPN NNP
##  2 2        says        say         VERB  VBZ
##  3 3        Trump       Trump       PROPN NNP
##  4 4        is          be          AUX   VBZ
##  5 5        welcome     welcome     ADJ   JJ
##  6 6        to          to          PART  TO
##  7 7        testify     testify     VERB  VB
##  8 8        in          in          ADP   IN
##  9 9        impeachment impeachment ADJ   JJ
## 10 10       inquiry     inquiry     NOUN  NN
## # ... with 23 more rows
```

As you can see, 'is' is lemmatized to 'be', and Pelosi and Trump are both recognized as proper nouns (names).

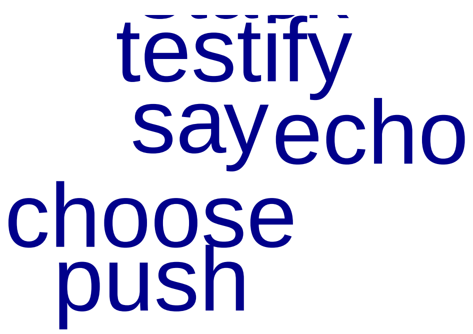
We can create a dfm of all lemmata, assinging to udpipe tokens to the corpus and proceeding as normal:

```
small_corpus$tokens <-  as.tokens(split(tokens$lemma, tokens$doc_id))
d <- dfm(small_corpus, remove=stopwords("english"), remove_punct=T)
textplot_wordcloud(d, min_count = 1)
```



More interesting, however, is to e.g. select only the verbs: Note that here I skip the corpus steps to show how you can also work directly with the texts and tokens:

```
tokens <- udpipe(small_text, "english", parser="none")
d <- tokens %>% filter(upos == "VERB") %>%
  with(split(lemma, doc_id)) %>% as.tokens() %>%
  dfm(remove=stopwords("german")) %>%
textplot_wordcloud(d, min_count=1)
```

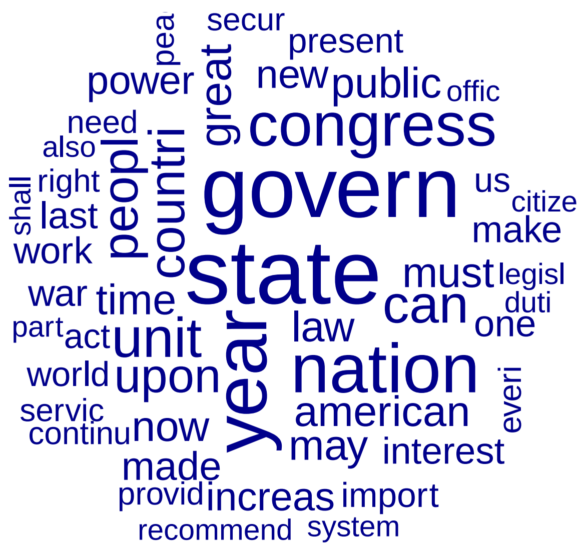


Of course, this is more meaningful if you run it on a larger text.

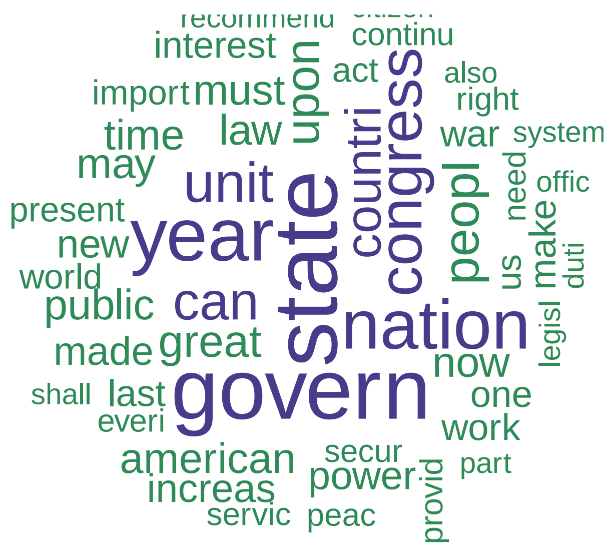
**Step 3: Analysis**: Using the dtm we can now employ various techniques.

1. Get most frequent words in corpus.

```
textplot_wordcloud(dtm, max_words = 50)      ## top 50 (most frequent) words
```



```r
textplot_wordcloud(dtm, max_words = 50, color = c('seagreen','slateblue4')) ## change colors
```



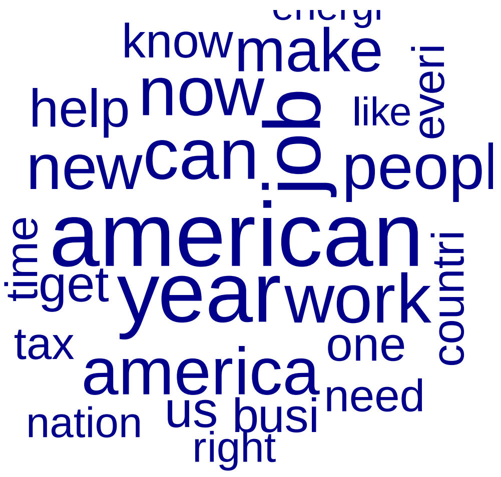
```r
textstat_frequency(dtm, n = 10)              ## view the frequencies
```

```
##     feature frequency rank docfreq group
## 1     state      9231    1    5579   all
## 2    govern      8576    2    5549   all
## 3      year      7241    3    4931   all
## 4    nation      6724    4    4843   all
## 5  congress      5685    5    4491   all
## 6      unit      5223    6    3715   all
## 7       can      4727    7    3627   all
## 8   countri      4664    8    3612   all
## 9     peopl      4467    9    3379   all
## 10     upon      4168   10    3004   all
```

You can also inspect a subcorpus. For example, looking only at Obama speeches. To subset the DTM we can use quanteda's `dtm_subset()`, but we can also use the more general R subsetting techniques (as discussed last week). Here we'll use the latter for illustration.

With `docvars(dtm)` we get a data.frame with the document variables. With `docvars(dtm)$President`, we get the character vector with president names. Thus, with `docvars(dtm)$President == 'Barack Obama'` we look for all documents where the president was Obama. To make this more explicit, we store the logical vector, that shows which documents are 'TRUE', as is_obama. We then use this to select these rows from the DTM.

```
is_obama <- docvars(dtm)$President == 'Barack Obama'
obama_dtm <- dtm[is_obama,]
textplot_wordcloud(obama_dtm, max_words = 25)
```



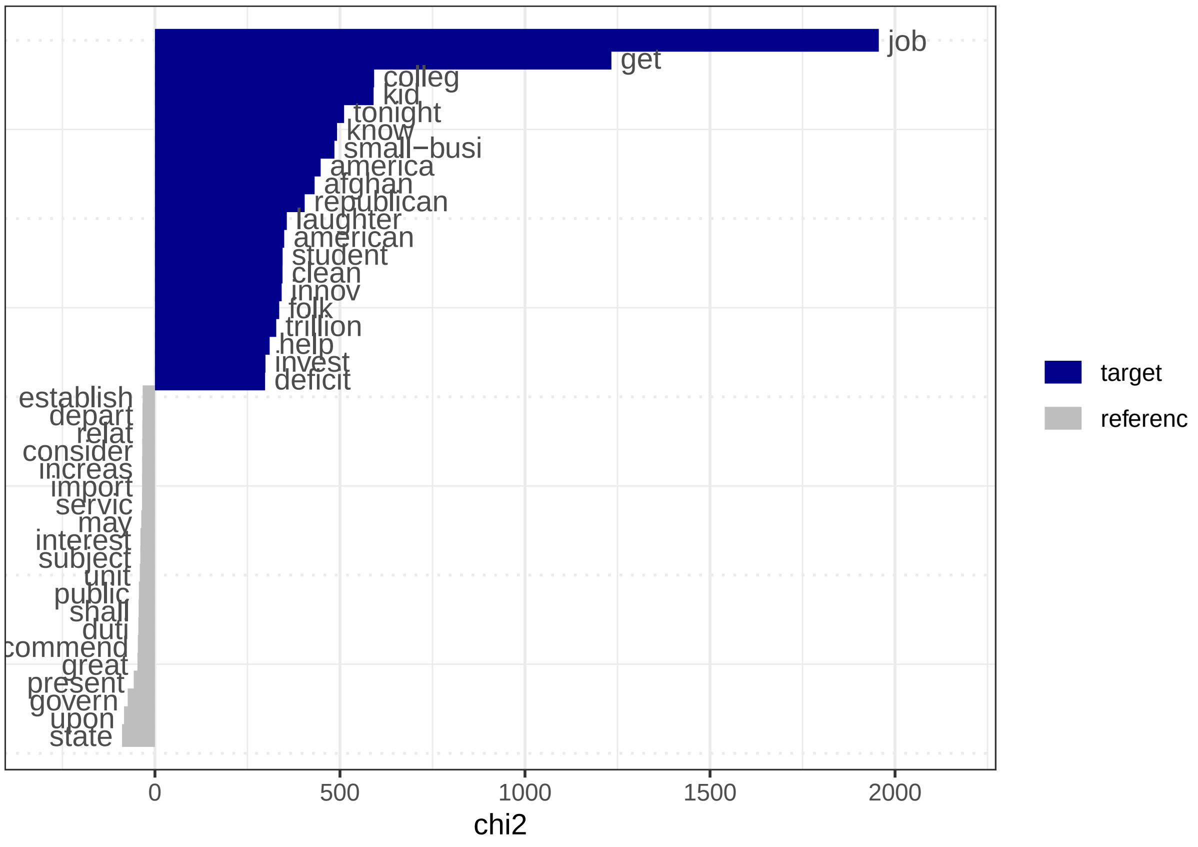
2. Compare word frequencies between two subcorpora. Here we (again) first use a comparison to get the is_obama vector. We then use this in the `textstat_keyness()` function to indicate that we want to compare the Obama documents (where is_obama is TRUE) to all other documents (where is_obama is FALSE).

```
is_obama <- docvars(dtm)$President == 'Barack Obama'
ts <- textstat_keyness(dtm, is_obama)
head(ts, n = 20)
```

```
##       feature      chi2 p n_target n_reference
## 1         job 1956.0965 0      202         628
## 2         get 1233.4504 0      121         355
## 3      colleg  592.3280 0       57         160
## 4         kid  590.8774 0       30          34
## 5     tonight  511.1499 0       82         399
## 6        know  492.1576 0      108         694
## 7  small-busi  485.0449 0       14           3
## 8     america  447.8391 0      169        1644
## 9      afghan  431.5014 0       17          11
## 10 republican  404.6363 0       41         121
## 11   laughter  356.3841 0       28          60
## 12   american  349.3285 0      240        3385
## 13    student  344.9895 0       41         144
## 14      clean  344.6813 0       35         103
## 15      innov  342.4968 0       27          58
## 16       folk  335.7901 0       14          10
## 17   trillion  327.7738 0       16          16
## 18       help  309.9923 0      126        1289
## 19     invest  298.5842 0       72         500
## 20    deficit  297.6741 0       56         315
```

We can visualize these results, stored under the name `ts`, by using the textplot_keyness function

```r
textplot_keyness(ts)
```



3. A keyword-in-context listing shows a given keyword in the context of its use. This is a good help for interpreting words from a wordcloud or keyness plot. Since a DTM only knows word frequencies, the `kwic()` function requires the corpus object as input.

```r
head(k <- kwic(corp, 'freedom', window = 7))
```

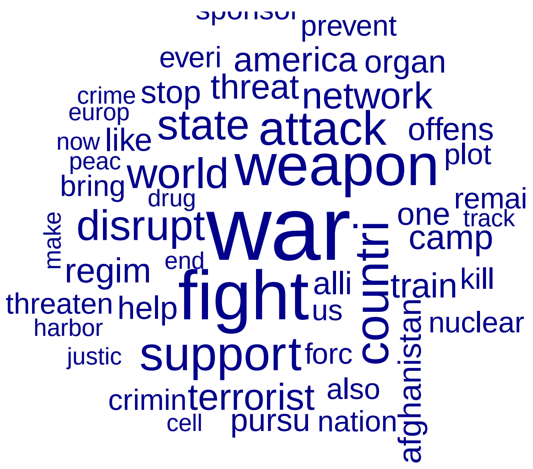
```
##
##  [text126, 62]      without harmony as far as consists with | freedom |
##  [text357, 84]           a wide spread for the blessings of | freedom |
##  [text466, 84] commerce of the United States its legitimate | freedom |
##  [text481, 89]    of cheaper materials and subsistence, the | freedom |
##  [text483, 23]      payment of the public debt whenever the | freedom |
##  [text626, 32]     its progress a force proportioned to its | freedom |
##
##  of sentiment its dignity may be lost
##  and equal laws.
##  . The instructions to our ministers with
##  of labor from taxation with us,
##  and safety of our commerce shall be
##  , and that the union of these
```

The `kwic()` function can also be used to focus an analysis on a specific search term. You can use the output of the kwic function to create a new DTM, in which only the words within the shown window are included in the DTM. With the following code, a DTM is created that only contains words that occur within 10 words from `terror*` (terrorism, terrorist, terror, etc.).

```r
terror <- kwic(corp, 'terror*')
terror_corp <- corpus(terror)
terror_dtm <- dfm(terror_corp, tolower=T, remove=stopwords('en'), stem=T, remove_punct=T)
```

Now you can focus an analysis on whether and how Presidents talk about `terror*`.

```r
textplot_wordcloud(terror_dtm, max_words = 50)      ## top 50 (most frequent) words
```



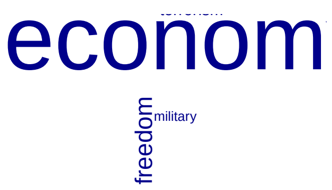
4. You can perform a basic dictionary search. Quanteda supports the use of existing dictionaries, for instance for sentiment analysis (but mostly for english dictionaries). An convenient way of using dictionaries is to make a DTM with the columns representing dictionary terms.

```r
dict <- dictionary(list(terrorism = 'terror*',
                        economy = c('econom*', 'tax*', 'job*'),
                        military = c('army','navy','military',
                                     'airforce','soldier'),
                        freedom = c('freedom','liberty')))
(dict_dtm <- dfm_lookup(dtm, dict, exclusive=TRUE))
```

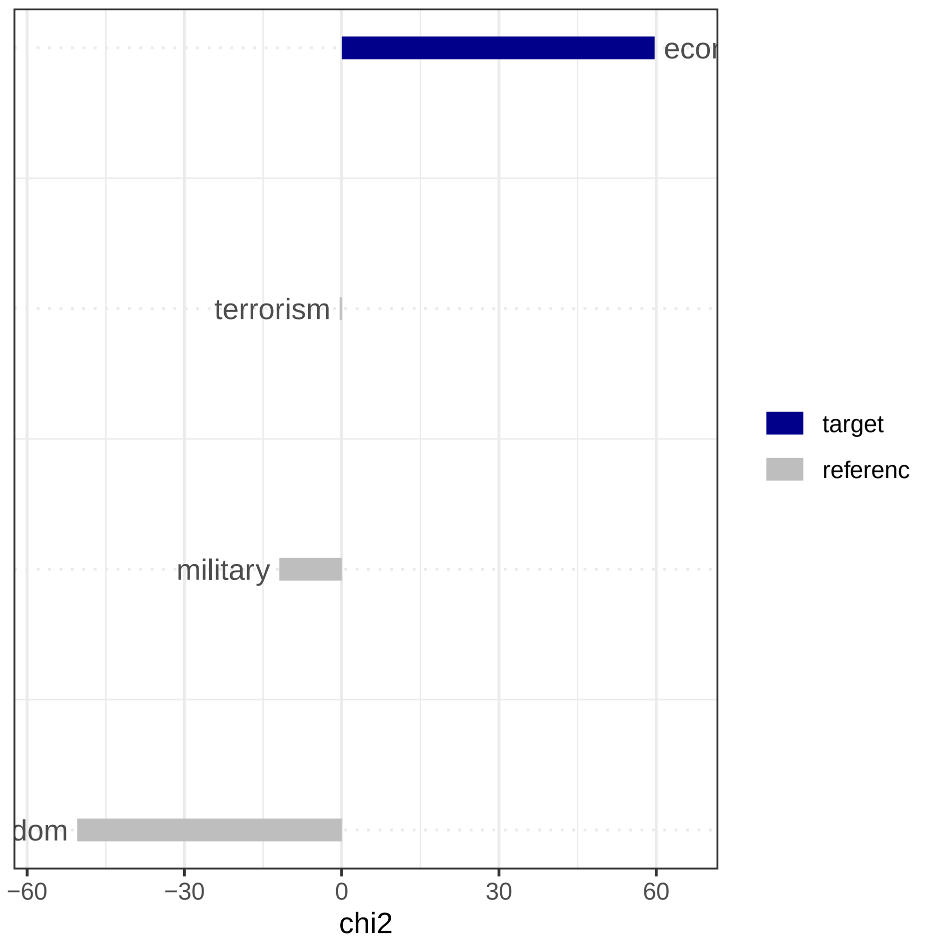
```
## Document-feature matrix of: 23,469 documents, 4 features (95.6% sparse).
```

The "4 features" are the four entries in our dictionary. Now you can perform all the analyses with dictionaries.

```r
textplot_wordcloud(dict_dtm)
```



```r
tk <- textstat_keyness(dict_dtm, docvars(dict_dtm)$President == 'Barack Obama')
textplot_keyness(tk)
```



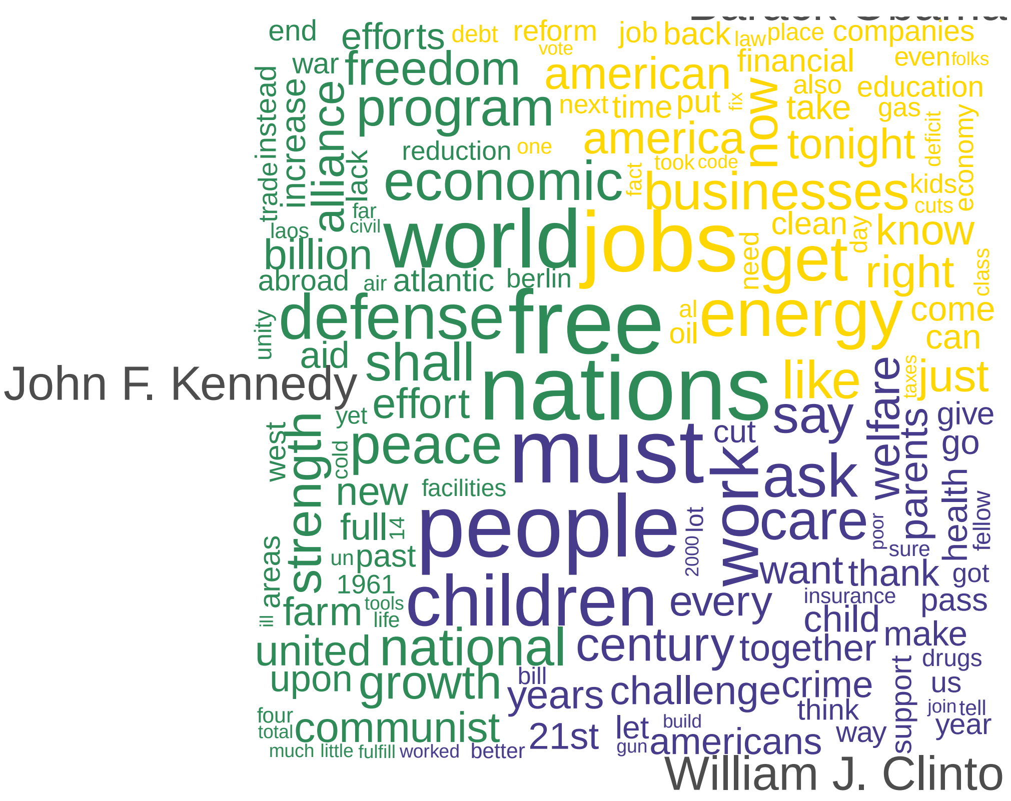
You can also convert the dtm to a data frame to get counts of each concept per document (which you can then match with e.g. survey data).

```
head(df <- convert(dict_dtm, to="data.frame"))
```

```
##   document terrorism economy military freedom
## 1    text1         0       0        0       0
## 2    text2         0       0        0       0
## 3    text3         0       0        0       0
## 4    text4         0       0        0       0
## 5    text5         0       1        1       0
## 6    text6         0       0        0       0
```

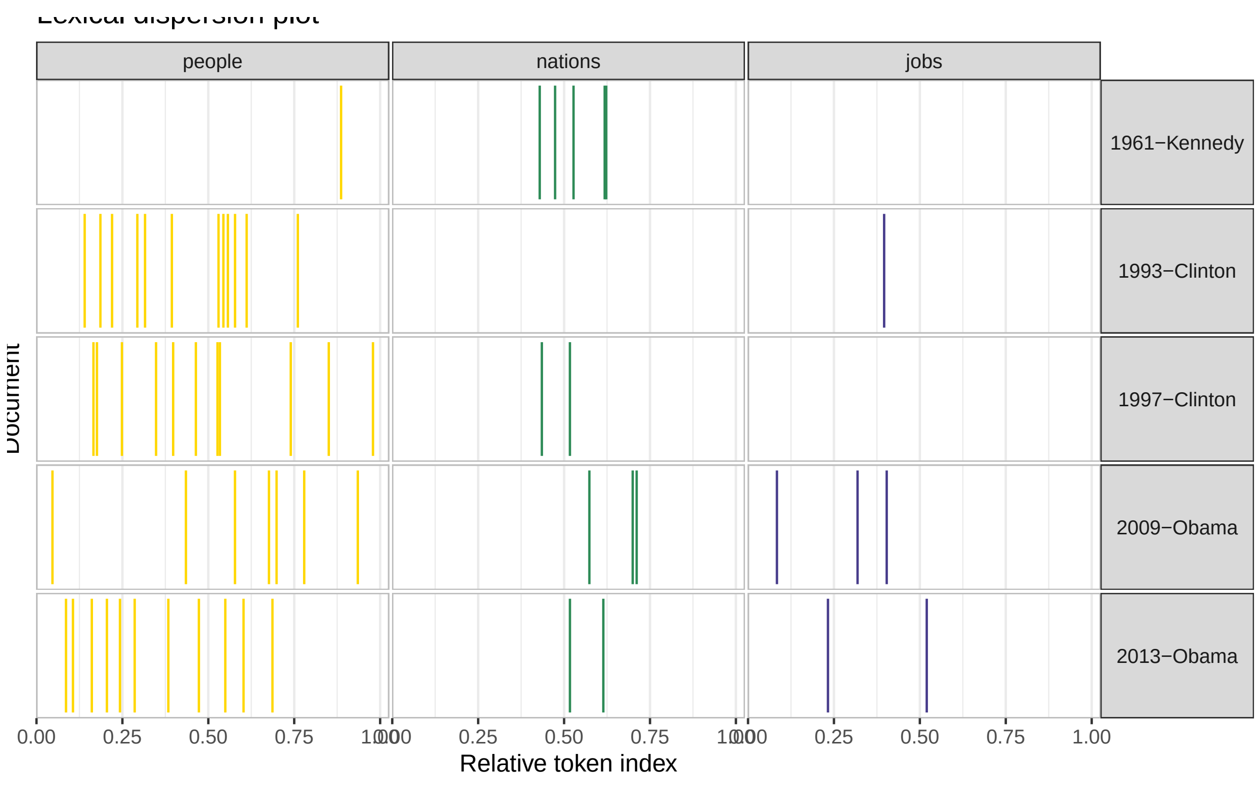
***Assignment: Textual Visualization***

*1a) Use the State of the Union Speeches that we imported during the explanation above. Compare the speeches of Barack Obama to those of John F. Kennedy and Bill Clinton (William J. Clinton) using a word cloud, shown in the visualization below. For help, you can look here*



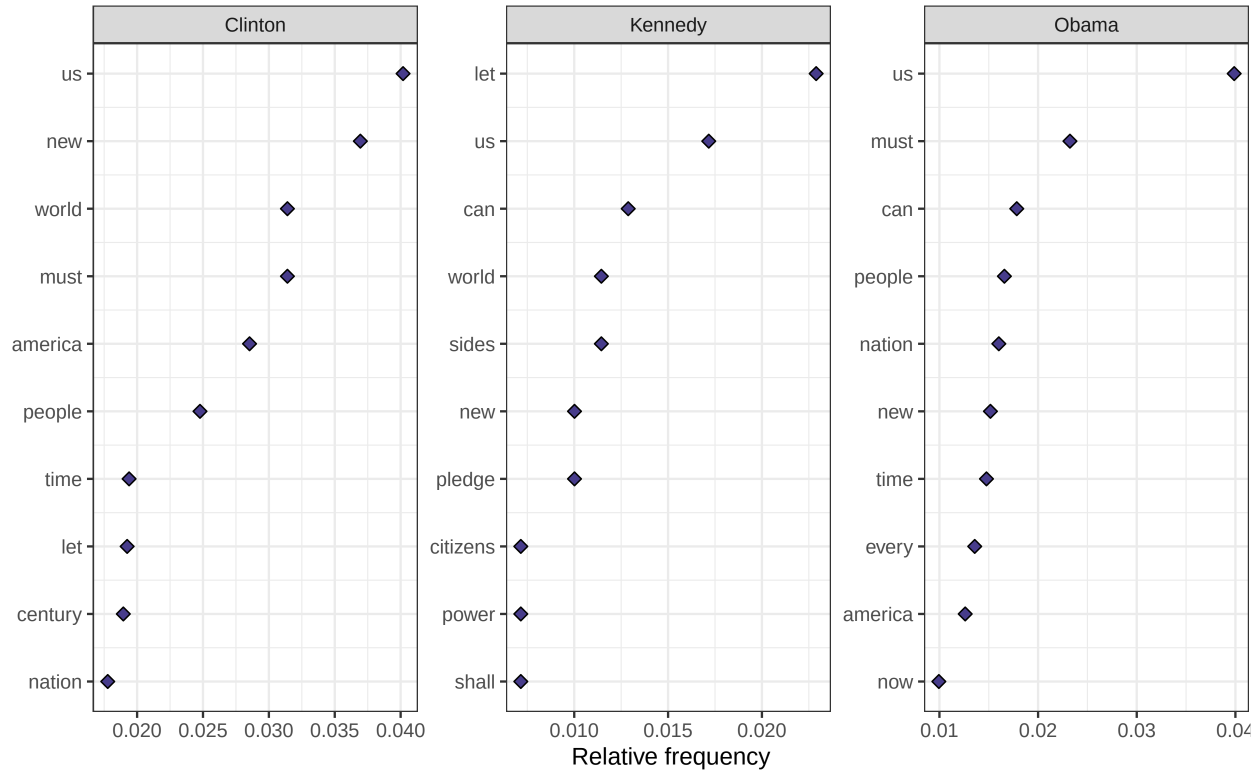
*1b) If you superficially compare the speeches given by these three presidents, what can you conclude? To be more precise in your comparison, what type of visualizations could you use? Or what type of annotations to the graph could you make?*

*1c) Based on the wordcloud above, we see that 'people', 'nations' and 'jobs' are words used a lot by the three presidents in their State of the Union speeches. Would that also hold for their inaugural speeches? Plotting a `kwic` object produces a lexical dispersion plot which allows us to visualize the occurrences of particular terms throughout the text. We call these "x-ray" plots due to their similarity to the data produced by Amazon's "x-ray" feature for Kindle books. Replicate the following visualization using `textplot_xray` and `kwic` and the `data_corpus_inaugural` corpus from quanteda.*

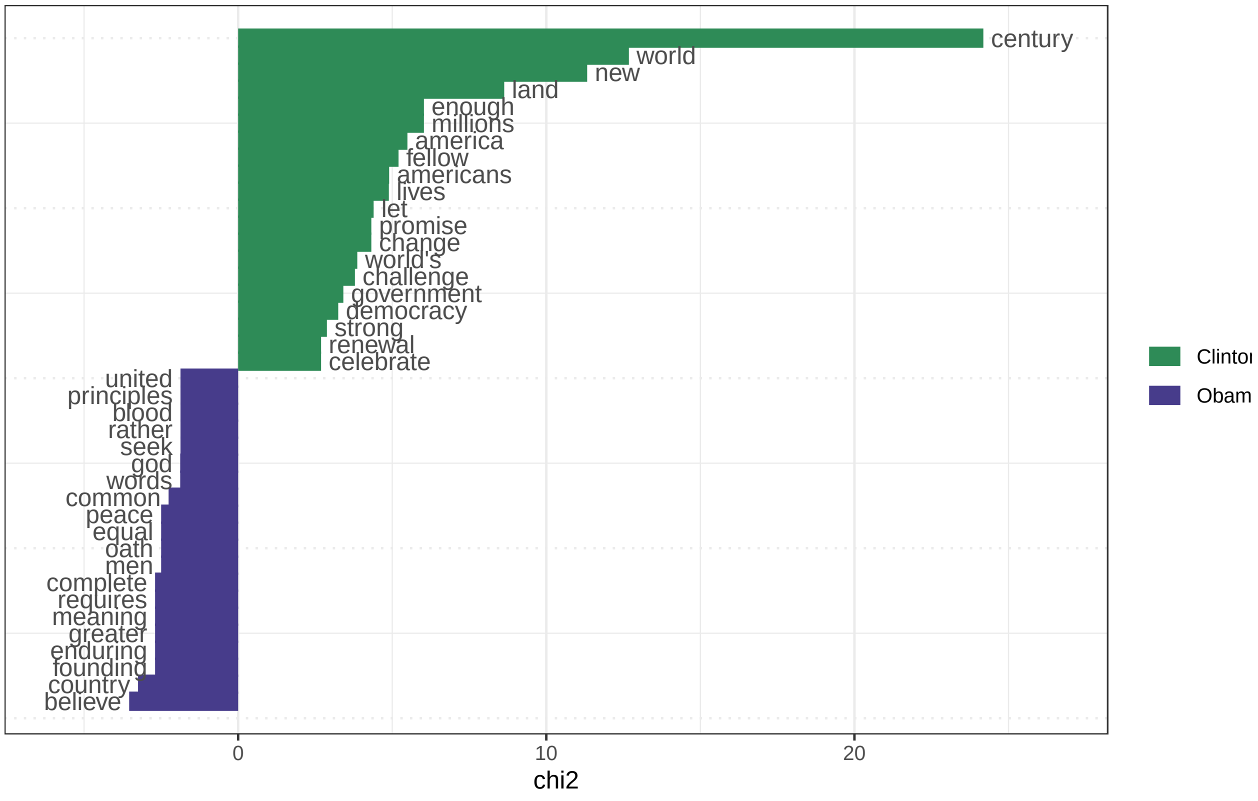


*1d) What does the x-ray plot show? Is this easier to compare the three presidents? Why (not)?*

*1e) Another way to compare text is to look at the frequency of the top features in a text, using* `topfeatures`*. Again, use the* `data_corpus_inaugural` *and replicate the following visualization. For some help, have a look* *here*



*1f) If you want to compare the differential associations of keywords in a target and reference group, you can calculate "keyness" which is based on* `textstat_keyness`*. Replicate the visualization where Obama is compared to Clinton.*



*1g) Make the same visualizations comparing Clinton and Kennedy and Kennedy and Obama. What conclusions can you draw from these comparisons?*

## Good Luck

The deadline for the assignment is **December 12th, 10am**!

## References

Healy, K. (2018). *Data visualization: a practical introduction*. Princeton University Press.

CCS Amsterdam's Github