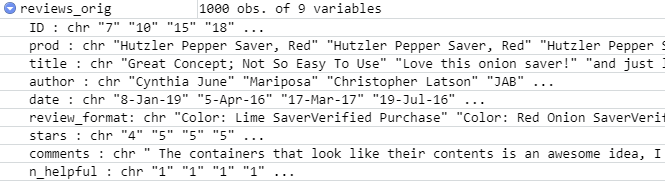


**MKTG 788 Exam 2**

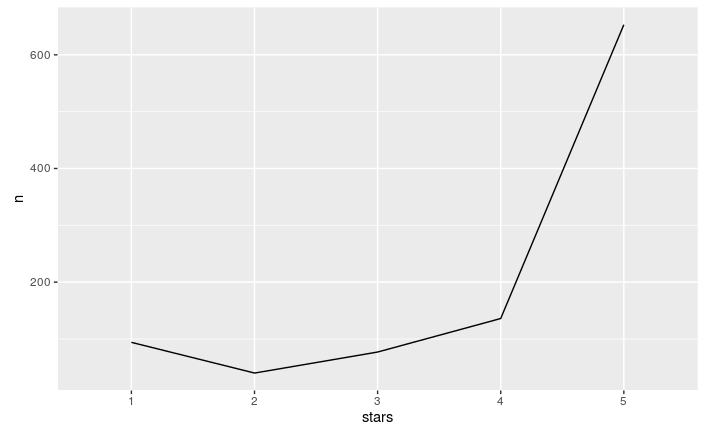
**Hutzler Pepper Saver, Red**

Text Analysis of Amazon Reviews

The dataset used for this analysis is B005BPZCUC\_Reviews for Exam 2.csv. The dataset consists of 1000 reviews with 9 variables. Here is a look at the data:



The following plot show the distribution of reviews as per their star ratings:



Around 650 reviews have a 5-star rating followed by ~130 reviews with a 4-star rating. Reviews with 1,2&3-star ratings together are ~200 in number.

We will begin by grouping the reviews by their Star rating (1-5) and then analyzing their sentence level sentiments using the Tyler Rinker’s dictionary. Tyler Rinker is the author of sentimentr-the package used to generate the sentence level sentiment score.

Below is a box plot for the review sentiment scores grouped by their star ratings. In the plot below, we can see that the medians of review sentiment scores are well above or below the Neutral sentiment (0.0). This aligns with the star ratings. Few outliers can be seen in the sentiment scores. This suggests high use of positive or negative sentiment words in that review. Surprisingly, some outliers in the 4&5-star rating groups lie in the negative side of the plot.

Medians of average sentiments of 1- & 2-star rated reviews lie below the neutral sentiment line, while median of 3-star rated review sentiments lies on the neutral line. Other 4 & 5 star rated reviews lie well above the neutral sentiment scores.

We can thus say that the sentiment scores and the star ratings of the reviews go well with each other. There are few contradictions (outliers) which, however, do not affect the results much.

The minimum (the smallest sentiment in the data set) is shown at the far bottom of the chart, at the end of the bottom “whisker.”

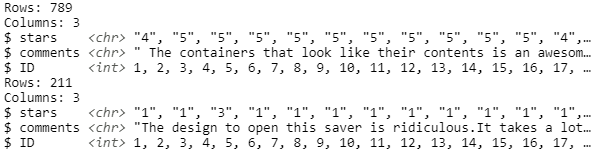
The [median](https://www.statisticshowto.com/probability-and-statistics/statistics-definitions/mean-median-mode/#median)is shown as a line somewhere in the middle of the box.

First [quartile](https://www.statisticshowto.com/what-are-quartiles/), Q1, is the bottom part of the box .

Third quartile, Q3, shown at the top of the median.

The maximum (the largest sentiment in the data set), shown at the top of the box.

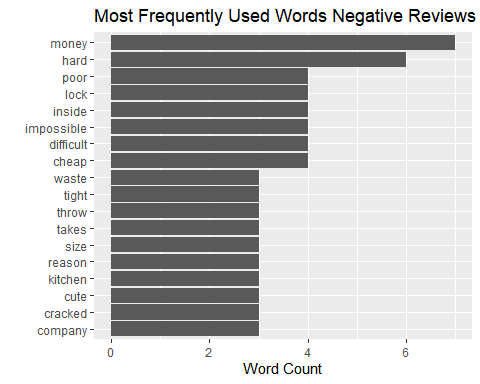
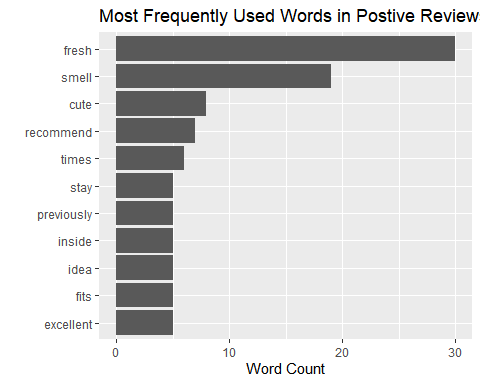
We will now divide the reviews as positive and negative based on their star ratings. Since sentiment scores of reviews with 1,2&3-star ratings have at least one quadrant below or intersected by the neutral sentiment line, we will consider them as negative review. Sentiment scores of reviews with 4&5-star ratings lie well above the neutral sentiment line; hence we will consider them as positive reviews. We get 789 positive reviews and 211 negative reviews as shown below.



Next, we will remove some undesirable words from both the datasets that might cause bias in the analysis.

"saver", "onion", "tomato", "lemon", "fridge", "garlic", "onions", "container", "pepper", "plastic", "half", "lemons", "product", "time", "close", "time", "store", "bought", "tomatoes", "grapefruit", "refrigerator", "days", "easily", "easy", "item", "peppers", "bottom", "storage", "savers", "perfect", "containers", "lime", "limes", "savers", "keeper", "pieces", "food", "makes", "save", "keeping", "love", "nice", "holder", "week", "save", "hold", "bags", "produce", "clean", "helps", "vegetable", "fruit", "wrap", "veggies", "holds"

Here is a plot that shows the most frequent words used in both the datasets:



|  |  |
| --- | --- |
| In the positive reviews, the reviewers have mostly written about the freshness, smell/odor, and 'cute' appearance of the pepper saver. Some have also mentioned about recommending the pepper saver | In the negative reviews, reviewers have written about money which implies that there is possibility of not finding the product "value for money". There are less chances of customers finding the pepper saver a "value for money" product in the negative reviews. Frequent use of words like "hard", "difficult" and "impossible" suggest on the usability of the product. |

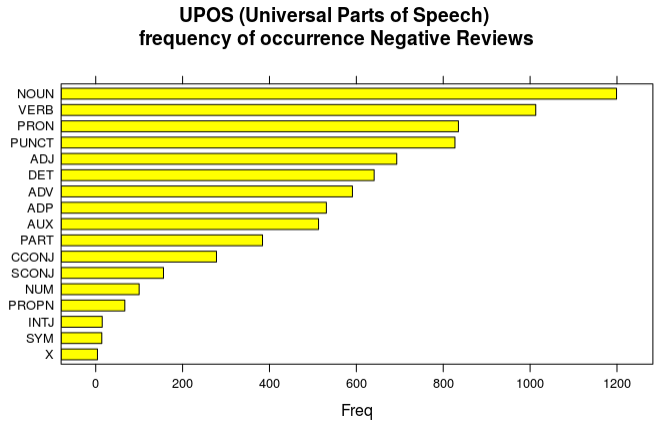
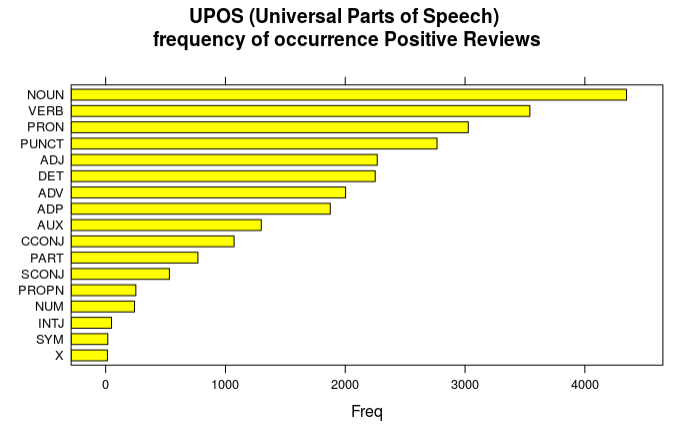
**Word clouds**

**Positive Reviews Negative Reviews**

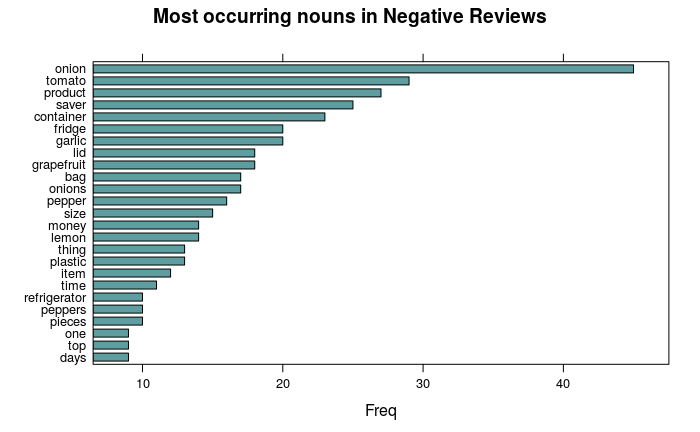
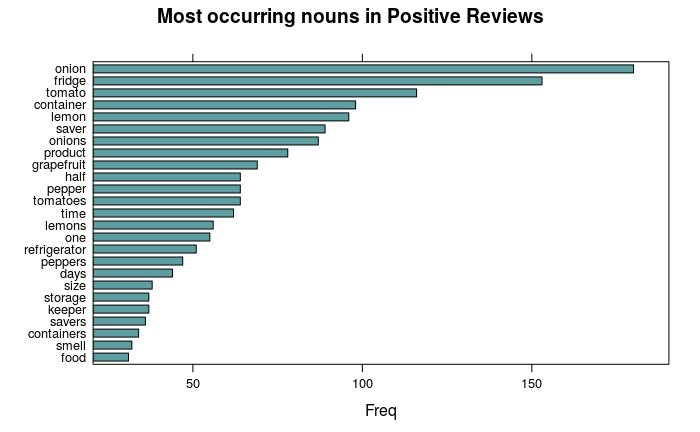


The word clouds above display some prominent words in both the datasets. Topics like “smell”, “fresh” are most frequent in the positive reviews, while topics like, “money”, “hard”, “impossible”, “cheap”, “poor” are some of the frequent topics among the negative reviews.

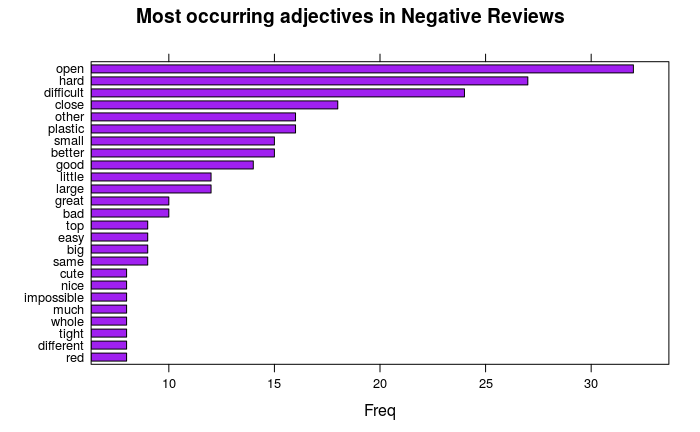
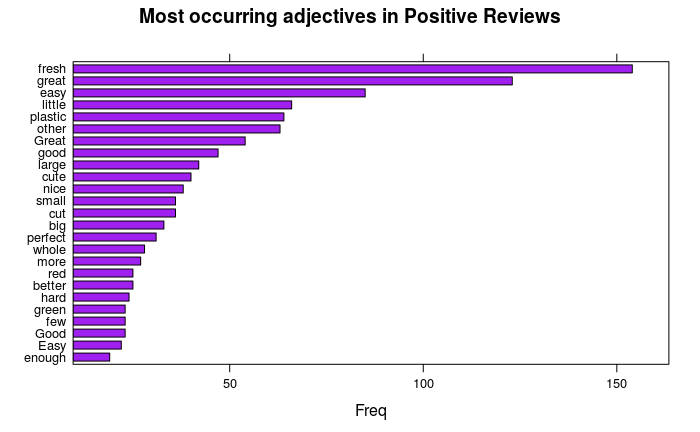
**Speech analysis using UDPIPE and RAKE**



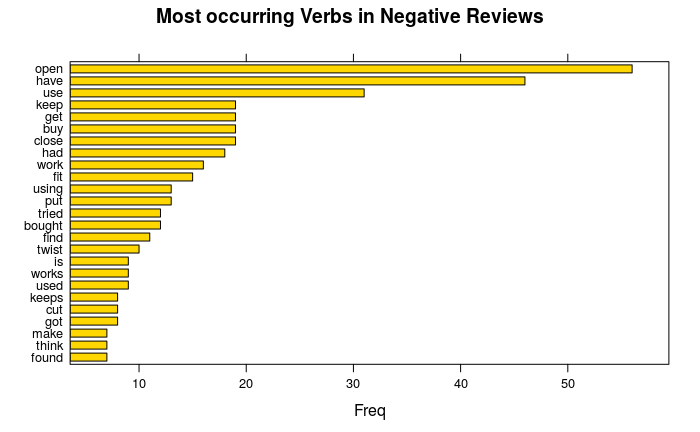
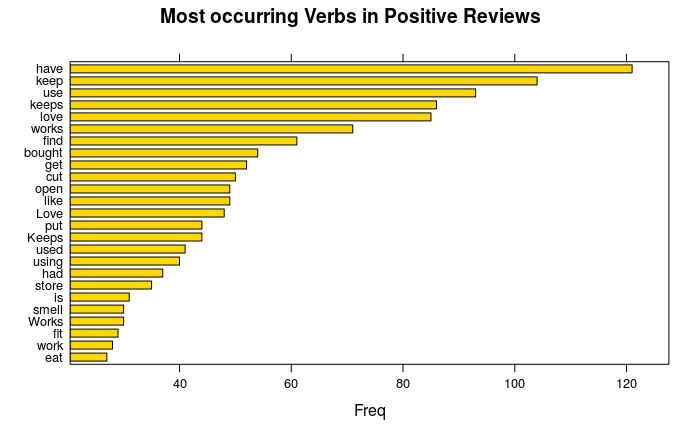
The plots above display the frequency of different parts of speech like nouns, verbs etc. The order of different parts of speech is almost similar in both the positive and negative reviews. During the analysis, we must consider the magnitude of reviews in each category, i.e. 789 positive reviews and 211 negative reviews in both datasets.



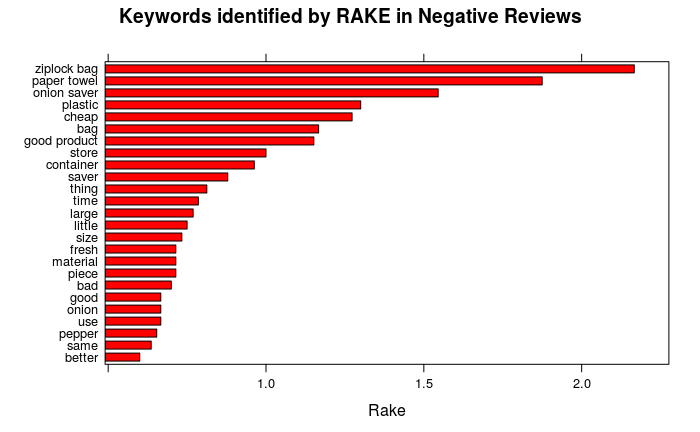
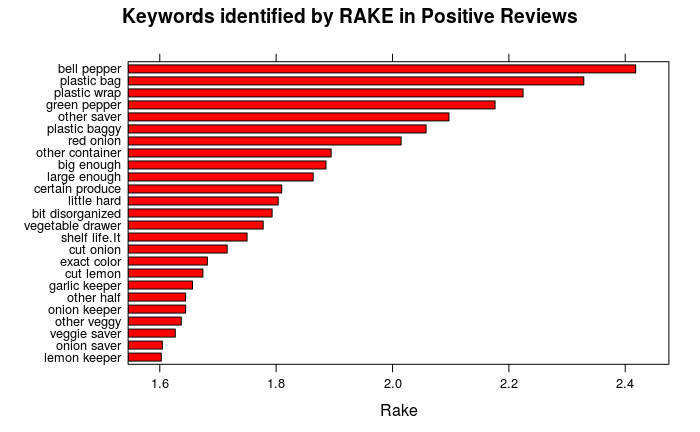
“Onion” and “tomato” are some of the most frequently used nouns in both the reviews. Surprisingly “pepper” is not amongst the top.



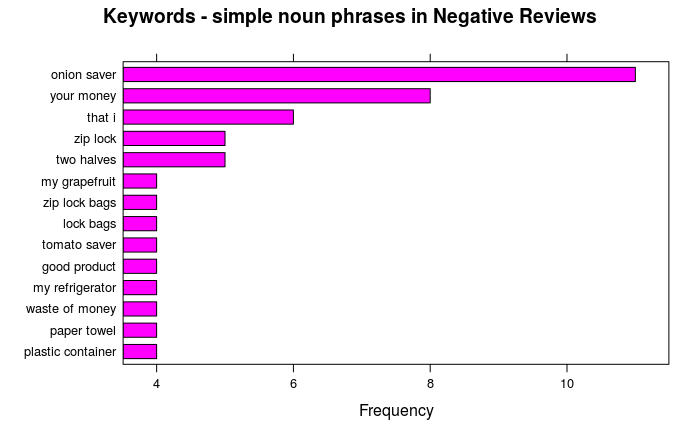
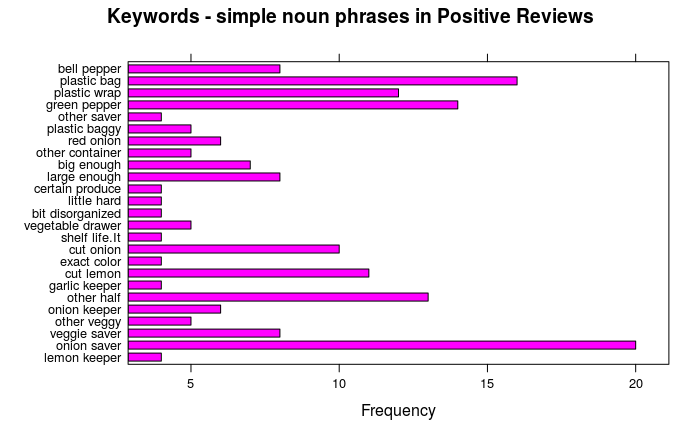
Positive reviews mostly have frequent words that talk about freshness, ease, size, and “cute” appearance of the pepper saver, while the negative reviews mostly contain frequent words that express “difficulty” in opening and closing the pepper saver. This contradicts the positive reviews that mention ease. However, after considering the frequency of these words in each category, we can say that overall, the pepper saver is easy to use.



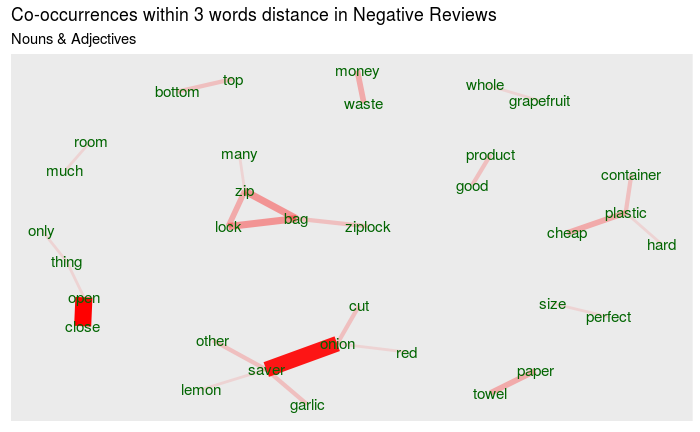
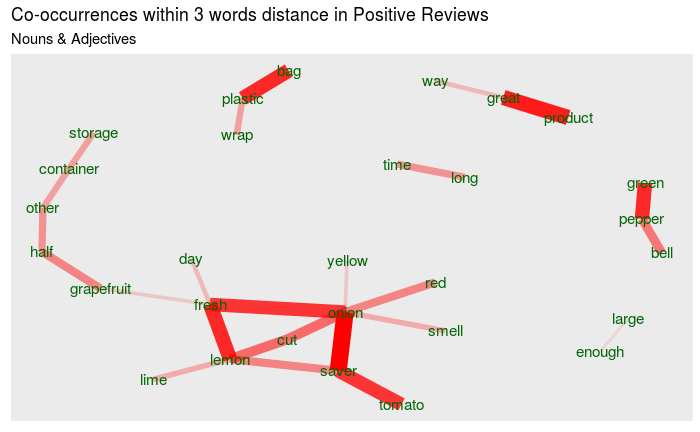
Some of the frequent verbs in the positive reviews are “love”, “open”, “smell”. These suggest that most of the user’s “love” the pepper saver. They might also find it easy to “open” in terms of usability. The veggies must “smell” good when stored in the pepper saver. While the most frequent verb in the negative reviews is “open”. Also, the verb “close” is frequent. These verbs suggest that a set of users find the usability of the product, i.e. opening and closing of pepper saver difficult.



Some of the keywords in positive reviews are “big enough”, “large enough” suggest that the reviewers find the pepper saver adequate for storage. One thing tp notice is that the dataset is contaminated and might have reviews related to other veggie savers.



The simple noun phrases in negative reviews mostly mention other storage products like “zip lock bags”. This might suggest that they find the alternatives better than the pepper saver.



The 3-word distant co-occurring words in the positive reviews mainly suggest that the saver keeps the veggies fresh for long time. While in the negative reviews, phrases like money-waste, cheap-plastic-container, hard-plastic-container, open-close, top-bottom, zip-lock-bag bring forward some of the issues related to the saver. These 2- or 3-word distant phrases are easy to draw meanings from. For example, money-waste suggests that reviewers find that purchasing the saver is waste of money.

**Topic Modeling**

tf–idf or TFIDF, short for term frequency–inverse document frequency, is a numerical statistic that is intended to reflect how important a word is to a [document](https://en.wikipedia.org/wiki/Document) in a collection or [corpus](https://en.wikipedia.org/wiki/Text_corpus)

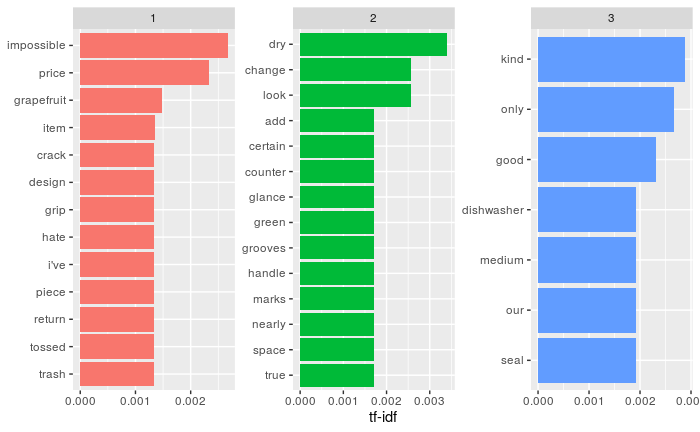
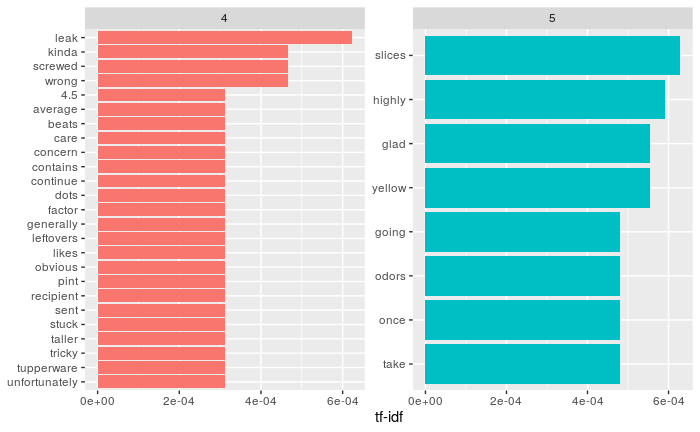
# First, we get the count of each word in each review.

# Second, get the number of words per text input.

# Third, combine these dfs and get the tf\_idf & order the words by degree of relevance

# Finally, we plot the 10 most informative terms per topic grouped by the grouping covariate

**Positive Reviews Negative Reviews**

****

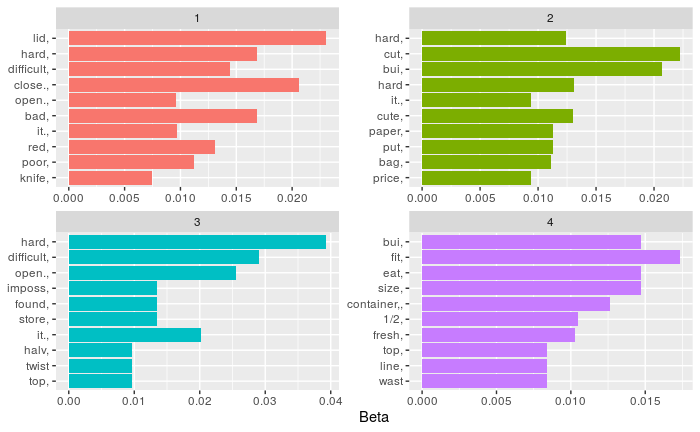
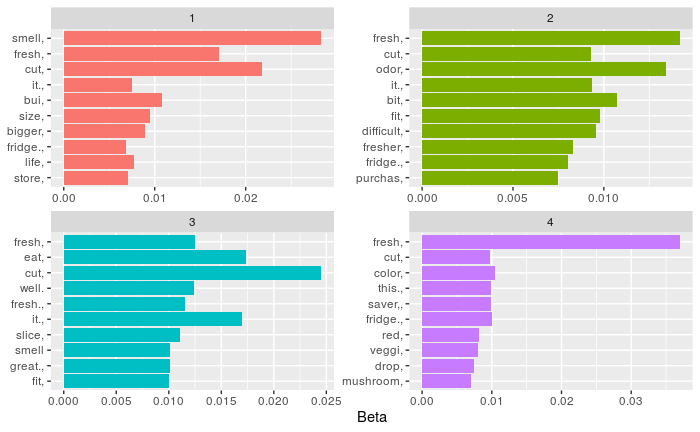
We have used star rating as the grouping covariate. Surprisingly the 4 star rated reviews have some topics that mentions “concern”, “screwed”, “leak” and “wrong” which seem negative. “Tupperware” is a competitor company that also produces such savers. This suggests some topics might contain comparison of the product with a competing company.

The tf-idf in negative reviews however justify the star ratings. Words like “price”, “crack”, “return”, “trash” suggest dissatisfied topics. The 3 star rated reviews seem to have neutral topics.

**LDA**

Latent Dirichlet allocation (LDA) is a particularly popular method for fitting a topic model. It treats each document as a mixture of topics, and each topic as a mixture of words. This allows documents to “overlap” each other in terms of content, rather than being separated into discrete groups, in a way that mirrors typical use of natural language.

**Positive Reviews Negative Reviews**

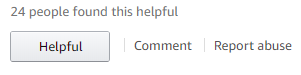


This visualization lets us understand the four topics that were extracted from each of the positive and negative reviews.

|  |  |
| --- | --- |
| * The most common words in topic 1 of positive reviews are “smell”, “fresh”, “size”, “bigger”. * All the topics show some overlap and mostly relate to “smell”, “fresh”, “cut” or “fridge”. | * The most common words in topic 1 of negative reviews are “hard”, “difficult”, “close”, “open” which again suggest on the usability of the saver. * Topic 2 has words like “cute”, “price” and some overlap from topic 1 words. Topic 2 suggests that a user might find the product “cute” in appearance, but the word “price” suggests that it is not value for money. |

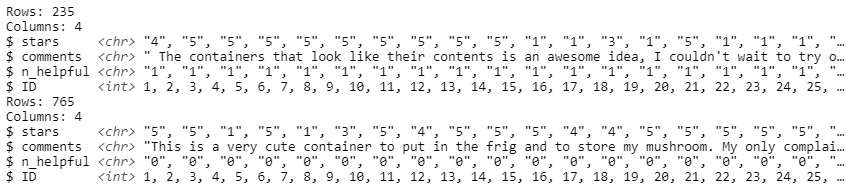
Topics in negative reviews have less overlap of words as compared to positive reviews.

Helpful Reviews

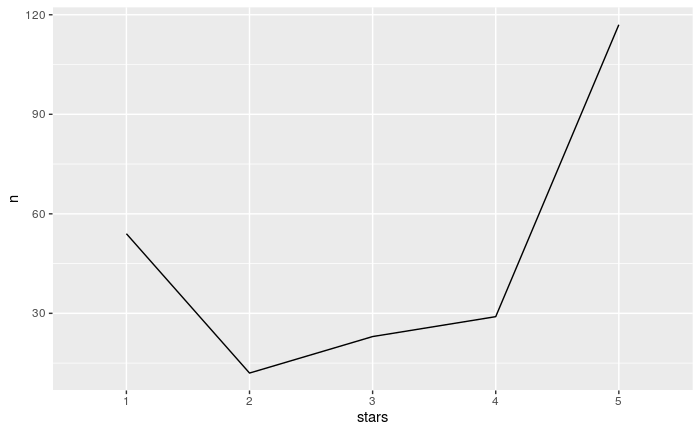


The dataset also includes a column called ‘n\_helpful’. This column has a count of upvotes for a review. The values range from 1 to 36 in our dataset. We can do some analysis on the reviews that have at least 1 upvote as “Helpful”. This will further help us to understand what features of a review make it helpful or not helpful for other potential customers.

For this let us divide the dataset into two, one marked as helpful and other that is not marked as helpful. There are 235 reviews out of 1000 marked as ‘helpful’ at least once.



Here is the distribution of star ratings for the helpful reviews:



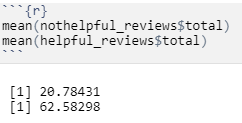
Again, most of the reviews that were found helpful were rated as 5-star followed by 1-star.

Comparison of Parts of speech and topic models on both helpful and not\_helpful datasets give similar results.

However, one striking difference between both is the length of the review or the number of words used in a review.

To find this we add a new column in these datasets that consists of the total number words used in these reviews. Please note that these reviews are raw and unfiltered.

Now let us find the mean of the total number of words used in each of these reviews.



It appears that on an average, the length of ‘helpful’ reviews is ~63 words while the ‘not\_helpful’ reviews have an average length of just ~21 words. Thus, we can say that, the more descriptive and lengthier a review is, the more helpful it is found by the readers. In other words, long reviews, or reviews of ~63 words in length are considered to be more helpful.

R-Code

---

title: "Mktg Final Exam"

author: "Rutvik Gavaskar"

date: "5/4/2020"

output:

word\_document: default

html\_document: default

pdf\_document: default

---

```{r}

# Step 0

# Install pacman in case you don’t have pacman already installed.

if(!"pacman" %in% installed.packages()[,"Package"]) install.packages("pacman")

```

```{r}

# Step 1 – add the required libraries and the data unless you have it already loaded

# Install and load the required packages.

pacman::p\_load(dplyr, ggplot2, tidytext, wordcloud2)

pacman::p\_load(tidyr, dplyr, stringr, data.table, sentimentr, ggplot2, text2vec, tm, ggrepel)

# load the data unless you already have it loaded

# remember to use the stringsAsFactors = F argument, otherwise errors galore.

# this dataset if provided, if you use others create it from the master set

reviews\_orig <- read.csv("B005BPZCUC\_Reviews for Exam 2.csv", stringsAsFactors = FALSE, header = FALSE, fileEncoding="latin1")

colnames(reviews\_orig) <- as.character(unlist(reviews\_orig[1,]))

reviews\_orig = reviews\_orig[-1, ]

#reviews\_orig$date <- as.Date(reviews\_orig$date,"%d/%b/%Y")

```

```{r}

# create a rowid for the reviews

review\_df <- reviews\_orig %>% mutate(id = row\_number())

# examine the structure

str(reviews\_orig)

```

# Step 2 – define the lexicon and any changes needed for our context

# get n rows – to see what we have in the lexicon –

# Tyler Rinker is the author of sentimentr

```{r}

nrow(lexicon::hash\_sentiment\_jockers\_rinker) # seems like 11,710 words

# words to replace – in this example, there are switch, brand names etc.

replace\_in\_lexicon <- tribble(

~x, ~y,

"huztler", 0, # not in dictionary

"pepper", 0, # not in dictionary

"red", 0, # original score: -.6

"saver", 0, # not in dictionary

)

# create a new lexicon with modified sentiment

review\_lexicon <- lexicon::hash\_sentiment\_jockers\_rinker %>%

filter(!x %in% replace\_in\_lexicon$x) %>%

bind\_rows(replace\_in\_lexicon) %>%

setDT() %>%

setkey("x")

```

# Step 3 – start by getting the sentence level sentiment for testing

# get sentence-level sentiment

```{r}

sent\_df <- review\_df %>%

get\_sentences() %>%

sentiment\_by(by = c('id', 'author', 'date', 'stars', 'review\_format'), polarity\_dt = review\_lexicon)

```

# Step 4 – start by getting the sentence level sentiment for testing

# check the relationship between star rating and sentiment

```{r}

ggplot(sent\_df, aes(x = stars, y = ave\_sentiment, color = factor(stars), group = stars)) +

geom\_boxplot() +

geom\_hline(yintercept=0, linetype="dashed", color = "red") +

geom\_text(aes(5.2, -0.05, label = "Neutral Sentiment", vjust = 0), size = 3, color = "red") +

guides(color = guide\_legend(title="Star Rating")) +

ylab("Average Sentiment") +

xlab("Review Star Rating") +

ggtitle("Sentiment of Amazon Reviews, by Star Rating")

```

######################Part 2

```{r}

#split dataset as per ratings

lst <- split(reviews\_orig,reviews\_orig$stars<4)

pos\_reviews <- as.data.frame(lst[1])

neg\_reviews <- as.data.frame(lst[2])

names(pos\_reviews)[names(pos\_reviews) == "FALSE.stars"] <- "stars"

names(pos\_reviews)[names(pos\_reviews) == "FALSE.comments"] <- "comments"

names(neg\_reviews)[names(neg\_reviews) == "TRUE.stars"] <- "stars"

names(neg\_reviews)[names(neg\_reviews) == "TRUE.comments"] <- "comments"

# select only the stars and comments from the data and examine it

pos\_reviews <-pos\_reviews %>% select(stars, comments)

pos\_reviews$ID <- seq.int(nrow(pos\_reviews)) #Adding ID column

glimpse(pos\_reviews)

neg\_reviews <-neg\_reviews %>% select(stars, comments)

neg\_reviews$ID <- seq.int(nrow(neg\_reviews)) #Adding ID column

glimpse(neg\_reviews)

```

```{r}

# Step 2 – delete all undesirable words, here we only delete things that may bias analyses

# adjust this list as you need it, basically eliminate all undesirable words

undesirable\_words <- c("saver", "onion", "tomato","lemon","fridge", "garlic","onions","container", "pepper","plastic", "half", "lemons", "product", "time", "close","time", "store", "bought", "tomatoes", "grapefruit", "refrigerator", "days", "easily", "easy","item", "peppers", "bottom", "storage","savers", "perfect", "containers", "lime", "limes", "savers", "keeper", "pieces", "food","makes", "save", "keeping", "love", "nice", "holder", "week", "save", "hold", "bags", "produce", "clean", "helps", "vegetable", "fruit", "wrap", "veggies", "holds")

# check out a small sample of stop words, randomly

head(sample(stop\_words$word, 15), 15)

```

```{r}

# Step 3 – unnest the comments, remove all stop and undesirable words and words smaller # than 3 characters and examine the result

#unnest and remove stop, undesirable and short words

pos\_reviews\_filtered <- pos\_reviews %>%

unnest\_tokens(word, comments) %>%

anti\_join(stop\_words) %>%

distinct() %>%

filter(!word %in% undesirable\_words) %>%

filter(nchar(word) > 3)

dim(pos\_reviews\_filtered)

neg\_reviews\_filtered <- neg\_reviews %>%

unnest\_tokens(word, comments) %>%

anti\_join(stop\_words) %>%

distinct() %>%

filter(!word %in% undesirable\_words) %>%

filter(nchar(word) > 3)

dim(neg\_reviews\_filtered)

```

```{r}

# Step 4 – get the full word count from the lyrics and quickly examine the results

pos\_reviews\_full\_word\_count <- pos\_reviews\_filtered %>%

unnest\_tokens(word, word) %>%

group\_by(stars) %>%

summarise(num\_words = n()) %>%

arrange(desc(num\_words))

neg\_reviews\_full\_word\_count <- neg\_reviews\_filtered %>%

unnest\_tokens(word, word) %>%

group\_by(stars) %>%

summarise(num\_words = n()) %>%

arrange(desc(num\_words))

pos\_reviews\_full\_word\_count

neg\_reviews\_full\_word\_count

```

```{r}

# Step 5 – plot the most commonly used words in the lyrics

pos\_reviews\_filtered %>%

count(word, sort = TRUE) %>%

top\_n(10) %>%

ungroup() %>%

mutate(word = reorder(word, n)) %>%

ggplot() +

geom\_col(aes(word, n)) +

xlab("") +

ylab("Word Count") +

ggtitle("Most Frequently Used Words in Postive Reviews ") +

coord\_flip()

# Step 5 – plot the most commonly used words in the lyrics

neg\_reviews\_filtered %>%

count(word, sort = TRUE) %>%

top\_n(10) %>%

ungroup() %>%

mutate(word = reorder(word, n)) %>%

ggplot() +

geom\_col(aes(word, n)) +

xlab("") +

ylab("Word Count") +

ggtitle("Most Frequently Used Words Negative Reviews") +

coord\_flip()

```

```{r}

# Step 6 – create a cool wordcloud of the words in the lyrics

pos\_reviews\_word\_counts <- pos\_reviews\_filtered %>% count(word, sort = TRUE)

wordcloud2(pos\_reviews\_word\_counts[1:300, ], size = .5)

neg\_reviews\_word\_counts <- neg\_reviews\_filtered %>% count(word, sort = TRUE)

wordcloud2(neg\_reviews\_word\_counts[1:300, ], size = .5)

```

# Step 4 –GloVe (Global Vectors for Word Representation). This step does two things.

# First, we calculate a contextual representation of a word in a vector form

# Similar words will have vectors that are similar or close to each other,

# while words that are different will be much further away.

# Second we use an unsupervised learning on n-gram / dimensionality reduction

# create lists of reviews split into individual words (iterator over tokens)

```{r include=FALSE}

pacman::p\_load(tidyr, dplyr, stringr, data.table, sentimentr, ggplot2, text2vec, tm, ggrepel)

tokens <- space\_tokenizer(pos\_reviews$comments %>% tolower() %>% removePunctuation())

# Create vocabulary. Terms will be unigrams (simple words).

it <- itoken(tokens, progressbar = FALSE)

vocab <- create\_vocabulary(it)

# prune (remove) words that appear less than 3 times

vocab <- prune\_vocabulary(vocab, term\_count\_min = 3L)

# Use our filtered vocabulary

vectorizer <- vocab\_vectorizer(vocab)

# use skip gram window of 5 for context words

tcm <- create\_tcm(it, vectorizer, skip\_grams\_window = 5L)

# fit the model. It can take several minutes based on how much data you have

glove = GloVe$new(rank = 100, x\_max = 5)

glove$fit\_transform(tcm, n\_iter = 20)

# get the processed word vector

word\_vectors = glove$components

```

# Step 5 – check which words have contextual similarity, expand to 20 if there are too

# many irrelevant words

# check for nintendo first

```{r}

pos\_reviews\_data <- word\_vectors[, "saver", drop = F]

# cosine similarity between word vectors tells us how similar they are

cos\_sim = sim2(x = t(word\_vectors), y = t(pos\_reviews\_data), method = "cosine", norm = "l2")

head(sort(cos\_sim[,1], decreasing = TRUE), 10)

```

# Step 6 – implement a quick t-SNE to visualize reviews by similarity of words

# load packages

```{r}

pacman::p\_load(tm, Rtsne, tibble, tidytext, scales)

#create vector of words to keep, before applying tsne (remove stop words)

keep\_words <- setdiff(colnames(word\_vectors), stopwords())

# keep words in vector

word\_vec <- word\_vectors[, keep\_words]

# prepare data frame to train

train\_df <- data.frame(t(word\_vec)) %>% rownames\_to\_column("word")

# train tsne for visualization

tsne <- Rtsne(train\_df[,-1], dims = 2, perplexity = 50, verbose=TRUE, max\_iter = 500)

# t-SNE maps high dimensional data such as word embedding into a lower dimension # in such that the distance between two words roughly describe the similarity. # Additionally, t-SNE begins to create naturally forming clusters.

# Interpretation of a t-SNE is straightforward

# Similar objects (or words) appear nearby each other in the plot and

# dissimilar objects appear far away from each other.

```

# Step 7 – plot the t-SNE and examine it

# create plot

```{r}

set.seed(12345)

colors = rainbow(length(unique(train\_df$word)))

names(colors) = unique(train\_df$word)

plot\_df <- data.frame(tsne$Y) %>% mutate(

word = train\_df$word,

col = colors[train\_df$word]

) %>% left\_join(vocab, by = c("word" = "term")) %>%

filter(doc\_count >= 10)

ggplot(plot\_df, aes(X1, X2)) +

geom\_text(aes(X1, X2, label = word, color = col), size = 3) +

xlab("") + ylab("") +

theme(legend.position = "none")

```

# Step 3 – start by getting the sentence level sentiment for testing

# get sentence-level sentiment

```{r}

# create a new lexicon with modified sentiment

nrow(lexicon::hash\_sentiment\_jockers\_rinker)

review\_lexicon <- lexicon::hash\_sentiment\_jockers\_rinker

sent\_df <- pos\_reviews %>%

get\_sentences() %>%

sentiment\_by(by = c('ID', 'comments'), polarity\_dt = review\_lexicon)

```

# Step 8 – calculate word level sentiment and overlay these on the t-SNE

# calculate word-level sentiment

```{r}

word\_sent <- sent\_df %>%

select(ID, comments, ave\_sentiment) %>%

unnest\_tokens(word, comments) %>%

group\_by(word) %>%

summarise(

count = n(),

avg\_sentiment = mean(ave\_sentiment),

sum\_sentiment = sum(ave\_sentiment),

sd\_sentiment = sd(ave\_sentiment)) %>%

anti\_join(stop\_words, by = "word")

# remove stop words

# filter to words that appear at least 5 times

pd\_sent <- plot\_df %>%

left\_join(word\_sent, by = "word") %>%

drop\_na() %>%

filter(count >=5)

```

# Step 9 – Plot the results

```{r}

ggplot(pd\_sent, aes(X1, X2)) +

geom\_point(aes(X1, X2, size = count, alpha = .1, color = avg\_sentiment)) +

geom\_text(aes(X1, X2, label = word), size = 2) +

scale\_colour\_gradient2(low = muted("red"), mid = "white",

high = muted("blue"), midpoint = 0) +

scale\_size(range = c(5, 20)) +

xlab("") + ylab("") +

ggtitle("2-dimensional t-SNE Mapping of Word Vectors") +

guides(color = guide\_legend(title="Avg. Sentiment"), size = guide\_legend(title = "Frequency"), alpha = NULL) +

scale\_alpha(range = c(1, 1), guide = "none")

```

########################################

```{r include=FALSE}

tokens <- space\_tokenizer(neg\_reviews$comments %>% tolower() %>% removePunctuation())

# Create vocabulary. Terms will be unigrams (simple words).

it <- itoken(tokens, progressbar = FALSE)

vocab <- create\_vocabulary(it)

# prune (remove) words that appear less than 3 times

vocab <- prune\_vocabulary(vocab, term\_count\_min = 3L)

# Use our filtered vocabulary

vectorizer <- vocab\_vectorizer(vocab)

# use skip gram window of 5 for context words

tcm <- create\_tcm(it, vectorizer, skip\_grams\_window = 5L)

# fit the model. It can take several minutes based on how much data you have

glove = GloVe$new(rank = 100, x\_max = 5)

glove$fit\_transform(tcm, n\_iter = 20)

# get the processed word vector

word\_vectors = glove$components

```

# Step 5 – check which words have contextual similarity, expand to 20 if there are too

# many irrelevant words

# check for nintendo first

```{r}

neg\_reviews\_data <- word\_vectors[, "saver", drop = F]

# cosine similarity between word vectors tells us how similar they are

cos\_sim = sim2(x = t(word\_vectors), y = t(pos\_reviews\_data), method = "cosine", norm = "l2")

head(sort(cos\_sim[,1], decreasing = TRUE), 10)

```

# Step 6 – implement a quick t-SNE to visualize reviews by similarity of words

# load packages

```{r}

set.seed(12345)

pacman::p\_load(tm, Rtsne, tibble, tidytext, scales)

#create vector of words to keep, before applying tsne (remove stop words)

keep\_words <- setdiff(colnames(word\_vectors), stopwords())

# keep words in vector

word\_vec <- word\_vectors[, keep\_words]

# prepare data frame to train

train\_df <- data.frame(t(word\_vec)) %>% rownames\_to\_column("word")

# train tsne for visualization

tsne <- Rtsne(train\_df[,-1], dims = 2, perplexity = 50, verbose=TRUE, max\_iter = 500)

# t-SNE maps high dimensional data such as word embedding into a lower dimension # in such that the distance between two words roughly describe the similarity. # Additionally, t-SNE begins to create naturally forming clusters.

# Interpretation of a t-SNE is straightforward

# Similar objects (or words) appear nearby each other in the plot and

# dissimilar objects appear far away from each other.

```

# Step 7 – plot the t-SNE and examine it

# create plot

```{r}

colors = rainbow(length(unique(train\_df$word)))

names(colors) = unique(train\_df$word)

plot\_df <- data.frame(tsne$Y) %>% mutate(

word = train\_df$word,

col = colors[train\_df$word]

) %>% left\_join(vocab, by = c("word" = "term")) %>%

filter(doc\_count >= 10)

ggplot(plot\_df, aes(X1, X2)) +

geom\_text(aes(X1, X2, label = word, color = col), size = 3) +

xlab("") + ylab("") +

theme(legend.position = "none")

```

# Step 3 – start by getting the sentence level sentiment for testing

# get sentence-level sentiment

```{r}

# create a new lexicon with modified sentiment

nrow(lexicon::hash\_sentiment\_jockers\_rinker)

review\_lexicon <- lexicon::hash\_sentiment\_jockers\_rinker

sent\_df <- neg\_reviews %>%

get\_sentences() %>%

sentiment\_by(by = c('ID', 'comments'), polarity\_dt = review\_lexicon)

```

# Step 8 – calculate word level sentiment and overlay these on the t-SNE

# calculate word-level sentiment

```{r}

word\_sent <- sent\_df %>%

select(ID, comments, ave\_sentiment) %>%

unnest\_tokens(word, comments) %>%

group\_by(word) %>%

summarise(

count = n(),

avg\_sentiment = mean(ave\_sentiment),

sum\_sentiment = sum(ave\_sentiment),

sd\_sentiment = sd(ave\_sentiment)) %>%

anti\_join(stop\_words, by = "word")

# remove stop words

# filter to words that appear at least 5 times

pd\_sent <- plot\_df %>%

left\_join(word\_sent, by = "word") %>%

drop\_na() %>%

filter(count >=5)

```

# Step 9 – Plot the results

```{r}

ggplot(pd\_sent, aes(X1, X2)) +

geom\_point(aes(X1, X2, size = count, alpha = .1, color = avg\_sentiment)) +

geom\_text(aes(X1, X2, label = word), size = 3) +

scale\_colour\_gradient2(low = muted("red"), mid = "white",

high = muted("blue"), midpoint = 0) +

scale\_size(range = c(5, 20)) +

xlab("") + ylab("") +

ggtitle("2-dimensional t-SNE Mapping of Word Vectors") +

guides(color = guide\_legend(title="Avg. Sentiment"), size = guide\_legend(title = "Frequency"), alpha = NULL) +

scale\_alpha(range = c(1, 1), guide = "none")

```

########################################

```{r}

# Step 1 – add the required libraries and the data unless you have it already loaded

# Install and load the required packages.

pacman::p\_load(dplyr, ggplot2, stringr, udpipe, lattice)

```

```{r}

# udpipe needs a model file loaded. This file can be downloaded and needs to # be placed in the working directory for it to loaded, otherwise full path is # needed. This is an important step.

# MAKE SURE YOUR WORKING DIRECTORY IS SET TO WHERE THIS FILE IS LOCATED.

udmodel\_english <- udpipe\_load\_model(file = "english-ewt-ud-2.5-191206.udpipe")

```

```{r}

# Step 2 – count the number of total headlines by date and plot the results to examine

pos\_reviews %>% group\_by(stars) %>% count() %>% arrange(desc(n))

pos\_reviews %>% group\_by(stars) %>% count() %>% ggplot() + geom\_line(aes(stars,n, group = 1))

neg\_reviews %>% group\_by(stars) %>% count() %>% arrange(desc(n))

neg\_reviews %>% group\_by(stars) %>% count() %>% ggplot() + geom\_line(aes(stars,n, group = 1))

```

```{r}

# Step 2 – count the number of total headlines by date and plot the results to examine

reviews\_orig %>% group\_by(stars) %>% count() %>% arrange(desc(n))

reviews\_orig %>% group\_by(stars) %>% count() %>% ggplot() + geom\_line(aes(stars,n, group = 1))

```

```{r}

# Step 4 – use udpipe to annotate the text in the headlines for 2016 and load into a frame # this may take a while depending on how much data you are analyzing.

s1 <- udpipe\_annotate(udmodel\_english, pos\_reviews$comments)

x1 <- data.frame(s1)

s2 <- udpipe\_annotate(udmodel\_english, neg\_reviews$comments)

x2 <- data.frame(s2)

```

```{r}

# Step 5 – extract and display frequencies for universal parts of speech (upos) in text

stats1 <- txt\_freq(x1$upos)

stats1$key <- factor(stats1$key, levels = rev(stats1$key))

barchart(key ~ freq, data = stats1, col = "yellow",

main = "UPOS (Universal Parts of Speech)\n frequency of occurrence Positive Reviews",

xlab = "Freq")

stats2 <- txt\_freq(x2$upos)

stats2$key <- factor(stats2$key, levels = rev(stats2$key))

barchart(key ~ freq, data = stats2, col = "yellow",

main = "UPOS (Universal Parts of Speech)\n frequency of occurrence Negative Reviews",

xlab = "Freq")

```

```{r}

# Step 5 –extract and display most occurring nouns in the headlines

## NOUNS – change the number from 25 to lower or higher as applicable.

stats1 <- subset(x1, upos %in% c("NOUN"))

stats1 <- txt\_freq(stats1$token)

stats1$key <- factor(stats1$key, levels = rev(stats1$key))

barchart(key ~ freq, data = head(stats1, 25), col = "cadetblue", main = "Most occurring nouns in Positive Reviews", xlab = "Freq")

stats2 <- subset(x2, upos %in% c("NOUN"))

stats2 <- txt\_freq(stats2$token)

stats2$key <- factor(stats2$key, levels = rev(stats2$key))

barchart(key ~ freq, data = head(stats2, 25), col = "cadetblue", main = "Most occurring nouns in Negative Reviews", xlab = "Freq")

```

```{r}

# Step 6 –extract and display most occurring adjectives in the headlines

## ADJECTIVES - change the number from 25 to lower or higher as applicable.

stats1 <- subset(x1, upos %in% c("ADJ"))

stats1 <- txt\_freq(stats1$token)

stats1$key <- factor(stats1$key, levels = rev(stats1$key))

barchart(key ~ freq, data = head(stats1, 25), col = "purple", main = "Most occurring adjectives in Positive Reviews", xlab = "Freq")

stats2 <- subset(x2, upos %in% c("ADJ"))

stats2 <- txt\_freq(stats2$token)

stats2$key <- factor(stats2$key, levels = rev(stats2$key))

barchart(key ~ freq, data = head(stats2, 25), col = "purple", main = "Most occurring adjectives in Negative Reviews", xlab = "Freq")

```

```{r}

# Step 7 –extract and display most occurring verbs in the headlines

## VERBS - change the number from 25 to lower or higher as applicable.

stats1 <- subset(x1, upos %in% c("VERB"))

stats1 <- txt\_freq(stats1$token)

stats1$key <- factor(stats1$key, levels = rev(stats1$key))

barchart(key ~ freq, data = head(stats1, 25), col = "gold", main = "Most occurring Verbs in Positive Reviews", xlab = "Freq")

stats2 <- subset(x2, upos %in% c("VERB"))

stats2 <- txt\_freq(stats2$token)

stats2$key <- factor(stats2$key, levels = rev(stats2$key))

barchart(key ~ freq, data = head(stats2, 25), col = "gold", main = "Most occurring Verbs in Negative Reviews", xlab = "Freq")

```

```{r}

# Step 8 – Use RAKE (Rapid Automatic Keyword Extraction algorithm) to # determine key phrases in a body of text by analyzing the frequency of word appearance # and its co-occurrence with other words in the text.

## RAKE - Adjust the frequency and the number of results by changing 3 and 25 ## appropriate numbers for your dataset.

stats1 <- keywords\_rake(x = x1, term = "lemma", group = "doc\_id", relevant = x1$upos %in% c("NOUN", "ADJ"))

stats1$key <- factor(stats1$keyword, levels = rev(stats1$keyword))

barchart(key ~ rake, data = head(subset(stats1, freq > 3), 25), col = "red", main = "Keywords identified by RAKE in Positive Reviews", xlab = "Rake")

stats2 <- keywords\_rake(x = x2, term = "lemma", group = "doc\_id", relevant = x2$upos %in% c("NOUN", "ADJ"))

stats2$key <- factor(stats2$keyword, levels = rev(stats2$keyword))

barchart(key ~ rake, data = head(subset(stats2, freq > 3), 25), col = "red", main = "Keywords identified by RAKE in Negative Reviews", xlab = "Rake")

```

```{r}

# Step 9 – In English (and in many other languages a phrase can be formed simply with a # noun and a verb (e.g., cat meows) This may be useful for understanding context of a # sentence or a review or headlines especially if they are clickbait like. This step is to just # extract top phrases that are keyword-topics.

## display by plot a sequence of POS tags (noun phrases / verb phrases)

## Adjust the frequency and the number of results by changing 3 and 25

## to appropriate numbers for your dataset.

## You can also change the ngram levels to higher than 2 (like 3 or 4)

## to get lengths of 3 word or 4 word combinations.

x1$phrase\_tag <- as\_phrasemachine(x1$upos, type = "upos")

stats <- keywords\_phrases(x = x1$phrase\_tag, term = tolower(x1$token), pattern = "(A|N)\*N(P+D\*(A|N)\*N)\*", is\_regex = TRUE, detailed = FALSE)

stats1 <- subset(stats1, ngram > 1 & freq > 3)

stats1$key <- factor(stats1$keyword, levels = rev(stats1$keyword))

barchart(key ~ freq, data = head(stats1, 25), col = "magenta", main = "Keywords - simple noun phrases in Positive Reviews", xlab = "Frequency")

x2$phrase\_tag <- as\_phrasemachine(x2$upos, type = "upos")

stats2 <- keywords\_phrases(x = x2$phrase\_tag, term = tolower(x2$token), pattern = "(A|N)\*N(P+D\*(A|N)\*N)\*", is\_regex = TRUE, detailed = FALSE)

stats2 <- subset(stats2, ngram > 1 & freq > 3)

stats2$key <- factor(stats2$keyword, levels = rev(stats2$keyword))

barchart(key ~ freq, data = head(stats2, 25), col = "magenta", main = "Keywords - simple noun phrases in Negative Reviews", xlab = "Frequency")

```

```{r}

# Step 10 –it would be helpful to explore the words that appear next to each other. We can

# do this with just nouns and adjectives to explore # the patterns to get focus topic areas. # Adjust the ngram max levels if needed. It is set to 4 to indicate that we want

# co-occurrences within 3 words of each other.

## Collocation identification – basically words following one another)

stats1 <- keywords\_collocation(x = x1, term = "token", group = c("doc\_id", "paragraph\_id", "sentence\_id"), ngram\_max = 4)

stats2 <- keywords\_collocation(x = x2, term = "token", group = c("doc\_id", "paragraph\_id", "sentence\_id"), ngram\_max = 4)

```

```{r}

## How frequently do words occur in the same sentence (nouns and adjectives)

stats1 <- cooccurrence(x = subset(x1, upos %in% c("NOUN", "ADJ")), term = "lemma", group = c("doc\_id", "paragraph\_id", "sentence\_id"))

stats2 <- cooccurrence(x = subset(x2, upos %in% c("NOUN", "ADJ")), term = "lemma", group = c("doc\_id", "paragraph\_id", "sentence\_id"))

```

```{r}

## Co-occurrences: How frequent do words follow one another

stats1 <- cooccurrence(x = x1$lemma, relevant = x1$upos %in% c("NOUN", "ADJ"))

stats2 <- cooccurrence(x = x2$lemma, relevant = x2$upos %in% c("NOUN", "ADJ"))

```

```{r}

## Co-occurrences: How frequent do words follow one another even if we would ## skip 2 words in between. You can adjust this if you need to.

stats1 <- cooccurrence(x = x1$lemma, relevant = x1$upos %in% c("NOUN", "ADJ"), skipgram = 2)

stats2 <- cooccurrence(x = x2$lemma, relevant = x2$upos %in% c("NOUN", "ADJ"), skipgram = 2)

#head(stats)

```

```{r}

# Step 11 – it may be helpful to explore this visually instead of inspecting it in a table. To

# do this you will need the igraph library and the ggraph library. Load these with pacman.

# adjust the number you would like displayed. It is now set to 25.

pacman::p\_load(igraph, ggraph)

wordnetwork <- head(stats1, 25)

wordnetwork <- graph\_from\_data\_frame(wordnetwork)

ggraph(wordnetwork, layout = "fr") + geom\_edge\_link(aes(width = cooc, edge\_alpha = cooc), edge\_colour = "red") +

geom\_node\_text(aes(label = name), col = "darkgreen", size = 4) +

# theme\_graph(base\_family = windowsFont("TT Arial")) +

theme(legend.position = "none") +

labs(title = "Co-occurrences within 3 words distance in Positive Reviews", subtitle = "Nouns & Adjectives")

pacman::p\_load(igraph, ggraph)

wordnetwork <- head(stats2, 25)

wordnetwork <- graph\_from\_data\_frame(wordnetwork)

ggraph(wordnetwork, layout = "fr") + geom\_edge\_link(aes(width = cooc, edge\_alpha = cooc), edge\_colour = "red") +

geom\_node\_text(aes(label = name), col = "darkgreen", size = 4) +

# theme\_graph(base\_family = windowsFont("TT Arial")) +

theme(legend.position = "none") +

labs(title = "Co-occurrences within 3 words distance in Negative Reviews", subtitle = "Nouns & Adjectives")

```

# Step 1 – load the required libraries and the data

```{r}

library(tidyverse) # organize workflow and for all text work

library(tidytext) # contains the NLP methods we need

library(topicmodels) # for LDA topic modelling – our main package

library(tm) # general text mining functions, DTM work.

library(SnowballC) # for stemming when needed.

library(stringr) # for cleaning

```

#using read\_csv instead of read.csv to avoid the stringAsFactors issue

#read\_csv is also known to be faster at reading large datasets

# Step 2 – Clean the data. This is an important step, check column names for your dataset

```{r}

#clean the review data, our reviews are in the ‘text’ column of the dataset

pos\_reviews$comments <- str\_replace\_all(pos\_reviews$comments,"[^[:graph:]]", " ")

neg\_reviews$comments <- str\_replace\_all(neg\_reviews$comments,"[^[:graph:]]", " ")

```

# Step 3– write a function that allows us to extract topic models using only TF and IDF

# Please make sure you copy paste the whole function if you plan without modifications.

# modifications. It is important to understand how this function works.

# First get the count of each word in each review

# Second, get the number of words per text# input.

# Third, combine these dfs and get the tf\_idf & order the words by degree of relevance

# Finally, plot the 10 most informative terms per topic grouped by the grouping covariate

#note that this approach is for those who know the details of the corpus.

# function that takes in a dataframe and the name of the columns

# with the document texts and the topic labels. If plot is set to

# false it will return the tf-idf output rather than a plot.

```{r}

top\_terms\_by\_topic\_tfidf <- function(text\_df, text\_column, group\_column, plot = T){

# name for the column we're going to unnest\_tokens\_ to

# (you only need to worry about enough stuff if you're

# writing a function using using tidyverse packages)

group\_column <- enquo(group\_column)

text\_column <- enquo(text\_column)

# get the count of each word in each review

words <- text\_df %>%

unnest\_tokens(word, !!text\_column) %>%

count(!!group\_column, word) %>%

ungroup()

# get the number of words per text

total\_words <- words %>%

group\_by(!!group\_column) %>%

summarize(total = sum(n))

# combine the two dataframes we just made

words <- left\_join(words, total\_words)

# get the tf\_idf & order the words by degree of relevance

tf\_idf <- words %>%

bind\_tf\_idf(word, !!group\_column, n) %>%

select(-total) %>%

arrange(desc(tf\_idf)) %>%

mutate(word = factor(word, levels = rev(unique(word))))

if(plot == T){

# convert "group" into a quote of a name

# (this is due to idiosyncrasies with calling ggplot2

# in functions)

group\_name <- quo\_name(group\_column)

# plot the 10 most informative terms per topic

tf\_idf %>%

group\_by(!!group\_column) %>%

top\_n(10) %>%

ungroup %>%

ggplot(aes(word, tf\_idf, fill = as.factor(group\_name))) +

geom\_col(show.legend = FALSE) +

labs(x = NULL, y = "tf-idf") +

facet\_wrap(reformulate(group\_name), scales = "free") +

coord\_flip()

}else{

# return the entire tf\_idf dataframe

return(tf\_idf)

}

}

```

# Step 4– get the top terms associated with each topic in our text.

# Choose and group the terms by the grouping covariate.

# here we are using the stars as the grouping variable.

# if you use large datasets be aware of how this will affect your readability

```{r}

pos\_reviews\_tfidf\_stars <- top\_terms\_by\_topic\_tfidf(text\_df = pos\_reviews, text\_column = comments, group = stars, plot = F)

pos\_reviews\_tfidf\_stars %>%

group\_by(stars) %>%

top\_n(5) %>%

ungroup %>%

ggplot(aes(word, tf\_idf, fill = stars)) +

geom\_col(show.legend = FALSE) +

labs(x = NULL, y = "tf-idf") +

facet\_wrap(~stars, ncol = 4, scales = "free", ) +

coord\_flip()

neg\_reviews\_tfidf\_stars <- top\_terms\_by\_topic\_tfidf(text\_df = neg\_reviews, text\_column = comments, group = stars, plot = F)

neg\_reviews\_tfidf\_stars %>%

group\_by(stars) %>%

top\_n(5) %>%

ungroup %>%

ggplot(aes(word, tf\_idf, fill = stars)) +

geom\_col(show.legend = FALSE) +

labs(x = NULL, y = "tf-idf") +

facet\_wrap(~stars, ncol = 4, scales = "free", ) +

coord\_flip()

```

############################################

LDA

#Step 1 – load the required libraries and the data

```{r}

library(tidyverse) # organize workflow and for all text work

library(tidytext) # contains the NLP methods we need

library(topicmodels) # for LDA topic modelling – our main package

library(tm) # general text mining functions, DTM work.

library(SnowballC) # for stemming when needed.

library(stringr) # for cleaning

```

```{r}

#Step 3– write a function that allows us to extract topic models quickly and easily. Please # make sure you copy pate the whole function if you plan to use this function without

# modifications. It is important to understand how this function works.

# First this function creates a corpus from the text you provide

# Second, the function will then use the tidytext package, which expects a corpus as the

# input. The corpus is just a type of data object.

# Third, the topic models are extracted based on how many were requested when the

# function was invoked.

# Finally, the top 10 terms that best represent each topic are extracted and returned.

# function to get & plot the most informative terms by a specified number

# of topics, using LDA

top\_terms\_by\_topic\_LDA <- function(input\_text, # should be a column from a data frame

plot = T, # return a plot? TRUE by defult

number\_of\_topics = 4) # number of topics (4 by default)

{

# create a corpus (type of object expected by tm) and document term matrix

Corpus <- Corpus(VectorSource(input\_text)) # make a corpus object

DTM <- DocumentTermMatrix(Corpus) # get the count of words/document

# remove any empty rows in our document term matrix (if there are any

# we'll get an error when we try to run our LDA)

unique\_indexes <- unique(DTM$i) # get the index of each unique value

DTM <- DTM[unique\_indexes,] # get a subset of only those indexes

# preform LDA & get the words/topic in a tidy text format

lda <- LDA(DTM, k = number\_of\_topics, control = list(seed = 1234))

topics <- tidy(lda, matrix = "beta")

# get the top ten terms for each topic,

# yes I made up the word informativeness

top\_terms <- topics %>% # take the topics data frame and..

group\_by(topic) %>% # treat each topic as a different group

top\_n(10, beta) %>% # get the top 10 most informative words

ungroup() %>% # ungroup

arrange(topic, -beta) # arrange words in descending informativeness

# if the user asks for a plot (TRUE by default)

if(plot == T){

# plot the top ten terms for each topic in order

top\_terms %>% # take the top terms

mutate(term = reorder(term, beta)) %>% # sort terms by beta value

ggplot(aes(term, beta, fill = factor(topic))) + # plot beta by theme

geom\_col(show.legend = FALSE) + # as a bar plot

facet\_wrap(~ topic, scales = "free") + # which each topic in a separate plot

labs(x = NULL, y = "Beta") + # no x label, change y label

coord\_flip() # turn bars sideways

}else{

# if the user does not request a plot

# return a list of sorted terms instead

return(top\_terms)

}

}

```

#Step 4– Test out the function to ensure everything works by starting with two topics.

# This step also allows you to identify any irrelevant words that may need to be

# eliminated and added to a stop word list.

#get just two topics to see how things pan out, carefully check words to see

#what needs to be added to stop words.

```{r}

top\_terms\_by\_topic\_LDA(pos\_reviews$comments, number\_of\_topics = 2)

top\_terms\_by\_topic\_LDA(neg\_reviews$comments, number\_of\_topics = 2)

```

#Step 5 – We are ready for the topic modeling now. Create a tidytext corpus

# Make a list of edited and customized stop words for our needs.

```{r}

# pay attention to column names

reviewsCorpus <- Corpus(VectorSource(pos\_reviews$comments))

reviewsDTM <- DocumentTermMatrix(reviewsCorpus)

# convert the document term matrix to a tidytext corpus

reviewsDTM\_tidy <- tidy(reviewsDTM)

# I'm going to add my own custom stop words that I don't think will be

# very informative in these reviews. My first test topic model indicated that # I should add room and Chicago to the list. Case is not relevant

custom\_stop\_words <- tibble(word = c("saver", "onion", "tomato","lemon","fridge", "garlic","onions","container", "pepper","plastic", "half", "lemons", "product", "time", "close","time", "store", "bought", "tomatoes", "grapefruit", "refrigerator", "days", "easily", "easy","item", "peppers", "bottom", "storage","savers", "perfect", "containers", "lime", "limes", "savers", "keeper", "pieces", "food","makes", "save", "keeping", "love", "nice", "holder", "week", "save", "hold", "bags", "produce", "clean", "helps", "vegetable", "fruit", "wrap", "veggies", "holds"))

# remove stopwords from the dataset of reviews

reviewsDTM\_tidy\_cleaned <- reviewsDTM\_tidy %>% # take our tidy dtm and...

anti\_join(stop\_words, by = c("term" = "word")) %>% # remove English stopwords and...

anti\_join(custom\_stop\_words, by = c("term" = "word")) # remove my custom stopwords

# reconstruct cleaned documents (so that each word shows up the correct number of times)

cleaned\_documents <- reviewsDTM\_tidy\_cleaned %>%

group\_by(document) %>%

mutate(terms = toString(rep(term, count))) %>%

select(document, terms) %>%

unique()

# check out what the cleaned documents look like (should just be a bunch of content words)

# in alphabetic order

head(cleaned\_documents)

#Step 6 – Start obtaining topic models. If you know enough about the reviews you will

# have a great starting point in the number of topics sought, otherwise we will have to

# make several models.

#try out two topics, expand to 3 or 4 or more. I found three topics to be

# ideal in this specific 2k review set but in the larger set I needed 4.

top\_terms\_by\_topic\_LDA(cleaned\_documents, number\_of\_topics = 2)

top\_terms\_by\_topic\_LDA(cleaned\_documents, number\_of\_topics = 3)

top\_terms\_by\_topic\_LDA(cleaned\_documents, number\_of\_topics = 4)

# Step 7 – Stemming is a controversial topic when it comes to topic models.

# Some research indicates that stemming actually harms the creation of topic models,

# but some data scientists claim that stemming increases interpretability.

# stem the words (e.g. convert each word to its stem, where applicable)

reviewsDTM\_tidy\_cleaned <- reviewsDTM\_tidy\_cleaned %>%

mutate(stem = wordStem(term))

# reconstruct our documents

cleaned\_documents <- reviewsDTM\_tidy\_cleaned %>%

group\_by(document) %>%

mutate(terms = toString(rep(stem, count))) %>%

select(document, terms) %>%

unique()

#Step 8 – Revisit the topic models and create the stemmed word topic models.

#try out the lower end of what was acceptable from step 6. This was 3 for me

#I then tried 4 for the larger set which seemed to work well enough.

# now let's look at the new most informative terms

top\_terms\_by\_topic\_LDA(cleaned\_documents$terms, number\_of\_topics = 3)

top\_terms\_by\_topic\_LDA(cleaned\_documents$terms, number\_of\_topics = 4)

```

#Step 5 – We are ready for the topic modeling now. Create a tidytext corpus

# Make a list of edited and customized stop words for our needs.

```{r}

# pay attention to column names

reviewsCorpus <- Corpus(VectorSource(neg\_reviews$comments))

reviewsDTM <- DocumentTermMatrix(reviewsCorpus)

# convert the document term matrix to a tidytext corpus

reviewsDTM\_tidy <- tidy(reviewsDTM)

# I'm going to add my own custom stop words that I don't think will be

# very informative in these reviews. My first test topic model indicated that # I should add room and Chicago to the list. Case is not relevant

custom\_stop\_words <- tibble(word = c("saver", "onion", "tomato","lemon","fridge", "garlic","onions","container", "pepper","plastic", "half", "lemons", "product", "time", "close","time", "store", "bought", "tomatoes", "grapefruit", "refrigerator", "days", "easily", "easy","item", "peppers", "bottom", "storage","savers", "perfect", "containers", "lime", "limes", "savers", "keeper", "pieces", "food","makes", "save", "keeping", "love", "nice", "holder", "week", "save", "hold", "bags", "produce", "clean", "helps", "vegetable", "fruit", "wrap", "veggies", "holds", "i've"))

# remove stopwords from the dataset of reviews

reviewsDTM\_tidy\_cleaned <- reviewsDTM\_tidy %>% # take our tidy dtm and...

anti\_join(stop\_words, by = c("term" = "word")) %>% # remove English stopwords and...

anti\_join(custom\_stop\_words, by = c("term" = "word")) # remove my custom stopwords

# reconstruct cleaned documents (so that each word shows up the correct number of times)

cleaned\_documents <- reviewsDTM\_tidy\_cleaned %>%

group\_by(document) %>%

mutate(terms = toString(rep(term, count))) %>%

select(document, terms) %>%

unique()

# check out what the cleaned documents look like (should just be a bunch of content words)

# in alphabetic order

head(cleaned\_documents)

#Step 6 – Start obtaining topic models. If you know enough about the reviews you will

# have a great starting point in the number of topics sought, otherwise we will have to

# make several models.

#try out two topics, expand to 3 or 4 or more. I found three topics to be

# ideal in this specific 2k review set but in the larger set I needed 4.

top\_terms\_by\_topic\_LDA(cleaned\_documents, number\_of\_topics = 2)

top\_terms\_by\_topic\_LDA(cleaned\_documents, number\_of\_topics = 3)

top\_terms\_by\_topic\_LDA(cleaned\_documents, number\_of\_topics = 4)

# Step 7 – Stemming is a controversial topic when it comes to topic models.

# Some research indicates that stemming actually harms the creation of topic models,

# but some data scientists claim that stemming increases interpretability.

# stem the words (e.g. convert each word to its stem, where applicable)

reviewsDTM\_tidy\_cleaned <- reviewsDTM\_tidy\_cleaned %>%

mutate(stem = wordStem(term))

# reconstruct our documents

cleaned\_documents <- reviewsDTM\_tidy\_cleaned %>%

group\_by(document) %>%

mutate(terms = toString(rep(stem, count))) %>%

select(document, terms) %>%

unique()

#Step 8 – Revisit the topic models and create the stemmed word topic models.

#try out the lower end of what was acceptable from step 6. This was 3 for me

#I then tried 4 for the larger set which seemed to work well enough.

# now let's look at the new most informative terms

top\_terms\_by\_topic\_LDA(cleaned\_documents$terms, number\_of\_topics = 3)

top\_terms\_by\_topic\_LDA(cleaned\_documents$terms, number\_of\_topics = 4)

```

```{r}

#split dataset as per ratings

reviews\_orig[c("n\_helpful")][is.na(reviews\_orig[c("n\_helpful")])] <- 0

lst1 <- split(reviews\_orig,reviews\_orig$n\_helpful<1)

helpful\_reviews <- as.data.frame(lst1[1])

nothelpful\_reviews <- as.data.frame(lst1[2])

names(helpful\_reviews)[names(helpful\_reviews) == "FALSE.stars"] <- "stars"

names(helpful\_reviews)[names(helpful\_reviews) == "FALSE.comments"] <- "comments"

names(helpful\_reviews)[names(helpful\_reviews) == "FALSE.n\_helpful"] <- "n\_helpful"

names(nothelpful\_reviews)[names(nothelpful\_reviews) == "TRUE.stars"] <- "stars"

names(nothelpful\_reviews)[names(nothelpful\_reviews) == "TRUE.comments"] <- "comments"

names(nothelpful\_reviews)[names(nothelpful\_reviews) == "TRUE.n\_helpful"] <- "n\_helpful"

# select only the stars and comments from the data and examine it

helpful\_reviews <-helpful\_reviews %>% select(stars, comments, n\_helpful)

helpful\_reviews$ID <- seq.int(nrow(helpful\_reviews)) #Adding ID column

glimpse(helpful\_reviews)

nothelpful\_reviews <- nothelpful\_reviews %>% select(stars, comments, n\_helpful)

nothelpful\_reviews$ID <- seq.int(nrow(nothelpful\_reviews)) #Adding ID column

glimpse(nothelpful\_reviews)

```

```{r}

# Step 2 – count the number of total headlines by date and plot the results to examine

helpful\_reviews %>% group\_by(stars) %>% count() %>% arrange(desc(n))

nothelpful\_reviews %>% group\_by(stars) %>% count() %>% ggplot() + geom\_line(aes(stars,n, group = 1))

```

```{r}

helpful\_reviews$total <- sapply(helpful\_reviews$comments, function(x) length(unlist(strsplit(as.character(x), "\\W+"))))

nothelpful\_reviews$total <- sapply(nothelpful\_reviews$comments, function(x) length(unlist(strsplit(as.character(x), "\\W+"))))

```

```{r}

mean(nothelpful\_reviews$total)

mean(helpful\_reviews$total)

```