Yelp-Project-v8.R

Gratton

2020-05-29

#Step 2

Define working directory of where the data files reside.  
setwd("c:/users/gratton/documents/Darcy/Ryerson/Capstone (CKME136)/Yelp Data")  
  
  
# "readr" is used to read in the records from the json data files.  
#install.packages("readr")  
library(readr)  
  
# "stringr" is used to combine the data from each file into a single json string for processing.  
#install.packages("stringr")  
library(stringr)  
  
# "jsonlite" is used to parse the json string.  
#install(jsonlite)  
library(jsonlite)  
  
# "dplyr" is needed by "jasonlite" to enable a tribble dataframe.  
#install.packages("dplyr")  
library(dplyr)

##   
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':  
##   
## filter, lag

## The following objects are masked from 'package:base':  
##   
## intersect, setdiff, setequal, union

#"textdata" is needed to obtain the AFINN lexicon used in the sentiment analysis.  
#install.packages("textdata")  
library(textdata)  
  
#"tidytext" is used to undertake the sentiment analysis.  
#install.packages("tidytext")  
library(tidytext)  
  
# "n\_max = 2,500,000" is used with the reviews dataset because of memory limitations but dataset will be  
# large enough for this analysis.  
lines <- read\_lines("yelp\_academic\_dataset\_review.json", n\_max = 2500000, progress = FALSE)  
  
# Each line in "review\_lines" is combined into a single JSON string to optimize processing.  
combined <- str\_c("[", str\_c(lines, collapse = ", "), "]")  
  
# The single JSON string is parsed and stored as a tribble dataframe, "flatten" is used to extract any  
# nested JSON data, "%>%" is a pipe function to cycle through all records.  
reviews <- fromJSON(combined) %>%  
 flatten() %>%  
 tbl\_df()  
  
# Same process is used to load the business JSON file, although the whole file is read.  
lines <- read\_lines("yelp\_academic\_dataset\_business.json", progress = FALSE)  
combined <- str\_c("[", str\_c(lines, collapse = ", "), "]")  
business <- fromJSON(combined) %>%  
 flatten() %>%  
 tbl\_df()

# Remove spaces in business$categories to prepare for word frequency analysis.  
business$categories <- gsub('\\s+', '', business$categories)  
  
# Word frequency in categories to determine what is the best way to capture businesses that are restaurants  
# by using the table function to separate each phrase split by a comma and start for most frequent words  
# what is found is "restaurants" will provide the most businesses.  
category\_freq\_work <- data.frame(table(unlist(strsplit(tolower(business$categories), ","))))  
head(category\_freq\_work[order(-category\_freq\_work$Freq),],30)

## Var1 Freq  
## 999 restaurants 59371  
## 1058 shopping 31878  
## 463 food 29989  
## 583 homeservices 19729  
## 117 beauty&spas 19370  
## 547 health&medical 17171  
## 707 localservices 13932  
## 81 automotive 13203  
## 804 nightlife 13095  
## 105 bars 11341  
## 417 eventplanning&services 10371  
## 7 activelife 9521  
## 431 fashion 7798  
## 1025 sandwiches 7332  
## 262 coffee&tea 7321  
## 432 fastfood 7257  
## 33 american(traditional) 7107  
## 532 hairsalons 6955  
## 911 pizza 6804  
## 569 home&garden 6489  
## 60 arts&entertainment 6304  
## 949 professionalservices 6276  
## 83 autorepair 6140  
## 602 hotels&travel 6033  
## 357 doctors 5867  
## 980 realestate 5677  
## 177 burgers 5404  
## 165 breakfast&brunch 5381  
## 792 nailsalons 5043  
## 1098 specialtyfood 4883

# Next is to refine the business dataframe to only include rows that have "restaurants"  
# in the category field.  
business <- dplyr::filter(business, grepl('Restaurant', categories))  
  
# Step 3

# To determine which cuisine to use, I will pick the top 10 most popular cuisines.  
category\_freq\_work <- data.frame(table(unlist(strsplit(tolower(business$categories), ","))))  
head(category\_freq\_work[order(-category\_freq\_work$Freq),],30)

## Var1 Freq  
## 582 restaurants 59371  
## 274 food 14804  
## 476 nightlife 8562  
## 67 bars 8182  
## 595 sandwiches 7332  
## 259 fastfood 7257  
## 23 american(traditional) 7107  
## 537 pizza 6804  
## 113 burgers 5404  
## 104 breakfast&brunch 5381  
## 22 american(new) 4882  
## 381 italian 4716  
## 448 mexican 4618  
## 149 chinese 4428  
## 163 coffee&tea 3647  
## 119 cafes 3232  
## 384 japanese 2716  
## 148 chickenwings 2705  
## 250 eventplanning&services 2685  
## 593 salad 2531  
## 600 seafood 2508  
## 659 sushibars 2258  
## 638 specialtyfood 2093  
## 206 delis 1955  
## 46 asianfusion 1953  
## 125 canadian(new) 1909  
## 445 mediterranean 1834  
## 60 bakeries 1833  
## 141 caterers 1829  
## 64 barbeque 1814

# Business dataframe is filter to include only the 10 most popular cuisines.  
business <- dplyr::filter(business, grepl('american\\(traditional\\)|american\\(new\\)|italian|mexican|chinese|japanese|sushibars|asianfusion|canadian\\(new\\)|mediterranean', categories, ignore.case = TRUE))  
  
# Create a new column in business called "cuisine" as it will be used for analysis.  
business$cuisine <- ifelse(grepl("american\\(traditional\\)", business$categories, ignore.case = T), "american(traditional)",   
 ifelse(grepl("american\\(new\\)", business$categories, ignore.case = T), "american(new)",  
 ifelse(grepl("italian", business$categories, ignore.case = T), "italian",  
 ifelse(grepl("mexican", business$categories, ignore.case = T), "mexican",  
 ifelse(grepl("chinese", business$categories, ignore.case = T), "chinese",  
 ifelse(grepl("japanese", business$categories, ignore.case = T), "japanese",  
 ifelse(grepl("sushibars", business$categories, ignore.case = T), "sushibars",  
 ifelse(grepl("asianfusion", business$categories, ignore.case = T), "asianfusion",  
 ifelse(grepl("canadian\\(new\\)", business$categories, ignore.case = T), "canadian(new)", "mediterranean")))))))))  
  
# Step 4

# To determine which cities to use, I will pick the top 10 most popular cities and filter to only  
# include businesses in those cities. Montreal was not included because KNIT kept rejecting the french "e".  
category\_freq\_work <- data.frame(table(unlist(strsplit(tolower(business$city), ","))))  
head(category\_freq\_work[order(-category\_freq\_work$Freq),],30)

## Var1 Freq  
## 226 las vegas 3599  
## 525 toronto 3564  
## 385 phoenix 2269  
## 303 montréal 1432  
## 83 charlotte 1431  
## 390 pittsburgh 1273  
## 68 calgary 1229  
## 470 scottsdale 964  
## 91 cleveland 732  
## 280 mesa 660  
## 291 mississauga 656  
## 246 madison 600  
## 518 tempe 547  
## 253 markham 517  
## 186 henderson 504  
## 81 chandler 463  
## 170 glendale 394  
## 168 gilbert 306  
## 408 richmond hill 294  
## 466 scarborough 241  
## 542 vaughan 227  
## 79 champaign 223  
## 380 peoria 212  
## 52 brampton 201  
## 348 north york 195  
## 96 concord 193  
## 340 north las vegas 185  
## 147 etobicoke 138  
## 513 surprise 130  
## 229 laval 126

# business <- dplyr::filter(business, grepl('las vegas|toronto|phoenix|montreal|charlotte|pittsburgh|calgary|scottsdale|cleveland|mesa', business$city, ignore.case = TRUE))  
business <- dplyr::filter(business, grepl('las vegas|toronto|phoenix|charlotte|pittsburgh|calgary|scottsdale|cleveland|mesa', business$city, ignore.case = TRUE))  
business$city <- trimws(business$city)  
  
# Some cities names need to be consolidated.  
business$city[which(business$city=="charlotte")]<-"Charlotte"  
business$city[which(business$city=="East Cleveland")]<-"Cleveland"  
business$city[which(business$city=="Cleveland Heights")]<-"Cleveland"  
business$city[which(business$city=="North Las Vegas")]<-"Las Vegas"  
business$city[which(business$city=="N. Las Vegas")]<-"Las Vegas"  
business$city[which(business$city=="N Las Vegas")]<-"Las Vegas"  
business$city[which(business$city=="South Las Vegas")]<-"Las Vegas"  
business$city[which(business$city=="East Pittsburgh")]<-"Pittsburgh"  
business$city[which(business$city=="North Toronto")]<-"Toronto"  
  
# Filter the reviews dataframe to exclude any businesses that are not in the refined business  
# dataframe based on business id.  
reviews <- reviews %>%  
 filter(business\_id %in% business$business\_id)  
  
# Based on business id, populate the reviews dataframe with the city and cuisine values from the business  
# dataframe.  
reviews$city <- business$city[match(reviews$business\_id, business$business\_id)]  
reviews$cuisine <- business$cuisine[match(reviews$business\_id, business$business\_id)]  
  
# Remove columns not needed in reviews.  
drops <- c("useful","funny", "cool", "date")  
reviews <- reviews[ , !(names(reviews) %in% drops)]  
  
# Display dataset size by city and cuisine.  
with(reviews, table(city, cuisine))

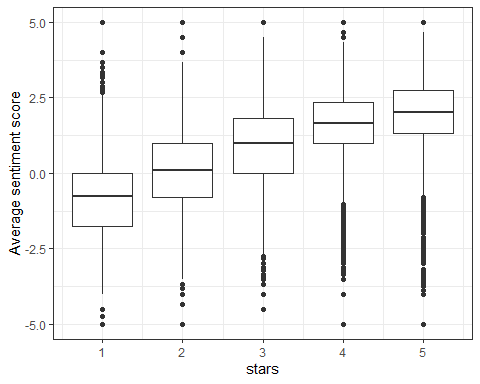
## cuisine  
## city american(new) american(traditional) asianfusion canadian(new)  
## Calgary 2215 1472 191 1335  
## Charlotte 13060 15413 1173 0  
## Cleveland 5255 5441 277 0  
## Las Vegas 69543 96317 5604 0  
## Mesa 2746 6892 288 0  
## Phoenix 20277 33099 1082 0  
## Pittsburgh 12358 10925 888 0  
## Scottsdale 20405 22563 989 0  
## Toronto 8081 7537 3915 8380  
## cuisine  
## city chinese italian japanese mediterranean mexican sushibars  
## Calgary 1644 1057 2031 758 356 152  
## Charlotte 3141 5149 4169 2380 7244 4070  
## Cleveland 1236 1776 1017 1025 3348 491  
## Las Vegas 27887 33195 39589 6599 36036 3691  
## Mesa 2062 1777 1518 857 3950 79  
## Phoenix 5934 11626 6389 3228 22861 707  
## Pittsburgh 2984 3927 1680 955 2988 111  
## Scottsdale 1857 10927 3981 690 8750 884  
## Toronto 10117 12059 13656 4102 6451 1734

# As can been seen, there are many reviews for each city and cuisine. Of course, American cities to  
# not have any Canadian cuisine reviews.  
  
# Step 5

# Start sentiment analysis  
  
# To analyze using the tidytext framework, we need to use the unnest\_tokens function.  
# This will create a record for each word in the review comments field.  
review\_words <- reviews %>%  
 select(review\_id, user\_id, business\_id, stars, text, city, cuisine) %>%  
 unnest\_tokens(word, text) %>%  
 filter(!word %in% stop\_words$word,  
 str\_detect(word, "^[a-z']+$"))  
  
# We will use the AFINN lexicon for our sentiment analysis. It provides a list of negative and  
# positive works and assigns a value ranging from -5 to +5. The lower the score the more negative  
# the sentiment of the word.  
AFINN <- get\_sentiments("afinn")  
  
# Step 6

# Using an inner\_join by using the word field in AFINN, we then group\_by review\_id to build a  
# sentiment score of the review based on the mean AFINN value of each review  
reviews\_sentiment <- review\_words %>%  
 inner\_join(AFINN, by = "word") %>%  
 group\_by(review\_id, city, cuisine, stars) %>%  
 summarize(sentiment = mean(value))  
  
# Step 7

# A box plot is created to check to visually inspect if there is a relationship between mean  
# sentiment value and the numbers of stars. Conclusion is that there is a positive correlation  
# between the two attributes but there are also some failures, example some of the 5 star ratings  
# have a negative sentiment value.  
library(ggplot2)  
theme\_set(theme\_bw())  
ggplot(reviews\_sentiment, aes(stars, sentiment, group = stars)) +  
 geom\_boxplot() +  
 ylab("Average sentiment score")



# To determine if the means of each star category is statistically significant, we perform an ANOVA  
# test since we have more than two categories to compare to each other. Before we perform the  
# test we must confirm all ANOVA assumptions have been met.  
  
# Assumption 1: All samples are independent and there are more than 2 categorical groups.  
# Assumption 1 is met since the reviews were randomly selected and we have 5 (stars)  
# categorical groups.  
  
# Assumption 2: Dependant variable (average sentiment score) is continuous  
# Assumption 2 is met since the average sentiment score per review between -5 to +5  
  
# Assumption 3: Each category has a normal distribution of sentiment scores and there are  
# no major outliers.  
  
# "tidyr" is required to filter by stars to get the sentiment during the testing of the   
# normalized distribution.  
# install.packages("tidyr")  
library(tidyr)  
  
# Extract the sentiment values by each category value in stars  
star1 <- reviews\_sentiment %>%   
 dplyr::filter(stars == 1) %>%  
 dplyr::ungroup() %>%  
 dplyr::select(sentiment)  
star1 <- as.numeric(unlist(star1))  
  
star2 <- reviews\_sentiment %>%   
 dplyr::filter(stars == 2) %>%  
 dplyr::ungroup() %>%  
 dplyr::select(sentiment)  
star2 <- as.numeric(unlist(star2))  
  
star3 <- reviews\_sentiment %>%   
 dplyr::filter(stars == 3) %>%  
 dplyr::ungroup() %>%  
 dplyr::select(sentiment)  
star3 <- as.numeric(unlist(star3))  
  
star4 <- reviews\_sentiment %>%   
 dplyr::filter(stars == 4) %>%  
 dplyr::ungroup() %>%  
 dplyr::select(sentiment)  
star4 <- as.numeric(unlist(star4))  
  
star5 <- reviews\_sentiment %>%   
 dplyr::filter(stars == 5) %>%  
 dplyr::ungroup() %>%  
 dplyr::select(sentiment)  
star5 <- as.numeric(unlist(star5))  
  
# "nortest" is required to run the Anderson-Darling Test to test for normalacity, Shaprio test  
# could not be used because sample size >5,000.  
#install.packages("nortest")  
library(nortest)  
ad.test(star1)

##   
## Anderson-Darling normality test  
##   
## data: star1  
## A = 236.54, p-value < 2.2e-16

ad.test(star2)

##   
## Anderson-Darling normality test  
##   
## data: star2  
## A = 89.198, p-value < 2.2e-16

ad.test(star3)

##   
## Anderson-Darling normality test  
##   
## data: star3  
## A = 432.46, p-value < 2.2e-16

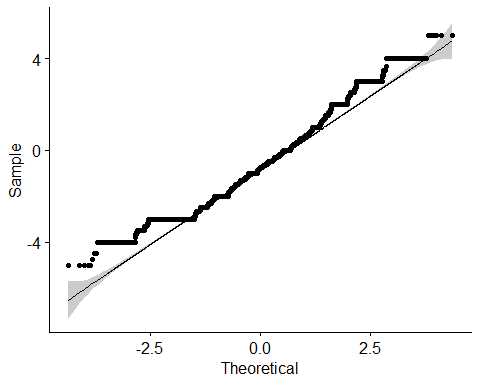
ad.test(star4)

##   
## Anderson-Darling normality test  
##   
## data: star4  
## A = 1418.7, p-value < 2.2e-16

ad.test(star5)

##   
## Anderson-Darling normality test  
##   
## data: star5  
## A = 2837.4, p-value < 2.2e-16

# Since all p values are less than 0.05, we cannot assume the data is normally distributed.   
# Hence, an ANOVA test cannot be used and we will need to use the Kruskal-Wallis test  
  
# "ggpubr" is used to plot a quintile plot to determine if data is normally distributed.  
# This is an additional test to determine normalacity.  
# install.packages("ggpubr")  
library(ggpubr)  
ggqqplot(star1)



# With this plot it can be seen that star1 is not normally distributed because Quantile-Quantile plot  
# does not follow the straight line and there are many values that do not fall on the straight line.   
  
# Assumption 4: Homogeneity of variances  
# "carData" is needed to run the Levene Test to determine homogeneity.  
# install.packages("carData")  
library(car)

## Loading required package: carData

##   
## Attaching package: 'car'

## The following object is masked from 'package:dplyr':  
##   
## recode

reviews\_sentiment$stars <- as.factor(reviews\_sentiment$stars)  
leveneTest(sentiment ~ stars, data = reviews\_sentiment, center = mean)

## Levene's Test for Homogeneity of Variance (center = mean)  
## Df F value Pr(>F)   
## group 4 2365.8 < 2.2e-16 \*\*\*  
## 676084   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

# Since Pr(>F) is less than 0.05, we reject the H0 and conclude there is no homogeneity.  
# Although homogeneity is not met, since the sample size is very large we will continue with a   
# Kruskal Wallis test.  
  
# The Kruskal Wallis test will determine if the mean of the sentiment of each star ratings are different.  
kruskal.test(list(star1,star2,star3,star4,star5))

##   
## Kruskal-Wallis rank sum test  
##   
## data: list(star1, star2, star3, star4, star5)  
## Kruskal-Wallis chi-squared = 213097, df = 4, p-value < 2.2e-16

# Since the p-value is less than 0.05 we can reject H0 and conclude the at least one of the means are  
# different.  
  
# A Dunn test will determine which specific means are not equal.  
# install.packages("dunn.test")  
# install.packages("FSA")  
library(dunn.test)  
library(FSA)

## ## FSA v0.8.30. See citation('FSA') if used in publication.  
## ## Run fishR() for related website and fishR('IFAR') for related book.

##   
## Attaching package: 'FSA'

## The following object is masked from 'package:car':  
##   
## bootCase

dunnTest(sentiment ~ stars, data = reviews\_sentiment, method="bonferroni")

## Dunn (1964) Kruskal-Wallis multiple comparison

## p-values adjusted with the Bonferroni method.

## Comparison Z P.unadj P.adj  
## 1 1 - 2 -68.89416 0 0  
## 2 1 - 3 -163.31212 0 0  
## 3 2 - 3 -83.47720 0 0  
## 4 1 - 4 -297.09957 0 0  
## 5 2 - 4 -197.80459 0 0  
## 6 3 - 4 -118.22807 0 0  
## 7 1 - 5 -394.69854 0 0  
## 8 2 - 5 -282.46743 0 0  
## 9 3 - 5 -212.11815 0 0  
## 10 4 - 5 -109.00119 0 0

# Since all of the p values are 0 we can conclude that all of the means are different from each other.  
# Hence the sentiment values increase, so does the star ratings. Therefore, sentiment values can be used  
# as an indicator of the expected star rating.  
  
# Step 8

# We will now determine if the mean sentiment value for each cuisine is different in each city.  
# First, we create a dataset for each cuisine.  
amer\_new <- reviews\_sentiment %>%   
 dplyr::filter(cuisine == "american(new)") %>%  
 dplyr::ungroup() %>%  
 dplyr::select(sentiment)  
amer\_new <- as.numeric(unlist(amer\_new))  
  
amer\_trad <- reviews\_sentiment %>%   
 dplyr::filter(cuisine == "american(traditional)") %>%  
 dplyr::ungroup() %>%  
 dplyr::select(sentiment)  
amer\_trad <- as.numeric(unlist(amer\_trad))  
  
asianfusion <- reviews\_sentiment %>%   
 dplyr::filter(cuisine == "asianfusion") %>%  
 dplyr::ungroup() %>%  
 dplyr::select(sentiment)  
asianfusion <- as.numeric(unlist(asianfusion))  
  
can\_new <- reviews\_sentiment %>%   
 dplyr::filter(cuisine == "canadian(new)") %>%  
 dplyr::ungroup() %>%  
 dplyr::select(sentiment)  
can\_new <- as.numeric(unlist(can\_new))  
  
chinese <- reviews\_sentiment %>%   
 dplyr::filter(cuisine == "chinese") %>%  
 dplyr::ungroup() %>%  
 dplyr::select(sentiment)  
chinese <- as.numeric(unlist(chinese))  
  
italian <- reviews\_sentiment %>%   
 dplyr::filter(cuisine == "italian") %>%  
 dplyr::ungroup() %>%  
 dplyr::select(sentiment)  
italian <- as.numeric(unlist(italian))  
  
japanese <- reviews\_sentiment %>%   
 dplyr::filter(cuisine == "japanese") %>%  
 dplyr::ungroup() %>%  
 dplyr::select(sentiment)  
japanese <- as.numeric(unlist(japanese))  
  
mediterranean <- reviews\_sentiment %>%   
 dplyr::filter(cuisine == "mediterranean") %>%  
 dplyr::ungroup() %>%  
 dplyr::select(sentiment)  
mediterranean <- as.numeric(unlist(mediterranean))  
  
mexican <- reviews\_sentiment %>%   
 dplyr::filter(cuisine == "mexican") %>%  
 dplyr::ungroup() %>%  
 dplyr::select(sentiment)  
mexican <- as.numeric(unlist(mexican))  
  
# The Anderson-Darling Test is used to test for normalacity.  
ad.test(amer\_new)

##   
## Anderson-Darling normality test  
##   
## data: amer\_new  
## A = 1559.2, p-value < 2.2e-16

ad.test(amer\_trad)

##   
## Anderson-Darling normality test  
##   
## data: amer\_trad  
## A = 1877.2, p-value < 2.2e-16

ad.test(asianfusion)

##   
## Anderson-Darling normality test  
##   
## data: asianfusion  
## A = 163.37, p-value < 2.2e-16

ad.test(can\_new)

##   
## Anderson-Darling normality test  
##   
## data: can\_new  
## A = 103.43, p-value < 2.2e-16

ad.test(chinese)

##   
## Anderson-Darling normality test  
##   
## data: chinese  
## A = 552.84, p-value < 2.2e-16

ad.test(italian)

##   
## Anderson-Darling normality test  
##   
## data: italian  
## A = 985.23, p-value < 2.2e-16

ad.test(japanese)

##   
## Anderson-Darling normality test  
##   
## data: japanese  
## A = 844.59, p-value < 2.2e-16

ad.test(mediterranean)

##   
## Anderson-Darling normality test  
##   
## data: mediterranean  
## A = 329.84, p-value < 2.2e-16

ad.test(mexican)

##   
## Anderson-Darling normality test  
##   
## data: mexican  
## A = 926.5, p-value < 2.2e-16

# Since all p values are less than 0.05, we cannot assume the data is normally distributed.   
# Hence, an ANOVA test cannot be used, and we will need to use the Kruskal-Wallis test.  
  
# The Kruskal Wallis test will determine if the mean of the sentiment of each star ratings are different.  
kruskal.test(list(amer\_new,amer\_trad,asianfusion,can\_new,chinese,italian,japanese,mediterranean,mexican))

##   
## Kruskal-Wallis rank sum test  
##   
## data: list(amer\_new, amer\_trad, asianfusion, can\_new, chinese, italian, japanese, mediterranean, mexican)  
## Kruskal-Wallis chi-squared = 3868.4, df = 8, p-value < 2.2e-16

# Since the p-value is less than 0.05 we can reject H0 and conclude the at least one of the means are  
# different.  
  
# A Dunn test will determine which specific means are not equal.  
dunnTest(sentiment ~ city, data = reviews\_sentiment, method = "holm")

## Warning: city was coerced to a factor.

## Dunn (1964) Kruskal-Wallis multiple comparison

## p-values adjusted with the Holm method.

## Comparison Z P.unadj P.adj  
## 1 Calgary - Charlotte -5.21848760 1.803901e-07 2.705851e-06  
## 2 Calgary - Cleveland -2.02883230 4.247537e-02 2.973276e-01  
## 3 Charlotte - Cleveland 3.63826583 2.744800e-04 3.568241e-03  
## 4 Calgary - Las Vegas -8.62161149 6.601836e-18 1.320367e-16  
## 5 Charlotte - Las Vegas -6.26930872 3.626545e-10 5.802472e-09  
## 6 Cleveland - Las Vegas -8.05503250 7.945769e-16 1.509696e-14  
## 7 Calgary - Mesa -1.05711710 2.904581e-01 8.713744e-01  
## 8 Charlotte - Mesa 5.03172356 4.860898e-07 6.805258e-06  
## 9 Cleveland - Mesa 1.14537642 2.520533e-01 1.000000e+00  
## 10 Las Vegas - Mesa 9.63531505 5.671775e-22 1.361226e-20  
## 11 Calgary - Phoenix -9.16583006 4.916710e-20 1.081676e-18  
## 12 Charlotte - Phoenix -7.06203644 1.640799e-12 2.789359e-11  
## 13 Cleveland - Phoenix -8.67638439 4.085497e-18 8.579543e-17  
## 14 Las Vegas - Phoenix -2.30815813 2.099034e-02 1.889131e-01  
## 15 Mesa - Phoenix -10.17002853 2.698296e-24 7.015570e-23  
## 16 Calgary - Pittsburgh -2.82698238 4.698891e-03 5.168780e-02  
## 17 Charlotte - Pittsburgh 3.50160094 4.624717e-04 5.549661e-03  
## 18 Cleveland - Pittsburgh -0.74233142 4.578866e-01 9.157731e-01  
## 19 Las Vegas - Pittsburgh 9.50758501 1.951400e-21 4.488220e-20  
## 20 Mesa - Pittsburgh -2.04825070 4.053544e-02 3.242835e-01  
## 21 Phoenix - Pittsburgh 9.99899879 1.539456e-23 3.848639e-22  
## 22 Calgary - Scottsdale -21.14577752 3.017761e-99 8.751507e-98  
## 23 Charlotte - Scottsdale -28.42631801 9.564212e-178 3.156190e-176  
## 24 Cleveland - Scottsdale -23.80033943 3.312852e-125 9.938555e-124  
## 25 Las Vegas - Scottsdale -31.85532148 1.110887e-222 3.888104e-221  
## 26 Mesa - Scottsdale -25.25355553 1.035163e-140 3.209006e-139  
## 27 Phoenix - Scottsdale -25.50460579 1.752473e-143 5.607915e-142  
## 28 Pittsburgh - Scottsdale -28.72351080 1.940896e-181 6.599047e-180  
## 29 Calgary - Toronto -1.28015558 2.004904e-01 1.000000e+00  
## 30 Charlotte - Toronto 7.36001152 1.838944e-13 3.310098e-12  
## 31 Cleveland - Toronto 1.38220925 1.669075e-01 1.000000e+00  
## 32 Las Vegas - Toronto 17.30712585 4.156519e-67 1.163825e-65  
## 33 Mesa - Toronto -0.05917191 9.528152e-01 9.528152e-01  
## 34 Phoenix - Toronto 16.40005153 1.910759e-60 5.159049e-59  
## 35 Pittsburgh - Toronto 2.76261637 5.734012e-03 5.734012e-02  
## 36 Scottsdale - Toronto 38.72993509 0.000000e+00 0.000000e+00

dunnTest(sentiment ~ cuisine, data = reviews\_sentiment, method = "holm")

## Warning: cuisine was coerced to a factor.

## Dunn (1964) Kruskal-Wallis multiple comparison  
## p-values adjusted with the Holm method.

## Comparison Z P.unadj  
## 1 american(new) - american(traditional) 35.0523680 3.588344e-269  
## 2 american(new) - asianfusion 1.7901981 7.342206e-02  
## 3 american(traditional) - asianfusion -11.9260371 8.659878e-33  
## 4 american(new) - canadian(new) 3.0133072 2.584172e-03  
## 5 american(traditional) - canadian(new) -8.4215488 3.715396e-17  
## 6 asianfusion - canadian(new) 1.2054809 2.280177e-01  
## 7 american(new) - chinese 49.2800294 0.000000e+00  
## 8 american(traditional) - chinese 26.0282089 2.374787e-149  
## 9 asianfusion - chinese 24.3392107 7.542082e-131  
## 10 canadian(new) - chinese 19.3318600 2.897803e-83  
## 11 american(new) - italian 2.8669565 4.144400e-03  
## 12 american(traditional) - italian -25.5513107 5.309556e-144  
## 13 asianfusion - italian -0.3547664 7.227646e-01  
## 14 canadian(new) - italian -1.7767236 7.561373e-02  
## 15 chinese - italian -42.0144523 0.000000e+00  
## 16 american(new) - japanese 14.0398178 8.894105e-45  
## 17 american(traditional) - japanese -12.9807117 1.574111e-38  
## 18 asianfusion - japanese 5.1685086 2.359694e-07  
## 19 canadian(new) - japanese 2.9110874 3.601732e-03  
## 20 chinese - japanese -32.1946417 2.097585e-227  
## 21 italian - japanese 9.9101139 3.762169e-23  
## 22 american(new) - mediterranean -8.6652845 4.503871e-18  
## 23 american(traditional) - mediterranean -24.9652465 1.458657e-137  
## 24 asianfusion - mediterranean -7.3511758 1.964708e-13  
## 25 canadian(new) - mediterranean -7.7943481 6.474183e-15  
## 26 chinese - mediterranean -37.6897068 0.000000e+00  
## 27 italian - mediterranean -9.8286737 8.472726e-23  
## 28 japanese - mediterranean -16.1095261 2.186934e-58  
## 29 american(new) - mexican 30.2970725 1.252791e-201  
## 30 american(traditional) - mexican 2.0063079 4.482341e-02  
## 31 asianfusion - mexican 12.3670017 3.942655e-35  
## 32 canadian(new) - mexican 8.9498038 3.561149e-19  
## 33 chinese - mexican -21.6719190 3.776837e-104  
## 34 italian - mexican 23.6672538 7.841578e-124  
## 35 japanese - mexican 12.8988107 4.570969e-38  
## 36 mediterranean - mexican 24.7128018 7.790915e-135  
## 37 american(new) - sushibars 6.3618388 1.993527e-10  
## 38 american(traditional) - sushibars -6.2082457 5.357934e-10  
## 39 asianfusion - sushibars 3.6117973 3.040822e-04  
## 40 canadian(new) - sushibars 2.1188169 3.410594e-02  
## 41 chinese - sushibars -18.1911446 6.066008e-74  
## 42 italian - sushibars 4.8964903 9.756348e-07  
## 43 japanese - sushibars -0.2497845 8.027540e-01  
## 44 mediterranean - sushibars 10.8456675 2.091038e-27  
## 45 mexican - sushibars -6.8342898 8.241228e-12  
## P.adj  
## 1 1.507104e-267  
## 2 3.671103e-01  
## 3 2.164969e-31  
## 4 2.584172e-02  
## 5 7.059252e-16  
## 6 6.840531e-01  
## 7 0.000000e+00  
## 8 9.261668e-148  
## 9 2.639729e-129  
## 10 9.272970e-82  
## 11 3.315520e-02  
## 12 2.017631e-142  
## 13 1.000000e+00  
## 14 3.024549e-01  
## 15 0.000000e+00  
## 16 2.579290e-43  
## 17 4.407510e-37  
## 18 3.067603e-06  
## 19 3.241559e-02  
## 20 8.600097e-226  
## 21 8.652990e-22  
## 22 9.007741e-17  
## 23 5.397031e-136  
## 24 3.340003e-12  
## 25 1.165353e-13  
## 26 0.000000e+00  
## 27 1.864000e-21  
## 28 6.560803e-57  
## 29 5.011163e-200  
## 30 2.689404e-01  
## 31 1.025090e-33  
## 32 7.478412e-18  
## 33 1.246356e-102  
## 34 2.666137e-122  
## 35 1.234162e-36  
## 36 2.804730e-133  
## 37 2.990290e-09  
## 38 7.501107e-09  
## 39 3.344904e-03  
## 40 2.387416e-01  
## 41 1.880462e-72  
## 42 1.170762e-05  
## 43 8.027540e-01  
## 44 5.018492e-26  
## 45 1.318596e-10

# Since alomst all of the p values are 0 we can conclude that most of of the means are different from  
# each other and different cities prefer different cuisines.  
  
# Next, we restate the data to show the preference of cuisine by city.  
# "reshape2" and "plyr" are required to calculate the average sentiment by city by cuisine.  
#install.packages("reshape2")  
library(reshape2)

##   
## Attaching package: 'reshape2'

## The following object is masked from 'package:tidyr':  
##   
## smiths

#install.packages("plyr")  
library(plyr)

## ------------------------------------------------------------------------------

## You have loaded plyr after dplyr - this is likely to cause problems.  
## If you need functions from both plyr and dplyr, please load plyr first, then dplyr:  
## library(plyr); library(dplyr)

## ------------------------------------------------------------------------------

##   
## Attaching package: 'plyr'

## The following object is masked from 'package:ggpubr':  
##   
## mutate

## The following objects are masked from 'package:dplyr':  
##   
## arrange, count, desc, failwith, id, mutate, rename, summarise,  
## summarize

reviews\_melt <- melt(reviews\_sentiment,id = c ("city", "cuisine"), measure = "sentiment")  
reviews\_dcast <- dcast(reviews\_melt, city~cuisine, fun.aggregate = mean)  
reviews\_dcast

## city american(new) american(traditional) asianfusion canadian(new)  
## 1 Calgary 1.292275 0.9220945 1.147415 1.378179  
## 2 Charlotte 1.335153 1.0554792 1.458641 NaN  
## 3 Cleveland 1.209003 1.0794547 1.136504 NaN  
## 4 Las Vegas 1.340788 1.1652829 1.393030 NaN  
## 5 Mesa 1.262758 1.0566157 1.170092 NaN  
## 6 Phoenix 1.450217 1.1969047 1.501030 NaN  
## 7 Pittsburgh 1.332703 1.1030504 1.509033 NaN  
## 8 Scottsdale 1.603538 1.4278296 1.386379 NaN  
## 9 Toronto 1.247435 1.0258488 1.260438 1.339968  
## chinese italian japanese mediterranean mexican sushibars  
## 1 0.8778550 1.297015 1.073100 1.331956 0.9977475 0.9244370  
## 2 0.7965461 1.195358 1.260619 1.526711 1.2077509 1.3822647  
## 3 0.9117185 1.335138 1.113067 1.448852 1.1850636 1.5522631  
## 4 1.0349764 1.361223 1.353869 1.541260 1.1373423 1.3076689  
## 5 0.8421314 1.080251 1.288593 1.768177 1.0099595 0.5688163  
## 6 0.8984381 1.340872 1.257301 1.499345 1.1590355 1.4244833  
## 7 1.0533608 1.142654 1.165263 1.177960 1.0405780 1.1370070  
## 8 0.9758751 1.470104 1.397138 1.360221 1.2159287 1.3130108  
## 9 0.9119880 1.311617 1.132967 1.309442 1.1886481 0.9428606

# A plot is used to visually display cuisine preferences by city.  
reviews\_plot <- melt(reviews\_dcast, id = c("city"))  
#install.packages("ggplot2")  
library(ggplot2)  
ggplot(reviews\_plot, aes(x=city, y=value, color=variable, group=variable)) +  
 geom\_line()

