Exploratory Data Analysis on Text Data

Promothesh Chatterjee*

Exploratory Analysis of Text Data

In these notes, we will try to answer some questions based on a hotel reviews dataset.

Assume you are the data analyst for Marriott:

- Identify how consumers view Marriott vis-a-vis competitors Hilton and Hyatt
- Are consumers more happy with competitors than Marriott?
- Do unhappy customers write longer reviews?
- Does it vary by the type of device used to write reviews?
- Can we extract and visualize key words in these reviews?

Let's read the original hotel reviews file in R and take a glimpse of the dataset.

```
# Load up the .CSV data and explore.
library(tidyverse)
hotel_raw <- read_csv("C:/Users/u0474728/Dropbox/Utah Department Stuff/Teaching/Text Analysis/Summer 20
glimpse(hotel_raw)
```

```
## Rows: 38,932
## Columns: 5
## $ User_ID      <chr> "id10326", "id10327", "id10328", "id10329", "id10330", "i~
## $ Description  <chr> "The room was kind of clean but had a VERY strong smell o~
## $ Browser_Used <chr> "Edge", "Internet Explorer", "Mozilla", "InternetExplorer~
## $ Device_Used  <chr> "Mobile", "Mobile", "Tablet", "Desktop", "Tablet", "Deskt~
## $ Is_Response  <chr> "not happy", "not happy", "not happy", "happy", "not happ~
```

As you can see, there are 38,932 observations (that is 38,932 reviews) and 5 variables all of which have been coded by R as character variables (indicating text variables).

Let's check data to see if there are missing values.

```
sum(!complete.cases(hotel_raw)) # checking the number of incomplete cases
```

```
## [1] 0
```

The negation of function complete cases checks if there is any incomplete cases, and we simply sum it up to find the total number of incomplete cases.

Since there are no missing cases, let's take three relevant variables that are coded as character variables, and let's convert them to Factor (R parlance for categorical variables) as they would be relevant for future analysis.

```
hotel_raw$Is_Response <- as.factor(hotel_raw$Is_Response)
hotel_raw$Device_Used <- as.factor(hotel_raw$Device_Used)
hotel_raw$Browser_Used <- as.factor(hotel_raw$Browser_Used)
```

Let's focus on three hotels (Hilton, Hyatt and Marriott) which are present in the dataset and do a comparative analysis.

```
hotel_raw <- hotel_raw %>%
  mutate(hotel_name = case_when(str_detect(Description, "Hilton")~ "Hilton",
                               str_detect(Description, "Hyatt")~ "Hyatt",
                               str_detect(Description, "Marriott")~ "Marriott"))
hotel_sub <- hotel_raw %>% filter(hotel_name=="Hilton"|hotel_name=="Hyatt"|hotel_name=="Marriott")
```

This R code is manipulating a data frame called "hotel_raw" to create a new data frame called "hotel_sub" that only includes rows where the "hotel_name" column is "Hilton", "Hyatt", or "Marriott".

The code achieves this by first adding a new column called "hotel_name" to the "hotel_raw" data frame using the mutate() function from the dplyr package. The values in this new column are determined using a series of case_when() statements, which test whether certain patterns are found in the "Description" column of the data frame.

For example, the first case_when() statement checks whether the string "Hilton" is found in the "Description" column using the str_detect() function from the stringr package. If this is true, the value "Hilton" is assigned to the "hotel_name" column for that row. Similarly, the second case_when() statement assigns the value "Hyatt" if the string "Hyatt" is found in the "Description" column, and the third case_when() statement assigns the value "Marriott" if the string "Marriott" is found.

Finally, the filter() function from dplyr is used to create the new data frame "hotel_sub" by selecting only the rows from "hotel_raw" where the "hotel_name" column is "Hilton", "Hyatt", or "Marriott".

Now, first, let's take a look at distribution of the class labels (i.e., what proportion of consumers are happy vs. not happy).

```
hotel_sub %>%
count(Is_Response)%>%
  mutate(freq=n/sum(n))
```

```
## # A tibble: 2 x 3
##   Is_Response     n  freq
##   <fct>       <int> <dbl>
## 1 happy        2010 0.618
## 2 not happy    1244 0.382
```

The count function will simply provide a count of variable Is_Response, then we use mutate function from dplyr to create a new variable named 'freq' by taking the count and dividing by sum of the count. Roughly, 62% consumers are happy with the hotels and others are not.

Let's see if any hotel is represented disproportionately.

```
hotel_sub %>%
    group_by(hotel_name) %>%
count(Is_Response)%>%
  mutate(freq=n/sum(n))
```

```
## # A tibble: 6 x 4
## # Groups:   hotel_name [3]
##   hotel_name Is_Response     n  freq
##   <chr>      <fct>       <int> <dbl>
## 1 Hilton     happy         847 0.597
## 2 Hilton     not happy     572 0.403
## 3 Hyatt      happy         474 0.629
## 4 Hyatt      not happy     280 0.371
## 5 Marriott   happy         689 0.637
## 6 Marriott   not happy     392 0.363
```

Hilton folks seem to be most unhappy. Before looking at the content of text reviews, let's just examine the distribution of the length of the reviews. We will create a variable called ReviewLength which counts the number of characters in the reviews.

```
hotel_sub<-hotel_sub %>%
  mutate(ReviewLength=nchar(Description))

library(quanteda)
data_corpus_hotelsub <- corpus(hotel_sub, text_field = "Description")# for subsequent analysis
```

Now let's see if the review length varies by the type of device. It could be possible that people write less on a mobile device than on a laptop. We use the function group_by (which is typically used in conjunction with summarise). Using function summarize, we create a variable called AvLength which is just a mean of ReviewLength.

```
hotel_sub%>%
  group_by( hotel_name, Device_Used) %>%
  summarise(AvLength=mean(ReviewLength))
```

```
## # A tibble: 9 x 3
## # Groups:   hotel_name [3]
##   hotel_name Device_Used AvLength
##   <chr>      <fct>          <dbl>
## 1 Hilton     Desktop        1201.
## 2 Hilton     Mobile         1189.
```

```
## 3 Hilton     Tablet          1241.
## 4 Hyatt      Desktop         1268.
## 5 Hyatt      Mobile          1213.
## 6 Hyatt      Tablet          1236.
## 7 Marriott   Desktop         1113.
## 8 Marriott   Mobile          1195.
## 9 Marriott   Tablet          1209.
```

The output suggests that there is a minimal difference between number of characters in each review between the hotels and devices.

Now let's check if review length differs based on whether consumers are happy or not with the hotel. We might suspect that people who are not happy might post longer reviews, let's visualize that using the tidyverse package GGPlot. The first argument of ggplot is the data frame hotel_raw, next we indicate the x variable in the function aesthetics (aes) and use the command fill to separate the plot by the factor variable Is_Response.

```r
library(ggplot2)

ggplot(hotel_sub, aes(x = ReviewLength, fill = Is_Response)) +
  geom_histogram(binwidth = 5) +  facet_wrap(~ hotel_name)+
  labs(y = "Review Count", x = "Length of Review",
       title = "Distribution of Review Lengths with Class Labels")
```

The relationship between length of reviews and customer happiness does not seem to be clear cut. While the number of happy customers is definitely more than not-happy customers, the length of the review of not-happy customers seems to be slightly more only in the case of Marriott.

So far,we haven't even gone to NLP. Let's now do the preprocessing (we have seen these in the previous section). We essentially tokenize, remove stopwords and do other preprocessing to create document-feature-matrix.

```
library(quanteda)
hotel_sub_tokens <- quanteda::tokens(hotel_sub$Description, what = "word",
                        remove_numbers = TRUE, remove_punct = TRUE,
                        remove_symbols = TRUE)

#Create DFM
hotel_sub_dfm<-hotel_sub_tokens %>%
  tokens_remove(stopwords(source = "smart")) %>%
  tokens_wordstem() %>%
  tokens_tolower() %>%
  dfm()

dim(hotel_sub_dfm)
```

```
## [1]  3254 14158
```

```
head(hotel_sub_dfm, n = 5)
```

```
## Document-feature matrix of: 5 documents, 14,158 features (99.35% sparse) and 0 docvars.
##        features
## docs    wonder staff great locat defin price high standard hotel free
##   text1      1     1     2     1     1     1    1        1     2    2
##   text2      0     0     0     0     0     2    0        0     9    4
##   text3      0     3     0     0     0     0    1        0     0    0
##   text4      0     0     0     0     0     1    0        0     4    0
##   text5      0     1     1     1     0     0    2        0     4    2
## [ reached max_nfeat ... 14,148 more features ]
```

Check the presence of the word 'hotel' multiple times across reviews (refer to the notes on TFIDF, we will perform TFIDF shortly).

Concordances Concordancing in Text Analysis involves extracting words from one or more texts (Lindquist 2009). Typically, these concordances are presented as Key Word In Context (KWIC), which displays the search term along with some preceding and following context. As a result, these displays are commonly known as Key Word In Context (KWIC) concordances.

Concordancing allows one to observe the usage of a term in the data, examine the frequency of a particular word in a text or group of texts, and extract examples. It also serves as a fundamental procedure and frequently serves as the initial step in more advanced analyses of language data.

```
hilt <- data_corpus_hotelsub %>%
    kwic(pattern = "Hilton.*", window = 5, valuetype = "regex")
```

You can check out the hilt dataset to understand the context in which the word Hilton appears.

We can also search for phrase patterns.

```
kwic_hiltonhonors <- quanteda::kwic(x = data_corpus_hotelsub,
                          pattern = quanteda::phrase("Hilton Honors"),
                          window = 5)
```

Word Frequency The majority of techniques employed in text analysis heavily rely on frequency data. Consequently, identifying the most common words present in a given text is a fundamental approach in text analytics. Essentially, frequency data constitutes the backbone of text analysis, frequently appearing in the form of word frequency lists that consist of word forms and their corresponding frequency in a given text or text collection.

```r
dfmat2 <- corpus_subset(data_corpus_hotelsub, hotel_name == "Hilton") %>%
tokens(remove_punct = TRUE) %>%
tokens_remove(stopwords("en")) %>%
dfm()

library(quanteda.textstats)
tstat1 <- textstat_frequency(dfmat2, groups =hotel_name)
head(tstat1, 10)
```

```
##    feature frequency rank docfreq  group
## 1    hotel      3206    1    1128 Hilton
## 2     room      2954    2    1030 Hilton
## 3   hilton      2010    3    1358 Hilton
## 4       --      1397    4     603 Hilton
## 5     stay      1238    5     794 Hilton
## 6      one      1004    6     589 Hilton
## 7    great       960    7     581 Hilton
## 8    staff       932    8     680 Hilton
## 9     nice       869    9     532 Hilton
## 10   rooms       861   10     597 Hilton
```

This code snippet performs the following tasks:

1. **`corpus_subset(data_corpus_hotelsub, hotel_name == "Hilton")`** filters a corpus **`data_corpus_hotelsub`** to only include documents where the **`hotel_name`** metadata variable is equal to "Hilton". This creates a new corpus object **`dfmat2`**.

2. **`%>%`** is a pipe operator that passes the output of the previous function to the next function.

3. **`tokens(remove_punct = TRUE)`** converts the corpus into a token object and removes all punctuation marks.

4. **`tokens_remove(stopwords("en"))`** removes all stopwords (common words that are not very informative) using the English language stopword list.

5. **`dfm()`** converts the token object to a document-feature matrix (dfm), which is a matrix where the rows represent the documents and the columns represent the features (words in this case).

6. **`textstat_frequency(dfmat2, groups = hotel_name)`** calculates the frequency of each word in the **`dfmat2`** matrix and groups them by **`hotel_name`**. This creates a new object **`tstat1`** which contains the frequency counts for each word grouped by the hotel name.

7. **`head(tstat1, 10)`** displays the top 10 most frequent words for each hotel in the **`tstat1`** object.

Word Cloud Let's now generate a word cloud for further visualization. The idea is pretty simple, the word size indicates the frequency of occurrence of the word. We will use the textplot_wordcloud function from quanteda (again, you check the help file by typing ?textplot_wordcloud in the console to look at the details of the arguments passed)

```r
library(quanteda.textplots)
textplot_wordcloud(
  hotel_sub_dfm,
  min_size = 0.5,
  max_size = 4,
  min_count = 3,
  max_words = 200,
  color = "darkblue",
  font = NULL,
  adjust = 0,
  rotation = 0.1,
  random_order = FALSE,
  random_color = FALSE,
  ordered_color = FALSE,
  labelcolor = "gray20",
  labelsize = 1.5,
  labeloffset = 0,
  fixed_aspect = TRUE,
  comparison = FALSE
)
```

As can be seen, words such as hotel, stay, room and great seem to be the dominant words. We know that in our data, we have roughly 38% people who are not happy with the hotels. If we can split the word cloud based on happy vs. not happy people, we can understand the specific words used in each case. To accomplish that, we need to create a corpus out of the raw data and use function corpus_subset to split the DFM into happy and not happy groups.

Comparative Word Cloud For Labeled Data To create a comparative word cloud for happy versus not happy customers in Quanteda, we first need to create a corpus of the text (i.e., the reviews) and convert it to tokens. We can then use the tokens created to split the Document Feature Matrix into Happy versus not Happy Customers by using function "groups" on the factor variable Is_Response. In fact you can play around by changing the groups (remember we created three factor variables in the beginning of analysis.)

```r
hotel_raw1<-corpus(hotel_sub, text_field="Description")
toks_hotel<-quanteda::tokens(hotel_raw1, remove_punct = TRUE) %>%
  tokens_remove(stopwords("english"))

dfmat2 <- dfm(toks_hotel) %>% dfm_remove(stopwords("english")) %>%
              dfm_group(groups = Is_Response) %>%
              dfm_trim(min_termfreq = 3)

library(quanteda.textplots)
```

```r
textplot_wordcloud(dfmat2, comparison = TRUE, max_words = 100,min_size = 1,
  max_size = 4,
  min_count = 3,
  color = c("blue", "red"))
```

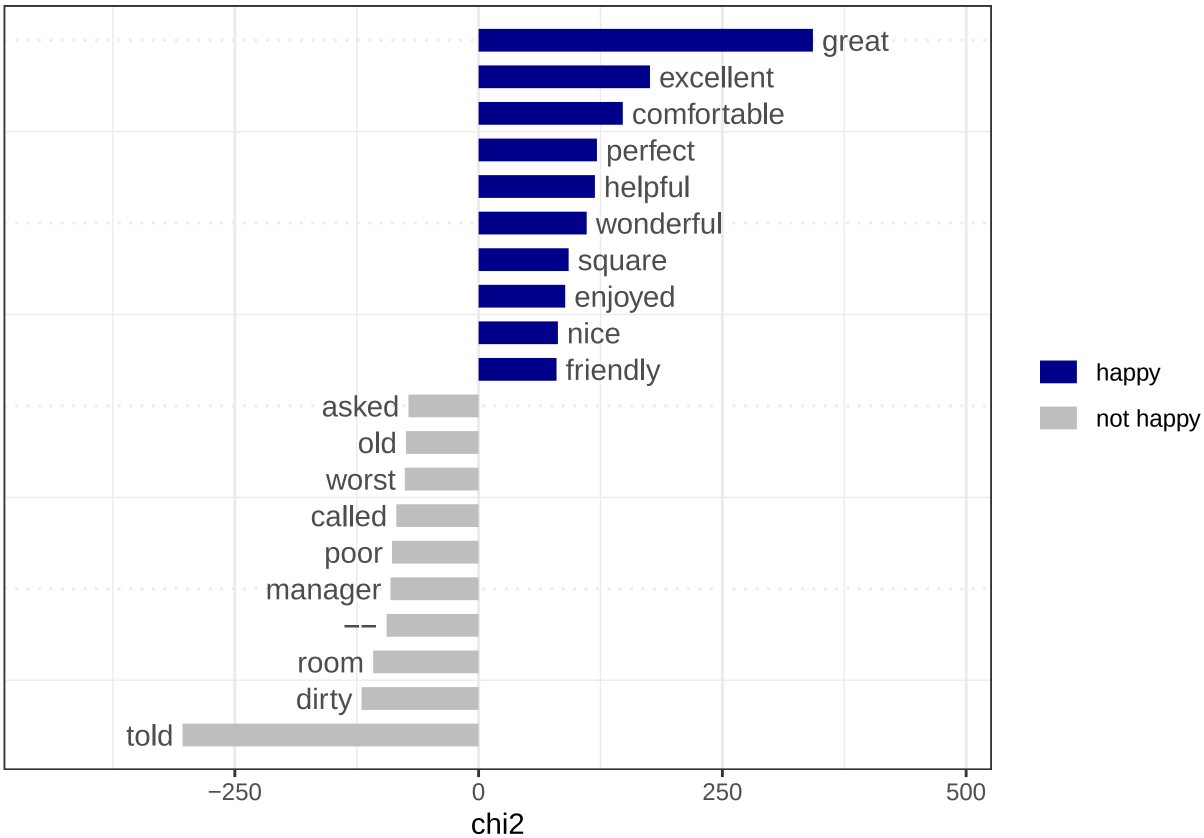
From the original word cloud, we knew the word "room" occurs frequently, but now we know that the "room" is more frequently used by not happy customers. We can further examine the issue using keyness of words analysis. Basically, we are asking Quanteda to generate keywords that are used by the happy and the not happy customers.

Next, let's look at Keyness of Words Analysis. The idea is fairly similar to that of comparative word cloud, except the graph is different.

Keyness of Words Analysis Keyness is a related concept that focuses on the relative frequency of a word in a particular context. It is used to determine the importance or significance of a word in a given text or corpus.

To generate Keyness of Words plot, we use the function textstat_keyness on the previously created DFM dfmat2. The argument target specifies that we are asking happy customers to be the top. Thus, we have generated top 10 words for happy and not happy customers.

```r
library(quanteda.textstats)
tstat1 <- textstat_keyness(dfmat2, target = "happy")
textplot_keyness(tstat1, margin = 0.2, n = 10)
```



Let's get the frequency weights

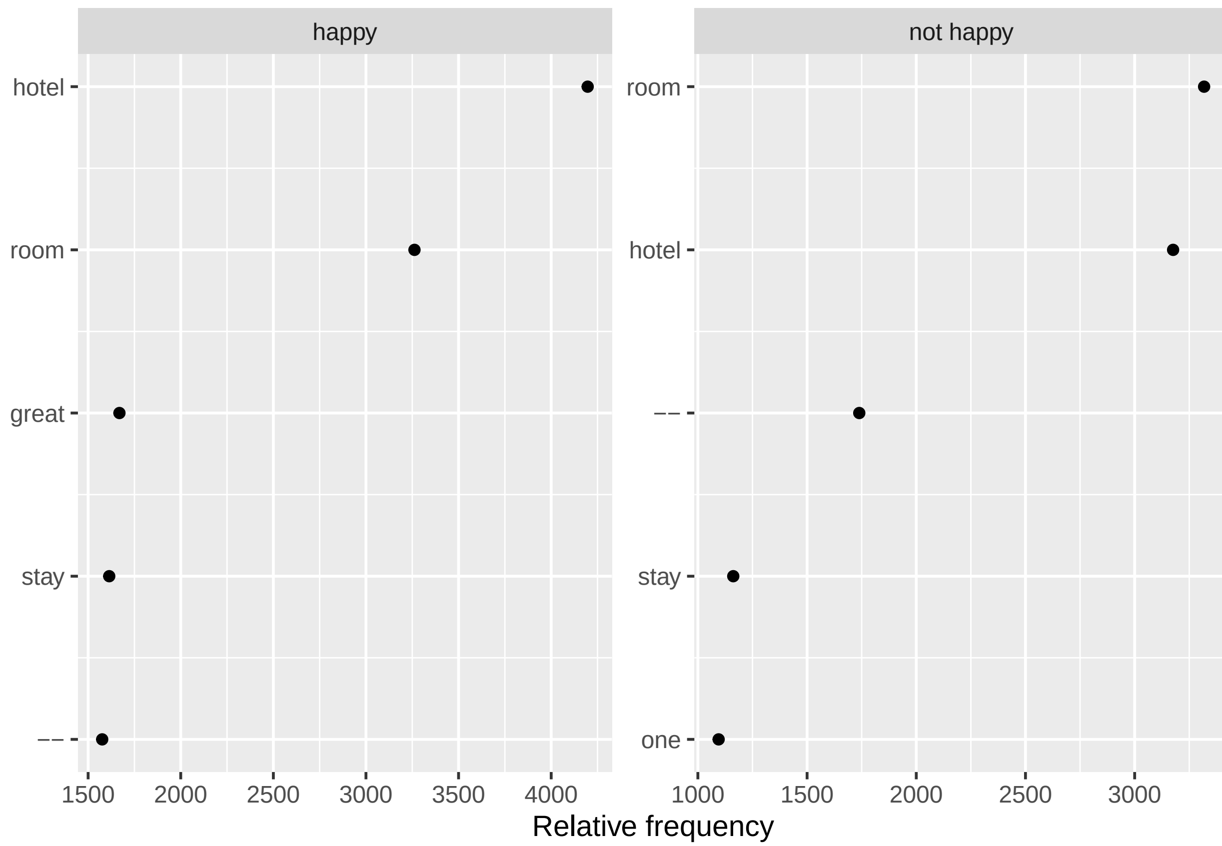
```r
freq_weight <- quanteda.textstats::textstat_frequency(dfmat2,
                                                      n = 5,
                                                      groups = dfmat2$Is_Response)
freq_weight
```

```
##    feature frequency rank docfreq     group
## 1    hotel      4197    1       1     happy
## 2     room      3262    2       1     happy
## 3    great      1669    3       1     happy
## 4     stay      1614    4       1     happy
## 5       --      1576    5       1     happy
## 6     room      3318    1       1 not happy
## 7    hotel      3176    2       1 not happy
## 8       --      1739    3       1 not happy
## 9     stay      1162    4       1 not happy
```

```
## 10      one       1095     5        1 not happy
```

GGPlot these

```r
ggplot(freq_weight, aes(nrow(freq_weight):1, frequency)) +
     geom_point() +
     facet_wrap(~ group, scales = "free") +
     coord_flip() +
     scale_x_continuous(breaks = nrow(freq_weight):1,
                        labels = freq_weight$feature) +
     labs(x = NULL, y = "Relative frequency")
```



This code is an example of how to create a scatterplot of relative frequencies for a set of features, grouped by a categorical variable, using the ggplot2 package in R.

Here's a breakdown of the code:

1. **`ggplot(freq_weight, aes(nrow(freq_weight):1, frequency))`**: This sets up the ggplot object and specifies the dataset (**`freq_weight`**) and the x and y variables for the plot. The **`nrow(freq_weight):1`** part creates a sequence of numbers from the number of rows in the dataset down to 1, which will be used as the x-axis values. The **`frequency`** variable is used as the y-axis values.
2. **`geom_point()`**: This adds a layer to the plot, which will display the data as points.

3. **`facet_wrap(~ group, scales = "free")`**: This creates a separate panel in the plot for each unique value of the **`group`** variable in the dataset, and sets the scales to be "free" (i.e., they will be scaled independently for each panel).

4. **`coord_flip()`**: This flips the x and y axes, so that the plot is displayed horizontally.

5. **`scale_x_continuous(breaks = nrow(freq_weight):1, labels = freq_weight$feature)`**: This sets the breaks and labels for the x-axis. The **`breaks`** argument specifies the location of tick marks on the x-axis (which will be the sequence of numbers created earlier), and the **`labels`** argument specifies the labels for each tick mark, which will be the **`feature`** variable in the **`freq_weight`** dataset.

6. **`labs(x = NULL, y = "Relative frequency")`**: This sets the axis labels, with **`x = NULL`** indicating that the x-axis should not be labeled, and **`y = "Relative frequency"`** indicating that the y-axis should be labeled "Relative frequency".

Overall, this code will produce a scatterplot of relative frequencies for each feature, grouped by the value of a categorical variable, with separate panels for each value of the categorical variable. The x-axis will display the feature names, and the y-axis will display the relative frequency values.

N-grams and Collocations N-grams, collocations, and keyness are all related concepts in linguistics. Collocation pertains to the occurrence of words together, such as the commonly used phrase "touch base" This phrase is an example of a collocation because the words "touch" and "base" frequently appear together, more often than by chance alone.

In contrast, n-grams represent a broader set of word combinations that occur together. For instance, bi-grams refer to two words that are frequently used together, while tri-grams refer to three words that are commonly used together, and so on.

```
hotel_ngrams<-toks_hotel %>%
  tokens_ngrams(n=2)
hotel_ngrams_dfm<-dfm(hotel_ngrams)
tstat1 <- textstat_frequency(hotel_ngrams_dfm)
head(tstat1, 10)
```

```
##              feature frequency rank docfreq group
## 1                $_--      1003    1     611   all
## 2          front_desk       882    2     681   all
## 3               $_---       393    3     285   all
## 4        room_service       335    4     267   all
## 5               --_--       324    5     234   all
## 6          --_minutes       299    6     244   all
## 7    walking_distance       287    7     269   all
## 8       across_street       284    8     256   all
## 9        times_square       261    9     183   all
## 10           new_york       256   10     169   all
```

We can get a similar output even with general collocations code

```
text_coll <- textstat_collocations(toks_hotel, size = 2, min_count = 20)
text_coll %>% arrange(across(starts_with("count"), desc)) %>% head()
```

```
##          collocation count count_nested length    lambda        z
## 1         front desk   881            0      2  7.939824 90.66670
## 30      room service   331            0      2  2.524822 40.58670
## 11  walking distance   287            0      2  9.371667 51.57749
## 3      across street   284            0      2  7.053361 65.00979
## 2       times square   259            0      2  7.045201 66.23640
## 396         new york   253            0      2 11.138878 17.48291
```

We can also run a customized ngram

```
cust_bigram<- tokens_compound(hotel_ngrams, phrase("room*"))
cust_bigram<- tokens_select(cust_bigram, phrase("room_*"))

cust_bigram<-dfm(cust_bigram)
tstat1 <- textstat_frequency(cust_bigram)
head(tstat1, 10)
```

```
##          feature frequency rank docfreq group
## 1   room_service       335    1     267   all
## 2     room_clean       182    2     181   all
## 3      room_nice       131    3     127   all
## 4     room_ready        85    4      72   all
## 5      room_--th        69    5      68   all
## 6  room_spacious        66    6      66   all
## 7      room_room        66    6      63   all
## 8     room_small        60    8      59   all
## 9     room_great        58    9      58   all
## 10    room_large        55   10      54   all
```

```
# You can visualize it as a word cloud as well
# textplot_wordcloud(cust_bigram, min_size = 0.5,   max_size = 4,
#   min_count = 3,   max_words = 200,   color = "darkblue",   font = NULL,
#   adjust = 0, rotation = 0.1, random_order = FALSE,   random_color = FALSE,
#   ordered_color = FALSE,   labelcolor = "gray20",   labelsize = 1.5,
#   labeloffset = 0,   fixed_aspect = TRUE,   comparison = FALSE)
```

We have seen that even without getting into any fancy analysis, we can get a basic understanding from exploratory data analysis.