Additional Linguistic Analysis

load libraries

```r
library(dplyr)
```

```

Attaching package: 'dplyr'

The following objects are masked from 'package:stats':

    filter, lag

The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union
```

```r
library(ggplot2)
library(tidyr)
library(tidytext)
library(tm)
```

```
Loading required package: NLP


Attaching package: 'NLP'

The following object is masked from 'package:ggplot2':

    annotate
```

```r
library(scales)
library(stringr)
```

load data

```r
df_post = read.csv(file = "post_data.csv")
df_comment = read.csv(file = "comment_data.csv")
```

tokenize

"Tokenizing" is a process of breaking text out of strings (which have arbitrary length) into a more meaningful unit, such as individual words. Note: this will greatly increase the number of rows in the dataframe.

```r
post_tokens <- df_post |> unnest_tokens(word, Submission.Text)
```

filter stopwords

```r
post_tokens <- post_tokens |> anti_join(get_stopwords())
```

```
Joining with `by = join_by(word)`
```

```r
## rename word column
colnames(post_tokens)[colnames(post_tokens) == "word"] <- "words"
```

Comparing Word Frequencies of Post Authors

Our text data is focused on reviews of computer hardware. This may lead to similar word choice across authors–brand names and technical terms are likely to reoccur. One way to assess how reviewers use language is to compare the frequencies with which they use words.

```r
# simple frequency count for one author
post_tokens |>
  count(words, sort = T) |>
  head()
```

```
   words   n
1  https 248
2    100 233
3  ryzen 113
```

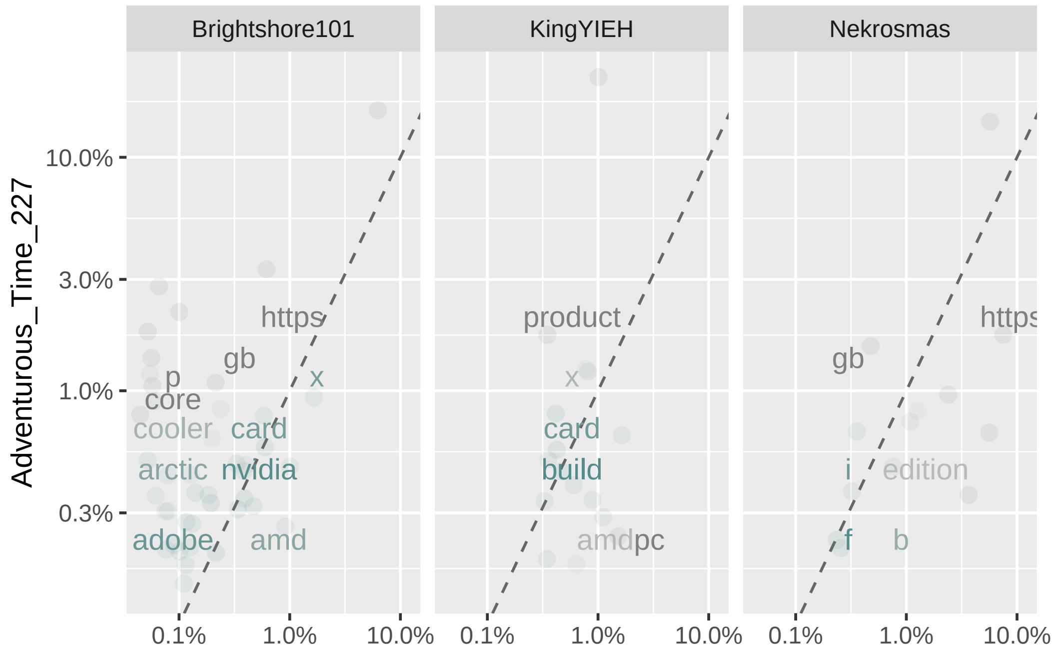
```
4    rtx 112
5   nbsp 103
6 gaming 101
```

```
frequency <- post_tokens |>
  mutate(words = str_extract(words, "[a-z']+")) |>
  count(Author, words) |>
  group_by(Author) |>
  mutate(proportion = n / sum(n)) |>
  select(-n) |>
  filter(Author == "Adventurous_Time_227" | Author == "Brightshore101" | Author == "KingYI
  pivot_wider(names_from = Author, values_from = proportion) |>
  pivot_longer(`Brightshore101`:`Nekrosmas`,
               names_to = "Author", values_to = "proportion")
```

```
# may throw a warning about missing values being removed
ggplot(frequency, aes(x = proportion, y = `Adventurous_Time_227`,
                      color = abs(`Adventurous_Time_227` - proportion))) +
  geom_abline(color = "gray40", lty = 2) +
  geom_jitter(alpha = 0.1, size = 2.5, width = 0.3, height = 0.3) +
  geom_text(aes(label = words), check_overlap = TRUE, vjust = 1.5) +
  scale_x_log10(labels = percent_format()) +
  scale_y_log10(labels = percent_format()) +
  scale_color_gradient(limits = c(0, 0.01),
                       low = "darkslategray4", high = "gray75") +
  facet_wrap(~Author, ncol = 3) +
  theme(legend.position="none") +
  labs(y = "Adventurous_Time_227", x = NULL)
```

```
Warning: Removed 2305 rows containing missing values or values outside the scale range
(`geom_point()`).
```

```
Warning: Removed 2308 rows containing missing values or values outside the scale range
(`geom_text()`).
```



The plots above compare the word usage of one author against three other authors. All four authors were selected at random from the dataset. Words that fall along the dotted line are used with the same frequency by the two authors.

Bigram Sentiment Check

Sentiment Analysis based on individual words sometimes runs into a problem: it can't tell when a word's meaning and sentiment are being altered by neighboring words. A classic example: "good" versus "not good". We can tokenize by 2 words, a "bigram", to assess this.

```
# tokenize by bigrams
df_bigram <- df_post |> unnest_tokens(bigram, Submission.Text, token = "ngrams", n = 2) |>
  filter(!is.na(bigram))

# separate the bigrams into individual words (temporarily)
df_bigram <- df_bigram |>
 separate(bigram, c("word1", "word2"), sep = " ")

# filter out stop words
df_bigram_filtered <- df_bigram |>
  filter(!word1 %in% stop_words$word) |>
```

```r
  filter(!word2 %in% stop_words$word)

# let's see the top few
df_bigram_count <- df_bigram_filtered |>
  count(word1, word2, sort = T)

head(df_bigram_count)
```

```
    word1           word2  n
1 geforce             rtx 81
2   ryzen               7 73
3     amd           ryzen 56
4     rtx            4070 51
5  nvidia         geforce 48
6   https www.youtube.com 38
```

With stopwords filtered out, we can rebuild the bigrams

```r
df_bigram_unite <- df_bigram_filtered |>
  unite(bigram, word1, word2, sep = " ")

head(df_bigram_unite)
```

```
  Submission.ID                           Post.Title
1       17synze Is Ryzen 7 7800x3D really that good?
2       17synze Is Ryzen 7 7800x3D really that good?
3       17synze Is Ryzen 7 7800x3D really that good?
4       17synze Is Ryzen 7 7800x3D really that good?
5       17synze Is Ryzen 7 7800x3D really that good?
6       17synze Is Ryzen 7 7800x3D really that good?
                                                                                  Post.Ul
1 https://www.reddit.com/r/LinusTechTips/comments/17synze/is_ryzen_7_7800x3d_really_that_good
2 https://www.reddit.com/r/LinusTechTips/comments/17synze/is_ryzen_7_7800x3d_really_that_good
3 https://www.reddit.com/r/LinusTechTips/comments/17synze/is_ryzen_7_7800x3d_really_that_good
4 https://www.reddit.com/r/LinusTechTips/comments/17synze/is_ryzen_7_7800x3d_really_that_good
5 https://www.reddit.com/r/LinusTechTips/comments/17synze/is_ryzen_7_7800x3d_really_that_good
6 https://www.reddit.com/r/LinusTechTips/comments/17synze/is_ryzen_7_7800x3d_really_that_good
  Author Score Number.of.Comments Upvote.Ratio        Product.Name
1  ro3rr    39                 85         0.76 AMD Ryzen 7 7800X3D
2  ro3rr    39                 85         0.76 AMD Ryzen 7 7800X3D
```

```
3  ro3rr    39                 85         0.76 AMD Ryzen 7 7800X3D
4  ro3rr    39                 85         0.76 AMD Ryzen 7 7800X3D
5  ro3rr    39                 85         0.76 AMD Ryzen 7 7800X3D
6  ro3rr    39                 85         0.76 AMD Ryzen 7 7800X3D
  Product.Category        Posting.Time            bigram
1        Processor 2023-11-11 17:01:47        main games
2        Processor 2023-11-11 17:01:47 marketing campaign
3        Processor 2023-11-11 17:01:47  sponsored reviews
4        Processor 2023-11-11 17:01:47      cherry picked
5        Processor 2023-11-11 17:01:47       picked games
6        Processor 2023-11-11 17:01:47        wins ignore
```

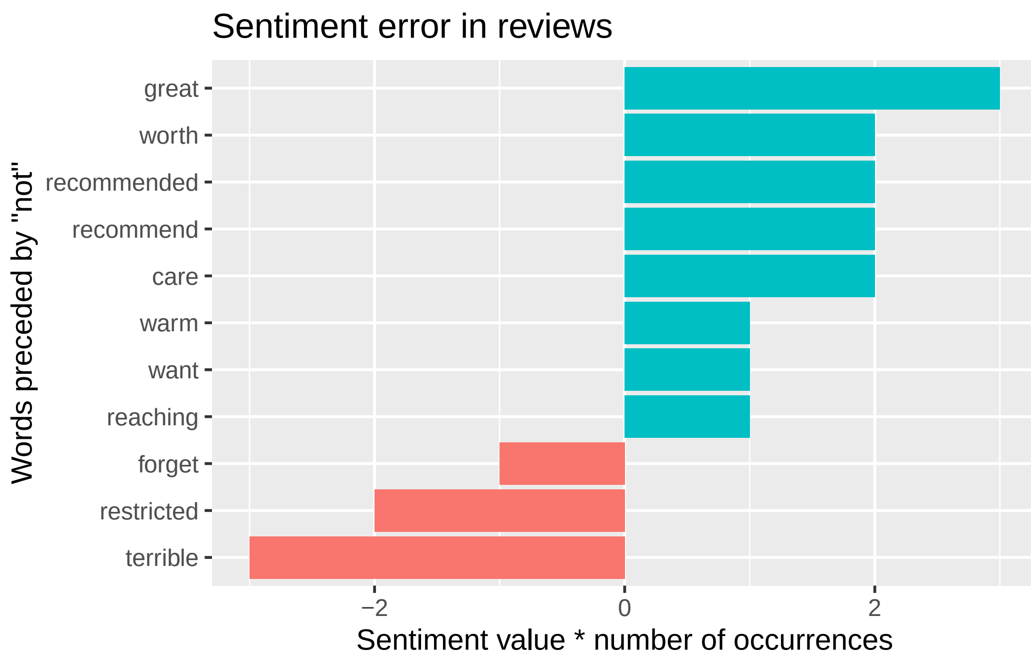
Now to introduce sentiment

```
# get word-sentiment scores
AFINN <- get_sentiments("afinn")

# filter for first word "not", then get sentiment of the second word
not_phrases <- df_bigram |>
  filter(word1 == "not") |>
  inner_join(AFINN, by = c(word2 = "word")) |>
  count(word2, value, sort = T)

not_phrases
```

```
         word2 value n
1         care     2 1
2       forget    -1 1
3        great     3 1
4     reaching     1 1
5    recommend     2 1
6  recommended     2 1
7   restricted    -2 1
8     terrible    -3 1
9         want     1 1
10        warm     1 1
11       worth     2 1
```

```
not_phrases |>
  mutate(contribution = n * value) |>
  arrange(desc(abs(contribution))) |>
```

```
  head(20) |>
  mutate(word2 = reorder(word2, contribution)) |>
  ggplot(aes(n * value, word2, fill = n * value > 0)) +
  geom_col(show.legend = FALSE) +
  labs(x = "Sentiment value * number of occurrences",
       y = "Words preceded by \"not\"",
       title = "Sentiment error in reviews")
```



Does the same hold true for the comments on reviews?

```
# tokenize by bigrams
comment_bigram <- df_comment |> unnest_tokens(bigram, Comment.Body, token = "ngrams", n =
  filter(!is.na(bigram))

# separate the bigrams into individual words (temporarily)
comment_bigram <- comment_bigram |>
 separate(bigram, c("word1", "word2"), sep = " ")
```

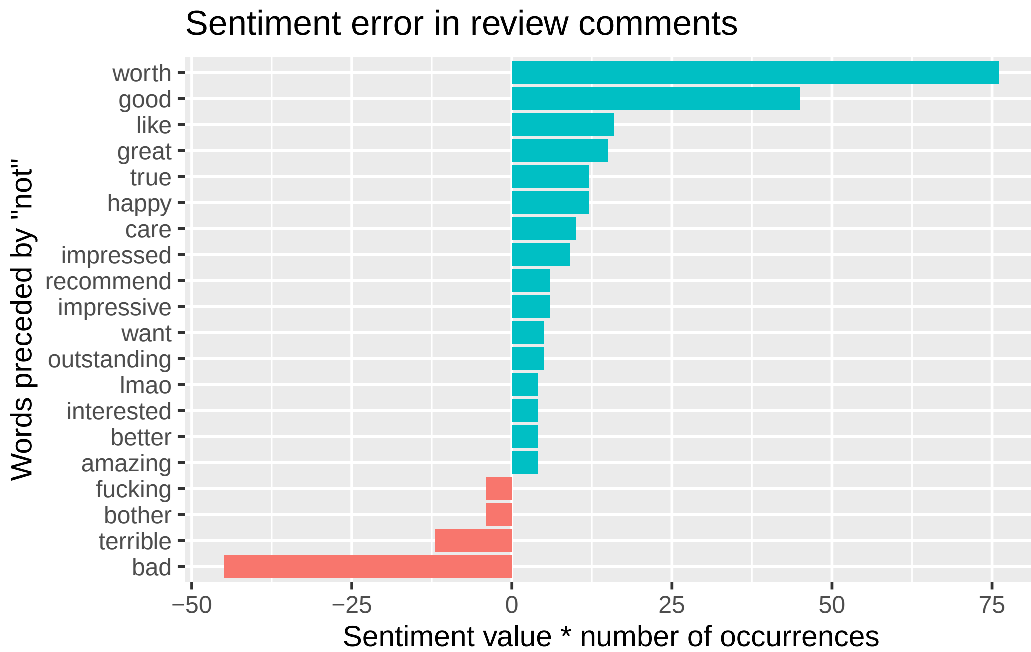
```
# filter for first word "not", then get sentiment of the second word
not_comment <- comment_bigram |>
  filter(word1 == "not") |>
```

```r
  inner_join(AFINN, by = c(word2 = "word")) |>
  count(word2, value, sort = T)

not_comment
```

```
          word2 value  n
1         worth     2 38
2           bad    -3 15
3          good     3 15
4          like     2  8
5          true     2  6
6          care     2  5
7         great     3  5
8          want     1  5
9         happy     3  4
10     terrible    -3  4
11    impressed     3  3
12    recommend     2  3
13       better     2  2
14       bother    -2  2
15        clear     1  2
16      falling    -1  2
17       forget    -1  2
18   impressive     3  2
19   interested     2  2
20          pay    -1  2
21       amazed     2  1
22      amazing     4  1
23  apocalyptic    -2  1
24        awful    -3  1
25      beating    -1  1
26      capable     1  1
27       chokes    -2  1
28  competitive     2  1
29    convinced     1  1
30         cool     1  1
31        crazy    -2  1
32      cutting    -1  1
33    difficult    -1  1
34 disappointed    -2  1
35         easy     1  1
36     exciting     3  1
```

```
37         fucking    -4  1
38       guarantee     1  1
39            hate    -3  1
40            help     2  1
41         helping     2  1
42            hide    -1  1
43   insignificant    -2  1
44             lag    -1  1
45         limited    -1  1
46            lmao     4  1
47          losing    -3  1
48            love     3  1
49             mad    -3  1
50          matter     1  1
51     outstanding     5  1
52         perfect     3  1
53        powerful     2  1
54          regret    -2  1
55       satisfied     2  1
56           scoop     3  1
57          shitty    -3  1
58     substantial     1  1
59           super     3  1
60             top     2  1
61           upset    -2  1
62         wasting    -2  1
63         wishing     1  1
64         worried    -3  1
65          worthy     2  1
66           wrong    -2  1
```

```
not_comment |>
  mutate(contribution = n * value) |>
  arrange(desc(abs(contribution))) |>
  head(20) |>
  mutate(word2 = reorder(word2, contribution)) |>
  ggplot(aes(n * value, word2, fill = n * value > 0)) +
  geom_col(show.legend = FALSE) +
  labs(x = "Sentiment value * number of occurrences",
       y = "Words preceded by \"not\"",
       title = "Sentiment error in review comments")
```



The diagram above shows that many more positive-sentiment words are being negated than negative-sentiment words in comments. There appears to be more variety in positive-sentiment words that commenters negate, and do so more often than with negative-sentiment words. This exploration suggests our sentiment analysis may, on average, judge comments more positive than they are.