Text Analysis and Sales Coaching

Applications of the R Language

Kyle Stone

Aug 6th, 2024

Hello, and welcome to this module on practical applications of text analysis for use in sales call coaching. Below we will run through 3 sections of analysis:

1. Unproductive Language Measurement.
2. Sentiment Analysis.
3. Word Relationship Analysis.

The sales call content used in this analysis was created by my prompting ChatGPT in various ways to create usable dialogue. You can view each individual call log, and the prompt used to receive it, on my GitHub: https://github.com/kylestone225/text_analysis_sales_coaching/tree/main.

Disclaimer: While ChatGPT was used in the creation of the sales call dialogue, it had ZERO part in the coding or the overall creation of this analysis. The applications demonstrated are translated from knowledge gained by working through the textbook "Text Mining With R: A Tidy Approach" found here: https://www.tidytextmining.com/

```
##########################################################################################

#                               1. Unproductive Language

##########################################################################################
```

To analyze our text we will be using a 'tidy' approach. This involves reading the word document and converting each individual word into a type of data called a 'token'. We then create a simple table in which each observation in the vector 'word' is each individual word of the sales call transcription.
We will repeat this approach in various ways throughout the module.

Our first sales call is a poor one. Our sales person is nervous and lacks confidence. They use a number of unproductive verbal crutches throughout the call. They do not win the business.

```
# Read your word document via the 'officer' library read_docx() function
sales_call <- read_docx("Sales Representative.docx")

# use the docx_summary() function to create a data frame of the docx content.
sales_call_content <- docx_summary(sales_call)

# we can now use this summary table to convert our text into tokens.  The below
# function will produce the proper tidy format for useful analysis.
sales_call_tokens <- sales_call_content |>
  unnest_tokens(word, text) |>
  mutate(row_number = row_number(),
         sales_rep = ifelse(doc_index %%2, word, ""),     # Split text into words
         customer = ifelse(doc_index %%2 == 0, word, "")) # by the rep or customer.

# Total words by sales person
sales_call_tokens |>
  filter(sales_rep != "") |>
  nrow()
```

```
## [1] 220
```

```
# Total by customer
sales_call_tokens |>
  filter(sales_rep == "") |>
  nrow()
```

```
## [1] 115
```

OK now that we have our data formatted properly let's do some basic analysis involving stop words. Stop words are unproductive language consisting of common but mostly meaningless words such as 'a', 'an', 'the' etc. We can use a stop words list to extract this unproductive language. The variance of which can be used as a measure for productive language.

```
s_words <- stop_words

sales_call_rem_stop_words <- sales_call_tokens |>
  anti_join(s_words)

# Non stop words by sales person
sales_call_rem_stop_words |>
  filter(sales_rep != "") |>
  nrow()
```

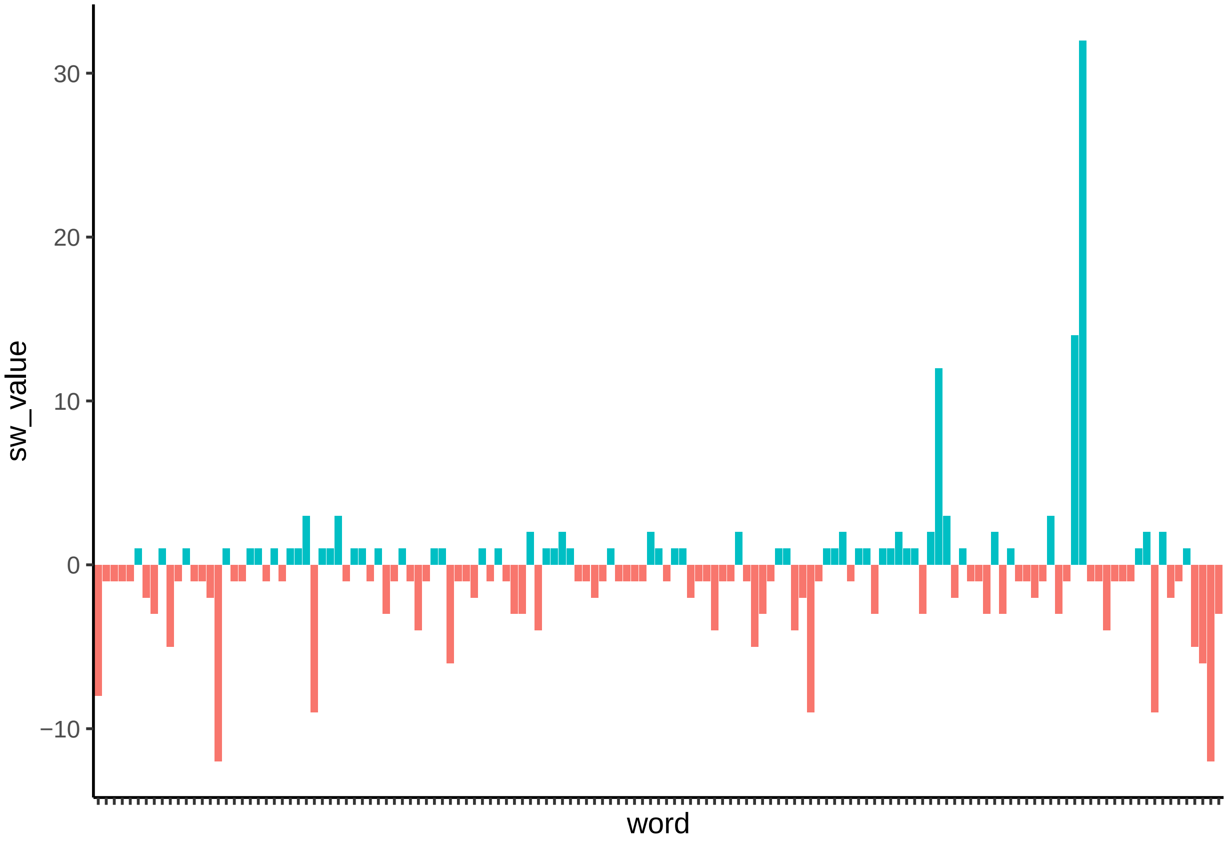
```
## [1] 102
```

The original call transcript totaled 220 words spoken by our sales person. By removing stop words we are left with 102 words. Meaning ~ 54% of the spoken words were unproductive.

Let's visualize where in the conversation our least productive language fell. This specific analysis does NOT tell us anything about the "productiveness" of our remaining words, yet.

```
# Collect your stop words
sc_sws <- sales_call_tokens |>
  filter(word %in% s_words$word)

sc_sws <- sc_sws|>
  rename(rep_sws = sales_rep) |>
  rename(cust_sws = customer)

sales_call_tokens <- left_join(sales_call_tokens,
                               sc_sws[,c(6:9)], by = c("word", "row_number"))
```

```
sales_call_tokens |>
  mutate(sw_value = ifelse(is.na(rep_sws), 1, -1)) |>
  ggplot(aes(word, sw_value, fill = ifelse(sw_value >= 0, "#00BFC4", "#F8766D"))) +
  geom_col(show.legend = FALSE) +
  scale_fill_identity(guide = "legend") +
  theme_classic() +
  theme(axis.text.x = element_blank())
```



The x-axis of this chart is each unique word spoken by the sales person in chronological order from the word's final occurrence in the transcript. The y-axis is the total number of times that word was spoken, with negative value given to each stop word.

The positive values in the above chart don't tell us much, yet.
Let's find out why by taking a look at the six most frequently used non stop words:

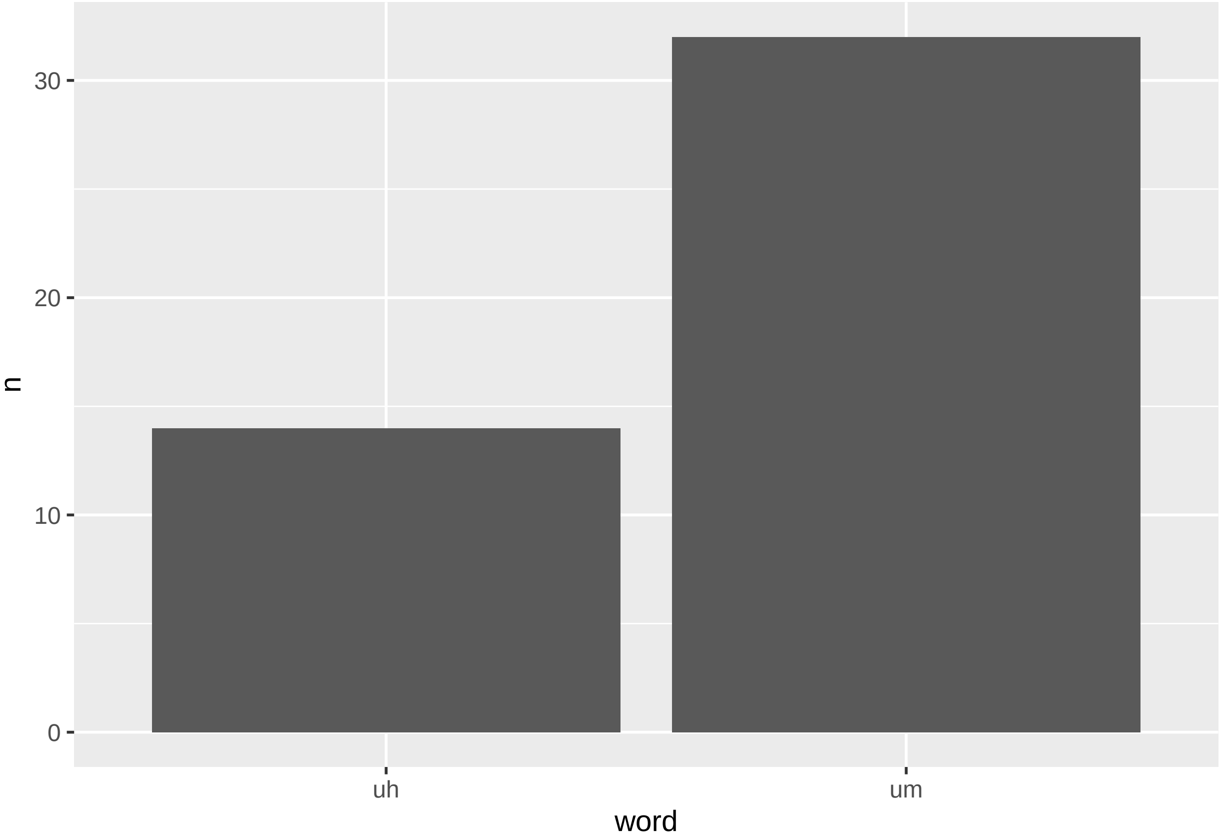
```
sales_call_rem_stop_words |>
  group_by(word) |>
  summarize(n = n()) |>
  arrange(-n) |>
  head()
```

```
## # A tibble: 6 x 2
##   word          n
##   <chr>     <int>
## 1 um           32
## 2 uh           14
## 3 sr           12
## 4 delivery      3
## 5 equipment     3
## 6 suppliers     3
```

One glaring problem with the sales representative's language: the over use of crutch words like 'uh' and 'um'. Inexperienced sales people will use crutch words to feel comfort when they are under prepared for an exchange, or they may simply lack the awareness. These words are indicators of a lack of confidence in the sales person's ability to properly deduce their customers problems, and build logic around their product's value proposition to solve these problems. Eliminating these words is crucial to instill confidence and produce more successful outcomes for new or inexperienced sales people.

Below we will do a simple word extraction and histogram visual for these problem words.

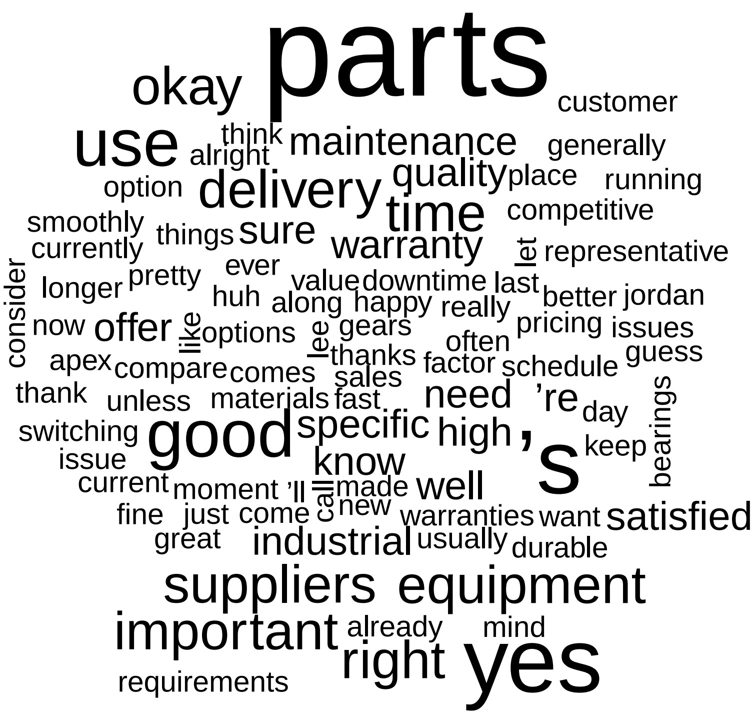
```
crutches <- c("uh", "um") # put whatever you want in here.

crutch_words <- sales_call_rem_stop_words |>
  filter(word %in% crutches)

# Visualize a histogram of the crutch word frequency
crutch_words |>
  group_by(word) |>
  summarise(n = n()) |>
  ggplot(aes(word, n)) +
  geom_col()
```



Identify problematic words in your department's communication, and build practice modules and metrics aiming for reduction in their use.

Word clouds are clumps of words sized by frequency, and can be helpful visuals. Let's visualize a word cloud of all words used in the original call. The code will automatically omit 'uh' and 'um' due to their overwhelming frequency.

```r
wordcloud(sales_call_tokens$word, min.freq = 1)
```



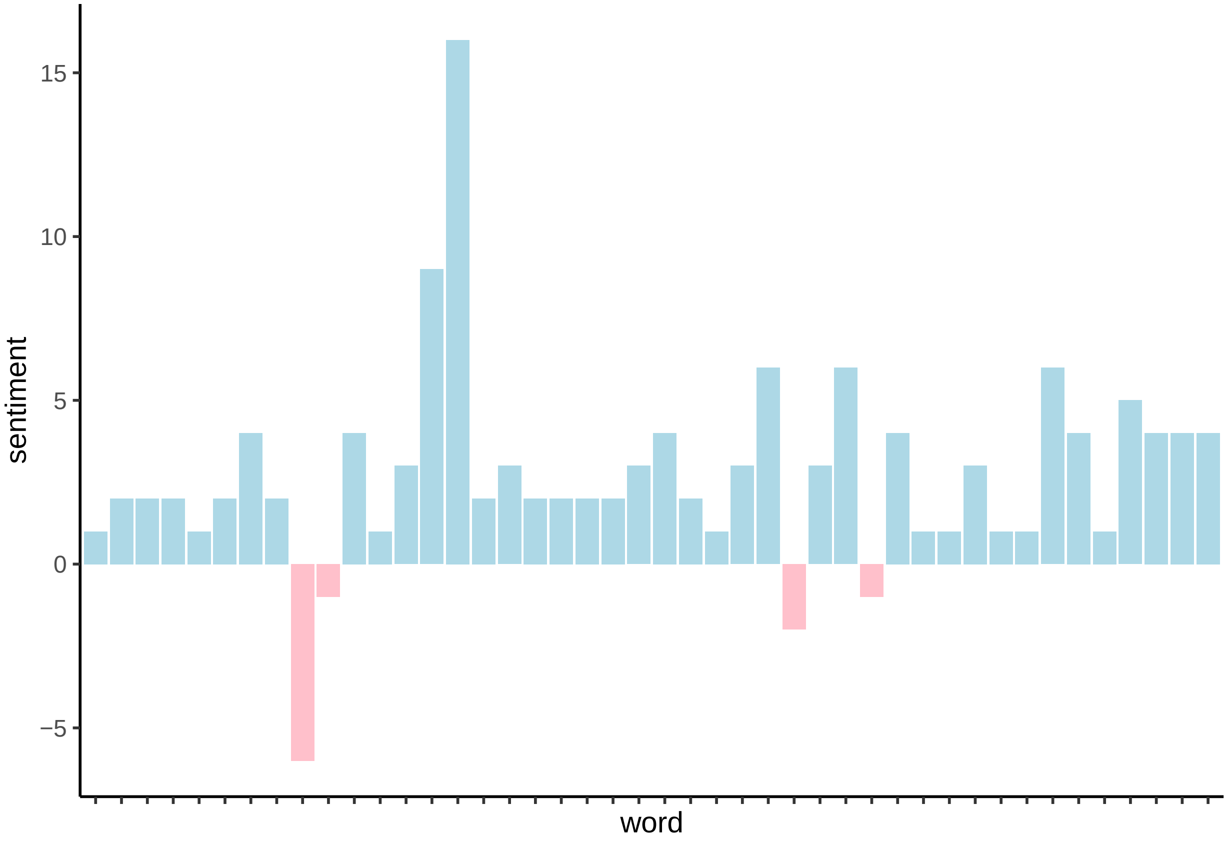
```r
#########################################################################################

#                                    2. Sentiment Analysis

#########################################################################################
```

Sentiment analysis can be an effective way to measure tone, mood, and even inflection of both your sales person and their audience. For this module I prompted ChatGPT to produce two sales call transcripts. In the first, the sales person is upbeat, enthusiastic, and charming. The customer returns the sentiment and is receptive, agreeing to a follow up appointment.

To begin the analysis I will use the freely available 'afinn' lexicon which ranks words from -5 to 5 based on their general positive or negative sentiment. IE the word 'abandoned' has a value of -3 and 'thrilled' is +5.

A reminder to the audience, this is a general use case of a freely available tool. Should your organization find value in an exercise like this, firms produce high-quality lexicon products which can be purchased to run far more complex analysis than demonstrated below.

```r
# Let's begin by reading our sales call transcript
happy_call <- read_docx("Upbeat Call 2.docx")

happy_content <- docx_summary(happy_call)

happy_tokens <- happy_content |>
  unnest_tokens(word, text) |>
  mutate(row = row_number()) |>
  anti_join(stop_words)

# Load the 'afinn' lexicon
afinn <- get_sentiments("afinn")

# Join the lexicon to your tokens table
happy_afinn <- happy_tokens |>
  inner_join(afinn) |>
  group_by(row, word) |>
  summarise(sentiment = sum(value)) |>
  mutate(method = "AFINN")

happy_afinn <- as.data.frame(happy_afinn)

happy_afinn <- happy_afinn |>
  mutate(row_number = row_number())

# Basic visualization of sentiment
happy_afinn |>
  ggplot(aes(word, sentiment, fill = ifelse(sentiment >= 0, "lightblue", "pink"))) +
  geom_col(show.legend = FALSE) +
  scale_fill_identity(guide = "legend") +
  theme_classic() +
  theme(axis.text.x = element_blank())
```



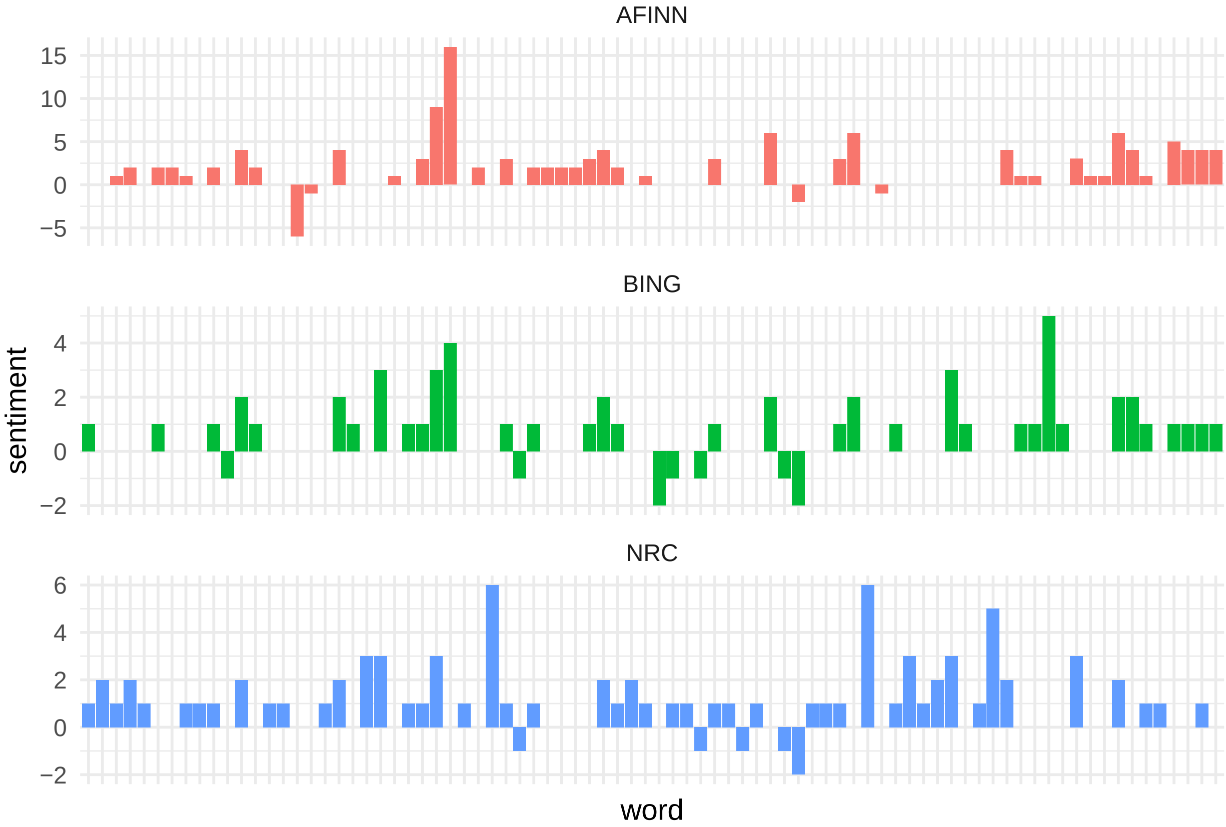
The x-axis of the produced bar chart is each unique word in the sales call that is also found in the 'afinn' lexicon. The y-axis is the sum of the sentiment value of all occurrences of that word. Most words here are quite positive!

Now let’s do a comparative analysis using two more lexicons: ‘Bing’ and ‘NRC’. Bing gives its words a sentiment value of either “positive” or “negative”. We convertthose values to 1, and -1. ‘NRC’ gives ten different sentiment values including words like “fear”, “sadness”, “joy”, and “trust”; in addition to “positive” and “negative”. We will filter out the former and assign 1, and -1 to the latter respectively. It is easy to see how a tool like the ‘NRC’ lexicon can start to add more complexity to analysis.

```
bing_and_nrc <- bind_rows(
  happy_tokens |>
    inner_join(get_sentiments("bing")) |>
    mutate(method = "BING"),
  happy_tokens |>
    inner_join(get_sentiments("nrc") |>
                 filter(sentiment %in% c("positive",
                                         "negative"))) |>
    mutate(method = "NRC")) |>
    count(method, word, sentiment) |>
    pivot_wider(names_from = sentiment,
              values_from = n,
              values_fill = 0) |>
    mutate(sentiment = positive - negative,
         row_number = ave(seq_along(method), method, FUN = seq_along))
```

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

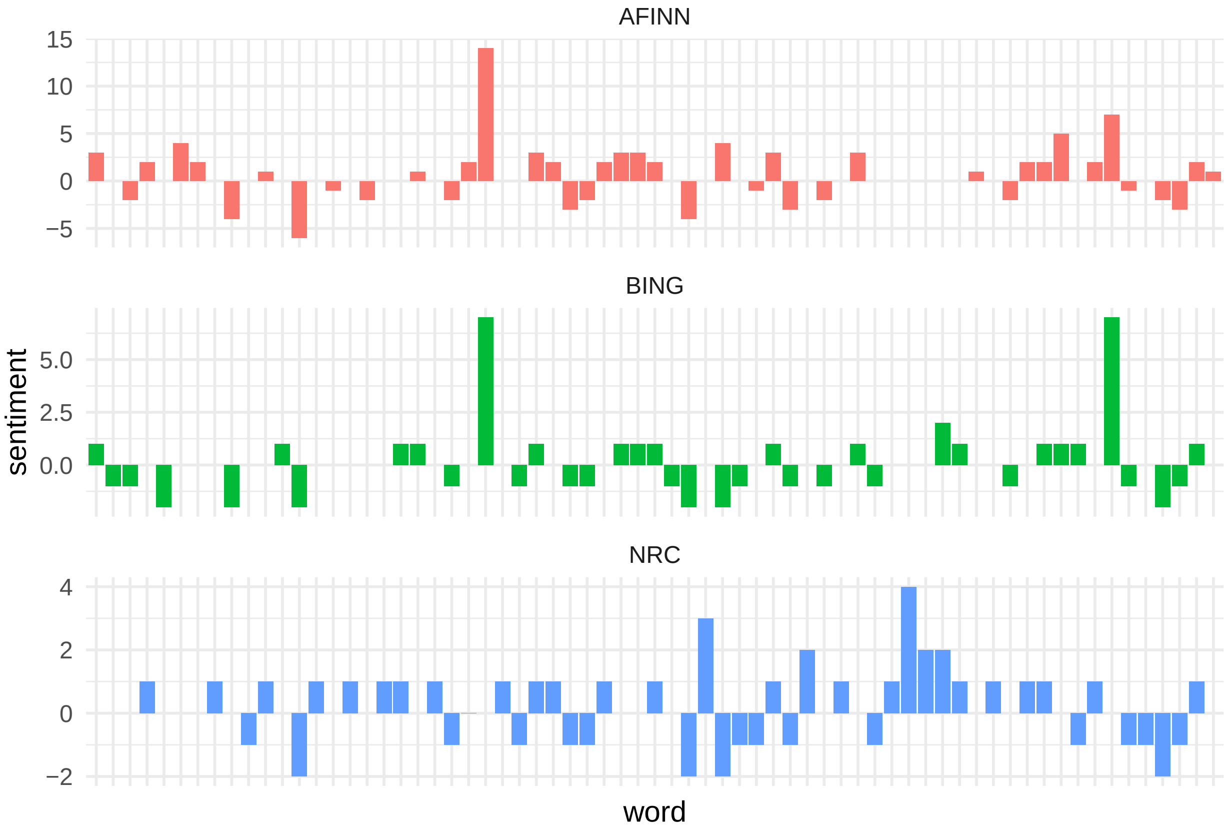
```
bind_rows(happy_afinn,
          bing_and_nrc) |>
  ggplot(aes(word, sentiment, fill = method)) +
  geom_col(show.legend = FALSE) +
  facet_wrap(~method, ncol = 1, scales = "free_y") +
  theme_minimal() +
  theme(axis.text.x = element_blank())
```



Blank x-axis values on this chart happen when a word found in the call transcript is also found in one or two lexicon's, but not the other. You find very few negative values in any of the three charts.

Let's take a look at a really negative call. I prompted ChatGPT to produce a sales call that goes very poorly and gets almost hostile. Let's run the same analysis!

```
grumpy_call <- read_docx("Grumpy Call 2.docx")

grumpy_content <- docx_summary(grumpy_call)

grumpy_tokens <- grumpy_content |>
  unnest_tokens(word, text) |>
  mutate(row = row_number()) |>
  anti_join(stop_words)

grumpy_afinn <- grumpy_tokens |>
  inner_join(afinn) |>
  group_by(row, word) |>
  summarise(sentiment = sum(value)) |>
  mutate(method = "AFINN")

grumpy_afinn <- as.data.frame(grumpy_afinn)

grumpy_afinn <- grumpy_afinn |>
  mutate(row_number = row_number())

grumpy_bing_and_nrc <- bind_rows(
  grumpy_tokens |>
    inner_join(get_sentiments("bing")) |>
    mutate(method = "BING"),
  grumpy_tokens |>
    inner_join(get_sentiments("nrc") |>
                 filter(sentiment %in% c("positive",
                                         "negative"))) |>
    mutate(method = "NRC")) |>
    count(method, word, sentiment) |>
    pivot_wider(names_from = sentiment,
              values_from = n,
              values_fill = 0) |>
    mutate(sentiment = positive - negative,
           row_number = ave(seq_along(method), method, FUN = seq_along))

bind_rows(grumpy_afinn,
          grumpy_bing_and_nrc) |>
  ggplot(aes(word, sentiment, fill = method), group = word) +
  geom_col(show.legend = FALSE) +
  facet_wrap(~method, ncol = 1, scales = "free_y") +
  theme_minimal() +
  theme(axis.text.x = element_blank())
```



Quite a difference in overall sentiment value!

These charts are meant to be easily digested visualizations of a very simple analysis. ChatGPT does a great job of producing content to be analyzed, but the exchanges last a few minutes at most. A full transcript of a 30 minute Teams call between multiple stakeholders will yield thousands of words to be analyzed by a more precise tool than the 'afinn' lexicon. Understanding the results and utilizing this methodology in coaching/training efforts will help sales people to align their cadence and attitude to match their audience.

Correct implementation should have conversations feel more natural to the customer and allow the sales person to appear more likable and trustworthy.

```
##########################################################################################

#                               3. N-gram Analysis

##########################################################################################
```

In the previous two modules we formatted our text in a single column of individual words. However, a lot of great analysis can be done by looking at the relationship between multiple words in a text. In the context of sales coaching, these relationships can be a measurement of the quality of questions your sales person is asking.

Open-ended questions are absolutely crucial for executing a successful sales cycle. The ability to get your audience to open up and elaborate on their business model, where they have success/failures, or what they like and don't like about their work, is key to building logical infrastructure in which you convey your product or service's value proposition.

Certain word relations like "how do you", "what kinds of", or "some examples of" elude to the types of questions that provoke longer, more elaborate answers. Conversely word relations like "how many", "did you", or "was that" provoke quantitative or single word responses.

For this call I prompted ChatGPT to produce a discovery call in which the sales person was very knowledgeable of the industry and asked many open ended questions prompting longer responses from the customer.

```
open_ended <- read_docx("Discovery Call.docx")

open_content <- docx_summary(open_ended)
```

For our analysis we will create three word relationships, or trigrams. The code will produce a table containing all of the three word relationships in the transcript. IE "How are you doing today?" would be arranged as three separate observations: "How are you", "are you doing", and "you doing today".

```
open_trigrams <- open_content |>
  unnest_tokens(trigram, text, token = "ngrams", n = 3)
```

Let's look at all 22 trigrams which were used more than once in the call.
You can see parts of what should be good discovery questions.

```
open_trigrams |>
  count(trigram, sort = TRUE) |>
  head(22)
```

```
##                       trigram n
## 1       engineering team and 3
## 2         landing gear parts 3
## 3     advanced materials and 2
## 4           anderson we have 2
## 5     commercial jet engines 2
## 6                 could be a 2
## 7        fuel efficiency and 2
## 8                 how do you 2
## 9                 how we can 2
## 10              how we might 2
## 11          into our designs 2
## 12            it sounds like 2
## 13 lightweight high strength 2
## 14        looking forward to 2
## 15             me more about 2
## 16    mr anderson absolutely 2
## 17            mr anderson we 2
## 18      our engineering team 2
## 19        our turbine blades 2
## 20         sarah that sounds 2
## 21              tell me more 2
## 22               you tell me 2
```

If we separate the trigram into individual word vectors we can apply more detailed analysis.

```r
separated_trigrams <- open_trigrams |>
  separate(trigram, c("word1", "word2", "word3"), sep = " ")
```

Take a look at all instances in which “how” and “could” began a trigram:

```r
separated_trigrams |>
  filter(word1 == "how") |>
  select(word1, word2, word3)
```

```
##   word1 word2    word3
## 1   how   are      you
## 2   how about yourself
## 3   how    we    might
## 4   how    do      you
## 5   how    do      you
## 6   how    we    might
## 7   how    we      can
## 8   how about       we
## 9   how    we      can
```

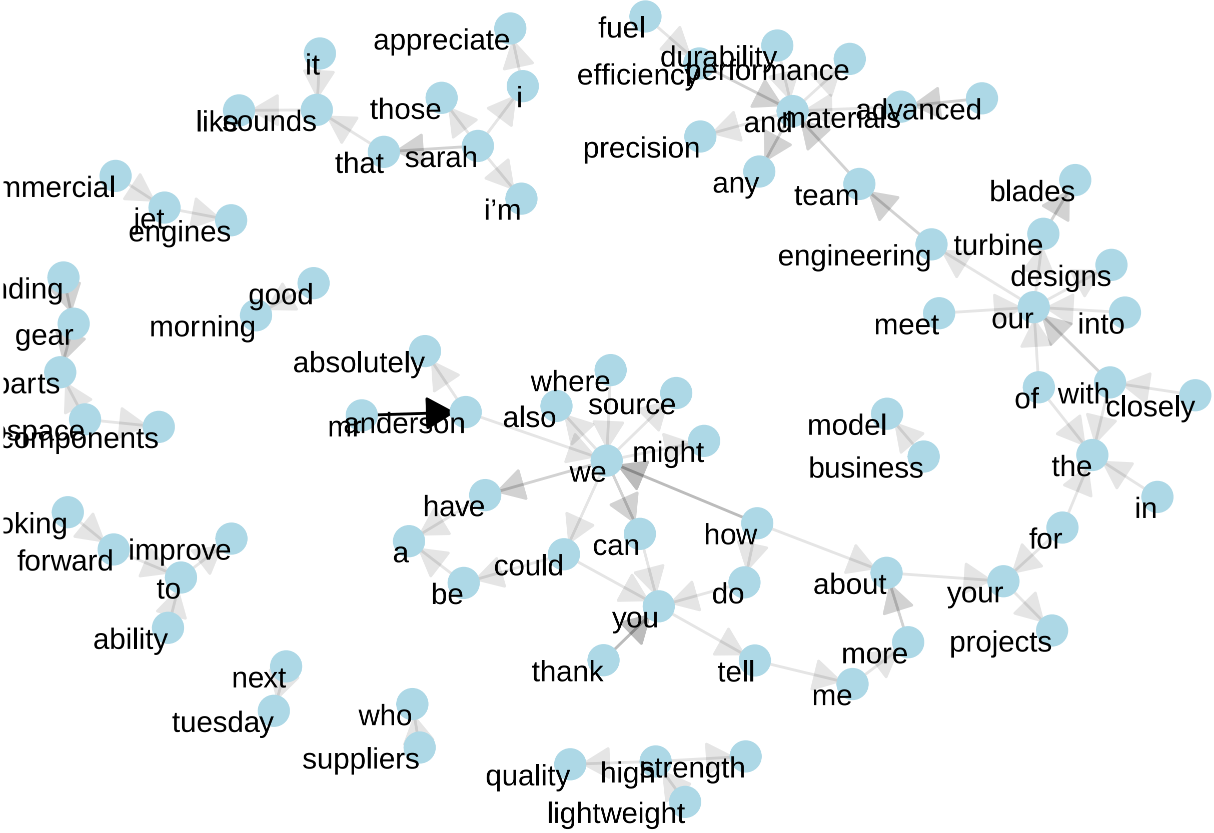
```r
separated_trigrams |>
  filter(word1 == "could") |>
  select(word1, word2, word3)
```

```
##   word1       word2       word3
## 1 could         you       start
## 2 could         you        tell
## 3 could potentially     provide
## 4 could        help      extend
## 5 could       offer significant
## 6 could          be           a
## 7 could          be           a
```

This demonstrates effective use of language! Many of the “how” trigrams should provoke elaborate responses. The “could” trigrams show earnestness in understanding the customer’s problems. This person got answers to their questions, as seen in the full transcript.

Finally, let's visualize a word relationship network.
For demonstration purposes we will visualize our network of two word relationships, or bigrams.

```r
# First arrange our bigram data
open_bigrams <- open_content |>
  unnest_tokens(bigram, text, token = "ngrams", n = 2)

separated_bigrams <- open_bigrams |>
  separate(bigram, c("word1", "word2"), sep = " ")

bigram_counts <- separated_bigrams |>
  count(word1, word2, sort = TRUE)

# Produce a graphical object to be plotted
bigram_graph <- bigram_counts |>
  filter(n >= 2) |>
  graph_from_data_frame()

# Visualize your network of bigrams!
set.seed(2020)

a <- grid::arrow(type = "closed", length = unit(.15, "inches"))

ggraph(bigram_graph, layout = "fr") +
  geom_edge_link(aes(alpha = n), show.legend = FALSE,
                 arrow = a, end_cap = circle(.07, 'inches')) +
  geom_node_point(color = "lightblue", size = 5) +
  geom_node_text(aes(label = name), vjust = 1, hjust = 1) +
  theme_void()
```



So there you have it. Through some basic math, critical thought, and visual tools, we have the beginnings of what could grow to be a thorough and complex analysis of your sales department's communication.

I hope you found some value in this module because these exercises are completely fascinating to me. I would love to speak with you further about how I would implement this and other sections of my R language tutorial as the manager of your sales department.

Kyle Stone

215.880.8774

kylestone225@gmail.com