Masters Programmes

Assignment Cover Sheet

Submitted by:	**Group 10, 1966891, 1992632, 1992799, 1966628**
Date Sent:	**26/03/2020**
Module Title:	**Text Analytics**
Module Code:	**IB9CW0**
Date/Year of Module:	**19/20**
Submission Deadline:	**26/03/2020**
Word Count:	**10675**
Number of Pages:	**72**

Question: ***[e.g. question number/title, or description of assignment]***

Downloading, ETL and Data Preprocessing

1. Packages Preparation

```r
knitr::opts_chunk$set(eval = FALSE)

rm(list = ls())
library(RSQLite)
library(XML)
library(rvest)
library(RCurl)
library(cld3)
library(tidytext)
library(textclean)
library(qdapDictionaries)
library(udpipe)
library(tm)
library(ggplot2)
library(gridExtra)
library(stm)
library(quanteda)
library(textfeatures)
library(sentimentr)
library(qdap)
library(lubridate)
library(stargazer)
library(readr)
library(stringr)
library(dplyr)
library(tidyr)
library(wordcloud)
library(data.table)
library(ggraph)
library(igraph)
library(text2vec)
library(geometry)
```

2. Downloading

The main goal of this section is to download the relevant datasets from the webpage.

Firstly, we crawl all files for each city from the webpage and save the information into a data frame in R. Then, we filter to get the useful files for this assignment and get the URLs to download them. We only want to keep the latest versions of listings and reviews files because we assume that they accumulate all the information from the beginning. Besides, we want to get average prices and occupancy rates before 2020, so we keep the calendar files that were uploaded before 2019. Finally, we have 3 loops to download listings, reviews, and calendar files respectively. Since some links on the webpage do not work, we use tryCatch function here to make sure our code can run smoothly in one go.

```r
# ----- Set the working directory
setwd("C:/Users/xu/Desktop/WBS/Term 2/Text analytics/TA_GroupAssignment")

# ----- Get info from the webpage
url_airbnb <- 'http://insideairbnb.com/get-the-data.html'
get_the_data <- read_html(url_airbnb)
tables <- get_the_data %>% html_nodes("table")

# ----- Get urls
cities_table <- data.frame()
for (i in 1:102) {
  table_h <- tables[[i]] %>% html_table()
  all_links <- tables[[i]] %>% html_nodes("a") %>% html_attr("href")
  table_h$links <- all_links
  cities_table <- plyr::rbind.fill(cities_table, table_h)
}

# ----- Normalise the column names
colnames(cities_table) <- gsub(" |/", "_", colnames(cities_table)) %>% tolowe
r()

# ----- Normalise the date format
lct <- Sys.getlocale("LC_TIME")
Sys.setlocale("LC_TIME", "C")
cities_table$date_compiled <- as.Date(cities_table$date_compiled, format = "%
d %b,%Y")

# ----- Get the latest listing file for each city
listings_data <- cities_table %>%
  group_by(country_city) %>%
  arrange(desc(date_compiled)) %>%
  filter(grepl("Detailed Listings",description)) %>%
  top_n(1)

# ----- Get the latest review files
reviews_data <- cities_table %>%
  group_by(country_city) %>%
  arrange(desc(date_compiled)) %>%
  filter(grepl("Detailed Review",description)) %>%
  top_n(1)

# ----- Get calendar files which contain information before 2020
calendar_data <- cities_table %>%
  mutate(year = lubridate::year(date_compiled)) %>%
  group_by(country_city) %>%
  arrange(desc(date_compiled)) %>%
  filter(grepl("Detailed Calendar",description),
         year <= 2018)

# ----- Set the downloading directory
datafolder <- "E:/airbnb_file"

# ----- Download listing files
for (i in 1:nrow(listings_data)) {
  tryCatch(download.file(url = listings_data$links[i], destfile = paste0(data
folder, "/listings/", tolower(listings_data$country_city[i]), "_listings.csv.
gz"), quiet = T), error = function(e) print("file did not work"))
}

# ----- Download review files
for (i in 1:nrow(reviews_data)) {
  tryCatch(download.file(url = reviews_data$links[i], destfile = paste0(dataf
older, "/reviews/", tolower(reviews_data$country_city[i]), "_reviews.csv.gz")
, quiet = T), error = function(e) print("file did not work"))
}

# ----- Download calendar files
for (i in 1:nrow(calendar_data)) {
  tryCatch(download.file(url = calendar_data$links[i],destfile = paste0(dataf
older, "/calendar/", calendar_data$date_compiled[i], "_", tolower(calendar_da
ta$country_city[i]),"_calendar.csv.gz"), quiet = T), error = function(e) prin
t("file did not work"))
}
```

3. Extracting, Transforming, and Loading data to a relational schema

The ETL flow is to get the relevant datasets into a normalized relational schema. This might be the most essential process for the whole assignment as for the famous phrase of data science, “garbage in, garbage out”, we don’t want that!

The ETL workflows are split into 3 major chunks: investigating the datasets, building a tailored fit ETL workflow, and running the workflow. Firstly, samples of listings and reviews data are taken from each country to know which columns to keep and which ones to remove. It is found that the listings dataset can be normalized into reviewer and review, which listing can be normalized into host and listing. Additionally, columns that have URLs are not needed, thus will not be considered in the data transformation.

The reviewer can write reviews for listings, which are owned by hosts and have calendars.

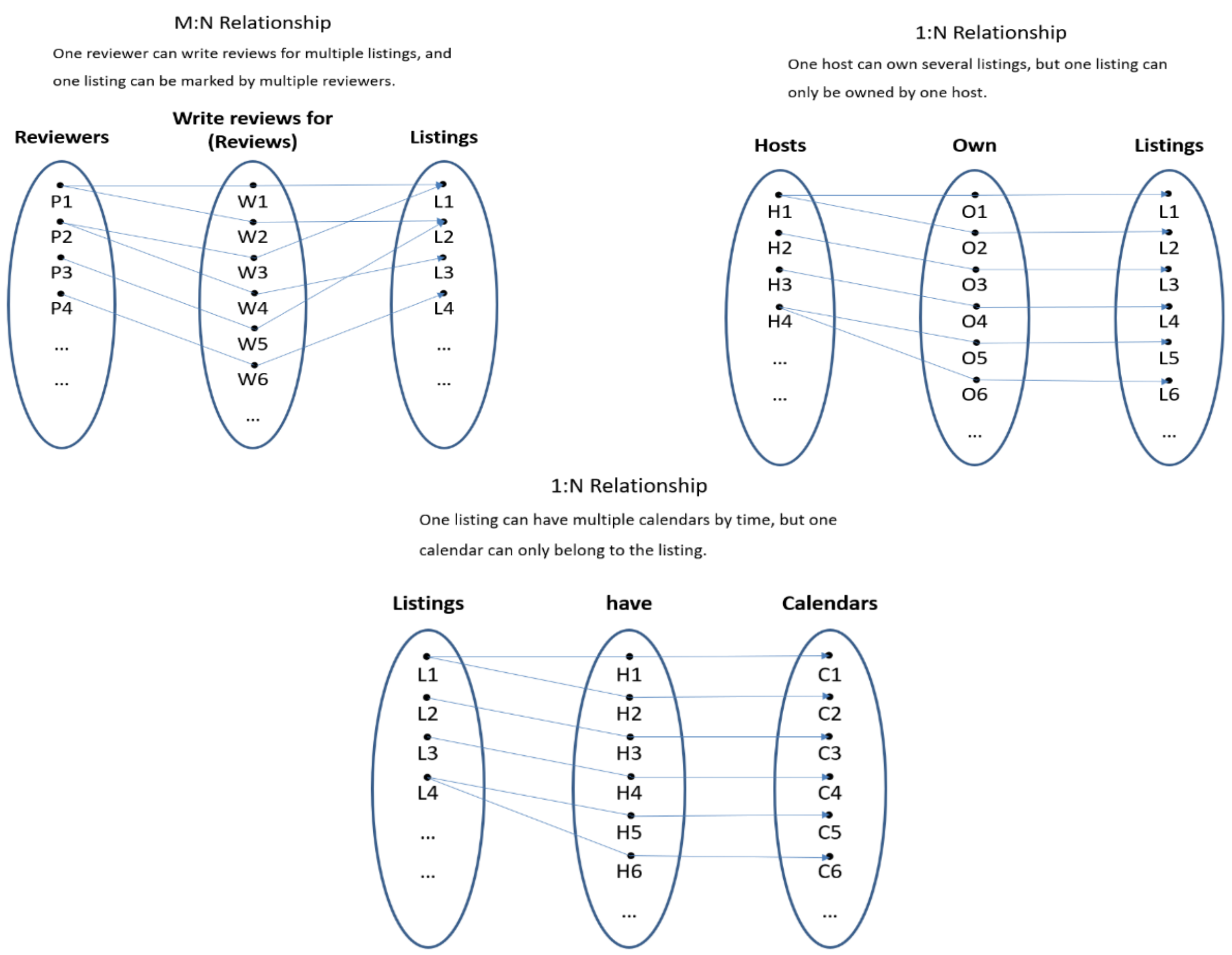

M:N Relationship
One reviewer can write reviews for multiple listings, and one listing can be marked by multiple reviewers.
Reviewers
Write reviews for (Reviews)
Listings
P1 P2 P3 P4 … …
W1 W2 W3 W4 W5 W6 …
L1 L2 L3 L4 … …
1:N Relationship
One host can own several listings, but one listing can only be owned by one host.
Hosts
Own
Listings
H1 H2 H3 H4 … …
O1 O2 O3 O4 O5 O6 …
L1 L2 L3 L4 L5 L6 …
1:N Relationship
One listing can have multiple calendars by time, but one calendar can only belong to the listing.
Listings
have
Calendars
L1 L2 L3 L4 … …
H1 H2 H3 H4 H5 H6 …
C1 C2 C3 C4 C5 C6 …

Figure 1 Relational Schemas

```r
# ----- Pre-ETL investigation to get all columns and select the columns neede
d

# Create a function to make a sample of 1 row from each dataset and combine t
hem to get an idea what columns we have
get_sample <- function(folder_path, pattern) {

  listed_files <- list.files(folder_path, pattern = pattern)
  main_df <- data.frame()

  for (i in 1:length(listed_files)) {

    file_path <-paste(folder_path, listed_files[i],sep="/")
    local_df <- read_csv(file_path, n_max = 1)
    local_df$file_name <-listed_files[i]
    local_df$pre_processed <- 0
    main_df <- plyr::rbind.fill(main_df, local_df)

  }

return(main_df)

}

# Generate the sample
listings_sample <- get_sample(folder_path = "dataset", pattern = "listings.cs
v.gz")
reviews_sample <- get_sample(folder_path = "dataset", pattern = "reviews.csv.
gz")
calendar_sample <- get_sample(folder_path = "dataset", pattern = "calendar.cs
v.gz")
reviews_sample$date <- as.Date(reviews_sample$date)
```

Building the ETL workflow requires the most effort of the ETL process as we need to balance between having too much data and too less of a data. After investigating the columns from get_sample function we defined earlier, we can already reduce the not-going to be used columns early. By filtering columns early we reduce the number of dimensions thus will reduce the amount of time later to process the datasets. Even delaying the decision to filter columns later will obviously result to more processing time.

- Thus, we build the transformation process on the following criterions: Listings and Reviews that are to be considered will be taken from the most recent datasets available in open data Airbnb site. * Listing should have at least 10 reviews
- Only consider reviews and calendar data of the selected listings * Listing and reviews which are written in English * Calendar data are taken from the archived, the most recent record of price and booking are the one to be taken

The columns which are less likely to be used such as that URLs are left out. Afterwards, the ETL work flow is built and immediately used to the sample data as it has all the columns from every country to avoid errors later since relational schema has to be predefined before data are inserted. Of course the tables are truncated to ensure clean sheet before loading the datasets (staging process).

```r
# ----- Initiation
conn <- dbConnect(RSQLite::SQLite(), "inside_airbnb.db") # connect to SQLite
db, in this case, it created a new db

# ----- Build ETL workflow for listings data

normalise_listings <- function(listings_data) {

  # Manually remove columns that we do not need
  remove_columns <- c('street', 'neighbourhood', 'latitude','longitude', 'is_
location_exact', 'square_feet', 'license', 'calculated_host_listings_count_en
tire_homes', 'calculated_host_listings_count_private_rooms', 'calculated_host
_listings_count_shared_rooms','reviews_per_month', 'last_searched', 'region_i
d', 'region_name', 'region_parent_id', 'region_parent_name', 'region_parent_p
arent_id', 'region_parent_parent_name', 'weekly_price', 'monthly_price', 'min
imum_nights', 'maximum_nights', 'minimum_minimum_nights', 'maximum_minimum_ni
ghts', 'minimum_maximum_nights', 'maximum_maximum_nights', 'has_availability'
, 'summary', 'description', 'neighborhood_overview', 'space', 'host_listings_
count', 'smart_location', 'scrape_id', 'experiences_offered', 'notes', 'acces
s', 'interaction', 'house_rules', 'jurisdiction_names', 'calendar_updated', '
last_review')

  # Listing Table
  listings_table <- listings_data %>%
    filter(number_of_reviews > 10) %>%
    unite(col=new_description,c(summary,description,neighborhood_overview, sp
ace),sep = " ", na.rm=TRUE) %>%
    mutate(lang = cld3::detect_language(new_description)) %>%
    filter(lang == 'en') %>%
    dplyr::rename(listing_id = id) %>%
    mutate(listing_id = as.character(listing_id),
           last_scraped = as.character(last_scraped),
           calendar_last_scraped = as.character(calendar_last_scraped),
           first_review = as.character(first_review),
           last_review = as.character(last_review)) %>%
    select(-c(contains("url"), host_name:host_identity_verified)) %>%
    select_if(!names(.) %in% remove_columns) %>%
    mutate(pre_processed = 0)

  # Host Table
  host_distinct <- unique(listings_table$host_id)

  hosts_table <- listings_data %>%
    mutate(host_since = as.character(host_since)) %>%
    filter(host_id %in% host_distinct) %>%
    select(starts_with('host'), -contains("url")) %>%
    distinct(host_id, .keep_all = TRUE)

  # Insert to db
  dbWriteTable(conn,"host", hosts_table, append = TRUE)
  dbWriteTable(conn,"listing", listings_table, append = TRUE)
}

# ----- Build ETL workflow for reviews data

normalise_reviews <- function(reviews_data, included_listing) {

  # Review table
  reviews_table <- reviews_data %>%
    mutate(listing_id = as.character(listing_id),
           lang = cld3::detect_language(comments),
           review_date = as.character(date)) %>%
    filter(listing_id %in% included_listing$listing_id,
           lang == 'en') %>%
    dplyr::rename(review_id = id) %>%
    mutate(review_id = as.character(review_id)) %>%
    select(-reviewer_name) %>%
    mutate(pre_processed = 0)

  # Reviewer Table
  review_distinct <- unique(reviews_table$review_id)

  reviewers_table <- reviews_data %>%
    dplyr::rename(review_id = id) %>%
    mutate(review_id = as.character(review_id)) %>%
    filter(review_id %in% review_distinct) %>%
    distinct(reviewer_id, reviewer_name)

  dbWriteTable(conn,"review", reviews_table, append = TRUE)
  dbWriteTable(conn,"reviewer", reviewers_table, append = TRUE)
}

# ----- Build ETL workflow for calendar data

normalise_calendar <- function(calendars_data, included_listing) {

   remove_columns <- c('adjusted_price', 'minimum_nights', 'maximum_nights',
'available', 'date')

  # Calendar table
  calendar_table <- calendars_data %>%
    mutate(listing_id = as.character(listing_id)) %>%
    filter(listing_id %in% included_listing$listing_id, year(date) <= 2019) %
>%
    mutate(booked = ifelse(available==FALSE, 1, 0),
           price = as.numeric(gsub(",", "", substring(price, 2))),
           bookingdate = as.character(date)) %>%
    select_if(!names(.) %in% remove_columns) %>%
    anti_join(calendar_tracker)
  dbWriteTable(conn,"calendar", calendar_table, append = TRUE)

  calendar_tracker <-
    dbGetQuery(conn,"SELECT distinct listing_id, bookingdate FROM calendar")

  assign("calendar_tracker", calendar_tracker, envir = .GlobalEnv)
}

# ----- Automatically use sample data to create schema
normalise_listings(listings_sample)
included_listing <- dbGetQuery(conn, 'SELECT listing_id FROM listing')
normalise_reviews(reviews_sample, included_listing)
normalise_calendar(calendar_sample, included_listing)

# ----- Clear tables, ready to be inserted.
dbExecute(conn, "DELETE FROM review")
dbExecute(conn, "DELETE FROM reviewer")
dbExecute(conn, "DELETE FROM listing")
dbExecute(conn, "DELETE FROM host")
dbExecute(conn, "DELETE FROM calendar")
```

Finally, the datasets are loaded by running the ETL workflow with the prebuilt functions on normalizing the datasets and the use of loops.

```
start_time <- Sys.time()

# ----- Get the list of files
listings_list <- list.files("E:/airbnb_file/listings")
reviews_list <- list.files("E:/airbnb_file/reviews")
calendar_list <- list.files("E:/airbnb_file/calendar")

# ----- Store listing data into SQL
for (i in 1:length(listings_list)) {

  file_path <-paste0("E:/airbnb_file/listings/", listings_list[i])
  listings_data <- read_csv(file_path)
  listings_data$file_name <-listings_list[i]

  normalise_listings(listings_data) # call function built especially to norma
lise listings

}

included_listing <- dbGetQuery(conn, 'SELECT listing_id FROM listing')

# ----- Store review data into SQL
for (i in 1:length(reviews_list)) {

  file_path <-paste0("E:/airbnb_file/reviews/", reviews_list[i])
  reviews_data <- read_csv(file_path)
  reviews_data$file_name <- reviews_list[i]

  normalise_reviews(reviews_data, included_listing) # call function built esp
ecially to normalise listings

}

# ----- Store calendar data into SQL
calendar_tracker <- data.frame(listing_id=character(), date=as.Date(character
())) # create empty df for function normalise_calendar

for (i in 1:length(calendar_list)) {

  file_path <- paste0("E:/airbnb_file/calendar/", calendar_list[i])
  calendars_data <- read_csv(file_path)
  calendars_data$file_name <- calendar_list[i]

  normalise_calendar(calendars_data, included_listing)
}

# ----- Check the table lists in SQLite
dbListTables(conn) # list all table names

end_time <- Sys.time()
end_time - start_time #record how long it takes
```

4. Data Pre-processing

In this session, we have two while loops to do the data pre-processing for fulfilling requests of each part in the assignment. Since we have stored all the data in SQLite, we make full use of the RSQLite package and SQL codes in this section.

In order to save time and memory space, we decided to select 3 representative cities as the main objects of our analysis: Amsterdam, Melbourne, New York City.

Firstly, some preparations that will be used in both processes can be done outside the loops, which can improve the efficiency of data cleaning. At this stage, we built a function to handle negations and set up our own stop words dictionaries and merged them with Fry_1000 and stop_words. The customized stop words include the names of the cities, the names of each landlord, the names of each neighborhood, and some high-frequency words that we don't consider meaningful. Besides, we also obtained the udpipe model that will be used later.

```
# ----- Create a function for negation
str_negate <- function(x) {
  gsub("not ","not not",gsub("n't ","n't not",x))
}
```

```r
# ----- Create customed stopwords dictionaries
data("stop_words")
data("Fry_1000")
Fry_1000 <- tibble(Fry_1000)

host_name <-
  dbGetQuery(conn, 'SELECT distinct host_name FROM host') %>%
  rename(word = host_name)

neighbourhood_cleansed <-
  dbGetQuery(conn, 'SELECT distinct neighbourhood_cleansed FROM listing WHERE
 file_name IN ("amsterdam_listings.csv.gz","melbourne_listings.csv.gz", "new
york city_listings.csv.gz")') %>%
  rename(word = neighbourhood_cleansed)

city_name <-
  dbGetQuery(conn, 'SELECT distinct city FROM listing WHERE file_name IN ("am
sterdam_listings.csv.gz","melbourne_listings.csv.gz", "new york city_listings
.csv.gz")') %>%
  rename(word = city)

customed_words <-
  tibble(c("can","good","stay","airbnb","apartment","great","everything","rea
lly", "airbnb", "bnb", "room", "house", "place", "flat", "accomodation"))

colnames(customed_words) <- "word"

# ----- Combine dictionaries into one
add_words <-
  bind_rows(customed_words, city_name, neighbourhood_cleansed, host_name) %>%
  na.omit()

# ----- Get the udpipe model
ud_model <- udpipe_download_model(language = "english", overwrite = F)
ud_model <- udpipe_load_model(ud_model$file_model)
```

Let’s deal with the listing data first.

In order to improve efficiency and save memory, we process a small part of the data at a time.

When we imported the data into SQLite, we added a column called pre_processed to both the listing table and the review table. At the beginning of the while loop, we each time select 1000 rows of data that have not been cleaned, that is, rows where pre_processed is equal to 0. When the data processing is finished, at the end of the loop, we update these observations with pre_processed equal to 1. Therefore, in the following iterations, the processed data will no longer be selected again.

At the end of each iteration, we count how many observations have not yet been processed. When the unprocessed data is gradually reduced to 0, the loop will stop running.

Since we combined the text columns about listings in the ETL stage, there are some duplicate sentences that need to be removed. Then we deal with negations and remove numbers, punctuation marks, white spaces, and capital letters from the text. Next, we remove stop words by using the dictionaries we created before and correct the misspellings by using hunspell package.

Because the process of udpipe is quite time consuming, we embed a new for loop for it to chunk the data. In addition, when performing udpipe, we found that storing the results into a list and then getting the data frame by calling rbindlist function is much faster than directly superimposing data frames.

Finally, we stored the results of udpipe and cleaned-up descriptions into new SQLite tables respectively.

```
# ----- Initialise While Loop

# calculate how many unprossed rows
# 38641 observations in total
query <- dbGetQuery(conn, 'SELECT count(listing_id) FROM listing WHERE pre_pr
ocessed = 0 AND file_name IN ("amsterdam_listings.csv.gz","melbourne_listings
.csv.gz", "new york city_listings.csv.gz")')

i = 0


# ----- Loop until all listing data are processed

while (query > 0) {

  # Select 1000 oveservations each time

  df <- dbGetQuery(conn, 'SELECT * FROM listing
                   WHERE pre_processed = 0 AND
                   file_name IN
                   ("amsterdam_listings.csv.gz","melbourne_listings.csv.gz",
"new york city_listings.csv.gz")
                   ORDER BY listing_id
                   LIMIT 1000')


# ----------------------- DATA PRE-PROCESSING  ------------------
```

```r
# Unite text columns in listing
description_cleaned <- df %>%
    select(listing_id,new_description) %>%
    unnest_tokens(sentence, new_description, token="sentences",to_lower = FAL
SE) %>%
    select(listing_id, sentence) %>%
    unique() %>%
    group_by(listing_id) %>%
    mutate(n = row_number()) %>%
    spread(n, sentence) %>%
    unite(new_description_cleaned,na.rm=TRUE,-listing_id)

df <- df %>% left_join(description_cleaned)

rm(description_cleaned)

# Clean the text
df$new_description_cleaned <-
  df$new_description_cleaned %>%
  str_negate() %>%
  removeNumbers() %>%
  removePunctuation() %>%
  replace_white() %>%
  tolower()

# Tokenize and remove stopwords
tokens_listing <-
  df %>%
  select(listing_id, new_description_cleaned) %>%
  unnest_tokens(word, new_description_cleaned) %>%
  group_by(listing_id, word) %>%
  count() %>%
  anti_join(Fry_1000, by = c("word" = "Fry_1000")) %>%
  anti_join(stop_words) %>%
  anti_join(add_words)

# Correct the mis-spellings by using the hunspell package
bad.words <- tokens_listing$word %>%
  unique() %>%
  hunspell::hunspell() %>%
  unlist() %>%
  unique()

sugg.words <- bad.words %>%
  hunspell::hunspell_suggest() %>%
  lapply(function(x) x[1]) %>%
  unlist()

word.list <- as.data.frame(cbind(bad.words, sugg.words)) %>%
  rename(word = bad.words)
```

```r
tokens_listing <- tokens_listing %>%
  left_join(word.list)

NA_index <- which(is.na(tokens_listing$sugg.words))
tokens_listing$sugg.words <- as.character(tokens_listing$sugg.words)
tokens_listing[NA_index,"sugg.words"] <- tokens_listing[NA_index,"word"]

# Chunk the data to run udpipe efficiently
split_size <- 5000
for_pos_list <- split(tokens_listing,
                      rep(1:ceiling(nrow(tokens_listing)/split_size),
                      each = split_size,
                      length.out = nrow(tokens_listing)))

annotated_description <- list()

for(k in 1:length(for_pos_list)){

    # Annotating
    this_dataframe <-
      udpipe_annotate(for_pos_list[[k]]$sugg.words
                      doc_id = for_pos_list[[k]]$listing_id,
                      object = ud_model) %>%
      as.data.frame()

    # Filter out the nouns
    this_annotated_description <- this_dataframe %>%
      filter(upos == "NOUN") %>%
      select(doc_id,lemma) %>%
      group_by(doc_id) %>%
      summarise(annotated_description = paste(lemma, collapse = " ")) %>%
      rename(listing_id = doc_id)

    # Store the data into lists we created before for loop
    annotated_description[[k]] <- this_annotated_description

    # To check progress
    print(paste(k,"out of",length(for_pos_list)))
    }

# Convert the lists to dataframes
annotated_description <- data.table::rbindlist(annotated_description)

df <- df %>%
  select(listing_id, new_description_cleaned)

# Insert into SQLite as new tables
dbWriteTable(conn, "new_description_cleaned", df, append = TRUE)
dbWriteTable(conn, "description_udpipe", annotated_description, append = TRUE
)

# ----- Prepare for the next loop
dbExecute(conn, 'UPDATE listing SET pre_processed = 1 WHERE listing_id IN
                (SELECT listing_id FROM listing
                 WHERE pre_processed = 0
                 ORDER BY listing_id
                 LIMIT 1000)') # updates already processed rows as 1

  i = i+1    # count iterations
  print(paste('Listing data chunk',i,'processed'))

  query <- dbGetQuery(conn, 'SELECT count(listing_id) FROM listing WHERE pre_
processed = 0') # recalculate how many unprocessed rows left
}
```

Next, let’s deal with reviews data.

The iteration method for reviews data is the same as the one for listings, but there are some changes in the data cleaning measures.

We firstly removed comments that are shorter than 144 characters and longer than 1000 characters. Because we want to extract features about syntactical marks and all-uppercase words later in partB, we got a fully-cleaned comments column and a semi-cleaned one in the reviews_cleaned table.

In the process of udpipe, two different lists were created to fulfill the different requests of partB and partC. Two tables were newly inserted into SQLite, one is for the output of the udpipe, the other one is for the cleaned up comments.

```
# ----- Initialise While Loop

# 1949150 observations in total
query <- dbGetQuery(conn, 'SELECT count(review_id) FROM review
                    WHERE pre_processed = 0 AND
                    file_name IN ("amsterdam_reviews.csv.gz","melbourne_revie
ws.csv.gz", "new york city_reviews.csv.gz")')

i = 0

# ----- Loop until all reviews data in are processed
while (query > 0) {

  # Select 50000 oveservations each time

  df <- dbGetQuery(conn, 'SELECT * FROM review
                   WHERE pre_processed = 0 AND
                   file_name IN ("amsterdam_reviews.csv.gz","melbourne_review
s.csv.gz", "new york city_reviews.csv.gz")
                   ORDER BY review_id
                   limit 50000')


# ---------------------- DATA PRE-PROCESSING ---------------------

  # Set the boundary for comments length
  review_cleaned <- df %>%
    mutate(comments_cleaned = comments,
           comments_semi_cleaned = comments,
           comments_length = nchar(comments_cleaned)) %>%
    select(review_id, comments_cleaned, comments_semi_cleaned, comments_lengt
h) %>%
    filter(comments_length > 144 & comments_length < 1000) %>%
    select(-comments_length)

  # Clean the text completely
  review_cleaned$comments_cleaned <-
    review_cleaned$comments_cleaned %>%
    str_negate() %>%
    removeNumbers() %>%
    removePunctuation() %>%
    replace_white() %>%
    tolower()

  # Clean the text but not remove punctuation marks and capital letters
  review_cleaned$comments_semi_cleaned <-
    review_cleaned$comments_semi_cleaned %>%
    str_negate() %>%
    removeNumbers() %>%
    replace_white()

  df <- df %>%
    left_join(review_cleaned) %>%
    na.omit()

  rm(review_cleaned)

  # Tokenize and remove stopwords
  tokens_review <- df %>%
    select(listing_id, review_id, comments_cleaned) %>%
    unnest_tokens(word, comments_cleaned) %>%
    anti_join(stop_words) %>%
    anti_join(Fry_1000, by = c("word" = "Fry_1000")) %>%
    anti_join(add_words)

  # Correct the mis-spellings by using the hunspell package
  bad.words <- tokens_review$word %>%
  unique() %>%
  hunspell::hunspell() %>%
  unlist() %>%
  unique()

  sugg.words <- bad.words %>%
  hunspell::hunspell_suggest() %>%
  lapply(function(x) x[1]) %>%
  unlist()

  word.list <- as.data.frame(cbind(bad.words, sugg.words)) %>%
  rename(word = bad.words)

  tokens_review <- tokens_review %>%
  left_join(word.list)

  NA_index <- which(is.na(tokens_review$sugg.words))
  tokens_review$sugg.words <- as.character(tokens_review$sugg.words)
  tokens_review[NA_index,"sugg.words"] <- tokens_review[NA_index,"word"]

  # Chunk the data to run udpipe efficiently
  split_size <- 5000
  for_pos_list <- split(tokens_review,
                        rep(1:ceiling(nrow(tokens_review)/split_size),
                        each = split_size,
                        length.out = nrow(tokens_review)))

  annotated_reviews_partb <- list()
  annotated_reviews_partc <- list()

  for(k in 1:length(for_pos_list)){

    # Annotating
    this_dataframe <-
      udpipe_annotate(for_pos_list[[k]]$word,
                      doc_id = for_pos_list[[k]]$review_id,
                      object = ud_model) %>%
      as.data.frame()

    # Write the udpipe results into SQLite as a new table
    dbWriteTable(conn,"review_udipipe_info", this_dataframe, append = TRUE)

    # Fulfill the requests of part B
    this_annotated_reviews_partb <- this_dataframe %>%
      filter(upos %in% c("ADV","ADJ","NOUN", "AUX", "PART")) %>%
      select(doc_id,lemma) %>%
      group_by(doc_id) %>%
      summarise(annotated_comments_partb = paste(lemma, collapse = " ")) %>%
      rename(review_id = doc_id)
    # Fulfill the requests of part C
    this_annotated_reviews_partc <- this_dataframe %>%
      filter(upos == "NOUN") %>%
      select(doc_id,lemma) %>%
      group_by(doc_id) %>%
      summarise(annotated_comments_partc = paste(lemma, collapse = " ")) %>%
      rename(review_id = doc_id)

    # Store the data into lists we created before for loop
    annotated_reviews_partb[[k]] <- this_annotated_reviews_partb
    annotated_reviews_partc[[k]] <- this_annotated_reviews_partc

    # To check progress
    print(paste(k,"out of",length(for_pos_list)))

    rm(this_annotated_reviews_partb, this_annotated_reviews_partc, this_udipi
pe_info)
    }

  # Convert the lists to dataframes
  annotated_reviews_partb <- data.table::rbindlist(annotated_reviews_partb)
  annotated_reviews_partc <- data.table::rbindlist(annotated_reviews_partc)

  df <- df %>%
    left_join(annotated_reviews_partb) %>%
    left_join(annotated_reviews_partc) %>%
    select(review_id, comments_cleaned, comments_semi_cleaned, annotated_comm
ents_partb, annotated_comments_partc)

  rm(annotated_reviews_partb, annotated_reviews_partc, tokens_review)

  # Write the cleased comments into SQLite as a new table
  dbWriteTable(conn, "comments_cleaned", df, append = TRUE)


  # ----- Prepare for the next loop
  dbExecute(conn, 'UPDATE review SET pre_processed = 1 WHERE review_id IN
                      (SELECT review_id FROM review
                      WHERE pre_processed = 0
                       ORDER BY review_id
                      LIMIT 50000)') # updates already processed rows as 1

  i = i+1 # count iterations
  print(paste('Review data chunk',i,'processed'))

  query <- dbGetQuery(conn, 'SELECT count(review_id) FROM review WHERE pre_pr
ocessed = 0') # recalculate how many unprocessed rows left

}
```

```r
# close the connection with SQLite
dbDisconnect(conn)
```

Part A Construction of Corpus – Airbnb reviews

1. Data preparation

Because the database is prepared in data pre-processing stage. The data of analysis can be obtained from the SQLite database by using “RSQLite” package. By extracting and combining the data, the text mining part can be started.

```
# ----- Connect the database
con <- dbConnect(RSQLite::SQLite(), "inside_airbnb.db")
dbListTables(con)

# ----- Select listing and review table about amsterdam from the sqlite
listings_df <- dbGetQuery(con, "SELECT * FROM listing WHERE file_name =
'amsterdam_listings.csv.gz'")
reviews_df <- dbGetQuery(con, "SELECT * FROM review WHERE file_name =
'amsterdam_reviews.csv.gz'")

# ----- Get the tokens from the sqlite database.
tokens_all <- dbGetQuery(con, "select listing.listing_id,lemma from review_udipipe_info
        left join review on review_udipipe_info.doc_id = review.review_id
        left join listing on review.listing_id = listing.listing_id where review.file_name =
'amsterdam_reviews.csv.gz' ")
host_name_list <- dbGetQuery(con,"select host_id,host_name from host")

# ----- Rename the tokens in order to combine the listing and reviews
tokens_all <- tokens_all %>% rename(word = lemma) %>%
group_by(word,listing_id)%>%count()
# ----- combine two tables
all_data_df <- listings_df %>%
  left_join(reviews_df,by="listing_id")

# ----- get the new_description
new_description <- dbGetQuery(con, "select listing_id,new_description_cleaned from
new_description_cleaned")
# ----- clean the environment
rm(reviews_df)
rm(listings_df)
dbDisconnect(con)
rm(con)
```

```
# ----- Add more stop words to the dictionary we built
customed_words <-
  tibble(c("nice", "easy", "not", "highly", "with", "to"))
```

```
colnames(customed_words) <- "word"

# ----- Combine dictionaries into one
add_words <-
  bind_rows(customed_words, add_words) %>%
  na.omit()

# ----- Remove the stop_words from the tokens
tokens_all <- tokens_all %>% anti_join(add_words)

all_data_df <- all_data_df %>%  left_join(host_name_list)

all_data_df <- all_data_df %>% left_join(unique(new_description))

# ----- Get the word frequency
tokens_all_listings <- all_data_df %>%

select(listing_id,new_description_cleaned) %>%

unique() %>%

unnest_tokens(word,new_description_cleaned) %>%

anti_join(add_words) %>%

group_by(listing_id,word)%>%

count()
```

2. Analyze the dominant word in comments per aggregation category

In the travelling accommodation industry, location is very important. Customers always choose the suitable location to satisfy their demands. Besides, property type is always took into consideration by travelers. Different properties have different target people, so they also represent different latent demands. So, there must be word ranking difference between each category. To analyze the different dominant words in each category, tokens will be allocated into each category and then the frequency will be calculated. To show the difference of the dominant words, bar chart, word cloud and ranking chart are available.

```
# ----- For neighbourhood category
# ----- Get the top 5 neighbourhood
top_5 <- all_data_df %>%
  select(listing_id,neighbourhood_cleansed) %>%
  unique(.) %>%
  group_by(neighbourhood_cleansed) %>%
```

```r
  summarise(total =n()) %>%
  arrange(desc(total)) %>%
  top_n(5)

# ----- Calculate the averge rating for top5 neighbourhood
avg_rating_neigh <- all_data_df %>%
  select(listing_id,neighbourhood_cleansed,review_scores_rating) %>%
  unique(.) %>%
  group_by(neighbourhood_cleansed) %>%
  summarise(avg.rating=mean(review_scores_rating)) %>%
  filter(neighbourhood_cleansed %in% top_5$neighbourhood_cleansed)

# ----- get the tokens in top5 neighbourhood
 tokens_top_5 <- all_data_df %>%
  select(listing_id,neighbourhood_cleansed) %>%
  filter(neighbourhood_cleansed %in% top_5$neighbourhood_cleansed) %>%
  unique(.)

# ----- Get the frequency of the word and sort them
neighbourhood_tokens <- neighbourhood_tokens <- tokens_all %>%
  right_join(tokens_top_5) %>%  group_by(neighbourhood_cleansed,word) %>% # word is b
etter
  summarise(total=sum(n)) %>%
  arrange(desc(total))

# ----- Now loop through and get the top 10 tokens
# ----- per neighbourhood for the review
for(neighb in 1:nrow(top_5)){
  print(paste0("For neighbourhood: ",top_5$neighbourhood_cleansed[neighb]))
  toprint <- neighbourhood_tokens %>% ungroup() %>% filter(neighbourhood_cleansed == t
op_5$neighbourhood_cleansed[neighb]) %>% top_n(30,total) %>% mutate(rank = row_num
ber())
  print(toprint)
}

# ----- Create a empty list to store the top words
top10_word_neigh <- list()

# ----- Word cloud

for(neighb in 1:nrow(top_5)){
  print(paste0("For neighbourhood: ",top_5$neighbourhood_cleansed[neighb]))

# ----- Rank the dominant words
  toprint <- neighbourhood_tokens %>% ungroup() %>% filter(neighbourhood_cleansed == t
op_5$neighbourhood_cleansed[neighb]) %>% top_n(10,total) %>% mutate(rank = row_num
ber())
  print(toprint)

# ----- Store the dominant words in the list.
 top10_word_neigh[[neighb]] <- toprint

# ----- wordcloud
  wordcloud(words = toprint$word, freq = toprint$total, min.freq = 1, random.order=FALSE, rot
.per=0.35,colors=brewer.pal(8, "Dark2"))
# ----- Bar plot
  top10bar <-toprint %>% ggplot(aes(x=fct_reorder(word,total),y=total)) + geom_bar(stat="id
entity",fill="lightblue")+coord_flip() +theme_bw() +
    theme(panel.border = element_blank(),panel.grid.major = element_blank(), panel.grid.mino
r = element_blank(),axis.line = element_line(colour = "black"),axis.title.x=element_text(size=
14,face="bold",hjust=0.5),axis.title.y =element_text(size=14,face="bold",hjust=0.5)) + labs(x="
Total", y="Dominant Words", title = paste0("For neighbourhood: ",top_5$neighbourhood_clean
sed[neighb]))
  print(top10bar)
}

# Transforming the list into df. The rindlist is much faster than bind_rouw
top10_word_neigh <- rbindlist(top10_word_neigh) %>% arrange(rank,neighbourhood_cleans
ed)

top10_word_neigh <- top10_word_neigh %>% right_join(avg_rating_neigh)

top10_word_neigh$neighbourhood_cleansed <- as.factor(top10_word_neigh$neighbourhood_cl
eansed)

# ----- Create the ranking chart
ggplot(data = top10_word_neigh, aes(x = fct_reorder(word,rank), y = rank, group =neighbourh
ood_cleansed )) +
  geom_line(aes(color = neighbourhood_cleansed, alpha = 1), size = 2) +
  geom_point(aes(color = neighbourhood_cleansed, alpha = 1), size = 4)+
  scale_y_reverse(breaks = 1:10) +
  labs(x = "Dominant Words",y = "Ranking",title = "Top10 Dominant Words Ranking Chart")+
  theme_set(theme_bw()) +
  theme(panel.grid.major = element_blank(),axis.text.x = element_text(angle = 30, hjust = 1, vj
ust = 1))

# ----- Property_type
# ----- Get the top 5 property_type
top_5 <- all_data_df %>%
  select(listing_id,property_type) %>%
  unique(.) %>%
  group_by(property_type) %>%
  summarise(total =n()) %>%
  arrange(desc(total)) %>%
  top_n(5)

# ----- Get tokens in top5 property_type
tokens_top_5 <- all_data_df %>%
  select(listing_id,property_type) %>%
  filter(property_type %in% top_5$property_type) %>%
  unique(.)
# -----
property_tokens <- tokens_all %>%
  right_join(tokens_top_5) %>%  group_by(property_type,word) %>%
  summarise(total=sum(n)) %>%
  arrange(desc(total))

# Now loop through and get the top 20 tokens
# per neighbourhood for the review
top10_word_property <- list()

for(i in 1:nrow(top_5)){
  print(paste0("For property_type: ",top_5$property_type[i]))

  toprint <- property_tokens %>% ungroup() %>% filter(property_type == top_5$property_ty
pe[i]) %>% top_n(10,total)  %>% mutate(rank = row_number())
  top10_word_property[[i]] <- toprint
  print(toprint)

# ----- Wordcloud
  wordcloud(words = toprint$word, freq = toprint$total, min.freq = 1, random.order=FALSE, rot
.per=0.35,colors=brewer.pal(8, "Dark2"))
# ----- Bar plot
  top10bar <-toprint %>% ggplot(aes(x=fct_reorder(word,total),y=total)) + geom_bar(stat="id
entity",fill="lightblue")+coord_flip() +theme_bw() +
    theme(panel.border = element_blank(),panel.grid.major = element_blank(), panel.grid.mino
r = element_blank(),axis.line = element_line(colour = "black"),axis.title.x=element_text(size=
14,face="bold",hjust=0.5),axis.title.y =element_text(size=14,face="bold",hjust=0.5)) + labs(x="
Total", y="Dominant Words", title = paste0("For Property Type: ",top_5$property_type[i]))
  print(top10bar)
}

top10_word_property <- rbindlist(top10_word_property)

ggplot(data = top10_word_property, aes(x = fct_reorder(word,rank), y = rank, group = property
_type )) +
  geom_line(aes(color = property_type, alpha = 1), size = 2) +
  geom_point(aes(color = property_type, alpha = 1), size = 4)+
  scale_y_reverse(breaks = 1:10) +
  labs(x = "Dominant Words",y = "Ranking",title = "Top10 Dominant Words Ranking Chart")+
  theme_set(theme_bw()) +
  theme(panel.grid.major = element_blank(),axis.text.x = element_text(angle = 30, hjust = 1, vj
ust = 1))
```

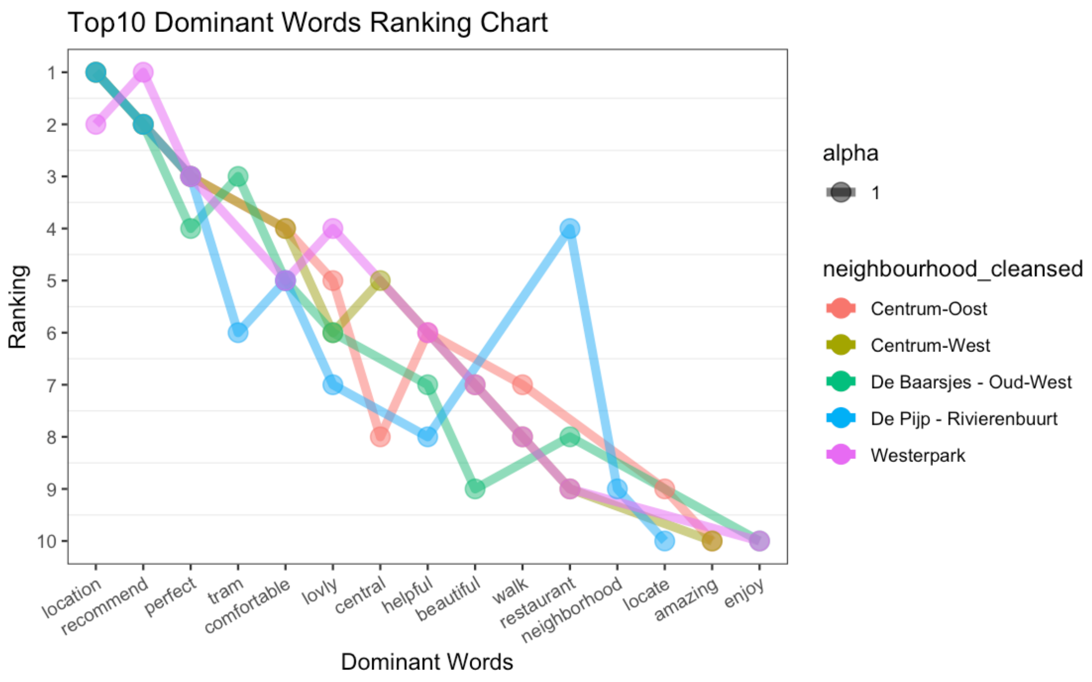


Figure 2 Top 10 Dominant Words Ranking Chart per Neighbourhood (Amsterdam)

It can be seen the top10 dominant words ranking chart of Amsterdam, these top 5 hottest neighborhoods show the similar feature. Location and tram are the most important word, and many words are used to describe the quality of the room. That means major customer pay a lot of attention to the location and the tram system. And the "recommend" represents these neighborhoods are popular and needs of customers are satisfied well. There also are some words in top 10 like "walk", "locate", "neighborhood" showing the importance of the transportation and location. Taking the "central" and "restaurant" into account, it can be seen that the convenience and quality of the room are most important factors in operation.

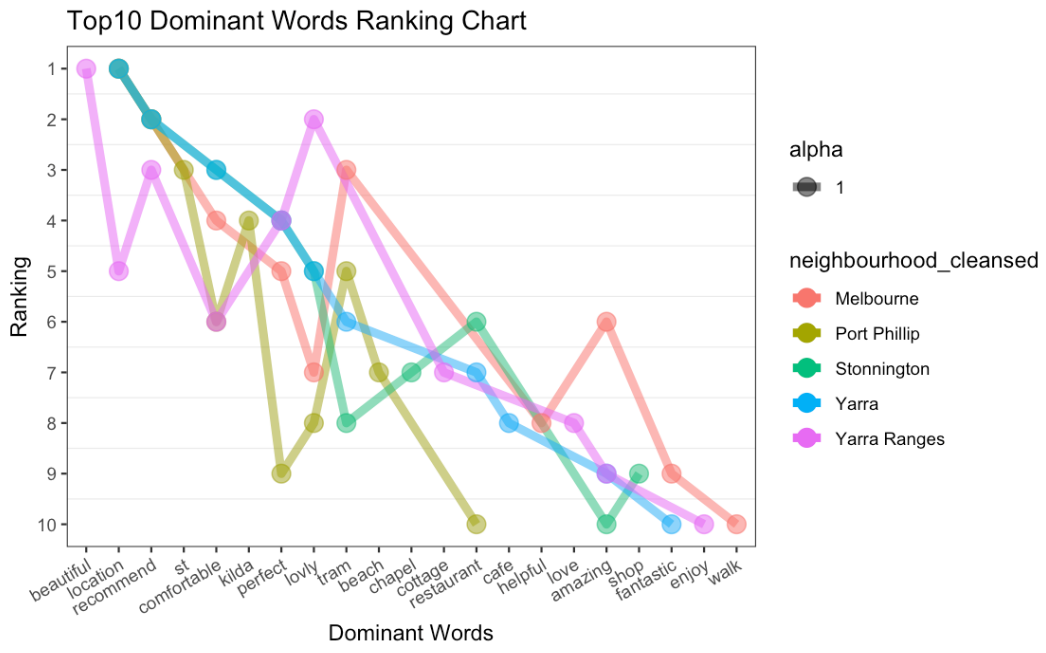


Figure 3 Top 10 Dominant Words Ranking chart per Neighbourhood (Melbourne)

In the second city (Melbourne), the dominant words are quite different from those in Amsterdam. “Café”, “beach”, “restaurant”, “shop”, “chapel” are included in the dominant words, which means there are many tourism themes in Melbourne. And the word “tram” is missing here, indicating the main transportations are different in two cities. In Amsterdam, taking a tram is a pretty convenient transportation method, but in Melbourne, maybe the private vehicle is more popular and convenient. In general, Melbourne is more interesting than Amsterdam because there are more themes.

For the difference between the top5 property type. The ranking chart is on the below.

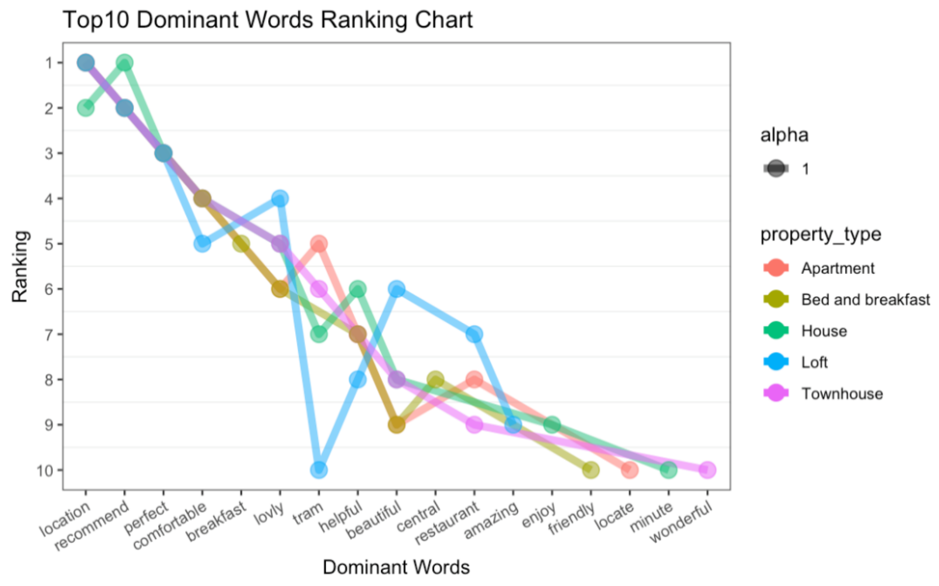


Figure 4 Top 10 Dominant Words Ranking chart per Property Type (Amsterdam)

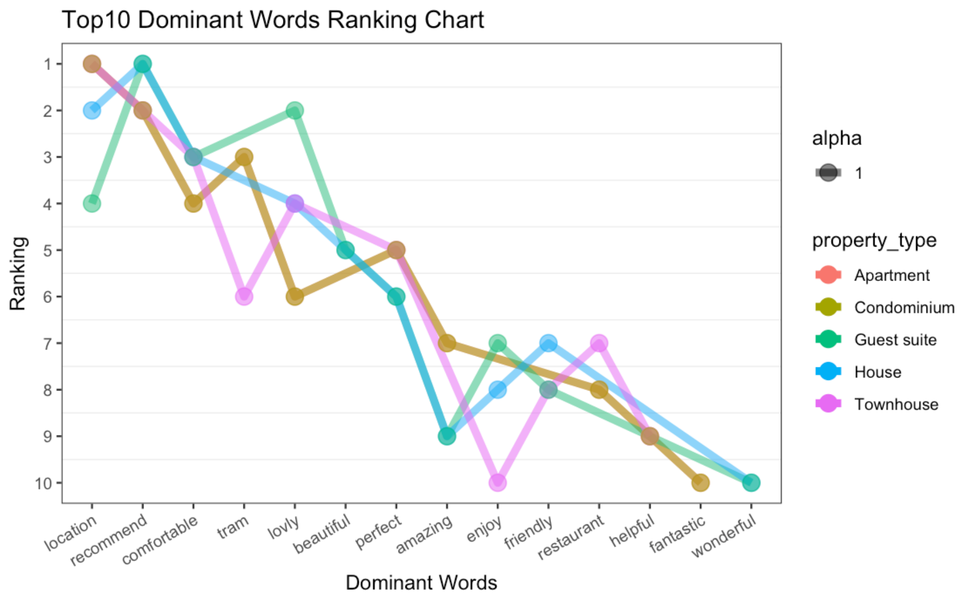


Figure 5 Top 10 Dominant Words Ranking chart per Property Type (Melbourne)

From two graphs above (Figure 4 and Figure 5), we can find that in two different cities, the major concerns of customers are similar. The interesting words are “breakfast” and “minute” in Amsterdam, which might indicate that people in Amsterdam enjoy the breakfast more and the size of Amsterdam may be smaller than Melbourne.

In conclude, these dominant words can enable people to get more information about these cities before they go there in person.

3. Analyze the common word combinations in comments

Word combinations can be interpreted as the n-grams or word association. In big data world, efficiency is important. Due to the tokens in the database, word association could be a very high-efficient method to find the common word combinations. Like a network, word association can help company understand their customers more and expand their business insight. At the beginning of this part, the top 10 dominant words in comments are found. Then document-term matrix or term-document matrix is necessary to calculate the correlation between these dominant words and other words. By ranking the correlation, the common word combination can be found. This association will be shown in the word network.

```
# ----- Take the tokens and cast them
# ----- to a dtm using the tm package
dominantwords <- tokens_all %>% group_by(word)%>% summarise(n = sum(n)) %>% arr
ange(desc(n)) %>% head(10) %>% pull(word) %>% as.character()

# ----- create the dtm of reviews
listings_dtm <- tokens_all %>%
  cast_dtm(listing_id,word,n)
dim(listings_dtm)

# ----- remove some sparse terms to reduce the dimension
listings_dtm <- removeSparseTerms(listings_dtm, 0.8)
dim(listings_dtm)
inspect(listings_dtm)

# ----- Get the correlation between terms
assocs_words <- findAssocs(listings_dtm,dominantwords,corlimit = 0.8)
assocs_words <- as.data.frame(unlist(assocs_words))
colnames(assocs_words)[1] <- "correlation"
assocs_words$combinations <- rownames(assocs_words)
assocs_words %>%
  filter(correlation > .15) %>%
  graph_from_data_frame() %>%
  ggraph(layout = "fr") +
    geom_edge_link() +
  geom_node_point(color = "lightblue", size = 5) +
  geom_node_text(aes(label = name), repel = TRUE) +
  theme_void() + labs(title = "Word Combinations")
```

Amsterdam

Melbourne

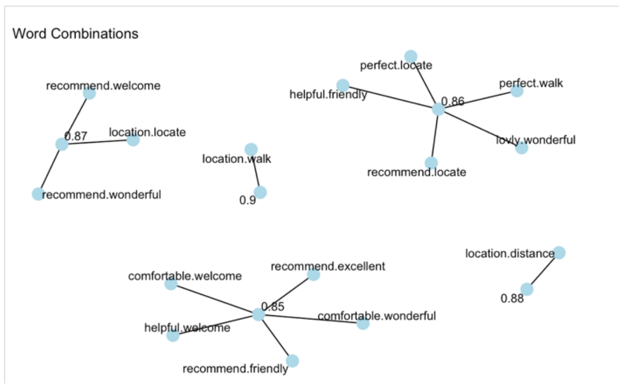


Figure 6 Word Networks in Amsterdam

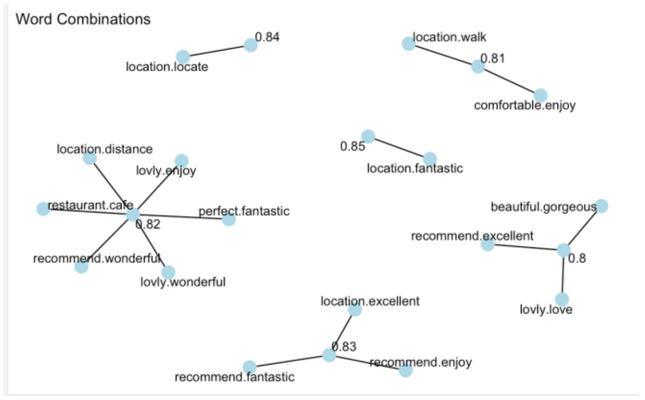


Figure 7 Word Networks in Melbourne

From the word networks (Figure 6 and Figure 7), we can see that the location has a great impact on the positive words. The most common word combinations are “location walk” in Amsterdam and “location fantastic” in Melbourne respectively. Many common combinations are Synonyms and are positive sentiment. Many word combinations are very similar. Therefore, There is no big difference between two cities.

4. Extract variables from text that can be related with the rating score

4.1 Is readability of the property description an important predictor of the satisfaction ratings?

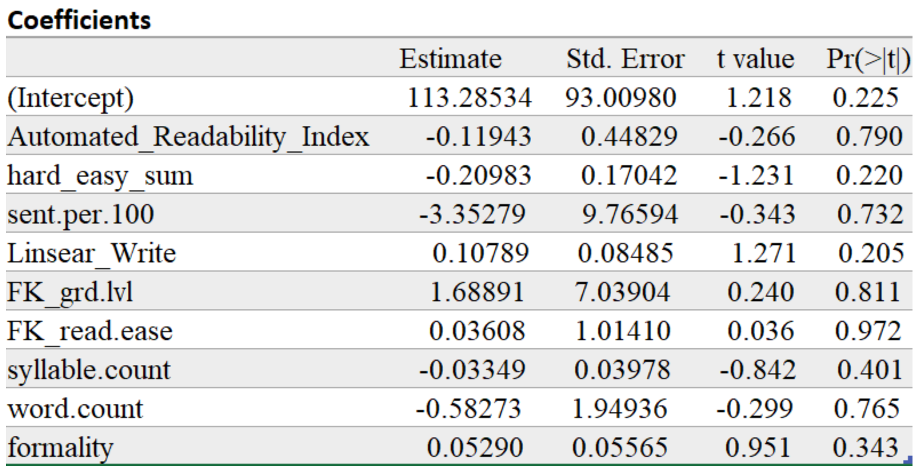
For text mining, many variables can be extracted from text, and “textfeature” could extract the features from the text content directly. But it depends on different research objectives. This report aims at finding the significant factors that will influence the rating score. Basically, Tf-idf is a common variable in the text mining. Besides, readability and formality are also the two main properties of the text. There are many readability tests in the world such as ARI… The null hypothesis here could be that complex and normal descriptions may unsatisfied the guests, which may decrease the rating score. For this question, the linear regression can show the relationships between the price and other variables from the text. Through the regression model, although some readabilities are significant, but the model only have very small Rsquare and the coefficient of the readability is small too. So, in conclusion, the readability is a not good predictor. And this report shows that mentioning name of host will influence the average rating.

```r
# ----- Tf-idf
tokens_all_tf_idf <- tokens_all %>%
  bind_tf_idf(word,listing_id,n)

# ----- lets try to filter important words
# ----- using the left and right trim
hist(tokens_all_tf_idf$tf_idf,breaks = 200,main="TF-IDF plot")

tokens_all_tf_idf <- tokens_all_tf_idf %>%
  filter(tf_idf<0.2)

hist(tokens_all_tf_idf$tf_idf,breaks = 200,main="TF-IDF plot")

# ----- Ok from the plot we see that the cut-off value is at 0.05
tokens_all_tf_idf <- tokens_all_tf_idf %>%
  filter(tf_idf<0.05)

hist(tokens_all_tf_idf$tf_idf,breaks = 200,main="TF-IDF plot")

# ----- Lets now remove also very common terms
# ----- Those with tf-idf <0.001 as shown in the chart below

tokens_all_tf_idf <- tokens_all_tf_idf %>%
  filter(tf_idf>0.001)

# ----- Calculate the average rating.
rating_categories <- all_data_df %>%
  group_by(listing_id) %>%
  summarise(avg_rating = mean(review_scores_rating)) %>%
  ungroup()

# ----- Lets find the levels that we want to aggregate the words
quantile(rating_categories$avg_rating,na.rm = TRUE)

# ----- Now assign them in a rating group
rating_categories$rating_category <- ifelse(rating_categories$avg_rating<98,1,2)

# ----- extract the feature from reviews
ratings_categories_tokens <- tokens_all %>% left_join(rating_categories) %>%
  group_by(rating_category,word) %>% summarise(total =sum(n))
```

```
ratings_categories_tokens %>% filter(rating_category==1) %>% arrange(desc(total)) %>% t
op_n(10)

ratings_categories_tokens %>% filter(rating_category==2) %>% arrange(desc(total)) %>% t
op_n(10)

# From the listings
# ------ a. Is readability of the property description an important predictor of the satisfaction rati
ngs?
all_listings <- all_data_df %>% select(listing_id,new_description,review_scores_rating) %>%
unique()

# Create the dataframe for storing the readability.
readability_fle_all <- data.frame()
readability_ari_all <- data.frame()
readability_lin_all <- data.frame()
readability_for_all <- data.frame()

# ----- Calculate the readability and fomality
for(i in 1:3000){
  readability_fle <- data.frame()
  readability_ari <- data.frame()
  readability_lin <- data.frame()
  readability_for <- data.frame()
  readability_all <- data.frame()

  this_text <- iconv(all_listings$new_description[i])
  this_text <- removeNumbers(this_text)
  this_text <- removePunctuation(this_text)

  tryCatch(readability_fle <- flesch_kincaid(this_text),error=function(e){
    cat("Error parsing")
  })

  tryCatch(readability_ari <- automated_readability_index(this_text),error=function(e){
    cat("Error parsing")
  })

  tryCatch(readability_lin <- linsear_write(this_text),error=function(e){
    cat("Error parsing")
  })

  tryCatch(readability_for <- formality(this_text),error=function(e){
    cat("Error parsing")
  })
  if(!is.null(readability_fle$Readability)){

    readability_fle <- readability_fle$Readability
    readability_fle$listing_id <- all_listings$listing_id[i]
    readability_fle_all <- bind_rows(readability_fle_all,readability_fle)
  }

  if(!is.null(readability_ari$Readability)){

    readability_ari <- readability_ari$Readability
    readability_ari$listing_id <- all_listings$listing_id[i]
    readability_ari_all <- bind_rows(readability_ari_all,readability_ari)
  }
  if(!is.null(readability_lin$Readability)){

    readability_lin <- readability_lin$Readability
    readability_lin$listing_id <- all_listings$listing_id[i]
    readability_lin_all <- bind_rows(readability_lin_all,readability_lin)
  }

  if(!is.null(readability_for$formality)){

    readability_for <- readability_for$formality
    readability_for$listing_id <- all_listings$listing_id[i]
    readability_for_all <- bind_rows(readability_for_all,readability_for)
   }
 print(i)
}

# ----- Clean the environment
rm(readability_fle)
rm(readability_ari)
rm(readability_lin)
rm(readability_for)
rm(listings_df)
# ----- Choose the readability index from each df
readability_fle_all <- readability_fle_all %>%
  select(listing_id,FK_grd.lvl,FK_read.ease ,syllable.count,word.count)

readability_ari_all <- readability_ari_all %>%
  select(listing_id,Automated_Readability_Index)

readability_lin_all <- readability_lin_all %>%
  select(listing_id,hard_easy_sum,sent.per.100,Linsear_Write)
```

```r
readability_for_all <- readability_for_all %>%
  select(listing_id,formality)

#-----merge multiple dataframe
readability_all <- Reduce(function(x, y) merge(x, y, all=TRUE), list(readability_ari_all,readabi
lity_lin_all,readability_fle_all,readability_for_all))
#-----clean the memory
rm(readability_ari_all)
rm(readability_for_all)
rm(readability_fle_all)
rm(readability_lin_all)
# ----- Replace the missing value with 0
readability_all[is.na(readability_all)] <- 0

# ----- Add rating in the df in order to run regression model
readability_rating <- all_data_df %>%
  select(listing_id,review_scores_rating) %>%
  unique() %>%
  right_join(readability_all) %>%
  select(-listing_id)

# ----- Build the lm model(using strpwise methond to select features)
model_readabiity <- lm(review_scores_rating~.,data = readability_rating)
summary(model_readabiity)
step(model_readabiity)


# ----- Using the textfeatures package get the feature from the description
feature_description <- all_data_df %>%
  select(listing_id,new_description_cleaned,review_scores_rating) %>%
  unique() %>%
  rename(text = new_description_cleaned) %>%
  textfeatures(sentiment = FALSE,word_dims=0)

# ----- Build the lm model(using strpwise methond to select features)
model_readabiity <- lm(review_scores_rating~.,data = readability_rating)
summary(model_readabiity)
step(model_readabiity)
# ----- b. Is mentioning the name of the owner important?

# ----- Choose the column we need from the all_data_df
hostname_reviews_df <- all_data_df %>% select(listing_id,review_id,review_scores_rating,hos
t_name,comments)

# ----- Because the comments
# ----- Romve punctuation in the hostname
hostname_reviews_df$host_name <- gsub('[[:punct:]]+',' ', hostname_reviews_df$host_name)
```

```r
# ----- Remove extra white space
hostname_reviews_df$host_name <- gsub("\\s+"," ",hostname_reviews_df$host_name)

# ----- For this we are going to add a new column
 hostname_reviews_df$host_name_mentioned <-NA


# fuzzy match is more suitable in the question
 for(i in 1:nrow(all_data_df)){
   check_h <- as.numeric(agrepl(hostname_reviews_df$host_name[i],
                               hostname_reviews_df$comments[i],
                               ignore.case = T,max.distance = 0.1))
   hostname_reviews_df$host_name_mentioned[i] <- check_h
 }

# How does it look graphically
ggplot(subset(hostname_reviews_df,!is.na(host_name_mentioned)),aes(x=factor(host_name_m
entioned),y=review_scores_rating))+geom_boxplot()
# ----- run the T-test
t.test(hostname_reviews_df$review_scores_rating~factor(hostname_reviews_df$host_name_me
ntioned)
```



Coefficients

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	113.28534	93.00980	1.218	0.225
Automated_Readability_Index	-0.11943	0.44829	-0.266	0.790
hard_easy_sum	-0.20983	0.17042	-1.231	0.220
sent.per.100	-3.35279	9.76594	-0.343	0.732
Linsear_Write	0.10789	0.08485	1.271	0.205
FK_grd.lvl	1.68891	7.03904	0.240	0.811
FK_read.ease	0.03608	1.01410	0.036	0.972
syllable.count	-0.03349	0.03978	-0.842	0.401
word.count	-0.58273	1.94936	-0.299	0.765
formality	0.05290	0.05565	0.951	0.343

Figure 7 Coefficients of Base model

4.2 Is mentioning the name of host important?

By observing the original data, some rooms have two hosts (such as Jo&Ray), however, the customers always mention only one of them in the comments. Besides, some reviewers like to use the nicknames of the hosts when they write comments for listings. So grepl() here may cause some mismatching. To increase the accuracy, the fuzzy matching is available. Before matching, the punctuation and the extra space should be removed.

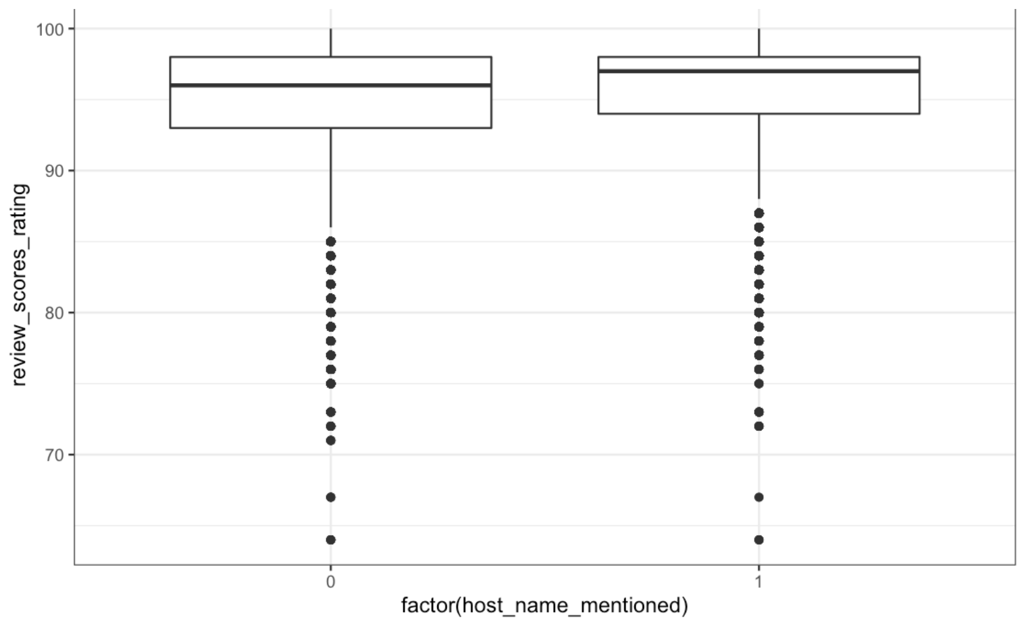
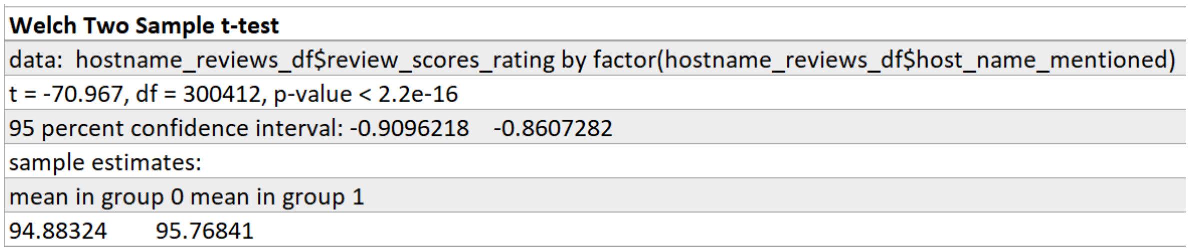


Figure 8 Boxplot of host_name_mentioned

From the Figure 9, we can see that there are very small differences between two groups. And in the Figure 10, the t-test of these two groups shows the p-value is extreme small. So, the average rating of two groups are different but the difference is slight.



Welch Two Sample t-test
data: hostname_reviews_df$review_scores_rating by factor(hostname_reviews_df$host_name_mentioned)
t = -70.967, df = 300412, p-value < 2.2e-16
95 percent confidence interval: -0.9096218   -0.8607282
sample estimates:
mean in group 0 mean in group 1
94.88324   95.76841

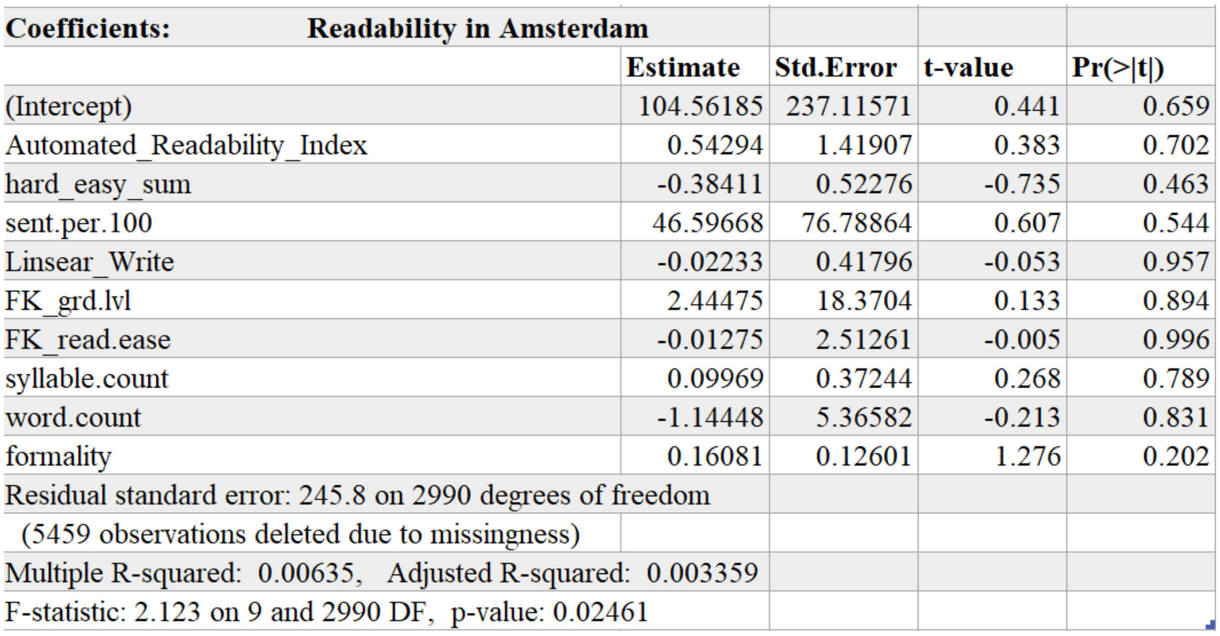
Figure 9 Welch Two Sample t-test

5. Investigate whether variables from description are related to the renting price

From the results from above stages, this report will investigate the relationship between the price and readability or normality. By the linear regression, the influence can be determined.

```
#----- get the price from the listings
all_listings <- all_data_df %>% select(listing_id,new_description,price) %>%
  unique(.)
```

```r
#-----Join the table to get the depedent variable
readability_price <- all_listings %>%
  left_join(readability_all)


# ----- For working with the price we need to parse it first as a numeric
readability_price$price <- as.numeric(gsub("\\$","",readability_price$price))

model1 <- lm(price~.,data=select(readability_price,-listing_id,-new_description))
summary(model1)
# ----- Using stepwise method to select significant variables
step(model1,steps = 10)
```

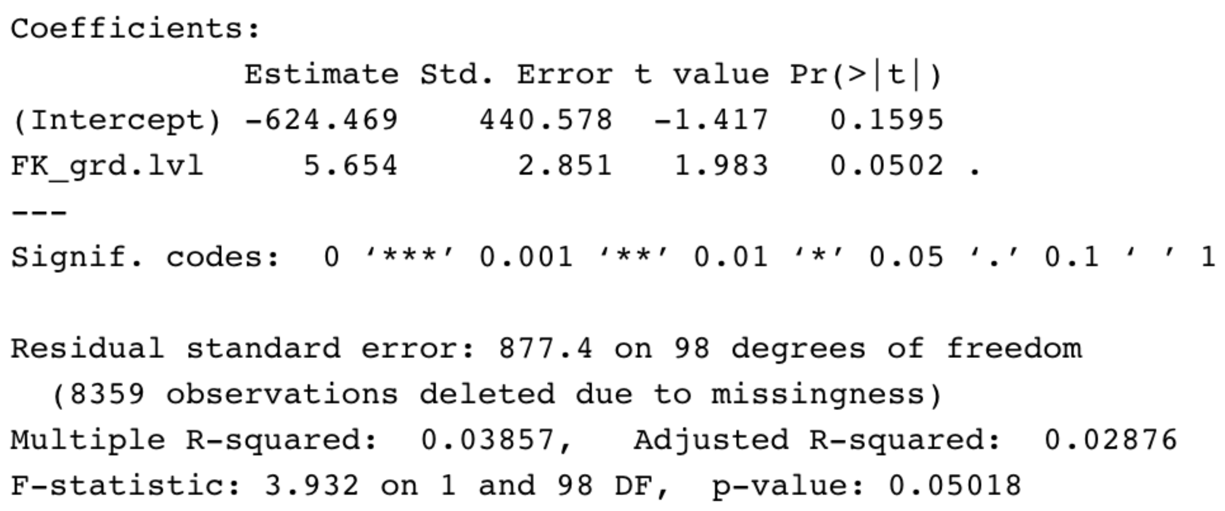


Coefficients: **Readability in Amsterdam**				
	Estimate	**Std.Error**	**t-value**	**Pr(>|t|)**
(Intercept)	104.56185	237.11571	0.441	0.659
Automated_Readability_Index	0.54294	1.41907	0.383	0.702
hard_easy_sum	-0.38411	0.52276	-0.735	0.463
sent.per.100	46.59668	76.78864	0.607	0.544
Linsear_Write	-0.02233	0.41796	-0.053	0.957
FK_grd.lvl	2.44475	18.3704	0.133	0.894
FK_read.ease	-0.01275	2.51261	-0.005	0.996
syllable.count	0.09969	0.37244	0.268	0.789
word.count	-1.14448	5.36582	-0.213	0.831
formality	0.16081	0.12601	1.276	0.202
Residual standard error: 245.8 on 2990 degrees of freedom				
(5459 observations deleted due to missingness)				
Multiple R-squared: 0.00635, Adjusted R-squared: 0.003359				
F-statistic: 2.123 on 9 and 2990 DF, p-value: 0.02461				

Figure 11 Coefficients of Readability in Amsterdam

From the output of the regression model (Figure 11) for Amsterdam, we can find that there is no significant factor. Besides, the R square and Adjusted R square are extremely small.

To choose effective variables, stepwise method is used. Only FK_grd.lvl is kept in the model (Figure 12). We can find that the coefficient of FK_grd.lvl is positive, which means the price will be higher if the readability of description increases.



```
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -624.469    440.578  -1.417   0.1595
FK_grd.lvl     5.654      2.851   1.983   0.0502 .
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 877.4 on 98 degrees of freedom
  (8359 observations deleted due to missingness)
Multiple R-squared:  0.03857,   Adjusted R-squared:  0.02876
F-statistic: 3.932 on 1 and 98 DF,  p-value: 0.05018
```

Figure 12 Coefficients of FK_grd.lvl

Part B Sentiment association with Occupancy rates and Price

```
start <- Sys.time()
```

1.Importing Data

The original R script to analyse the cities we selected uses a list of cities to iterate the analysis for part B. However, for readability, we put an example of city[1] which is Amsterdam, the first city in the list, as base case.

```
set.seed(10)
t1 <- Sys.time()

# ----- Data Importing
conn <- dbConnect(RSQLite::SQLite(), "dataset/inside_airbnb.db")

# ----- Get list of cities (markets) to run analyse and for loop for every ci
ty
city <- dbGetQuery(conn, 'SELECT distinct(market) FROM listing')
city <- city$market

# ----- Select listings
listing_sample <- dbGetQuery(conn, paste0('SELECT listing.*, annotated_descri
ption FROM listing
           LEFT JOIN description_udpipe ON listing.listing_id = description_u
dpipe.listing_id
           WHERE market = "',city[1], '"'))


listing_sample <- listing_sample[, !duplicated(colnames(listing_sample))] %>%
  mutate(security_deposit = as.numeric(gsub(",", "", substring(security_depos
it, 2)))) %>%
  mutate(cleaning_fee = as.numeric(gsub(",", "", substring(cleaning_fee, 2)))
) %>%
  mutate(extra_people = as.numeric(gsub(",", "", substring(extra_people, 2)))
) %>%
  sample_frac(0.40)


# ----- Select owners of the listings
host_distinct <- unique(listing_sample$host_id)
host_sample <- dbGetQuery(conn, paste('SELECT host_id, host_name FROM host WH
ERE host_id IN(', paste(host_distinct, collapse = ","), ')'))  %>%
  distinct(host_id, host_name)
```

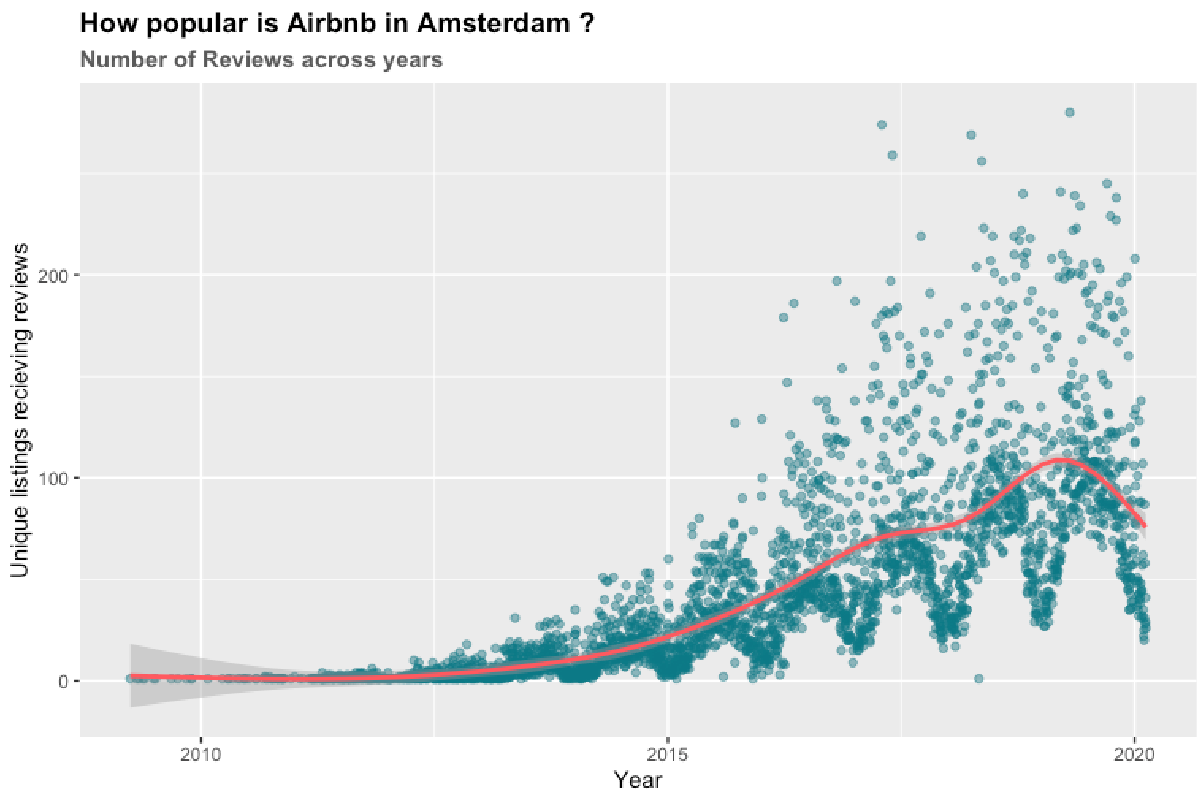
```r
# ----- Select reviews of the listings
listing_distinct <- listing_sample %>%  distinct(listing_id)

review_sample <- dbGetQuery(conn,paste('
                        SELECT listing_id, review.review_id, review_date, com
ments, comments_semi_cleaned, annotated_comments_partb, file_name
                        FROM review LEFT JOIN comments_cleaned ON review.revi
ew_id = comments_cleaned.review_id WHERE listing_id IN', substring(paste(list
ing_distinct,collapse = ","),2))) %>%
  mutate(review_date = as.Date(review_date))

# ----- Select calendar of the listings
calendar <- dbGetQuery(conn, paste('SELECT * FROM calendar WHERE listing_id I
N', substring(paste(listing_distinct,collapse = ","),2)))
save(calendar, file = 'temp/calendar.rda')
rm(listing_distinct)
rm(host_distinct)

dbDisconnect(conn)

t2 <- Sys.time()
t2-t1
```

2.Exploratory Data Analysis

To put analysis to context on a city level, this section provides basic EDA on Airbnb reviews growth trend, price trend and occupancy rate trend.

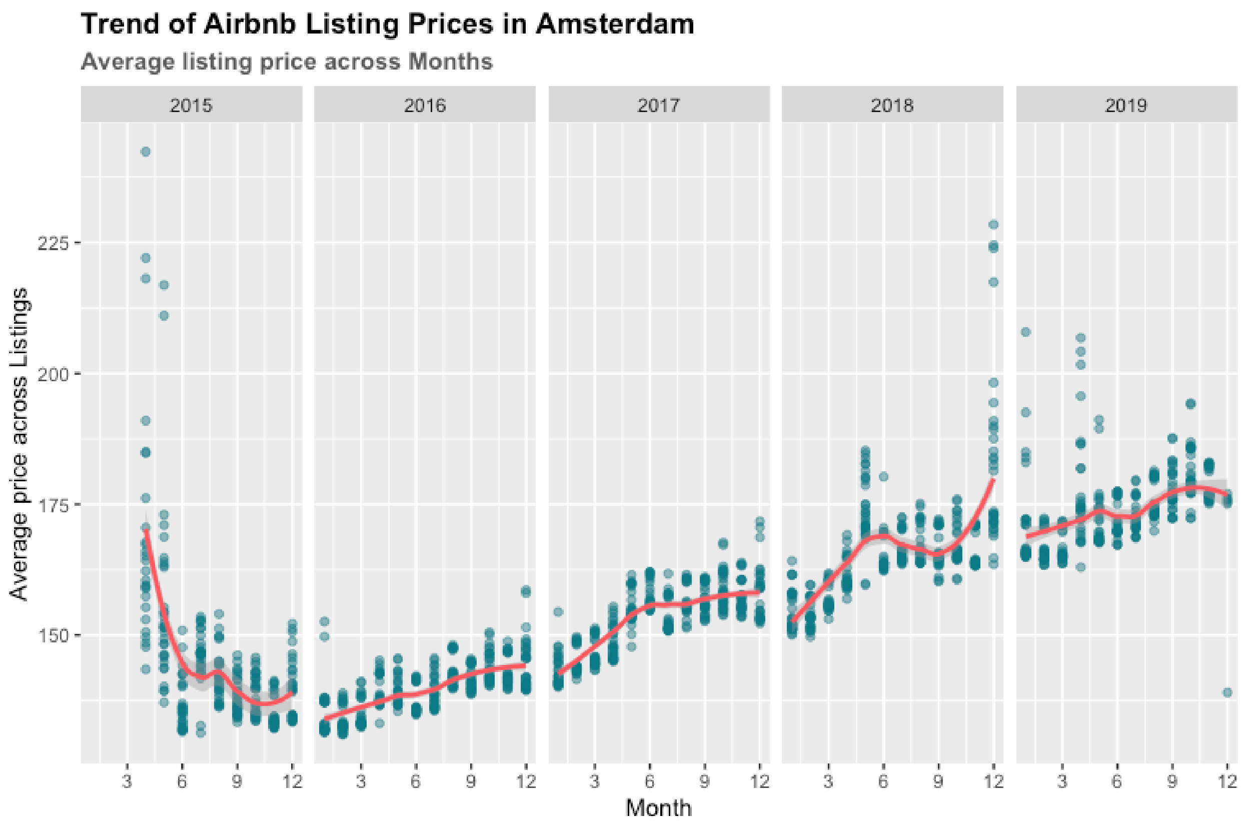
2.1 Understanding Reviews Growth

```
reviewsNum <- review_sample %>% group_by(review_date) %>% summarise(number =
n())

ggplot(reviewsNum, aes(review_date, number)) +
           geom_point(na.rm=TRUE, color = "#007A87", alpha=0.5) + geom_smooth
(color = "#FF5A5F")+
  ggtitle("How popular is Airbnb?",
          subtitle = "Number of Reviews across years") +
  labs(x = "Year", y = "Unique listings recieving reviews") +
  theme(plot.title = element_text(face = "bold")) +
  theme(plot.subtitle = element_text(face = "bold", color = "grey35")) +
  theme(plot.caption = element_text(color = "grey68"))

rm(reviewsNum)
```



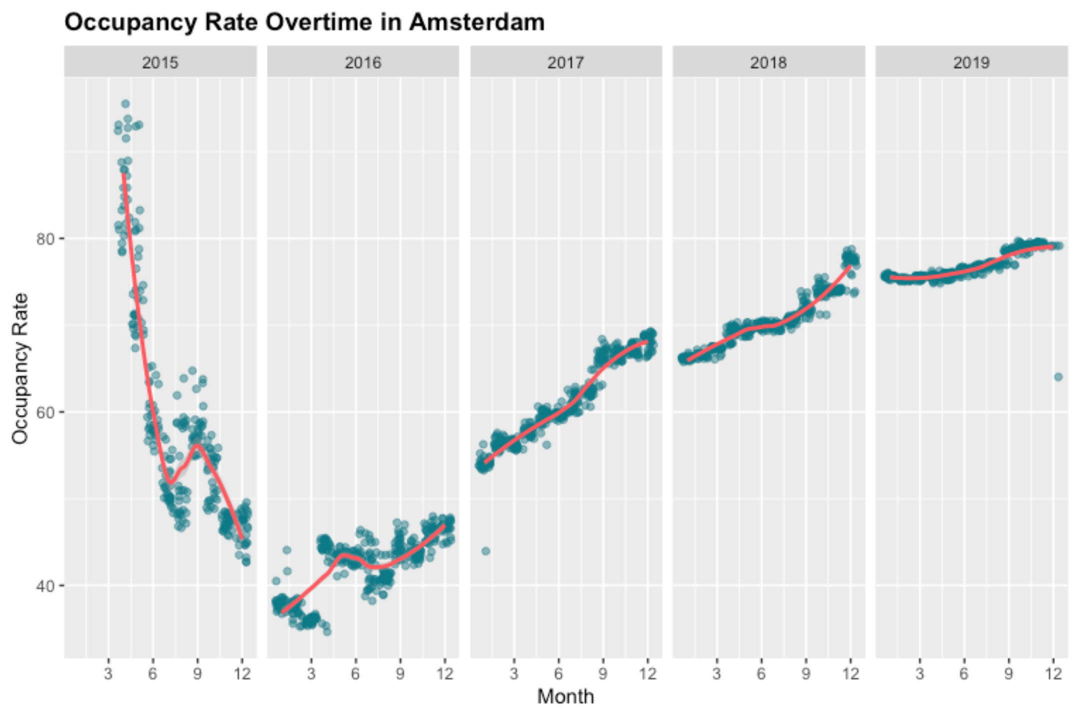
2.2 Understanding Price

```
load(file = 'temp/calendar.rda')

groupedCalendarAll <- calendar %>%
  group_by(bookingdate) %>%
  summarise(average_price = mean(price, na.rm = TRUE)) %>%
  mutate(year = as.factor(as.character(year(bookingdate))))

# ----- Trend in Listing Price
ggplot(groupedCalendarAll, aes(x = month(bookingdate), y=average_price)) +
           geom_point(na.rm=TRUE, alpha=0.5, color = "#007A87") + geom_smooth
(color = "#FF5A5F")+ facet_grid(~year)+
  ggtitle("Trend of Airbnb Listing Prices",
          subtitle = "Average listing price across Months") +
  labs(x = "Month", y = "Average price across Listings") +
  theme(plot.title = element_text(face = "bold")) +
  theme(plot.subtitle = element_text(face = "bold", color = "grey35")) +
  scale_x_continuous(breaks = c(3, 6, 9, 12))

rm(groupedCalendarAll)
```



2.3 Understanding Occupancy Rate

```r
airbnb_occ_rate <- calendar %>%
  group_by(bookingdate) %>%
  summarise(totalBooked = sum(booked, na.rm = TRUE), totalListings = n()) %>%

  mutate(percent_booked = (totalBooked/totalListings)*100) %>%
  mutate(year = year(bookingdate))

ggplot(airbnb_occ_rate, aes(x = month(bookingdate), y = percent_booked)) +
  geom_jitter(na.rm=TRUE, alpha=0.5, color = "#007A87") +
  geom_smooth(color = "#FF5A5F") +
  facet_grid(~year) +
  ggtitle("Occupancy Rate Overtime") +
  labs(x = "Month", y = "Occupancy Rate") +
  theme(plot.title = element_text(face = "bold")) +
  theme(plot.subtitle = element_text(face = "bold", color = "grey35")) +
  scale_x_continuous(breaks = c(3, 6, 9, 12))

rm(airbnb_occ_rate)
```



3. Feature Extraction

There are many data points made public by airbnb on reviews textual data and listing descriptions. Review dataset only contains textual comments with the date of the review and to which listing was the review made for. While the listing dataset has an extensive number of structured features. This section focuses on extracting as many useful text features as possible both from listings and reviews data, which then later be used to build predictive models.

3.1 Feature Extraction from reviews

One of the most important feature we could extract from customer reviews are the sentiment scores, as they will vary a lot per experience of a customer at a particular airbnb listing. Instead of using tidyr approach of sentiment tokenisiation, textfeatures package covers some of the most common text feature extraction. Additionally, the package is run on top of text2vec, which is a fast and memory-friendly text vectorization package that is going to be used later in listing feature extractions.

Sentiments scores extracted includes affin, syuzhet, bing, and vader. After investigating the github repository (https://github.com/mkearney/textfeatures), we found out that the approach follows the same logic of tidyr sentiment analysis where the words are tokenised in the back-end and then scored. Thus, we beleive getting sentiment from this approach is valid. As a bonus, generic features such as n_char, lower_cases, uppercases, commas, periods, line breaks are also extracted. In addition to the bag of word approach on sentiment scoring, we added sentimentr that attempts to take into account valence shifters (i.e.,negators, amplifiers (intensifiers), de-amplifiers (downtoners), and adversative conjunctions) while maintaining speed.

```
t1 <- Sys.time()

# ----- Get features from uncleaned comments
features <- textfeatures(review_sample$comments, normalize = FALSE, word_dims
 = FALSE, sentiment = TRUE)

# ----- Get sentiment from cleaned comments with sentimentr package
text <- get_sentences(review_sample$annotated_comments_partb) #changeee-partb
cleaned
review_sample$sentimentr <- as.data.frame(sentiment_by(text))$ave_sentiment

# ----- Whether host name is mentioned in the comments
review_sample <- review_sample %>%
  left_join(listing_sample %>% select(listing_id, host_id)) %>%
  left_join(host_sample %>% distinct(host_id, host_name)) %>%
  mutate(host_mentioned = as.numeric(grepl(host_name,comments, ignore.case =
TRUE)))

# ----- DF about comments features
review_features <-  bind_cols(review_sample, features)

save(review_features, file = 'temp/review_features.rda')
rm(text)
rm(features)
rm(host_sample)

# ----- Aggregate the features from review accordingly
review_features_agg <- review_features %>%
  select(-c(host_name)) %>%
  group_by(listing_id, file_name) %>%
  summarise_if(is.numeric, mean, na.rm = TRUE)
save(review_features_agg, file = 'temp/review_features_agg.rda')
rm(review_features_agg)

review_sentiment_monthly <- review_features %>%
  mutate(year_month =  format(review_date,"%Y-%m"), year_month_back = format(
review_date - years(1), "%Y-%m")) %>%
  select(-c(host_name)) %>%
  group_by(listing_id, file_name, year_month, year_month_back) %>%
  summarise_if(is.numeric, mean, na.rm = TRUE) %>%
  select(listing_id, year_month, file_name, year_month_back, year_month_back,
 sent_afinn, sent_bing, sent_syuzhet, sent_vader, sentimentr)

save(review_sentiment_monthly,file = "temp/review_sentiment_monthly.rda")
rm(review_sentiment_monthly, review_features, review_sample)

t2 <- Sys.time()
t2 - t1
```

3.2 Feature Extraction from listing data

In contrast to listing data, it is assumed that the description of the listing do not change over period as only the latest listing description were considered. Additionally, scoring a sentiment on a listing's description would not be ideal either since hosts are always going to tell positive sentiment about their properties given their position as listing owner. Therefore, other ways to extract features from listing textual descriptions has to be conducted.

The Term Document Frequency - Inverse Document Frequency (TF-IDF), a technique to quantify a term in a documents wheras a weight is computed and assigned to each term-document. A TF-IDF is highest when a term occurs many times within a small range of documents. Thus help discriminating between documents. In contrast, TF-IDF is lowest when a term occurs frequently in all documents. This measure was known to be a good base case for classifying spam texts. We decided to exploit such characterstics to extract features to use as inputs for regression models later in the chapter.

To be able to build TF-IDF feature matrix, first a Document-Term Matrix (DTM) has to be built on vocabs of the cleaned and lemmatised textual description column, where only nouns are used. The decision to keep only nouns is driven by the assumption that having some specific words might increase or decrease the value of a lisitng such as price. For instance, a listing which might have the word 'view' or 'swimming pool' is expected to charge more for such luxury. And mostly, nouns as part-of-speech can capture exactly that. Afterwards, the DTM is transformed to TF-IDF matrix with the function in text2vec package. Finally, the TF-IDF features are merged with the listing dataset as extension to the available structured listing data.

```
library(text2vec)

t1 <- Sys.time()

# ----- define preprocessing function and tokenization function
prep_fun <- tolower
```

```r
tok_fun <- word_tokenizer
term_min <- round(length(listing_sample$annotated_description)*0.01)
# term_max <- round(length(listing_sample$new_description_cleaned)*0.1)

# ----- Initiate tokeniser
it_train <- itoken(listing_sample$annotated_description,
             preprocessor = prep_fun,
             tokenizer = tok_fun,
             ids = listing_sample$listing_id,
             progressbar = TRUE)

# ----- Building vocabularies for document term matrix
vocab <- create_vocabulary(it_train) #  ngram = c(1L, 2L)
pruned_vocab <- prune_vocabulary(vocab,
                                 # doc_count_min = doc_min,
                                 term_count_min = term_min,
                                 # term_count_max = term_max,
                                 doc_proportion_max = 0.5,
                                 doc_proportion_min = 0.001)
rm(vocab)
rm(term_min)

# ----- Define how to ransform list of tokens into vector space to build dtm
vectorizer <- vocab_vectorizer(pruned_vocab)
rm(pruned_vocab)

# ----- Build DTM
dtm_train  <- create_dtm(it_train, vectorizer)
rm(it_train)

# ----- Build weighted TF-IDF matrix
tfidf <- TfIdf$new() # initiate instance
dtm_train_tfidf <- fit_transform(dtm_train, tfidf)

rm(tfidf)
rm(dtm_train)

tf_idf_dataframe <- as.data.frame(as.matrix(dtm_train_tfidf))
rm(dtm_train_tfidf)

colnames(tf_idf_dataframe) <- paste0(colnames(tf_idf_dataframe), ("_word__"))
tf_idf_dataframe$listing_id <- substring(rownames(tf_idf_dataframe), 2)

# ----- Finalise Listing Features
listing_features <- listing_sample %>%
  inner_join(tf_idf_dataframe, by = 'listing_id')

save(listing_features,file = "temp/listing_features.rda")
save(tf_idf_dataframe,file = "temp/tf_idf_dataframe.rda")
```

```r
rm(listing_features, tf_idf_dataframe)

load(file = 'temp/tf_idf_dataframe.rda')

tf_idf_dataframe
t2 <- Sys.time()
t2 - t1
```

3.3 Feature Extraction from calendar

There will be two types of aggregation types will be used in further analysis; aggregation by listing, and aggregation by listing and year-month.

```r
load(file = 'temp/calendar.rda')
load(file = 'temp/listing_features.rda')

# ----- Generate aggregated calendar features per listing
calendar_features_perlisting <- calendar %>%
  mutate(bookingdate = as.Date(bookingdate), year_month =  format(bookingdate
,"%Y-%m")) %>%
  filter(year(bookingdate) %in% c('2017','2018','2019')) %>%
  group_by(listing_id) %>%
  summarise(total_booked = sum(booked, na.rm = TRUE), total_dates = n(), aver
age_price = mean(price, na.rm = TRUE)) %>%
  left_join(listing_features %>% dplyr::select(listing_id, price), by = 'list
ing_id') %>%
  mutate(average_price = ifelse(is.nan(average_price), price, average_price))
 %>%
  mutate(occupancy_rate = (total_booked/total_dates)*100) %>%
  dplyr::select(-c(total_booked, total_dates, price))

save(calendar_features_perlisting, file = 'temp/calendar_features_perlisting.
rda')
rm(calendar_features_perlisting)

# ----- Generate aggregated calendar features per listing per month
calendar_features_monthly <- calendar %>%
  mutate(bookingdate = as.Date(bookingdate), year_month =  format(bookingdate
,"%Y-%m")) %>%
  filter(year(bookingdate) %in% c('2017','2018','2019')) %>%
  group_by(listing_id, year_month) %>%
  summarise(total_booked = sum(booked, na.rm = TRUE), total_dates = n(), mont
hly_average_price = mean(price, na.rm = TRUE)) %>%
  left_join(listing_features %>% dplyr::select(listing_id, price), by = 'list
ing_id') %>%
  mutate(monthly_average_price = ifelse(is.nan(monthly_average_price), price,
 monthly_average_price)) %>%
  mutate(monthly_occupancy_rate = (total_booked/total_dates)*100) %>%
  dplyr::select(-c(total_booked, total_dates, price))

save(calendar_features_monthly, file = 'temp/calendar_features_monthly.rda')
rm(calendar_features_monthly, calendar)
```

3.4 Final Dataframe

```r
# ----- Final DF per Listing
load(file='temp/calendar_features_perlisting.rda')
load(file = 'temp/review_features_agg.rda')
load(file = 'temp/listing_features.rda')

final_df_perlisting <- calendar_features_perlisting %>%
  inner_join(listing_features, by = 'listing_id') %>%
  inner_join(review_features_agg, by = 'listing_id')

save(final_df_perlisting, file = 'temp/final_df_perlisting.rda')
rm(calendar_features_perlisting, final_df_perlisting)

# ----- Final DF per Listing per year-month
load(file = 'temp/calendar_features_monthly.rda')
load(file = 'temp/review_sentiment_monthly.rda')

final_df_monthly <- calendar_features_monthly %>%
  inner_join(review_sentiment_monthly, by = 'listing_id')


save(final_df_monthly, file = 'temp/final_df_monthly.rda')
rm(calendar_features_monthly, review_features_agg, listing_features, final_df
_monthly, review_sentiment_monthly)
```

4. How does different features of text review affect the sentiment score of a review?

We are interested to understand more about sentiment scoring. We started with an assumption like having more exclamation marks may indicate a more satisfied reviewer while more questions marks indicate doubt. Again, instead of pondering, let’s put the idea to the test.

4.1 Attempt to create additional feature based on asssumption

```r
load(file = 'temp/review_features.rda')
```

```r
# ----- Exaggeration Sentiment: Create our own sentiment on the assumption th
at double or more exclamation marks mean the reviewer is more satisfied and d
ouble or more question marks mean the reviewer has more doubt to the listing
or the service

creative_measures <- review_features %>%
select(review_id, comments_semi_cleaned) %>%
mutate(exclamation_2 = str_detect(comments_semi_cleaned, fixed("!!")),
       exclamation_3 = str_detect(comments_semi_cleaned, fixed("!!!")),
       exclamation_4 = str_detect(comments_semi_cleaned, fixed("!!!!")),
       question_2 = str_detect(comments_semi_cleaned, fixed("??")),
       question_3 = str_detect(comments_semi_cleaned, fixed("???")),
       question_4 = str_detect(comments_semi_cleaned, fixed("????")),
       exclamation_sent = exclamation_2 + exclamation_3 +exclamation_4,
       question_sent = question_2 + question_3 + question_4,
       sent_exaggeration = exclamation_sent - question_sent) %>%
select(review_id, comments_semi_cleaned, sent_exaggeration)



# ----- Capslock Sentiment: we assumed that the presence of all-capitalized w
ords in airbnb's comments is more a sign of customer satisfaction, rather tha
n a sign that customers are yelling

creative_measures <- creative_measures %>%
  mutate(comments_semi_cleaned = removePunctuation(comments_semi_cleaned))

capwords <- unlist(regmatches(creative_measures$comments_semi_cleaned, gregex
pr("(?<!\\S)(?:(?=\\S*\\p{L})(?=\\S*\\d)\\S+|(?:\\S*\\p{Lu}){3}\\S*)", creati
ve_measures$comments_semi_cleaned, perl=TRUE))) # List words in the comments
that consist of more than 3 uppercase letters


capwords <- unique(capwords)
capwords <- capwords[-matches(vars = capwords, "airb|bnb|rai|ndsm|ams|eur|kfc
|dvd|bbq|atm|gps|fyi|usa|nyc|wifi|cbd|usb", ignore.case = TRUE)] # Manually r
emove some capital stopwords

creative_measures <-
  creative_measures %>%
  mutate(caps = sapply(stringi::stri_extract_all_regex(str = creative_measure
s$comments_semi_cleaned, pattern = paste(capwords, collapse = "|")), toString
), sent_capslock = ifelse(caps == "NA", 0, 1)) %>%
  select(-c(comments_semi_cleaned, caps))

load(file= 'temp/review_features.rda')

hist(review_features$n_chars, breaks = 200)
hist(review_features$n_polite, breaks = 200)
```

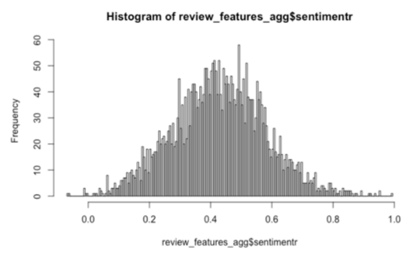
```r
model_data <- review_features %>%
  left_join(creative_measures, by = 'review_id') %>%
  select(n_chars, n_polite, sentimentr, host_mentioned, n_periods, sent_exagg
eration, sent_capslock)

sentimentr_model_features <- lm(sentimentr ~n_chars + n_polite + host_mention
ed + n_periods + sent_exaggeration + sent_capslock, data=model_data)
summary(sentimentr_model_features)

rm(sentimentr_model_features)
```

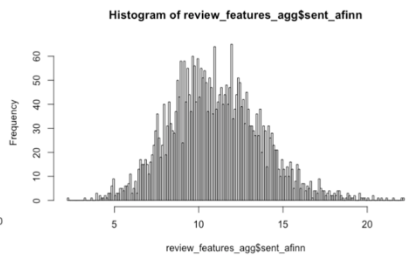
```
Call:
lm(formula = sentimentr ~ n_chars + n_polite + host_mentioned +
    n_periods + sent_exaggeration + sent_capslock, data = model_data)

Residuals:
     Min       1Q   Median       3Q      Max
-2.31273 -0.28098 -0.00509  0.27554  2.39538

Coefficients:
                    Estimate Std. Error t value Pr(>|t|)
(Intercept)        5.722e-01  3.199e-03 178.901  < 2e-16 ***
n_chars            1.745e-04  1.263e-05  13.810  < 2e-16 ***
n_polite          -1.128e-01  2.532e-03 -44.554  < 2e-16 ***
host_mentioned     2.163e-02  3.086e-02   0.701   0.4833
n_periods         -4.228e-03  7.652e-04  -5.525 3.31e-08 ***
sent_exaggeration  2.407e-02  3.965e-03   6.072 1.27e-09 ***
sent_capslock     -1.801e-02  8.120e-03  -2.218   0.0265 *
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.4272 on 95717 degrees of freedom
  (43522 observations deleted due to missingness)
Multiple R-squared:  0.02445,	Adjusted R-squared:  0.02438
F-statistic: 399.8 on 6 and 95717 DF,  p-value: < 2.2e-16
```

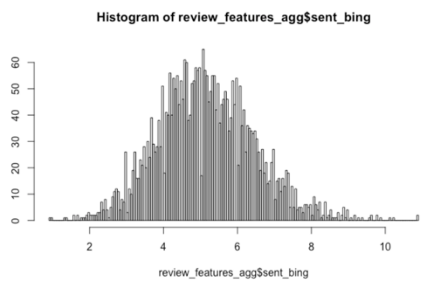
Interestingly, both the new added sentiment mesures are significant on the sentimentr score. Number of characters, politeness level, number of sentences, exaggerations and capslock are significant on sentiment, together explaining 2% variance in sentiment score.

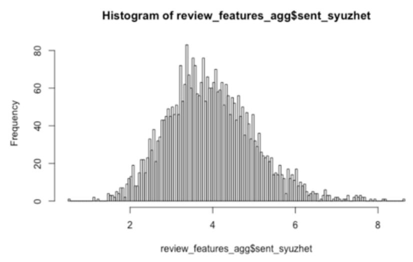
5. Which sentiment measure predicts a listing’s average rating better?

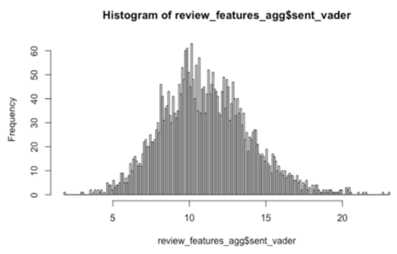
As we have generated different sentiment scores, the question that comes first in mind is which of them predicts rating better. Since the rating score in the listing file is aggregated by Airbnb, we need to be aware that this analysis might figure more of how Airbnb aggregates the rating than it is to predict a reviewer’s individual rating.











There are five sentiment scores we would like to play around with; Sentimentr, Bing, Afinn, Syuzhet, and Vader. We managed to find that each sentiment has similar distribution which confirming with the normal distribution. Therefore, at least we can be sure that they are most likely has done their work on discriminating between reviews properly.

```r
load(file = 'temp/review_features_agg.rda')
load(file= "temp/listing_features.rda")

# ----- Look into distribution of sentiments
hist(review_features_agg$sentimentr, breaks = 200)
hist(review_features_agg$sent_bing, breaks = 200)
hist(review_features_agg$sent_afinn, breaks = 200)
hist(review_features_agg$sent_syuzhet, breaks = 200)
hist(review_features_agg$sent_vader, breaks = 200)

rating_model_data <- review_features_agg %>% left_join(listing_features %>% s
elect(listing_id, review_scores_rating), by = 'listing_id')
rm(review_features_agg, listing_features)
```

```
# ----- Build sentiment models
rating_model_sentimentr <- lm(log(review_scores_rating)~lag(sentimentr), data
=rating_model_data)
rating_model_sent_bing <- lm(log(review_scores_rating)~lag(sent_bing), data=r
ating_model_data)
rating_model_sent_afinn <- lm(log(review_scores_rating)~lag(sent_afinn), data
=rating_model_data)
rating_model_sent_syuzhet <- lm(log(review_scores_rating)~lag(sent_syuzhet),
data=rating_model_data)
rating_model_sent_vader <- lm(log(review_scores_rating)~lag(sent_vader), data
=rating_model_data)


# ----- Compare models
stargazer::stargazer(rating_model_sentimentr,rating_model_sent_bing,rating_mo
del_sent_afinn,rating_model_sent_vader,type = "text")

rm(rating_model_data, rating_model_sentimentr, rating_model_sent_bing, rating
_model_sent_afinn, rating_model_sent_syuzhet, rating_model_sent_vader)
```

```
=================================================================
                                     Dependent variable:
                              -----------------------------------
                                  log(review_scores_rating)
                                (1)      (2)      (3)      (4)
-----------------------------------------------------------------
lag(sentimentr)                0.006
                              (0.007)

lag(sent_bing)                          0.001
                                       (0.001)

lag(sent_afinn)                                  0.001*
                                                (0.0004)

lag(sent_vader)                                           0.001*
                                                         (0.0004)

Constant                      4.557*** 4.555*** 4.551*** 4.552***
                              (0.003)  (0.004)  (0.004)  (0.004)

-----------------------------------------------------------------
Observations                   1,369    1,369    1,369    1,369
R2                             0.001    0.001    0.003    0.002
Adjusted R2                   -0.0002  0.0001   0.002    0.002
Residual Std. Error (df = 1367) 0.041   0.041    0.041    0.041
F Statistic (df = 1; 1367)     0.794    1.196    3.823*   3.250*
=================================================================
Note:                                *p<0.1; **p<0.05; ***p<0.01
```

Sentiment Afinn performs best relative to other sentiment measures with R-squared of 0.03.

6. How good sentiment on predicting average monthly price of a listing?

```r
load(file = 'temp/final_df_monthly.rda')

# Bing Liu
price_model_bingliu <- lm(log(monthly_average_price)~lag(sentimentr),
             data=final_df_monthly)

# Afinn
price_model_affin <- lm(log(monthly_average_price)~lag(sent_afinn),
             data=final_df_monthly)
# Syuzhet
price_model_syuzhet <- lm(log(monthly_average_price)~lag(sent_syuzhet),
             data=final_df_monthly)

# Vader
price_model_vader <- lm(log(monthly_average_price)~lag(sent_vader),
             data=final_df_monthly)

stargazer::stargazer(price_model_bingliu,price_model_affin,price_model_syuzhe
t,price_model_vader,type = "text")

rm(price_model_bingliu, price_model_affin,price_model_syuzhet,price_model_vad
er)
```

```
=================================================================================
                                           Dependent variable:
                                  -----------------------------------------------
                                          log(monthly_average_price)
                                     (1)         (2)          (3)          (4)
---------------------------------------------------------------------------------
lag(sentimentr)                   0.032***
                                  (0.001)

lag(sent_afinn)                              0.005***
                                             (0.0001)

lag(sent_syuzhet)                                         0.009***
                                                          (0.0002)

lag(sent_vader)                                                        0.005***
                                                                       (0.0001)

Constant                          4.862***   4.816***     4.839***     4.821***
                                  (0.001)    (0.001)      (0.001)      (0.001)

---------------------------------------------------------------------------------
Observations                      740,255    740,255      740,255      740,255
R2                                 0.001      0.004        0.002        0.004
Adjusted R2                        0.001      0.004        0.002        0.004
Residual Std. Error (df = 740253)  0.487      0.486        0.487        0.486
F Statistic (df = 1; 740253)     465.362*** 3,336.070*** 1,453.543*** 2,918.913***
=================================================================================
Note:                                                 *p<0.1; **p<0.05; ***p<0.01
```

It turns out that sentiments can only explain maximum 0.004 variance in monthly average price of a listing. This might be due to the fact that regardless of the review, a listing with more facilities will tend to have higher price as the host needs to pay more for the rent.

7. Predicting Price and Occupancy Rate using a Listing's information

Now, let's put the generated features from the feature extraction section are put into test. This section aims to prove whether adding text data as features could improve one's model to predict a listing's price or its occupancy rate.

As a base case, models are first built using the features given in the datasets, such as the average sentiments and the number of rooms. Then, the additive predictability will be compared using the adjusted R-squared as it explains the explanatory power of regression models while taking into account the number of predictors (Minitab Editor, 2013). Thus, it would be comparable as the base R-square will always increase as the number of predictors increase.

Base Models with structured data

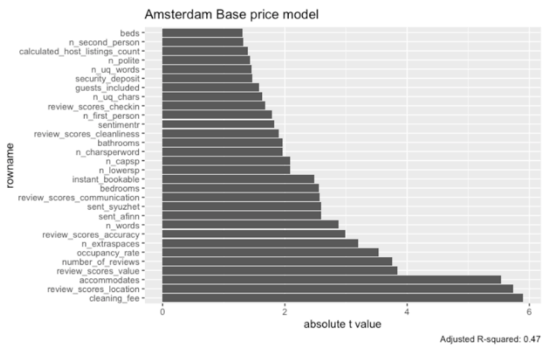
```
remove_columns <- c('listing_id', 'last_scraped', 'name', 'new_description',
'transit', 'host_id.x','city', 'neighbourhood_group_cleansed','state','zipcod
e','country_code', 'country','amenities','calendar_last_scraped','first_revie
w', 'file_name.y', 'price','pre_processed.x', 'pre_processed.y','file_name.x'
,'lang','market', 'neighbourhood_cleansed', 'new_description_cleaned','host_i
d.y','pre_processed','n_urls', 'availability_365', 'availability_90', 'availa
bility_60', 'availability_30')

load(file = 'temp/final_df_perlisting.rda')
load('temp/tf_idf_dataframe.rda')

# ----- Data Prep for the plot
data_without_tfidfmatrix <- final_df_perlisting %>%
  ungroup() %>%
  filter(average_price != 0) %>%
  select_if(!names(.) %in% remove_columns) %>%
  select_if(!names(.) %in% names(tf_idf_dataframe)) %>%
  select_if(is.numeric)

data_without_tfidfmatrix <- data_without_tfidfmatrix[,colSums(is.na(data_with
out_tfidfmatrix))<nrow(data_without_tfidfmatrix)] # remove NA columns

rm(tf_idf_dataframe)

price_base_model <- lm(log(average_price)~., data = data_without_tfidfmatrix)
occupancy_base_model <- lm(occupancy_rate~., data = data_without_tfidfmatrix)

summary(price_base_model)

rm(data_without_tfidfmatrix)
```

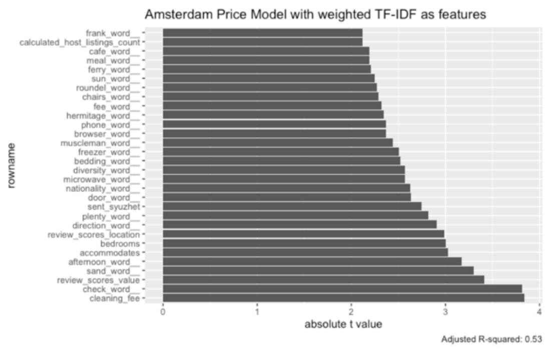
Models with TF-IDF Matrix added

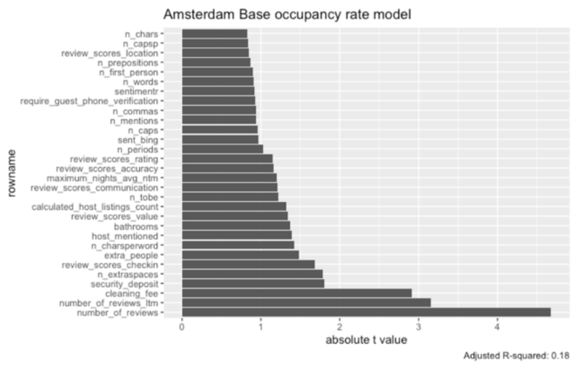
```
data_with_tfidfmatrix <- final_df_perlisting %>%
  filter(average_price != 0) %>%
  ungroup() %>%
  select_if(!names(.) %in% remove_columns) %>%
  select_if(is.numeric)

# data_with_tfidfmatrix <- data_with_tfidfmatrix[,colSums(is.na(data_with_tfi
dfmatrix))<nrow(data_with_tfidfmatrix)] # remove NA columns

price_model_with_tfidf <- lm(log(average_price)~., data = data_with_tfidfmatr
ix)
occupancy_model_with_tfidf <- lm(occupancy_rate~., data = data_with_tfidfmatr
ix)

summary(price_model_with_tfidf)

rm(data_with_tfidfmatrix)
```

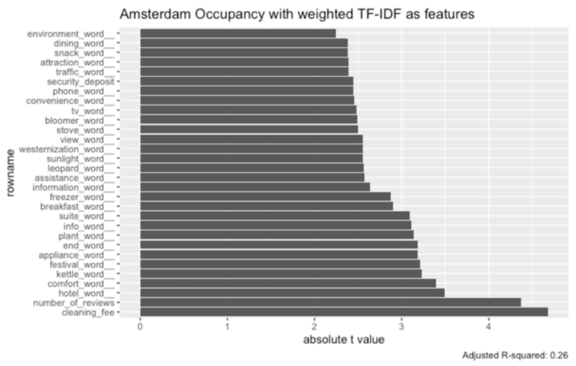
Comparing Models

```r
# ----- Plot the most significant variables
head(as.data.frame(summary(price_base_model)$coefficients) %>%
  tibble::rownames_to_column() %>%
  mutate(`absolute t value` = abs(`t value`)) %>%
  arrange(`Pr(>|t|)`) , 30) %>%
  mutate(rowname=factor(rowname, levels=rowname)) %>%
  ggplot(aes(x = rowname, y = `absolute t value`)) + geom_col() + coord_flip(
) + labs(title = paste(city[2],'Base price model'), caption = paste('Adjusted
 R-squared:', round(summary(price_base_model)$adj.r.squared,3)))


head(as.data.frame(summary(price_model_with_tfidf)$coefficients) %>%
  tibble::rownames_to_column() %>%
  mutate(`absolute t value` = abs(`t value`)) %>%
  arrange(`Pr(>|t|)`) , 30) %>%
  mutate(rowname=factor(rowname, levels=rowname)) %>%
  ggplot(aes(x = rowname, y = `absolute t value`)) + geom_col() + coord_flip(
) + labs(title = paste(city[2],'Price Model with weighted TF-IDF as features'
),caption = paste('Adjusted R-squared:', round(summary(price_model_with_tfidf
)$adj.r.squared,3)))


# ----- Plot the most significant variables
head(as.data.frame(summary(occupancy_base_model)$coefficients) %>%
  tibble::rownames_to_column() %>%
  mutate(`absolute t value` = abs(`t value`)) %>%
  arrange(`Pr(>|t|)`) , 30) %>%
  mutate(rowname=factor(rowname, levels=rowname)) %>%
  ggplot(aes(x = rowname, y = `absolute t value`)) + geom_col() + coord_flip(
) + labs(title = paste(city[2],'Base occupancy rate model'), caption = paste(
'Adjusted R-squared:', round(summary(occupancy_base_model)$adj.r.squared,3)))


head(as.data.frame(summary(occupancy_model_with_tfidf)$coefficients) %>%
  tibble::rownames_to_column() %>%
  mutate(`absolute t value` = abs(`t value`)) %>%
  arrange(`Pr(>|t|)`) , 30) %>%
  mutate(rowname=factor(rowname, levels=rowname)) %>%
  ggplot(aes(x = rowname, y = `absolute t value`)) + geom_col() + coord_flip(
) + labs(title = paste(city[2],'Occupancy with weighted TF-IDF as features'),
caption = paste('Adjusted R-squared:', round(summary(occupancy_model_with_tfi
df)$adj.r.squared,3)))


save(price_base_model, file ='price_base_model.rda')
save(price_model_with_tfidf, file ='price_model_with_tfidf.rda')
rm(price_base_model, price_model_with_tfidf)

end <- Sys.time()
end - start
```









In both price and occupancy rate, the adjusted R-squared has quite decent increase. The adjusted R-squared for price model changes from 0.47 to 0.53, while for occupancy rate, from 0.18 to 0.26.

But the most interesting finding is that the significance of the variables changes quite dramatically, where in the model with TF-IDF features more words variables are dominant in the top 30 most significant variables. This proves that there are many potentials could be unlocked by quantifying textual data and put them into use for predictive modelling.

We need to acknowledge that TF-IDF matrix might not be the most efficient way to use for modelling as the dataset becomes wide quite easily. A more prominent approach would be to turn the words into vectors thus reducing the dimensionality and build the regression model.

Part C: Topic Modelling

In this part, we will focus on exploring what topics customers pay attention to and how these topics predict price and review scores. Therefore, in order to fulfil these requirements, we will process all the comments firstly which are cleaned, tokenized and lemmatized in the preprocessing part, and then create the documents for topic modelling stage. Through executing the modelling techniques, specific topics can be obtained and used for next stage – evaluating the relationships among these topics, price and review scores rating. In this stage, we will discuss from two aspects – how these topics are affected and how they estimate the price and rating scores.

1. Data Preparation

1.1 Data preparation

As mentioned before, all of files are preprocessed before and stored in a db file. Thus, we can scrap documents and variables from the db file through RSQLite package.

```
#-----Accessing the database

library(RSQLite)

conn <- dbConnect(RSQLite::SQLite(), "inside_airbnb.db/inside_airbnb.db")
#----- Scraping the review file
review <- dbGetQuery(conn, paste('SELECT review.* FROM review LEFT JOIN

WHERE market = "', city[1], '"'))

#-----Scraping the annotated review stored in the comments_clean table.

#Those annotated comments are all nouns since the use of only nouns in the to
pic modelling will improve the efficiency of the process (Martin.Fand Johnson
.M, 2015)
comments_clean <- dbGetQuery(conn,"SELECT comments_cleaned.annotated_comments
_partc,comments_cleaned.review_id FROM comments_cleaned")
review <- review %>% left_join(comments_clean)


#-----Scraping some variables from listing file                          listin
g <- dbGetQuery(conn, paste('SELECT review_scores_rating,price,listing_id FRO
M listing WHERE market = "', city[1], '"'))

ams_for_stm <- listing %>% left_join(review) %>% na.omit()
dbDisconnect(conn)
#In order to speed the process, we decide to use sampling
```

```
set.seed(123)
ams_for_stm <- ams_for_stm %>% sample_frac(0.4) %>% na.omit()

#----Cleaning data again – removing those listings whose reviews are less tha
n 5

to_remove_listing_ids <- ams_for_stm %>% group_by(listing_id) %>% summarise(t
otal=n()) %>% filter(total <= 5) %>% pull(listing_id)

ams_for_stm <- ams_for_stm %>% filter(!(listing_id %in% to_remove_listing_ids
))
```

2. Text Processing

In this stage, we will extract documents used for topic modelling, some vocabs which can be used as key words in topics and the dataset.

```
#-----Text Processing - removing customstopwords
processed <- textProcessor(ams_for_stm$annotated_comments_partc,
                           metadata = ams_for_stm,
                           customstopwords = c("airbnb","apartment","amsterda
m","house","thanks","thing","flat","welcome","lot","place","end","bit","one",
"part","stay","accomodation","plenty","frank","jordaan","liive"),
                           stem = F)

#-----Removing missing value in the prevalence covariates. In the process of
topic modelling ,missing value in the prevalence covariates can not be existe
d as well. But for this city, there are no missing value.

#nrow(processed$meta)
#nrow(processed$meta) - sum(is.na(processed$meta$review_scores_rating))
#nrow(processed$meta) - sum(is.na(processed$meta$price)) # The results show t
hat there are some missing value in the price.

#keep_price <- !is.na(processed$meta$price) # Detecting missing value
#meta <- processed$meta[keep_price,] # Keeping only obeserved values in meta
and docs
#docs <- processed$documents[keep_price]

#-----Generalising prepDocuments for stm
threshold <- round(1/100 * length(processed$documents),0)


out <- prepDocuments(processed$documents,
                     processed$vocab,
                     processed$meta,
                     lower.thresh = threshold)
```

```r
#Checking words without specific meaning and treating them as stopwords and r
unning again.
#out$vocab


#save(out,file = "out.rda")
```

3. Run the model to get topics

After documents preparation, we can execute the topic modelling to find optimal topics through stm() function. But the number of topics (K) first need to be decided first. Based on the topic res ults of K=0, an optimal number 29 is given automatically. For the further check and decision ma king, search function is utilized to evaluate topics after K setting as 25, 30 and 35 which are all c lose to 29. The results of summary show that 30 and 35 topics will be more readable. However, as Mimno (2011) mentioned, the size of topics has a strong relationship with the probability of to pics being nonsensical. Therefore, although the results of 35 is better than that of 30, we still us e 30 as topic number.

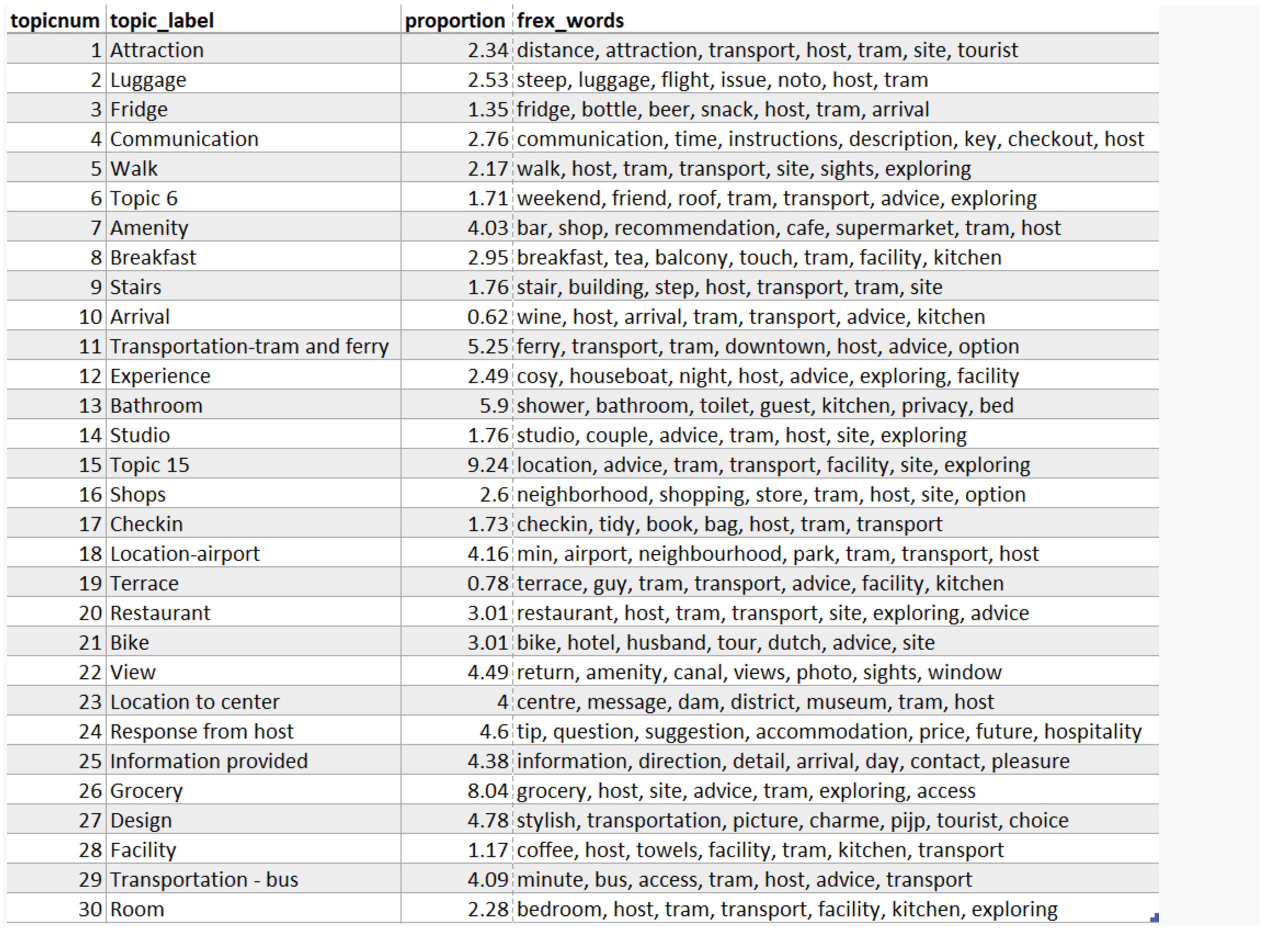
```r
set.seed(123)
airbnbfit <- stm(documents = out$documents,
                   vocab = out$vocab,
                   # Specifying the topic numbers
                   #K=0,
                   K = 30,
                   prevalence =~ price+review_scores_rating,
                   max.em.its = 75,
                   data = out$meta,
                   reportevery=5,
                   # gamma.prior = "L1",
                   sigma.prior = 0.7,
                   init.type = "Spectral")
#save(airbnbfit,file = "airbnbfit.rda")
#-----Finding optimal K

# K <- c(25,30,35)

#kr <- searchK(out$documents,out$vocab,K)

#plot(kr)

summary(airbnbfit)
```

4. Topic labels and Proportion table

After obtaining all topics, in order to make all topics more understandable, topic labels are set manually and conclude the main idea.

```r
topic_labels <- c("Attraction","Luggage",
                  "Fridge","Communication","Walk",
                  "Topic 6",
                  "Amenity","Breakfast","Stairs",
                  "Arrival","Transportation-tram and ferry",
                  "Experience",
                  "Bathroom",
                  "Studio","Topic 15",
                  "Shops",
                  "Checkin","Location-airport",
                "Terrace","Restaurant","Bike","View","Location to center","Re
sponse from host","Information provided","Grocery","Design","Facility","Trans
portation - bus","Room")

#-----Summarising the proportions of each topics based on theta

topic_proportions <- colMeans(airbnbfit$theta)
topic_summary <- summary(airbnbfit)
convergence <- as.data.frame(airbnbfit$theta)

colnames(convergence) <- topic_labels
table_towrite_labels <- data.frame()
for(i in 1:length(topic_summary$topicnums)){

   row_here <- tibble(topicnum= topic_summary$topicnums[i],
                      topic_label = topic_labels[i],
                      proportion = 100*round(topic_proportions[i],4),
                     frex_words = paste(topic_summary$frex[i,1:7],
                                        collapse = ", "))
   table_towrite_labels <- rbind(row_here,table_towrite_labels)
}
table_towrite_labels %>% arrange(topicnum)
```

With the proportion of each topics, we can find the most popular topic for Amsterdam and Melbourne are similar which are all about Location. But there are some little difference between them, which people in Melbourne care more about whether the listing is easy to find, while customers in Amsterdam pay attention to Location to public transportation.



topicnum	topic_label	proportion	frex_words
1	Attraction	2.34	distance, attraction, transport, host, tram, site, tourist
2	Luggage	2.53	steep, luggage, flight, issue, noto, host, tram
3	Fridge	1.35	fridge, bottle, beer, snack, host, tram, arrival
4	Communication	2.76	communication, time, instructions, description, key, checkout, host
5	Walk	2.17	walk, host, tram, transport, site, sights, exploring
6	Topic 6	1.71	weekend, friend, roof, tram, transport, advice, exploring
7	Amenity	4.03	bar, shop, recommendation, cafe, supermarket, tram, host
8	Breakfast	2.95	breakfast, tea, balcony, touch, tram, facility, kitchen
9	Stairs	1.76	stair, building, step, host, transport, tram, site
10	Arrival	0.62	wine, host, arrival, tram, transport, advice, kitchen
11	Transportation-tram and ferry	5.25	ferry, transport, tram, downtown, host, advice, option
12	Experience	2.49	cosy, houseboat, night, host, advice, exploring, facility
13	Bathroom	5.9	shower, bathroom, toilet, guest, kitchen, privacy, bed
14	Studio	1.76	studio, couple, advice, tram, host, site, exploring
15	Topic 15	9.24	location, advice, tram, transport, facility, site, exploring
16	Shops	2.6	neighborhood, shopping, store, tram, host, site, option
17	Checkin	1.73	checkin, tidy, book, bag, host, tram, transport
18	Location-airport	4.16	min, airport, neighbourhood, park, tram, transport, host
19	Terrace	0.78	terrace, guy, tram, transport, advice, facility, kitchen
20	Restaurant	3.01	restaurant, host, tram, transport, site, exploring, advice
21	Bike	3.01	bike, hotel, husband, tour, dutch, advice, site
22	View	4.49	return, amenity, canal, views, photo, sights, window
23	Location to center	4	centre, message, dam, district, museum, tram, host
24	Response from host	4.6	tip, question, suggestion, accommodation, price, future, hospitality
25	Information provided	4.38	information, direction, detail, arrival, day, contact, pleasure
26	Grocery	8.04	grocery, host, site, advice, tram, exploring, access
27	Design	4.78	stylish, transportation, picture, charme, pijp, tourist, choice
28	Facility	1.17	coffee, host, towels, facility, tram, kitchen, transport
29	Transportation - bus	4.09	minute, bus, access, tram, host, advice, transport
30	Room	2.28	bedroom, host, tram, transport, facility, kitchen, exploring

Figure 10 Topics and Proportion

5. STM solutions

The evaluation of stm solutions is also a significant part before further stages. In this part, topicQuality function evaluate the quality of solutions through exclusivity and semantic coherence.

```
quality <- topicQuality(airbnbfit,documents = out$documents)

   stm_solutions <- plot(airbnbfit,custom.labels = topic_labels,main = "")
              #save.image(stm_solutions,file = "Topic 30.img")
```

In the Figure below, it can be found that the performance of these topics are quite good, since most of topics have higher exclusivity and higher semantic coherence. The high exclusivity shows that words in topics are unique and the high semantic coherence demonstrates topics are also readable.

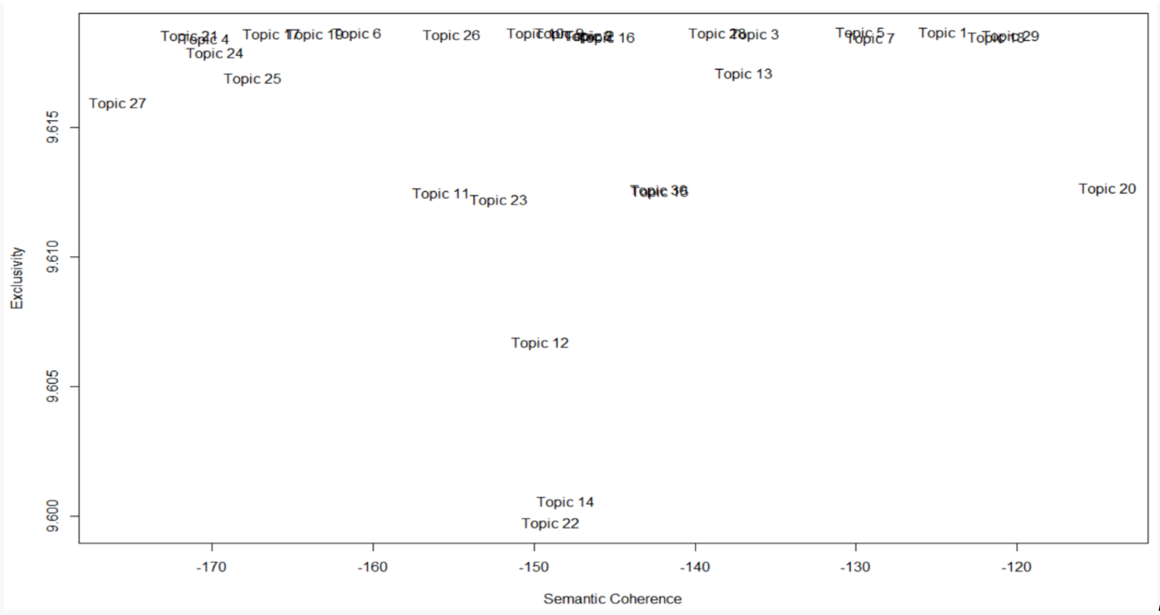


Figure 11 Quality of stm result

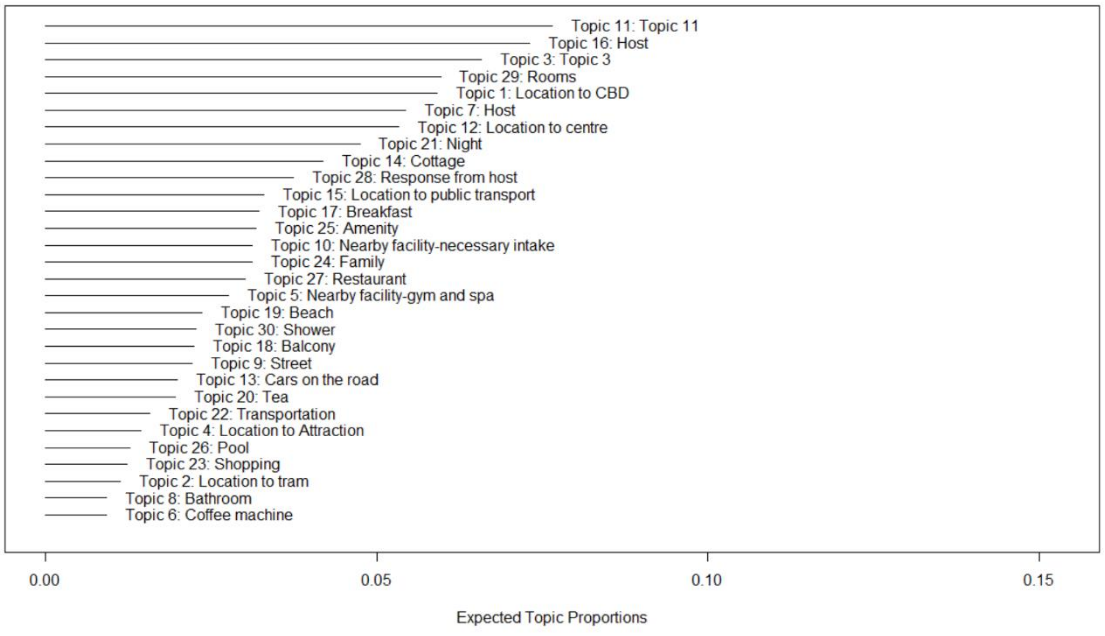


Figure 12 Plots of all topics

6. Estimate effects of variables price and review scores ratings on topics

In this stage, the effects of price and review scores ratings on topics are discussed.

```
effects <- estimateEffect(~review_scores_rating+price,
                          stmobj = airbnbfit,
                          metadata = out$meta)

#-----Ploting results of effects - review_score_rating
plot(effects, covariate = "review_scores_rating",
     topics = c(1:30),
     model = airbnbfit, method = "difference",
     cov.value1 = "100", cov.value2 = "0",
     xlab = "Low Rating ... High Rating",
     xlim = c(-0.004,0.004),
     main = "",
     custom.labels = topic_labels,
     labeltype = "custom")
```

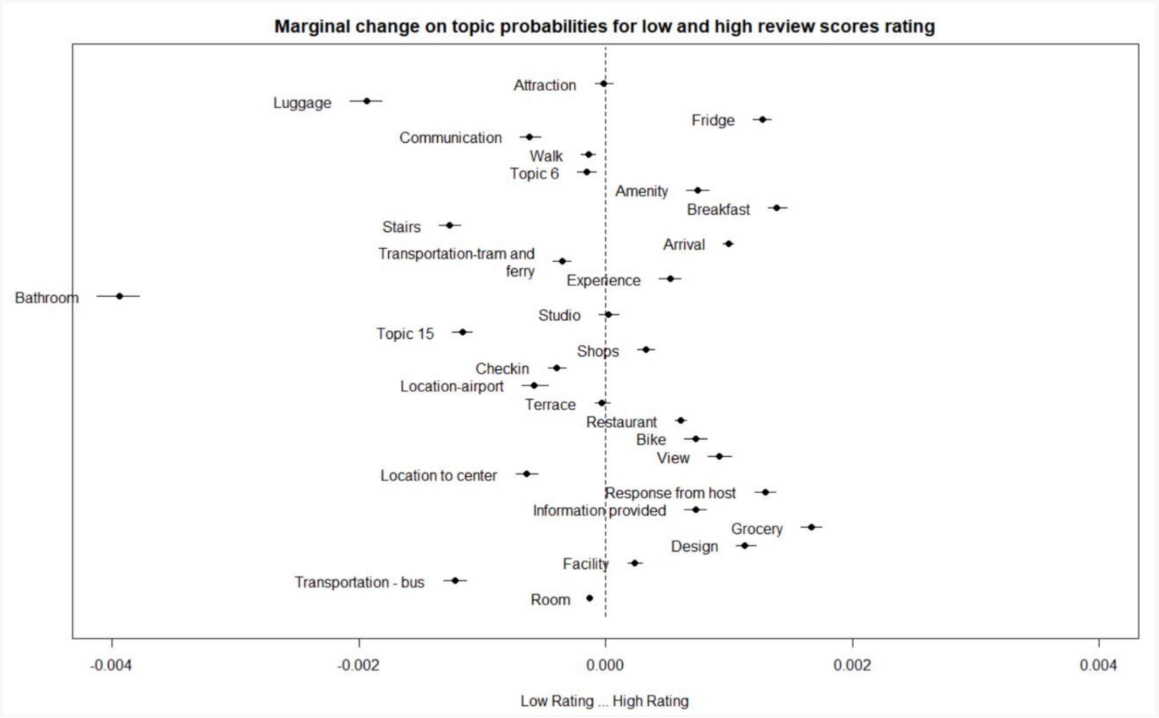


Figure 13 Effects of Rating on topics

In terms of Figure 4, we can find that with the increase of rating scores, customers will focus more on the service and facility the listing provided and experience people obtain. However, customers will complain more about transportations and locations for lower ratings.

```
#-----Ploting the result if effects - price. Becasue price is continuous vari
able, we plot the result for each topic
for(i in 1:length(topic_labels)){
plot(effects, covariate = "price",
     topics = i,
     model = airbnbfit, method = "continuous",
     # For this plotting we get the uper quantile
     # and low quantile of the price
     xlab = "Price",
     xlim = c(0,500),
     main = topic_labels[i],
     printlegend = FALSE,
     custom.labels =topic_labels[i],
     labeltype = "custom")
}

margin1 <- as.numeric(quantile(out$meta$price)[2])
margin2 <- as.numeric(quantile(out$meta$price)[4])

plot(effects, covariate = "price",
     topics = c(1:30),
     model = airbnbfit, method = "difference",
     cov.value1 = margin2, cov.value2 = margin1,
     xlab = "Low Price ... High Price",
     xlim = c(-0.004,0.004),
     main = "Marginal change on topic probabilities for low and high price",
     custom.labels =topic_labels,
     labeltype = "custom")
```

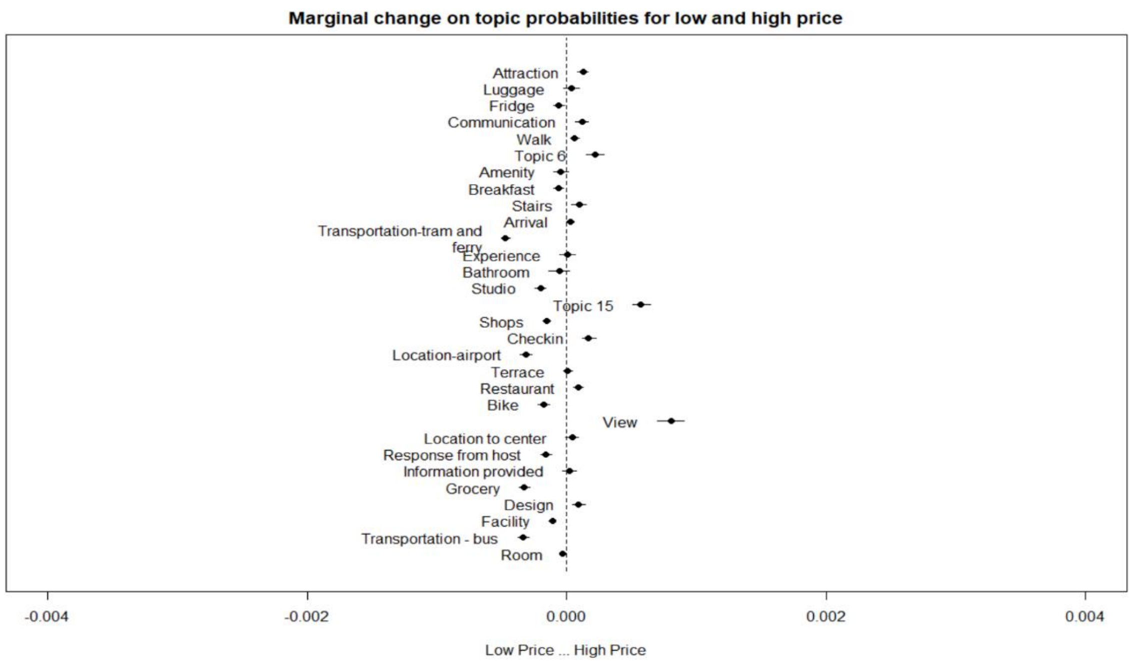


Figure 14 Effects of Price on Topics

Compared with the outcome of effects of review scores rating, the influence of price is not quite significant. However, some findings can be still concluded from the Figure 5. The higher price is, the more attention people will pay more to the view of the listing. The good facility like food in fridge and location of listings will lead to the growth of price, while the listing with lower price will be complained more about the transportations.

7. Effects of topics variables price and review scores ratings

In order to evaluate the predictability of adding topics on variables price and review scores ratings, linear regression models will be built. In this part, we will discuss from two aspects -the effects of all topics on price and review scores rating and the effects of some topics on price on these two variables separately.

7.1 Combine with calendar file

In this part, we will use the average price from the calendar file. The calculation has already don
e in part b.

```
load("~/group_TA/temp/final_df_monthly.rda")

for_regression <- cbind(out$meta,convergence) %>% na.omit()
for_regression$price <- as.numeric(for_regression$price)

#-----Combinig with calendar file in order to get average price per month
final_df_monthly <- final_df_monthly %>% select(listing_id,monthly_average_pr
ice,year_month.x) %>%
group_by(listing_id,year_month.x)%>%
  summarise(avg_price=mean(monthly_average_price))

for_regression <- for_regression %>% select(review_scores_rating,listing_id,r
eview_id,review_date,topic_labels)

for_regression <- left_join(for_regression,final_df_monthly) %>% na.omit()


#-----Converting the date object to the month year format
for_regression$month_year <- format(as.Date(for_regression$review_date),"%Y-%
m")
```

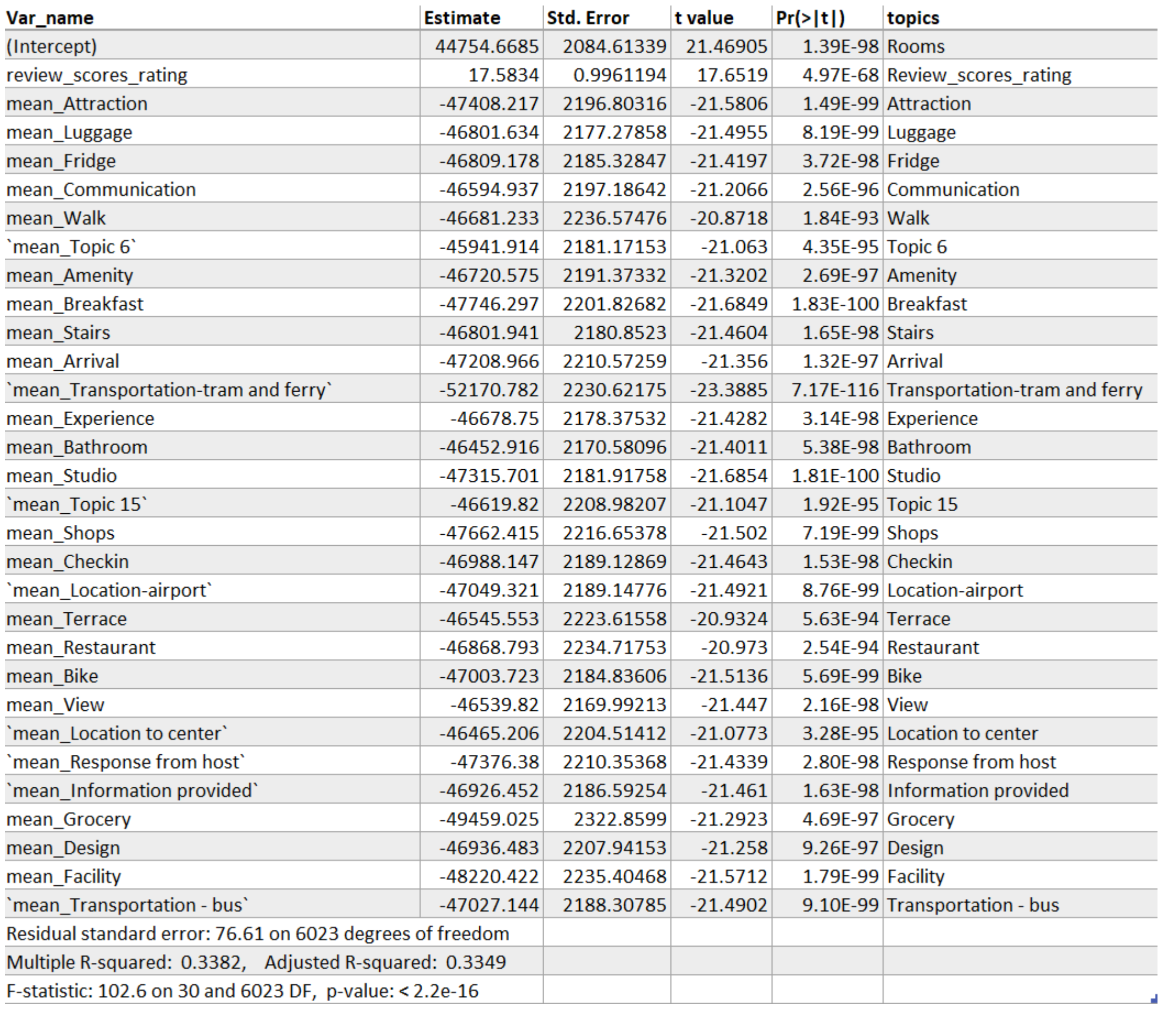
7.2 Create dataset for regression model

```
#-----Creating topic summary

regress_stm_topics <- for_regression %>% group_by(month_year,listing_id) %>%
      summarise_at(vars(topic_labels),mean,) %>% na.omit()
colnames <- paste0("mean_",topic_labels)

names(regress_stm_topics)[3:32]<- colnames

#-----Creating Sumaries of price and review score ratings
for_regression %>% group_by(month_year,listing_id)%>%
        summarise(avg_price= mean(avg_price),review_scores_rating= mean(revie
w_scores_rating)) -> regress_stm_scores_price

#-----Generating final file
regression_stm_all <- bind_cols(regress_stm_scores_price,regress_stm_topics)
rm(regress_stm_scores_price)
rm(regress_stm_topics)
regression_stm_all <-regression_stm_all[,-c(5:6)]
```

7.3 Exploring the effect of all of topics on price and review scores rating

```
#Price
summary(lm(avg_price~.,data = regression_stm_all[3:34]))

price_summary <- summary(lm(avg_price~.,data = regression_stm_all[3:34]))

estimation_price <- as.data.frame(price_summary$coefficients)

estimation_price$topics <- c("Rooms","Review_scores_rating",topic_labels[1:29
])

rm(price_summary)
```

The Figure 6 shows that all of topics are significant on estimating price and can contribute 33% of changes in price. Most of topics will negatively affect the price based on the value of estimation, while the review scores have a positive relationship with price that the higher review score is, the more expensive the listing is.

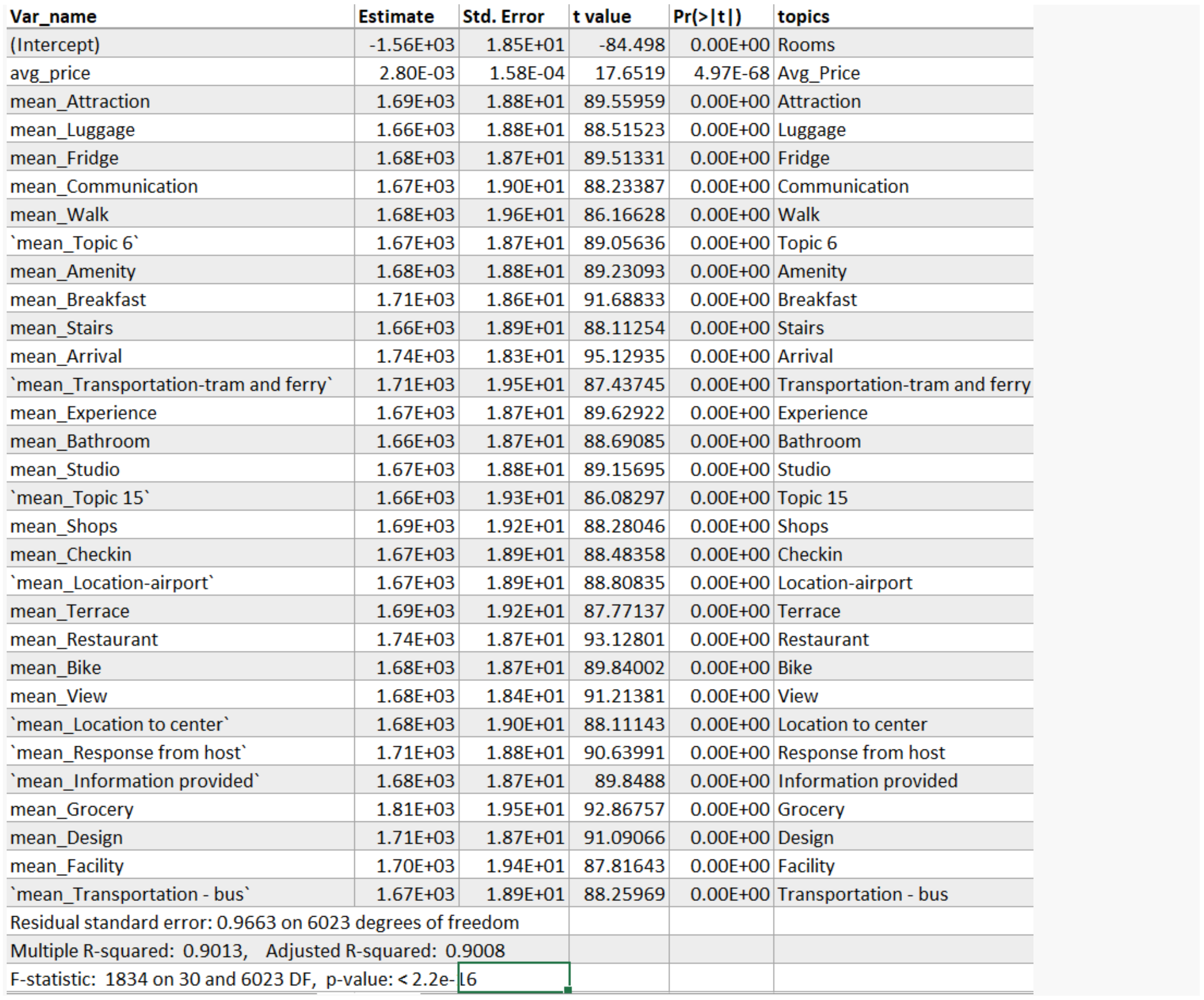


Var_name	Estimate	Std. Error	t value	Pr(>|t|)	topics
(Intercept)	44754.6685	2084.61339	21.46905	1.39E-98	Rooms
review_scores_rating	17.5834	0.9961194	17.6519	4.97E-68	Review_scores_rating
mean_Attraction	-47408.217	2196.80316	-21.5806	1.49E-99	Attraction
mean_Luggage	-46801.634	2177.27858	-21.4955	8.19E-99	Luggage
mean_Fridge	-46809.178	2185.32847	-21.4197	3.72E-98	Fridge
mean_Communication	-46594.937	2197.18642	-21.2066	2.56E-96	Communication
mean_Walk	-46681.233	2236.57476	-20.8718	1.84E-93	Walk
\`mean_Topic 6\`	-45941.914	2181.17153	-21.063	4.35E-95	Topic 6
mean_Amenity	-46720.575	2191.37332	-21.3202	2.69E-97	Amenity
mean_Breakfast	-47746.297	2201.82682	-21.6849	1.83E-100	Breakfast
mean_Stairs	-46801.941	2180.8523	-21.4604	1.65E-98	Stairs
mean_Arrival	-47208.966	2210.57259	-21.356	1.32E-97	Arrival
\`mean_Transportation-tram and ferry\`	-52170.782	2230.62175	-23.3885	7.17E-116	Transportation-tram and ferry
mean_Experience	-46678.75	2178.37532	-21.4282	3.14E-98	Experience
mean_Bathroom	-46452.916	2170.58096	-21.4011	5.38E-98	Bathroom
mean_Studio	-47315.701	2181.91758	-21.6854	1.81E-100	Studio
\`mean_Topic 15\`	-46619.82	2208.98207	-21.1047	1.92E-95	Topic 15
mean_Shops	-47662.415	2216.65378	-21.502	7.19E-99	Shops
mean_Checkin	-46988.147	2189.12869	-21.4643	1.53E-98	Checkin
\`mean_Location-airport\`	-47049.321	2189.14776	-21.4921	8.76E-99	Location-airport
mean_Terrace	-46545.553	2223.61558	-20.9324	5.63E-94	Terrace
mean_Restaurant	-46868.793	2234.71753	-20.973	2.54E-94	Restaurant
mean_Bike	-47003.723	2184.83606	-21.5136	5.69E-99	Bike
mean_View	-46539.82	2169.99213	-21.447	2.16E-98	View
\`mean_Location to center\`	-46465.206	2204.51412	-21.0773	3.28E-95	Location to center
\`mean_Response from host\`	-47376.38	2210.35368	-21.4339	2.80E-98	Response from host
\`mean_Information provided\`	-46926.452	2186.59254	-21.461	1.63E-98	Information provided
mean_Grocery	-49459.025	2322.8599	-21.2923	4.69E-97	Grocery
mean_Design	-46936.483	2207.94153	-21.258	9.26E-97	Design
mean_Facility	-48220.422	2235.40468	-21.5712	1.79E-99	Facility
\`mean_Transportation - bus\`	-47027.144	2188.30785	-21.4902	9.10E-99	Transportation - bus
Residual standard error: 76.61 on 6023 degrees of freedom					
Multiple R-squared: 0.3382, Adjusted R-squared: 0.3349					
F-statistic: 102.6 on 30 and 6023 DF, p-value: < 2.2e-16					

Figure 15 Estimation On Price (All)

```
#Review score rating
summary(lm(review_scores_rating~.,data = regression_stm_all[3:34]))
```

The chart below shows effects of all topics and price on review score rating. All of topics and avg_price are significant on estimating review scores rating and can contribute 90% of changes in review scores rating. Most of topics will positively affect the review scores based on the value of estimation, and the price have a strong positive relationship with review scores that the more expensive the listing is, the higher review score is in general.



Var_name	Estimate	Std. Error	t value	Pr(>|t|)	topics
(Intercept)	-1.56E+03	1.85E+01	-84.498	0.00E+00	Rooms
avg_price	2.80E-03	1.58E-04	17.6519	4.97E-68	Avg_Price
mean_Attraction	1.69E+03	1.88E+01	89.55959	0.00E+00	Attraction
mean_Luggage	1.66E+03	1.88E+01	88.51523	0.00E+00	Luggage
mean_Fridge	1.68E+03	1.87E+01	89.51331	0.00E+00	Fridge
mean_Communication	1.67E+03	1.90E+01	88.23387	0.00E+00	Communication
mean_Walk	1.68E+03	1.96E+01	86.16628	0.00E+00	Walk
`mean_Topic 6`	1.67E+03	1.87E+01	89.05636	0.00E+00	Topic 6
mean_Amenity	1.68E+03	1.88E+01	89.23093	0.00E+00	Amenity
mean_Breakfast	1.71E+03	1.86E+01	91.68833	0.00E+00	Breakfast
mean_Stairs	1.66E+03	1.89E+01	88.11254	0.00E+00	Stairs
mean_Arrival	1.74E+03	1.83E+01	95.12935	0.00E+00	Arrival
`mean_Transportation-tram and ferry`	1.71E+03	1.95E+01	87.43745	0.00E+00	Transportation-tram and ferry
mean_Experience	1.67E+03	1.87E+01	89.62922	0.00E+00	Experience
mean_Bathroom	1.66E+03	1.87E+01	88.69085	0.00E+00	Bathroom
mean_Studio	1.67E+03	1.88E+01	89.15695	0.00E+00	Studio
`mean_Topic 15`	1.66E+03	1.93E+01	86.08297	0.00E+00	Topic 15
mean_Shops	1.69E+03	1.92E+01	88.28046	0.00E+00	Shops
mean_Checkin	1.67E+03	1.89E+01	88.48358	0.00E+00	Checkin
`mean_Location-airport`	1.67E+03	1.89E+01	88.80835	0.00E+00	Location-airport
mean_Terrace	1.69E+03	1.92E+01	87.77137	0.00E+00	Terrace
mean_Restaurant	1.74E+03	1.87E+01	93.12801	0.00E+00	Restaurant
mean_Bike	1.68E+03	1.87E+01	89.84002	0.00E+00	Bike
mean_View	1.68E+03	1.84E+01	91.21381	0.00E+00	View
`mean_Location to center`	1.68E+03	1.90E+01	88.11143	0.00E+00	Location to center
`mean_Response from host`	1.71E+03	1.88E+01	90.63991	0.00E+00	Response from host
`mean_Information provided`	1.68E+03	1.87E+01	89.8488	0.00E+00	Information provided
mean_Grocery	1.81E+03	1.95E+01	92.86757	0.00E+00	Grocery
mean_Design	1.71E+03	1.87E+01	91.09066	0.00E+00	Design
mean_Facility	1.70E+03	1.94E+01	87.81643	0.00E+00	Facility
`mean_Transportation - bus`	1.67E+03	1.89E+01	88.25969	0.00E+00	Transportation - bus
Residual standard error: 0.9663 on 6023 degrees of freedom					
Multiple R-squared: 0.9013, Adjusted R-squared: 0.9008					
F-statistic: 1834 on 30 and 6023 DF, p-value: < 2.2e-16					

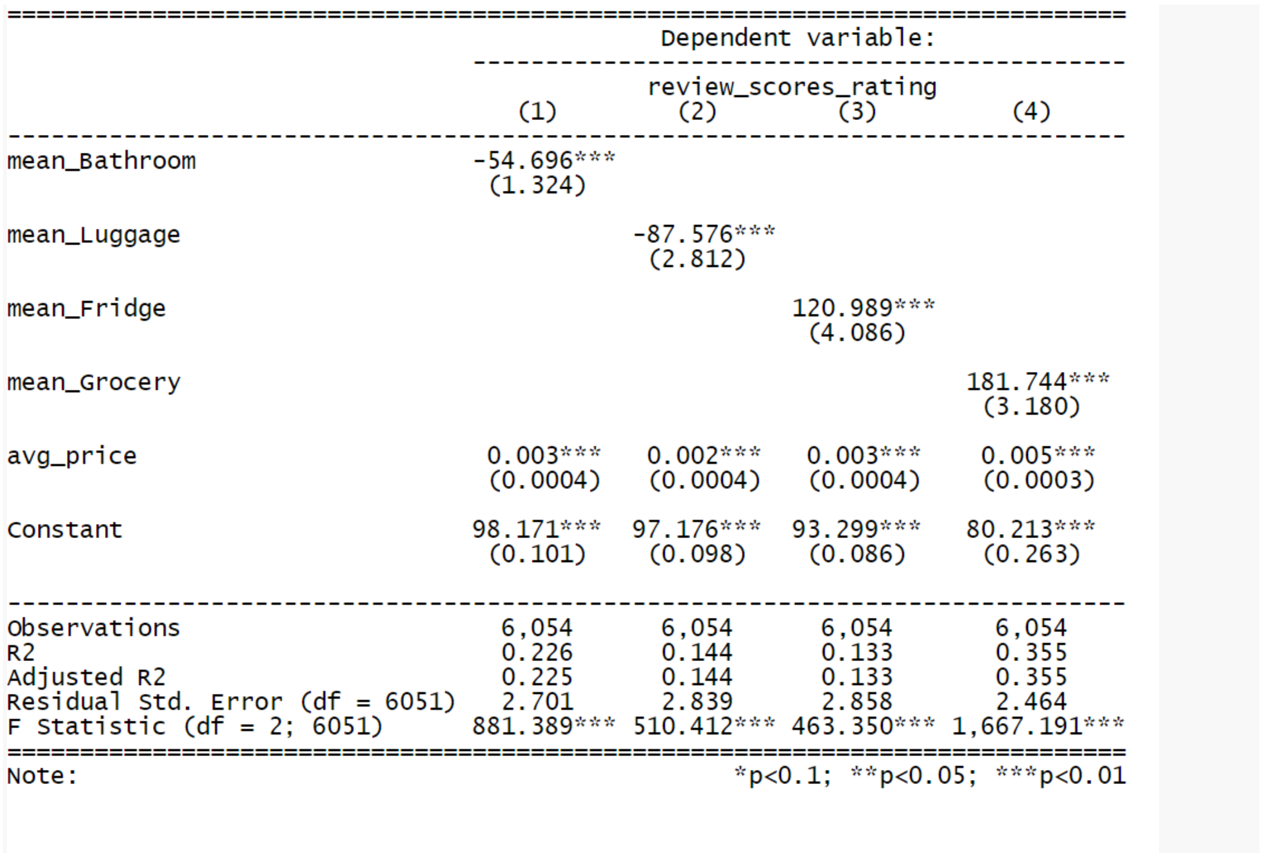
Figure 16 Estimation on Review Scores Rating (All)

7.4 Results of some topics on review scores rating

Based on the result of the estimated effects, we find review scores have influence on topic Bathroom, Luggage, Fridge and Grocery. Thus, we will check the effects of these topics on review scores separately.

```
#Review score rating
model1_review_score <- lm(review_scores_rating~mean_Bathroom+avg_price,data=r
egression_stm_all)
model2_review_score <- lm(review_scores_rating~mean_Luggage+avg_price,data=re
gression_stm_all)
model3_review_score <- lm(review_scores_rating~mean_Fridge+avg_price,data=reg
ression_stm_all)
model4_review_score <- lm(review_scores_rating~mean_Grocery+avg_price,data=re
gression_stm_all)
stargazer::stargazer(model1_review_score,model2_review_score,
                     model3_review_score,model4_review_score,
                     type = "text")
```

From figures below, we can summarize that variables are all significant on their predictors. Especially, variables for review scores have quite great impacts on the floating of review scores. For example, the R square of variable mean_Grocery R is 35.5% meaning that this variable can account for 35.5% change of the review scores. However, in terms of variables for estimating price, the contribution of most variables to change of price is quite small around 1% each. However, mean_Transportation-bus and ferry still significantly affect the predictor price, accounting for 24.8% of floating.

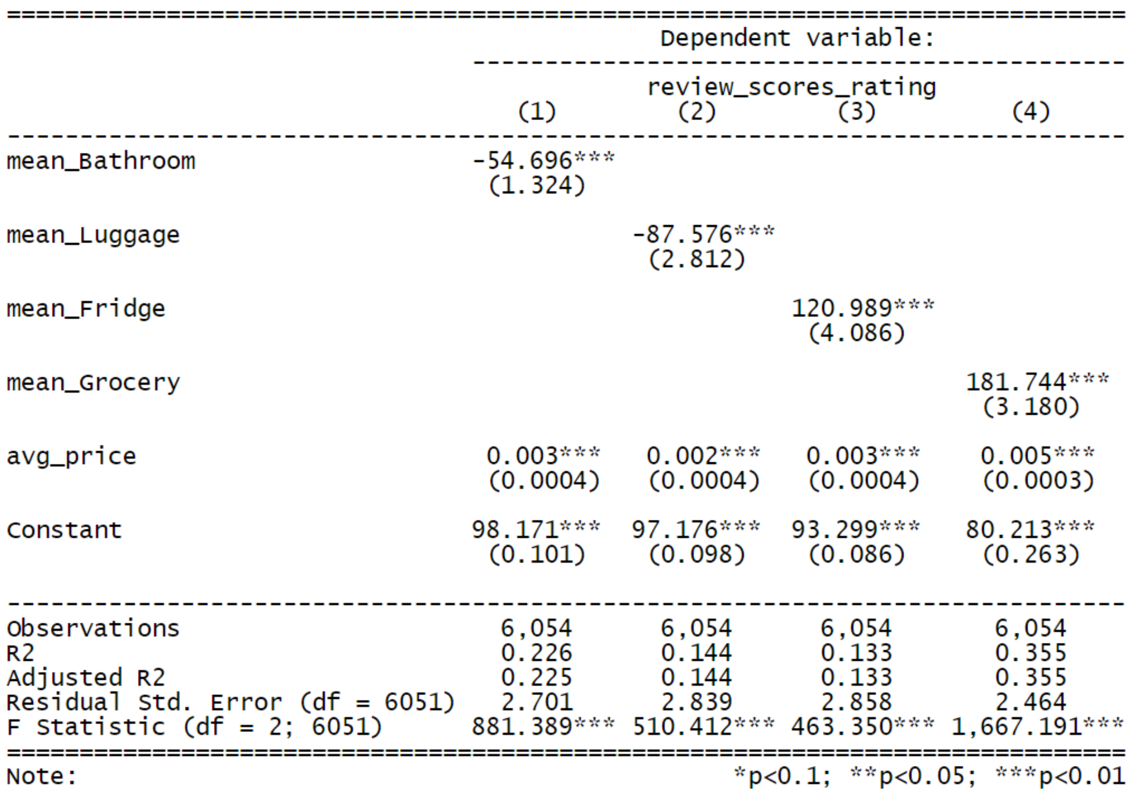


```
==========================================================================
                                         Dependent variable:
                              --------------------------------------------
                                         review_scores_rating
                                 (1)        (2)        (3)         (4)
--------------------------------------------------------------------------
mean_Bathroom                -54.696***
                              (1.324)

mean_Luggage                            -87.576***
                                         (2.812)

mean_Fridge                                        120.989***
                                                    (4.086)

mean_Grocery                                                   181.744***
                                                                (3.180)

avg_price                     0.003***   0.002***   0.003***    0.005***
                              (0.0004)   (0.0004)   (0.0004)    (0.0003)

Constant                     98.171***  97.176***  93.299***   80.213***
                              (0.101)    (0.098)    (0.086)     (0.263)

--------------------------------------------------------------------------
Observations                   6,054      6,054      6,054       6,054
R2                             0.226      0.144      0.133       0.355
Adjusted R2                    0.225      0.144      0.133       0.355
Residual Std. Error (df = 6051) 2.701     2.839      2.858       2.464
F Statistic (df = 2; 6051)   881.389*** 510.412*** 463.350*** 1,667.191***
==========================================================================
Note:                                         *p<0.1; **p<0.05; ***p<0.01
```

Figure 17 Estimation on Review Scores Rating (Some Topics)

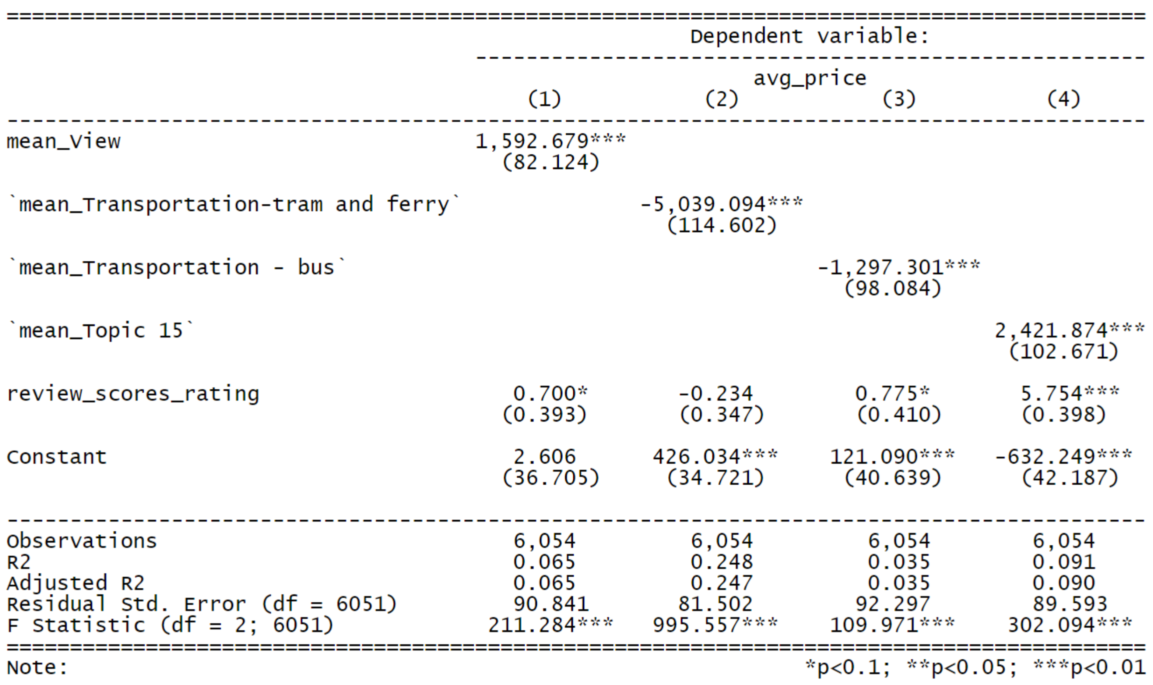
```
#Effects of some topics on price
model1_price <- lm(avg_price~mean_View+review_scores_rating,data=regression_s
tm_all)

model2_price <-
  lm(avg_price~`mean_Transportation-tram and ferry`+review_scores_rating,data
=regression_stm_all)

model3_price <-
```

```r
  lm(avg_price~`mean_Transportation - bus`+review_scores_rating,data=regressi
on_stm_all)

model4_price <-
  lm(avg_price~`mean_Topic 15`+review_scores_rating,data=regression_stm_all)
stargazer::stargazer(model1_price,model2_price,
                     model3_price,model4_price,
                     type = "text")
```



```
===========================================================================
                                         Dependent variable:
                             ----------------------------------------------
                                         review_scores_rating
                                (1)        (2)        (3)          (4)
---------------------------------------------------------------------------
mean_Bathroom                -54.696***
                              (1.324)

mean_Luggage                            -87.576***
                                         (2.812)

mean_Fridge                                        120.989***
                                                    (4.086)

mean_Grocery                                                    181.744***
                                                                 (3.180)

avg_price                     0.003***   0.002***   0.003***     0.005***
                             (0.0004)   (0.0004)   (0.0004)     (0.0003)

Constant                     98.171***  97.176***  93.299***    80.213***
                              (0.101)    (0.098)    (0.086)      (0.263)

---------------------------------------------------------------------------
Observations                   6,054      6,054      6,054        6,054
R2                             0.226      0.144      0.133        0.355
Adjusted R2                    0.225      0.144      0.133        0.355
Residual Std. Error (df = 6051) 2.701     2.839      2.858        2.464
F Statistic (df = 2; 6051)   881.389*** 510.412*** 463.350*** 1,667.191***
===========================================================================
Note:                                         *p<0.1; **p<0.05; ***p<0.01
```

Figure 18 Estimation on Review Scores Rating (Some Topics)



```
==========================================================================================
                                                   Dependent variable:
                                    ------------------------------------------------------
                                                        avg_price
                                         (1)           (2)           (3)           (4)
------------------------------------------------------------------------------------------
mean_View                           1,592.679***
                                      (82.124)

`mean_Transportation-tram and ferry`             -5,039.094***
                                                   (114.602)

`mean_Transportation - bus`                                   -1,297.301***
                                                                 (98.084)

`mean_Topic 15`                                                             2,421.874***
                                                                              (102.671)

review_scores_rating                  0.700*        -0.234        0.775*       5.754***
                                     (0.393)       (0.347)       (0.410)       (0.398)

Constant                              2.606       426.034***    121.090***   -632.249***
                                     (36.705)      (34.721)      (40.639)      (42.187)

------------------------------------------------------------------------------------------
Observations                          6,054         6,054         6,054         6,054
R2                                    0.065         0.248         0.035         0.091
Adjusted R2                           0.065         0.247         0.035         0.090
Residual Std. Error (df = 6051)       90.841        81.502        92.297        89.593
F Statistic (df = 2; 6051)          211.284***    995.557***    109.971***    302.094***
==========================================================================================
Note:                                                        *p<0.1; **p<0.05; ***p<0.01
```

Figure 19 Estimation on Price (Some Topics)

Reference

Minitab Editor, 2013. *Multiple Regression Analysis: Use Adjusted R-Squared And Predicted R-Squared To Include The Correct Number Of Variables*. [online] Blog.minitab.com. Available at: <https://blog.minitab.com/blog/adventures-in-statistics-2/multiple-regession-analysis-use-adjusted-r-squared-and-predicted-r-squared-to-include-the-correct-number-of-variables> [Accessed 26 March 2020].

Martin, F. and Johnson,M. (2012) More Efficient Topic Modelling Through a Noun Only Approach, Computer Science, Available at: https://www.aclweb.org/anthology/U15-1013.pdf (Access 20/03/2019)