# Packages

To run the following code, make sure you have loaded (and installed) the following packages. We like to set **ggplot2** to use the minimal theme, but this is of course entirely optional.

```
library(cleanNLP)
library(dplyr)
library(readr)
library(stringi)
library(ggplot2)
library(tokenizers)
```

```
## Warning: package 'tokenizers' was built under R version 3.4.4
```

```
theme_set(theme_minimal())
```

# Splitting text into words

Consider a string in R containing the first paragraph of text from the novel *L'Étranger* of Albert Camus (we'll use `stri_wrap` just to fit the output on the slide):

```
stri_wrap(letranger)
```

```
## [1] "Aujourd'hui, maman est morte. Ou peut-être hier, je ne"
## [2] "sais pas.J'ai reçu un télégramme de l'asile: «Mère décédée."
## [3] "Enterrement demain.Sentiments distingués.» Cela ne veut rien"
## [4] "dire. C'était peut-êtrehier."
```

In order to work with this text, a good first step is to split it apart into its constituent words.

# Splitting with whitespace

Splitting on whitespace alone works reasonably well, though there are some issues with punctuation marks:

```
stri_split(letranger, fixed = " ")
```

```
## [[1]]
##  [1] "Aujourd'hui,"        "maman"               "est"
##  [4] "morte."              "Ou"                  "peut-être"
##  [7] "hier,"               "je"                  "ne"
## [10] "sais"                "pas.J'ai"            "reçu"
## [13] "un"                  "télégramme"          "de"
## [16] "l'asile:"            "«Mère"               "décédée."
## [19] "Enterrement"         "demain.Sentiments" "distingués.»"
## [22] "Cela"                "ne"                  "veut"
## [25] "rien"                "dire."               "C'était"
## [28] "peut-êtrehier."
```

# Splitting with cleanNLP

There are a number of packages that support the more complex logic needed to deal with many of these errors. Here we'll use the **cleanNLP** package as it will be easy to adapt our approach to work with more complex annotators in the next section.

# Running annotations

We start by initialising the tokenizers back end within the **cleanNLP** package. We'll indicate that we want a French locale as this input text is in French.

```
library(cleanNLP)
init_tokenizers(locale = "fr")
```

Then, we run the annotators over the text. We set the option `as_strings` because we are passing the text into the function as a raw string:

```
letranger_anno <- run_annotators(letranger, as_strings = TRUE)
letranger_anno
```

```
##
## A CleanNLP Annotation:
##   num. documents: 1
```

# An annotation object

The result seems to be wrapped up in a fairly complex object; however, it is nothing more than a list of data frames. To collapse all of these lists into a one table summary of the tokenisation process, we will call the function `get_combine` on the annotation object:

```
letranger_tokens <- get_combine(letranger_anno)
```

```
## NOTE: get_combine has been renamed cnlp_get_tif
```

# An annotation object

The result is a data frame with one row for each token. Meta data about each token, such as the sentence number and character offset, are included as columns.

```
letranger_tokens
```

```
## # A tibble: 42 x 6
##   id      sid   tid word          cid spaces
##   <chr> <int> <int> <chr>       <int>  <dbl>
## 1 doc1      1     1 Aujourd'hui     1   0
## 2 doc1      1     2 ,              12   1.00
## 3 doc1      1     3 maman          14   1.00
## 4 doc1      1     4 est            20   1.00
## 5 doc1      1     5 morte          24   0
## 6 doc1      1     6 .              29   1.00
## # ... with 36 more rows
```

# cleanNLP tokenization results

Notice that the resulting tokens fix most of the problems in the original white space based technique:

```
letranger_tokens$word
```

```
##  [1] "Aujourd'hui"         ","              "maman"
##  [4] "est"                 "morte"          "."
##  [7] "Ou"                  "peut"           "-"
## [10] "être"                "hier"           ","
## [13] "je"                  "ne"             "sais"
## [16] "pas.J'ai"            "reçu"           "un"
## [19] "télégramme"          "de"             "l'asile"
## [22] ":"                   "«"              "Mère"
## [25] "décédée"             "."              "Enterrement"
## [28] "demain.Sentiments"   "distingués"     "."
## [31] "»"                   "Cela"           "ne"
## [34] "veut"                "rien"           "dire"
## [37] "."                   "C'était"        "peut"
## [40] "-"                   "êtrehier"       "."
```

## Corpus Metadata

A dataset where each record is its own text is known as a *corpus*. In the remainder of this session, we will be working with a corpus of the 56 short stories featuring Sherlock Holmes.

We start by constructing a meta data table of these stories:

```
paths <- dir("data/holmes_stories", full.names = TRUE)
sh_meta <- data_frame(id = as.integer(seq_along(paths)),
                      story = stri_sub(basename(paths), 4, -5))
sh_meta %>% print(n = 5)
```

```
## # A tibble: 56 x 2
##      id story
##   <int> <chr>
## 1     1 a_scandal_in_bohemia
## 2     2 the_redheaded_league
## 3     3 a_case_of_identity
## 4     4 the_boscombe_valley_mystery
## 5     5 the_five_orange_pips
## # ... with 51 more rows
```

# Annotating files on disk

We want to construct a similar data frame of tokens for all of the short stories in our corpus. As a first step, we will re-initalise the tokenizers backend using an English locale:

```
library(cleanNLP)
init_tokenizers(locale = "en_GB")
```

Then, we call the annotation engine with the paths to the files instead of the raw text:

```
sh_anno <- run_annotators(paths)
```

And once again, collapse the object into a single table.

```
sh_tokens <- get_combine(sh_anno)
```

```
## NOTE: get_combine has been renamed cnlp_get_tif
```

# Sherlock Holmes tokens

The resulting table, as before, has one row for each token in the original dataset.

```
library(magrittr)
sh_tokens %<>% mutate(id = as.integer(gsub("doc","",id)))
sh_tokens
```

```
## # A tibble: 550,697 x 6
##        id   sid   tid word        cid spaces
##     <int> <int> <int> <chr>     <int>  <dbl>
## 1       1     1     1 To            2   1.00
## 2       1     1     2 Sherlock      5   1.00
## 3       1     1     3 Holmes       14   1.00
## 4       1     1     4 she          21   1.00
## 5       1     1     5 is           25   1.00
## 6       1     1     6 always       28   1.00
## # ... with 5.507e+05 more rows
```

# Sherlock Holmes tokens, sentence 10

```
sh_tokens %>% filter(id == 1) %>% filter(sid == 10) %>% print(n = 12)
```

```
## # A tibble: 35 x 6
##         id   sid   tid word           cid spaces
##      <int> <int> <int> <chr>        <int>  <dbl>
##  1       1    10     1 Grit           875   1.00
##  2       1    10     2 in             880   1.00
##  3       1    10     3 a              883   1.00
##  4       1    10     4 sensitive      885   1.00
##  5       1    10     5 instrument     895   0
##  6       1    10     6 ,              905   1.00
##  7       1    10     7 or             907   1.00
##  8       1    10     8 a              910   1.00
##  9       1    10     9 crack          912   1.00
## 10       1    10    10 in             918   1.00
## 11       1    10    11 one            921   1.00
## 12       1    10    12 of             925   1.00
## # ... with 23 more rows
```

## Using tokens

One can often learn a lot about a corpus by simply finding the occurances of certain tokens or patterns of tokens within it.

For example:

- length of sentences
- number of citations
- presence of known characters
- count of hashtags in a tweet
- ratio of quotes/dialogue to raw text

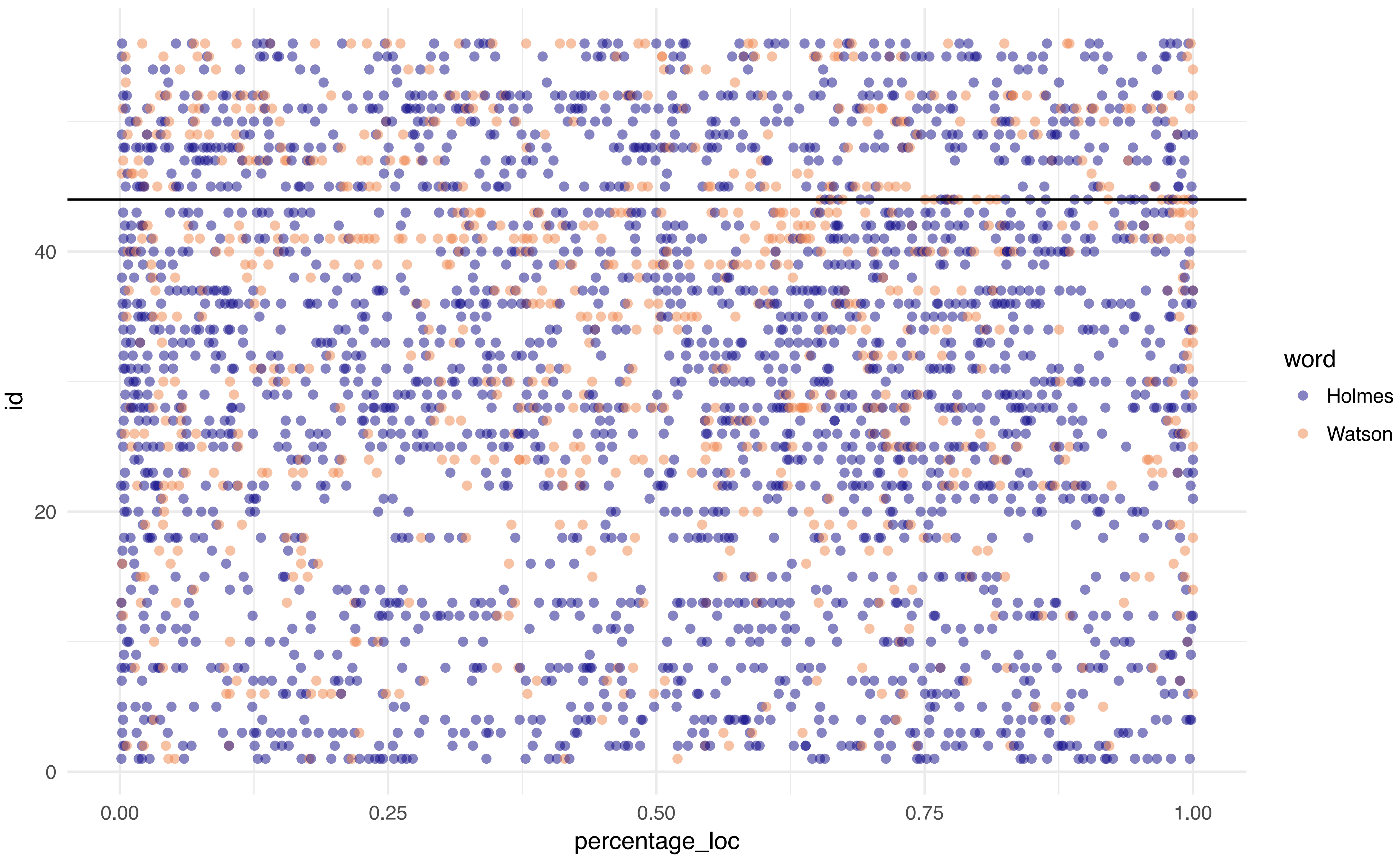
# Visualising Watson and Holmes

Where do Watson and Holmes occur within each text?

```r
sh_tokens %>%
  group_by(id) %>%
  mutate(percentage_loc = sid / max(sid)) %>%
  filter(word %in% c("Watson", "Holmes")) %>%
  ggplot(aes(percentage_loc, id)) +
    geom_point(aes(color = word), alpha = 0.5) +
    geom_hline(yintercept = 44)
```

# Visualising Watson and Holmes

```
## Warning: package 'viridis' was built under R version 3.4.4
```

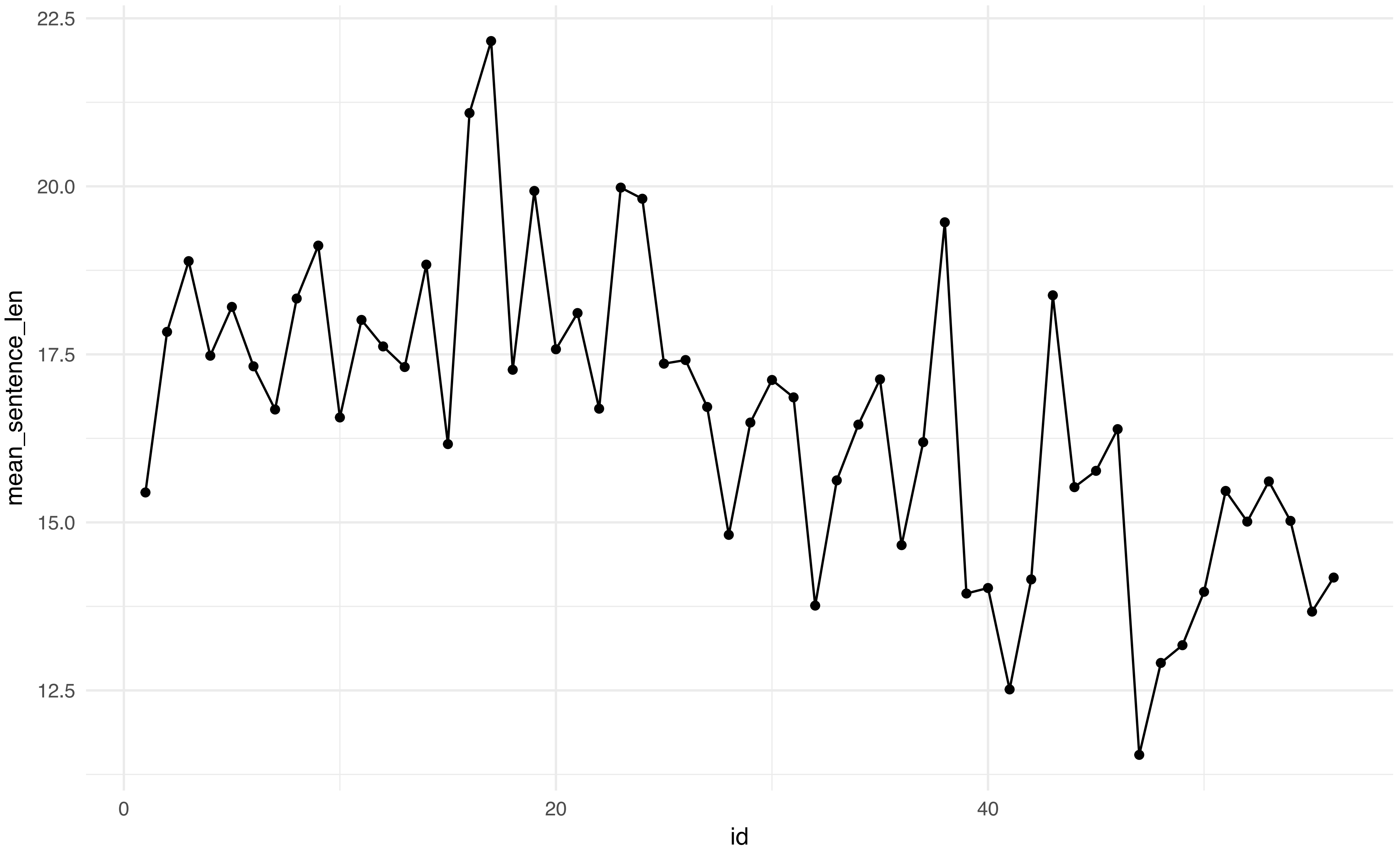


# Average sentence length

By counting the number of periods, question marks, and exclamation marks, we can approximate the average length of each sentence in the text.

```
sh_tokens %>%
  mutate(sentence_end = word %in% c(".", "?", "!")) %>%
  group_by(id) %>%
  summarize(mean_sentence_len = n() / sum(sentence_end)) %>%
  ggplot(aes(id, mean_sentence_len)) +
    geom_line() +
    geom_point()
```

# Average sentence length



## Frequency tokens

In our first set of analyses, we used our prior knowledge to determine which tokens would be interesting to identify and tabulate.

Alternatively, we can let the corpus itself tell us which terms would be the most interesting. Often just knowing which terms are the most frequent, or occur in certain patterns, is interesting itself.

# Naïve approach

To begin, we can see what the most common words are across the corpus using the `count` function:

```
sh_tokens %>%
  count(id, word, sort = TRUE)
```

```
## # A tibble: 105,460 x 3
##       id word      n
##    <int> <chr> <int>
## 1     22 ,       827
## 2     28 .       822
## 3     37 ,       794
## 4     40 .       778
## 5     22 .       770
## 6     28 ,       765
## # ... with 1.055e+05 more rows
```

# Stop words

What is interesting about these top terms? Cynically, we might say very little. The problem is that the most common terms are simply punctuation marks.

If we look farther down the list, common function words such as 'the' and 'my' dominate the counts.

```
## # A tibble: 6 x 3
##      id word      n
##   <int> <chr> <int>
## 1     1 I       259
## 2     1 to      242
## 3     1 of      235
## 4     1 and     227
## 5     1 a       212
## 6     1 in      152
```

A popular way of dealing with this problem is to define a list of *stop words*, those tokens that are common enough to be thematically uninteresting.

# Stop words

We have included a simple list of stop words in the dataset for today, which you should read in with the following.

```
stopwords <- readLines("data/stopwords_en.txt")
sample(stopwords, 25L)
```

```
##  [1] "opened"    "said"       "puts"     "'"
##  [5] "alone"     "ended"      "nor"      "by"
##  [9] "use"       "rooms"      "cannot"   "can"
## [13] "together"  "it"         "almost"   "asks"
## [17] "their"     "greater"    "gives"    "against"
## [21] "presents"  "yourselves" "likely"   "shouldn't"
## [25] "beings"
```

Take a moment to look at some of the words. What parts of speech dominate the list?

## Most common non-stopwords

We will make use of the `top_n` function to select just the top 10 occurrences within each text. We also add a call to `left_join` to explicitly add the names of each story to the output:

```
sh_toptokens <- sh_tokens %>%
  filter(!(tolower(word) %in% stopwords)) %>%
  count(id, word, sort = TRUE) %>%
  group_by(id) %>%
  top_n(n = 10, n) %>%
  left_join(sh_meta, by = "id")
```

# Tokens from 'A Scandal in Bohemia'

```
sh_toptokens %>% filter(id == 1) %>% print(n = Inf)
```

```
## # A tibble: 11 x 4
##       id word           n story
##    <int> <chr>      <int> <chr>
##  1     1 Holmes        48 a_scandal_in_bohemia
##  2     1 photograph    21 a_scandal_in_bohemia
##  3     1 King          17 a_scandal_in_bohemia
##  4     1 Majesty       16 a_scandal_in_bohemia
##  5     1 house         14 a_scandal_in_bohemia
##  6     1 little        14 a_scandal_in_bohemia
##  7     1 Adler         13 a_scandal_in_bohemia
##  8     1 door          13 a_scandal_in_bohemia
##  9     1 hand          13 a_scandal_in_bohemia
## 10     1 Irene         13 a_scandal_in_bohemia
## 11     1 minutes       13 a_scandal_in_bohemia
```

# Tokens from 'His Last Bow'

```r
sh_toptokens %>% filter(id == 44) %>% print(n = Inf)
```

```
## # A tibble: 12 x 4
##         id word           n story
##      <int> <chr>      <int> <chr>
##  1      44 Von           38 his_last_bow
##  2      44 Bork          34 his_last_bow
##  3      44 Holmes        21 his_last_bow
##  4      44 Watson        17 his_last_bow
##  5      44 American      12 his_last_bow
##  6      44 German        12 his_last_bow
##  7      44 little        12 his_last_bow
##  8      44 country       11 his_last_bow
##  9      44 car           10 his_last_bow
## 10      44 papers        10 his_last_bow
## 11      44 safe          10 his_last_bow
## 12      44 secretary     10 his_last_bow
```

# Why characters?

One particular category that floats to the top of our lists of most frequent tokens are the main characters for each story. Identifying the people mentioned in a corpus of text has many applications, including:

- on social media it indicates trending issues
- in news articles, the people mentioned give a good clue as to what topics are being discussed (politics, food, culture, local events, ..)
- in fiction, as we have seen, the presence and absence of characters is a major indicator of plot arcs

# Proper nouns

Some of the most frequenct non stop words in the texts refer to the names of the characters. How might we extract these directly?

```
sh_propn <- sh_tokens %>%
  filter(!(tolower(word) %in% stopwords)) %>%
  filter((tolower(word) != word)) %>%
  count(id, word, sort = TRUE) %>%
  group_by(id) %>%
  top_n(n = 10, n) %>%
  left_join(sh_meta, by = "id")
```

## Proper nouns from ‘A Scandal in Bohemia’

```r
sh_propn %>% filter(id == 1) %>% print(n = Inf)
```

```
## # A tibble: 11 x 4
##       id word         n story
##    <int> <chr>    <int> <chr>
##  1     1 Holmes      48 a_scandal_in_bohemia
##  2     1 King        17 a_scandal_in_bohemia
##  3     1 Majesty     16 a_scandal_in_bohemia
##  4     1 Adler       13 a_scandal_in_bohemia
##  5     1 Irene       13 a_scandal_in_bohemia
##  6     1 Briony      11 a_scandal_in_bohemia
##  7     1 Lodge       11 a_scandal_in_bohemia
##  8     1 Sherlock    11 a_scandal_in_bohemia
##  9     1 Bohemia      7 a_scandal_in_bohemia
## 10     1 Norton       7 a_scandal_in_bohemia
## 11     1 Street       7 a_scandal_in_bohemia
```

## Proper nouns from 'His Last Bow'

```r
sh_propn %>% filter(id == 44) %>% print(n = Inf)
```

```
## # A tibble: 10 x 4
##        id word          n story
##     <int> <chr>     <int> <chr>
##  1     44 Von          38 his_last_bow
##  2     44 Bork         34 his_last_bow
##  3     44 Holmes       21 his_last_bow
##  4     44 Watson       17 his_last_bow
##  5     44 American     12 his_last_bow
##  6     44 German       12 his_last_bow
##  7     44 Altamont      8 his_last_bow
##  8     44 England       8 his_last_bow
##  9     44 Martha        7 his_last_bow
## 10     44 Baron         6 his_last_bow
```

# One top character

Of course, two major characters are always going to be Sherlock Holmes and John Watson. Let us remove them from the list, as well as any names shorter than four characters (these are usually honorific rather than names). We will then take the most mentioned character from each text:

```
holmes_watson <- c("Sherlock", "Holmes", "John", "Watson")
sh_topchar <- sh_tokens %>%
  filter(stri_length(word) > 4) %>%
  filter(!(word %in% holmes_watson)) %>%
  filter(!(tolower(word) %in% stopwords)) %>%
  filter((tolower(word) != word)) %>%
  count(id, word) %>%
  left_join(sh_meta, by = "id") %>%
  group_by(id) %>%
  top_n(n = 1, n)
```

# One top character, cont.

```
sh_topchar %>% filter(id %in% c(1, 45)) %>%
  print(n = Inf)
```

```
## # A tibble: 2 x 4
##      id word         n story
##   <int> <chr>    <int> <chr>
## 1     1 Majesty     16 a_scandal_in_bohemia
## 2    45 Baron       17 the_illustrious_client
```

## Top name

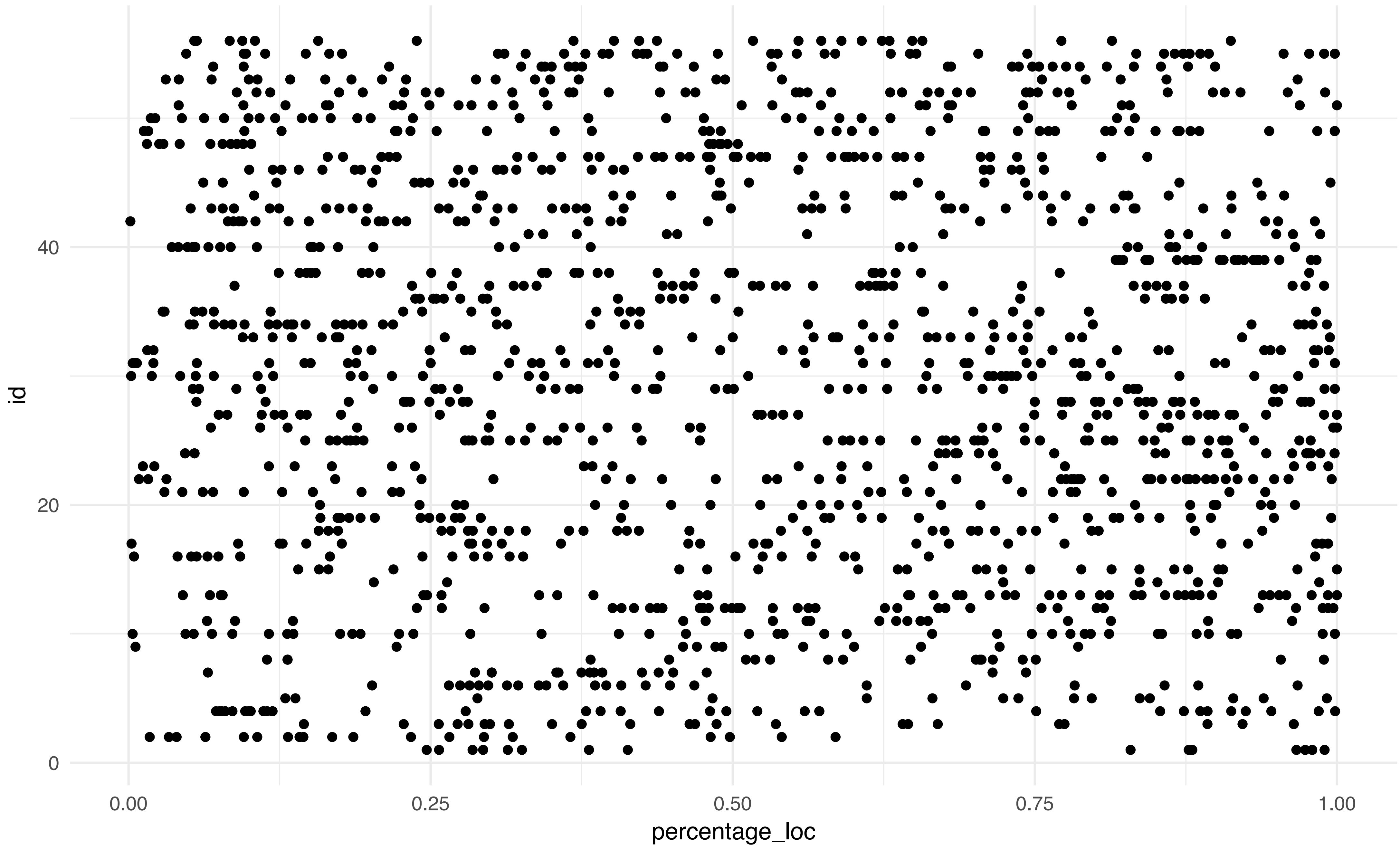
Sometimes this works well, sometimes is picks up the right idea but not enough to really know who the character is (i.e., "Colonel" or "Majesty"), and sometimes it works very well. We will see in the next session how to do a better job of this using more advanced annotation engines.

## Visualising main characters

We will make use of the `semi_join` function to plot the character locations:

```
sh_tokens %>%
  group_by(id) %>%
  mutate(percentage_loc = sid / max(sid)) %>%
  semi_join(sh_topchar, by = c("id", "word")) %>%
  ggplot(aes(percentage_loc, id)) +
    geom_point()
```

# Visualising main characters



# Textual topics and themes

When we look back at our original list of top tokens, many of those instances that are not characters describe the main topics, themes, or artefacts of interest in the story.

Finding these frequent, non-proper nouns can indicate the theme or topics of interest within a corpus of texts.

# Non-proper words

Our original code can be easily modified to only count those with all lower-case letters:

```
sh_theme <- sh_tokens %>%
  filter(!(tolower(word) %in% stopwords)) %>%
  filter((tolower(word) == word)) %>%
  count(id, word) %>%
  group_by(id) %>%
  top_n(n = 10, n) %>%
  left_join(sh_meta, by = "id")
```

# Non-proper words, cont.

```
sh_theme %>% filter(id == 1) %>% print(n = 10)
```

```
## # A tibble: 10 x 4
##       id word            n story
##    <int> <chr>       <int> <chr>
##  1     1 door           13 a_scandal_in_bohemia
##  2     1 half           11 a_scandal_in_bohemia
##  3     1 hand           13 a_scandal_in_bohemia
##  4     1 house          14 a_scandal_in_bohemia
##  5     1 little         14 a_scandal_in_bohemia
##  6     1 matter         11 a_scandal_in_bohemia
##  7     1 minutes        13 a_scandal_in_bohemia
##  8     1 photograph     21 a_scandal_in_bohemia
##  9     1 street         11 a_scandal_in_bohemia
## 10     1 woman          12 a_scandal_in_bohemia
```

## Word frequencies

What we need is something stronger than a stop word list; conveniently such a dataset is included in the **cleanNLP** package as the dataset `word_frequency`.

```
word_frequency %>% print(n = 9)
```

```
## # A tibble: 150,000 x 3
##   language word  frequency
##   <chr>    <chr>     <dbl>
## 1 en       the       3.93
## 2 en       of        2.24
## 3 en       and       2.21
## 4 en       to        2.06
## 5 en       a         1.54
## 6 en       in        1.44
## 7 en       for       1.01
## 8 en       is        0.800
## 9 en       on        0.638
## # ... with 1.5e+05 more rows
```

# Filtering by word frequency

Instead of a stopword list, we filter out those words with a certain frequency cut-off. By changing this tuning parameter, we can tweak the results until they look reasonable.

```
sh_wordfreq <- sh_tokens %>%
  mutate(lemma = tolower(word)) %>%
  inner_join(word_frequency, by = c("lemma" = "word")) %>%
  filter(frequency < 0.01) %>%
  filter((tolower(word) == word)) %>%
  count(id, word) %>%
  group_by(id) %>%
  top_n(n = 10, n) %>%
  left_join(sh_meta, by = "id") %>%
  arrange(id, desc(n))
```

A more complex method could compare these global probabilities to the frequency in our text and identify the most deviant probabilities.

# Filtering by word frequency, cont.

```
sh_wordfreq %>% filter(id == 1) %>% print(n = 12)
```

```
## # A tibble: 10 x 4
##         id word            n story
##      <int> <chr>       <int> <chr>
##  1       1 photograph     21 a_scandal_in_bohemia
##  2       1 door           13 a_scandal_in_bohemia
##  3       1 myself         11 a_scandal_in_bohemia
##  4       1 cried          10 a_scandal_in_bohemia
##  5       1 eyes            9 a_scandal_in_bohemia
##  6       1 lady            9 a_scandal_in_bohemia
##  7       1 heard           8 a_scandal_in_bohemia
##  8       1 indeed          8 a_scandal_in_bohemia
##  9       1 looked          8 a_scandal_in_bohemia
## 10       1 remarked        8 a_scandal_in_bohemia
```

# Annotation engines

# Back ends

We have been able to get some real, interesting results by splitting our raw text into tokens. Some clever filtering and use of external datasets has gotten us some rough results in terms of character identification and the detection of themes.

To go deeper though, we need a more advanced natural language processing engine. These extract more granular features of the text, such as identifying parts of speech and tagging particular known entities.

# Back end, cont.

In **cleanNLP**, we currently provide back ends to two of the most well-known such libraries:

- **spaCy** a Python library primarily built for speed and stability
- **CoreNLP** a Java library built to have bleeding-edge functionality

# Initialising back ends

To use one of these backends in **cleanNLP**, simply run an alternative `init_` function before annotating the text. Either use:

```
library(cleanNLP)
init_spaCy(model_name = "en")
anno <- run_annotators(paths)
nlp <- get_combine(anno)
```

Or:

```
library(cleanNLP)
init_coreNLP(language = "en")
anno <- run_annotators(paths)
nlp <- get_combine(anno)
```

# Annotation results

The resulting data set `nlp` also has one row per token, but now there are many additional features that have been learned from the text:

```
## # A tibble: 551,463 x 15
##      id   sid   tid word   lemma  upos  pos     cid source relation
##   <int> <int> <int> <chr>  <chr>  <chr> <chr> <int>  <int> <chr>
## 1     1     1     2 To     to     ADP   IN        1      0 ROOT
## 2     1     1     3 Sherl~ sherl~ PROPN NNP       4      4 compound
## 3     1     1     4 Holmes holmes PROPN NNP      13      2 pobj
## 4     1     1     5 she    -PRON- PRON  PRP      20      6 nsubj
## 5     1     1     6 is     be     VERB  VBZ      24      2 ccomp
## 6     1     1     7 always always ADV   RB       27      6 advmod
## # ... with 5.515e+05 more rows, and 5 more variables: word_source
## #   <chr>, lemma_source <chr>, entity_type <chr>, entity <chr>,
## #   spaces <int>
```

# Annotation tasks

NLP backends use models to learn features about the words and sentences in our raw text. Common tasks include:

- tokenisation
- lemmatisation
- sentence boundaries
- part of speech tags
- dependencies
- named entities
- coreferences
- sentiment analysis
- word embeddings

A collection of these running together (as they typically need to), is known as an **NLP Pipeline**. We will explain the meaning behind and some applications of many of these annotation tasks in these slides.

# Back end details

- options passed to the `init_` functions control which models and annotations are selected
- models have to be trained specifically for every natural language that they support
- more complex annotation tasks need to be trained seperately for different styles of speech (i.e., Twitter versus Newspapers)
- libraries needed for these types of annotations require large external dependencies in order to run correctly
- in the interest of time, today we will simply provide the annotation objects for our corpora of study.

More detailed instructions for setting up either back end can be found on the cleanNLP repository and we are happy to help as best we can during the break or after the tutorial.

# Reading data

As mentioned above, we have already run the spaCy annotators on the corpus of Sherlock Holmes stories and made them available in the GitHub repository:

```
paths <- dir("data/holmes_stories", full.names = TRUE)
sh_meta <- data_frame(id = seq_along(paths),
                      story = stri_sub(basename(paths), 4, -5))
sh_nlp <- read_csv("data/sh_nlp.csv.gz")
```

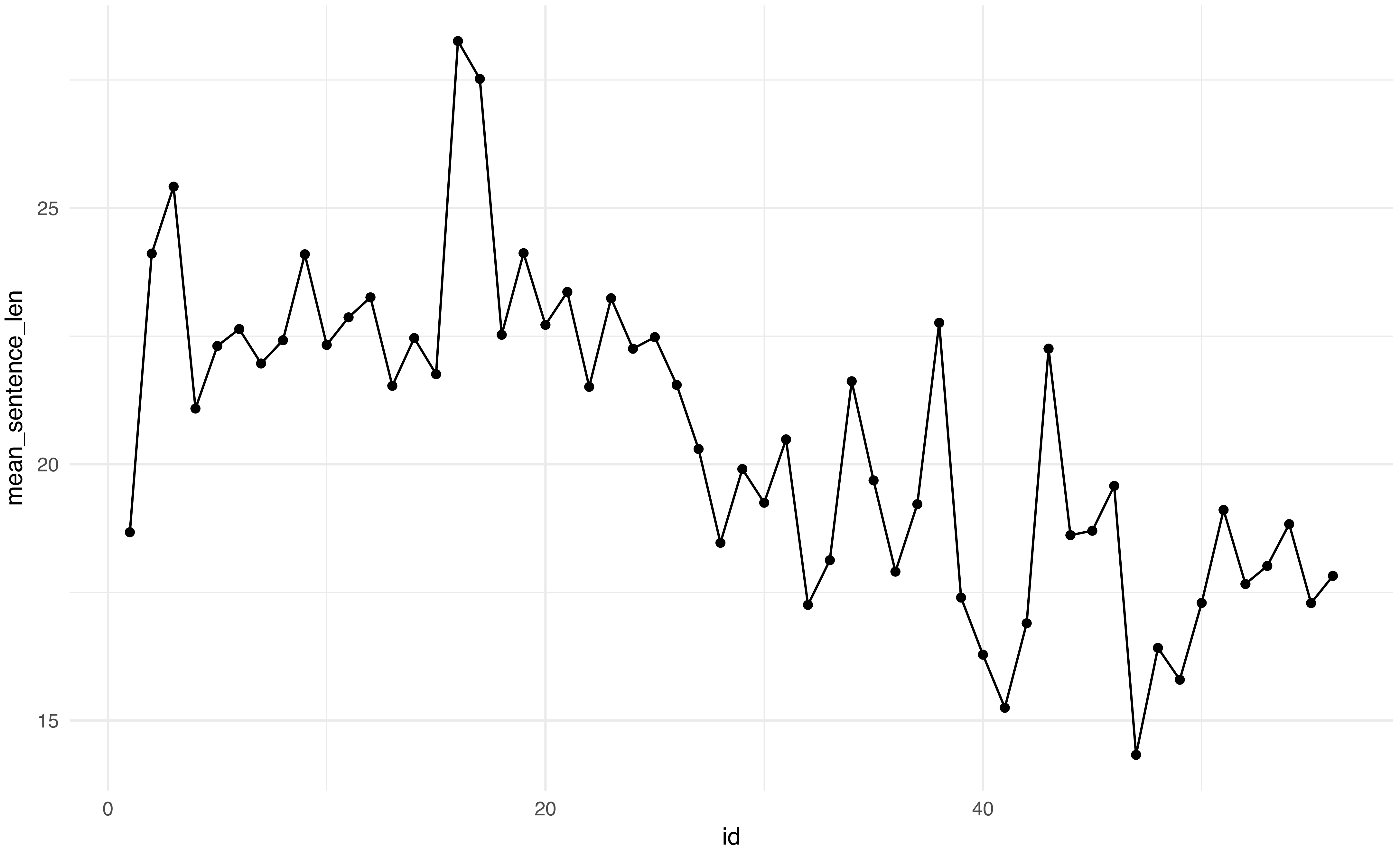
## More accurate average sentence length

By using the sentence boundaries learned by the NLP pipeline, we can more accurately count the average length of the sentences in each text.

```
sh_nlp %>%
  group_by(id, sid) %>%
  mutate(sentence_end = tid == max(tid)) %>%
  group_by(id) %>%
  summarize(mean_sentence_len = n() / sum(sentence_end)) %>%
  ggplot(aes(id, mean_sentence_len)) +
    geom_line() +
    geom_point()
```

Errors that might occur in our original method primarily include abbrevations such as "Dr." and "S.O.S.".

# More accurate average sentence length



## Lemmas

The most simple new column is the one titled `lemma`. This contains a reduced form of the token, for example converting all verbs into the same tense and all nouns into the singular case.

# Lemmas, examples

```r
sh_nlp %>% filter(tolower(word) != lemma) %>%
  select(word, lemma) %>% print(n = 10)
```

```
## # A tibble: 134,920 x 2
##    word     lemma
##    <chr>    <chr>
##  1 she      -PRON-
##  2 is       be
##  3 I        -PRON-
##  4 heard    hear
##  5 him      -PRON-
##  6 her      -PRON-
##  7 his      -PRON-
##  8 eyes     eye
##  9 she      -PRON-
## 10 eclipses eclipse
## # ... with 1.349e+05 more rows
```

# Using lemmas

While minor, this assists with the topic discovery we did in the previous session by using the lemma frequency rather than the word frequency.

```
sh_lemmafr <- sh_nlp %>%
  left_join(word_frequency, by = c("lemma" = "word")) %>%
  filter(!is.na(frequency)) %>%
  filter(frequency < 0.01) %>%
  filter((tolower(word) == word)) %>%
  count(id, lemma) %>%
  group_by(id) %>%
  top_n(n = 10, n) %>%
  left_join(sh_meta, by = "id") %>%
  arrange(id, desc(n))
```

# Using lemmas

```
sh_lemmafr %>% filter(id == 1) %>% print(n = 12)
```

```
## # A tibble: 10 x 4
##       id lemma          n story
##    <int> <chr>      <int> <chr>
##  1     1 photograph    21 a_scandal_in_bohemia
##  2     1 cry           15 a_scandal_in_bohemia
##  3     1 door          13 a_scandal_in_bohemia
##  4     1 minute        13 a_scandal_in_bohemia
##  5     1 eye           11 a_scandal_in_bohemia
##  6     1 hear          11 a_scandal_in_bohemia
##  7     1 lady          10 a_scandal_in_bohemia
##  8     1 rush          10 a_scandal_in_bohemia
##  9     1 gentleman      9 a_scandal_in_bohemia
## 10     1 throw          9 a_scandal_in_bohemia
```

## Using POS tags

Many of the tricks we used in the last session revolved around finding ways to approximate part of speech tags:

- stop words list, for example, removes (amongst other things) punctuation marks, pronouns, conjunctions, and interjections
- checking for upper case marks is really a hunt to identify proper nouns
- the frequency table is largely trying to remove verbs (there are far fewer of these and they tend to be more common), as well as common nouns

Proper part of speech tags can let us do these things more accurately as well as make other types of analysis possible.

# POS granularity

In primary or secondary school, you probably learned about a dozen or so parts of speech. These include nouns, verbs, adjectives, and so forth. Linguists in fact identify a far more granular set of part of speech tags, and even amongst themselves do not agree on a fixed set of such tags.

A commonly used one, and the one implemented by spaCy, are the Penn Treebank codes. These are given in our dataset under the `pos` variable.

# Penn Treebank

**Table 2**
The Penn Treebank POS tagset.

| | | | | | |
|---|---|---|---|---|---|
| 1. | CC | Coordinating conjunction | 25. | TO | *to* |
| 2. | CD | Cardinal number | 26. | UH | Interjection |
| 3. | DT | Determiner | 27. | VB | Verb, base form |
| 4. | EX | Existential *there* | 28. | VBD | Verb, past tense |
| 5. | FW | Foreign word | 29. | VBG | Verb, gerund/present participle |
| 6. | IN | Preposition/subordinating conjunction | 30. | VBN | Verb, past participle |
| 7. | JJ | Adjective | 31. | VBP | Verb, non-3rd ps. sing. present |
| 8. | JJR | Adjective, comparative | 32. | VBZ | Verb, 3rd ps. sing. present |
| 9. | JJS | Adjective, superlative | 33. | WDT | *wh*-determiner |
| 10. | LS | List item marker | 34. | WP | *wh*-pronoun |
| 11. | MD | Modal | 35. | WP$ | Possessive *wh*-pronoun |
| 12. | NN | Noun, singular or mass | 36. | WRB | *wh*-adverb |
| 13. | NNS | Noun, plural | 37. | # | Pound sign |
| 14. | NNP | Proper noun, singular | 38. | $ | Dollar sign |
| 15. | NNPS | Proper noun, plural | 39. | . | Sentence-final punctuation |
| 16. | PDT | Predeterminer | 40. | , | Comma |
| 17. | POS | Possessive ending | 41. | : | Colon, semi-colon |
| 18. | PRP | Personal pronoun | 42. | ( | Left bracket character |
| 19. | PP$ | Possessive pronoun | 43. | ) | Right bracket character |
| 20. | RB | Adverb | 44. | ″ | Straight double quote |
| 21. | RBR | Adverb, comparative | 45. | ‘ | Left open single quote |
| 22. | RBS | Adverb, superlative | 46. | “ | Left open double quote |
| 23. | RP | Particle | 47. | ’ | Right close single quote |
| 24. | SYM | Symbol (mathematical or scientific) | 48. | ” | Right close double quote |

# Universal part of speech

Work has also been done to map these granular codes to language-agnostic codes known as universal parts of speech. Coincidentally, these universal parts of speech mimic those commonly taught in schools:

- *VERB*: verbs (all tenses and modes)
- *NOUN*: nouns (common and proper)
- *PRON*: pronouns
- *ADJ*: adjectives
- *ADV*: adverbs
- *ADP*: adpositions (prepositions and postpositions)
- *CONJ*: conjunctions
- *DET*: determiners
- *NUM*: cardinal numbers
- *PRT*: particles or other function words
- *X*: other: foreign words, typos, abbreviations
- .: punctuation

These are contained in the variable `upos`, and for today will be the most useful for our analysis.

# Top characters, again

Here, for example, is the analysis of key characters with our trick replaced by filtering on the proper noun tag “PROPN”:

```
sh_topchar <- sh_nlp %>%
  filter(upos == "PROPN") %>%
  count(id, word) %>%
  group_by(id) %>%
  top_n(n = 10, n) %>%
  left_join(sh_meta, by = "id") %>%
  arrange(id, desc(n))
```

# Top characters, again

```
sh_topchar %>% filter(id == 1) %>% print(n = Inf)
```

```
## # A tibble: 12 x 4
##         id word         n story
##      <int> <chr>    <int> <chr>
##  1       1 Holmes      48 a_scandal_in_bohemia
##  2       1 Majesty     18 a_scandal_in_bohemia
##  3       1 Irene       14 a_scandal_in_bohemia
##  4       1 Adler       13 a_scandal_in_bohemia
##  5       1 Briony      11 a_scandal_in_bohemia
##  6       1 King        11 a_scandal_in_bohemia
##  7       1 Lodge       11 a_scandal_in_bohemia
##  8       1 Sherlock    11 a_scandal_in_bohemia
##  9       1 Mr.          9 a_scandal_in_bohemia
## 10       1 Bohemia      7 a_scandal_in_bohemia
## 11       1 Norton       7 a_scandal_in_bohemia
## 12       1 Street       7 a_scandal_in_bohemia
```

## Compound words (optional)

A major shortcoming in our tabulation of proper nouns is that many of the proper nouns, in fact most in this case, are actually compound words. The proper way to analyse this data would be to collapse the compound words into a single combined token. It is relatively easy to do this in a slow way with loops. A fast, vectorized method with **dplyr** verbs is show in the code chunk below:

```
sh_compound <- sh_nlp %>%
  filter(upos == "PROPN") %>%
  group_by(id, sid) %>%
  mutate(d = tid - lag(tid) - 1) %>%
  mutate(d = ifelse(is.na(d), 1, d)) %>%
  ungroup() %>%
  mutate(d = cumsum(d)) %>%
  group_by(d) %>%
  summarize(id = first(id), sid = first(sid),
            tid = first(tid),
            thing = stri_c(word, collapse = " ")) %>%
  select(-d) %>%
  inner_join(sh_nlp, by = c("id", "sid", "tid"))
```

# Compound words (optional)

```
sh_compound %>% select(id, thing) %>% print(n = 10)
```

```
## # A tibble: 12,793 x 2
##       id thing
##    <int> <chr>
##  1     1 Sherlock Holmes
##  2     1 Irene Adler
##  3     1 Irene Adler
##  4     1 Holmes
##  5     1 Holmes
##  6     1 Baker Street
##  7     1 Odessa
##  8     1 Trepoff
##  9     1 Atkinson
## 10     1 Trincomalee
## # ... with 1.278e+04 more rows
```

## Entities

The task of finding characters, places, and other references to proper objects is common enough that it has been wrapped up into a specific annotation task known as named entity recognition (NER). Here are the first few entities from the annotation of our stories

```
results <- sh_nlp %>%
  select(id, entity, entity_type) %>%
  filter(!is.na(entity))
```

# Entities, cont.

```r
results %>% print(n = 10)
```

```
## # A tibble: 18,939 x 3
##        id entity          entity_type
##     <int> <chr>           <chr>
##  1      1 Sherlock Holmes PERSON
##  2      1 Irene Adler     PERSON
##  3      1 one             CARDINAL
##  4      1 Grit            FAC
##  5      1 one             CARDINAL
##  6      1 one             CARDINAL
##  7      1 Irene Adler     PERSON
##  8      1 Holmes          PERSON
##  9      1 first           ORDINAL
## 10      1 Holmes          PERSON
## # ... with 1.893e+04 more rows
```

# NER characters

One benefit of this is that NER distinguishes between people and places, making our tabulation even more accurate:

```
sh_nerchar <- sh_nlp %>%
  select(id, entity, entity_type) %>%
  filter(!is.na(entity)) %>%
  filter(entity_type == "PERSON") %>%
  count(id, entity) %>%
  group_by(id) %>%
  top_n(n = 10, n) %>%
  left_join(sh_meta, by = "id") %>%
  arrange(id, desc(n))
```

```
sh_nerchar <- ungroup(sh_nerchar)
```

# NER characters, cont.

```
sh_nerchar %>% filter(id == 1) %>% print(n = Inf)
```

```
## # A tibble: 10 x 4
##       id entity              n story
##    <int> <chr>           <int> <chr>
##  1     1 Holmes             36 a_scandal_in_bohemia
##  2     1 Irene Adler        11 a_scandal_in_bohemia
##  3     1 Sherlock Holmes     9 a_scandal_in_bohemia
##  4     1 Briony Lodge        8 a_scandal_in_bohemia
##  5     1 Watson              6 a_scandal_in_bohemia
##  6     1 Godfrey Norton      4 a_scandal_in_bohemia
##  7     1 Your Majesty        4 a_scandal_in_bohemia
##  8     1 Temple              3 a_scandal_in_bohemia
##  9     1 Adler               2 a_scandal_in_bohemia
## 10     1 Count Von Kramm     2 a_scandal_in_bohemia
```

## Other entity categories

There are many other categories of named entities available within the spaCy and CoreNLP libraries, including:

- *ORGA*: Companies, agencies, institutions, etc.
- *MONEY*: Monetary values, including unit.
- *PERCENT*: Percentages.
- *DATE*: Absolute or relative dates or periods.
- *TIME*: Times smaller than a day.
- *NORP*: Nationalities or religious or political groups.
- *FACILITY*: Buildings, airports, highways, bridges, etc.
- *GPE*: Countries, cities, states.
- *LOC*: Non-GPE locations, mountain ranges, bodies of water.
- *PRODUCT*: Objects, vehicles, foods, etc. (Not services.)
- *EVENT*: Named hurricanes, battles, wars, sports events, etc.
- *WORK_OF_ART*: Titles of books, songs, etc.
- *LANGUAGE*: Any named language.
- *QUANTITY*: Measurements, as of weight or distance.
- *ORDINAL*: "first", "second", etc.
- *CARDINAL*: Numerals that do not fall under another type.

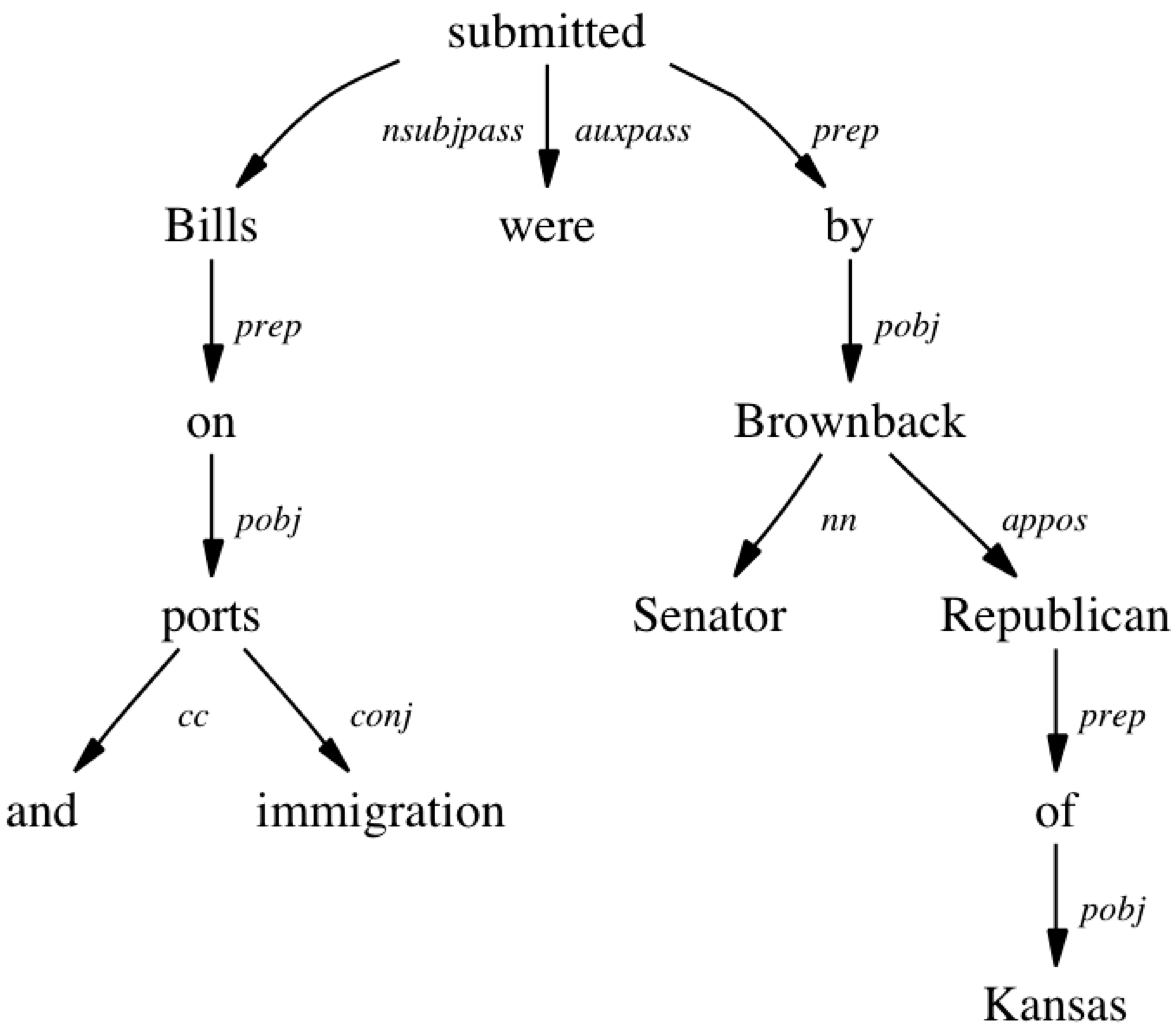
How might these be useful in various textual analyses?

# Dependencies

Dependencies are links between tokens within a sentence that indicate grammatical relationships.

For example, they link adjectives to the nouns they describe and adverbs to the verbs they modify. One of the most common dependencies is the direct object tag “dobj”, linking a verb to the noun that receives the action of the verb.

## Fully parsed sentence



# Dependencies, example

```
sh_nlp %>% filter(id == 1, sid == 1) %>%
  select(word, source, relation, word_source)
```

```
## # A tibble: 9 x 4
##   word      source relation word_source
##   <chr>      <int> <chr>    <chr>
## 1 To             0 ROOT     ROOT
## 2 Sherlock       4 compound Holmes
## 3 Holmes         2 pobj     To
## 4 she            6 nsubj    is
## 5 is             2 ccomp    To
## 6 always         6 advmod   is
## # ... with 3 more rows
```

# What are characters doing?

One way that dependencies can be useful is by determining which verbs are associated with each character by way of the 'nsubj' relation. Amongst other things, this can help identify sentiment, biases, and power dynamics.

In our corpus, we can use the 'nsubj' tag to identify verbs associated with our main characters:

```
sh_whatchar <- sh_nlp %>%
  filter(relation == "nsubj") %>%
  filter(upos == "PROPN") %>%
  count(id, word, lemma_source) %>%
  filter(n > 1)
```

# What are characters doing?

```
sh_whatchar %>% print(n = 12)
```

```
## # A tibble: 344 x 4
##       id word         lemma_source     n
##    <int> <chr>        <chr>        <int>
##  1     1 Holmes       murmur           2
##  2     1 Holmes       say              8
##  3     2 Holmes       remark           2
##  4     2 Holmes       say             11
##  5     2 I.           say              2
##  6     2 Merryweather be               2
##  7     2 Ross         be               3
##  8     2 Spaulding    say              2
##  9     2 Wilson       be               2
## 10     2 Wilson       say              4
## 11     3 Angel        come             2
## 12     3 Holmes       remark           2
## # ... with 332 more rows
```

## Packages

To run the following code, we again make sure that the following packages are loaded (and installed) the following packages.

```
library(cleanNLP)
library(dplyr)
library(readr)
library(stringi)
library(ggplot2)
library(topicmodels)
library(glmnet)
```

```
## Warning: package 'glmnet' was built under R version 3.4.4
```

```
library(ggrepel)
library(viridis)
```

```
## Warning: package 'viridis' was built under R version 3.4.4
```

```
library(magrittr)
theme_set(theme_minimal())
```

You will also need to download and set-up the tutorial's datasets.

# The Data

The President of the United States is constitutionally obligated to provide a report known as the 'State of the Union'. The report summarizes the current challenges facing the country and the president's upcoming legislative agenda.

We have run the spaCy NLP pipeline over this corpus and provide the output data in the GitHub repository.

```
sotu_nlp <- read_csv("data/sotu.csv.gz")
sotu_meta <- read_csv("data/sotu_meta.csv")
```

# Sentence lengths

Just because we are doing text analysis is no excuse for not doing basic exploratory analysis of our data. What, for example, is the distribution of sentence lengths in the corpus?

```
sotu_nlp %>%
  count(id, sid) %$%
  quantile(n, seq(0,1,0.1))
```

```
##    0%  10%  20%  30%  40%  50%  60%  70%  80%  90% 100%
##     1   11   16   19   23   27   31   37   44   58  681
```

# Common nouns

What are the most common nouns in the corpus?

```
sotu_nlp %>%
  filter(upos == "NOUN") %>%
  count(lemma) %>%
  top_n(n = 40, n) %>%
  use_series(lemma)
```

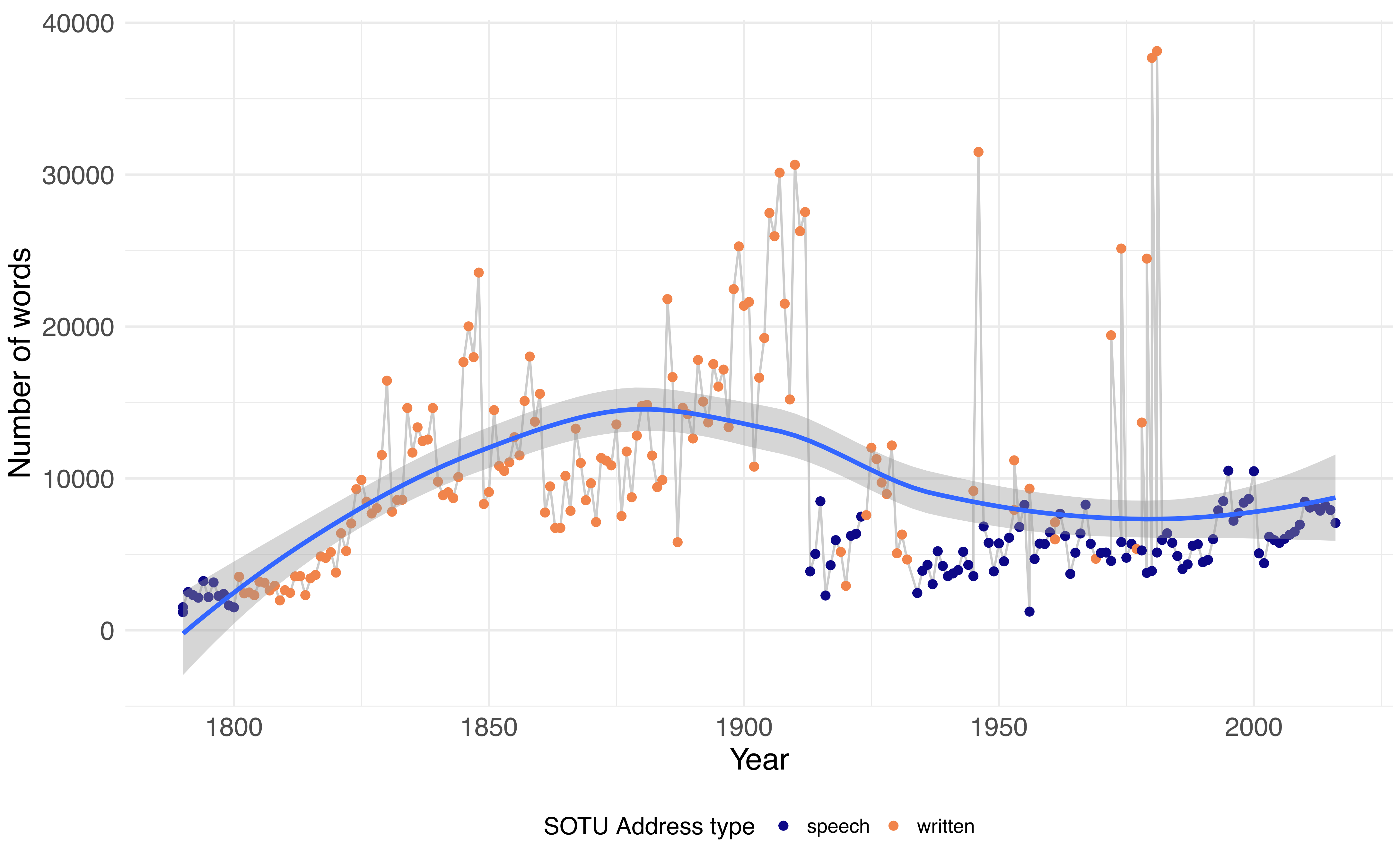
```
##  [1] "act"         "action"      "business"    "citizen"
##  [5] "condition"   "country"     "duty"        "effort"
##  [9] "force"       "government"  "interest"    "land"
## [13] "law"         "legislation" "man"         "measure"
## [17] "nation"      "part"        "peace"       "people"
## [21] "policy"      "power"       "program"     "purpose"
## [25] "question"    "right"       "service"     "state"
## [29] "subject"     "system"      "tax"         "time"
## [33] "treaty"      "war"         "way"         "what"
## [37] "who"         "work"        "world"       "year"
```

## Length in words

Now, how long is each State of the Union in words? Does this differ based on whether it was given as a speech or a written document?

```
sotu_nlp %>%
  count(id) %>%
  group_by(id) %>%
  left_join(sotu_meta, by = "id") %>%
  ggplot(aes(year, n)) +
    geom_line(color = grey(0.8)) +
    geom_point(aes(color = sotu_type)) +
    geom_smooth()
```

# Length in words



# Summarising with dependencies

A straightforward way of extracting a high-level summary of the content of a speech is to extract all direct object object dependencies where the target noun is not a very common word.

Here is an example of this using the first address made by George W. Bush in 2001:

```
summary_2001 <- sotu_nlp %>%
  left_join(sotu_meta, by = "id") %>%
  filter(year == 2001, relation == "dobj") %>%
  left_join(word_frequency, by = "word") %>%
  filter(frequency < 0.001) %>%
  select(id, word, word_source) %$%
  sprintf("%s => %s", word_source, word)
```

# George W. Bush (2001)

```r
summary_2001
```

```
##  [1] "take => oath"                "increasing => layoffs"
##  [3] "buying => prescriptions"     "protects => trillion"
##  [5] "makes => welcoming"          "accelerating => cleanup"
##  [7] "fight => homelessness"       "allowing => taxpayers"
##  [9] "provide => mentor"           "fight => illiteracy"
## [11] "promotes => compassion"      "end => profiling"
## [13] "stopping => abuses"          "pay => trillion"
## [15] "throw => darts"              "restores => fairness"
## [17] "restructure => defenses"     "promoting => internationalism"
## [19] "makes => downpayment"        "deploy => defenses"
## [21] "discard => relics"           "confronting => shortage"
## [23] "sound => footing"            "bridge => divides"
## [25] "minding => manners"          "divided => conscience"
## [27] "done => servants"
```

# George W. Bush (2002)

```r
head(summary_2002, 34)
```

```
##  [1] "faces => dangers"              "urged => followers"
##  [3] "brought => sorrow"             "found => diagrams"
##  [5] "hold => hostages"              "eliminate => parasites"
##  [7] "prevent => regimes"            "flaunt => hostility"
##  [9] "develop => anthrax"            "kicked => inspectors"
## [11] "match => hatred"               "attack => allies"
## [13] "deploy => defenses"            "permit => regimes"
## [15] "increased => vigilance"        "develop => vaccines"
## [17] "fight => anthrax"              "expand => patrols"
## [19] "track => arrivals"             "mean => neighborhoods"
## [21] "thank => attendants"           "defeat => recession"
## [23] "want => paycheck"              "set => posturing"
## [25] "reduce => dependency"          "offer => dignity"
## [27] "enact => safeguards"           "keeping => commitments"
## [29] "saw => selves"                 "embracing => ethic"
## [31] "extending => compassion" "extend => compassion"
## [33] "await => knock"                "owns => aspirations"
```

# Woodrow Wilson (1919)

```r
head(summary_1919,  34)
```

```
##  [1] "save => inconvenience"        "produce => stagnation"
##  [3] "produce => stagnation"        "made => interruption"
##  [5] "keep => armies"               "-and => necessaries"
##  [7] "arrive => permitting"         "urge => necessity"
##  [9] "produced => bitterness"       "remove => grievances"
## [11] "produces => dissatisfaction" "stir => disturbances"
## [13] "shown => willingness"         "bring => democratization"
## [15] "analyze => particulars"       "bid => pause"
## [17] "saps => vitality"             "treat => manifestations"
## [19] "touch => tissues"             "come => unrest"
## [21] "settle => disputes"           "devise => tribunal"
## [23] "lose => composure"            "realize => fruition"
```

## NLP and matrices

So far, we have done all of our analysis using a data frame where each token is given its own row. For modelling purposes, we often want to calculate the term frequency matrix. This matrix has one row per document and one column per unique token in the data set (although we can limit which tokens actually have a column).

Conveniently, **cleanNLP** provides the function `get_tfidf` for calculated this matrix.

## Document term frequency matrix

$$
\begin{array}{r c}
 & \begin{array}{cccccc} \text{I} & \text{and} & \dots & \text{commute} & \dots & \text{lol} \end{array} \\
\begin{array}{r} \text{Text \#0001} \\ \text{Text \#0002} \\ \text{Text \#0003} \\ \vdots \\ \text{Text \#9500} \end{array} &
\begin{pmatrix}
20 & 55 & \ldots & 0 & \ldots & 0 \\
34 & 72 & \ldots & 5 & \ldots & 0 \\
6 & 34 & \ldots & 0 & \ldots & 4 \\
\vdots & \vdots & \ldots & \vdots & \ddots & \vdots \\
150 & 87 & \ldots & 0 & \ldots & 30
\end{pmatrix}
\end{array}
$$

# get_tfidf()

Here we will construct a term frequency matrix from only non-proper nouns:

```
sotu_tfidf <- sotu_nlp %>%
  filter(pos %in% c("NN", "NNS")) %>%
  get_tfidf(min_df = 0.05, max_df = 0.95,
                     type = "tfidf", tf_weight = "dnorm")
```

```
## NOTE: get_tfidf has been renamed cnlp_get_tfidf
```

```
## NOTE: returning legacy output format from get_tfidf
```

## get_tfidf()

The output is a list with three elements: the term frequency inverse document frequency matrix, the ids of the documents corresponding to row names, and the vocabulary corresponding to the column names.

```
head(sotu_tfidf$vocab, 20)
```

```
##  [1] "world"       "citizen"     "service"     "duty"
##  [5] "system"      "right"       "man"         "program"
##  [9] "policy"      "work"        "act"         "condition"
## [13] "subject"     "legislation" "force"       "effort"
## [17] "treaty"      "purpose"     "land"        "business"
```

```
head(sotu_tfidf$id, 20)
```

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

```
dim(sotu_tfidf$tfidf)
```

```
## [1]  236 2356
```

## PCA

What specifically can we do with this data? As a starting point, we will compute the principal components of the matrix. While base-R has great functions for doing this, we'll make use of the **cleanNLP** function `tidy_pca` which returns a data frame that makes plotting in **ggplot2** easier:

```
sotu_pca <- tidy_pca(sotu_tfidf$tfidf, sotu_meta)
```

```
## NOTE: tidy_pca has been renamed cnlp_pca
```

```
select(sotu_pca, president, party, PC1, PC2)
```

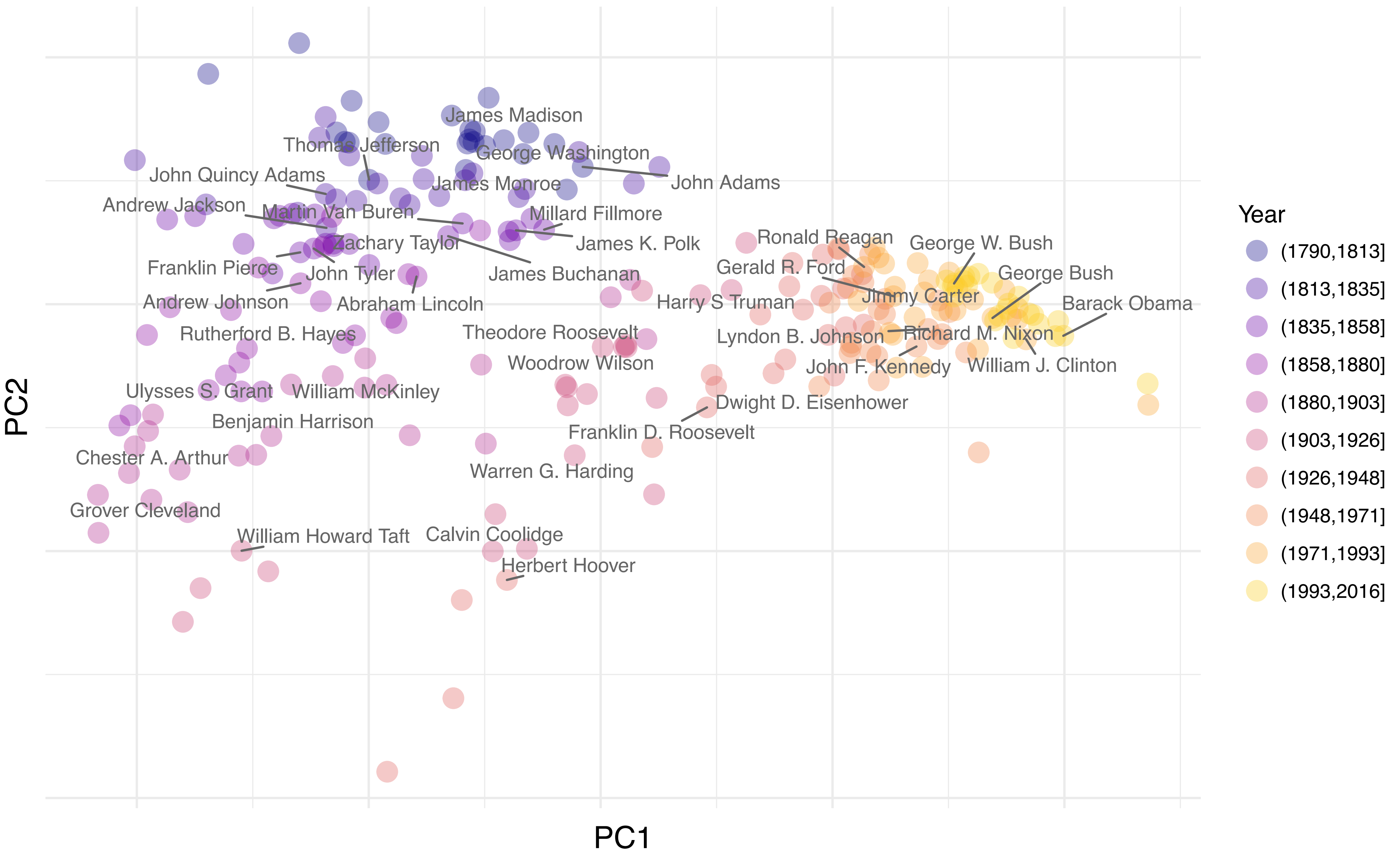
```
## # A tibble: 236 x 4
##   president         party          PC1   PC2
##   <chr>             <chr>        <dbl> <dbl>
## 1 George Washington Nonpartisan - 1.99  13.0
## 2 George Washington Nonpartisan - 4.83  16.7
## 3 George Washington Nonpartisan - 5.74  13.0
## 4 George Washington Nonpartisan - 3.34  12.2
## 5 George Washington Nonpartisan -16.9   18.7
## 6 George Washington Nonpartisan - 5.66  13.3
## # ... with 230 more rows
```

# PCA plot

While a simple scatter plot of this is easy to construct, we can tweak some of the default settings to get a really nice visualization of where each President's speeches cluster:

```
ggplot(sotu_pca, aes(PC1, PC2)) +
  geom_point(aes(color = cut(year, 10, dig.lab = 4))) +
  geom_text(data = filter(sotu_pca, !duplicated(president))
```

# PCA plot, cont.



# Topic Models

Topic models are a collection of statistical models for describing abstract themes within a textual corpus. Each theme is characterized by a collection of words that commonly co-occur; for example, the words ‘crop’, ‘dairy’, ‘tractor’, and ‘hectare’, might define a *farming* theme.

One of the most popular topic models is latent Dirichlet allocation (LDA), a Bayesian model where each topic is described by a probability distribution over a vocabulary of words. Each document is then characterized by a probability distribution over the available topics.

# LDA

To fit LDA on a corpus of text parsed by the **cleanNLP** package, the output of `get_tfidf` can be piped directly to the `LDA` function in the package **topicmodels**. The topic model function requires raw counts, so the type variable in `get_tfidf` is set to “tf”.

```
sotu_tf <- sotu_nlp %>%
  filter(pos %in% c("NN", "NNS")) %>%
  get_tfidf(min_df = 0.05, max_df = 0.95,
                    type = "tf", tf_weight = "raw")
tm <- LDA(sotu_tf$tf, k = 16, control = list(verbose = 1))
```

```
## NOTE: get_tfidf has been renamed cnlp_get_tfidf
```

```
## NOTE: returning legacy output format from get_tfidf
```

# Describing topics

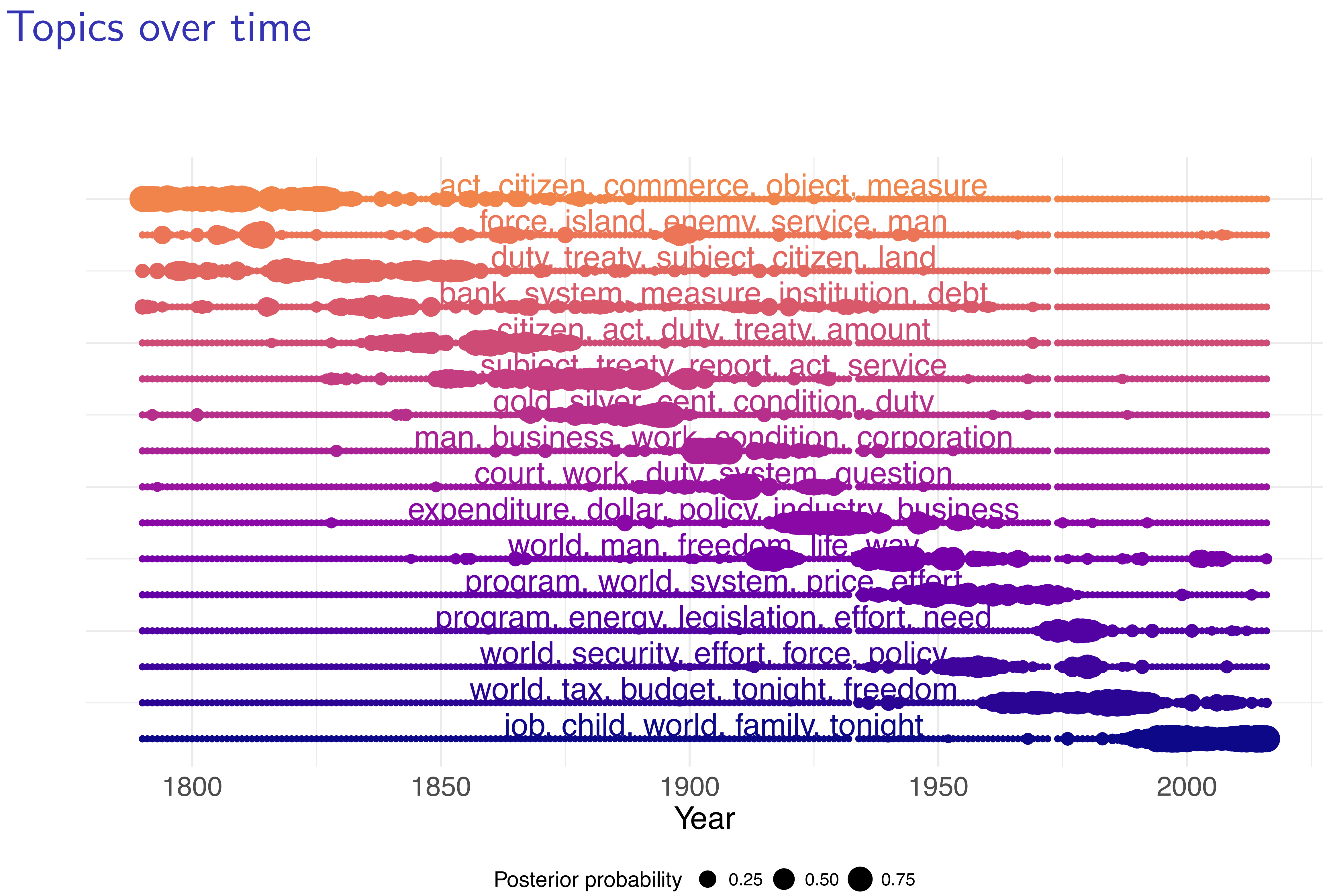
We can describe each topic by giving the five most important words in each topic:

```
terms <- posterior(tm)$terms
topics <- posterior(tm)$topics
topic_df <- data_frame(topic = as.integer(col(topics)),
                       id = sotu_meta$id[as.integer(row(topics))],
                       val = as.numeric(topics)) %>%
              left_join(sotu_meta, by = "id")
top_terms <- apply(terms, 1,
               function(v) paste(sotu_tf$vocab[order(v,
                  decreasing = TRUE)[1:5]], collapse = ", "))
top_terms <- as.character(top_terms)
```

# Describing topics

```
top_terms
```

```
##  [1] "act, citizen, commerce, object, measure"
##  [2] "man, business, work, condition, corporation"
##  [3] "duty, treaty, subject, citizen, land"
##  [4] "world, man, freedom, life, way"
##  [5] "bank, system, measure, institution, debt"
##  [6] "program, world, system, price, effort"
##  [7] "world, tax, budget, tonight, freedom"
##  [8] "job, child, world, family, tonight"
##  [9] "world, security, effort, force, policy"
## [10] "program, energy, legislation, effort, need"
## [11] "citizen, act, duty, treaty, amount"
## [12] "expenditure, dollar, policy, industry, business"
## [13] "court, work, duty, system, question"
## [14] "gold, silver, cent, condition, duty"
## [15] "force, island, enemy, service, man"
## [16] "subject, treaty, report, act, service"
```



Topics over time
act, citizen, commerce, object, measure
force, island, enemy, service, man
duty, treaty, subject, citizen, land
bank, system, measure, institution, debt
citizen, act, duty, treaty, amount
subject, treaty, report, act, service
gold, silver, cent, condition, duty
man, business, work, condition, corporation
court, work, duty, system, question
expenditure, dollar, policy, industry, business
world, man, freedom, life, way
program, world, system, price, effort
program, energy, legislation, effort, need
world, security, effort, force, policy
world, tax, budget, tonight, freedom
job, child, world, family, tonight
1800
1850
1900
1950
2000
Year
Posterior probability
0.25
0.50
0.75

# Predictive models

A classifier that distinguishes speeches made by two presidents will be constructed here for the purpose of illustrating the topical and stylistic differences between them and their speech writers.

As a first step, a term-frequency matrix is extracted using the same technique as was used with the topic modeling function. However, here the frequency is computed for each sentence in the corpus rather than the document as a whole.

## ‘George Bush (2001-2008)’

## ‘Barack Obama (2009-2016)’

# Design matrix

The ability to do this seamlessly with a single additional `mutate` function defining a new id illustrates the flexibility of the `get_tfidf` function.

```
df <- sotu_nlp %>%
  left_join(sotu_meta, by = "id") %>%
  filter(president %in% c("Barack Obama", "George W. Bush")) %>%
  mutate(new_id = paste(id, sid, sep = "-")) %>%
  filter(pos %in% c("NN", "NNS"))
mat <- get_tfidf(df, min_df = 0, max_df = 1, type = "tf",
                 tf_weight = "raw", doc_var = "new_id")
```

```
## NOTE: get_tfidf has been renamed cnlp_get_tfidf
```

```
## NOTE: returning legacy output format from get_tfidf
```

# Training and testing sets

It will be necessary to define a response variable y indicating whether this is a speech made by President Obama as well as a training flag indicating which speeches were made in odd numbered years.

```
m2 <- data_frame(new_id = mat$id) %>%
  left_join(df[!duplicated(df$new_id),]) %>%
  mutate(y = as.numeric(president == "Barack Obama")) %>%
  mutate(train = (year %% 2 == 0))
```

# Elastic net

The output may now be used as input to the elastic net function provided by the **glmnet** package. This function fits a model of the form:

$$\beta = \text{argmin}_b \left\{ ||y - Xb||_2 + \lambda \cdot (\alpha)||b||_1 + \lambda \cdot (1-\alpha)||b||_2^2 \right\}$$

Cross-validation is used in order to select the best value of the model's tuning parameter $\lambda$.

```
model <- cv.glmnet(mat$tf[m2$train,], m2$y[m2$train],
                   family = "binomial", alpha = 0.9)
```

The response is set to the binomial family given the binary nature of the response and training is trained on only those speeches occurring in odd-numbered years.

# Predicted probabilities

We can add the predicted probabilites to the dataset `m2` with the following:

```
m2$pred <- predict(model, newx = mat$tf, type = "response",
                          s = model$lambda.1se)
select(m2, new_id, id, sid, president, year, pred)
```

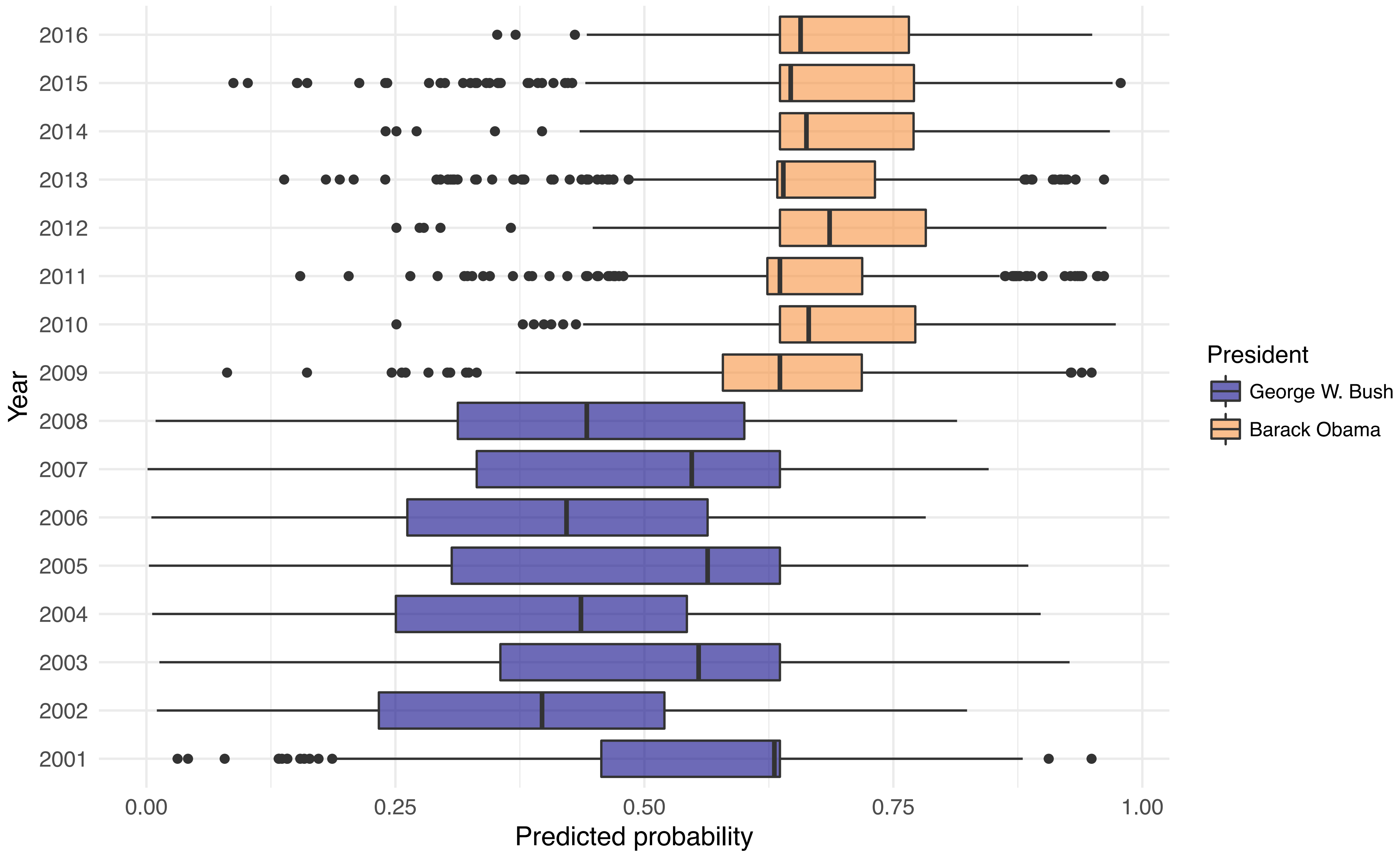
```
## # A tibble: 4,821 x 6
##   new_id    id   sid president          year  pred
##   <chr>  <int> <int> <chr>             <int> <dbl>
## 1 221-1    221     1 George W. Bush  2001 0.598
## 2 221-2    221     2 George W. Bush  2001 0.636
## 3 221-3    221     3 George W. Bush  2001 0.636
## 4 221-4    221     4 George W. Bush  2001 0.784
## 5 221-5    221     5 George W. Bush  2001 0.636
## 6 221-6    221     6 George W. Bush  2001 0.784
## # ... with 4,815 more rows
```

## Predicted probabilities

A boxplot of the predicted classes for each sentence within a speach is a good way of evaluating the model:

```
ggplot(m2, aes(factor(year), pred)) +
  geom_boxplot(aes(fill = president))
```

# Predicted probabilities



## Model coefficients

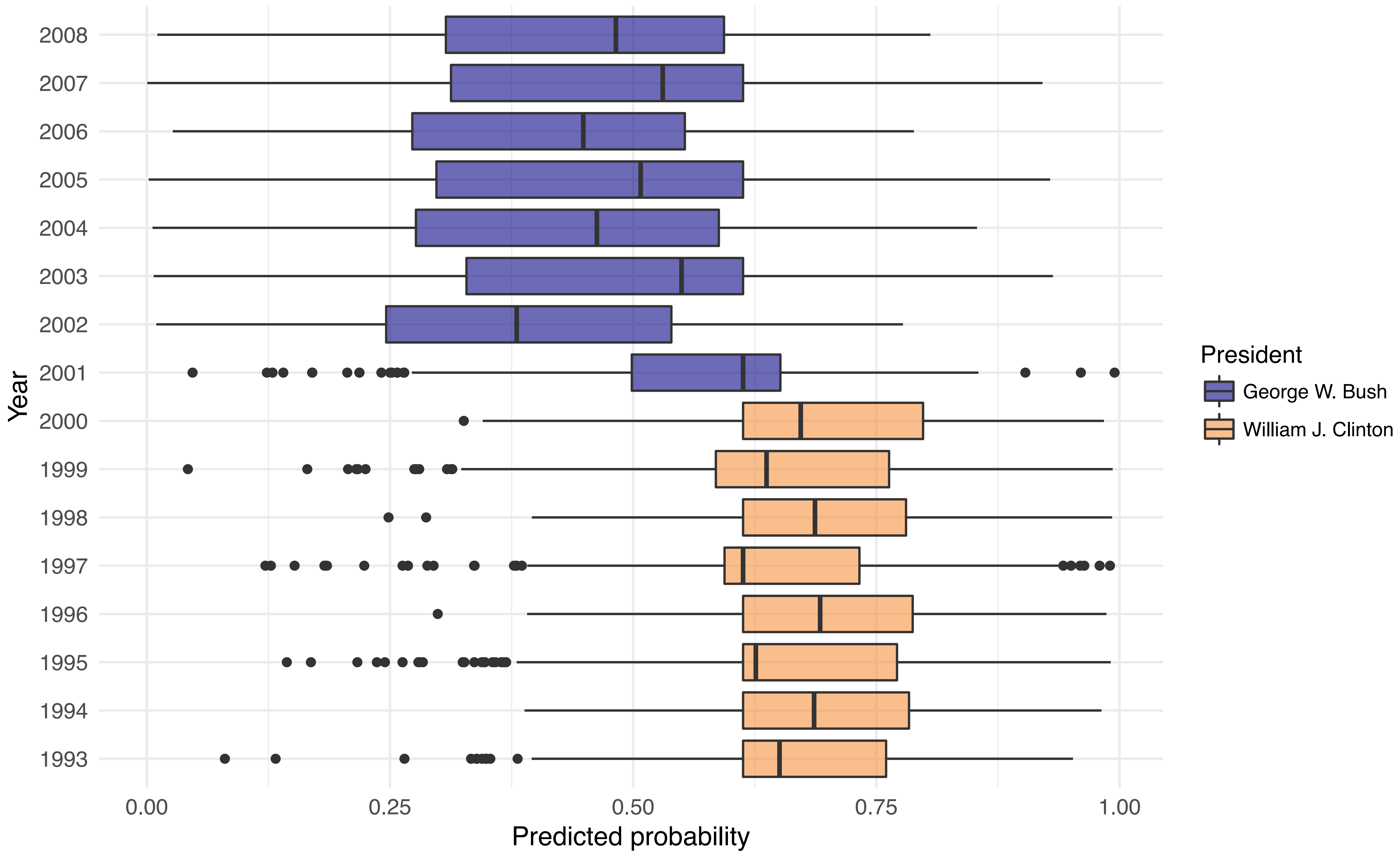
One benefit of the penalized linear regression model is that it is possible to interpret the coefficients in a meaningful way. Here are the non-zero elements of the regression vector, coded as whether the have a positive (more Obama) or negative (more Bush) sign:

```r
beta <- coef(model, s = model[["lambda"]][10])[-1]
sprintf("%s (%d)", mat$vocab, sign(beta))[beta != 0]
```

```
##  [1] "job (1)"            "nation (-1)"        "business (1)"
##  [4] "child (-1)"         "terrorist (-1)"     "freedom (-1)"
##  [7] "college (1)"        "company (1)"        "thing (1)"
## [10] "peace (-1)"         "change (1)"         "enemy (-1)"
## [13] "terror (-1)"        "hope (-1)"          "drug (-1)"
## [16] "kid (1)"            "regime (-1)"        "class (1)"
## [19] "industry (1)"       "member (-1)"        "relief (-1)"
## [22] "liberty (-1)"       "compassion (-1)"    "enforcement (-1)"
## [25] "medicine (-1)"      "death (-1)"         "11th (-1)"
## [28] "homeland (-1)"      "will (-1)"          "character (-1)"
## [31] "culture (-1)"
```

# ‘Bill Clinton (1993-2000)’

# Predicted probabilities (Bush vs. Clinton)

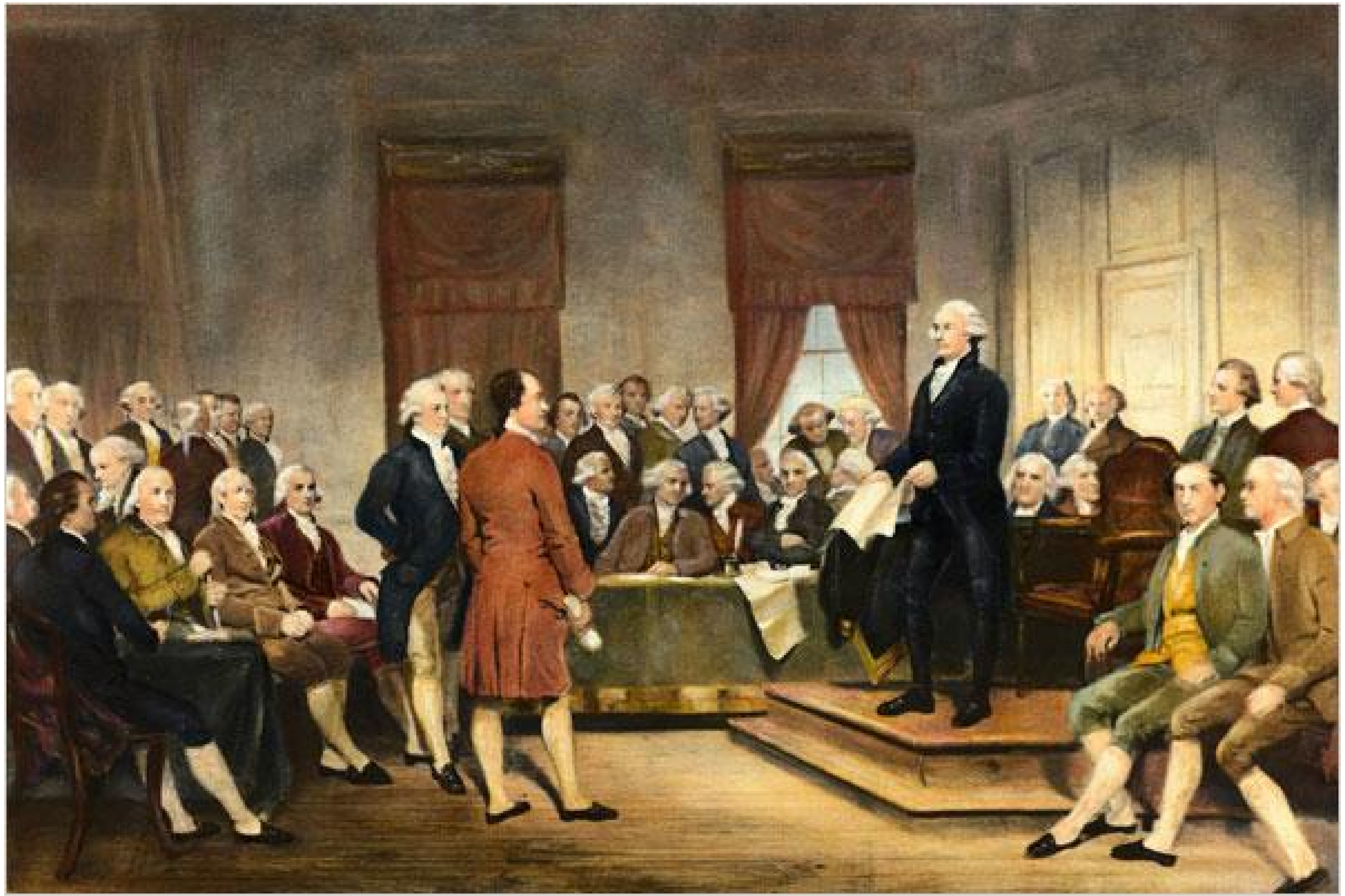


# Model coefficents (Bush vs. Clinton)

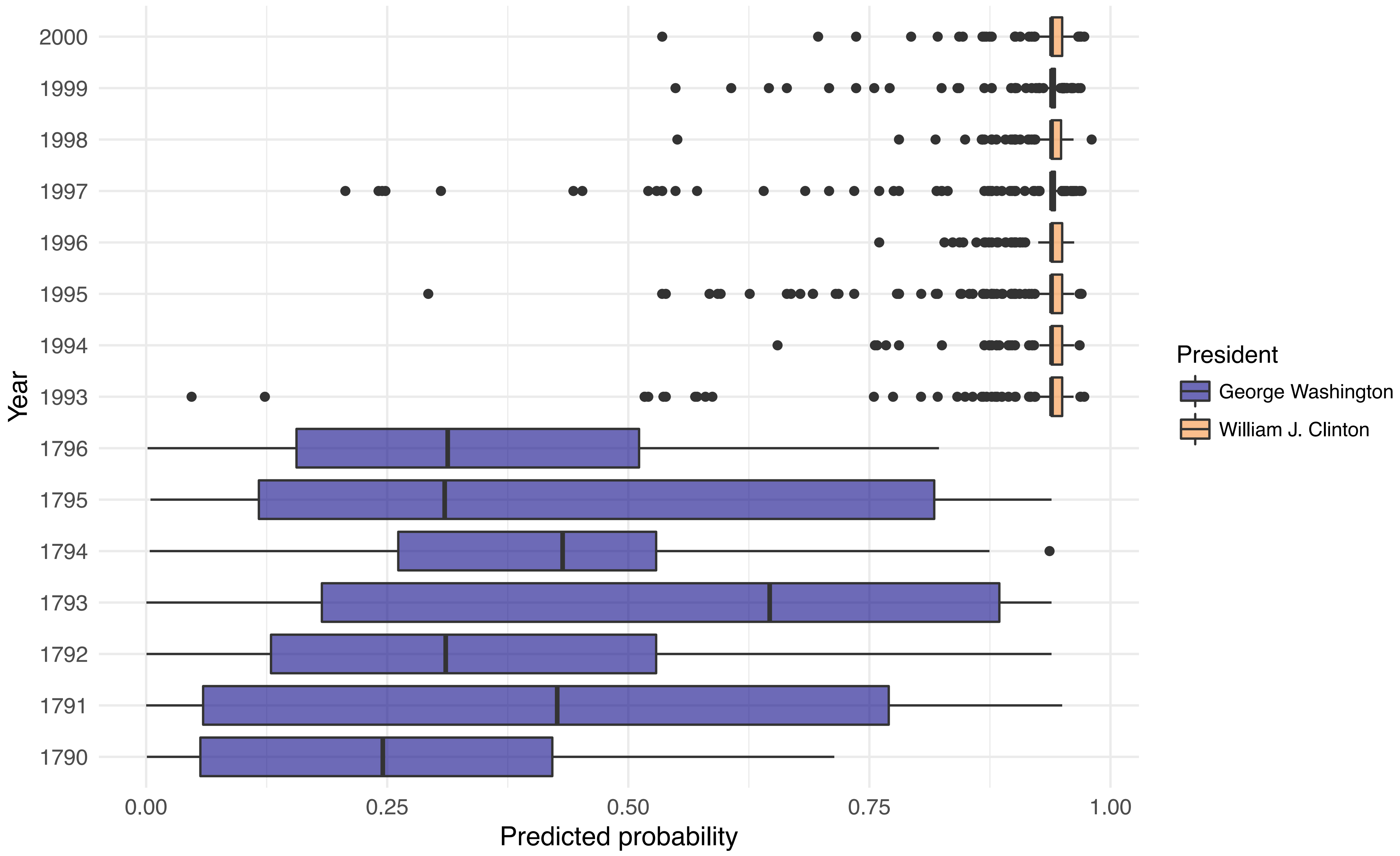
```
beta <- coef(model, s = model[["lambda"]][9])[-1]
sprintf("%s (%d)", mat$vocab, sign(beta))[beta != 0]
```

```
##  [1] "year (1)"          "child (1)"         "family (1)"
##  [4] "care (1)"          "community (1)"     "freedom (-1)"
##  [7] "century (1)"       "parent (1)"        "thing (1)"
## [10] "welfare (1)"       "terrorist (-1)"    "challenge (1)"
## [13] "woman (-1)"        "crime (1)"         "terror (-1)"
## [16] "enemy (-1)"        "hope (-1)"         "something (1)"
## [19] "idea (1)"          "troop (-1)"        "gun (1)"
## [22] "regime (-1)"       "relief (-1)"       "liberty (-1)"
## [25] "danger (-1)"       "attack (-1)"       "institution (-1)"
## [28] "compassion (-1)"   "coalition (-1)"    "dignity (-1)"
## [31] "11th (-1)"         "oil (-1)"          "homeland (-1)"
```

## ‘George Washington (1789-1797)’



# Predicted probabilities (Washington vs. Clinton)



# Model coefficients (Washington vs. Clinton)

```
##  [1] "provision (-1)"     "measure (-1)"       "object (-1)"
##  [4] "attention (-1)"     "mean (-1)"          "session (-1)"
##  [7] "consideration (-1)" "militia (-1)"       "establishment (-1)"
## [10] "circumstance (-1)"  "commerce (-1)"      "satisfaction (-1)"
## [13] "tribe (-1)"         "commissioner (-1)"  "execution (-1)"
## [16] "case (-1)"
```