Quantium Virtual Internship - Retail Strategy and Analytics - Task 1

Solution template for Task 1

This file is a solution template for the Task 1 of the Quantium Virtual Internship. It will walk you through the analysis, providing the scaffolding for your solution with gaps left for you to fill in yourself. Look for comments that say "over to you" for places where you need to add your own code! Often, there will be hints about what to do or what function to use in the text leading up to a code block - if you need a bit of extra help on how to use a function, the internet has many excellent resources on R coding, which you can find using your favourite search engine. ## Load required libraries and datasets Note that you will need to install these libraries if you have never used these before.

```
#### Example code to install packages
install.packages("ggmosaic", repos = "https://cloud.r-project.org")
#### Load required libraries
library(data.table)
library(ggplot2)
library(ggmosaic)
library(readr)
```

```
#### Point the filePath to where you have downloaded the datasets to and
#### assign the data files to data.tables
# over to you! fill in the path to your working directory. If you are on a Windows
↪ machine, you will need to use forward slashes (/) instead of backshashes (\)
getwd()  # To check the current working directory
```

```
## [1] "/Users/huilinng/Desktop/Forage/Quantum Data Analytics"
```

```
setwd("/Users/huilinng/Desktop/Forage/Quantum Data Analytics") # To set the working
↪ directory
transactionData <- fread(paste0("QVI_transaction_data.csv"))
customerData <- fread(paste0("QVI_purchase_behaviour.csv"))
```

Exploratory data analysis

The first step in any analysis is to first understand the data. Let's take a look at each of the datasets provided. ### Examining transaction data We can use `str()` to look at the format of each column and see a sample of the data. As we have read in the dataset as a `data.table` object, we can also run `transactionData` in the console to see a sample of the data or use `head(transactionData)` to look at the first 10 rows. Let's check if columns we would expect to be numeric are in numeric form and date columns are in date format.

```r
#### Examine transaction data
# Over to you! Examine the data using one or more of the methods described above.
summary(transactionData)
```

```
##       DATE          STORE_NBR     LYLTY_CARD_NBR        TXN_ID
##  Min.   :43282   Min.   :  1.0   Min.   :   1000   Min.   :      1
##  1st Qu.:43373   1st Qu.: 70.0   1st Qu.:  70021   1st Qu.:  67602
##  Median :43464   Median :130.0   Median : 130358   Median : 135138
##  Mean   :43464   Mean   :135.1   Mean   : 135550   Mean   : 135158
##  3rd Qu.:43555   3rd Qu.:203.0   3rd Qu.: 203094   3rd Qu.: 202701
##  Max.   :43646   Max.   :272.0   Max.   :2373711   Max.   :2415841
##     PROD_NBR       PROD_NAME           PROD_QTY         TOT_SALES
##  Min.   :  1.00   Length:264836      Min.   :  1.000   Min.   :  1.500
##  1st Qu.: 28.00   Class :character   1st Qu.:  2.000   1st Qu.:  5.400
##  Median : 56.00   Mode  :character   Median :  2.000   Median :  7.400
##  Mean   : 56.58                      Mean   :  1.907   Mean   :  7.304
##  3rd Qu.: 85.00                      3rd Qu.:  2.000   3rd Qu.:  9.200
##  Max.   :114.00                      Max.   :200.000   Max.   :650.000
```

We can see that the date column is in an integer format. Let's change this to a date format.

```r
#### Convert DATE column to a date format
#### A quick search online tells us that CSV and Excel integer dates begin on 30 Dec 1899
transactionData$DATE <- as.Date(transactionData$DATE, origin = "1899-12-30")
```

We should check that we are looking at the right products by examining PROD_NAME.

```r
#### Examine PROD_NAME
# Over to you! Generate a summary of the PROD_NAME column.

library(data.table)
prod_name_freq <- transactionData[, .N, by = PROD_NAME]
prod_name_freq <- prod_name_freq[order(-N)]
print(head(prod_name_freq, 10))
```

```
##                                    PROD_NAME     N
##                                       <char> <int>
##   1:   Kettle Mozzarella   Basil & Pesto 175g  3304
##   2: Kettle Tortilla ChpsHny&Jlpno Chili 150g  3296
##   3: Cobs Popd Swt/Chlli &Sr/Cream Chips 110g  3269
##   4:   Tyrrells Crisps     Ched & Chives 165g  3268
##   5:            Cobs Popd Sea Salt  Chips 110g  3265
##   6:              Kettle 135g Swt Pot Sea Salt  3257
##   7:             Tostitos Splash Of  Lime 175g  3252
##   8: Infuzions Thai SweetChili PotatoMix 110g  3242
##   9:   Smiths Crnkle Chip  Orgnl Big Bag 380g  3233
##  10:     Thins Potato Chips  Hot & Spicy 175g  3229
```

Looks like we are definitely looking at potato chips but how can we check that these are all chips? We can do some basic text analysis by summarising the individual words in the product name.

```r
#### Examine the words in PROD_NAME to see if there are any incorrect entries
#### such as products that are not chips
productWords <- data.table(unlist(strsplit(unique(transactionData[, PROD_NAME]), "
")))
setnames(productWords, 'words')
```

As we are only interested in words that will tell us if the product is chips or not, let's remove all words with digits and special characters such as '&' from our set of product words. We can do this using `grepl()`.

```r
# Over to you! Remove digits, and special characters, and then sort the distinct words by
↪  frequency of occurrence.
library(stringr)
#### Removing digits
#### Removing special characters
# Define a function to clean and filter words
filter_words <- function(words) {
  # Create a logical vector to identify words with digits or special characters
  keep_word <- grepl("^[a-zA-Z]+$", words)  # Only keep words with alphabetic characters

  # Filter out words with digits or special characters
  filtered_words <- words[keep_word]

  return(filtered_words)
}

# Clean the PROD_NAME column
transactionData[, CLEANED_PROD_NAME := gsub("[0-9]", "", PROD_NAME)]
transactionData[, CLEANED_PROD_NAME := gsub("[^a-zA-Z\\s]", "", CLEANED_PROD_NAME)]
transactionData[, CLEANED_PROD_NAME := str_squish(CLEANED_PROD_NAME)]

# Split cleaned product names into words
words <- unlist(str_split(transactionData$CLEANED_PROD_NAME, "\\s+"))

# Remove empty words
words <- words[words != ""]

# Filter words using the filter_words function
filtered_words <- filter_words(words)

#### Let's look at the most common words by counting the number of times a word appears
↪  and
#### sorting them by this frequency in order of highest to lowest frequency
# Create a data.table of word frequencies
word_freq <- data.table(word = filtered_words)[, .N, by = word]

# Sort by frequency in descending order
word_freq <- word_freq[order(-N)]
print(word_freq)
```

```
##                                  word     N
##                                <char> <int>
##    1:      KettleMozzarellaBasilPestog  3304
```

```
##   2: KettleTortillaChpsHnyJlpnoChilig  3296
##   3:    CobsPopdSwtChlliSrCreamChipsg  3269
##   4:        TyrrellsCrispsChedChivesg  3268
##   5:            CobsPopdSeaSaltChipsg  3265
##  ---
## 110:                    RRDPcSeaSaltg  1431
## 111:           WoolworthsMediumSalsag  1430
## 112:        NCCSourCreamGardenChivesg  1419
## 113:          FrenchFriesPotatoChipsg  1418
## 114:            WWCrinkleCutOriginalg  1410
```

There are salsa products in the dataset but we are only interested in the chips category, so let's remove these.

```r
#### Remove salsa products
transactionData[, SALSA := grepl("salsa", tolower(PROD_NAME))]
transactionData <- transactionData[SALSA == FALSE, ][, SALSA := NULL]
```

Next, we can use `summary()` to check summary statistics such as mean, min and max values for each feature to see if there are any obvious outliers in the data and if there are any nulls in any of the columns (`NA's : number of nulls` will appear in the output if there are any nulls).

```r
#### Summarise the data to check for nulls and possible outliers
# Over to you!
summary(transactionData)
```

```
##       DATE              STORE_NBR     LYLTY_CARD_NBR        TXN_ID
##  Min.   :2018-07-01   Min.   :  1.0   Min.   :   1000   Min.   :      1
##  1st Qu.:2018-09-30   1st Qu.: 70.0   1st Qu.:  70015   1st Qu.:  67569
##  Median :2018-12-30   Median :130.0   Median : 130367   Median : 135183
##  Mean   :2018-12-30   Mean   :135.1   Mean   : 135531   Mean   : 135131
##  3rd Qu.:2019-03-31   3rd Qu.:203.0   3rd Qu.: 203084   3rd Qu.: 202654
##  Max.   :2019-06-30   Max.   :272.0   Max.   :2373711   Max.   :2415841
##     PROD_NBR       PROD_NAME           PROD_QTY         TOT_SALES
##  Min.   :  1.00   Length:246742      Min.   :  1.000   Min.   :  1.700
##  1st Qu.: 26.00   Class :character   1st Qu.:  2.000   1st Qu.:  5.800
##  Median : 53.00   Mode  :character   Median :  2.000   Median :  7.400
##  Mean   : 56.35                      Mean   :  1.908   Mean   :  7.321
##  3rd Qu.: 87.00                      3rd Qu.:  2.000   3rd Qu.:  8.800
##  Max.   :114.00                      Max.   :200.000   Max.   :650.000
##  CLEANED_PROD_NAME
##  Length:246742
##  Class :character
##  Mode  :character
##
##
##
```

There are no nulls in the columns but product quantity appears to have an outlier which we should investigate further. Let's investigate further the case where 200 packets of chips are bought in one transaction.

```r
#### Filter the dataset to find the outlier
# Over to you! Use a filter to examine the transactions in question.
# Filter transactions where PROD_QTY is 200 or more
outlier_transactions <- transactionData[PROD_QTY >= 200]

# Display the outlier transactions
print(outlier_transactions)
```

```
##          DATE STORE_NBR LYLTY_CARD_NBR TXN_ID PROD_NBR
##        <Date>     <int>          <int>  <int>    <int>
## 1: 2018-08-19       226         226000 226201        4
## 2: 2019-05-20       226         226000 226210        4
##                             PROD_NAME PROD_QTY TOT_SALES     CLEANED_PROD_NAME
##                                <char>    <int>     <num>                <char>
## 1: Dorito Corn Chp     Supreme 380g      200       650 DoritoCornChpSupremeg
## 2: Dorito Corn Chp     Supreme 380g      200       650 DoritoCornChpSupremeg
```

There are two transactions where 200 packets of chips are bought in one transaction and both of these transactions were by the same customer.

```r
#### Let's see if the customer has had other transactions
# Over to you! Use a filter to see what other transactions that customer made.
# Get the customer ID and quantity of interest
customer_id <- unique(outlier_transactions$LYLTY_CARD_NBR)
quantity_of_interest <- 200

# Filter for transactions with 200 packets by this customer
specific_transactions <- transactionData[PROD_QTY == quantity_of_interest &
 ↪ LYLTY_CARD_NBR == customer_id]

# Display the specific transactions
print("Specific transactions involving 200 packets:")
```

```
## [1] "Specific transactions involving 200 packets:"
```

```r
print(specific_transactions)
```

```
##          DATE STORE_NBR LYLTY_CARD_NBR TXN_ID PROD_NBR
##        <Date>     <int>          <int>  <int>    <int>
## 1: 2018-08-19       226         226000 226201        4
## 2: 2019-05-20       226         226000 226210        4
##                             PROD_NAME PROD_QTY TOT_SALES     CLEANED_PROD_NAME
##                                <char>    <int>     <num>                <char>
## 1: Dorito Corn Chp     Supreme 380g      200       650 DoritoCornChpSupremeg
## 2: Dorito Corn Chp     Supreme 380g      200       650 DoritoCornChpSupremeg
```

It looks like this customer has only had the two transactions over the year and is not an ordinary retail customer. The customer might be buying chips for commercial purposes instead. We'll remove this loyalty card number from further analysis.

```r
#### Filter out the customer based on the loyalty card number
# Over to you!
# Exclude transactions where LYLTY_CARD_NBR matches the customer_id
filtered_transactions <- transactionData[!(LYLTY_CARD_NBR %in% customer_id)]

#### Re-examine transaction data
# Over to you!
# Display the filtered transactions
print("Filtered transactions (excluding the specific customer):")
```

```
## [1] "Filtered transactions (excluding the specific customer):"
```

```r
summary(filtered_transactions)
```

```
##       DATE              STORE_NBR     LYLTY_CARD_NBR        TXN_ID
##  Min.   :2018-07-01   Min.   :  1.0   Min.   :   1000   Min.   :      1
##  1st Qu.:2018-09-30   1st Qu.: 70.0   1st Qu.:  70015   1st Qu.:  67569
##  Median :2018-12-30   Median :130.0   Median : 130367   Median : 135182
##  Mean   :2018-12-30   Mean   :135.1   Mean   : 135530   Mean   : 135130
##  3rd Qu.:2019-03-31   3rd Qu.:203.0   3rd Qu.: 203083   3rd Qu.: 202652
##  Max.   :2019-06-30   Max.   :272.0   Max.   :2373711   Max.   :2415841
##     PROD_NBR       PROD_NAME            PROD_QTY       TOT_SALES
##  Min.   :  1.00   Length:246740      Min.   :1.000   Min.   : 1.700
##  1st Qu.: 26.00   Class :character   1st Qu.:2.000   1st Qu.: 5.800
##  Median : 53.00   Mode  :character   Median :2.000   Median : 7.400
##  Mean   : 56.35                      Mean   :1.906   Mean   : 7.316
##  3rd Qu.: 87.00                      3rd Qu.:2.000   3rd Qu.: 8.800
##  Max.   :114.00                      Max.   :5.000   Max.   :29.500
##  CLEANED_PROD_NAME
##  Length:246740
##  Class :character
##  Mode  :character
##
##
##
```

That's better. Now, let's look at the number of transaction lines over time to see if there are any obvious data issues such as missing data.

```r
#### Count the number of transactions by date
# Over to you! Create a summary of transaction count by date.
# Count the number of transactions by date
transaction_count_by_date <- transactionData[, .N, by = DATE]

# Rename the columns for clarity
setnames(transaction_count_by_date, old = c("DATE", "N"), new = c("Date",
 ↪  "Transaction_Count"))

# Display the summary
print("Transaction count by date:")
```

```
## [1] "Transaction count by date:"
```

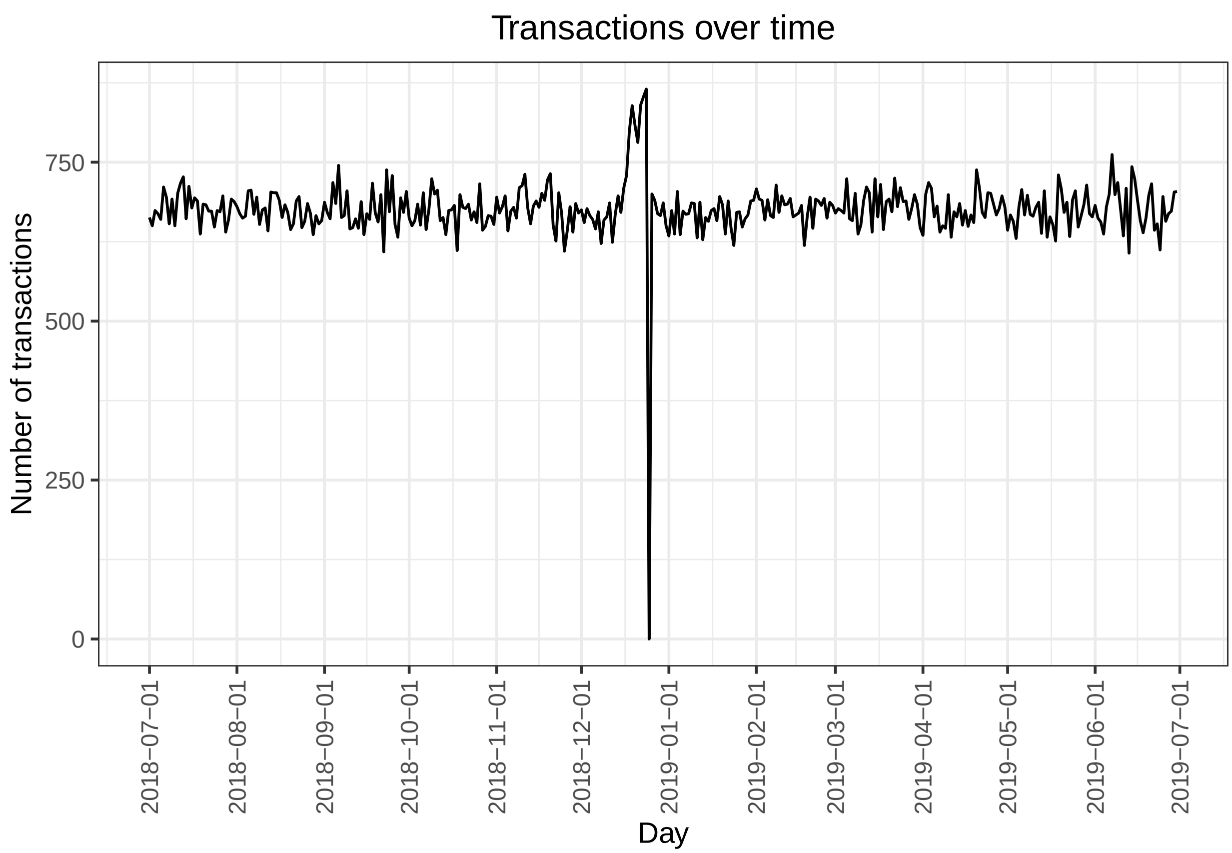
```r
print(transaction_count_by_date)
```

```
##            Date Transaction_Count
##          <Date>             <int>
##   1: 2018-10-17               682
##   2: 2019-05-14               705
##   3: 2019-05-20               708
##   4: 2018-08-17               663
##   5: 2018-08-18               683
##  ---
## 360: 2018-12-08               622
## 361: 2019-01-30               689
## 362: 2019-02-09               671
## 363: 2018-08-31               658
## 364: 2019-02-12               684
```

There's only 364 rows, meaning only 364 dates which indicates a missing date. Let's create a sequence of dates from 1 Jul 2018 to 30 Jun 2019 and use this to create a chart of number of transactions over time to find the missing date.

```r
#### Create a sequence of dates and join this the count of transactions by date

# Create a sequence of dates from 1 Jul 2018 to 30 Jun 2019
date_sequence <- seq.Date(from = as.Date("2018-07-01"), to = as.Date("2019-06-30"), by =
↪  "day")

# Convert to data.table
date_sequence_dt <- data.table(DATE = date_sequence)

# Count the number of transactions by date
transaction_count_by_date <- transactionData[, .N, by = DATE]

# Rename columns for clarity
setnames(transaction_count_by_date, old = c("DATE", "N"), new = c("Date",
↪  "Transaction_Count"))

# Merge the date sequence with the transaction count data
transactions_by_day <- merge(date_sequence_dt, transaction_count_by_date, by.x = "DATE",
↪  by.y = "Date", all.x = TRUE)

# Fill NA values with 0 (indicating no transactions on those dates)
transactions_by_day[is.na(Transaction_Count), Transaction_Count := 0]


# Over to you - create a column of dates that includes every day from 1 Jul 2018 to 30
↪  Jun 2019, and join it onto the data to fill in the missing day.

#### Setting plot themes to format graphs
theme_set(theme_bw())
theme_update(plot.title = element_text(hjust = 0.5))
#### Plot transactions over time
```

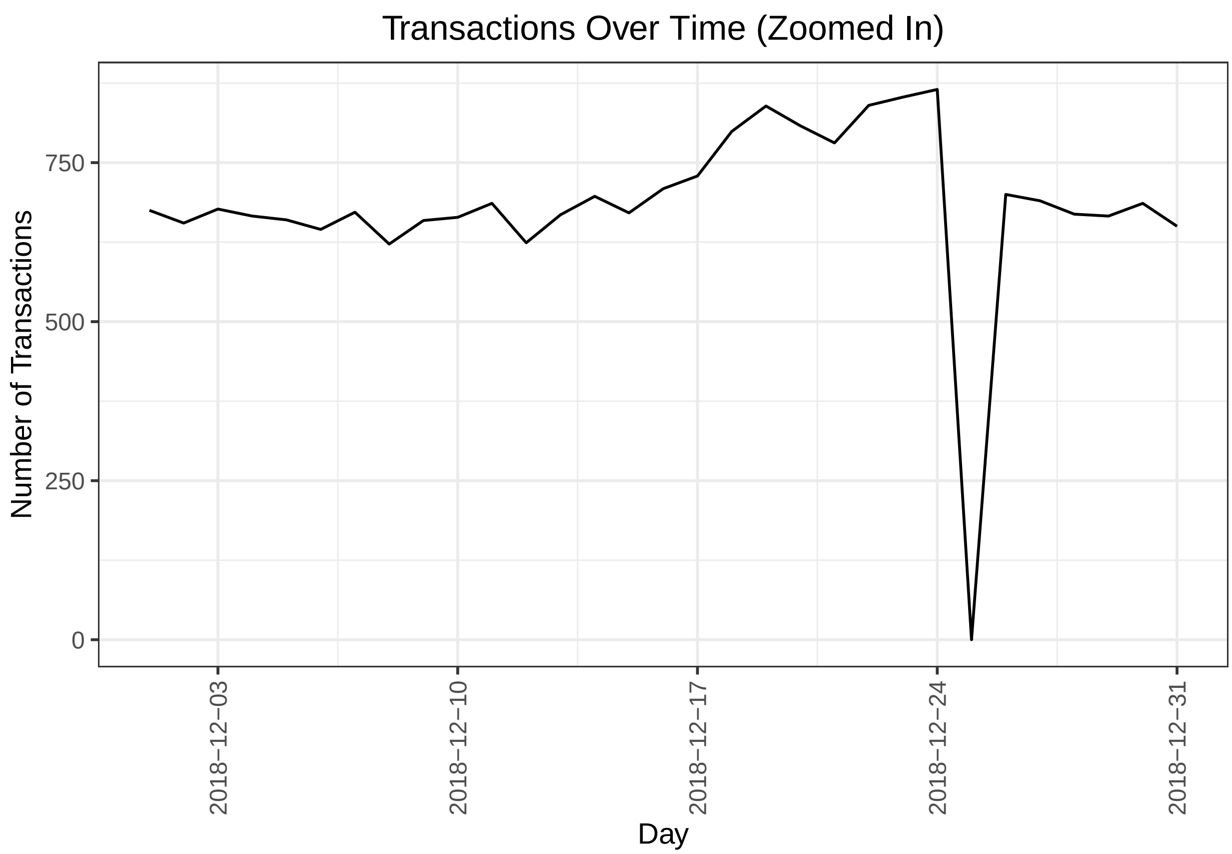
```r
ggplot(transactions_by_day, aes(x = DATE, y = Transaction_Count)) +
 geom_line() +
 labs(x = "Day", y = "Number of transactions", title = "Transactions over time") +
 scale_x_date(breaks = "1 month") +
 theme(axis.text.x = element_text(angle = 90, vjust = 0.5))
```



We can see that there is an increase in purchases in December and a break in late December. Let's zoom in on this.

```r
#### Filter to December and look at individual days
# Over to you - recreate the chart above zoomed in to the relevant dates.
# Plot transactions over time, zoomed in on December and late December
theme_set(theme_bw())
theme_update(plot.title = element_text(hjust = 0.5))

ggplot(transactions_by_day, aes(x = DATE, y = Transaction_Count)) +
  geom_line() +
  labs(x = "Day", y = "Number of Transactions", title = "Transactions Over Time (Zoomed
   ↪  In)") +
  scale_x_date(
    breaks = "1 week",  # Set the breaks to a weekly interval for better granularity
    limits = as.Date(c("2018-12-01", "2018-12-31"))  # Zoom in on December 2018
  ) +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5))
```

```
## Warning: Removed 334 rows containing missing values or values outside the scale range
## (`geom_line()`).
```



We can see that the increase in sales occurs in the lead-up to Christmas and that there are zero sales on Christmas day itself. This is due to shops being closed on Christmas day. Now that we are satisfied that the data no longer has outliers, we can move on to creating other features such as brand of chips or pack size from PROD_NAME. We will start with pack size.

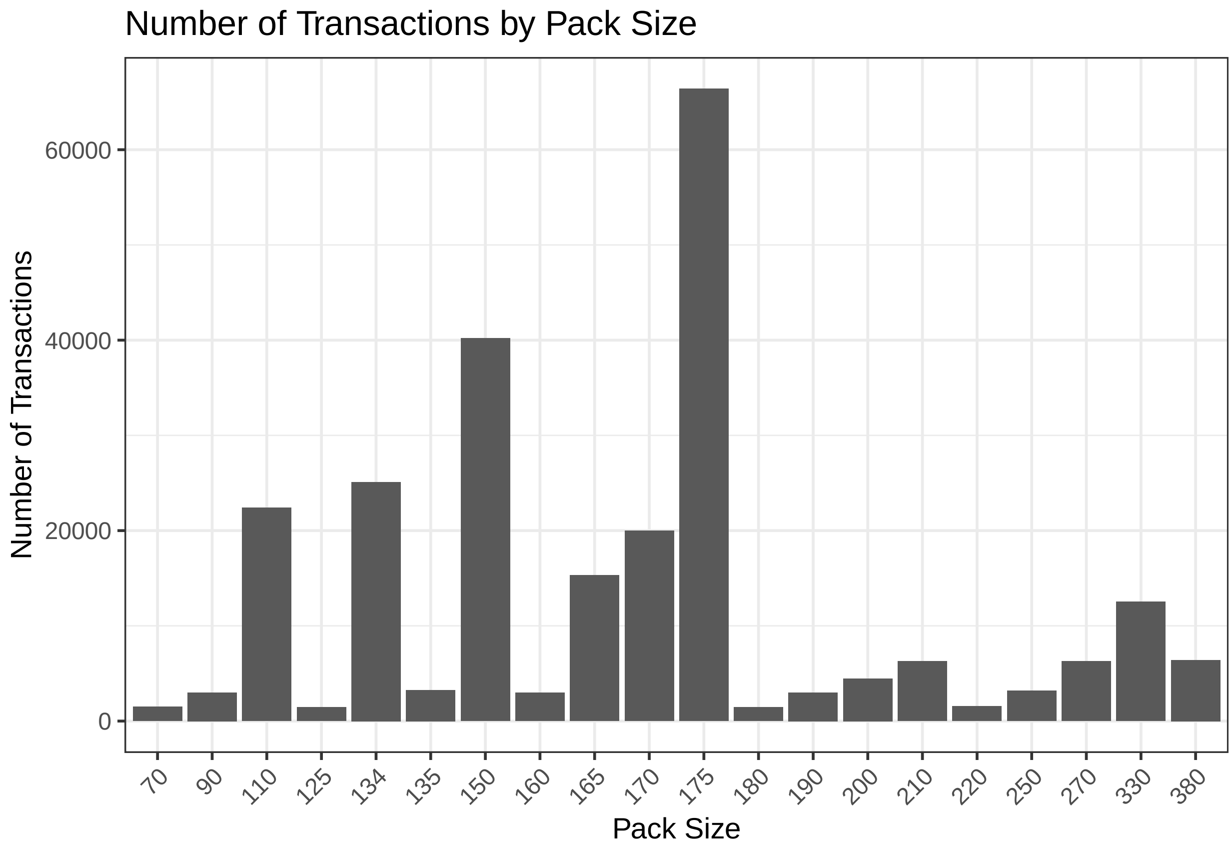
```
#### Pack size
#### We can work this out by taking the digits that are in PROD_NAME
transactionData[, PACK_SIZE := parse_number(PROD_NAME)]
#### Always check your output
#### Let's check if the pack sizes look sensible
transactionData[, .N, PACK_SIZE][order(PACK_SIZE)]
```

```
##     PACK_SIZE     N
##         <num> <int>
##  1:        70  1507
##  2:        90  3008
##  3:       110 22387
##  4:       125  1454
##  5:       134 25102
##  6:       135  3257
##  7:       150 40203
```

```
##  8:       160  2970
##  9:       165 15297
## 10:       170 19983
## 11:       175 66390
## 12:       180  1468
## 13:       190  2995
## 14:       200  4473
## 15:       210  6272
## 16:       220  1564
## 17:       250  3169
## 18:       270  6285
## 19:       330 12540
## 20:       380  6418
##     PACK_SIZE     N
```

The largest size is 380g and the smallest size is 70g - seems sensible!

```r
#### Let's plot a histogram of PACK_SIZE since we know that it is a categorical variable
↪  and not a continuous variable even though it is numeric.
# Over to you! Plot a histogram showing the number of transactions by pack size.
# Convert PACK_SIZE to a factor to ensure it's treated as a categorical variable
transactionData[, PACK_SIZE := as.factor(PACK_SIZE)]

# Create a bar plot showing the number of transactions by pack size
ggplot(transactionData, aes(x = PACK_SIZE)) +
  geom_bar() +
  labs(x = "Pack Size", y = "Number of Transactions", title = "Number of Transactions by
  ↪  Pack Size") +
  theme_bw() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```



Pack sizes created look reasonable. Now to create brands, we can use the first word in PROD_NAME to work out the brand name…

```r
#### Brands
# Over to you! Create a column which contains the brand of the product, by extracting it
↪  from the product name.
#### Checking brands
# Over to you! Check the results look reasonable.

# Extract the first word from PROD_NAME to create the brand name
transactionData[, BRAND := tstrsplit(PROD_NAME, " ")[[1]]]

# Check the results to ensure the brands look reasonable
# View a sample of the data with the new BRAND column
head(transactionData[, .(PROD_NAME, BRAND)])
```

```
##                                   PROD_NAME   BRAND
##                                      <char>  <char>
## 1:   Natural Chip        Compny SeaSalt175g Natural
## 2:                 CCs Nacho Cheese    175g     CCs
## 3:   Smiths Crinkle Cut  Chips Chicken 170g  Smiths
## 4:   Smiths Chip Thinly  S/Cream&Onion 175g  Smiths
## 5: Kettle Tortilla ChpsHny&Jlpno Chili 150g  Kettle
## 6: Smiths Crinkle Chips Salt & Vinegar 330g  Smiths
```

Some of the brand names look like they are of the same brands - such as RED and RRD, which are both Red Rock Deli chips. Let's combine these together.

```r
#### Clean brand names transactionData[BRAND == "RED", BRAND := "RRD"]
unique(transactionData$BRAND)
```

```
##  [1] "Natural"    "CCs"        "Smiths"     "Kettle"     "Grain"
##  [6] "Doritos"    "Twisties"   "WW"         "Thins"      "Burger"
## [11] "NCC"        "Cheezels"   "Infzns"     "Red"        "Pringles"
## [16] "Dorito"     "Infuzions"  "Smith"      "GrnWves"    "Tyrrells"
## [21] "Cobs"       "French"     "RRD"        "Tostitos"   "Cheetos"
## [26] "Woolworths" "Snbts"      "Sunbites"
```

```r
# Over to you! Add any additional brand adjustments you think may be required.
#### Check again
# Over to you! Check the results look reasonable.
# Add any additional brand adjustments as necessary
transactionData[BRAND %in% c("Natural", "Nat"), BRAND := "NATURAL"]
transactionData[BRAND %in% c("CCs", "CC"), BRAND := "CCS"]
transactionData[BRAND %in% c("Smiths", "Smith"), BRAND := "Smith"]

# Check the results to ensure the brands look reasonable
# View a sample of the data with the cleaned BRAND column
head(transactionData[, .(PROD_NAME, BRAND)], 10)
```

```
##                                     PROD_NAME   BRAND
##                                        <char>  <char>
##  1:   Natural Chip        Compny SeaSalt175g NATURAL
##  2:                 CCs Nacho Cheese    175g     CCS
##  3:   Smiths Crinkle Cut  Chips Chicken 170g   Smith
##  4:   Smiths Chip Thinly  S/Cream&Onion 175g   Smith
##  5: Kettle Tortilla ChpsHny&Jlpno Chili 150g  Kettle
##  6: Smiths Crinkle Chips Salt & Vinegar 330g   Smith
##  7:    Grain Waves         Sweet Chilli 210g   Grain
##  8:  Doritos Corn Chip Mexican Jalapeno 150g Doritos
##  9:    Grain Waves Sour    Cream&Chives 210G   Grain
## 10: Smiths Crinkle Chips Salt & Vinegar 330g   Smith
```

```r
#Check for unique brand names to confirm cleaning
unique(transactionData$BRAND)
```

```
##  [1] "NATURAL"    "CCS"        "Smith"      "Kettle"     "Grain"
##  [6] "Doritos"    "Twisties"   "WW"         "Thins"      "Burger"
## [11] "NCC"        "Cheezels"   "Infzns"     "Red"        "Pringles"
## [16] "Dorito"     "Infuzions"  "GrnWves"    "Tyrrells"   "Cobs"
## [21] "French"     "RRD"        "Tostitos"   "Cheetos"    "Woolworths"
## [26] "Snbts"      "Sunbites"
```

Examining customer data

Now that we are happy with the transaction dataset, let's have a look at the customer dataset.

```r
#### Examining customer data
# Over to you! Do some basic summaries of the dataset, including distributions of any key
↪  columns.
# View the structure and summary of the customerData
str(customerData)
```

```
## Classes 'data.table' and 'data.frame': 72637 obs. of 3 variables:
## $ LYLTY_CARD_NBR : int 1000 1002 1003 1004 1005 1007 1009 1010 1011 1012 ...
## $ LIFESTAGE : chr "YOUNG SINGLES/COUPLES" "YOUNG SINGLES/COUPLES" "YOUNG
FAMILIES" "OLDER SINGLES/COUPLES" ...
## $ PREMIUM_CUSTOMER: chr "Premium" "Mainstream" "Budget" "Mainstream" ...
## - attr(*, ".internal.selfref")=<externalptr>
```

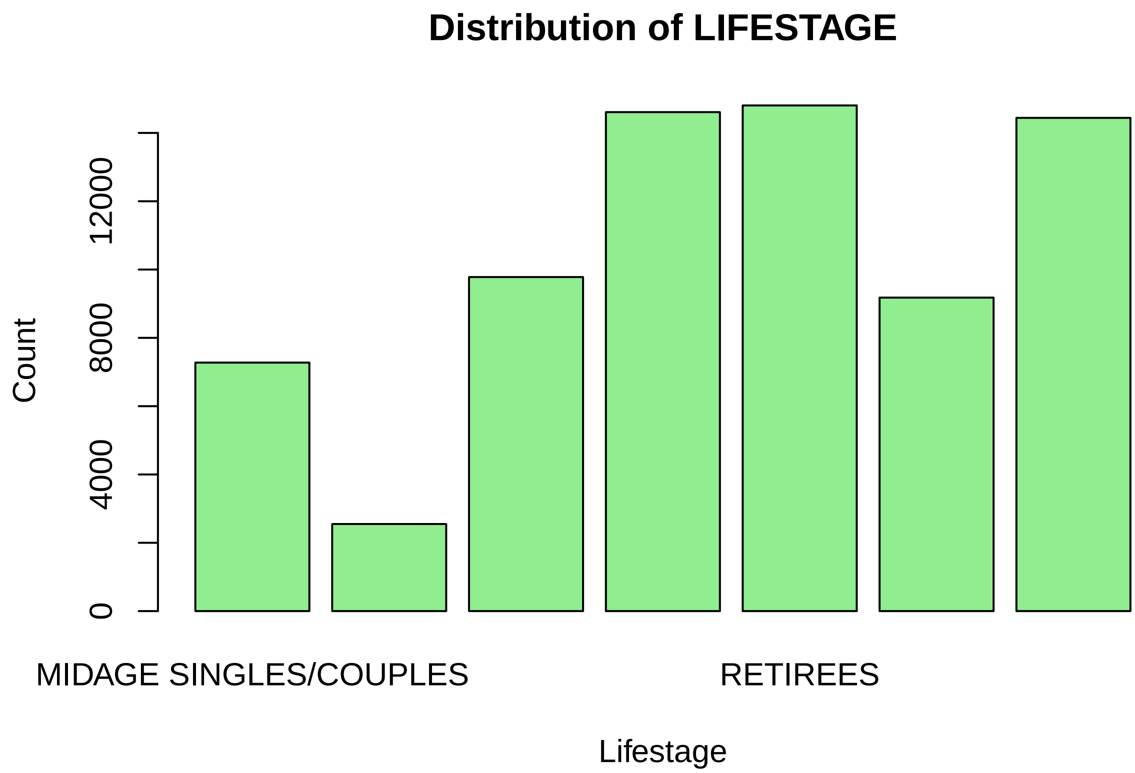
```r
summary(customerData)
```

```
##  LYLTY_CARD_NBR     LIFESTAGE          PREMIUM_CUSTOMER
##  Min.   :   1000   Length:72637       Length:72637
##  1st Qu.:  66202   Class :character   Class :character
##  Median : 134040   Mode  :character   Mode  :character
##  Mean   : 136186
##  3rd Qu.: 203375
##  Max.   :2373711
```

```r
# Example of examining key columns' distributions
# Assuming key columns are CUSTOMER_ID, AGE, and GENDER

# Distribution of LIFESTAGE (assuming it is a categorical variable)
table(customerData$LIFESTAGE)
```

```
##
## MIDAGE SINGLES/COUPLES           NEW FAMILIES         OLDER FAMILIES
##                   7275                   2549                   9780
##  OLDER SINGLES/COUPLES               RETIREES         YOUNG FAMILIES
##                  14609                  14805                   9178
##  YOUNG SINGLES/COUPLES
##                  14441
```

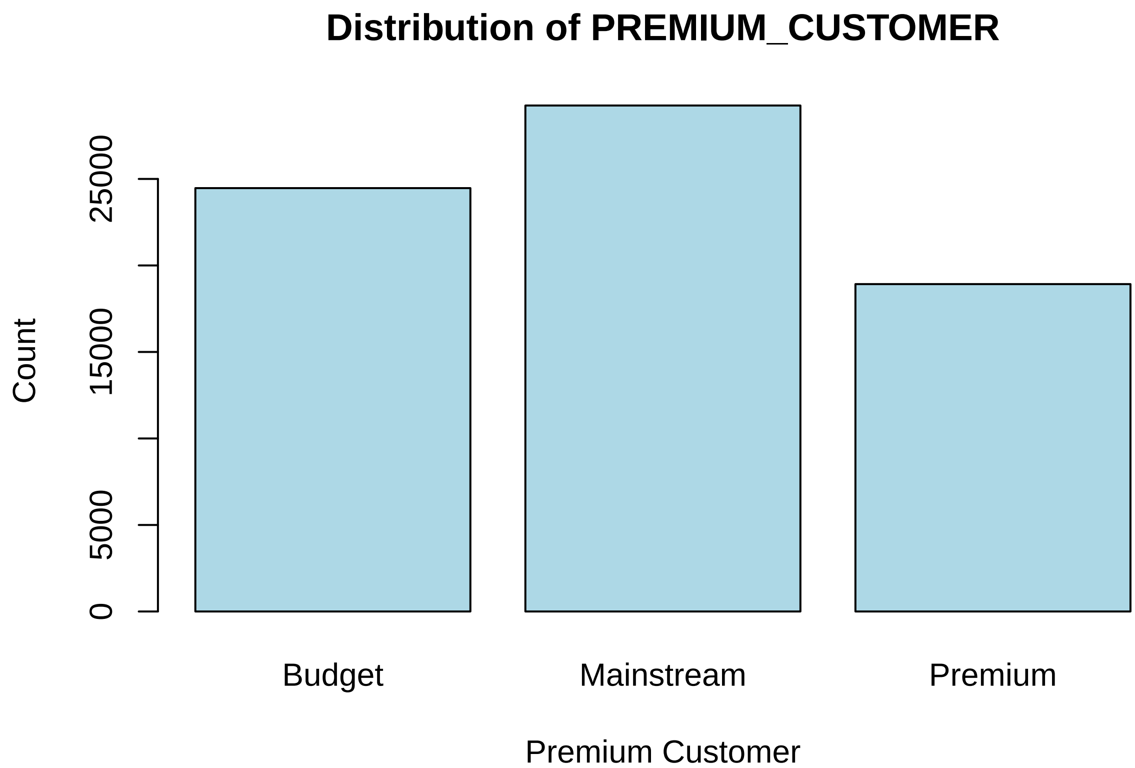
```r
barplot(table(customerData$LIFESTAGE), main = "Distribution of LIFESTAGE", xlab =
↪  "Lifestage", ylab = "Count", col = "lightgreen")
```




```r
# Distribution of PREMIUM_CUSTOMER (assuming it is a categorical variable)
table(customerData$PREMIUM_CUSTOMER)
```

```
##
##     Budget Mainstream    Premium
##      24470      29245      18922
```

```r
barplot(table(customerData$PREMIUM_CUSTOMER), main = "Distribution of PREMIUM_CUSTOMER",
  xlab = "Premium Customer", ylab = "Count", col = "lightblue")
```




```
# Check for missing values
sapply(customerData, function(x) sum(is.na(x)))
```

```
##   LYLTY_CARD_NBR        LIFESTAGE PREMIUM_CUSTOMER
##                0                0                0
```

```
# View a sample of the customerData to manually inspect
head(customerData)
```

```
##    LYLTY_CARD_NBR              LIFESTAGE PREMIUM_CUSTOMER
##             <int>                 <char>           <char>
## 1:           1000  YOUNG SINGLES/COUPLES          Premium
## 2:           1002  YOUNG SINGLES/COUPLES       Mainstream
## 3:           1003         YOUNG FAMILIES           Budget
## 4:           1004  OLDER SINGLES/COUPLES       Mainstream
## 5:           1005 MIDAGE SINGLES/COUPLES       Mainstream
## 6:           1007  YOUNG SINGLES/COUPLES           Budget
```

```
#### Merge transaction data to customer data
data <- merge(transactionData, customerData, all.x = TRUE)
```

As the number of rows in `data` is the same as that of `transactionData`, we can be sure that no duplicates were created. This is because we created `data` by setting `all.x = TRUE` (in other words, a left join) which means take all the rows in `transactionData` and find rows with matching values in shared columns and then joining the details in these rows to the `x` or the first mentioned table. Let's also check if some customers were not matched on by checking for nulls.

```
# Over to you! See if any transactions did not have a matched customer.
summary(data)
```

```
##  LYLTY_CARD_NBR         DATE              STORE_NBR         TXN_ID
##  Min.   :   1000   Min.   :2018-07-01   Min.   :  1.0   Min.   :      1
##  1st Qu.:  70015   1st Qu.:2018-09-30   1st Qu.: 70.0   1st Qu.:  67569
##  Median : 130367   Median :2018-12-30   Median :130.0   Median : 135183
##  Mean   : 135531   Mean   :2018-12-30   Mean   :135.1   Mean   : 135131
##  3rd Qu.: 203084   3rd Qu.:2019-03-31   3rd Qu.:203.0   3rd Qu.: 202654
##  Max.   :2373711   Max.   :2019-06-30   Max.   :272.0   Max.   :2415841
##
##     PROD_NBR       PROD_NAME            PROD_QTY         TOT_SALES
##  Min.   :  1.00   Length:246742      Min.   :  1.000   Min.   :  1.700
##  1st Qu.: 26.00   Class :character   1st Qu.:  2.000   1st Qu.:  5.800
##  Median : 53.00   Mode  :character   Median :  2.000   Median :  7.400
##  Mean   : 56.35                      Mean   :  1.908   Mean   :  7.321
##  3rd Qu.: 87.00                      3rd Qu.:  2.000   3rd Qu.:  8.800
##  Max.   :114.00                      Max.   :200.000   Max.   :650.000
##
##  CLEANED_PROD_NAME    PACK_SIZE        BRAND             LIFESTAGE
##  Length:246742      175    :66390   Length:246742      Length:246742
##  Class :character   150    :40203   Class :character   Class :character
##  Mode  :character   134    :25102   Mode  :character   Mode  :character
##                     110    :22387
##                     170    :19983
##                     165    :15297
##                     (Other):57380
##  PREMIUM_CUSTOMER
##  Length:246742
##  Class :character
##  Mode  :character
##
##
##
##
```

Great, there are no nulls! So all our customers in the transaction data has been accounted for in the customer dataset. Note that if you are continuing with Task 2, you may want to retain this dataset which you can write out as a csv

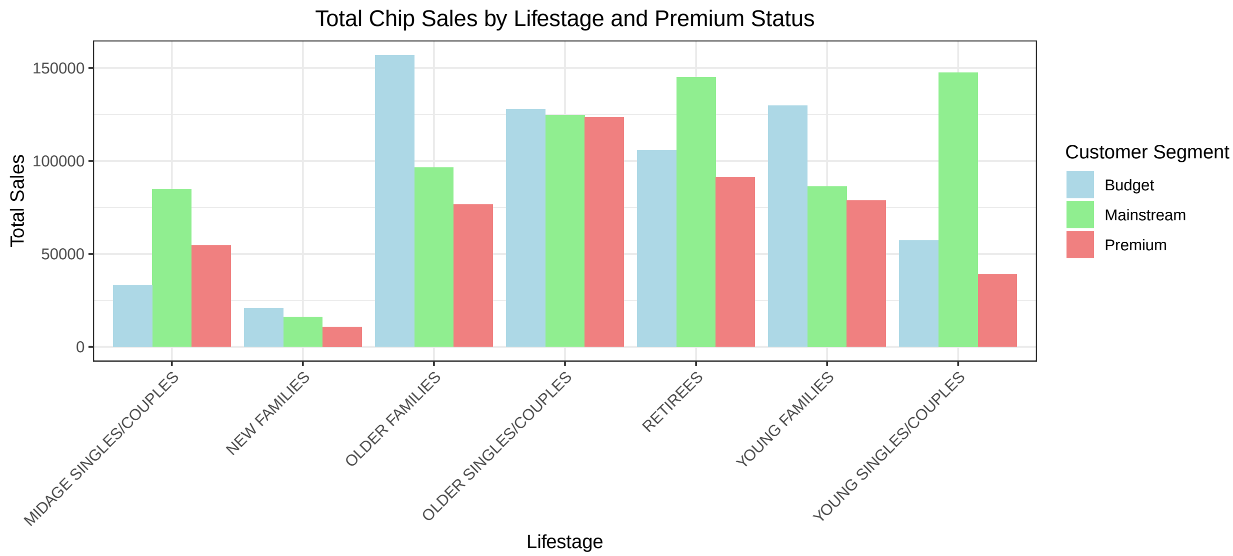
```
# File path
directory <- "/Users/huilinng/Desktop/Forage/Quantum Data Analytics"
file_name <- "QVI_data.csv"
file_path <- paste0(directory, "/", file_name)
fwrite(data, file = file_path)
```

Data exploration is now complete!

Data analysis on customer segments

Now that the data is ready for analysis, we can define some metrics of interest to the client: - Who spends the most on chips (total sales), describing customers by lifestage and how premium their general purchasing behaviour is - How many customers are in each segment - How many chips are bought per customer by segment - What's the average chip price by customer segment We could also ask our data team for more information. Examples are: - The customer's total spend over the period and total spend for each transaction to understand what proportion of their grocery spend is on chips - Proportion of customers in each customer segment overall to compare against the mix of customers who purchase chips Let's start with calculating total sales by LIFESTAGE and PREMIUM_CUSTOMER and plotting the split by these segments to describe which customer segment contribute most to chip sales.

```
#### Total sales by LIFESTAGE and PREMIUM_CUSTOMER
# Over to you! Calculate the summary of sales by those dimensions and create a plot.
# Assuming 'data' is the merged dataset
# Load necessary libraries
library(data.table)
library(ggplot2)

# Convert data to data.table if not already
data <- as.data.table(data)

# 1. Total Sales by LIFESTAGE and PREMIUM_CUSTOMER
sales_by_segment <- data[, .(Total_Sales = sum(TOT_SALES, na.rm = TRUE)), by =
↪  .(LIFESTAGE, PREMIUM_CUSTOMER)]

# Plot total sales by LIFESTAGE and PREMIUM_CUSTOMER
ggplot(sales_by_segment, aes(x = LIFESTAGE, y = Total_Sales, fill = PREMIUM_CUSTOMER)) +
  geom_bar(stat = "identity", position = "dodge") +
  labs(x = "Lifestage", y = "Total Sales", title = "Total Chip Sales by Lifestage and
  ↪  Premium Status") +
  scale_fill_manual(values = c("lightblue", "lightgreen", "lightcoral"),
                    name = "Customer Segment",
                    labels = c("Budget", "Mainstream", "Premium")) +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```




```r
# 2. Number of Customers per Segment
customer_counts <- data[, .(Num_Customers = uniqueN(LYLTY_CARD_NBR)), by = .(LIFESTAGE,
 ↪  PREMIUM_CUSTOMER)]

# Merge with sales data for full segment summary
segment_summary <- merge(sales_by_segment, customer_counts, by = c("LIFESTAGE",
 ↪  "PREMIUM_CUSTOMER"))

# 3. Chips Bought per Customer by Segment
chips_per_customer <- data[, .(Total_Chips_Bought = sum(PROD_QTY, na.rm = TRUE)), by =
 ↪  .(LIFESTAGE, PREMIUM_CUSTOMER)]
chips_per_customer <- merge(chips_per_customer, customer_counts, by = c("LIFESTAGE",
 ↪  "PREMIUM_CUSTOMER"))
chips_per_customer[, Avg_Chips_Per_Customer := Total_Chips_Bought / Num_Customers]

# 4. Average Chip Price by Segment
data[, Avg_Price := TOT_SALES / PROD_QTY]  # Calculate price per unit
avg_price_per_segment <- data[, .(Avg_Chip_Price = mean(Avg_Price, na.rm = TRUE)), by =
 ↪  .(LIFESTAGE, PREMIUM_CUSTOMER)]

# Print the summaries
print(segment_summary)
```

```
## Key: <LIFESTAGE, PREMIUM_CUSTOMER>
##                  LIFESTAGE PREMIUM_CUSTOMER Total_Sales Num_Customers
##                     <char>           <char>       <num>         <int>
##  1: MIDAGE SINGLES/COUPLES           Budget    33345.70          1474
##  2: MIDAGE SINGLES/COUPLES       Mainstream    84734.25          3298
##  3: MIDAGE SINGLES/COUPLES          Premium    54443.85          2369
##  4:           NEW FAMILIES           Budget    20607.45          1087
##  5:           NEW FAMILIES       Mainstream    15979.70           830
##  6:           NEW FAMILIES          Premium    10760.80           575
##  7:         OLDER FAMILIES           Budget   156863.75          4611
##  8:         OLDER FAMILIES       Mainstream    96413.55          2788
##  9:         OLDER FAMILIES          Premium    76542.60          2232
```

```
## 10:  OLDER SINGLES/COUPLES           Budget   127833.60           4849
## 11:  OLDER SINGLES/COUPLES       Mainstream   124648.50           4858
## 12:  OLDER SINGLES/COUPLES          Premium   123537.55           4682
## 13:               RETIREES           Budget   105916.30           4385
## 14:               RETIREES       Mainstream   145168.95           6358
## 15:               RETIREES          Premium    91296.65           3812
## 16:         YOUNG FAMILIES           Budget   129717.95           3953
## 17:         YOUNG FAMILIES       Mainstream    86338.25           2685
## 18:         YOUNG FAMILIES          Premium    78571.70           2398
## 19:  YOUNG SINGLES/COUPLES           Budget    57122.10           3647
## 20:  YOUNG SINGLES/COUPLES       Mainstream   147582.20           7917
## 21:  YOUNG SINGLES/COUPLES          Premium    39052.30           2480
##                  LIFESTAGE PREMIUM_CUSTOMER Total_Sales Num_Customers
```

```r
print(chips_per_customer)
```

```
## Key: <LIFESTAGE, PREMIUM_CUSTOMER>
##                  LIFESTAGE PREMIUM_CUSTOMER Total_Chips_Bought Num_Customers
##                     <char>           <char>              <int>         <int>
##  1: MIDAGE SINGLES/COUPLES           Budget               8883          1474
##  2: MIDAGE SINGLES/COUPLES       Mainstream              21213          3298
##  3: MIDAGE SINGLES/COUPLES          Premium              14400          2369
##  4:           NEW FAMILIES           Budget               5241          1087
##  5:           NEW FAMILIES       Mainstream               4060           830
##  6:           NEW FAMILIES          Premium               2769           575
##  7:         OLDER FAMILIES           Budget              41853          4611
##  8:         OLDER FAMILIES       Mainstream              25804          2788
##  9:         OLDER FAMILIES          Premium              20639          2232
## 10:  OLDER SINGLES/COUPLES           Budget              32883          4849
## 11:  OLDER SINGLES/COUPLES       Mainstream              32607          4858
## 12:  OLDER SINGLES/COUPLES          Premium              31695          4682
## 13:               RETIREES           Budget              26932          4385
## 14:               RETIREES       Mainstream              37677          6358
## 15:               RETIREES          Premium              23266          3812
## 16:         YOUNG FAMILIES           Budget              34482          3953
## 17:         YOUNG FAMILIES       Mainstream              23194          2685
## 18:         YOUNG FAMILIES          Premium              20901          2398
## 19:  YOUNG SINGLES/COUPLES           Budget              15500          3647
## 20:  YOUNG SINGLES/COUPLES       Mainstream              36225          7917
## 21:  YOUNG SINGLES/COUPLES          Premium              10575          2480
##                  LIFESTAGE PREMIUM_CUSTOMER Total_Chips_Bought Num_Customers
##     Avg_Chips_Per_Customer
##                      <num>
##  1:               6.026459
##  2:               6.432080
##  3:               6.078514
##  4:               4.821527
##  5:               4.891566
##  6:               4.815652
##  7:               9.076773
##  8:               9.255380
##  9:               9.246864
## 10:               6.781398
```

```
## 11:                6.712021
## 12:                6.769543
## 13:                6.141847
## 14:                5.925920
## 15:                6.103358
## 16:                8.722995
## 17:                8.638361
## 18:                8.716013
## 19:                4.250069
## 20:                4.575597
## 21:                4.264113
##     Avg_Chips_Per_Customer
```

```
print(avg_price_per_segment)
```

```
##                  LIFESTAGE PREMIUM_CUSTOMER Avg_Chip_Price
##                     <char>           <char>          <num>
##  1:  YOUNG SINGLES/COUPLES          Premium       3.665414
##  2:  YOUNG SINGLES/COUPLES       Mainstream       4.065642
##  3:         YOUNG FAMILIES           Budget       3.760737
##  4:  OLDER SINGLES/COUPLES       Mainstream       3.814665
##  5: MIDAGE SINGLES/COUPLES       Mainstream       3.994241
##  6:  YOUNG SINGLES/COUPLES           Budget       3.657366
##  7:           NEW FAMILIES          Premium       3.872110
##  8:         OLDER FAMILIES       Mainstream       3.737077
##  9:               RETIREES           Budget       3.924404
## 10:  OLDER SINGLES/COUPLES          Premium       3.893182
## 11:         OLDER FAMILIES           Budget       3.745340
## 12: MIDAGE SINGLES/COUPLES          Premium       3.770698
## 13:         OLDER FAMILIES          Premium       3.716910
## 14:               RETIREES       Mainstream       3.844294
## 15:               RETIREES          Premium       3.920942
## 16:         YOUNG FAMILIES       Mainstream       3.724533
## 17: MIDAGE SINGLES/COUPLES           Budget       3.743328
## 18:           NEW FAMILIES       Mainstream       3.916133
## 19:  OLDER SINGLES/COUPLES           Budget       3.882096
## 20:         YOUNG FAMILIES          Premium       3.762150
## 21:           NEW FAMILIES           Budget       3.917688
##                  LIFESTAGE PREMIUM_CUSTOMER Avg_Chip_Price
```

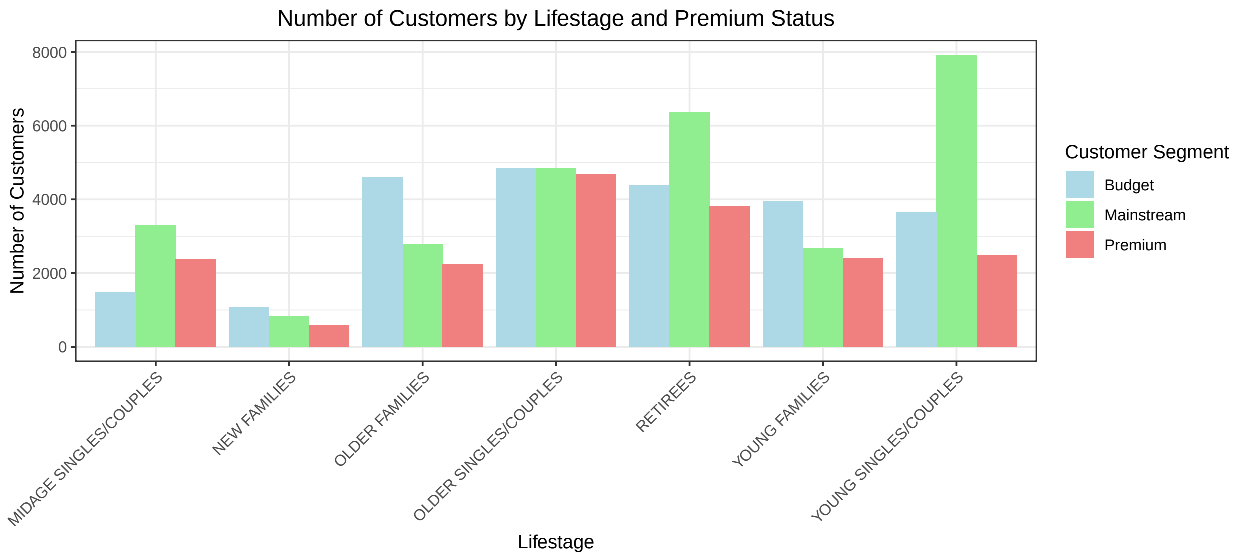
Sales are coming mainly from Budget - older families, Mainstream - young singles/couples, and Mainstream - retirees Let's see if the higher sales are due to there being more customers who buy chips.

```
#### Number of customers by LIFESTAGE and PREMIUM_CUSTOMER
# Over to you! Calculate the summary of number of customers by those dimensions and
↪  create a plot.
# Convert data to data.table if not already
data <- as.data.table(data)

# 1. Count the Number of Customers by LIFESTAGE and PREMIUM_CUSTOMER
customer_counts <- data[, .(Num_Customers = uniqueN(LYLTY_CARD_NBR)), by = .(LIFESTAGE,
↪  PREMIUM_CUSTOMER)]
```

```r
# Print the customer counts
print(customer_counts)
```

```
##                  LIFESTAGE PREMIUM_CUSTOMER Num_Customers
##                     <char>           <char>         <int>
##  1:  YOUNG SINGLES/COUPLES          Premium          2480
##  2:  YOUNG SINGLES/COUPLES       Mainstream          7917
##  3:         YOUNG FAMILIES           Budget          3953
##  4:  OLDER SINGLES/COUPLES       Mainstream          4858
##  5: MIDAGE SINGLES/COUPLES       Mainstream          3298
##  6:  YOUNG SINGLES/COUPLES           Budget          3647
##  7:           NEW FAMILIES          Premium           575
##  8:         OLDER FAMILIES       Mainstream          2788
##  9:               RETIREES           Budget          4385
## 10:  OLDER SINGLES/COUPLES          Premium          4682
## 11:         OLDER FAMILIES           Budget          4611
## 12: MIDAGE SINGLES/COUPLES          Premium          2369
## 13:         OLDER FAMILIES          Premium          2232
## 14:               RETIREES       Mainstream          6358
## 15:               RETIREES          Premium          3812
## 16:         YOUNG FAMILIES       Mainstream          2685
## 17: MIDAGE SINGLES/COUPLES           Budget          1474
## 18:           NEW FAMILIES       Mainstream           830
## 19:  OLDER SINGLES/COUPLES           Budget          4849
## 20:         YOUNG FAMILIES          Premium          2398
## 21:           NEW FAMILIES           Budget          1087
##                  LIFESTAGE PREMIUM_CUSTOMER Num_Customers
```

```r
# 2. Visualize the Number of Customers by LIFESTAGE and PREMIUM_CUSTOMER
ggplot(customer_counts, aes(x = LIFESTAGE, y = Num_Customers, fill = PREMIUM_CUSTOMER)) +
  geom_bar(stat = "identity", position = "dodge") +
  labs(x = "Lifestage", y = "Number of Customers", title = "Number of Customers by
   ↪ Lifestage and Premium Status") +
  scale_fill_manual(values = c("lightblue", "lightgreen", "lightcoral"),
                    name = "Customer Segment",
                    labels = c("Budget", "Mainstream", "Premium")) +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```



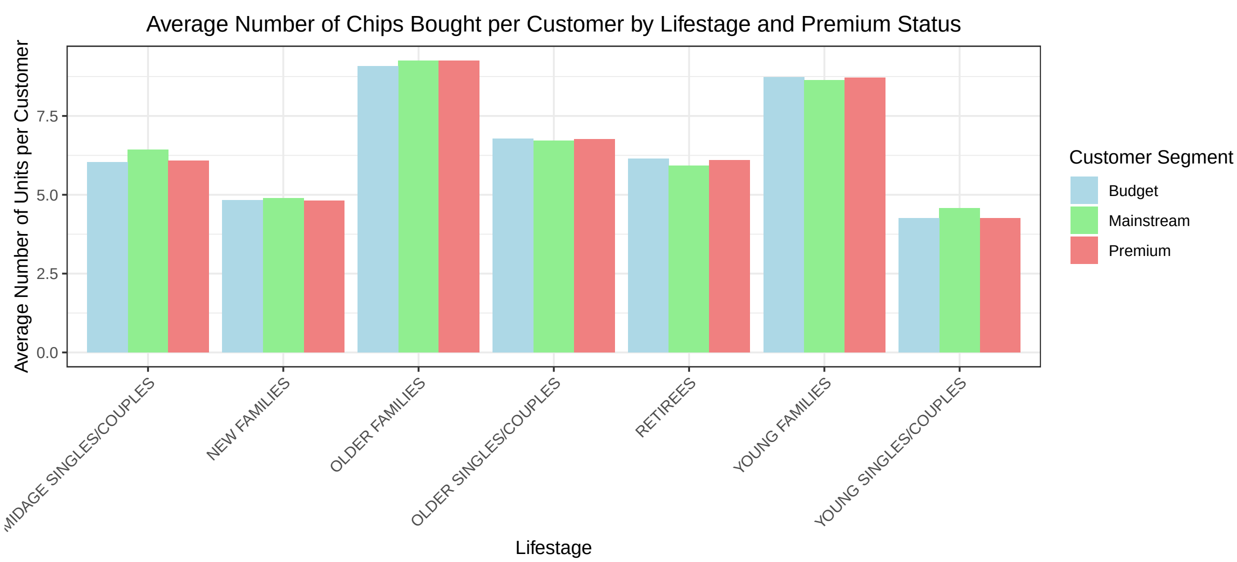
There are more Mainstream - young singles/couples and Mainstream - retirees who buy chips. This contributes to there being more sales to these customer segments but this is not a major driver for the Budget - Older families segment. Higher sales may also be driven by more units of chips being bought per customer. Let's have a look at this next.

```r
#### Average number of units per customer by LIFESTAGE and PREMIUM_CUSTOMER
# Over to you! Calculate and plot the average number of units per customer by those two
↪ dimensions.
# 1. Calculate the Total Number of Units Bought by Each Customer
units_per_customer <- data[, .(Total_Chips_Bought = sum(PROD_QTY, na.rm = TRUE)), by =
↪ LYLTY_CARD_NBR]

# 2. Calculate Average Number of Units per Customer by LIFESTAGE and PREMIUM_CUSTOMER
average_units_per_customer <- data[, .(Total_Chips_Bought = sum(PROD_QTY, na.rm = TRUE)),
↪ by = .(LYLTY_CARD_NBR, LIFESTAGE, PREMIUM_CUSTOMER)]
average_units_per_customer <- average_units_per_customer[, .(Avg_Chips_Per_Customer =
↪ mean(Total_Chips_Bought, na.rm = TRUE)), by = .(LIFESTAGE, PREMIUM_CUSTOMER)]

# Print the average units per customer
print(average_units_per_customer)
```

```
##                  LIFESTAGE PREMIUM_CUSTOMER Avg_Chips_Per_Customer
##                     <char>           <char>                  <num>
##  1:  YOUNG SINGLES/COUPLES          Premium               4.264113
##  2:  YOUNG SINGLES/COUPLES       Mainstream               4.575597
##  3:         YOUNG FAMILIES           Budget               8.722995
##  4:  OLDER SINGLES/COUPLES       Mainstream               6.712021
##  5: MIDAGE SINGLES/COUPLES       Mainstream               6.432080
##  6:  YOUNG SINGLES/COUPLES           Budget               4.250069
##  7:           NEW FAMILIES          Premium               4.815652
##  8:         OLDER FAMILIES       Mainstream               9.255380
##  9:               RETIREES           Budget               6.141847
## 10:  OLDER SINGLES/COUPLES          Premium               6.769543
## 11:         OLDER FAMILIES           Budget               9.076773
```

```
## 12: MIDAGE SINGLES/COUPLES          Premium               6.078514
## 13:         OLDER FAMILIES          Premium               9.246864
## 14:               RETIREES       Mainstream               5.925920
## 15:               RETIREES          Premium               6.103358
## 16:         YOUNG FAMILIES       Mainstream               8.638361
## 17: MIDAGE SINGLES/COUPLES           Budget               6.026459
## 18:           NEW FAMILIES       Mainstream               4.891566
## 19:  OLDER SINGLES/COUPLES           Budget               6.781398
## 20:         YOUNG FAMILIES          Premium               8.716013
## 21:           NEW FAMILIES           Budget               4.821527
##                  LIFESTAGE PREMIUM_CUSTOMER Avg_Chips_Per_Customer
```

```r
# 3. Visualize the Average Number of Units per Customer
ggplot(average_units_per_customer, aes(x = LIFESTAGE, y = Avg_Chips_Per_Customer, fill =
↪ PREMIUM_CUSTOMER)) +
  geom_bar(stat = "identity", position = "dodge") +
  labs(x = "Lifestage", y = "Average Number of Units per Customer", title = "Average
  ↪ Number of Chips Bought per Customer by Lifestage and Premium Status") +
  scale_fill_manual(values = c("lightblue", "lightgreen", "lightcoral"),
                    name = "Customer Segment",
                    labels = c("Budget", "Mainstream", "Premium")) +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```



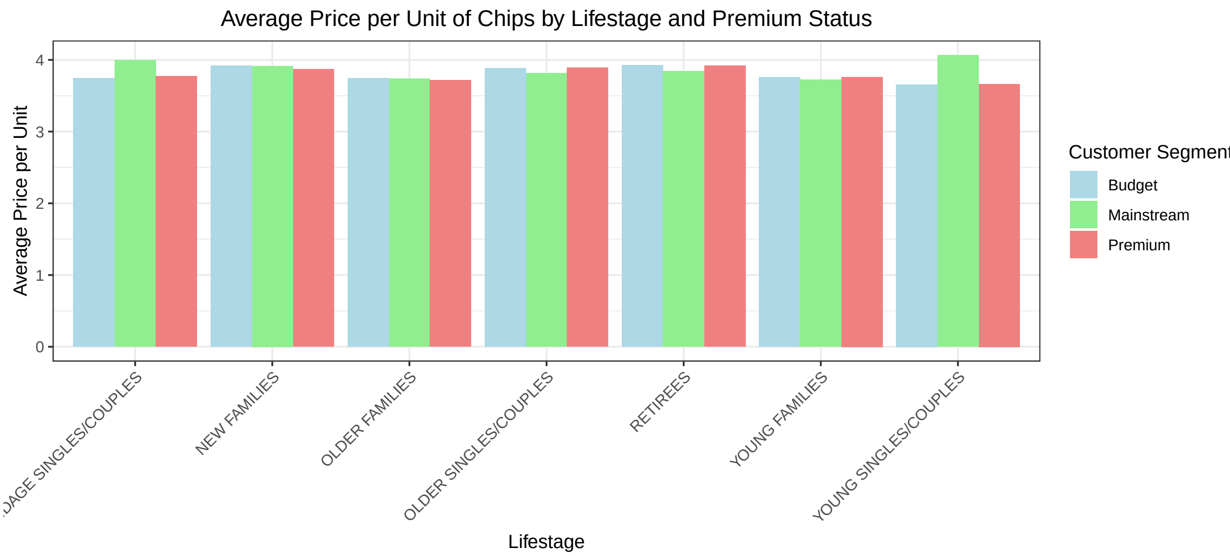
Older families and young families in general buy more chips per customer Let's also investigate the average price per unit chips bought for each customer segment as this is also a driver of total sales.

```r
#### Average price per unit by LIFESTAGE and PREMIUM_CUSTOMER
# Over to you! Calculate and plot the average price per unit sold (average sale price) by
↪ those two customer dimensions.
# 1. Calculate the Average Price per Unit by LIFESTAGE and PREMIUM_CUSTOMER
# Add a new column for price per unit
data[, Avg_Price_Per_Unit := TOT_SALES / PROD_QTY]
```

```r
# Calculate the average price per unit for each segment
avg_price_per_unit <- data[, .(Avg_Price_Per_Unit = mean(Avg_Price_Per_Unit, na.rm =
  TRUE)), by = .(LIFESTAGE, PREMIUM_CUSTOMER)]

# Print the average price per unit
print(avg_price_per_unit)
```

```
##                  LIFESTAGE PREMIUM_CUSTOMER Avg_Price_Per_Unit
##                     <char>           <char>              <num>
##  1:  YOUNG SINGLES/COUPLES          Premium           3.665414
##  2:  YOUNG SINGLES/COUPLES       Mainstream           4.065642
##  3:         YOUNG FAMILIES           Budget           3.760737
##  4:  OLDER SINGLES/COUPLES       Mainstream           3.814665
##  5: MIDAGE SINGLES/COUPLES       Mainstream           3.994241
##  6:  YOUNG SINGLES/COUPLES           Budget           3.657366
##  7:           NEW FAMILIES          Premium           3.872110
##  8:         OLDER FAMILIES       Mainstream           3.737077
##  9:               RETIREES           Budget           3.924404
## 10:  OLDER SINGLES/COUPLES          Premium           3.893182
## 11:         OLDER FAMILIES           Budget           3.745340
## 12: MIDAGE SINGLES/COUPLES          Premium           3.770698
## 13:         OLDER FAMILIES          Premium           3.716910
## 14:               RETIREES       Mainstream           3.844294
## 15:               RETIREES          Premium           3.920942
## 16:         YOUNG FAMILIES       Mainstream           3.724533
## 17: MIDAGE SINGLES/COUPLES           Budget           3.743328
## 18:           NEW FAMILIES       Mainstream           3.916133
## 19:  OLDER SINGLES/COUPLES           Budget           3.882096
## 20:         YOUNG FAMILIES          Premium           3.762150
## 21:           NEW FAMILIES           Budget           3.917688
##                  LIFESTAGE PREMIUM_CUSTOMER Avg_Price_Per_Unit
```

```r
# 2. Visualize the Average Price per Unit
ggplot(avg_price_per_unit, aes(x = LIFESTAGE, y = Avg_Price_Per_Unit, fill =
  PREMIUM_CUSTOMER)) +
  geom_bar(stat = "identity", position = "dodge") +
  labs(x = "Lifestage", y = "Average Price per Unit", title = "Average Price per Unit of
    Chips by Lifestage and Premium Status") +
  scale_fill_manual(values = c("lightblue", "lightgreen", "lightcoral"),
                    name = "Customer Segment",
                    labels = c("Budget", "Mainstream", "Premium")) +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```



Mainstream midage and young singles and couples are more willing to pay more per packet of chips compared to their budget and premium counterparts. This may be due to premium shoppers being more likely to buy healthy snacks and when they buy chips, this is mainly for entertainment purposes rather than their own consumption. This is also supported by there being fewer premium midage and young singles and couples buying chips compared to their mainstream counterparts. As the difference in average price per unit isn't large, we can check if this difference is statistically different.

```r
# Check the first few rows of the dataset to confirm values
head(data)
```

```
## Key: <LYLTY_CARD_NBR>
##    LYLTY_CARD_NBR       DATE STORE_NBR TXN_ID PROD_NBR
##             <int>     <Date>     <int>  <int>    <int>
## 1:           1000 2018-10-17         1      1        5
## 2:           1002 2018-09-16         1      2       58
## 3:           1003 2019-03-07         1      3       52
## 4:           1003 2019-03-08         1      4      106
## 5:           1004 2018-11-02         1      5       96
## 6:           1005 2018-12-28         1      6       86
##                                 PROD_NAME PROD_QTY TOT_SALES
##                                    <char>    <int>     <num>
## 1: Natural Chip        Compny SeaSalt175g        2       6.0
## 2:  Red Rock Deli Chikn&Garlic Aioli 150g        1       2.7
## 3:  Grain Waves Sour    Cream&Chives 210G        1       3.6
## 4: Natural ChipCo      Hony Soy Chckn175g        1       3.0
## 5:         WW Original Stacked Chips 160g        1       1.9
## 6:                    Cheetos Puffs 165g        1       2.8
##              CLEANED_PROD_NAME PACK_SIZE   BRAND              LIFESTAGE
##                         <char>    <fctr>  <char>                 <char>
## 1:    NaturalChipCompnySeaSaltg       175 NATURAL  YOUNG SINGLES/COUPLES
## 2: RedRockDeliChiknGarlicAiolig       150     Red  YOUNG SINGLES/COUPLES
## 3:   GrainWavesSourCreamChivesG       210   Grain         YOUNG FAMILIES
## 4:   NaturalChipCoHonySoyChckng       175 NATURAL         YOUNG FAMILIES
## 5:      WWOriginalStackedChipsg       160      WW  OLDER SINGLES/COUPLES
```

```
## 6:                CheetosPuffsg       165 Cheetos MIDAGE SINGLES/COUPLES
##    PREMIUM_CUSTOMER Avg_Price Avg_Price_Per_Unit
##              <char>     <num>              <num>
## 1:          Premium       3.0                3.0
## 2:       Mainstream       2.7                2.7
## 3:           Budget       3.6                3.6
## 4:           Budget       3.0                3.0
## 5:       Mainstream       1.9                1.9
## 6:       Mainstream       2.8                2.8
```

```r
# Check the number of rows before filtering
print(nrow(data))
```

```
## [1] 246742
```

```r
# Filter for midage and young singles and couples
subset_data <- data[LIFESTAGE %in% c("Midage", "Young Singles/Couples")]
print(nrow(subset_data))  # Check the number of rows after filtering
```

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

```r
# Check the first few rows of the subset
head(subset_data)
```

```
## Key: <LYLTY_CARD_NBR>
## Empty data.table (0 rows and 15 cols):
LYLTY_CARD_NBR,DATE,STORE_NBR,TXN_ID,PROD_NBR,PROD_NAME...
```

```r
# Verify unique values in PREMIUM_CUSTOMER
unique_premium_customer <- unique(data$PREMIUM_CUSTOMER)
print(unique_premium_customer)
```

```
## [1] "Premium"    "Mainstream" "Budget"
```

```r
# Check unique values in LIFESTAGE
unique_lifestage <- unique(data$LIFESTAGE)
print(unique_lifestage)
```

```
## [1] "YOUNG SINGLES/COUPLES" "YOUNG FAMILIES" "OLDER SINGLES/COUPLES"
## [4] "MIDAGE SINGLES/COUPLES" "NEW FAMILIES" "OLDER FAMILIES"
## [7] "RETIREES"
```

```r
# Filter by LIFESTAGE alone
subset_lifestage <- data[LIFESTAGE %in% c("MIDAGE SINGLES/COUPLES", "YOUNG
↪  SINGLES/COUPLES")]
print(nrow(subset_lifestage))
```

```
## [1] 57367
```

```r
head(subset_lifestage)
```

```
## Key: <LYLTY_CARD_NBR>
## LYLTY_CARD_NBR DATE STORE_NBR TXN_ID PROD_NBR
## <int> <Date> <int> <int> <int>
## 1: 1000 2018-10-17 1 1 5
## 2: 1002 2018-09-16 1 2 58
## 3: 1005 2018-12-28 1 6 86
## 4: 1007 2018-12-04 1 7 49
## 5: 1007 2018-12-05 1 8 10
## 6: 1010 2018-09-09 1 10 51
## PROD_NAME PROD_QTY TOT_SALES
## <char> <int> <num>
## 1: Natural Chip Compny SeaSalt175g 2 6.0
## 2: Red Rock Deli Chikn&Garlic Aioli 150g 1 2.7
## 3: Cheetos Puffs 165g 1 2.8
## 4: Infuzions SourCream&Herbs Veg Strws 110g 1 3.8
## 5: RRD SR Slow Rst Pork Belly 150g 1 2.7
## 6: Doritos Mexicana 170g 2 8.8
## CLEANED_PROD_NAME PACK_SIZE BRAND LIFESTAGE
## <char> <fctr> <char> <char>
## 1: NaturalChipCompnySeaSaltg 175 NATURAL YOUNG SINGLES/COUPLES
## 2: RedRockDeliChiknGarlicAioligg 150 Red YOUNG SINGLES/COUPLES
## 3: CheetosPuffsg 165 Cheetos MIDAGE SINGLES/COUPLES
## 4: InfuzionsSourCreamHerbsVegStrwsg 110 Infuzions YOUNG SINGLES/COUPLES
## 5: RRDSRSlowRstPorkBellyg 150 RRD YOUNG SINGLES/COUPLES
## 6: DoritosMexicanag 170 Doritos YOUNG SINGLES/COUPLES
## PREMIUM_CUSTOMER Avg_Price Avg_Price_Per_Unit
## <char> <num> <num>
## 1: Premium 3.0 3.0
## 2: Mainstream 2.7 2.7
## 3: Mainstream 2.8 2.8
## 4: Budget 3.8 3.8
## 5: Budget 2.7 2.7
## 6: Mainstream 4.4 4.4
```

```r
# Apply filters with corrected values
subset_data <- data[LIFESTAGE %in% c("MIDAGE SINGLES/COUPLES", "YOUNG SINGLES/COUPLES") &
                      PREMIUM_CUSTOMER %in% c("Mainstream", "Budget", "Premium")]

print(nrow(subset_data))
```

```
## [1] 57367
```

```r
head(subset_data)
```

```
## Key: <LYLTY_CARD_NBR>
## LYLTY_CARD_NBR DATE STORE_NBR TXN_ID PROD_NBR
## <int> <Date> <int> <int> <int>
## 1: 1000 2018-10-17 1 1 5
## 2: 1002 2018-09-16 1 2 58
```

```
## 3: 1005 2018-12-28 1 6 86
## 4: 1007 2018-12-04 1 7 49
## 5: 1007 2018-12-05 1 8 10
## 6: 1010 2018-09-09 1 10 51
## PROD_NAME PROD_QTY TOT_SALES
## <char> <int> <num>
## 1: Natural Chip Compny SeaSalt175g 2 6.0
## 2: Red Rock Deli Chikn&Garlic Aioli 150g 1 2.7
## 3: Cheetos Puffs 165g 1 2.8
## 4: Infuzions SourCream&Herbs Veg Strws 110g 1 3.8
## 5: RRD SR Slow Rst Pork Belly 150g 1 2.7
## 6: Doritos Mexicana 170g 2 8.8
## CLEANED_PROD_NAME PACK_SIZE BRAND LIFESTAGE
## <char> <fctr> <char> <char>
## 1: NaturalChipCompnySeaSaltg 175 NATURAL YOUNG SINGLES/COUPLES
## 2: RedRockDeliChiknGarlicAiolig 150 Red YOUNG SINGLES/COUPLES
## 3: CheetosPuffsg 165 Cheetos MIDAGE SINGLES/COUPLES
## 4: InfuzionsSourCreamHerbsVegStrwsg 110 Infuzions YOUNG SINGLES/COUPLES
## 5: RRDSRSlowRstPorkBellyg 150 RRD YOUNG SINGLES/COUPLES
## 6: DoritosMexicanag 170 Doritos YOUNG SINGLES/COUPLES
## PREMIUM_CUSTOMER Avg_Price Avg_Price_Per_Unit
## <char> <num> <num>
## 1: Premium 3.0 3.0
## 2: Mainstream 2.7 2.7
## 3: Mainstream 2.8 2.8
## 4: Budget 3.8 3.8
## 5: Budget 2.7 2.7
## 6: Mainstream 4.4 4.4
```

```r
# Create a new column to classify segments for t-test
subset_data[, Segment_Group := ifelse(PREMIUM_CUSTOMER %in% c("Mainstream"),
↪  "Mainstream", "Other")]

# 2. Perform an Independent t-Test
# Perform t-test comparing Mainstream vs. Other (Budget and Premium)
t_test_result <- t.test(Avg_Price_Per_Unit ~ Segment_Group, data = subset_data)

# Print the t-test result
print(t_test_result)
```

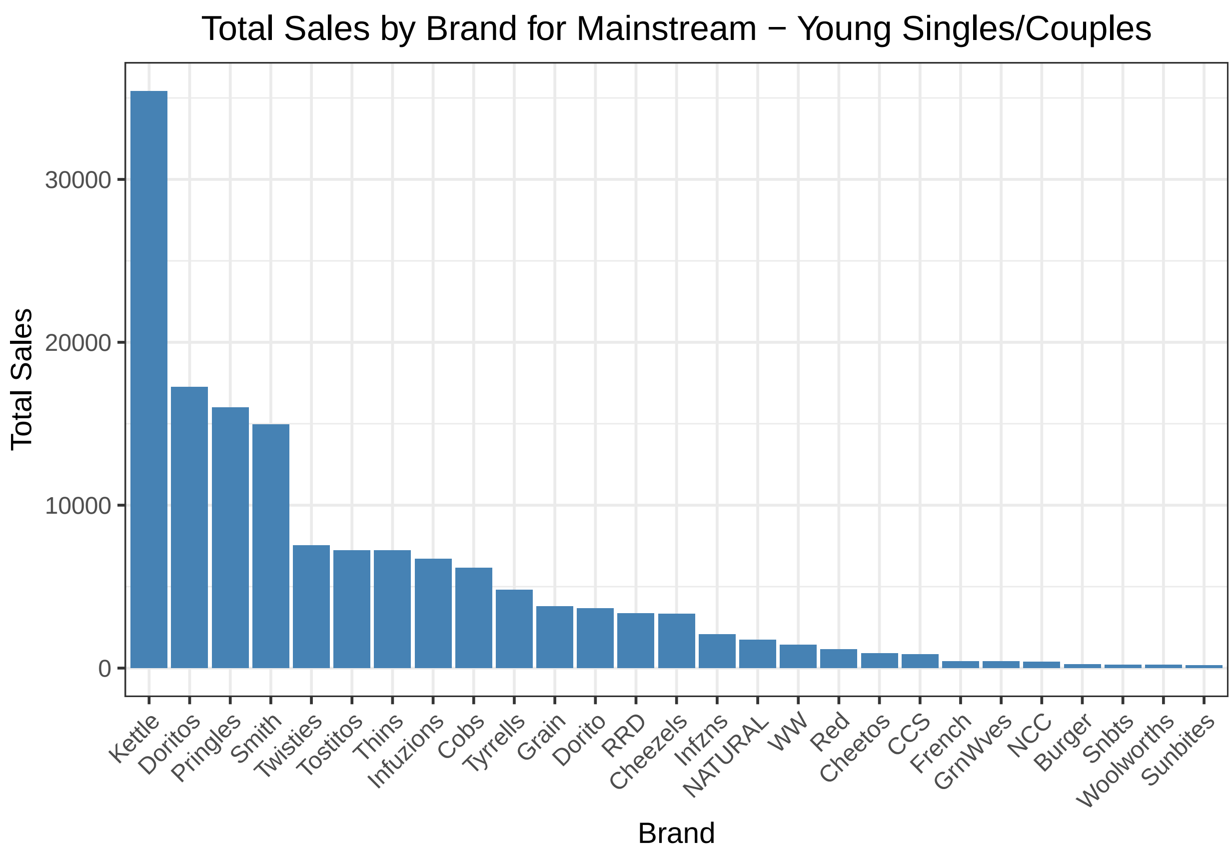
```
##
## Welch Two Sample t-test
##
## data: Avg_Price_Per_Unit by Segment_Group
## t = 37.624, df = 54791, p-value < 2.2e-16
## alternative hypothesis: true difference in means between group Mainstream
and group Other is not equal to 0
## 95 percent confidence interval:
## 0.3159319 0.3506572
## sample estimates:
## mean in group Mainstream mean in group Other
## 4.039786 3.706491
```

The t-test results in a p-value of 37.624, i.e. the unit price for mainstream, young and mid-age singles and couples [ARE] significantly higher than that of budget or premium, young and midage singles and couples.

Deep dive into specific customer segments for insights

We have found quite a few interesting insights that we can dive deeper into. We might want to target customer segments that contribute the most to sales to retain them or further increase sales. Let's look at Mainstream - young singles/couples. For instance, let's find out if they tend to buy a particular brand of chips.

```r
#### Deep dive into Mainstream, young singles/couples
# Over to you! Work out of there are brands that these two customer segments prefer more
↪ than others. You could use a technique called affinity analysis or a-priori analysis
↪ (or any other method if you prefer)

# Filter data for Mainstream - young singles/couples
mainstream_young_singles_couples <- data[LIFESTAGE == "YOUNG SINGLES/COUPLES" &
↪ PREMIUM_CUSTOMER == "Mainstream"]

# Aggregate data to find total sales by brand
brand_sales <- mainstream_young_singles_couples[, .(Total_Sales = sum(TOT_SALES)), by =
↪ BRAND]

library(ggplot2)

# Plot the total sales by brand for the segment
ggplot(brand_sales, aes(x = reorder(BRAND, -Total_Sales), y = Total_Sales)) +
  geom_bar(stat = "identity", fill = "steelblue") +
  labs(title = "Total Sales by Brand for Mainstream - Young Singles/Couples",
       x = "Brand", y = "Total Sales") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```




```
install.packages("arules", repos = "https://cloud.r-project.org")
```

```
##
## The downloaded binary packages are in
##
/var/folders/9_/jxz9vp3x1ds19kl7dz74z2pw0000gn/T//Rtmpnjs5L1/downloaded_packages
```

```
library(arules)
```

```
## Warning: package 'arules' was built under R version 4.3.3

## Loading required package: Matrix

##
## Attaching package: 'arules'

## The following objects are masked from 'package:base':
##
##     abbreviate, write
```

```
# Prepare transaction data for the analysis
transaction_data <- as(split(mainstream_young_singles_couples$BRAND,
↪  mainstream_young_singles_couples$TXN_ID), "transactions")
```

```
## Warning in asMethod(object): removing duplicated items in transactions
```

```r
# Perform association rule mining
rules <- apriori(transaction_data, parameter = list(supp = 0.01, conf = 0.5, target =
  "rules"))
```

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.5    0.1    1 none FALSE            TRUE       5    0.01      1
##  maxlen target  ext
##      10  rules TRUE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 194
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[27 item(s), 19482 transaction(s)] done [0.00s].
## sorting and recoding items ... [19 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 done [0.00s].
## writing ... [0 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

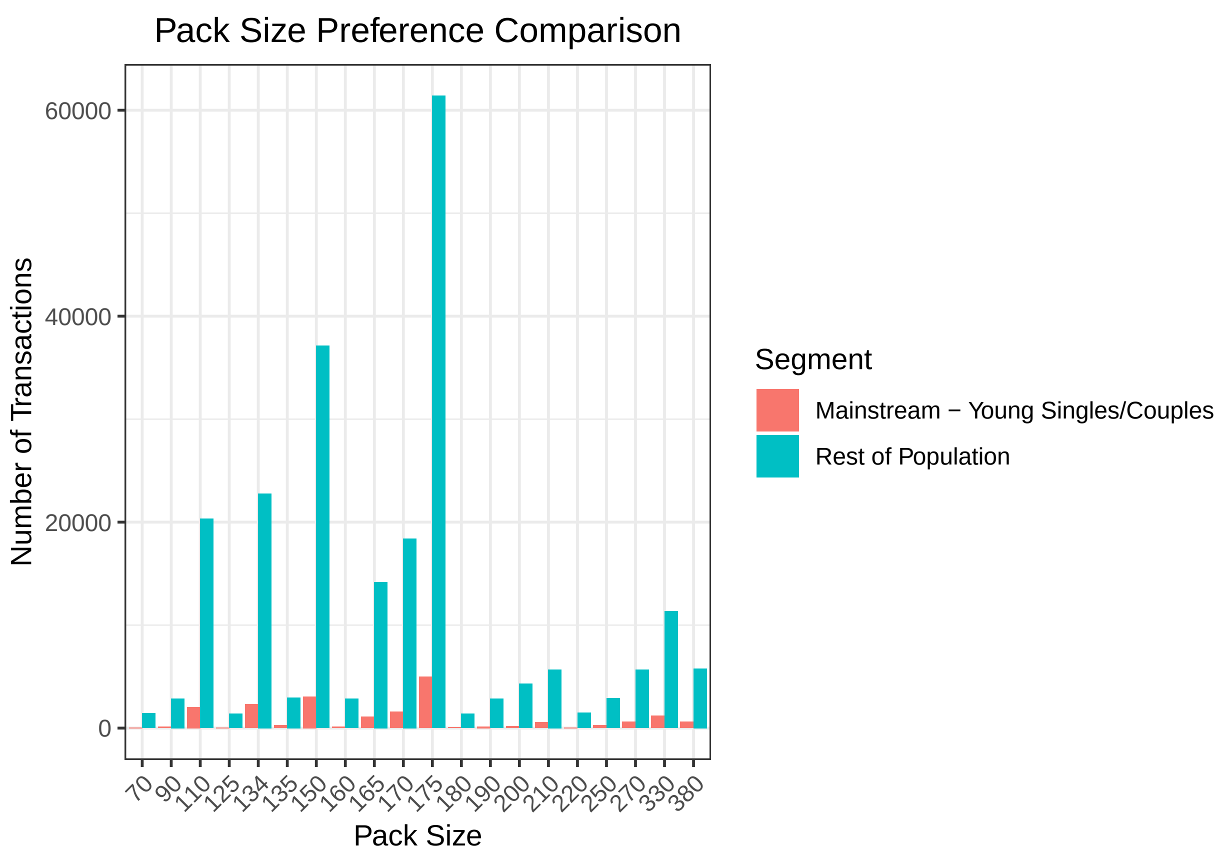
```r
# Inspect rules
inspect(rules)
```

We can see that Kettle is the most popular brand for mainstream young singles/couples followed by Doritos and Pringles with total sales of Kettle at 35000 which is around double of Doritos and Pringles.

Let's also find out if our target segment tends to buy larger packs of chips.

```r
#### Preferred pack size compared to the rest of the population
# Over to you! Do the same for pack size.
# Filter data for Mainstream - young singles/couples
mainstream_young_singles_couples <- data[LIFESTAGE == "YOUNG SINGLES/COUPLES" &
  PREMIUM_CUSTOMER == "Mainstream"]

# Filter data for the rest of the population
rest_of_population <- data[!(LIFESTAGE == "YOUNG SINGLES/COUPLES" & PREMIUM_CUSTOMER ==
  "Mainstream")]

# Aggregate pack sizes by count for both segments
pack_size_comparison <- rbindlist(list(
  mainstream_young_singles_couples[, .(Count = .N), by = PACK_SIZE][, Segment :=
  "Mainstream - Young Singles/Couples"],
  rest_of_population[, .(Count = .N), by = PACK_SIZE][, Segment := "Rest of Population"]
))
```

```r
library(ggplot2)

# Plot the comparison of pack sizes between the segments
ggplot(pack_size_comparison, aes(x = PACK_SIZE, y = Count, fill = Segment)) +
  geom_bar(stat = "identity", position = "dodge") +
  labs(title = "Pack Size Preference Comparison",
       x = "Pack Size", y = "Number of Transactions",
       fill = "Segment") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```



Although mainstream - young singles/couples do buy more larger pack size chips compared to the rest of the population, they generally buy more of chips from 100g to 200g.