PDS Assessment 2_22049939

Damian Nguyen

2025-05-07

Assumptions
- Any rows having NA or blank strings in user_id or review_id will be removed as they are not meaningful for further calculations.
- Other variables which are not used in the analysis but having NA or blank strings may not need to be removed.
- In reviews dataset, users are assumed to be in the same State with the business they reviewed.

Packages installation

```
library(tidyverse)
```

```
## Warning: package 'tidyverse' was built under R version 4.4.3
```

```
## Warning: package 'forcats' was built under R version 4.4.3
```

```
## Warning: package 'lubridate' was built under R version 4.4.3
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.5.1     v tibble    3.2.1
## v lubridate 1.9.4     v tidyr     1.3.1
## v purrr     1.0.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
library(knitr)
library(ggplot2)
library(dplyr)
library(kableExtra)
```

```
## Warning: package 'kableExtra' was built under R version 4.4.3
```

```
## Warning in attr(.knitEnv$meta, "knit_meta_id"): 'xfun::attr()' is deprecated.
## Use 'xfun::attr2()' instead.
## See help("Deprecated")
## Warning in attr(.knitEnv$meta, "knit_meta_id"): 'xfun::attr()' is deprecated.
```

```
## Use 'xfun::attr2()' instead.
## See help("Deprecated")
## Warning in attr(.knitEnv$meta, "knit_meta_id"): 'xfun::attr()' is deprecated.
## Use 'xfun::attr2()' instead.
## See help("Deprecated")
## Warning in attr(.knitEnv$meta, "knit_meta_id"): 'xfun::attr()' is deprecated.
## Use 'xfun::attr2()' instead.
## See help("Deprecated")
## Warning in attr(.knitEnv$meta, "knit_meta_id"): 'xfun::attr()' is deprecated.
## Use 'xfun::attr2()' instead.
## See help("Deprecated")
## Warning in attr(.knitEnv$meta, "knit_meta_id"): 'xfun::attr()' is deprecated.
## Use 'xfun::attr2()' instead.
## See help("Deprecated")
## Warning in attr(.knitEnv$meta, "knit_meta_id"): 'xfun::attr()' is deprecated.
## Use 'xfun::attr2()' instead.
## See help("Deprecated")
## Warning in attr(.knitEnv$meta, "knit_meta_id"): 'xfun::attr()' is deprecated.
## Use 'xfun::attr2()' instead.
## See help("Deprecated")
## Warning in attr(.knitEnv$meta, "knit_meta_id"): 'xfun::attr()' is deprecated.
## Use 'xfun::attr2()' instead.
## See help("Deprecated")
## Warning in attr(.knitEnv$meta, "knit_meta_id"): 'xfun::attr()' is deprecated.
## Use 'xfun::attr2()' instead.
## See help("Deprecated")
## Warning in attr(.knitEnv$meta, "knit_meta_id"): 'xfun::attr()' is deprecated.
## Use 'xfun::attr2()' instead.
## See help("Deprecated")
## Warning in attr(.knitEnv$meta, "knit_meta_id"): 'xfun::attr()' is deprecated.
## Use 'xfun::attr2()' instead.
## See help("Deprecated")
## Warning in attr(.knitEnv$meta, "knit_meta_id"): 'xfun::attr()' is deprecated.
## Use 'xfun::attr2()' instead.
## See help("Deprecated")
```

```
##
## Attaching package: 'kableExtra'
##
## The following object is masked from 'package:dplyr':
##
##     group_rows
```

Question 1

1.1) Data Wrangling

For this question, users dataset will be used. The first step is to review the dataset, and then format data if required.

```r
users <- read.csv("users.csv") #import data

#Review data
head(users)
```

```
##   user_id     name review_count average_stars member_since
## 1     u_0     Alan           32          2.08   2019-04-05
## 2     u_1     Joel           90          1.97   2015-11-15
## 3     u_2   Claire           93          1.10   2021-10-05
## 4     u_3 Samantha           59          3.01   2017-05-15
## 5     u_4  Monique           42          4.44   2021-04-05
## 6     u_5    Lucas           62          1.63   2023-09-16
```

```r
str(users)
```

```
## 'data.frame':    38801 obs. of  5 variables:
##  $ user_id      : chr  "u_0" "u_1" "u_2" "u_3" ...
##  $ name         : chr  "Alan" "Joel" "Claire" "Samantha" ...
##  $ review_count : int  32 90 93 59 42 62 19 93 35 76 ...
##  $ average_stars: num  2.08 1.97 1.1 3.01 4.44 1.63 3.37 3.88 2.47 3.81 ...
##  $ member_since : chr  "2019-04-05" "2015-11-15" "2021-10-05" "2017-05-15" ...
```

```r
colSums(is.na(users)) #count if there are any NA values in each column
```

```
##       user_id          name  review_count average_stars  member_since
##             0             0             0             0             0
```

```r
colSums(users == "") #count if there are any blank strings ("") in each column
```

```
##       user_id          name  review_count average_stars  member_since
##             1          1163             0             0          1160
```

Findings: Despite there is no NA values from the users dataset, the following columns - user_id, name, member_since have the blank strings (" ").

Reviewing top 15 users by review_count:

```r
top15_ReviewCount <- users %>%  arrange(desc(review_count)) %>% select(name, review_count) %>% head(15)
top15_ReviewCount
```

```
##        name review_count
## 1    Joshua           99
## 2    Edward           99
## 3   Michael           99
## 4    Daniel           99
## 5     Terri           99
## 6    Cheryl           99
## 7      Kyle           99
```

```
## 8   Kaitlin           99
## 9       Amy           99
## 10 Veronica           99
## 11     Gary           99
## 12    Sarah           99
## 13  Rebecca           99
## 14  Brandon           99
## 15 Jennifer           99
```

Conclusion: Since top 15 users have the same number of review count (99) which is not meaningful for intepretation afterward. users dataset will be merged with reviews dataset for better analaysis.

Before joining, reviews data would be examined for usability:

```r
reviews <- read.csv("reviews.csv") #import reviews dataset

#Examine data
str(reviews)
```

```
## 'data.frame':    194001 obs. of  6 variables:
##  $ review_id  : chr  "r_0" "r_1" "r_2" "r_3" ...
##  $ user_id    : chr  "u_11073" "u_35221" "u_3710" "u_23891" ...
##  $ business_id: chr  "b_4559" "b_10665" "b_7683" "b_9113" ...
##  $ stars      : int  5 3 5 3 4 2 3 2 1 4 ...
##  $ date       : chr  "2023-02-01" "2023-03-12" "2025-02-19" "2023-01-10" ...
##  $ text       : chr  "Audience hour west television. Live central spend machine. Agree would claim be
```

```r
colSums(is.na(reviews)) #No NA Values
```

```
##   review_id     user_id business_id       stars        date        text
##           0           0           0           0           0           0
```

```r
colSums((reviews==""))
```

```
##   review_id     user_id business_id       stars        date        text
##           1        5829        5834           0        5819        5802
```

```r
#check duplicated data
colSums(sapply(reviews, duplicated))
```

```
##   review_id     user_id business_id       stars        date        text
##           0      154361      174000      193996      192904        5801
```

Conclusion
- There is no NA values from the reviews dataset. However, there are empty string values in review_id user_id.
- Despite having duplicated in other values, the review_id which is essential to identify a particular information about a review is still unique. Therefore, other duplicates are acceptable.
- Only user_id variable should be addressed if there are any duplicates for further analysis.

Remove any rows having empty strings values in user_id and review_id from reviews for further analysis:

```r
cleaned_reviews <- reviews %>% filter(review_id != "") %>% filter(user_id != "")
colSums((cleaned_reviews==""))
```

```
##   review_id     user_id business_id       stars        date        text
##           0           0        5645           0        5655        5633
```

Joint data: reviews will left joint with **users** since a user can review multiple times. Therefore, this approach will ensure not missing any review_id, which will be used for counting the number of review later per user later on.

```r
reviewsUsers <- cleaned_reviews %>% left_join(users, by = c("user_id" = "user_id"))
head(reviewsUsers)
```

```
##   review_id user_id business_id stars       date
## 1       r_0 u_11073      b_4559     5 2023-02-01
## 2       r_1 u_35221     b_10665     3 2023-03-12
## 3       r_2  u_3710      b_7683     5 2025-02-19
## 4       r_3 u_23891      b_9113     3 2023-01-10
## 5       r_4 u_10374      b_7612     4 2023-01-02
## 6       r_5 u_30798      b_5793     2 2022-08-21
##
## 1 Audience hour west television. Live central spend machine. Agree would claim behavior table prevent
## 2                                                             Summer ability art beat race else large
## 3                                                              Reason range future the chair house TV
## 4                                                             Up change final prepare area difference
## 5                                                                  Size pass including performance sho
## 6                                            Pm yeah laugh necessary else store. Cut fine school phone
##          name review_count average_stars member_since
## 1                       59          4.94
## 2 Christopher            7          1.04   2020-10-18
## 3      Rhonda            9          3.72   2020-01-08
## 4        Erik           65          1.60   2021-11-27
## 5 Christopher            3          2.71   2018-01-02
## 6    Danielle           25          3.14   2021-01-24
```

1.2) Three User Groups:

After checking, the variable member_since should be formatted to Date variable in order to categorise into three groups later on

```r
reviewsUsers$member_since <- as.Date(reviewsUsers$member_since) #change to Date variable.
head(reviewsUsers$member_since) #double check the reformatted member_since
```

```
## [1] NA           "2020-10-18" "2020-01-08" "2021-11-27" "2018-01-02"
## [6] "2021-01-24"
```

This step is to create 3 different user groups - Veteran, Intermediate and New based on their joining date, using member_since Note: When filtering, blank strings are automatically transferred to NA values, and will be removed using drop_na().

```r
Veteran <- reviewsUsers %>%
  filter(reviewsUsers$member_since < as.Date('2017-01-01')) %>%
  drop_na(member_since) #removes any rows where the member_since column is NA (missing)

Intermediate <- reviewsUsers %>%
  filter(between(reviewsUsers$member_since, as.Date('2017-01-01'), as.Date('2022-12-31'))) %>% drop_na(m

New <- reviewsUsers %>%
  filter(reviewsUsers$member_since > as.Date('2022-12-31')) %>% drop_na(member_since)

#Count if there are NA values in user_id columns
sum(is.na(Veteran$user_id))
```

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

```r
sum(is.na(Intermediate$user_id))
```

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

```r
sum(is.na(New$user_id))
```

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

Conclusion:
-There are no NA values in three datasets - Veteran, Intermediate and New.
-The three datasets are ready for further analysis.

1.3) Calculate the numbers of users, their average review stars and average number of reviews per user.

Calculate the number of unique users

```r
#numbers of unique users, using count distinct as there are duplicates in each User Group
numVeteran <- Veteran %>% summarise(count = n_distinct(user_id))
numIntermediate <- Intermediate %>% summarise(count = n_distinct(user_id))
numNew<- New %>% summarise(count = n_distinct(user_id))

#convert to numberic for tabulating
numVeteran <- as.numeric(numVeteran)
numIntermediate <- as.numeric(numIntermediate)
numNew <- as.numeric(numNew)
```

Calculate the average review rating

```r
#average review
avg_Veteran <- Veteran %>% filter(!is.na(stars), stars != "") %>%# Remove NA and blank strings
  mutate(stars = as.numeric(stars)) %>% # Convert to numeric
  summarise(avg_star = mean(stars, na.rm = TRUE))
```

```r
avg_Intermediate <- Intermediate %>% filter(!is.na(stars), stars != "") %>%# Remove NA and blank string
  mutate(stars = as.numeric(stars)) %>% # Convert to numeric
  summarise(avg_star = mean(stars, na.rm = TRUE))

avg_New <- New %>% filter(!is.na(stars), stars != "") %>%# Remove NA and blank strings
  mutate(stars = as.numeric(stars)) %>% # Convert to numeric
  summarise(avg_star = mean(stars, na.rm = TRUE))

## convert to numeric for tabulation
avg_Veteran <- as.numeric(avg_Veteran)
avg_Intermediate <- as.numeric(avg_Intermediate)
avg_New <- as.numeric(avg_New)
```

Since the goal is to calculate the average number of reviews **per user**, unique number of users will be used instead of the number of user as a whole.

```r
#average review count - unique r
avgReCount_Veteran <- length(Veteran$review_id) / numVeteran
avgReCount_Intermediate <- length(Intermediate$review_id) / numIntermediate
avgReCount_New <- length(New$review_id) / numNew
```

Tabulate the data using kable:

```r
# Create a summary data frame
summaryTable <- data.frame(row.names = Group <- c("Veteran", "Intermediate", "New"),
  Number_of_Unique_Users = c(numVeteran, numIntermediate, numNew),
  Average_Review_Stars = c(avg_Veteran, avg_Intermediate, avg_New),
  Average_Review_Count = c(avgReCount_Veteran, avgReCount_Intermediate,avgReCount_New))

colnames(summaryTable) <- c("Unique Users", "Average Stars", "Average Review Count") #rename headers

# Display with table using kable()
kable(summaryTable, caption = "User Summary by Groups", digits = 3) %>% #round to 3 decimals
  kable_styling(bootstrap_options = c("striped", "hover", "condensed", "responsive"),
                full_width = FALSE,
                position = "center")
```

```
## Warning in attr(x, "align"): 'xfun::attr()' is deprecated.
## Use 'xfun::attr2()' instead.
## See help("Deprecated")

## Warning in attr(.knitEnv$meta, "knit_meta_id"): 'xfun::attr()' is deprecated.
## Use 'xfun::attr2()' instead.
## See help("Deprecated")
## Warning in attr(.knitEnv$meta, "knit_meta_id"): 'xfun::attr()' is deprecated.
## Use 'xfun::attr2()' instead.
## See help("Deprecated")
## Warning in attr(.knitEnv$meta, "knit_meta_id"): 'xfun::attr()' is deprecated.
## Use 'xfun::attr2()' instead.
## See help("Deprecated")
## Warning in attr(.knitEnv$meta, "knit_meta_id"): 'xfun::attr()' is deprecated.
## Use 'xfun::attr2()' instead.
```

```
## See help("Deprecated")
## Warning in attr(.knitEnv$meta, "knit_meta_id"): 'xfun::attr()' is deprecated.
## Use 'xfun::attr2()' instead.
## See help("Deprecated")
## Warning in attr(.knitEnv$meta, "knit_meta_id"): 'xfun::attr()' is deprecated.
## Use 'xfun::attr2()' instead.
## See help("Deprecated")
## Warning in attr(.knitEnv$meta, "knit_meta_id"): 'xfun::attr()' is deprecated.
## Use 'xfun::attr2()' instead.
## See help("Deprecated")
## Warning in attr(.knitEnv$meta, "knit_meta_id"): 'xfun::attr()' is deprecated.
## Use 'xfun::attr2()' instead.
## See help("Deprecated")
## Warning in attr(.knitEnv$meta, "knit_meta_id"): 'xfun::attr()' is deprecated.
## Use 'xfun::attr2()' instead.
## See help("Deprecated")
## Warning in attr(.knitEnv$meta, "knit_meta_id"): 'xfun::attr()' is deprecated.
## Use 'xfun::attr2()' instead.
## See help("Deprecated")
## Warning in attr(.knitEnv$meta, "knit_meta_id"): 'xfun::attr()' is deprecated.
## Use 'xfun::attr2()' instead.
## See help("Deprecated")
## Warning in attr(.knitEnv$meta, "knit_meta_id"): 'xfun::attr()' is deprecated.
## Use 'xfun::attr2()' instead.
## See help("Deprecated")
## Warning in attr(.knitEnv$meta, "knit_meta_id"): 'xfun::attr()' is deprecated.
## Use 'xfun::attr2()' instead.
## See help("Deprecated")
## Warning in attr(.knitEnv$meta, "knit_meta_id"): 'xfun::attr()' is deprecated.
## Use 'xfun::attr2()' instead.
## See help("Deprecated")

## Warning in attr(x, "align"): 'xfun::attr()' is deprecated.
## Use 'xfun::attr2()' instead.
## See help("Deprecated")

## Warning in attr(x, "format"): 'xfun::attr()' is deprecated.
## Use 'xfun::attr2()' instead.
## See help("Deprecated")
```

Table 1: User Summary by Groups

	Unique Users	Average Stars	Average Review Count
Veteran	6518	2.993	4.746
Intermediate	22470	2.999	4.751
New	8311	3.010	4.758

Findings
- Intermediate has a highest number of members (2.247×10^4), while Veteran has the lowest number.
- There are less significant differences between their average review length, indicating similar user behaviours across three groups.
- However, as old users (Veteran), their average review should be higher compared to other groups, while

their figure is the lowest, indicating quite low user behaviour from this group. - Given the considerable number of members, Intermediate average rating is the second-highest, implying a good engagement from this group.
- The average rating from Veteran (old customers) is the lowest, along with low average review length, implying their low engagement with the community.
- The average review star of New users is the highest, along with their second-highest position in number of unique users, indicating a good engagement from them and good attraction from the community recently.

1.4) Visualisation of Average Review Stars by User Groups.

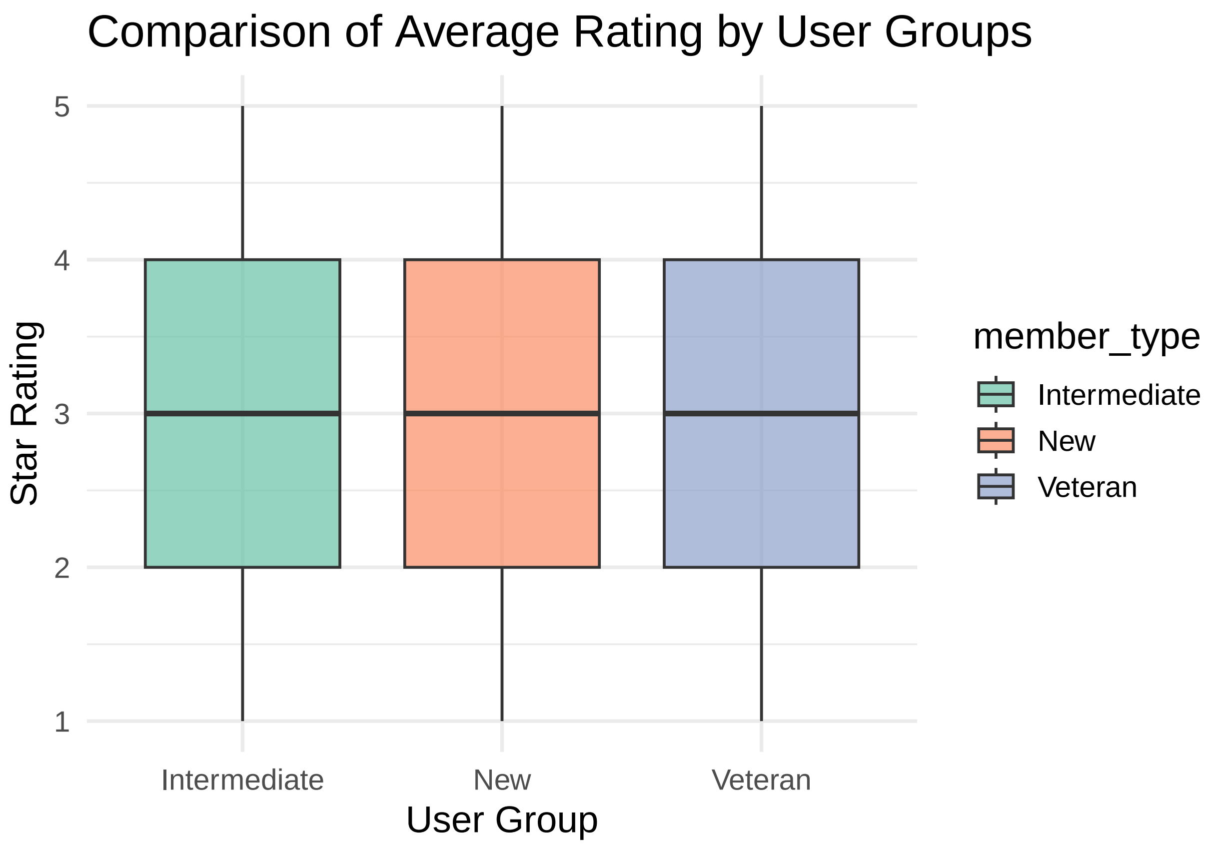
```
# Add a column member_type to the reviewsUsers dataset for data visualisation
users2 <- reviewsUsers %>% mutate( member_type = case_when(member_since < as.Date("2017-01-01") ~ "Vete
        TRUE ~ NA_character_ # Handles NA values
)) %>% drop_na(member_type) #remove NA values if required

head(users2) #check if the data is correct
```

```
##   review_id user_id business_id stars       date
## 1       r_1 u_35221     b_10665     3 2023-03-12
## 2       r_2  u_3710      b_7683     5 2025-02-19
## 3       r_3 u_23891      b_9113     3 2023-01-10
## 4       r_4 u_10374      b_7612     4 2023-01-02
## 5       r_5 u_30798      b_5793     2 2022-08-21
## 6       r_6 u_24924      b_8921     3 2025-01-23
##
## 1
## 2
## 3
## 4
## 5
## 6 Today loss experience account commercial individual specific. Hair decide run sell culture evening
##          name review_count average_stars member_since  member_type
## 1 Christopher            7          1.04   2020-10-18 Intermediate
## 2      Rhonda            9          3.72   2020-01-08 Intermediate
## 3        Erik           65          1.60   2021-11-27 Intermediate
## 4 Christopher            3          2.71   2018-01-02 Intermediate
## 5    Danielle           25          3.14   2021-01-24 Intermediate
## 6      Ronald           19          1.25   2017-08-24 Intermediate
```

```
# users2 is ready for visualisation
```

Visualisation of the Average Rating by User Groups. Boxplot is used since it can demonstrate the distribution and mean of each group.

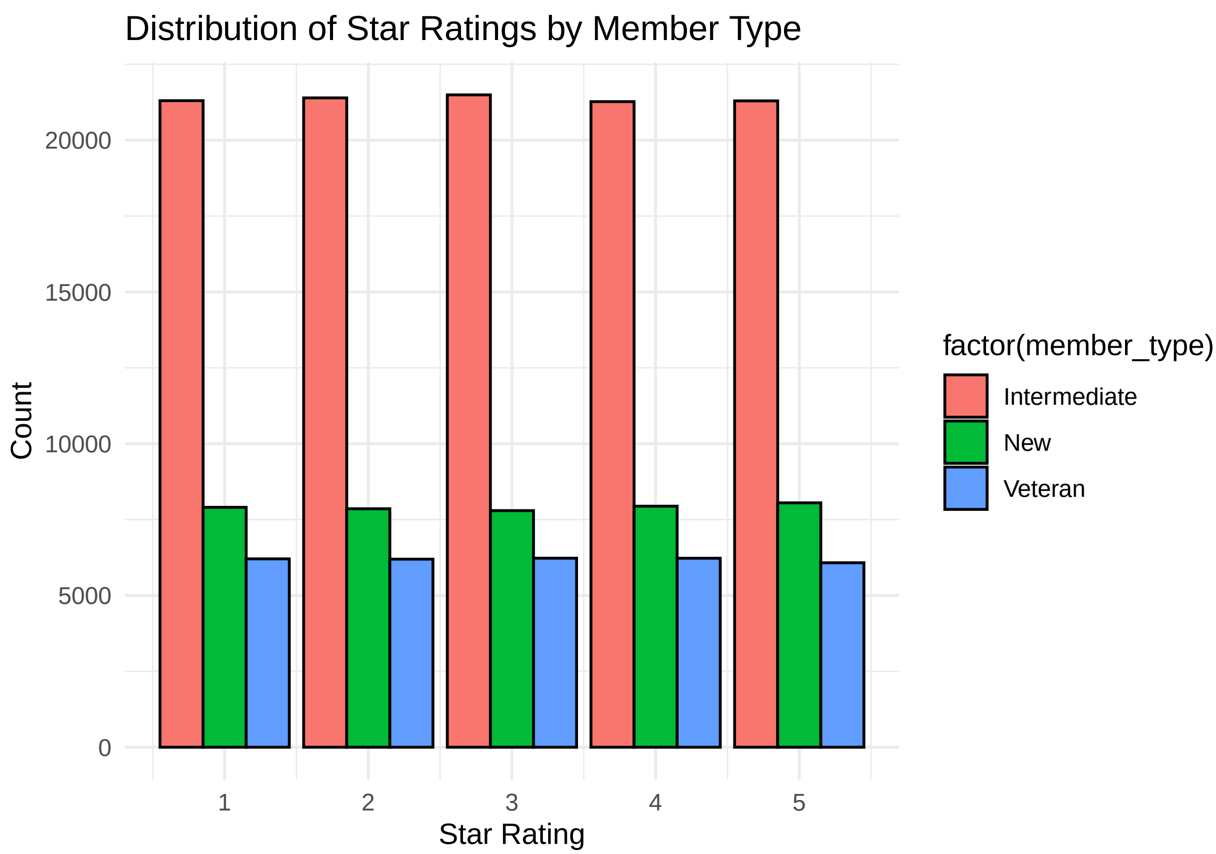
```
ggplot(users2, aes(x = member_type, y = stars, fill = member_type)) + geom_boxplot(outlier.shape = NA, a
```



Findings
- There is no statistically significant difference between the means of average rating, and the distribution (IQR) across the group.
- Most of the ratings mostly fall around 3 which is similar to the table above.

Since there is less difference between the average rating, the distribution of rating will be further examined. Therefore, bar chart will be applied in this case to visualise the distribution of 3 user groups.

```
#barplot visualising the count of rating stars by user groups
ggplot(users2, mapping = aes(x = stars, fill = factor(member_type))) + geom_bar(position = "dodge", colo
labs(title = "Distribution of Star Ratings by Member Type", x = "Star Rating",y = "Count") + theme_mini
```



Findings: From rating 1 to 5, the distribution across 3 user groups is relatively similar. This could potentially caused the least difference of their average rating between user groups.

1.5) Conclusions:

Three user groups have low differences between their average rating and average length of review.
Intermediate group has the dominant number of users with second position in average rating, indicating a relatively good engagement and behaviour.
New users with shorter time of joining but have the equivalent average length of reviews. They could be potential for future expansion of the community.

Question 2:

2.1) Data Wrangling for businessPGA

```
PGA <- read.csv("businessesPGA.csv") #import data

#examine data
str(PGA)
```

```
## 'data.frame':    11641 obs. of  9 variables:
```

```
##  $ X             : int  1 2 5 6 8 9 10 11 13 14 ...
##  $ business_id   : chr  "b_0" "b_1" "b_4" "b_5" ...
##  $ name          : chr  "Steele, Hampton and Odonnell" "Kim, Andrews and Joyce" "" "Dean, Martin and
##  $ city          : chr  "Michaelbury" "East Susan" "East Thomasshire" "Bakerberg" ...
##  $ state         : chr  "NV" "KY" "GA" "DC" ...
##  $ stars         : num  2.5 4.8 1.6 1.6 4.5 3.4 3.8 1.1 4.3 1.8 ...
##  $ review_count  : int  351 267 278 320 287 354 484 64 463 244 ...
##  $ categories    : chr  "anything, week, if" "right" "hour, rest" "success" ...
##  $ business_group: chr  "A" "A" "" "B" ...
```

```r
head(PGA)
```

```
##   X business_id                         name             city state stars
## 1 1         b_0 Steele, Hampton and Odonnell      Michaelbury    NV   2.5
## 2 2         b_1       Kim, Andrews and Joyce       East Susan    KY   4.8
## 3 5         b_4                                 East Thomasshire    GA   1.6
## 4 6         b_5       Dean, Martin and Grant        Bakerberg    DC   1.6
## 5 8         b_7                      Lee PLC  Jenniferchester    MD   4.5
## 6 9         b_8                  Griffin Inc       Vargasfurt    WI   3.4
##   review_count              categories business_group
## 1          351      anything, week, if              A
## 2          267                   right              A
## 3          278              hour, rest
## 4          320                 success              B
## 5          287                  always
## 6          354 join, could, statement              A
```

```r
colSums(is.na(PGA)) #no NA values
```

```
##              X    business_id           name           city          state
##              0              0              0              0              0
##          stars   review_count     categories business_group
##              0              0              0              0
```

```r
colSums((PGA=="")) #check if there are blank strings
```

```
##              X    business_id           name           city          state
##              0              1            350            344            342
##          stars   review_count     categories business_group
##              0              0            334            332
```

Findings:
-There is no NA values from the dataset PGA. However, the important variables for further analysis have blank strings, they are state, business_id.
- Some variables should be changed to factor variable: state, categories, business_group.

This step is to format the data:

```r
length(unique(PGA$business_id)) #to check whether all value in business_id is unique
```

```
## [1] 11641
```

```
#format data,
PGA$state <- as.factor(PGA$state)
PGA$categories <- as.factor(PGA$categories)
PGA$business_group <- as.factor(PGA$business_group)

#recheck if there are any NA values after formatting
colSums(is.na(PGA))
```

```
##              X    business_id           name           city          state
##              0              0              0              0              0
##          stars   review_count     categories business_group
##              0              0              0              0
```

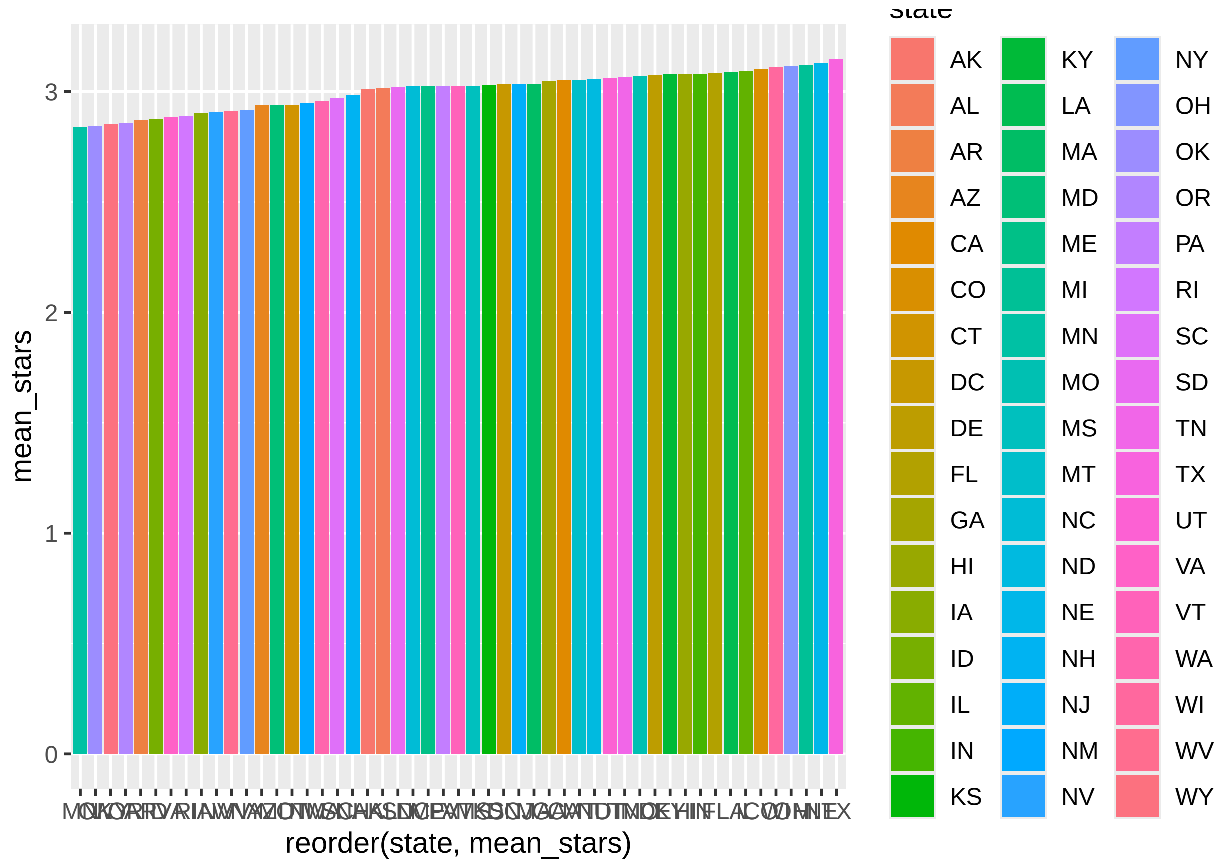
```
#removing any rows that have blank strings values for state
cleaned_PGA <- PGA %>% filter(!is.na(state), state != "")
colSums(is.na(cleaned_PGA))
```

```
##              X    business_id           name           city          state
##              0              0              0              0              0
##          stars   review_count     categories business_group
##              0              0              0              0
```

Findings: There is no NA values and blank strings after formatting. The dataset `cleaned_PGA` is ready to use

2.2) The average reviews star by State (PGA)

```
#Calculate average raring by states
avgTable_byState <- aggregate(cleaned_PGA$stars, list(cleaned_PGA$state), FUN = mean)

#Add column names for clarity in visualisation
colnames(avgTable_byState) <- c("state", "mean_stars")
avgTable_byState <- avgTable_byState %>% drop_na(c(state,mean_stars)) %>% #Remove NA values in both col
  filter(!is.na(state), state != "")

#Visualisation
ggplot(avgTable_byState, aes(x = reorder(state, mean_stars), y = mean_stars, fill = state)) + geom_bar(
```

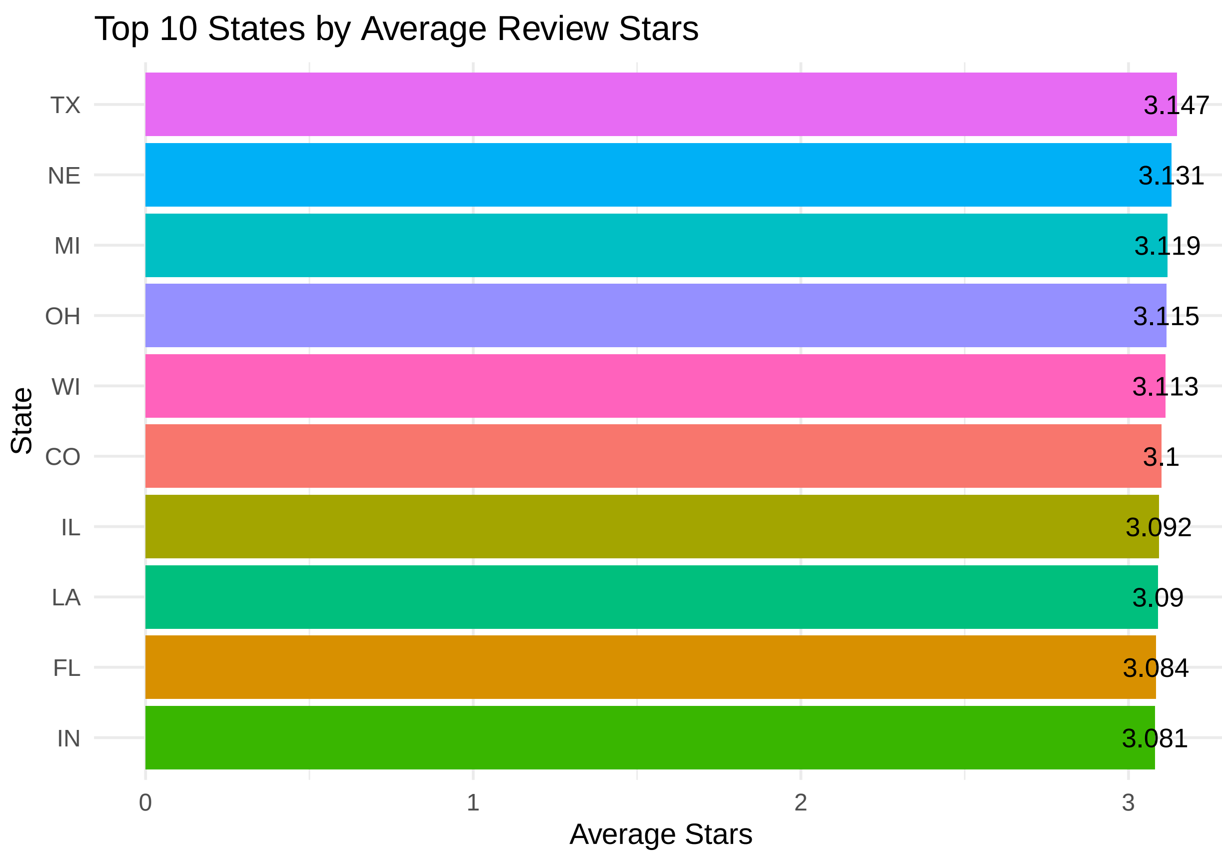



```
#count numbers of state
length(unique(PGA$state)) #52 (50 states, 1 DC, 1 Blank row calculated from blank string values)
```

```
## [1] 52
```

Findings: As there are 50 different States, the plot lacks of clarity and interpretation from data. For appropriate interpretability, only top 10 States having highest average rating will be visualised for further analysis.

```
#Select top 10
top10 <- avgTable_byState %>% arrange(desc(mean_stars)) %>% slice_head(n = 10)

#Visualisation of top 10
ggplot(top10, aes(x = reorder(state, mean_stars), y = mean_stars, fill = state)) + geom_bar(stat = "ide
  geom_text(aes(label = round(mean_stars, 3)), size = 3.5) + #adding numbers for better reading
  coord_flip() +
  labs(title = "Top 10 States by Average Review Stars", x = "State", y = "Average Stars") + theme_minima
```



Findings
- The average review stars are slightly different accross the states.
- Top 3 states with highest average rating are Texas, New York, and Milano.

2.3) The number of reviews and the number of unique users (PGA)

To count unique number of users by state, joining database is required. Two datasets will be used joinining - reviews and `PGA`. reviews will left join with `PGA` dataset as we need to calculate the number of reviews later on.
If we right join, some review data will be lost.

```
cleaned_PGA <- cleaned_PGA %>% drop_na(business_id) #remove any rows having missing business_id

#joining reviews and PGA
merge_PGA <- cleaned_reviews %>% left_join(cleaned_PGA, by = "business_id") %>%
  drop_na(c(user_id,state)) #remove any rows having missing user_id or states

#recheck NA values or blank strings after joining
colSums(is.na(merge_PGA))
```

```
##      review_id        user_id    business_id        stars.x           date
##              0              0              0              0              0
##           text              X           name           city          state
```

```
##              0              0              0              0              0
##        stars.y   review_count     categories business_group
##              0              0              0              0
```

```r
colSums((merge_PGA==""))
```

```
##      review_id        user_id    business_id        stars.x           date
##              0              0           5645              0           3311
##           text              X           name           city          state
##           3218              0           3065           3007              0
##        stars.y   review_count     categories business_group
##              0              0           2992           2928
```

Findings:
- There are still blank strings values in business_id.
- Therefore, any rows with blank strings in business_id will be removed.

```r
cleaned_merge_PGA <- merge_PGA %>% filter(!is.na(business_id), business_id != "")
colSums((cleaned_merge_PGA==""))
```

```
##      review_id        user_id    business_id        stars.x           date
##              0              0              0              0           3133
##           text              X           name           city          state
##           3042              0           3065           3007              0
##        stars.y   review_count     categories business_group
##              0              0           2992           2928
```

After joining, duplicates are more likely to appear. This step is to check if there are any duplicates and whether they are acceptable

```r
#check duplicated data
colSums(sapply(cleaned_merge_PGA, duplicated))
```

```
##      review_id        user_id    business_id        stars.x           date
##              0          66200          91929         103222         102130
##           text              X           name           city          state
##           3041          91929          93714          94840         103176
##        stars.y   review_count     categories business_group
##         103186         102737          94943         103224
```

Findings:
- There is no NA values from the joint dataset - `cleaned_merge_PGA`.
- Despite having duplicated in other values, the review_id which is essential to identify a particular information about a review is still unique. Therefore, other duplicates are acceptable.
- Only user_id variable should be addressed if there are any duplicates for further analysis.

Count the number of unique users by States

```r
uniqueUserCount <- cleaned_merge_PGA %>% distinct(user_id, state) %>% count(state, name = "unique_users
head(uniqueUserCount)
```

```
##   state unique_users
## 1    AK         2051
## 2    AL         1878
## 3    AR         1902
## 4    AZ         2175
## 5    CA         2044
## 6    CO         1882
```

Assumption: A user can be in more than 1 States, as long as their user id is unique in that particular State.

Count the number of reviews by States

```r
#count based on numbers of review_id
numReviews <- aggregate(review_id ~ state, data = cleaned_merge_PGA, FUN = length)

colnames(numReviews) <- c("state", "review_count")

# Convert state to factor variable
numReviews$state <- as.factor(numReviews$state)
numReviews <- numReviews %>% filter(!is.na(state), state != "") #Remove rows having blank strings in st
head(numReviews)
```

```
##   state review_count
## 1    AK         2102
## 2    AL         1917
## 3    AR         1937
## 4    AZ         2233
## 5    CA         2097
## 6    CO         1924
```

Summary Table of Average Star, Count of Review, and Count of Unique Users by States **Note**: Count of Review and Count of Unique Users will be calculated based on the top 10 States having highest average rating.

```r
joined_data <- top10 %>% left_join(numReviews, by = "state") %>% left_join(uniqueUserCount, by = "state

colnames(joined_data) <- c("State", "Average Stars", "Review Count", "Unique Users")

#using kable to tabulate the top 10 States
kable(joined_data, caption = "Summary of 10 States (PGA)", digits = 3) %>%
  kable_styling(bootstrap_options = c("striped", "hover", "condensed", "responsive"),
                full_width = FALSE, position = "center")
```

```
## Warning in attr(x, "align"): 'xfun::attr()' is deprecated.
## Use 'xfun::attr2()' instead.
## See help("Deprecated")
```

```
## Warning in attr(.knitEnv$meta, "knit_meta_id"): 'xfun::attr()' is deprecated.
## Use 'xfun::attr2()' instead.
## See help("Deprecated")
## Warning in attr(.knitEnv$meta, "knit_meta_id"): 'xfun::attr()' is deprecated.
## Use 'xfun::attr2()' instead.
## See help("Deprecated")
## Warning in attr(.knitEnv$meta, "knit_meta_id"): 'xfun::attr()' is deprecated.
## Use 'xfun::attr2()' instead.
## See help("Deprecated")
## Warning in attr(.knitEnv$meta, "knit_meta_id"): 'xfun::attr()' is deprecated.
## Use 'xfun::attr2()' instead.
## See help("Deprecated")
## Warning in attr(.knitEnv$meta, "knit_meta_id"): 'xfun::attr()' is deprecated.
## Use 'xfun::attr2()' instead.
## See help("Deprecated")
## Warning in attr(.knitEnv$meta, "knit_meta_id"): 'xfun::attr()' is deprecated.
## Use 'xfun::attr2()' instead.
## See help("Deprecated")
## Warning in attr(.knitEnv$meta, "knit_meta_id"): 'xfun::attr()' is deprecated.
## Use 'xfun::attr2()' instead.
## See help("Deprecated")
## Warning in attr(.knitEnv$meta, "knit_meta_id"): 'xfun::attr()' is deprecated.
## Use 'xfun::attr2()' instead.
## See help("Deprecated")
## Warning in attr(.knitEnv$meta, "knit_meta_id"): 'xfun::attr()' is deprecated.
## Use 'xfun::attr2()' instead.
## See help("Deprecated")
## Warning in attr(.knitEnv$meta, "knit_meta_id"): 'xfun::attr()' is deprecated.
## Use 'xfun::attr2()' instead.
## See help("Deprecated")
## Warning in attr(.knitEnv$meta, "knit_meta_id"): 'xfun::attr()' is deprecated.
## Use 'xfun::attr2()' instead.
## See help("Deprecated")
## Warning in attr(.knitEnv$meta, "knit_meta_id"): 'xfun::attr()' is deprecated.
## Use 'xfun::attr2()' instead.
## See help("Deprecated")
## Warning in attr(.knitEnv$meta, "knit_meta_id"): 'xfun::attr()' is deprecated.
## Use 'xfun::attr2()' instead.
## See help("Deprecated")
## Warning in attr(.knitEnv$meta, "knit_meta_id"): 'xfun::attr()' is deprecated.
## Use 'xfun::attr2()' instead.
## See help("Deprecated")

## Warning in attr(x, "align"): 'xfun::attr()' is deprecated.
## Use 'xfun::attr2()' instead.
## See help("Deprecated")

## Warning in attr(x, "format"): 'xfun::attr()' is deprecated.
## Use 'xfun::attr2()' instead.
## See help("Deprecated")
```

Table 2: Summary of 10 States (PGA)

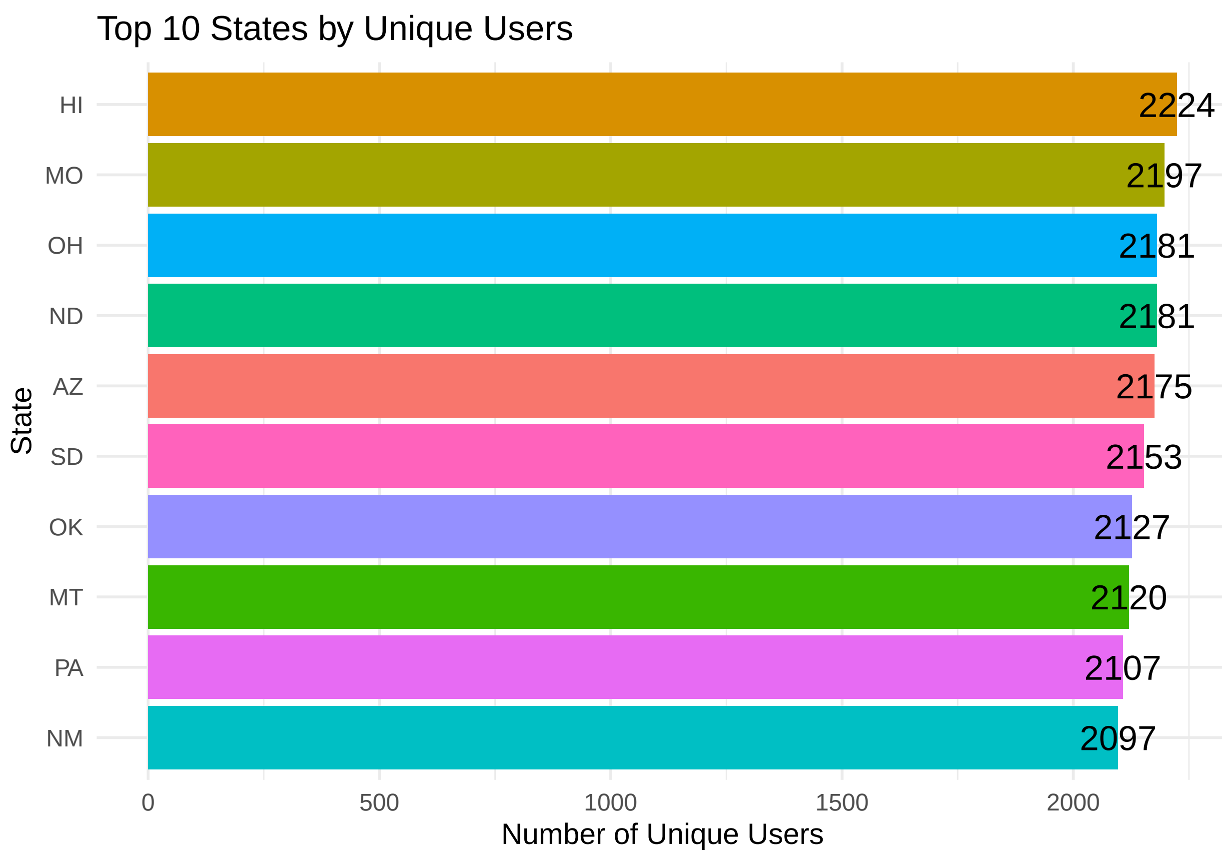
State	Average Stars	Review Count	Unique Users
TX	3.147	2087	2043
NE	3.131	2009	1963
MI	3.119	1863	1825
OH	3.115	2234	2181
WI	3.113	1850	1814
CO	3.100	1924	1882
IL	3.092	1862	1817
LA	3.090	1984	1932
FL	3.084	1898	1852
IN	3.081	1687	1645

Findings
- Texas (TX) with the highest average rating also has the relatively high number of number of unique users and reviews, indicating a positive and active engagement from users in this state.
- Ohio (OH) has the highest number number of unique users and reviews and ranked in 4 out of 10 among top rating States. The company could take advantages of this high number of users for future expansion.
- Indiana (IN) has a lowest rating among top 10, and quite low number of unique users and review count, indicating low potential. Thus, the company should not prioritise IN over other States in top 10.

2.4) Visualisation of Unique users by State (PGA)

As there are 51 different States, only top 10 States having the highest number of unique users will be used.

```
#Select top 10
top10_Unqiue <- uniqueUserCount %>% arrange(desc(unique_users)) %>% drop_na(state) %>% slice_head(n = 10

#visualisation
ggplot(top10_Unqiue, aes(x = reorder(state, unique_users), y = unique_users, fill = state)) + geom_bar(s
  geom_text(aes(label = unique_users,  size = 3)) +
  coord_flip() +
  labs(title = "Top 10 States by Unique Users", x = "State", y = "Number of Unique Users") + theme_mini
```



Findings:
- As ranked in the third for number of unique users, along with its relatively high rank in average rating, Ohio could be the potential State with high number of user and high engagement. The company should consider OH for future targeting.

(Find the common State in top 10 by average rating and unique users)

```
common_10 <- top10_Unqiue %>% inner_join(top10, by = "state")
common_10
```

```
##   state unique_users mean_stars
## 1    OH         2181   3.115385
```

- HI and MO also have the highest number of users from the community, indicating a potential base for expansion as well.

2.4a) New Test - Git Branch:

Test objective: To identify any differences in States outcome if taking top 10 States having highest number of unique users as reference when merging.

```r
#Create table having unique user count by States
uniqueUserCount <- cleaned_merge_PGA %>% distinct(user_id, state) %>% count(state, name = "unique_users
head(uniqueUserCount) #check data before proceeding
```

```
##   state unique_users
## 1    AK         2051
## 2    AL         1878
## 3    AR         1902
## 4    AZ         2175
## 5    CA         2044
## 6    CO         1882
```

```r
Top10_UniqueUsers <- uniqueUserCount %>% arrange(desc(unique_users))
Top10_UniqueUsers[1:10,] #pick top 10 after sort descending
```

```
##    state unique_users
## 1     HI         2224
## 2     MO         2197
## 3     ND         2181
## 4     OH         2181
## 5     AZ         2175
## 6     SD         2153
## 7     OK         2127
## 8     MT         2120
## 9     PA         2107
## 10    NM         2097
```

Merging with average star and review count tables based on top 10 States having highest unique users.

```r
Top10_States <- Top10_UniqueUsers %>% left_join(numReviews, by = "state") %>% left_join(avgTable_byStat

colnames(Top10_States) <- c("State", "Unique Users","Review Count","Average Stars")
Top10_States #print output
```

```
##    State Unique Users Review Count Average Stars
## 1     HI         2224         2292      3.079447
## 2     MO         2197         2246      3.071967
## 3     ND         2181         2231      3.058403
## 4     OH         2181         2234      3.115385
## 5     AZ         2175         2233      2.939918
## 6     SD         2153         2207      3.021212
## 7     OK         2127         2190      2.845148
## 8     MT         2120         2167      3.053586
## 9     PA         2107         2171      3.025106
## 10    NM         2097         2149      2.947436
## 11    MD         2082         2133      2.940773
## 12    GA         2078         2135      3.047807
## 13    MS         2057         2111      3.026432
## 14    WV         2054         2123      2.913734
## 15    AK         2051         2102      3.010593
```

```
## 16    IA          2048           2114       2.904762
## 17    TN          2048           2107       3.066812
## 18    WA          2048           2094       2.957589
## 19    CA          2044           2097       3.051055
## 20    TX          2043           2087       3.147391
## 21    NY          2042           2094       2.918421
## 22    SC          2017           2069       2.968996
## 23    OR          2016           2072       2.857534
## 24    MN          2014           2066       2.841410
## 25    VA          2011           2071       2.883843
## 26    ME          2003           2063       3.025000
## 27    KY          1995           2038       3.077477
## 28    WY          1976           2022       2.855172
## 29    NE          1963           2009       3.131429
## 30    NC          1948           2000       3.024186
## 31    VT          1942           1989       3.025822
## 32    LA          1932           1984       3.090222
## 33    AR          1902           1937       2.872685
## 34    DC          1898           1946       3.033175
## 35    MA          1893           1951       3.035714
## 36    NJ          1891           1936       3.033945
## 37    RI          1883           1927       2.890868
## 38    CO          1882           1924       3.100000
## 39    AL          1878           1917       3.017241
## 40    NH          1855           1902       2.982160
## 41    FL          1852           1898       3.083732
## 42    KS          1852           1889       3.029612
## 43    CT          1844           1883       2.940865
## 44    UT          1829           1876       3.060000
## 45    MI          1825           1863       3.119048
## 46    IL          1817           1862       3.092417
## 47    WI          1814           1850       3.112871
## 48    DE          1767           1799       3.073737
## 49    ID          1734           1782       2.875130
## 50    NV          1660           1698       2.907027
## 51    IN          1645           1687       3.080899
```

Findings:
- If sorting by top 10 unique users instead of average stars, three States that have relatively good rankings with unique users count, review count, and average rating are MO, HI, and OH. These are the optimal States for future targeting.
- TX State was removed from the outcome, indicating its low optimal performance if using number of unique users as reference.

2.5) Data Wrangling for businessPGB

```r
PGB <- read.csv("businessesPGB.csv")

#examine data
str(PGB)
```

```
## 'data.frame':    7760 obs. of  9 variables:
##  $ X             : int  3909 4211 16137 8108 19289 1482 6049 18758 13834 19284 ...
##  $ business_id   : chr  "b_4030" "b_4339" "b_16620" "b_8350" ...
##  $ name          : chr  "Chandler Group" "Cook-Anderson" "Powell, Medina and Kennedy" "Owen, Francis
##  $ city          : chr  "" "New Brandon" "New Scott" "Fergusonburgh" ...
##  $ state         : chr  "NH" "ID" "KS" "MO" ...
##  $ stars         : num  3.7 1.9 3.3 4.3 4.8 3.4 4.9 1.4 4.3 1.7 ...
##  $ review_count  : int  481 116 413 226 346 196 340 161 415 151 ...
##  $ categories    : chr  "ground" "million, heart" "environment, might" "when, eat, too" ...
##  $ business_group: chr  "B" "A" "B" "A" ...
```

```
head(PGB)
```

```
##       X business_id                       name          city state stars
## 1  3909      b_4030             Chandler Group                  NH   3.7
## 2  4211      b_4339              Cook-Anderson   New Brandon    ID   1.9
## 3 16137     b_16620 Powell, Medina and Kennedy     New Scott    KS   3.3
## 4  8108      b_8350   Owen, Francis and Franco Fergusonburgh    MO   4.3
## 5 19289     b_19880                Clark-Banks                  DC   4.8
## 6  1482      b_1531             Collier-Krause      Ellenton    SD   3.4
##   review_count         categories business_group
## 1          481             ground              B
## 2          116     million, heart              A
## 3          413 environment, might              B
## 4          226     when, eat, too              A
## 5          346    respond, sister              A
## 6          196        send, again              A
```

```
colSums(is.na(PGB)) #no NA values
```

```
##              X    business_id           name           city          state
##              0              0              0              0              0
##          stars   review_count     categories business_group
##              0              0              0              0
```

```
colSums((PGB=="")) #check if there are blank strings
```

```
##              X    business_id           name           city          state
##              0              0            235            245            238
##          stars   review_count     categories business_group
##              0              0            251            245
```

Conclusions:
- There is no NA values from `PGB` dataset.
- There is blank strings from the dataset.

This step is to format the data:

```
length(unique(PGB$business_id)) #to check whether all value in business_id is unique
```

```
## [1] 7760
```

```r
# conclusion: all business id is unique

#format data,
PGB$state <- as.factor(PGB$state)
PGB$categories <- as.factor(PGB$categories)
PGB$business_group <- as.factor(PGB$business_group)

#recheck if there are any NA values after formatting
colSums(is.na(PGB))
```

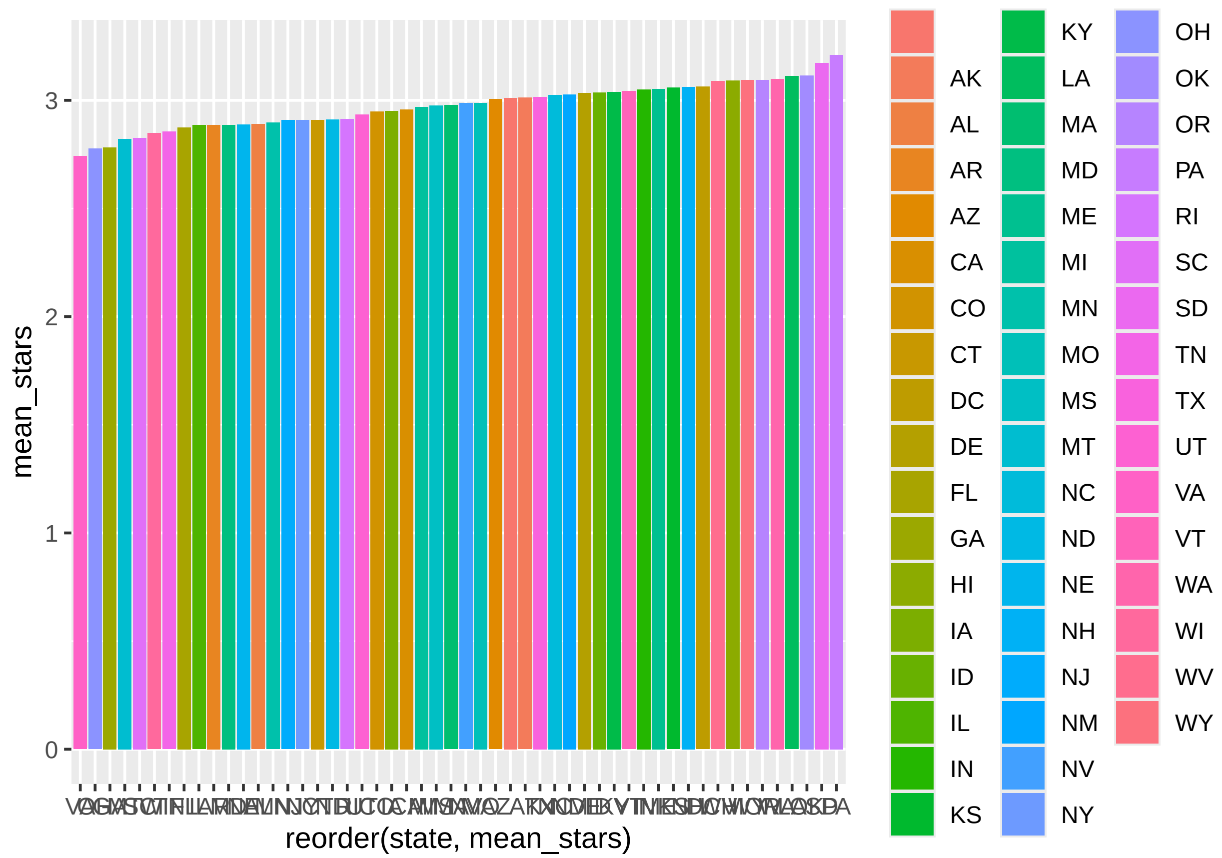
```
##              X    business_id           name           city          state
##              0              0              0              0              0
##          stars   review_count     categories business_group
##              0              0              0              0
```

```r
#removing any rows that have blank strings values for state
cleaned_PGB <- PGB %>% filter(!is.na(state), state != "")
colSums(is.na(cleaned_PGB))
```

```
##              X    business_id           name           city          state
##              0              0              0              0              0
##          stars   review_count     categories business_group
##              0              0              0              0
```

2.6) The average review stars by State (PGB)

```r
#Calculate average raring by states
avg_byStateB <- aggregate(PGB$stars, list(PGB$state), FUN = mean)

#Add column names for clarity in visualisation
colnames(avg_byStateB) <- c("state", "mean_stars")
avg_byState <- avg_byStateB %>% drop_na(c(state,mean_stars)) #Remove NA values in both columns

#Visualisation
ggplot(avg_byStateB, aes(x = reorder(state, mean_stars), y = mean_stars, fill = state)) + geom_bar(stat
```

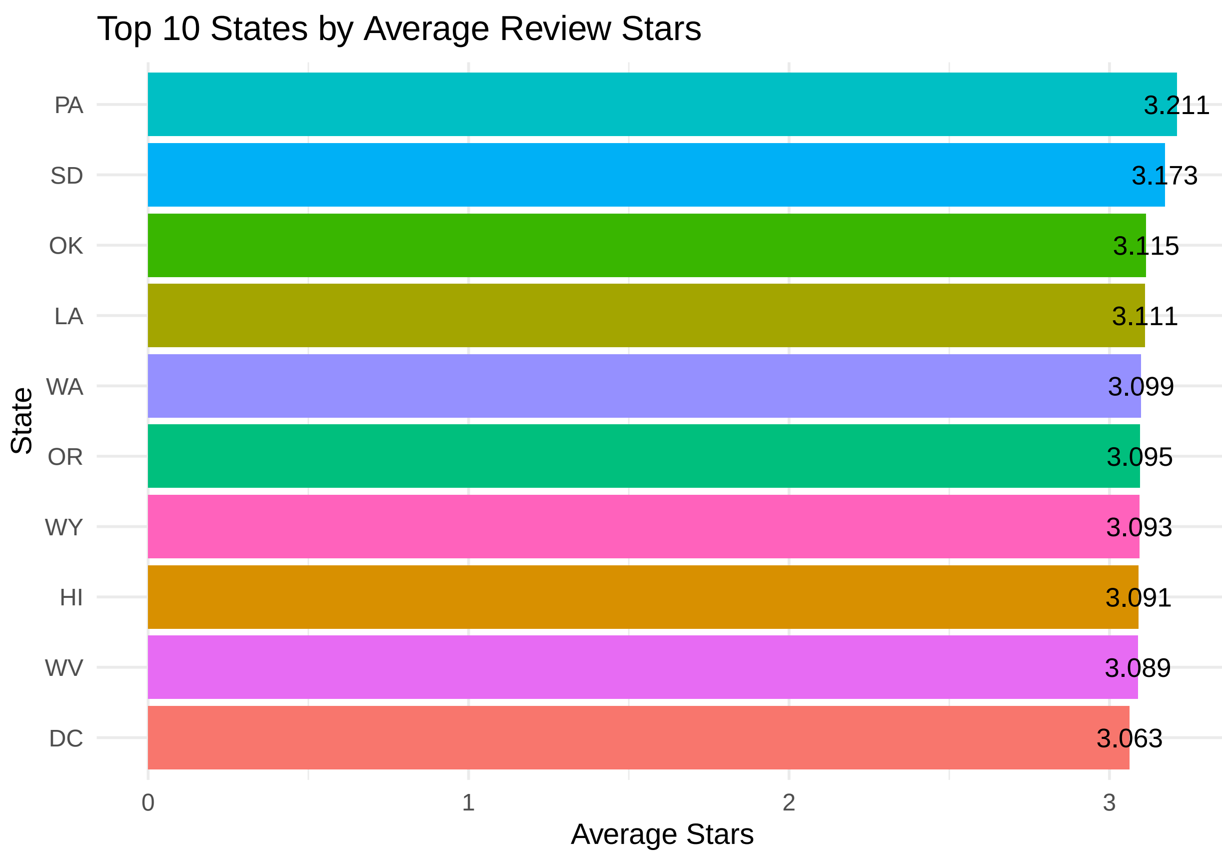



```
#count numbers of state
length(unique(PGB$state)) #52
```

```
## [1] 52
```

Conclusion: As mentioned in 2.2, only top 10 states by average review star will be considered for appropriate interpretability and further analysis afterward.

```
#Select top 10 of PGB
top10B <- avg_byStateB %>% arrange(desc(mean_stars)) %>% slice_head(n = 10)

#Visualisation of top 10
ggplot(top10B, aes(x = reorder(state, mean_stars), y = mean_stars, fill = state)) + geom_bar(stat = "ide
  geom_text(aes(label = round(mean_stars, 3)), size = 3.5) + #adding numbers for better reading
  coord_flip() +
  labs(title = "Top 10 States by Average Review Stars", x = "State", y = "Average Stars") + theme_minima
```



2.7) The number of reviews and the number of unique users (PGB)

To count unique number of users by state, joining database is required. Two datasets will be used joining - reviews and `PGB`. reviews will left join with `PGB` dataset as we need to calculate the number of reviews later on.
If we right join, some review data will be lost.

```
cleaned_PGB <- cleaned_PGB %>% drop_na(business_id) #remove any rows having missing business_id

#joining reviews and PGA
merge_PGB <- cleaned_reviews %>% left_join(cleaned_PGB, by = "business_id") %>%
  drop_na(c(user_id,state)) #remove any rows having missing user_id or states

#recheck NA values or blank strings after joining
colSums(is.na(merge_PGB))
```

```
##       review_id         user_id     business_id         stars.x            date
##               0               0               0               0               0
##            text               X            name            city           state
##               0               0               0               0               0
##         stars.y    review_count      categories business_group
##               0               0               0               0
```

```r
colSums((merge_PGB==""))
```

```
##       review_id         user_id     business_id          stars.x            date 
##               0               0               0                0            2017 
##            text               X            name             city           state 
##            2061               0            2060             2115               0 
##         stars.y    review_count      categories   business_group 
##               0               0            2171             2061
```

Findings:
- There are still blank strings values in business_id.
- Therefore, any rows with blank strings in business_id will be removed.

```r
cleaned_merge_PGB <- merge_PGB %>% filter(!is.na(business_id), business_id != "")
colSums((cleaned_merge_PGB==""))
```

```
##       review_id         user_id     business_id          stars.x            date 
##               0               0               0                0            2017 
##            text               X            name             city           state 
##            2061               0            2060             2115               0 
##         stars.y    review_count      categories   business_group 
##               0               0            2171             2061
```

After joining, duplicates are more likely to appear. This step is to check if there are any duplicates and whether they are acceptable.

```r
#check duplicated data
colSums(sapply(cleaned_merge_PGB, duplicated))
```

```
##       review_id         user_id     business_id          stars.x            date 
##               0           35796           61034            68551           67459 
##            text               X            name             city           state 
##            2060           61034           62059            62608           68505 
##         stars.y    review_count      categories   business_group 
##           68515           68066           62768            68553
```

Findings:
- There is no NA values from the joint dataset - `cleaned_merge_PGB`.
- Despite having duplicated in other values, the review_id which is essential to identify a particular information about a review is still unique. Therefore, other duplicates are acceptable.
- Only user_id variable should be addressed if there are any duplicates for further analysis.

Count the number of unique users by States

```r
uniqueUserCountB <- cleaned_merge_PGB %>% distinct(user_id, state) %>% count(state, name = "unique_users
head(uniqueUserCountB)
```

```
##   state unique_users
## 1    AK         1225
```

```
## 2    AL          1487
## 3    AR          1416
## 4    AZ          1250
## 5    CA          1323
## 6    CO          1385
```

Assumption: A user can be in more than 1 States, as long as their user id is unique in that particular State.

Count the number of reviews by States

```
#count based on numbers of review_id
numReviewsB <- aggregate(review_id ~ state, data = cleaned_merge_PGB, FUN = length)

colnames(numReviewsB) <- c("state", "review_count")

# Convert state to factor variable
numReviewsB$state <- as.factor(numReviewsB$state)
numReviewsB <- numReviewsB %>% filter(!is.na(state), state != "") #Remove rows having blank strings in
head(numReviewsB)
```

```
##   state review_count
## 1    AK         1249
## 2    AL         1518
## 3    AR         1442
## 4    AZ         1277
## 5    CA         1341
## 6    CO         1415
```

Summary Table of Average Star, Count of Review, and Count of Unique Users by States.
Note: Count of Review and Count of Unique Users will be calculated based on the top 10 States having highest average rating.

```
joined_dataB <- top10B %>% left_join(numReviewsB, by = "state") %>% left_join(uniqueUserCountB, by = "s

colnames(joined_dataB) <- c("State", "Average Stars", "Review Count", "Unique Users") #rename headers

#using kable to tabulate the top 10 States
kable(joined_dataB, caption = "Summary of 10 States (PGB)", digits = 3) %>%
  kable_styling(bootstrap_options = c("striped", "hover", "condensed", "responsive"),
                full_width = FALSE, position = "center")
```

```
## Warning in attr(x, "align"): 'xfun::attr()' is deprecated.
## Use 'xfun::attr2()' instead.
## See help("Deprecated")
```

```
## Warning in attr(.knitEnv$meta, "knit_meta_id"): 'xfun::attr()' is deprecated.
## Use 'xfun::attr2()' instead.
## See help("Deprecated")
## Warning in attr(.knitEnv$meta, "knit_meta_id"): 'xfun::attr()' is deprecated.
## Use 'xfun::attr2()' instead.
## See help("Deprecated")
## Warning in attr(.knitEnv$meta, "knit_meta_id"): 'xfun::attr()' is deprecated.
```

```
## Use 'xfun::attr2()' instead.
## See help("Deprecated")
## Warning in attr(.knitEnv$meta, "knit_meta_id"): 'xfun::attr()' is deprecated.
## Use 'xfun::attr2()' instead.
## See help("Deprecated")
## Warning in attr(.knitEnv$meta, "knit_meta_id"): 'xfun::attr()' is deprecated.
## Use 'xfun::attr2()' instead.
## See help("Deprecated")
## Warning in attr(.knitEnv$meta, "knit_meta_id"): 'xfun::attr()' is deprecated.
## Use 'xfun::attr2()' instead.
## See help("Deprecated")
## Warning in attr(.knitEnv$meta, "knit_meta_id"): 'xfun::attr()' is deprecated.
## Use 'xfun::attr2()' instead.
## See help("Deprecated")
## Warning in attr(.knitEnv$meta, "knit_meta_id"): 'xfun::attr()' is deprecated.
## Use 'xfun::attr2()' instead.
## See help("Deprecated")
## Warning in attr(.knitEnv$meta, "knit_meta_id"): 'xfun::attr()' is deprecated.
## Use 'xfun::attr2()' instead.
## See help("Deprecated")
## Warning in attr(.knitEnv$meta, "knit_meta_id"): 'xfun::attr()' is deprecated.
## Use 'xfun::attr2()' instead.
## See help("Deprecated")
## Warning in attr(.knitEnv$meta, "knit_meta_id"): 'xfun::attr()' is deprecated.
## Use 'xfun::attr2()' instead.
## See help("Deprecated")
## Warning in attr(.knitEnv$meta, "knit_meta_id"): 'xfun::attr()' is deprecated.
## Use 'xfun::attr2()' instead.
## See help("Deprecated")
## Warning in attr(.knitEnv$meta, "knit_meta_id"): 'xfun::attr()' is deprecated.
## Use 'xfun::attr2()' instead.
## See help("Deprecated")
## Warning in attr(.knitEnv$meta, "knit_meta_id"): 'xfun::attr()' is deprecated.
## Use 'xfun::attr2()' instead.
## See help("Deprecated")
```

```
## Warning in attr(x, "align"): 'xfun::attr()' is deprecated.
## Use 'xfun::attr2()' instead.
## See help("Deprecated")
```

```
## Warning in attr(x, "format"): 'xfun::attr()' is deprecated.
## Use 'xfun::attr2()' instead.
## See help("Deprecated")
```

Table 3: Summary of 10 States (PGB)

State	Average Stars	Review Count	Unique Users
PA	3.211	1393	1377
SD	3.173	1350	1323
OK	3.115	1462	1430
LA	3.111	1408	1379
WA	3.099	1559	1525

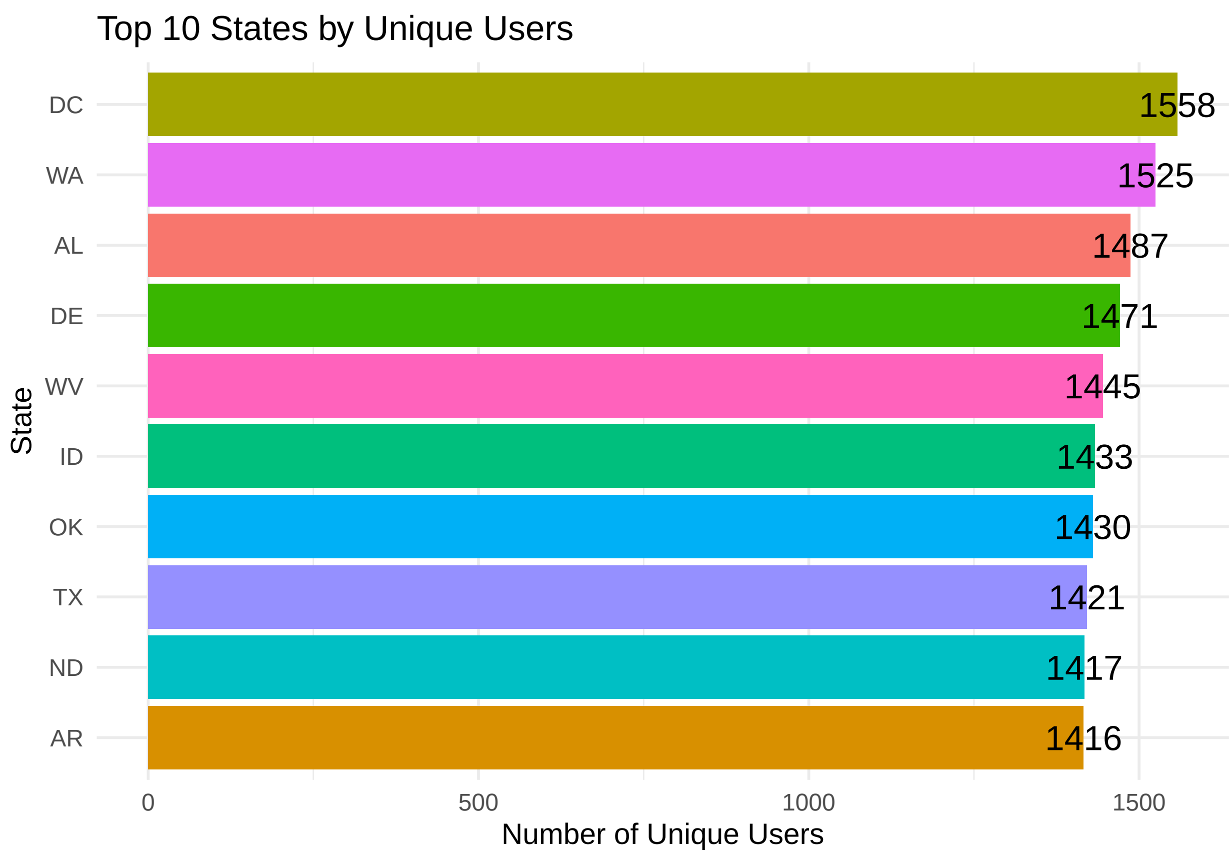
OR	3.095	1376	1355
WY	3.093	1221	1198
HI	3.091	1325	1305
WV	3.089	1465	1445
DC	3.063	1581	1558

Findings
- PA State has the highest number of average rating, but relatively low number in number of unique users and review count, indicating a good engagement on average per user here.
- DC while has the lowest average rating among top 10 but have the highest number of unique users and review count. Thus, the firm should come up with strategies to improve customer engagement here.
- WA state could be the optimal State for targeting as it ranked in the 5th in average rating, and have relatively high number of unique users and review count.

2.8) Visualisation of Unique Users by States (PGB)

As there are 50 different States, only top 10 States having the highest number of unique users will be used.

```
#Select top 10
top10_UnqiueB <- uniqueUserCountB %>% arrange(desc(unique_users)) %>% drop_na(state) %>% slice_head(n =

#visualisation
ggplot(top10_UnqiueB, aes(x = reorder(state, unique_users), y = unique_users, fill = state)) + geom_bar
  geom_text(aes(label = unique_users,  size = 3)) +
  coord_flip() +
  labs(title = "Top 10 States by Unique Users", x = "State", y = "Number of Unique Users") + theme_mini
```



Findings:
- As mentioned above, although DC has the highest number of unique users, its average rating is the lowest among top 10.
- The following states are the optimal ones between high average rating and high number of unique users in top 10, implying a high potential for targeting.

```
common_10B <- top10_UnqiueB %>% inner_join(top10B, by = "state")
common_10B
```

```
##   state unique_users mean_stars
## 1    DC         1558   3.063006
## 2    WA         1525   3.098780
## 3    WV         1445   3.088742
## 4    OK         1430   3.114570
```

- OK, WA, and WV could be potential States for the firm's community expansion, with their relatively high and balanced ranks in both unique users and average stars.

2.9) Differences between PGA and PGB

Create a summary data for both PGA and PGB, having 52 States describing both

```r
summaryPGA <- avgTable_byState %>% left_join(numReviews, by = "state") %>%
  left_join(uniqueUserCount, by = "state") %>% mutate(Group = "PGA")  #add column Group to categorise

summaryPGB <- avg_byStateB %>% left_join(numReviewsB, by = "state") %>% left_join(uniqueUserCountB, by =

#review the data
summaryPGA
```

```
##    state mean_stars review_count unique_users Group
## 1     AK   3.010593         2102         2051   PGA
## 2     AL   3.017241         1917         1878   PGA
## 3     AR   2.872685         1937         1902   PGA
## 4     AZ   2.939918         2233         2175   PGA
## 5     CA   3.051055         2097         2044   PGA
## 6     CO   3.100000         1924         1882   PGA
## 7     CT   2.940865         1883         1844   PGA
## 8     DC   3.033175         1946         1898   PGA
## 9     DE   3.073737         1799         1767   PGA
## 10    FL   3.083732         1898         1852   PGA
## 11    GA   3.047807         2135         2078   PGA
## 12    HI   3.079447         2292         2224   PGA
## 13    IA   2.904762         2114         2048   PGA
## 14    ID   2.875130         1782         1734   PGA
## 15    IL   3.092417         1862         1817   PGA
## 16    IN   3.080899         1687         1645   PGA
## 17    KS   3.029612         1889         1852   PGA
## 18    KY   3.077477         2038         1995   PGA
## 19    LA   3.090222         1984         1932   PGA
## 20    MA   3.035714         1951         1893   PGA
## 21    MD   2.940773         2133         2082   PGA
## 22    ME   3.025000         2063         2003   PGA
## 23    MI   3.119048         1863         1825   PGA
## 24    MN   2.841410         2066         2014   PGA
## 25    MO   3.071967         2246         2197   PGA
## 26    MS   3.026432         2111         2057   PGA
## 27    MT   3.053586         2167         2120   PGA
## 28    NC   3.024186         2000         1948   PGA
## 29    ND   3.058403         2231         2181   PGA
## 30    NE   3.131429         2009         1963   PGA
## 31    NH   2.982160         1902         1855   PGA
## 32    NJ   3.033945         1936         1891   PGA
## 33    NM   2.947436         2149         2097   PGA
## 34    NV   2.907027         1698         1660   PGA
## 35    NY   2.918421         2094         2042   PGA
## 36    OH   3.115385         2234         2181   PGA
## 37    OK   2.845148         2190         2127   PGA
## 38    OR   2.857534         2072         2016   PGA
## 39    PA   3.025106         2171         2107   PGA
## 40    RI   2.890868         1927         1883   PGA
## 41    SC   2.968996         2069         2017   PGA
## 42    SD   3.021212         2207         2153   PGA
## 43    TN   3.066812         2107         2048   PGA
## 44    TX   3.147391         2087         2043   PGA
```

```
## 45    UT   3.060000         1876         1829   PGA
## 46    VA   2.883843         2071         2011   PGA
## 47    VT   3.025822         1989         1942   PGA
## 48    WA   2.957589         2094         2048   PGA
## 49    WI   3.112871         1850         1814   PGA
## 50    WV   2.913734         2123         2054   PGA
## 51    WY   2.855172         2022         1976   PGA
```

```
summaryPGB
```

```
##    state mean_stars review_count unique_users Group
## 1     AK   3.012319         1249         1225   PGB
## 2     AL   2.889506         1518         1487   PGB
## 3     AR   2.886364         1442         1416   PGB
## 4     AZ   3.005479         1277         1250   PGB
## 5     CA   2.958333         1341         1323   PGB
## 6     CO   2.948026         1415         1385   PGB
## 7     CT   2.909868         1398         1376   PGB
## 8     DC   3.063006         1581         1558   PGB
## 9     DE   3.034568         1498         1471   PGB
## 10    FL   2.874342         1352         1329   PGB
## 11    GA   2.782639         1280         1261   PGB
## 12    HI   3.091034         1325         1305   PGB
## 13    IA   2.951351         1302         1281   PGB
## 14    ID   3.037037         1456         1433   PGB
## 15    IL   2.885106         1321         1301   PGB
## 16    IN   3.051079         1263         1236   PGB
## 17    KS   3.058741         1278         1249   PGB
## 18    KY   3.038562         1339         1312   PGB
## 19    LA   3.111333         1408         1379   PGB
## 20    MA   2.977863         1238         1229   PGB
## 21    MD   2.886719         1126         1107   PGB
## 22    ME   3.052941         1231         1207   PGB
## 23    MI   2.969427         1392         1370   PGB
## 24    MN   2.897143         1379         1359   PGB
## 25    MO   2.987662         1393         1370   PGB
## 26    MS   2.976552         1254         1236   PGB
## 27    MT   2.821233         1351         1332   PGB
## 28    NC   3.024667         1419         1390   PGB
## 29    ND   2.910828         1448         1417   PGB
## 30    NE   2.888235         1096         1083   PGB
## 31    NH   3.061806         1330         1314   PGB
## 32    NJ   2.908553         1401         1383   PGB
## 33    NM   3.026994         1386         1362   PGB
## 34    NV   2.986928         1314         1282   PGB
## 35    NY   2.908633         1254         1233   PGB
## 36    OH   2.776923         1267         1247   PGB
## 37    OK   3.114570         1462         1430   PGB
## 38    OR   3.094771         1376         1355   PGB
## 39    PA   3.210526         1393         1377   PGB
## 40    RI   2.912857         1323         1307   PGB
## 41    SC   2.825974         1432         1403   PGB
## 42    SD   3.172727         1350         1323   PGB
## 43    TN   2.854962         1129         1114   PGB
```

```
## 44     TX   3.015094          1444          1421   PGB
## 45     UT   2.933571          1193          1171   PGB
## 46     VA   2.741844          1317          1299   PGB
## 47     VT   3.042446          1231          1212   PGB
## 48     WA   3.098780          1559          1525   PGB
## 49     WI   2.848252          1339          1313   PGB
## 50     WV   3.088742          1465          1445   PGB
## 51     WY   3.093233          1221          1198   PGB
```

Create a data combining all metrics from PGA and PGB

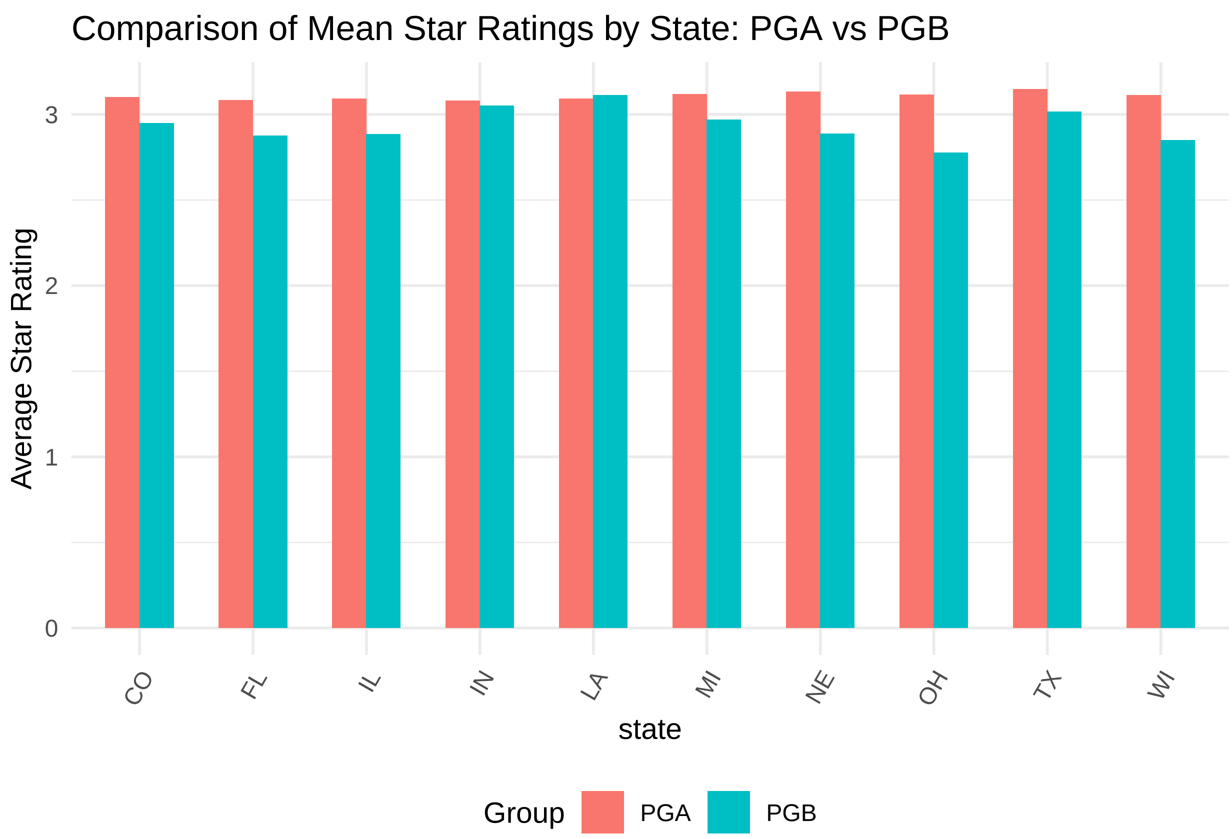
```r
StatesSummary <- bind_rows(summaryPGA, summaryPGB)
head(StatesSummary)
```

```
##   state mean_stars review_count unique_users Group
## 1    AK   3.010593         2102         2051   PGA
## 2    AL   3.017241         1917         1878   PGA
## 3    AR   2.872685         1937         1902   PGA
## 4    AZ   2.939918         2233         2175   PGA
## 5    CA   3.051055         2097         2044   PGA
## 6    CO   3.100000         1924         1882   PGA
```

Comparison of Average Rating

Average will be used as it is more representative for comparison. Top 10 States by average rating of PGA will be used as reference:

```r
top10_average <- top10 %>% inner_join(StatesSummary, by ="state")

ggplot(top10_average, aes(x = state, y = mean_stars.y, fill = Group)) + geom_col(position = "dodge", wi
```

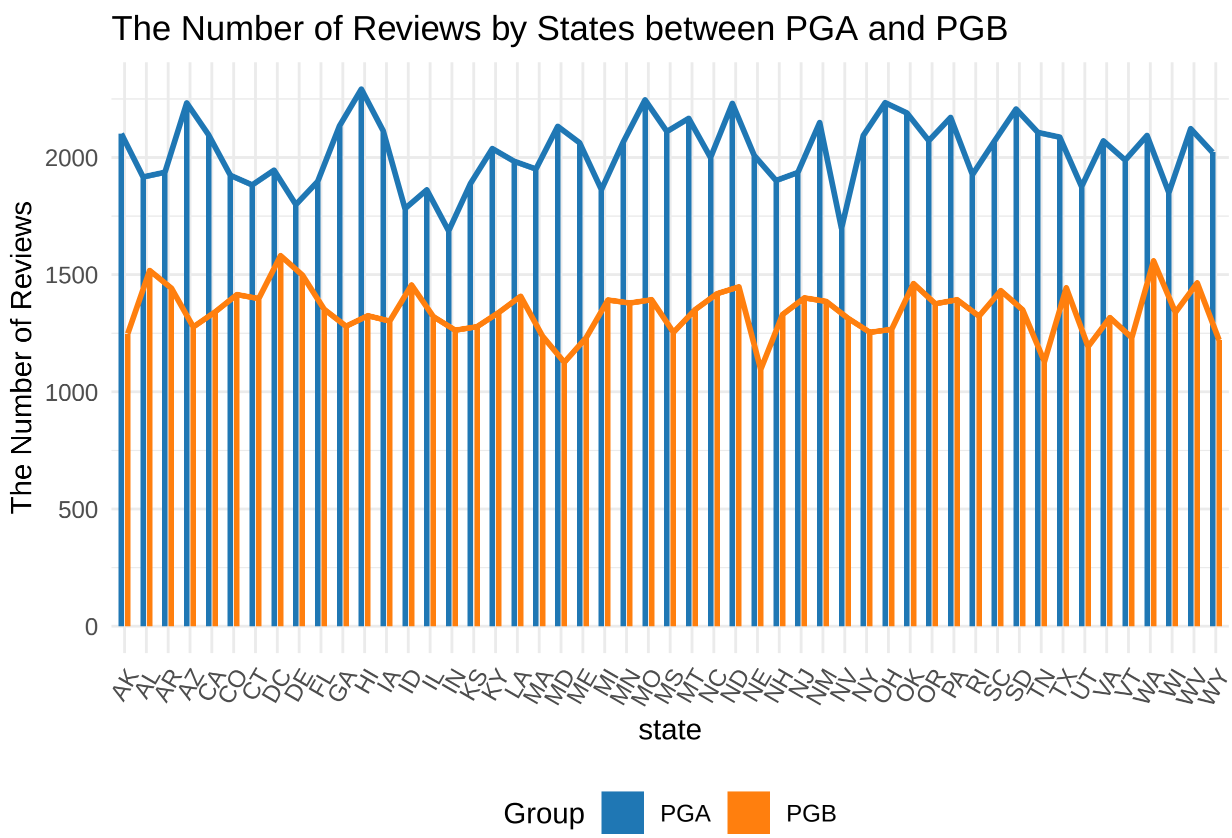


Findings:
- There are slight differences between average rating by top 10 States.
- Only the average rating of LA from PGB is higher than the figure for PGA, while other States of PGA all have higher average stars.

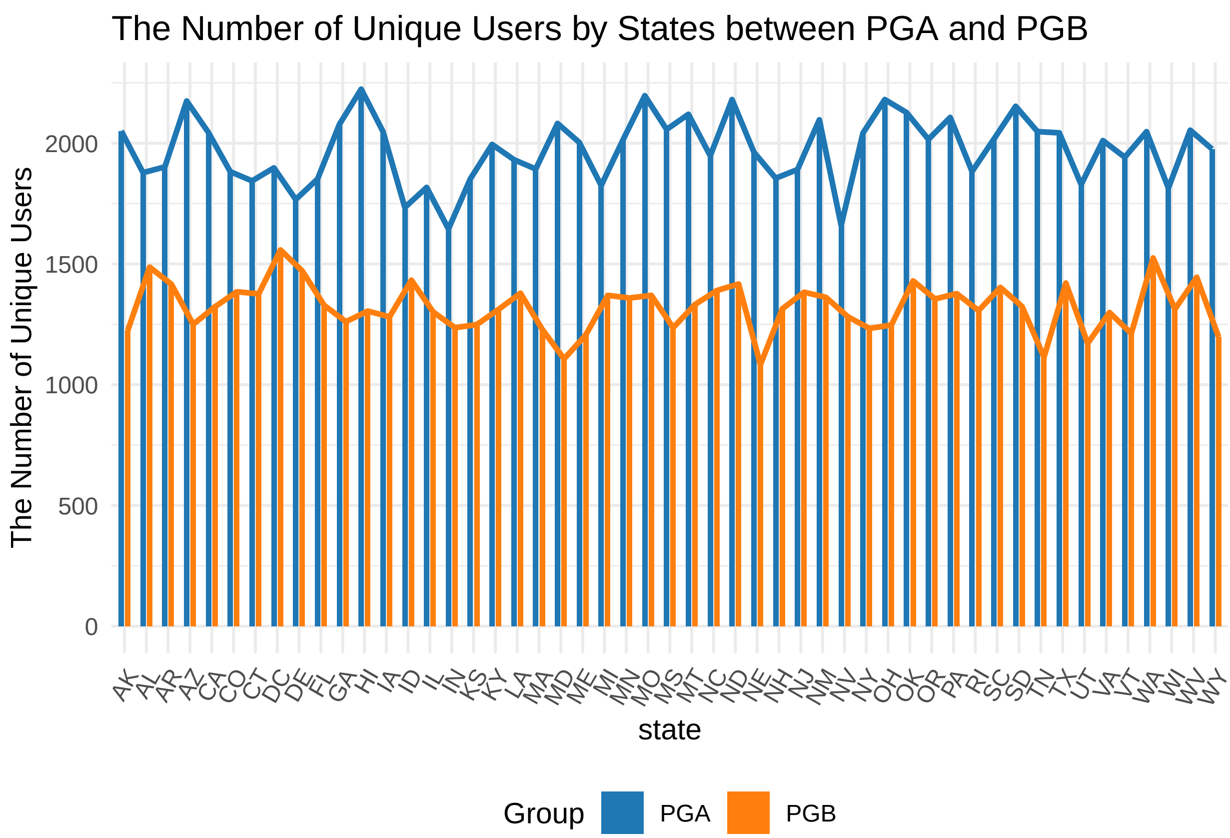
Comparison of Review Count and Unique Users

```r
#Visualisation of review count by 52 States
ggplot(data = StatesSummary, aes(x = state, y = review_count, fill = Group)) +
  geom_col(position = position_dodge(width = 0.6), width = 0.5) +
  geom_line(aes(group = Group, color = Group), position = position_dodge(width = 0.6), size = 1) +
  scale_fill_manual(values = c("PGA" = "#1f77b4", "PGB" = "#ff7f0e")) + # Custom line colors
  scale_color_manual(values = c("PGA" = "#1f77b4", "PGB" = "#ff7f0e")) +
  theme_minimal() + theme(axis.text.x = element_text(angle = 60, hjust = 1), legend.position = "bottom")
  labs(y = "The Number of Reviews", title = "The Number of Reviews by States between PGA and PGB")
```

```
## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
## i Please use `linewidth` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```




```r
#Visualisation of Unique Users by 52 States
ggplot(data = StatesSummary, aes(x = state, y = unique_users, fill = Group)) +
  geom_col(position = position_dodge(width = 0.6), width = 0.5) +
  geom_line(aes(group = Group, color = Group), position = position_dodge(width = 0.6), size = 1) +
  scale_fill_manual(values = c("PGA" = "#1f77b4", "PGB" = "#ff7f0e")) + # Custom line colors
  scale_color_manual(values = c("PGA" = "#1f77b4", "PGB" = "#ff7f0e")) +
  theme_minimal() + theme(axis.text.x = element_text(angle = 60, hjust = 1),
legend.position = "bottom") +
  labs(y = "The Number of Unique Users", title = "The Number of Unique Users by States between PGA and P
```



Findings: Despite the differences in statistics of reviews and unique users, both PGA and PGB tend to have the same pattern of these metrics across the 50 States.

2.10) Conclusions:

In comparison with average rating, there are differences of average rating between PGA and PGB groups. Nevertheless, although the PGA's unique users and review count are higher than those of PGB, both groups have the same patterns in these metrics.

Question 3:

3.1) Dataset selection for analysis

For this question, reviewsUsers will be used for the analysis.

```
head(reviewsUsers)
```

```
##   review_id user_id business_id stars       date
## 1       r_0 u_11073      b_4559     5 2023-02-01
## 2       r_1 u_35221     b_10665     3 2023-03-12
## 3       r_2  u_3710      b_7683     5 2025-02-19
## 4       r_3 u_23891      b_9113     3 2023-01-10
## 5       r_4 u_10374      b_7612     4 2023-01-02
```

```
## 6       r_5 u_30798      b_5793     2 2022-08-21
##
## 1 Audience hour west television. Live central spend machine. Agree would claim behavior table preven
## 2                                                              Summer ability art beat race else large
## 3                                                               Reason range future the chair house TV
## 4                                                              Up change final prepare area difference
## 5                                                                   Size pass including performance sh
## 6                                             Pm yeah laugh necessary else store. Cut fine school phon
##          name review_count average_stars member_since
## 1                       59          4.94         <NA>
## 2 Christopher            7          1.04   2020-10-18
## 3      Rhonda            9          3.72   2020-01-08
## 4        Erik           65          1.60   2021-11-27
## 5 Christopher            3          2.71   2018-01-02
## 6    Danielle           25          3.14   2021-01-24
```

3.2) Top 10 users by the review count, and their average review stars accordingly

```r
#Top 10 users by review count
reviewsTable <- reviewsUsers %>% group_by(user_id, name) %>%
  filter(!is.na(name), name != "") %>% #remove any blank strings or NA in names
  summarise(review_count = n(), avg_starRating = mean(stars, na.rm = TRUE),
          avg_reviewLength = mean(nchar(text), na.rm = TRUE)) #add and calculate new columns -review
```

```
## `summarise()` has grouped output by 'user_id'. You can override using the
## `.groups` argument.
```

```r
Top10_ReviewCount <- reviewsTable %>% arrange(desc(review_count))

#rename the headers
colnames(Top10_ReviewCount) <- c("User Id", "Name", "Review Count", "Average Rating", "Average Review L

Top10_ReviewCount<- Top10_ReviewCount[1:10,] #pick top 10 users only

#Tabulate the summary of the data (kable/kableextra).
kable(Top10_ReviewCount, caption = "Top 10 Users by Number of Reviews and Their Average Rating", digits
  kable_styling(bootstrap_options = c("striped", "hover", "condensed", "responsive"),
                full_width = FALSE,
                position = "center")
```

```
## Warning in attr(x, "align"): 'xfun::attr()' is deprecated.
## Use 'xfun::attr2()' instead.
## See help("Deprecated")
```

```
## Warning in attr(.knitEnv$meta, "knit_meta_id"): 'xfun::attr()' is deprecated.
## Use 'xfun::attr2()' instead.
## See help("Deprecated")
## Warning in attr(.knitEnv$meta, "knit_meta_id"): 'xfun::attr()' is deprecated.
## Use 'xfun::attr2()' instead.
## See help("Deprecated")
## Warning in attr(.knitEnv$meta, "knit_meta_id"): 'xfun::attr()' is deprecated.
```

```
## Use 'xfun::attr2()' instead.
## See help("Deprecated")
## Warning in attr(.knitEnv$meta, "knit_meta_id"): 'xfun::attr()' is deprecated.
## Use 'xfun::attr2()' instead.
## See help("Deprecated")
## Warning in attr(.knitEnv$meta, "knit_meta_id"): 'xfun::attr()' is deprecated.
## Use 'xfun::attr2()' instead.
## See help("Deprecated")
## Warning in attr(.knitEnv$meta, "knit_meta_id"): 'xfun::attr()' is deprecated.
## Use 'xfun::attr2()' instead.
## See help("Deprecated")
## Warning in attr(.knitEnv$meta, "knit_meta_id"): 'xfun::attr()' is deprecated.
## Use 'xfun::attr2()' instead.
## See help("Deprecated")
## Warning in attr(.knitEnv$meta, "knit_meta_id"): 'xfun::attr()' is deprecated.
## Use 'xfun::attr2()' instead.
## See help("Deprecated")
## Warning in attr(.knitEnv$meta, "knit_meta_id"): 'xfun::attr()' is deprecated.
## Use 'xfun::attr2()' instead.
## See help("Deprecated")
## Warning in attr(.knitEnv$meta, "knit_meta_id"): 'xfun::attr()' is deprecated.
## Use 'xfun::attr2()' instead.
## See help("Deprecated")
## Warning in attr(.knitEnv$meta, "knit_meta_id"): 'xfun::attr()' is deprecated.
## Use 'xfun::attr2()' instead.
## See help("Deprecated")
## Warning in attr(.knitEnv$meta, "knit_meta_id"): 'xfun::attr()' is deprecated.
## Use 'xfun::attr2()' instead.
## See help("Deprecated")
## Warning in attr(.knitEnv$meta, "knit_meta_id"): 'xfun::attr()' is deprecated.
## Use 'xfun::attr2()' instead.
## See help("Deprecated")
## Warning in attr(.knitEnv$meta, "knit_meta_id"): 'xfun::attr()' is deprecated.
## Use 'xfun::attr2()' instead.
## See help("Deprecated")

## Warning in attr(x, "align"): 'xfun::attr()' is deprecated.
## Use 'xfun::attr2()' instead.
## See help("Deprecated")

## Warning in attr(x, "format"): 'xfun::attr()' is deprecated.
## Use 'xfun::attr2()' instead.
## See help("Deprecated")
```

Table 4: Top 10 Users by Number of Reviews and Their Average Rating

User Id	Name	Review Count	Average Rating	Average Review Length
u_27070	Rebecca	18	2.833	42.222
u_11551	Christopher	15	3.267	71.333
u_6766	Tracy	15	3.267	58.400
u_11229	Benjamin	14	3.071	63.143

u_14899	Jason	14	2.571	49.643
u_17629	Andrew	14	2.214	53.571
u_22933	Stephanie	14	2.929	58.500
u_27907	Jesse	14	3.429	61.714
u_29224	Rebecca	14	3.500	65.857
u_32335	Barry	14	2.857	47.643

Findings:
- There is less significant differences between review count from top 10 users, while there are considerable differences between their average rating and average texts for review.
- Even customer Rebecca (u_27070) has the highest review count (18), this customer has low average rating and average review length compared to others, indicating a low-quality engagement from them.
- Customer Christopher and Rebecca (u_29224) could be potential customers with high engagement, as they have relative high balance between their review count and average rating for each review. Their average review length, particularly of Christopher is high in comparison to other top 10.
- Customer Andrew has the lowest average stars even has quite high review count, indicating low-quality engagement from them. The company should come up with further strategy improving their engagements.

3.3) Visualisation of their rating distrubtion (using ggplot2)

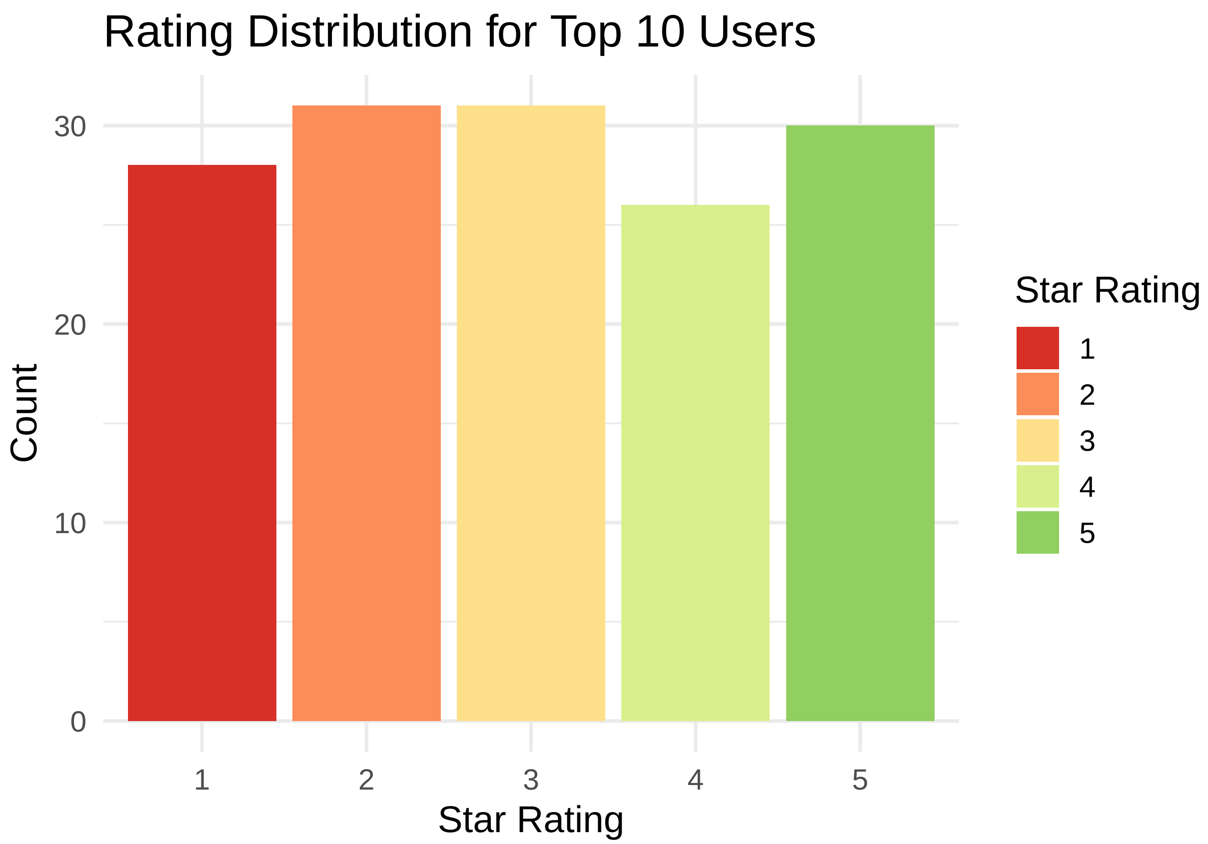
To count the number of each rating from the top 10 users appropriately, the `Top10_ReviewCount` data will left join the `cleaned_reviews` dataset.
Note: Duplicates are acceptable as there is a unique column to identify a particular row - review_id. A user can have multiple reviews

```
detailedRating <- Top10_ReviewCount %>% left_join(cleaned_reviews, by = c("User Id" = "user_id"))
detailedRating #check the output
```

```
## # A tibble: 146 x 10
## # Groups:   User Id [10]
##    `User Id` Name    `Review Count` `Average Rating` `Average Review Length`
##    <chr>     <chr>            <int>            <dbl>                   <dbl>
##  1 u_27070   Rebecca             18             2.83                    42.2
##  2 u_27070   Rebecca             18             2.83                    42.2
##  3 u_27070   Rebecca             18             2.83                    42.2
##  4 u_27070   Rebecca             18             2.83                    42.2
##  5 u_27070   Rebecca             18             2.83                    42.2
##  6 u_27070   Rebecca             18             2.83                    42.2
##  7 u_27070   Rebecca             18             2.83                    42.2
##  8 u_27070   Rebecca             18             2.83                    42.2
##  9 u_27070   Rebecca             18             2.83                    42.2
## 10 u_27070   Rebecca             18             2.83                    42.2
## # i 136 more rows
## # i 5 more variables: review_id <chr>, business_id <chr>, stars <int>,
## #   date <chr>, text <chr>
```

The Visualisation of Rating Distribution

```
ggplot(detailedRating, aes(x = factor(stars), fill = factor(stars))) +
geom_bar() + scale_fill_manual(
values = c("1" = "#d73027","2" = "#fc8d59","3" = "#fee08b","4" = "#d9ef8b","5" = "#91cf60"), name = "Sta
  labs(title = "Rating Distribution for Top 10 Users", x = "Star Rating", y = "Count") + theme_minimal(
```



Findings
- Overall, the engagement from top 10 active users tend be moderate from low rather than positive engagement. - The number of low rating (1-2) tend to be higher than high rating (4-5), indicating a quite low engagement despite being these top 10 active users.
- The number of rating 3 for reviews is the highest across all star levels. This indicates the moderate experience from these top 10 users with the community.

3.4) Conclusions:

Despite being top 10 active users with highest review counts, their user engagement and behaviour is slightly low-quality as more than 60% of their ratings fall from 1 to 3 in rating stars.

Question 4:

Write the code to analyse if there is a major difference between the review behavior of users who joined before and after 2020.

4.1) Data Selection:

For this question, reviewsUsers will be used again for the analysis.

```
head(reviewsUsers)
```

```
##   review_id user_id business_id stars       date
## 1       r_0 u_11073      b_4559     5 2023-02-01
## 2       r_1 u_35221     b_10665     3 2023-03-12
## 3       r_2  u_3710      b_7683     5 2025-02-19
## 4       r_3 u_23891      b_9113     3 2023-01-10
## 5       r_4 u_10374      b_7612     4 2023-01-02
## 6       r_5 u_30798      b_5793     2 2022-08-21
##
## 1 Audience hour west television. Live central spend machine. Agree would claim behavior table preven
## 2                                                               Summer ability art beat race else large
## 3                                                                Reason range future the chair house TV
## 4                                                               Up change final prepare area difference
## 5                                                                    Size pass including performance sh
## 6                                             Pm yeah laugh necessary else store. Cut fine school phon
##          name review_count average_stars member_since
## 1                       59          4.94         <NA>
## 2 Christopher            7          1.04   2020-10-18
## 3      Rhonda            9          3.72   2020-01-08
## 4        Erik           65          1.60   2021-11-27
## 5 Christopher            3          2.71   2018-01-02
## 6   Danielle            25          3.14   2021-01-24
```

4.2) Create 2 groups of users

```
#Form 2 groups of users
before2020 <- reviewsUsers %>% filter(reviewsUsers$member_since < as.Date('2020-01-01'))
head(before2020) #recheck data before proceeding
```

```
##   review_id user_id business_id stars       date
## 1       r_4 u_10374      b_7612     4 2023-01-02
## 2       r_6 u_24924      b_8921     3 2025-01-23
## 3       r_7  u_4847     b_16018     2 2025-04-10
## 4      r_11 u_11140      b_3606     5 2023-04-04
## 5      r_13  u_7012      b_1571     3 2024-07-21
## 6      r_14 u_21010      b_2426     1 2022-07-08
##
## 1
## 2 Today loss experience account commercial individual specific. Hair decide run sell culture evening
## 3
## 4
## 5
## 6
##          name review_count average_stars member_since
## 1 Christopher            3          2.71   2018-01-02
## 2      Ronald           19          1.25   2017-08-24
```

```
## 3      Brenda           67          3.81   2016-03-08
## 4        Mary           26          3.27   2016-06-04
## 5       Karen           83          1.78   2017-11-20
## 6        Gina           59          3.62   2017-12-04
```

```
after2020 <-reviewsUsers %>% filter(reviewsUsers$member_since >= as.Date('2020-01-01'))
head(after2020) #recheck data before proceeding
```

```
##   review_id user_id business_id stars       date
## 1       r_1 u_35221     b_10665     3 2023-03-12
## 2       r_2  u_3710      b_7683     5 2025-02-19
## 3       r_3 u_23891      b_9113     3 2023-01-10
## 4       r_5 u_30798      b_5793     2 2022-08-21
## 5       r_9 u_21910      b_9549     4
## 6      r_10 u_35468     b_16230     2
##                                                                                     text
## 1                                                   Summer ability art beat race else large space.
## 2                                                    Reason range future the chair house TV final.
## 3                                                   Up change final prepare area difference peace.
## 4                                  Pm yeah laugh necessary else store. Cut fine school phone seat.
## 5 Day for participant increase expect next talk source. Image difficult admit compare general say.
## 6                                                                       Car data move live type.
##          name review_count average_stars member_since
## 1 Christopher            7          1.04   2020-10-18
## 2      Rhonda            9          3.72   2020-01-08
## 3        Erik           65          1.60   2021-11-27
## 4    Danielle           25          3.14   2021-01-24
## 5      Robert           40          2.53   2023-05-16
## 6      Alexis           40          1.83   2021-01-19
```

4.3) Compare their star rating behaviour and the length of the reviews (number of characters in the review text).

Create another column called text_count to count the numbers of characters in each review:

```
#Auto change NA or blank strings into 0 in char_count
before2020 <- before2020 %>% mutate(text = if_else(is.na(text) |
                    str_trim(text) == "", "", text),
                  char_count = if_else(text == "", 0L, nchar(text)))

after2020 <- after2020 %>% mutate(text = if_else(is.na(text) |
                    str_trim(text) == "", "", text),
                  char_count = if_else(text == "", 0L, nchar(text)))
```

Combine two user groups data for visualisation

```
#add a column date_category to categorise before and after 2020
merge_data <- reviewsUsers %>% mutate(member_type = if_else(as.Date(member_since) < as.Date("2020-01-01

#add a column char_count the merge_dataset
merge_data<- merge_data %>% mutate(text = if_else(is.na(text) |
                    str_trim(text) == "", "", text),
                  char_count = if_else(text == "", 0L, nchar(text)))
```

```r
#Compare average star rating
aggregate(stars~member_type, merge_data, mean)
```

```
##   member_type    stars
## 1   after2020 3.002616
## 2  before2020 2.997296
```

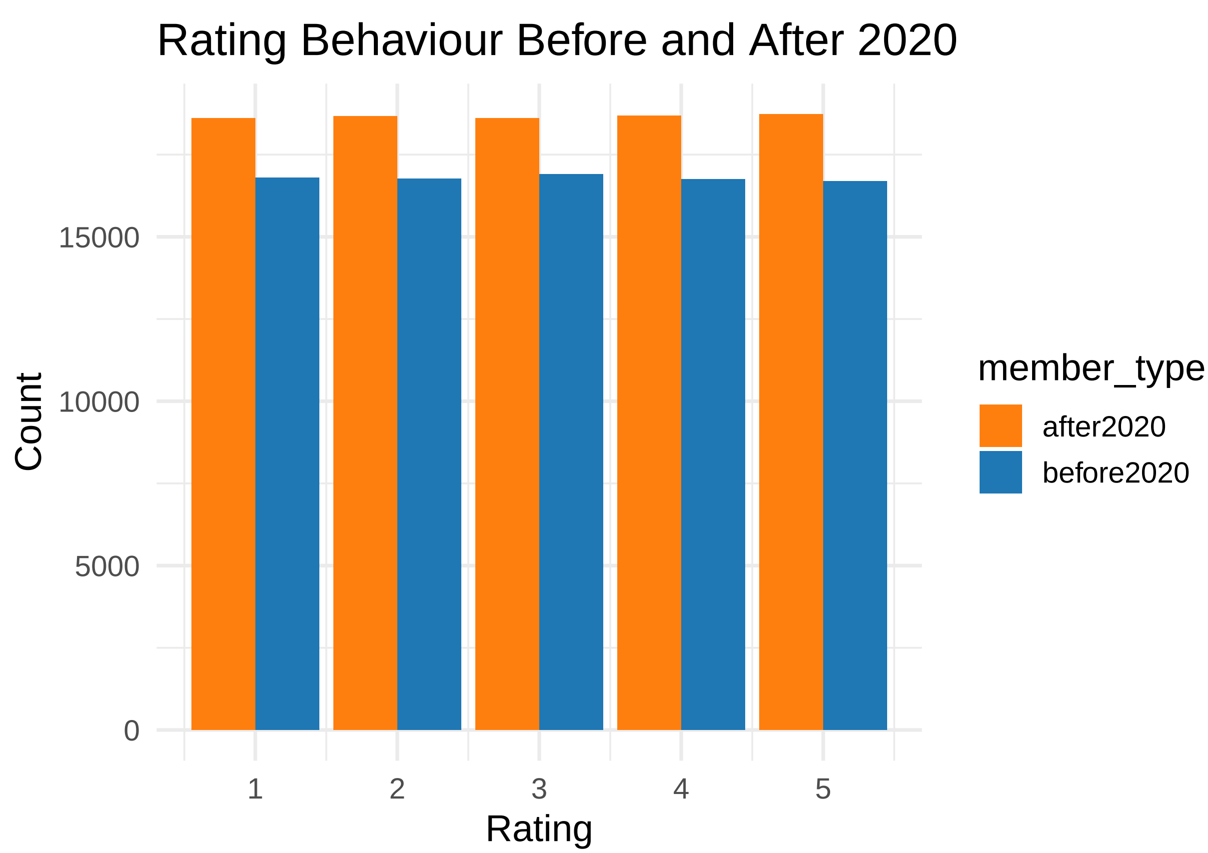
```r
#more detailed with IQR, and min max
summary(after2020$stars)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.000   2.000   3.000   3.003   4.000   5.000
```

```r
summary(before2020$stars)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.000   2.000   3.000   2.997   4.000   5.000
```

```r
ggplot(merge_data, aes(x = stars, fill = member_type)) +
  geom_bar(position = "dodge") +
  labs(title = "Rating Behaviour Before and After 2020", x = "Rating", y = "Count") + scale_fill_manual
  theme_minimal(base_size = 14)
```



Findings
- Users joining after 2020 are more active with the community as their rating count (from 1 to 5) is all higher than those joining before 2020.
- Combining with mean in average star, users joining after 2020 have slightly higher average rating then those before 2020, indicating more engagement from this group.

Summarise review length:

```
#Compare review length
aggregate(char_count ~ member_type, merge_data, mean)
```

```
##   member_type char_count
## 1   after2020   58.94690
## 2  before2020   59.09321
```

```
#more detailed with IQR, and min max
summary(after2020$char_count)
```

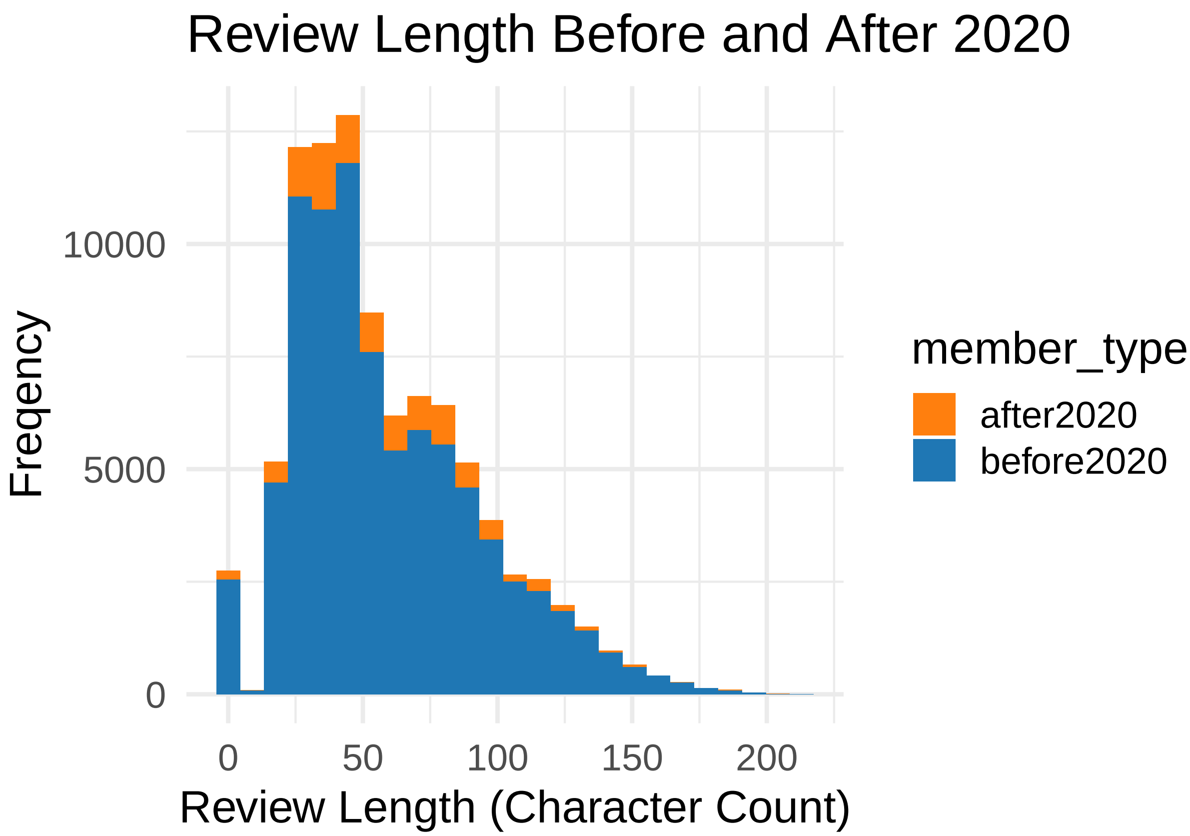
```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    0.00   34.00   50.00   58.95   80.00  212.00
```

```
summary(before2020$char_count)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    0.00   33.00   49.00   59.09   80.00  213.00
```

Histogram visualise the distribution of character count

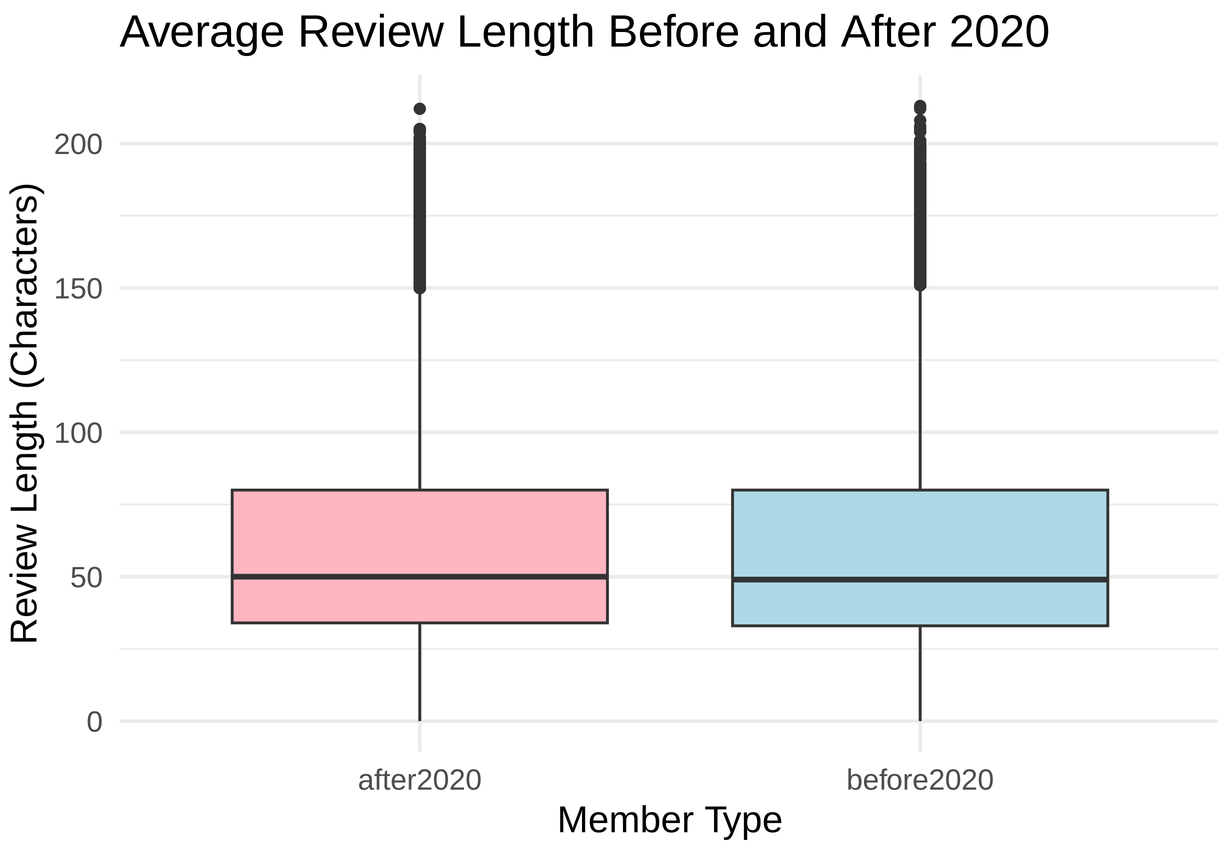
```
ggplot(merge_data, aes(x = char_count, fill = member_type)) +
  geom_histogram(position = "identity", bins = 25) +
  scale_fill_manual(values = c("before2020" = "#1f77b4", "after2020" = "#ff7f0e")) +
  labs(title = "Review Length Before and After 2020",
       x = "Review Length (Character Count)",
       y = "Freqency") +
  theme_minimal(base_size = 17)
```



Findings
- Despite having the same patterns of distribution, members joining after 2020 tend to review more requently than those before 2020, indicating active behaviours from after 2020 members.
- The gaps are particularly clear when review length starting from 25 to 50.

4.4) Visualise the average review length by the two groups.

```
#visualise the average review length
ggplot(merge_data, aes(x = member_type, y = char_count, fill = member_type)) + geom_boxplot() + labs(
title = "Average Review Length Before and After 2020", x = "Member Type", y = "Review Length (Characters
  theme_minimal(base_size = 14) + theme(legend.position = "none")
```



Findings:
- Both of the groups has nearly equal mean in average review length. There distribution and IQR also have high similarity as well.
- Thus, there is no major differences between the distribution of average review length from users joining before and after 2020.

4.5) Conclusions:

User engagement and behaviour of those joining after 2020 tend to be higher than those before 2020. They reviewed more frequently, and are more likely to have higher review length on each of their review.