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("data.table")
#### Load required libraries
library("data.table")
```

```
## Warning: package 'data.table' was built under R version 4.3.1
```

```
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 4.3.1
```

```
library(ggmosaic)
```

```
## Warning: package 'ggmosaic' was built under R version 4.3.1
```

```
library(readr)
```

```
## Warning: package 'readr' was built under R version 4.3.1
```

```
library(plyr)
```

```
## Warning: package 'plyr' was built under R version 4.3.1
```

```r
library(dplyr)
```

```
## Warning: package 'dplyr' was built under R version 4.3.1
```

```
##
## Attaching package: 'dplyr'
```

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

```
## The following objects are masked from 'package:data.table':
##
##     between, first, last
```

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

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

```r
library(lubridate)
```

```
## Warning: package 'lubridate' was built under R version 4.2.3
```

```
##
## Attaching package: 'lubridate'
```

```
## The following objects are masked from 'package:data.table':
##
##     hour, isoweek, mday, minute, month, quarter, second, wday, week,
##     yday, year
```

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

```r
library(stringr)
```

```
## Warning: package 'stringr' was built under R version 4.3.1
```

```r
library(arules)
```

```
## Warning: package 'arules' was built under R version 4.2.3
```

```
## Loading required package: Matrix

##
## Attaching package: 'arules'

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

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

```
library(arulesViz)
```

```
## Warning: package 'arulesViz' was built under R version 4.2.3
```

```
#### 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 (\)
filePath <- paste0(getwd(),"/")
transactionData <- fread(file = paste0(filePath,"QVI_transaction_data.csv"), header =
↪  TRUE)
customerData <- fread(file = paste0(filePath,"QVI_purchase_behaviour.csv"), header =
↪  TRUE)
```

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.

```
#### Examine transaction data
# Over to you! Examine the data using one or more of the methods described above.

str(transactionData)
```

```
## Classes 'data.table' and 'data.frame': 264836 obs. of 8 variables:
## $ DATE : int 43390 43599 43605 43329 43330 43604 43601 43601 43332 43330 ...
## $ STORE_NBR : int 1 1 1 2 2 4 4 4 5 7 ...
## $ LYLTY_CARD_NBR: int 1000 1307 1343 2373 2426 4074 4149 4196 5026 7150 ...
## $ TXN_ID : int 1 348 383 974 1038 2982 3333 3539 4525 6900 ...
## $ PROD_NBR : int 5 66 61 69 108 57 16 24 42 52 ...
## $ PROD_NAME : chr "Natural Chip Compny SeaSalt175g" "CCs Nacho Cheese 175g"
"Smiths Crinkle Cut Chips Chicken 170g" "Smiths Chip Thinly S/Cream&Onion 175g"
...
## $ PROD_QTY : int 2 3 2 5 3 1 1 1 1 2 ...
## $ TOT_SALES : num 6 6.3 2.9 15 13.8 5.1 5.7 3.6 3.9 7.2 ...
## - attr(*, ".internal.selfref")=<externalptr>
```

```r
transactionData
```

```
##          DATE STORE_NBR LYLTY_CARD_NBR TXN_ID PROD_NBR
##      1: 43390         1           1000      1        5
##      2: 43599         1           1307    348       66
##      3: 43605         1           1343    383       61
##      4: 43329         2           2373    974       69
##      5: 43330         2           2426   1038      108
##     ---
## 264832: 43533       272         272319 270088       89
## 264833: 43325       272         272358 270154       74
## 264834: 43410       272         272379 270187       51
## 264835: 43461       272         272379 270188       42
## 264836: 43365       272         272380 270189       74
##                                         PROD_NAME PROD_QTY TOT_SALES
##      1:   Natural Chip        Compny SeaSalt175g        2       6.0
##      2:                 CCs Nacho Cheese    175g        3       6.3
##      3:   Smiths Crinkle Cut  Chips Chicken 170g        2       2.9
##      4:   Smiths Chip Thinly  S/Cream&Onion 175g        5      15.0
##      5: Kettle Tortilla ChpsHny&Jlpno Chili 150g        3      13.8
##     ---
## 264832:  Kettle Sweet Chilli And Sour Cream 175g        2      10.8
## 264833:            Tostitos Splash Of  Lime 175g        1       4.4
## 264834:                 Doritos Mexicana    170g        2       8.8
## 264835:  Doritos Corn Chip Mexican Jalapeno 150g        2       7.8
## 264836:            Tostitos Splash Of  Lime 175g        2       8.8
```

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.

transactionData$PROD_NAME[1:10]
```

```
##  [1] "Natural Chip        Compny SeaSalt175g"
##  [2] "CCs Nacho Cheese    175g"
##  [3] "Smiths Crinkle Cut  Chips Chicken 170g"
##  [4] "Smiths Chip Thinly  S/Cream&Onion 175g"
##  [5] "Kettle Tortilla ChpsHny&Jlpno Chili 150g"
##  [6] "Old El Paso Salsa   Dip Tomato Mild 300g"
##  [7] "Smiths Crinkle Chips Salt & Vinegar 330g"
##  [8] "Grain Waves         Sweet Chilli 210g"
##  [9] "Doritos Corn Chip Mexican Jalapeno 150g"
## [10] "Grain Waves Sour    Cream&Chives 210G"
```

```r
summary(transactionData$PROD_NAME)
```

```
##    Length     Class      Mode
##    264836 character character
```

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 summarizing 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

# make dataframe of all chars in all data names
productWords <- data.table(unlist(strsplit(unique(transactionData[, PROD_NAME]), "")))

# set column name as words
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.

#### Removing digits
productWords <- productWords[!grepl("\\d",productWords$words)]


#### Removing special characters
# remove punct
productWords <- productWords[!grepl("[[:punct:]]",productWords$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

# make vector of each word by extracting each sequence of letters surrounded by spaces
# for each entry in vector char, if entry is space, then everything before it is a word
word_vec <- character(0)
beg_indx <- 1
for(i in seq_along(productWords$words))
{
  # if space
  if (productWords$words[i] == " ")
  {
    # make all characters appended to word
    str_word <- ""
    for(j in (seq_len(i - beg_indx) + beg_indx - 1))
    {
      str_word <- paste0(str_word,productWords$words[j])
    }

    # add word to word_vec
```

```r
    word_vec <- c(word_vec,str_word)

    # afterward, change beginning index to index after i
    beg_indx <- i + 1
  }

  # else, increment counter
}

# with word_vec, count most common words, exclude ""
summary_word_vec <- sort(table(word_vec))
summary_word_vec <- rev(summary_word_vec[-length(summary_word_vec )])

# print out sorted vector
summary_word_vec
```

```
## word_vec
##              Chips              gSmiths                  Cut
##                 21                   14                   14
##            Crinkle               Cheese                 Salt
##                 14                   12                   11
##            gKettle             Original                Salsa
##                 11                   10                    9
##           gDoritos                 Chip                 gRRD
##                  9                    9                    8
##               Corn                  gWW            gPringles
##                  8                    7                    7
##            Chicken                 Sour                  Sea
##                  7                    6                    6
##             Chilli              Vinegar               Thinly
##                  6                    5                    5
##             gThins               Crisps              Supreme
##                  5                    5                    4
##               Rock                 gRed           gInfuzions
##                  4                    4                    4
##               Deli                Cream             Tortilla
##                  4                    4                    3
##             Tomato                Sweet                  Soy
##                  3                    3                    3
##         Sensations                 Popd                 Paso
##                  3                    3                    3
##               Mild                 Lime          gWoolworths
##                  3                    3                    3
##          gTwisties            gTostitos                 gOld
##                  3                    3                    3
##           gNatural                   El                  Dip
##                  3                    3                    3
##             Chives                Waves                 Thai
##                  3                    2                    2
##              Tangy                  Swt                   SR
##                  2                    2                    2
##             Salted                Rings               Potato
##                  2                    2                    2
```

```
##           Nacho          Medium         Lightly
##               2               2               2
##           Honey       gTyrrells          gSmith
##               2               2               2
##          gGrain           gCobs        gCheetos
##               2               2               2
##            gCCs          ChipCo             BBQ
##               2               2               2
##             And          Whlgrn         Whlegrn
##               2               1               1
##          Vingar          Vinegr             Veg
##               1               1               1
##         Truffle             Tom           Tmato
##               1               1               1
##           Tasty        SwtChlli       SweetSpcy
##               1               1               1
##      SweetChili           Strws           Sthrn
##               1               1               1
##           Steak         Stacked        SrCream
##               1               1               1
##          Splash           Spicy              Sp
##               1               1               1
##        Southern  SourCreamHerbs       SourCream
##               1               1               1
##       SnagSauce          Smoked             Slt
##               1               1               1
##            Slow         Siracha Seasonedchicken
##               1               1               1
##     SeaSaltgCCs     SCreamOnion    SaltPringles
##               1               1               1
##           saltd             Rst           Roast
##               1               1               1
##             Rib           Puffs           Prawn
##               1               1               1
##       PotatoMix             Pot            Pork
##               1               1               1
##            Plus           Pesto          Pepper
##               1               1               1
##              Pc        Papadums         Originl
##               1               1               1
##           Orgnl    OnionStacked        OnionDip
##               1               1               1
##           Onion              Of         Natural
##               1               1               1
##               N         Mzzrlla         Mystery
##               1               1               1
##      Mozzarella        Mexicana         Mexican
##               1               1               1
##             Med           Mango             Mac
##               1               1               1
##           Light             Jam        Jalapeno
##               1               1               1
##         HtgCobs        HrbSpce             Hot
##               1               1               1
```

```
##                 Hony              gSunbites                 gSnbts
##                    1                      1                      1
##              GSmiths                   gNCC                GKettle
##                    1                      1                      1
##              gInfzns               gGrnWves              gCheezels
##                    1                      1                      1
##              Gcamole                gBurger                 Garden
##                    1                      1                      1
##                    g                  Fries            FriedChicken
##                    1                      1                      1
##               French               FrchOnin                Flavour
##                    1                      1                      1
##                  Fig                 DStyle CutSaltVinegrgCheezels
##                    1                      1                      1
##               Crnkle               Crnchers                    Crn
##                    1                      1                      1
##                  Crm                  Crips            CreamChives
##                    1                      1                      1
##             Crackers                 Compny                Coconut
##                    1                      1                      1
##                   Co                 Chutny       ChsOniongFrench
##                    1                      1                      1
##                  Chs           ChpsHnyJlpno          ChpsFetaGarlic
##                    1                      1                      1
##     ChpsBtrootRicotta                    Chp                  Chnky
##                    1                      1                      1
##     ChliSCreamGKettle               Chipotle            Chimuchurri
##                    1                      1                      1
##            ChilliLime                  Chili            ChiknGarlic
##                    1                      1                      1
##         ChickengSmiths            CheddrMstrd                   Ched
##                    1                      1                      1
##          ChckngDorito              Camembert                 Burger
##                    1                      1                      1
##               Btroot                    Box              Bolognese
##                    1                      1                      1
##                  Big                  Belly               BBQMaple
##                    1                      1                      1
##                Basil               Barbeque               Barbecue
##                    1                      1                      1
##                Balls                    Bag                  Bacon
##                    1                      1                      1
##                Aioli
##                    1
```

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).

```
#### 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  
```

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.

```
#### Filter the dataset to find the outlier
# Over to you! Use a filter to examine the transactions in question.

transactionData[transactionData$PROD_QTY == 200.000]
```

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

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

```
#### Let's see if the customer has had other transactions
# Over to you! Use a filter to see what other transactions that customer made.

transactionData[transactionData$LYLTY_CARD_NBR == 226000]
```

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

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!

transactionData <- transactionData[!transactionData$LYLTY_CARD_NBR == 226000]

#### Re-examine transaction data
# 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 : 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
```

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.

# get vec of dates in data, for each entry in data
count_trans <- numeric(0)
for(i in seq_along(unique(transactionData$DATE)))
{
  # filter by each date, get row count, then place count in result vec
  count_trans[i] <- nrow(transactionData[transactionData$DATE ==
↪  unique(transactionData$DATE)[i]])
}

# rename names of count_trans
names(count_trans) <- unique(transactionData$DATE)

# sort transaction dates
names(count_trans) <- sort(names(count_trans))

# get number of dates
length(count_trans)
```

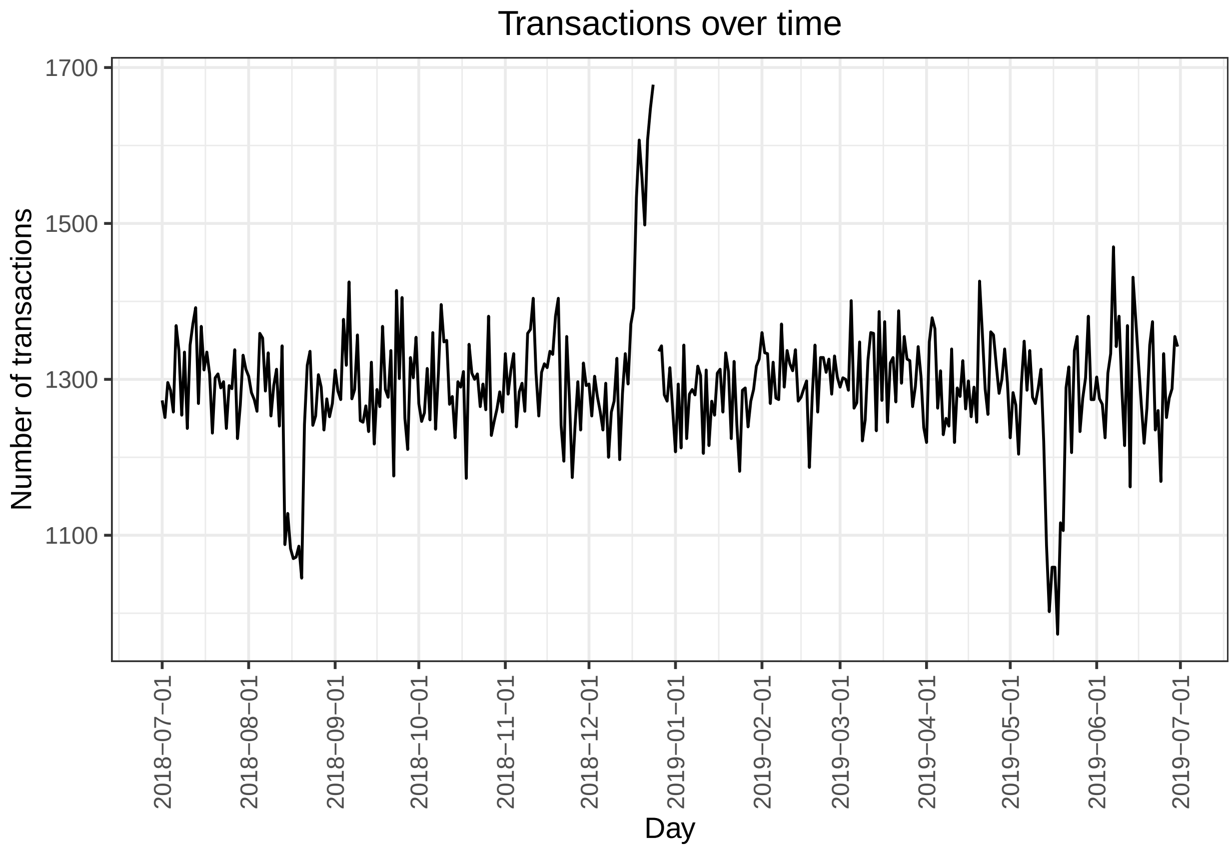
```
## [1] 364
```

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.

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

dates_seq <- character(0)

for(i in seq_len(365))
{
  dates_seq <- c(dates_seq, format(as.Date(i, origin = "2018-06-30"), "%Y-%m-%d"))
}

transactions_by_day <- transactionData %>% group_by(DATE) %>% summarise(sum(PROD_QTY))
names(transactions_by_day)
```

```
## [1] "DATE"          "sum(PROD_QTY)"
```

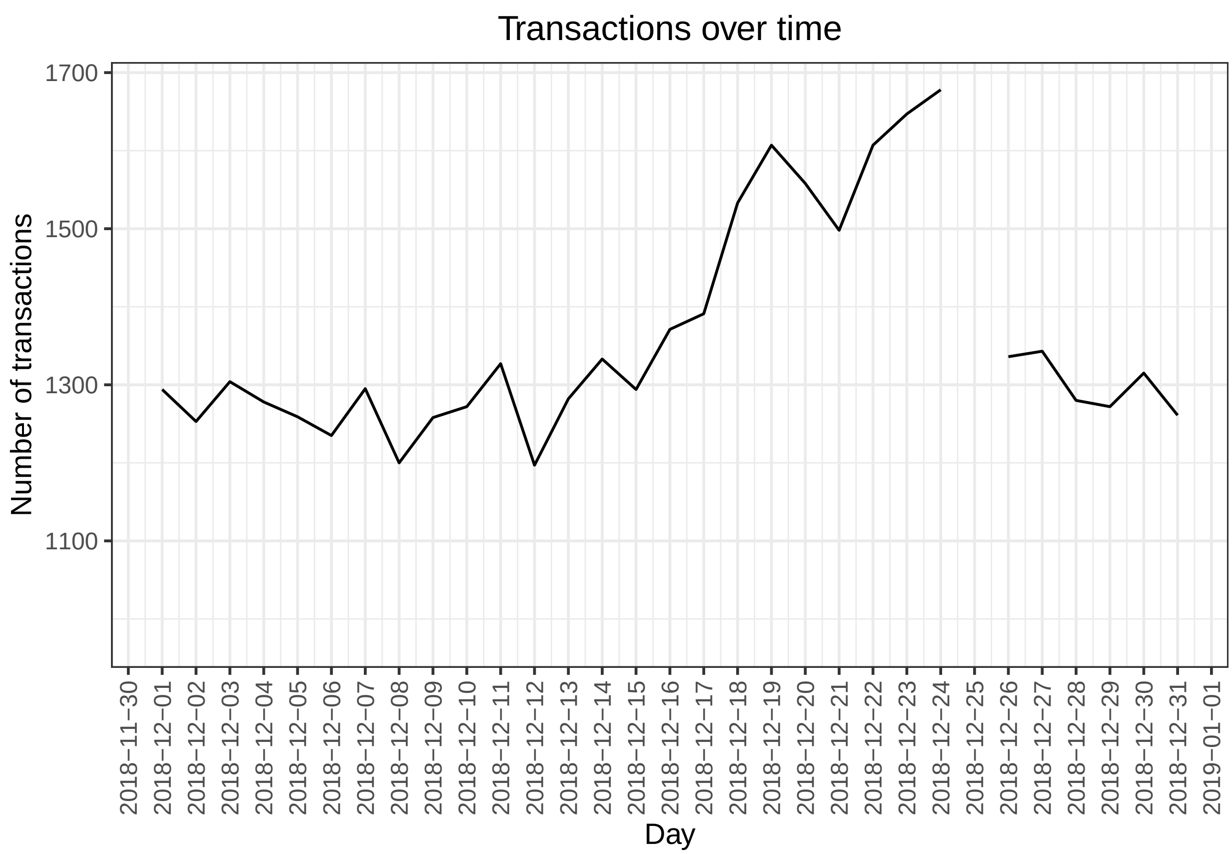
```
# make full transactions dataframe
full_transactions_by_day <- list()
full_transactions_by_day$DATE <- as.Date(dates_seq)
full_transactions_by_day <- data.frame(full_transactions_by_day)
transactions_by_day <- left_join(full_transactions_by_day,transactions_by_day, by =
↪  "DATE")

#### Setting plot themes to format graphs
theme_set(theme_bw())
theme_update(plot.title = element_text(hjust = 0.5))

# change col names
names(transactions_by_day) <- c("DATE","COUNT")

#### Plot transactions over time
ggplot(transactions_by_day, aes(x = DATE, y = 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.

```
#### Filter to December and look at individual days
# Over to you - recreate the chart above zoomed in to the relevant dates.

ggplot(transactions_by_day, aes(x = DATE, y = COUNT)) +
 geom_line() +
 labs(x = "Day", y = "Number of transactions", title = "Transactions over time") +
 scale_x_date(breaks = "1 day", limits = as.Date(c("2018-12-01","2018-12-31"))) +
 theme(axis.text.x = element_text(angle = 90, vjust = 0.5))
```

```
## Warning: Removed 334 rows containing missing values (`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)]
```

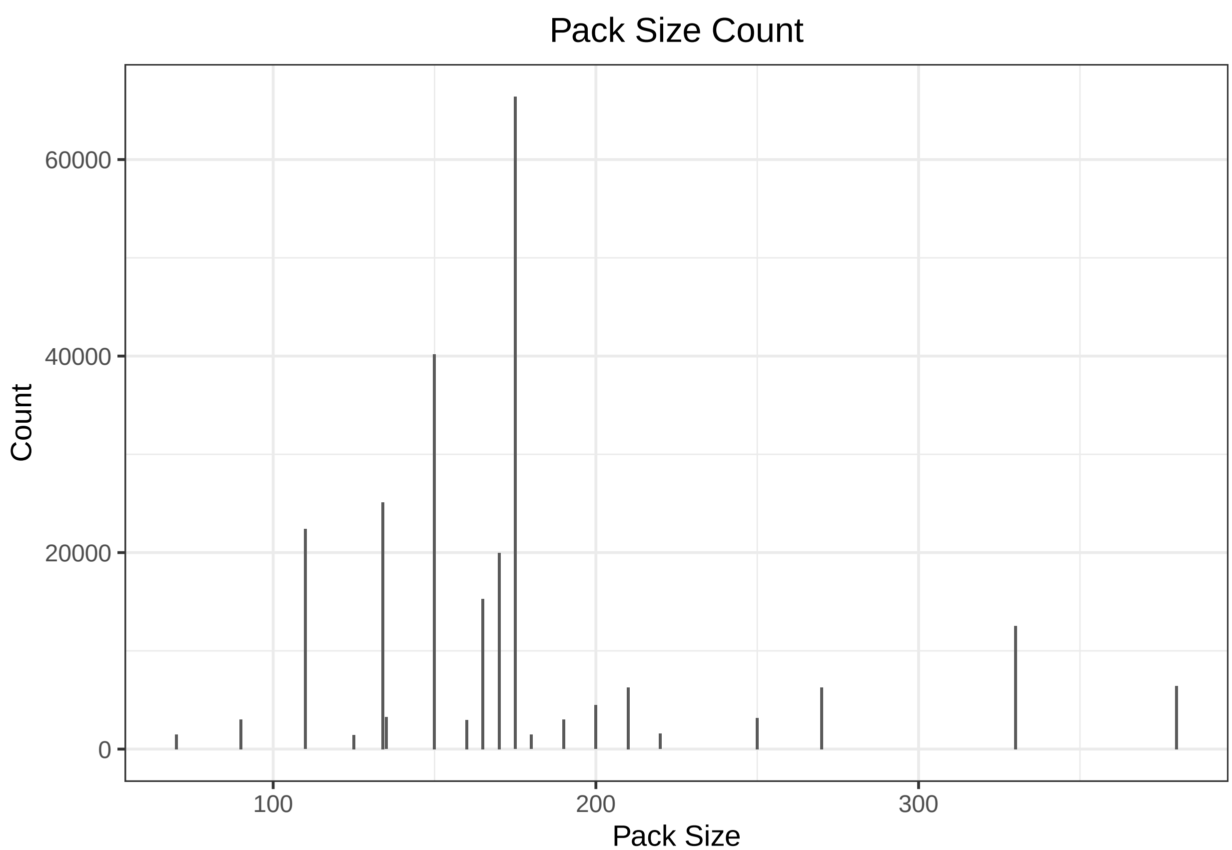
```
##     PACK_SIZE     N
##  1:        70  1507
##  2:        90  3008
##  3:       110 22387
##  4:       125  1454
##  5:       134 25102
##  6:       135  3257
##  7:       150 40203
##  8:       160  2970
##  9:       165 15297
## 10:       170 19983
## 11:       175 66390
## 12:       180  1468
```

```
## 13:       190  2995
## 14:       200  4473
## 15:       210  6272
## 16:       220  1564
## 17:       250  3169
## 18:       270  6285
## 19:       330 12540
## 20:       380  6416
```

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

```
#### 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.

ggplot(transactionData[, .N, PACK_SIZE]) + geom_histogram(aes(x = PACK_SIZE, y = N),
↪  stat="identity") + labs(x = "Pack Size", y = "Count", title = "Pack Size Count")
```

```
## Warning in geom_histogram(aes(x = PACK_SIZE, y = N), stat = "identity"):
## Ignoring unknown parameters: `binwidth`, `bins`, and `pad`
```



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.

# subset the first word from the name as the brand
# parse through names of each row and strsplit
transactionData <- transactionData %>% mutate(BRAND = str_split(PROD_NAME, " ", simplify
↪  = TRUE)[,1])

#### Checking brands
# Over to you! Check the results look reasonable.

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"
```

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"]
# Over to you! Add any additional brand adjustments you think may be required.

transactionData[BRAND == "Infuzions", BRAND := "Infzns"]

transactionData[BRAND == "Woolworths", BRAND := "WW"]

transactionData[BRAND == "Natural", BRAND := "NCC"]

transactionData[BRAND == "Grain", BRAND := "GrnWves"]

transactionData[BRAND == "Sunbites", BRAND := "Snbts"]

transactionData[BRAND == "Smith", BRAND:="Smiths"]

#### Check again
# Over to you! Check the results look reasonable.

unique(transactionData[,"BRAND"])
```

```
##        BRAND
##  1:      NCC
##  2:      CCs
##  3:   Smiths
##  4:   Kettle
##  5:  GrnWves
##  6:  Doritos
```

```
##  7: Twisties
##  8:       WW
##  9:    Thins
## 10:   Burger
## 11: Cheezels
## 12:   Infzns
## 13:      Red
## 14: Pringles
## 15:   Dorito
## 16: Tyrrells
## 17:     Cobs
## 18:   French
## 19:      RRD
## 20: Tostitos
## 21:  Cheetos
## 22:    Snbts
##         BRAND
```

Examining customer data

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

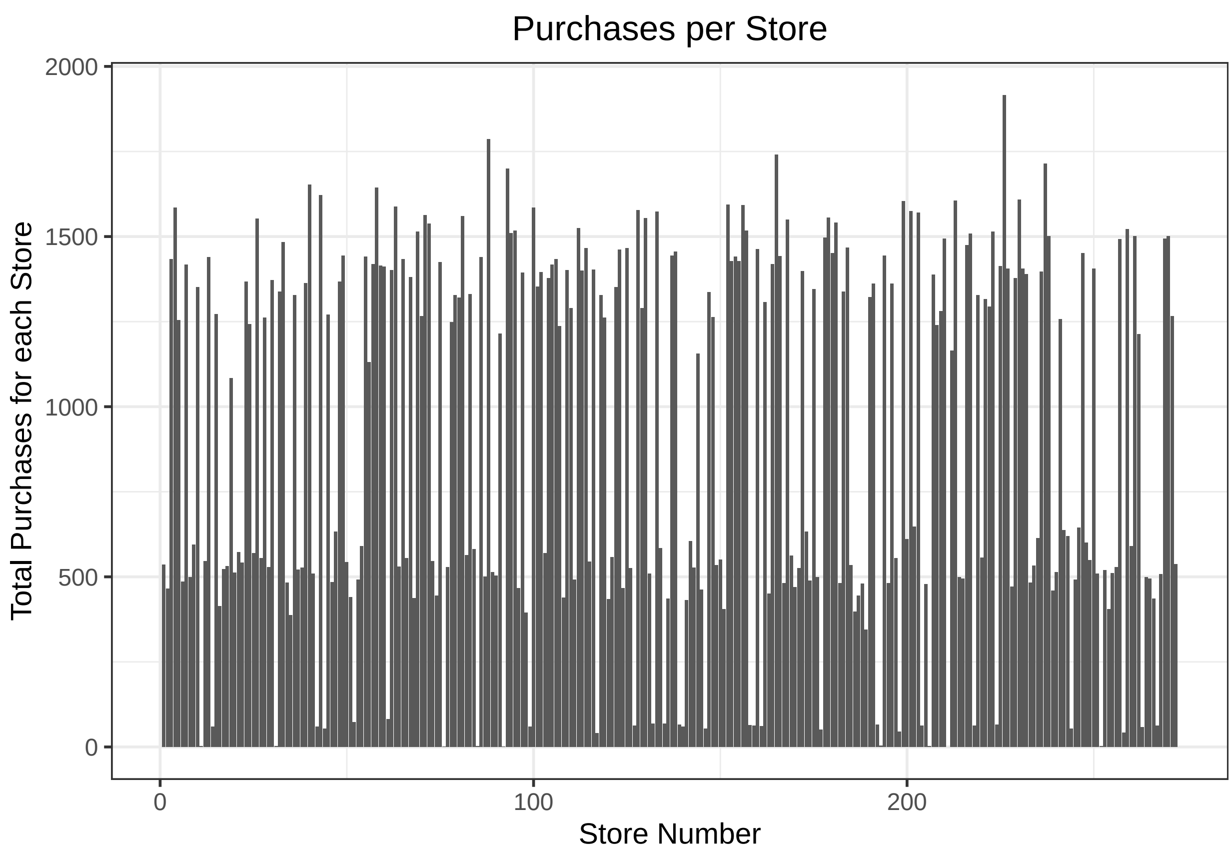
```
#### Examining customer data
# Over to you! Do some basic summaries of the dataset, including distributions of any key
↪  columns.

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 : 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
##    PACK_SIZE        BRAND
##  Min.   : 70.0   Length:246740
##  1st Qu.:150.0   Class :character
##  Median :170.0   Mode  :character
##  Mean   :175.6
##  3rd Qu.:175.0
##  Max.   :380.0
```

```
# get distribution for total purchase for each store
store_nbr_dist <- table(transactionData$STORE_NBR)
```

```r
# graph
ggplot() + geom_bar(aes(x = as.numeric(names(store_nbr_dist)), y = store_nbr_dist), stat
↪   = "identity") + labs(x = "Store Number", y = "Total Purchases for each Store", title
↪   = "Purchases per Store")
```

```
## Don't know how to automatically pick scale for object of type <table>.
## Defaulting to continuous.
```




```r
# store number 271 is missing
```

```r
#### 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.

```r
# Over to you! See if any transactions did not have a matched customer.

for(col in colnames(transactionData))
```

```r
{
  print(any(is.na(data[,..col])))
  print(any(is.null(data[,..col])))
}
```

```
## [1] FALSE
## [1] FALSE
## [1] FALSE
## [1] FALSE
## [1] FALSE
## [1] FALSE
## [1] FALSE
## [1] FALSE
## [1] FALSE
## [1] FALSE
## [1] FALSE
## [1] FALSE
## [1] FALSE
## [1] FALSE
## [1] FALSE
## [1] FALSE
## [1] FALSE
## [1] FALSE
## [1] FALSE
## [1] FALSE
```

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

```r
fwrite(data, paste0(filePath,"QVI_data.csv"))
```

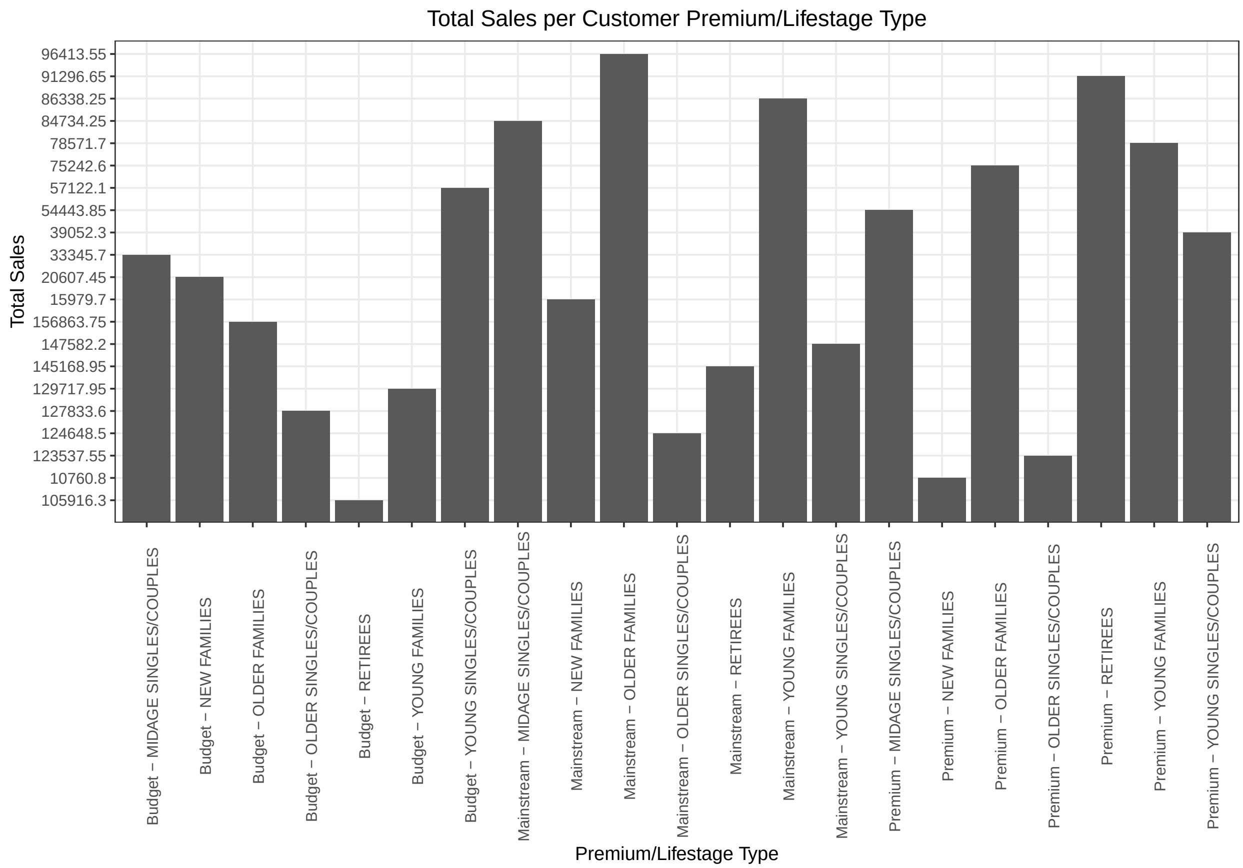
Data exploration is now complete!

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.

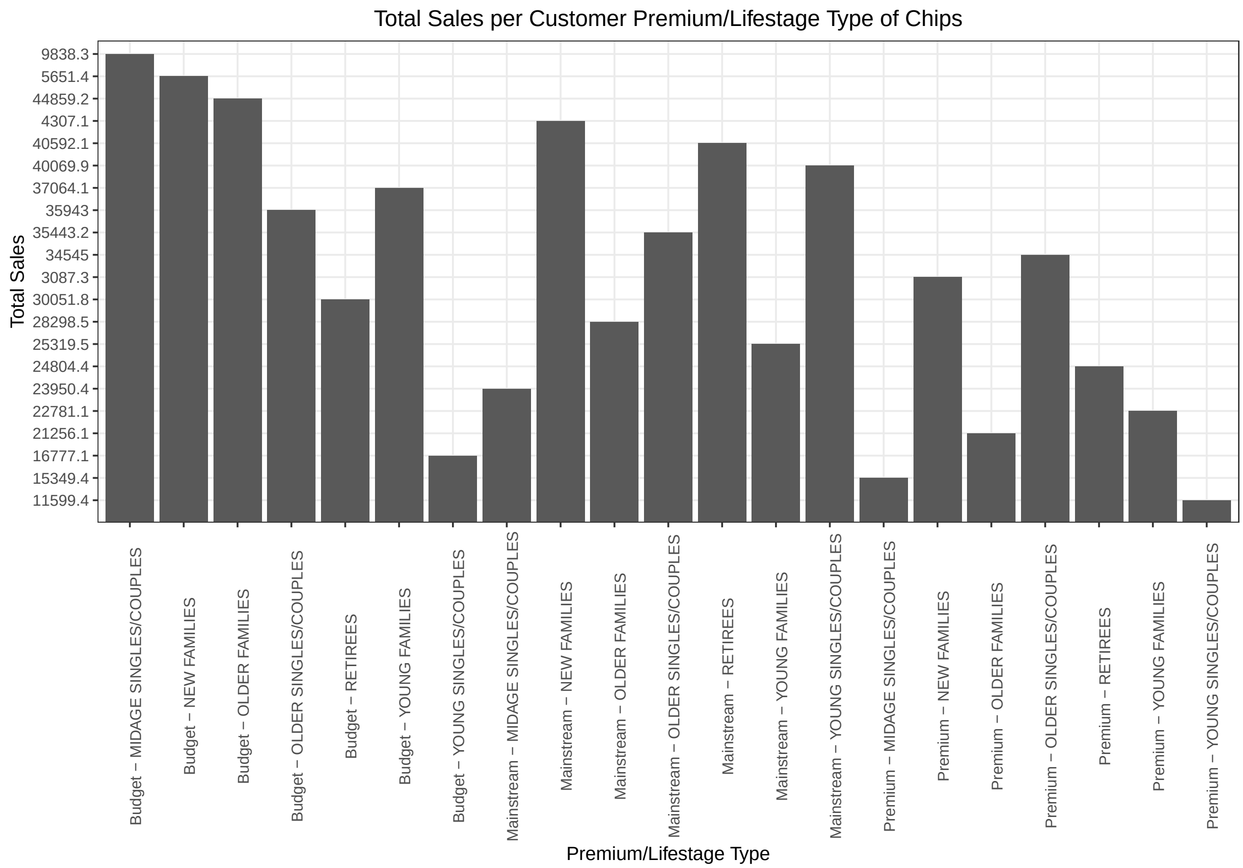
```r
#### Total sales by LIFESTAGE and PREMIUM_CUSTOMER
# Over to you! Calculate the summary of sales by those dimensions and create a plot.

cust_seg <- data[,c("TOT_SALES","LIFESTAGE","PREMIUM_CUSTOMER")]
```

```r
# parse through info, for each premium type and lifestyle
sumPremLife <- function(df)
{

  # make data frame holding:
  # total sales, customer segment
  res <- data.frame(matrix(ncol = 3))

  for(premStat in unique(df$PREMIUM_CUSTOMER))
  {
    for(lifeStat in unique(df$LIFESTAGE))
    {
      # filter data by premstat and lifestat
      m <- df %>% filter(PREMIUM_CUSTOMER == premStat & LIFESTAGE == lifeStat)

      # sum total sales
      totSales <- sum(m$TOT_SALES)

      # add onto matrix
      res <- rbind(res,c(paste0(premStat, " - ", lifeStat), totSales, nrow(m)))

    }
  }

  # remove first row
  res <- res[2:nrow(res),]
  # reset indices
  rownames(res) <- NULL
  # name columns
  colnames(res) <- c("Customer_Segment","Total_Sales", "N_Customer")

  return(res)
}

segdata <- sumPremLife(cust_seg)


# plot
ggplot(segdata) + geom_bar(aes(x = Customer_Segment, y = Total_Sales), stat = "identity")
↪  + theme(axis.text.x = element_text(angle = 90)) + labs(title = "Total Sales per
↪  Customer Premium/Lifestage Type", x = "Premium/Lifestage Type", y = "Total Sales")
```



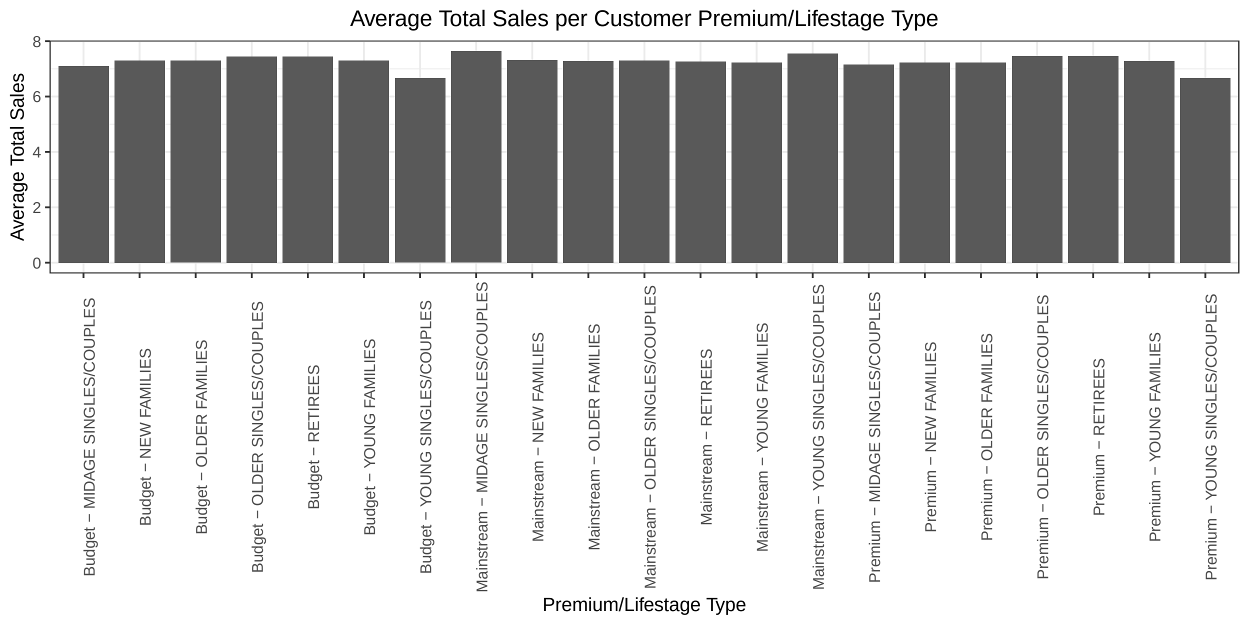
Sales are coming mainly from Budget - older families, Mainstream - young singles/couples, and Mainstream - Older retirees. Let's see if the higher sales are due to there being more customers who buy chips.

```
#### 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.

# subset all products with "chip" in product name
cust_seg_chip <-
↪  data[grepl("CHIP",toupper(data$PROD_NAME)),c("TOT_SALES","LIFESTAGE","PREMIUM_CUSTOMER")]

# prem life sum sales based on chip
chip_premlife <- sumPremLife(cust_seg_chip)

# plot
ggplot(chip_premlife) + geom_bar(aes(x = Customer_Segment, y = Total_Sales), stat =
↪  "identity") + theme(axis.text.x = element_text(angle = 90)) + labs(title = "Total
↪  Sales per Customer Premium/Lifestage Type of Chips", x = "Premium/Lifestage Type", y
↪  = "Total Sales")
```



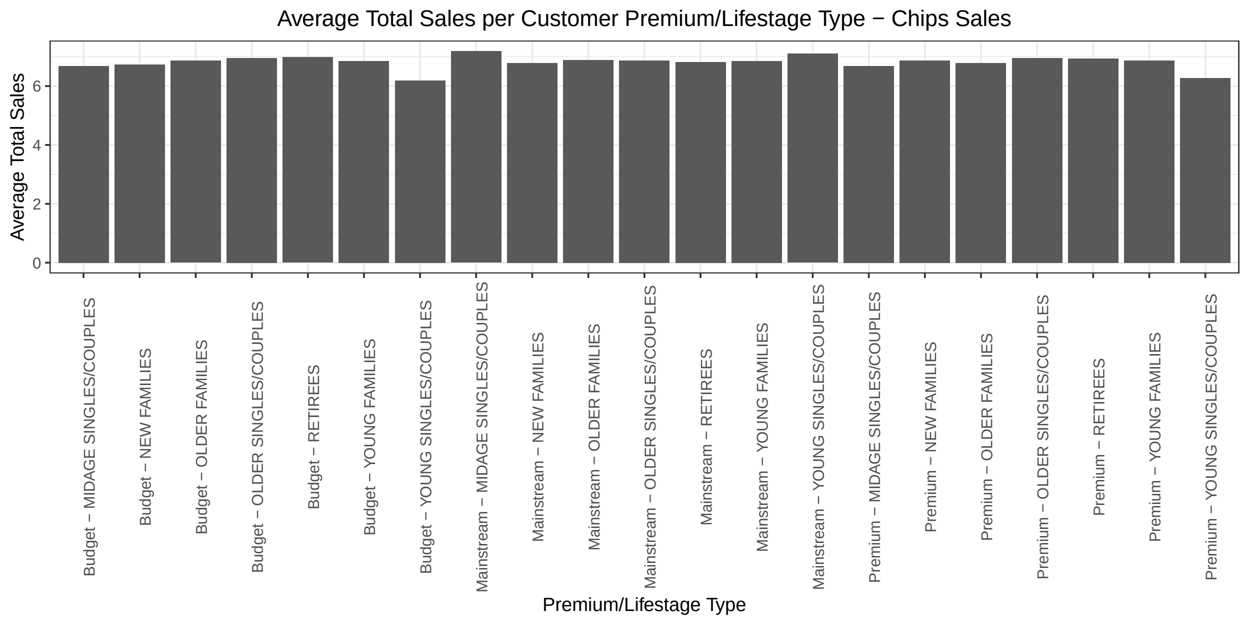
There are more Budget - Older families, Mainstream - young singles/couples and Mainstream - retirees who buy chips. This contributes to there being more sales to these customer segments but this is not a major driver for the Budget - Older families segment. Higher sales may also be driven by more units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

# change into numeric
segdata$Total_Sales <- as.numeric(segdata$Total_Sales)
segdata$N_Customer <- as.numeric(segdata$N_Customer)

# mutate for avg column
segdata_v1 <- segdata %>% mutate(Avg_Total = Total_Sales / N_Customer)

# plot
ggplot(segdata_v1) + geom_bar(aes(x = Customer_Segment, y = Avg_Total), stat =
↪  "identity") + theme(axis.text.x = element_text(angle = 90)) + labs(title = "Average
↪  Total Sales per Customer Premium/Lifestage Type", x = "Premium/Lifestage Type", y =
↪  "Average Total Sales")
```



Mainstream Midage families and Young singles/couples 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.

```
#### 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.

# update to numeric versions
chip_premlife <- chip_premlife %>% transform(Total_Sales = as.numeric(Total_Sales),
↪  N_Customer = as.numeric(N_Customer))

# mutate average column
chip_premlife_v1 <- chip_premlife %>% mutate(Avg_Total = Total_Sales / N_Customer)

# plot
ggplot(chip_premlife_v1) + geom_bar(aes(x = Customer_Segment, y = Avg_Total), stat =
↪  "identity") + theme(axis.text.x = element_text(angle = 90)) + labs(title = "Average
↪  Total Sales per Customer Premium/Lifestage Type - Chips Sales", x =
↪  "Premium/Lifestage Type", y = "Average Total Sales")
```



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.

```
#### Perform an independent t-test between mainstream vs premium and budget midage and
#### young singles and couples
# Over to you! Perform a t-test to see if the difference is significant.

# get dataframe of chip purchases of all midage and young singles/couples
chip_t_test <- cust_seg_chip[LIFESTAGE == "YOUNG SINGLES/COUPLES" | LIFESTAGE == "MIDAGE
↪  SINGLES/COUPLES",]

# mainstream filter
ms <- chip_t_test[PREMIUM_CUSTOMER == "Mainstream"]

# not mainstream filter
nms <- chip_t_test[PREMIUM_CUSTOMER != "Mainstream"]

# test mainstream vs not mainstream
# with equal variance
t.test(x = ms$TOT_SALES, y = nms$TOT_SALES, alternative = "g", var.equal = T, conf.level
↪  = 0.95)
```

```
##
##  Two Sample t-test
##
## data:  ms$TOT_SALES and nms$TOT_SALES
## t = 19.86, df = 17303, p-value < 2.2e-16
## alternative hypothesis: true difference in means is greater than 0
```

```
## 95 percent confidence interval:
##  0.6432478       Inf
## sample estimates:
## mean of x mean of y
##  7.132386  6.431048
```

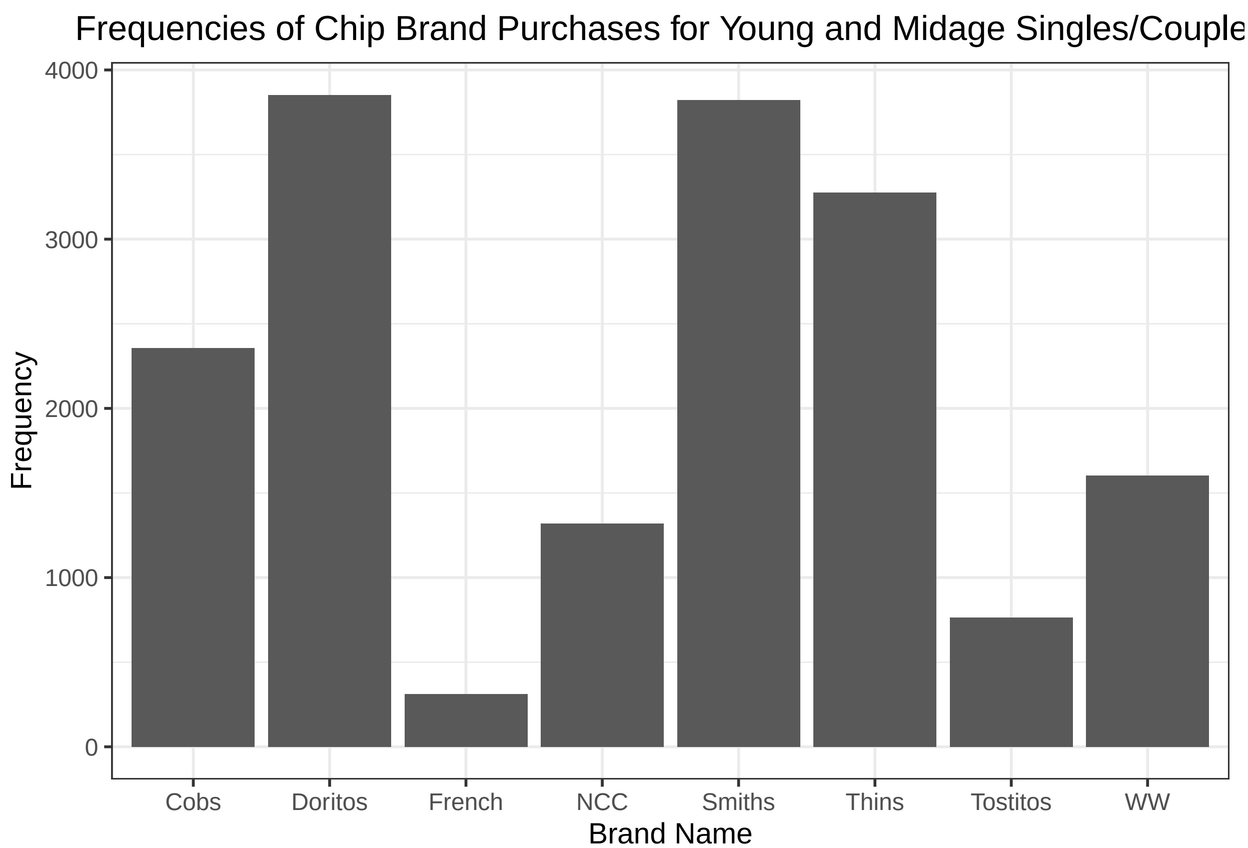
```r
# not equal variance
t.test(x = ms$TOT_SALES, y = nms$TOT_SALES, alternative = "g", var.equal = F, conf.level
↪  = 0.95)
```

```
##
##  Welch Two Sample t-test
##
## data:  ms$TOT_SALES and nms$TOT_SALES
## t = 19.842, df = 17136, p-value < 2.2e-16
## alternative hypothesis: true difference in means is greater than 0
## 95 percent confidence interval:
##  0.6431959       Inf
## sample estimates:
## mean of x mean of y
##  7.132386  6.431048
```

The t-test results in a p-value of < 2.2e-16, 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)

# make frequency plot of different brands for young mid age singles/couples
life_prem_brand_data <- data[(LIFESTAGE == "YOUNG SINGLES/COUPLES" | LIFESTAGE == "MIDAGE
↪  SINGLES/COUPLES") &
↪  grepl("CHIP",toupper(data$PROD_NAME)),c("LYLTY_CARD_NBR","PROD_NAME","LIFESTAGE","PREMIUM_CUSTOMER"

freq_brand <- table(life_prem_brand_data$BRAND)

ggplot() + geom_bar(aes(x = names(freq_brand), y = freq_brand), stat="identity") +
↪  labs(title = "Frequencies of Chip Brand Purchases for Young and Midage
↪  Singles/Couples", x = "Brand Name", y = "Frequency")
```

```
## Don't know how to automatically pick scale for object of type <table>.
## Defaulting to continuous.
```




```r
sum(freq_brand)
```

```
## [1] 17305
```

```r
# prepare data to convert into transactions for apriori

# get for young

young_brand <- life_prem_brand_data[LIFESTAGE == "YOUNG SINGLES/COUPLES","BRAND"]
young_trans <- life_prem_brand_data[LIFESTAGE == "YOUNG
↪  SINGLES/COUPLES","LYLTY_CARD_NBR"]

# change into factors
young_brand <- factor(young_brand$BRAND)
young_trans <- factor(young_trans$LYLTY_CARD_NBR)

young_transactions <- split(young_brand,young_trans)

young_rules <- apriori(young_transactions, parameter = list(supp = 0.0001, conf = 0.8))
```

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

```
## Apriori
##
```

```
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.8    0.1    1 none FALSE            TRUE       5   1e-04      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: 0
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[8 item(s), 7170 transaction(s)] done [0.00s].
## sorting and recoding items ... [8 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 5 done [0.00s].
## writing ... [15 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

```
# remove empty col
young_rules_df <- inspect(young_rules) %>% select(-2)
```

```
##      lhs                                 rhs       support    confidence
## [1]  {French, Smiths, Tostitos}       => {NCC}     0.00013947 1
## [2]  {Cobs, Tostitos, WW}             => {NCC}     0.00013947 1
## [3]  {Doritos, NCC, Tostitos}         => {WW}      0.00013947 1
## [4]  {NCC, Thins, Tostitos}           => {Smiths}  0.00013947 1
## [5]  {Cobs, Tostitos, WW}             => {Smiths}  0.00013947 1
## [6]  {French, NCC, Thins, WW}         => {Smiths}  0.00027894 1
## [7]  {French, NCC, Smiths, WW}        => {Thins}   0.00027894 1
## [8]  {Doritos, French, NCC, Smiths}   => {Thins}   0.00013947 1
## [9]  {Doritos, French, Smiths, Thins} => {NCC}     0.00013947 1
## [10] {Cobs, NCC, Tostitos, WW}        => {Smiths}  0.00013947 1
## [11] {NCC, Smiths, Tostitos, WW}      => {Cobs}    0.00013947 1
## [12] {Cobs, NCC, Smiths, Tostitos}    => {WW}      0.00013947 1
## [13] {Cobs, Smiths, Tostitos, WW}     => {NCC}     0.00013947 1
## [14] {Cobs, NCC, Thins, WW}           => {Doritos} 0.00013947 1
## [15] {Cobs, Doritos, NCC, Thins}      => {WW}      0.00013947 1
##      coverage   lift     count
## [1]  0.00013947 9.862448 1
## [2]  0.00013947 9.862448 1
## [3]  0.00013947 8.405627 1
## [4]  0.00013947 3.567164 1
## [5]  0.00013947 3.567164 1
## [6]  0.00027894 3.567164 2
## [7]  0.00027894 3.952591 2
## [8]  0.00013947 3.952591 1
## [9]  0.00013947 9.862448 1
## [10] 0.00013947 3.567164 1
## [11] 0.00013947 5.395034 1
## [12] 0.00013947 8.405627 1
## [13] 0.00013947 9.862448 1
```

```
## [14] 0.00013947 3.437200 1
## [15] 0.00013947 8.405627 1
```

```r
# get rules of young, focus on support for more frequent combinations
young_rules_df %>% arrange(desc(support))
```

```
##                                    lhs       rhs    support confidence
## [6]          {French, NCC, Thins, WW}  {Smiths} 0.00027894          1
## [7]         {French, NCC, Smiths, WW}   {Thins} 0.00027894          1
## [1]        {French, Smiths, Tostitos}     {NCC} 0.00013947          1
## [2]              {Cobs, Tostitos, WW}     {NCC} 0.00013947          1
## [3]          {Doritos, NCC, Tostitos}      {WW} 0.00013947          1
## [4]            {NCC, Thins, Tostitos}  {Smiths} 0.00013947          1
## [5]              {Cobs, Tostitos, WW}  {Smiths} 0.00013947          1
## [8]    {Doritos, French, NCC, Smiths}   {Thins} 0.00013947          1
## [9]  {Doritos, French, Smiths, Thins}     {NCC} 0.00013947          1
## [10]        {Cobs, NCC, Tostitos, WW}  {Smiths} 0.00013947          1
## [11]      {NCC, Smiths, Tostitos, WW}    {Cobs} 0.00013947          1
## [12]    {Cobs, NCC, Smiths, Tostitos}      {WW} 0.00013947          1
## [13]     {Cobs, Smiths, Tostitos, WW}     {NCC} 0.00013947          1
## [14]           {Cobs, NCC, Thins, WW} {Doritos} 0.00013947          1
## [15]      {Cobs, Doritos, NCC, Thins}      {WW} 0.00013947          1
##        coverage     lift count
## [6]  0.00027894 3.567164     2
## [7]  0.00027894 3.952591     2
## [1]  0.00013947 9.862448     1
## [2]  0.00013947 9.862448     1
## [3]  0.00013947 8.405627     1
## [4]  0.00013947 3.567164     1
## [5]  0.00013947 3.567164     1
## [8]  0.00013947 3.952591     1
## [9]  0.00013947 9.862448     1
## [10] 0.00013947 3.567164     1
## [11] 0.00013947 5.395034     1
## [12] 0.00013947 8.405627     1
## [13] 0.00013947 9.862448     1
## [14] 0.00013947 3.437200     1
## [15] 0.00013947 8.405627     1
```

```r
# get for midage
mid_brand <- life_prem_brand_data[LIFESTAGE == "MIDAGE SINGLES/COUPLES","BRAND"]
mid_trans <- life_prem_brand_data[LIFESTAGE == "MIDAGE SINGLES/COUPLES","LYLTY_CARD_NBR"]

mid_brand <- factor(mid_brand$BRAND)
mid_trans <- factor(mid_trans$LYLTY_CARD_NBR)

mid_transactions <- split(mid_brand, mid_trans)

mid_rules <- apriori(mid_transactions,parameter = list(supp = 0.0001, conf = 0.8))
```

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

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.8    0.1    1 none FALSE            TRUE       5   1e-04      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: 0
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[8 item(s), 4289 transaction(s)] done [0.00s].
## sorting and recoding items ... [8 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 5 6 done [0.00s].
## writing ... [28 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

```
mid_rules_df <- inspect(mid_rules) %>% select(-2)
```

```
##      lhs                                         rhs       support
## [1]  {Doritos, French, NCC}                   => {Thins}   0.0002331546
## [2]  {Cobs, Doritos, French}                  => {WW}      0.0002331546
## [3]  {Cobs, NCC, Tostitos}                    => {WW}      0.0002331546
## [4]  {NCC, Tostitos, WW}                      => {Smiths}  0.0004663092
## [5]  {Cobs, NCC, Tostitos}                    => {Smiths}  0.0002331546
## [6]  {NCC, Thins, Tostitos}                   => {Doritos} 0.0004663092
## [7]  {Cobs, Tostitos, WW}                     => {Smiths}  0.0006994637
## [8]  {French, NCC, Thins, WW}                 => {Smiths}  0.0002331546
## [9]  {Doritos, French, Thins, WW}             => {Smiths}  0.0002331546
## [10] {Cobs, NCC, Tostitos, WW}                => {Smiths}  0.0002331546
## [11] {Cobs, NCC, Smiths, Tostitos}            => {WW}      0.0002331546
## [12] {NCC, Thins, Tostitos, WW}               => {Doritos} 0.0002331546
## [13] {Doritos, NCC, Tostitos, WW}             => {Thins}   0.0002331546
## [14] {Doritos, Thins, Tostitos, WW}           => {NCC}     0.0002331546
## [15] {NCC, Thins, Tostitos, WW}               => {Smiths}  0.0002331546
## [16] {NCC, Smiths, Thins, Tostitos}           => {WW}      0.0002331546
## [17] {Smiths, Thins, Tostitos, WW}            => {NCC}     0.0002331546
## [18] {Doritos, NCC, Tostitos, WW}             => {Smiths}  0.0002331546
## [19] {NCC, Smiths, Thins, Tostitos}           => {Doritos} 0.0002331546
## [20] {Cobs, Doritos, Tostitos, WW}            => {Smiths}  0.0002331546
## [21] {Doritos, Thins, Tostitos, WW}           => {Smiths}  0.0002331546
## [22] {Smiths, Thins, Tostitos, WW}            => {Doritos} 0.0002331546
## [23] {Cobs, Doritos, NCC, Thins}              => {Smiths}  0.0002331546
## [24] {Doritos, NCC, Thins, Tostitos, WW}      => {Smiths}  0.0002331546
## [25] {NCC, Smiths, Thins, Tostitos, WW}       => {Doritos} 0.0002331546
## [26] {Doritos, NCC, Smiths, Tostitos, WW}     => {Thins}   0.0002331546
## [27] {Doritos, NCC, Smiths, Thins, Tostitos} => {WW}      0.0002331546
## [28] {Doritos, Smiths, Thins, Tostitos, WW}   => {NCC}     0.0002331546
```

```
##      confidence coverage     lift     count
## [1]  1          0.0002331546 3.613311 1
## [2]  1          0.0002331546 7.054276 1
## [3]  1          0.0002331546 7.054276 1
## [4]  1          0.0004663092 3.142125 2
## [5]  1          0.0002331546 3.142125 1
## [6]  1          0.0004663092 3.076758 2
## [7]  1          0.0006994637 3.142125 3
## [8]  1          0.0002331546 3.142125 1
## [9]  1          0.0002331546 3.142125 1
## [10] 1          0.0002331546 3.142125 1
## [11] 1          0.0002331546 7.054276 1
## [12] 1          0.0002331546 3.076758 1
## [13] 1          0.0002331546 3.613311 1
## [14] 1          0.0002331546 8.560878 1
## [15] 1          0.0002331546 3.142125 1
## [16] 1          0.0002331546 7.054276 1
## [17] 1          0.0002331546 8.560878 1
## [18] 1          0.0002331546 3.142125 1
## [19] 1          0.0002331546 3.076758 1
## [20] 1          0.0002331546 3.142125 1
## [21] 1          0.0002331546 3.142125 1
## [22] 1          0.0002331546 3.076758 1
## [23] 1          0.0002331546 3.142125 1
## [24] 1          0.0002331546 3.142125 1
## [25] 1          0.0002331546 3.076758 1
## [26] 1          0.0002331546 3.613311 1
## [27] 1          0.0002331546 7.054276 1
## [28] 1          0.0002331546 8.560878 1
```

```
mid_rules_df %>% arrange(desc(support))
```

```
## lhs rhs support confidence
## [7] {Cobs, Tostitos, WW} {Smiths} 0.0006994637 1
## [4] {NCC, Tostitos, WW} {Smiths} 0.0004663092 1
## [6] {NCC, Thins, Tostitos} {Doritos} 0.0004663092 1
## [1] {Doritos, French, NCC} {Thins} 0.0002331546 1
## [2] {Cobs, Doritos, French} {WW} 0.0002331546 1
## [3] {Cobs, NCC, Tostitos} {WW} 0.0002331546 1
## [5] {Cobs, NCC, Tostitos} {Smiths} 0.0002331546 1
## [8] {French, NCC, Thins, WW} {Smiths} 0.0002331546 1
## [9] {Doritos, French, Thins, WW} {Smiths} 0.0002331546 1
## [10] {Cobs, NCC, Tostitos, WW} {Smiths} 0.0002331546 1
## [11] {Cobs, NCC, Smiths, Tostitos} {WW} 0.0002331546 1
## [12] {NCC, Thins, Tostitos, WW} {Doritos} 0.0002331546 1
## [13] {Doritos, NCC, Tostitos, WW} {Thins} 0.0002331546 1
## [14] {Doritos, Thins, Tostitos, WW} {NCC} 0.0002331546 1
## [15] {NCC, Thins, Tostitos, WW} {Smiths} 0.0002331546 1
## [16] {NCC, Smiths, Thins, Tostitos} {WW} 0.0002331546 1
## [17] {Smiths, Thins, Tostitos, WW} {NCC} 0.0002331546 1
## [18] {Doritos, NCC, Tostitos, WW} {Smiths} 0.0002331546 1
## [19] {NCC, Smiths, Thins, Tostitos} {Doritos} 0.0002331546 1
## [20] {Cobs, Doritos, Tostitos, WW} {Smiths} 0.0002331546 1
```

```
## [21] {Doritos, Thins, Tostitos, WW} {Smiths} 0.0002331546 1
## [22] {Smiths, Thins, Tostitos, WW} {Doritos} 0.0002331546 1
## [23] {Cobs, Doritos, NCC, Thins} {Smiths} 0.0002331546 1
## [24] {Doritos, NCC, Thins, Tostitos, WW} {Smiths} 0.0002331546 1
## [25] {NCC, Smiths, Thins, Tostitos, WW} {Doritos} 0.0002331546 1
## [26] {Doritos, NCC, Smiths, Tostitos, WW} {Thins} 0.0002331546 1
## [27] {Doritos, NCC, Smiths, Thins, Tostitos} {WW} 0.0002331546 1
## [28] {Doritos, Smiths, Thins, Tostitos, WW} {NCC} 0.0002331546 1
## coverage lift count
## [7] 0.0006994637 3.142125 3
## [4] 0.0004663092 3.142125 2
## [6] 0.0004663092 3.076758 2
## [1] 0.0002331546 3.613311 1
## [2] 0.0002331546 7.054276 1
## [3] 0.0002331546 7.054276 1
## [5] 0.0002331546 3.142125 1
## [8] 0.0002331546 3.142125 1
## [9] 0.0002331546 3.142125 1
## [10] 0.0002331546 3.142125 1
## [11] 0.0002331546 7.054276 1
## [12] 0.0002331546 3.076758 1
## [13] 0.0002331546 3.613311 1
## [14] 0.0002331546 8.560878 1
## [15] 0.0002331546 3.142125 1
## [16] 0.0002331546 7.054276 1
## [17] 0.0002331546 8.560878 1
## [18] 0.0002331546 3.142125 1
## [19] 0.0002331546 3.076758 1
## [20] 0.0002331546 3.142125 1
## [21] 0.0002331546 3.142125 1
## [22] 0.0002331546 3.076758 1
## [23] 0.0002331546 3.142125 1
## [24] 0.0002331546 3.142125 1
## [25] 0.0002331546 3.076758 1
## [26] 0.0002331546 3.613311 1
## [27] 0.0002331546 7.054276 1
## [28] 0.0002331546 8.560878 1
```

```r
# do it for the rest of the population

life_prem_brand_data_v2 <- data[(LIFESTAGE != "YOUNG SINGLES/COUPLES" & LIFESTAGE !=
↪  "MIDAGE SINGLES/COUPLES") &
↪  grepl("CHIP",toupper(data$PROD_NAME)),c("LYLTY_CARD_NBR","PROD_NAME","LIFESTAGE","PREMIUM_CUSTOMER"

# prepare data to convert into transactions for apriori

other_brand <- life_prem_brand_data_v2[,"BRAND"]
other_trans <- life_prem_brand_data_v2[,"LYLTY_CARD_NBR"]


# change into factor
other_brand <- factor(other_brand$BRAND)
other_trans <- factor(other_trans$LYLTY_CARD_NBR)
```

```r
# make transactions list
other_transactions <- split(other_brand,other_trans)

# apriori
other_rules <- apriori(other_transactions, parameter = list(supp = 0, conf = 0.8))
```

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

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.8    0.1    1 none FALSE            TRUE       5       0      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: 0
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[8 item(s), 32166 transaction(s)] done [0.00s].
## sorting and recoding items ... [8 item(s)] done [0.00s].
## creating transaction tree ... done [0.01s].
## checking subsets of size 1 2 3 4 5 6 done [0.00s].
## writing ... [62 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

```r
other_rules_df <- inspect(other_rules) %>% select(-2)
```

```
##      lhs                                        rhs       support
## [1]  {Doritos, French, Tostitos, WW}         => {NCC}     0.000000e+00
## [2]  {Cobs, French, Thins, Tostitos}         => {NCC}     0.000000e+00
## [3]  {Cobs, French, NCC, Tostitos}           => {Doritos} 3.108873e-05
## [4]  {Cobs, French, NCC, Tostitos}           => {Smiths}  3.108873e-05
## [5]  {Cobs, French, Thins, Tostitos}         => {WW}      0.000000e+00
## [6]  {Doritos, French, Tostitos, WW}         => {Cobs}    0.000000e+00
## [7]  {Doritos, French, Tostitos, WW}         => {Thins}   0.000000e+00
## [8]  {French, Thins, Tostitos, WW}           => {Smiths}  3.108873e-05
## [9]  {Doritos, French, Tostitos, WW}         => {Smiths}  0.000000e+00
## [10] {Cobs, French, Thins, Tostitos}         => {Doritos} 0.000000e+00
## [11] {Cobs, French, Thins, Tostitos}         => {Smiths}  0.000000e+00
## [12] {Cobs, Doritos, French, Tostitos}       => {Smiths}  6.217745e-05
## [13] {Cobs, French, Smiths, Tostitos}        => {Doritos} 6.217745e-05
## [14] {Cobs, French, NCC, Tostitos, WW}       => {Thins}   0.000000e+00
## [15] {French, NCC, Thins, Tostitos, WW}      => {Cobs}    0.000000e+00
## [16] {Cobs, French, NCC, Thins, Tostitos}    => {WW}      0.000000e+00
## [17] {Cobs, French, Thins, Tostitos, WW}     => {NCC}     0.000000e+00
## [18] {Cobs, French, NCC, Tostitos, WW}       => {Doritos} 0.000000e+00
## [19] {Doritos, French, NCC, Tostitos, WW}    => {Cobs}    0.000000e+00
```

```
## [20] {Cobs, Doritos, French, Tostitos, WW}    => {NCC}      0.000000e+00
## [21] {Cobs, French, NCC, Tostitos, WW}        => {Smiths}   0.000000e+00
## [22] {Cobs, French, Smiths, Tostitos, WW}     => {NCC}      0.000000e+00
## [23] {Cobs, French, NCC, Smiths, WW}          => {Tostitos} 0.000000e+00
## [24] {French, NCC, Thins, Tostitos, WW}       => {Doritos}  0.000000e+00
## [25] {Doritos, French, NCC, Tostitos, WW}     => {Thins}    0.000000e+00
## [26] {Doritos, French, NCC, Thins, Tostitos}  => {WW}       0.000000e+00
## [27] {Doritos, French, Thins, Tostitos, WW}   => {NCC}      0.000000e+00
## [28] {French, NCC, Thins, Tostitos, WW}       => {Smiths}   0.000000e+00
## [29] {French, NCC, Smiths, Thins, Tostitos}   => {WW}       0.000000e+00
## [30] {Doritos, French, NCC, Tostitos, WW}     => {Smiths}   0.000000e+00
## [31] {Doritos, French, Smiths, Tostitos, WW}  => {NCC}      0.000000e+00
## [32] {Cobs, French, NCC, Thins, Tostitos}     => {Doritos}  0.000000e+00
## [33] {Doritos, French, NCC, Thins, Tostitos}  => {Cobs}     0.000000e+00
## [34] {Cobs, Doritos, French, Thins, Tostitos} => {NCC}      0.000000e+00
## [35] {Cobs, French, NCC, Thins, Tostitos}     => {Smiths}   0.000000e+00
## [36] {French, NCC, Smiths, Thins, Tostitos}   => {Cobs}     0.000000e+00
## [37] {Cobs, French, Smiths, Thins, Tostitos}  => {NCC}      0.000000e+00
## [38] {Cobs, Doritos, French, NCC, Tostitos}   => {Smiths}   3.108873e-05
## [39] {Cobs, French, NCC, Smiths, Tostitos}    => {Doritos}  3.108873e-05
## [40] {Doritos, French, NCC, Smiths, Tostitos} => {Cobs}     3.108873e-05
## [41] {Cobs, Doritos, French, NCC, Smiths}     => {Tostitos} 3.108873e-05
## [42] {Doritos, French, NCC, Thins, Tostitos}  => {Smiths}   0.000000e+00
## [43] {French, NCC, Smiths, Thins, Tostitos}   => {Doritos}  0.000000e+00
## [44] {Cobs, French, Thins, Tostitos, WW}      => {Doritos}  0.000000e+00
## [45] {Cobs, Doritos, French, Tostitos, WW}    => {Thins}    0.000000e+00
## [46] {Doritos, French, Thins, Tostitos, WW}   => {Cobs}     0.000000e+00
## [47] {Cobs, Doritos, French, Thins, Tostitos} => {WW}       0.000000e+00
## [48] {Cobs, French, Thins, Tostitos, WW}      => {Smiths}   0.000000e+00
## [49] {Cobs, French, Smiths, Tostitos, WW}     => {Thins}    0.000000e+00
## [50] {Cobs, French, Smiths, Thins, Tostitos}  => {WW}       0.000000e+00
## [51] {Cobs, Doritos, French, Tostitos, WW}    => {Smiths}   0.000000e+00
## [52] {Cobs, French, Smiths, Tostitos, WW}     => {Doritos}  0.000000e+00
## [53] {Doritos, French, Smiths, Tostitos, WW}  => {Cobs}     0.000000e+00
## [54] {Doritos, French, Thins, Tostitos, WW}   => {Smiths}   0.000000e+00
## [55] {Doritos, French, Smiths, Tostitos, WW}  => {Thins}    0.000000e+00
## [56] {Cobs, Doritos, French, Thins, Tostitos} => {Smiths}   0.000000e+00
## [57] {Cobs, French, Smiths, Thins, Tostitos}  => {Doritos}  0.000000e+00
## [58] {Cobs, French, NCC, Smiths, WW}          => {Thins}    0.000000e+00
## [59] {Cobs, French, NCC, Smiths, WW}          => {Doritos}  0.000000e+00
## [60] {Doritos, French, NCC, Thins, WW}        => {Smiths}   6.217745e-05
## [61] {Doritos, NCC, Thins, Tostitos, WW}      => {Smiths}   3.108873e-05
## [62] {Cobs, NCC, Smiths, Thins, Tostitos}     => {Doritos}  3.108873e-05
##      confidence coverage     lift     count
## [1]  1          0.000000e+00  7.489173 0
## [2]  1          0.000000e+00  7.489173 0
## [3]  1          3.108873e-05  3.058477 1
## [4]  1          3.108873e-05  2.956434 1
## [5]  1          0.000000e+00  6.189340 0
## [6]  1          0.000000e+00  4.771696 0
## [7]  1          0.000000e+00  3.354119 0
## [8]  1          3.108873e-05  2.956434 1
## [9]  1          0.000000e+00  2.956434 0
## [10] 1          0.000000e+00  3.058477 0
```

```
## [11] 1          0.000000e+00  2.956434 0
## [12] 1          6.217745e-05  2.956434 2
## [13] 1          6.217745e-05  3.058477 2
## [14] 1          0.000000e+00  3.354119 0
## [15] 1          0.000000e+00  4.771696 0
## [16] 1          0.000000e+00  6.189340 0
## [17] 1          0.000000e+00  7.489173 0
## [18] 1          0.000000e+00  3.058477 0
## [19] 1          0.000000e+00  4.771696 0
## [20] 1          0.000000e+00  7.489173 0
## [21] 1          0.000000e+00  2.956434 0
## [22] 1          0.000000e+00  7.489173 0
## [23] 1          0.000000e+00 13.870634 0
## [24] 1          0.000000e+00  3.058477 0
## [25] 1          0.000000e+00  3.354119 0
## [26] 1          0.000000e+00  6.189340 0
## [27] 1          0.000000e+00  7.489173 0
## [28] 1          0.000000e+00  2.956434 0
## [29] 1          0.000000e+00  6.189340 0
## [30] 1          0.000000e+00  2.956434 0
## [31] 1          0.000000e+00  7.489173 0
## [32] 1          0.000000e+00  3.058477 0
## [33] 1          0.000000e+00  4.771696 0
## [34] 1          0.000000e+00  7.489173 0
## [35] 1          0.000000e+00  2.956434 0
## [36] 1          0.000000e+00  4.771696 0
## [37] 1          0.000000e+00  7.489173 0
## [38] 1          3.108873e-05  2.956434 1
## [39] 1          3.108873e-05  3.058477 1
## [40] 1          3.108873e-05  4.771696 1
## [41] 1          3.108873e-05 13.870634 1
## [42] 1          0.000000e+00  2.956434 0
## [43] 1          0.000000e+00  3.058477 0
## [44] 1          0.000000e+00  3.058477 0
## [45] 1          0.000000e+00  3.354119 0
## [46] 1          0.000000e+00  4.771696 0
## [47] 1          0.000000e+00  6.189340 0
## [48] 1          0.000000e+00  2.956434 0
## [49] 1          0.000000e+00  3.354119 0
## [50] 1          0.000000e+00  6.189340 0
## [51] 1          0.000000e+00  2.956434 0
## [52] 1          0.000000e+00  3.058477 0
## [53] 1          0.000000e+00  4.771696 0
## [54] 1          0.000000e+00  2.956434 0
## [55] 1          0.000000e+00  3.354119 0
## [56] 1          0.000000e+00  2.956434 0
## [57] 1          0.000000e+00  3.058477 0
## [58] 1          0.000000e+00  3.354119 0
## [59] 1          0.000000e+00  3.058477 0
## [60] 1          6.217745e-05  2.956434 2
## [61] 1          3.108873e-05  2.956434 1
## [62] 1          3.108873e-05  3.058477 1
```

```r
other_rules_df <- other_rules_df %>% arrange(desc(support))
```

We can see that:

NCC and WW are the most common brands bought by young and midage singles/couples.

Cobbs, French, and Tostitos seem to be the most popular for other customer segments.

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.

# do t test if average packsize is higher for mainstream young singles/couples as opposed
↪  to others

# get mainstream young singles/couples
ms_young <- data[LIFESTAGE == "YOUNG SINGLES/COUPLES" & PREMIUM_CUSTOMER ==
↪  "Mainstream",]
# get others
not_ms_young <- data[xor(LIFESTAGE != "YOUNG SINGLES/COUPLES", PREMIUM_CUSTOMER !=
↪  "Mainstream"),]


# apriori for ms_young
ms_young_pack <- factor(ms_young$PACK_SIZE)
ms_young_id <- factor(ms_young$LYLTY_CARD_NBR)

ms_young_trans <- split(ms_young_pack,ms_young_id)

ms_young_rules <- apriori(ms_young_trans, parameter = list(supp = 0.001, conf = 0.8))
```

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

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.8    0.1    1 none FALSE            TRUE       5   0.001      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: 7
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[20 item(s), 7917 transaction(s)] done [0.00s].
## sorting and recoding items ... [20 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 5 6 done [0.00s].
```

```
## writing ... [32 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

```r
ms_young_rules_df <- inspect(ms_young_rules) %>% select(-2) %>% arrange(desc(support))
```

```
## lhs rhs support confidence coverage lift
## [1] {220, 380} => {175} 0.001136794 0.9000000 0.001263105 1.963975
## [2] {90, 270} => {175} 0.001263105 0.8333333 0.001515726 1.818495
## [3] {90, 330} => {175} 0.002147278 0.8095238 0.002652520 1.766538
## [4] {90, 110} => {175} 0.002399899 0.8636364 0.002778830 1.884622
## [5] {160, 190} => {175} 0.001136794 0.9000000 0.001263105 1.963975
## [6] {190, 270} => {175} 0.001010484 0.8000000 0.001263105 1.745755
## [7] {190, 380} => {175} 0.001642036 0.8125000 0.002020968 1.773033
## [8] {135, 200} => {175} 0.001263105 0.8333333 0.001515726 1.818495
## [9] {90, 134, 330} => {175} 0.001010484 0.8888889 0.001136794 1.939728
## [10] {90, 165, 170} => {175} 0.001136794 1.0000000 0.001136794 2.182194
## [11] {90, 110, 165} => {175} 0.001010484 1.0000000 0.001010484 2.182194
## [12] {90, 110, 170} => {175} 0.001136794 1.0000000 0.001136794 2.182194
## [13] {90, 134, 170} => {175} 0.001010484 0.8888889 0.001136794 1.939728
## [14] {90, 110, 134} => {175} 0.001515726 0.9230769 0.001642036 2.014333
## [15] {160, 165, 170} => {175} 0.001389415 0.8461538 0.001642036 1.846472
## [16] {110, 160, 170} => {175} 0.001136794 0.8181818 0.001389415 1.785431
## [17] {170, 190, 380} => {175} 0.001010484 0.8888889 0.001136794 1.939728
## [18] {134, 165, 190} => {170} 0.001136794 0.8181818 0.001389415 4.626818
## [19] {165, 200, 330} => {150} 0.001010484 0.8888889 0.001136794 2.833065
## [20] {170, 200, 330} => {150} 0.001136794 0.8181818 0.001389415 2.607708
## [21] {110, 135, 170} => {175} 0.001136794 0.9000000 0.001263105 1.963975
## [22] {210, 270, 330} => {175} 0.001010484 1.0000000 0.001010484 2.182194
## [23] {134, 210, 270} => {175} 0.001389415 0.8461538 0.001642036 1.846472
## [24] {210, 330, 380} => {175} 0.001515726 0.8571429 0.001768347 1.870452
## [25] {150, 210, 380} => {175} 0.001768347 0.8750000 0.002020968 1.909420
## [26] {170, 330, 380} => {175} 0.001642036 0.8666667 0.001894657 1.891235
## [27] {150, 210, 330, 380} => {175} 0.001010484 1.0000000 0.001010484
2.182194
## [28] {110, 150, 210, 330} => {175} 0.001136794 0.9000000 0.001263105
1.963975
## [29] {110, 150, 170, 210} => {175} 0.001263105 1.0000000 0.001263105
2.182194
## [30] {134, 150, 170, 210} => {175} 0.001010484 0.8000000 0.001263105
1.745755
## [31] {110, 165, 170, 330} => {175} 0.001136794 0.9000000 0.001263105
1.963975
## [32] {110, 134, 170, 330} => {175} 0.001515726 0.8000000 0.001894657
1.745755
## count
## [1] 9
## [2] 10
## [3] 17
## [4] 19
## [5] 9
## [6] 8
## [7] 13
## [8] 10
```

```
## [9] 8
## [10] 9
## [11] 8
## [12] 9
## [13] 8
## [14] 12
## [15] 11
## [16] 9
## [17] 8
## [18] 9
## [19] 8
## [20] 9
## [21] 9
## [22] 8
## [23] 11
## [24] 12
## [25] 14
## [26] 13
## [27] 8
## [28] 9
## [29] 10
## [30] 8
## [31] 9
## [32] 12
```

```r
# apriori for not_ms_young
not_ms_young_pack <- factor(not_ms_young$PACK_SIZE)
not_ms_young_id <- factor(not_ms_young$LYLTY_CARD_NBR)

not_ms_young_trans <- split(not_ms_young_pack, not_ms_young_id)

not_ms_young_rules <- apriori(not_ms_young_trans, parameter = list(supp = 0.001,conf =
↪  0.8))
```

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

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.8    0.1    1 none FALSE            TRUE       5   0.001      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: 26
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[20 item(s), 26944 transaction(s)] done [0.00s].
## sorting and recoding items ... [20 item(s)] done [0.00s].
```

```
## creating transaction tree ... done [0.01s].
## checking subsets of size 1 2 3 4 5 6 done [0.00s].
## writing ... [82 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

```
not_ms_young_rules_df <- inspect(not_ms_young_rules) %>% select(-2) %>%
↪  arrange(desc(support))
```

```
##      lhs                         rhs   support     confidence coverage
## [1]  {180, 270}               => {175} 0.001224762 0.8048780  0.001521675
## [2]  {70, 330}                => {175} 0.002486639 0.8589744  0.002894893
## [3]  {125, 270}               => {175} 0.001373219 0.8222222  0.001670131
## [4]  {220, 270}               => {175} 0.001558789 0.8400000  0.001855701
## [5]  {190, 250}               => {175} 0.001336105 0.8780488  0.001521675
## [6]  {90, 270}                => {175} 0.002969121 0.8080808  0.003674287
## [7]  {70, 150, 330}           => {175} 0.001484561 0.8333333  0.001781473
## [8]  {70, 165, 170}           => {175} 0.001633017 0.8301887  0.001967043
## [9]  {125, 150, 270}          => {175} 0.001002078 0.9310345  0.001076306
## [10] {125, 150, 170}          => {175} 0.003006235 0.8100000  0.003711401
## [11] {150, 210, 220}          => {175} 0.001039192 0.9333333  0.001113420
## [12] {150, 170, 220}          => {175} 0.003414489 0.8141593  0.004193884
## [13] {90, 150, 250}           => {175} 0.001002078 0.8437500  0.001187648
## [14] {160, 165, 190}          => {175} 0.001113420 0.8333333  0.001336105
## [15] {150, 160, 190}          => {175} 0.001633017 0.8979592  0.001818587
## [16] {160, 165, 200}          => {175} 0.001261876 0.8095238  0.001558789
## [17] {150, 160, 200}          => {175} 0.002115499 0.8028169  0.002635095
## [18] {160, 165, 330}          => {175} 0.001521675 0.8200000  0.001855701
## [19] {160, 170, 330}          => {175} 0.001892815 0.8500000  0.002226841
## [20] {110, 160, 330}          => {175} 0.001224762 0.8048780  0.001521675
## [21] {134, 160, 330}          => {175} 0.001595903 0.8431373  0.001892815
## [22] {150, 160, 330}          => {175} 0.002894893 0.8210526  0.003525831
## [23] {150, 160, 165}          => {175} 0.004861936 0.8238994  0.005901128
## [24] {90, 170, 270}           => {175} 0.001150534 0.8857143  0.001298990
## [25] {90, 110, 270}           => {175} 0.001002078 0.8181818  0.001224762
## [26] {90, 134, 270}           => {175} 0.001113420 0.9375000  0.001187648
## [27] {90, 150, 270}           => {175} 0.002115499 0.8260870  0.002560867
## [28] {90, 170, 330}           => {175} 0.001595903 0.8775510  0.001818587
## [29] {90, 110, 330}           => {175} 0.001373219 0.8409091  0.001633017
## [30] {90, 134, 330}           => {175} 0.001558789 0.8235294  0.001892815
## [31] {90, 134, 165}           => {175} 0.002523753 0.8192771  0.003080463
## [32] {90, 150, 170}           => {175} 0.006160926 0.8058252  0.007645487
## [33] {170, 190, 200}          => {175} 0.001447447 0.8125000  0.001781473
## [34] {170, 190, 330}          => {175} 0.001967043 0.8153846  0.002412411
## [35] {110, 190, 330}          => {175} 0.001670131 0.8333333  0.002004157
## [36] {165, 170, 190}          => {175} 0.003154691 0.8095238  0.003896971
## [37] {110, 165, 190}          => {175} 0.002820665 0.8172043  0.003451603
## [38] {134, 165, 190}          => {175} 0.002746437 0.8131868  0.003377375
## [39] {110, 170, 190}          => {175} 0.002969121 0.8163265  0.003637173
## [40] {134, 170, 190}          => {175} 0.003191805 0.8190476  0.003896971
## [41] {110, 134, 190}          => {175} 0.002301069 0.8266667  0.002783551
## [42] {170, 200, 270}          => {175} 0.001224762 0.8684211  0.001410333
## [43] {110, 200, 270}          => {175} 0.001002078 0.8709677  0.001150534
## [44] {110, 200, 380}          => {175} 0.001076306 0.8529412  0.001261876
```

```
## [45] {170, 200, 330}          => {175} 0.002486639 0.8271605  0.003006235
## [46] {150, 200, 330}          => {175} 0.003859857 0.8000000  0.004824822
## [47] {165, 170, 200}          => {175} 0.004824822 0.8125000  0.005938242
## [48] {134, 170, 200}          => {175} 0.004750594 0.8000000  0.005938242
## [49] {150, 170, 200}          => {175} 0.008907363 0.8026756  0.011097090
## [50] {70, 150, 165, 170}      => {175} 0.001002078 0.8181818  0.001224762
## [51] {125, 150, 165, 170}     => {175} 0.001039192 0.8235294  0.001261876
## [52] {125, 134, 150, 170}     => {175} 0.001002078 0.8181818  0.001224762
## [53] {110, 150, 250, 330}     => {175} 0.001261876 0.8095238  0.001558789
## [54] {150, 160, 165, 330}     => {175} 0.001002078 0.8437500  0.001187648
## [55] {150, 160, 170, 330}     => {175} 0.001150534 0.8611111  0.001336105
## [56] {150, 160, 165, 170}     => {175} 0.001744359 0.8245614  0.002115499
## [57] {110, 150, 160, 165}     => {175} 0.001707245 0.8518519  0.002004157
## [58] {134, 150, 160, 165}     => {175} 0.001484561 0.8695652  0.001707245
## [59] {90, 134, 150, 165}      => {175} 0.001670131 0.8181818  0.002041271
## [60] {90, 110, 134, 170}      => {175} 0.001039192 0.8235294  0.001261876
## [61] {90, 134, 150, 170}      => {175} 0.002412411 0.8441558  0.002857779
## [62] {90, 110, 134, 150}      => {175} 0.001558789 0.8936170  0.001744359
## [63] {150, 170, 190, 330}     => {175} 0.001336105 0.8571429  0.001558789
## [64] {110, 150, 190, 330}     => {175} 0.001113420 0.8823529  0.001261876
## [65] {110, 165, 170, 190}     => {175} 0.001298990 0.8750000  0.001484561
## [66] {150, 165, 170, 190}     => {175} 0.001929929 0.8524590  0.002263955
## [67] {134, 150, 165, 190}     => {175} 0.001558789 0.8400000  0.001855701
## [68] {110, 150, 170, 190}     => {175} 0.001781473 0.8135593  0.002189727
## [69] {134, 150, 170, 190}     => {175} 0.001781473 0.8275862  0.002152613
## [70] {110, 134, 150, 190}     => {175} 0.001410333 0.8444444  0.001670131
## [71] {134, 170, 200, 330}     => {175} 0.001076306 0.8529412  0.001261876
## [72] {150, 170, 200, 330}     => {175} 0.001410333 0.8837209  0.001595903
## [73] {134, 150, 200, 330}     => {175} 0.001521675 0.8367347  0.001818587
## [74] {110, 165, 170, 200}     => {175} 0.001633017 0.8461538  0.001929929
## [75] {134, 165, 170, 200}     => {175} 0.001744359 0.8103448  0.002152613
## [76] {150, 165, 170, 200}     => {175} 0.003191805 0.8600000  0.003711401
## [77] {110, 134, 170, 200}     => {175} 0.001484561 0.8163265  0.001818587
## [78] {110, 150, 210, 270}     => {175} 0.001002078 0.8181818  0.001224762
## [79] {110, 165, 330, 380}     => {175} 0.001039192 0.8750000  0.001187648
## [80] {110, 150, 165, 380}     => {175} 0.002486639 0.8375000  0.002969121
## [81] {110, 150, 165, 170, 200} => {175} 0.001002078 0.8709677  0.001150534
## [82] {110, 134, 150, 165, 380} => {175} 0.001113420 0.8333333  0.001336105
##      lift     count
## [1]  1.431839  33
## [2]  1.528074  67
## [3]  1.462693  37
## [4]  1.494319  42
## [5]  1.562006  36
## [6]  1.437537  80
## [7]  1.482460  40
## [8]  1.476865  44
## [9]  1.656265  27
## [10] 1.440951  81
## [11] 1.660355  28
## [12] 1.448350  92
## [13] 1.500990  27
## [14] 1.482460  30
## [15] 1.597426  44
```

```
## [16] 1.440104  34
## [17] 1.428172  57
## [18] 1.458740  41
## [19] 1.512109  51
## [20] 1.431839  33
## [21] 1.499900  43
## [22] 1.460613  78
## [23] 1.465677 131
## [24] 1.575643  31
## [25] 1.455506  27
## [26] 1.667767  30
## [27] 1.469569  57
## [28] 1.561121  43
## [29] 1.495937  37
## [30] 1.465019  42
## [31] 1.457454  68
## [32] 1.433524 166
## [33] 1.445398  39
## [34] 1.450530  53
## [35] 1.482460  45
## [36] 1.440104  85
## [37] 1.453767  76
## [38] 1.446620  74
## [39] 1.452205  80
## [40] 1.457046  86
## [41] 1.470600  62
## [42] 1.544879  33
## [43] 1.549409  27
## [44] 1.517341  29
## [45] 1.471478  67
## [46] 1.423161 104
## [47] 1.445398 130
## [48] 1.423161 128
## [49] 1.427921 240
## [50] 1.455506  27
## [51] 1.465019  28
## [52] 1.455506  27
## [53] 1.440104  34
## [54] 1.500990  27
## [55] 1.531875  31
## [56] 1.466855  47
## [57] 1.515403  46
## [58] 1.546914  40
## [59] 1.455506  45
## [60] 1.465019  28
## [61] 1.501712  65
## [62] 1.589701  42
## [63] 1.524816  36
## [64] 1.569663  30
## [65] 1.556583  35
## [66] 1.516483  52
## [67] 1.494319  42
## [68] 1.447283  48
## [69] 1.472236  48
```

```
## [70] 1.502226  38
## [71] 1.517341  29
## [72] 1.572097  38
## [73] 1.488510  41
## [74] 1.505267  44
## [75] 1.441564  47
## [76] 1.529898  86
## [77] 1.452205  40
## [78] 1.455506  27
## [79] 1.556583  28
## [80] 1.489872  67
## [81] 1.549409  27
## [82] 1.482460  30
```

Mainstream-Young Singles/Couples tend to pick 90, 330, and 380 grams for their pack size.

Other segments tend to pick 150 and 170 grams.

This shows mainstream young singles/couples either pick small chips or very large chips, while other segments tend to pick sizes in the middle.