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)
library(ggplot2)
library(ggmosaic)
library(readr)
library(stringr)
library(tidyverse)
library(arules)

#### 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 <- "D:/Documents/Coding/R/Quantium/"
transactionData <- fread(paste0(filePath,"QVI_transaction_data.csv"))
customerData <- fread(paste0(filePath,"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.

```
#### Examine transaction data
# Over to you! Examine the data using one or more of the methods described above.
head(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
## 6: 43604         4           4074   2982       57
##                                   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
## 6: Old El Paso Salsa   Dip Tomato Mild 300g        1       5.1
```

```
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>
```

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

```
#### 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.

```
#### Examine PROD_NAME
# Over to you! Generate a summary of the PROD_NAME column.
transactionData[, .N, PROD_NAME]
```

```
##                                     PROD_NAME    N
##   1:   Natural Chip        Compny SeaSalt175g 1468
##   2:                 CCs Nacho Cheese    175g 1498
##   3:   Smiths Crinkle Cut  Chips Chicken 170g 1484
##   4:   Smiths Chip Thinly  S/Cream&Onion 175g 1473
##   5: Kettle Tortilla ChpsHny&Jlpno Chili 150g 3296
##  ---
## 110:    Red Rock Deli Chikn&Garlic Aioli 150g 1434
## 111:      RRD SR Slow Rst     Pork Belly 150g 1526
## 112:                 RRD Pc Sea Salt     165g 1431
## 113:       Smith Crinkle Cut   Bolognese 150g 1451
## 114:                 Doritos Salsa Mild  300g 1472
```

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
# Fixing PROD_NAME format
transactionData$PROD_NAME <- gsub("(\\D)(\\d)", "\\1 \\2", transactionData$PROD_NAME)
transactionData$PROD_NAME <- gsub("&", " & ",  transactionData$PROD_NAME)
transactionData$PROD_NAME <- gsub("([a-z])([A-Z])", "\\1 \\2", transactionData$PROD_NAME)
transactionData$PROD_NAME <- gsub("\\s+", " ", transactionData$PROD_NAME)

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.
# Removing digits
productWords <- productWords[grepl("\\d", words) == FALSE, ]

# Removing special characters
productWords <- productWords[grepl("&", words) == FALSE, ]

#### 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
wordrank <- productWords[, .N, words][order(N, decreasing = TRUE)]
```

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
```

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.
transactionData[PROD_QTY == 200, ]
```

```
##          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.

```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.
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[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
```

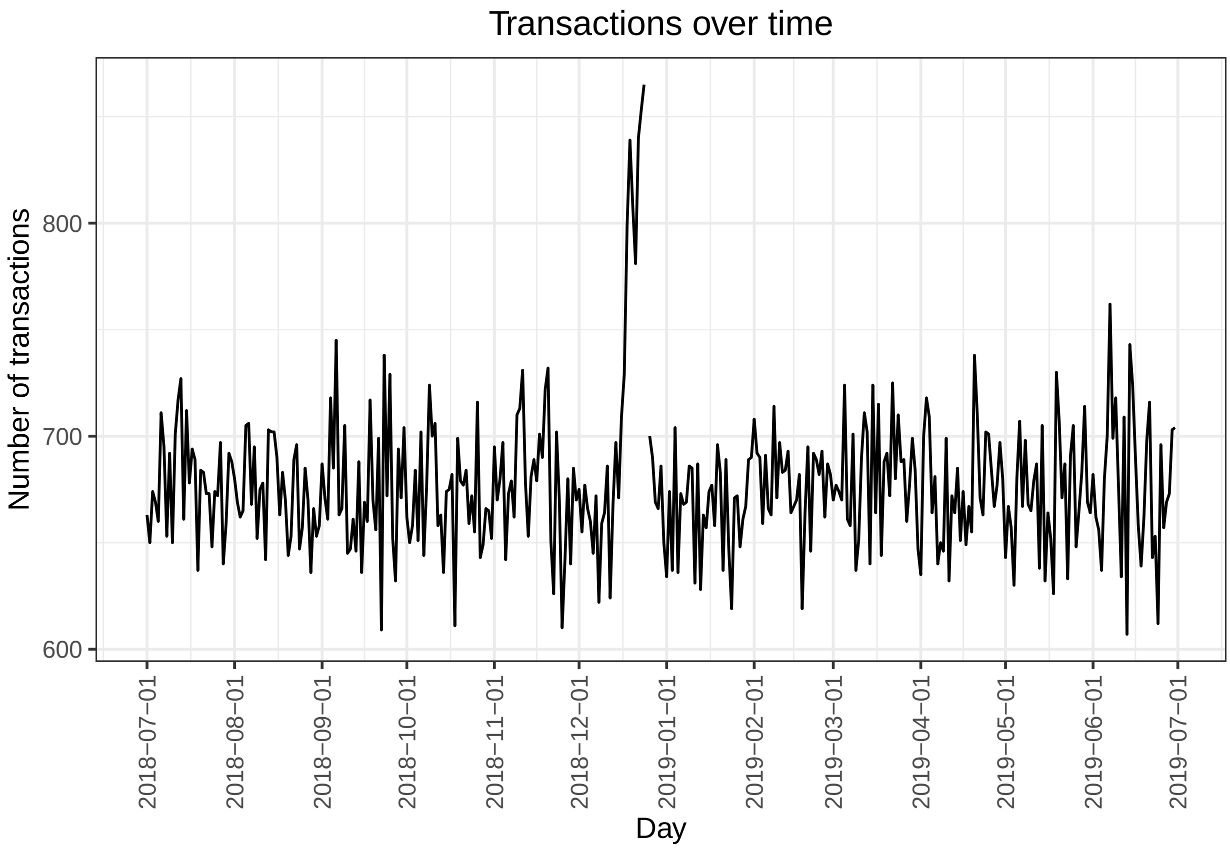
That's better. Now, let's look at the number of transaction lines over time to see if there are any obvious data issues such as missing data.

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

```
##            DATE   N
##   1: 2018-10-17 682
##   2: 2019-05-14 705
##   3: 2019-05-20 707
##   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
```

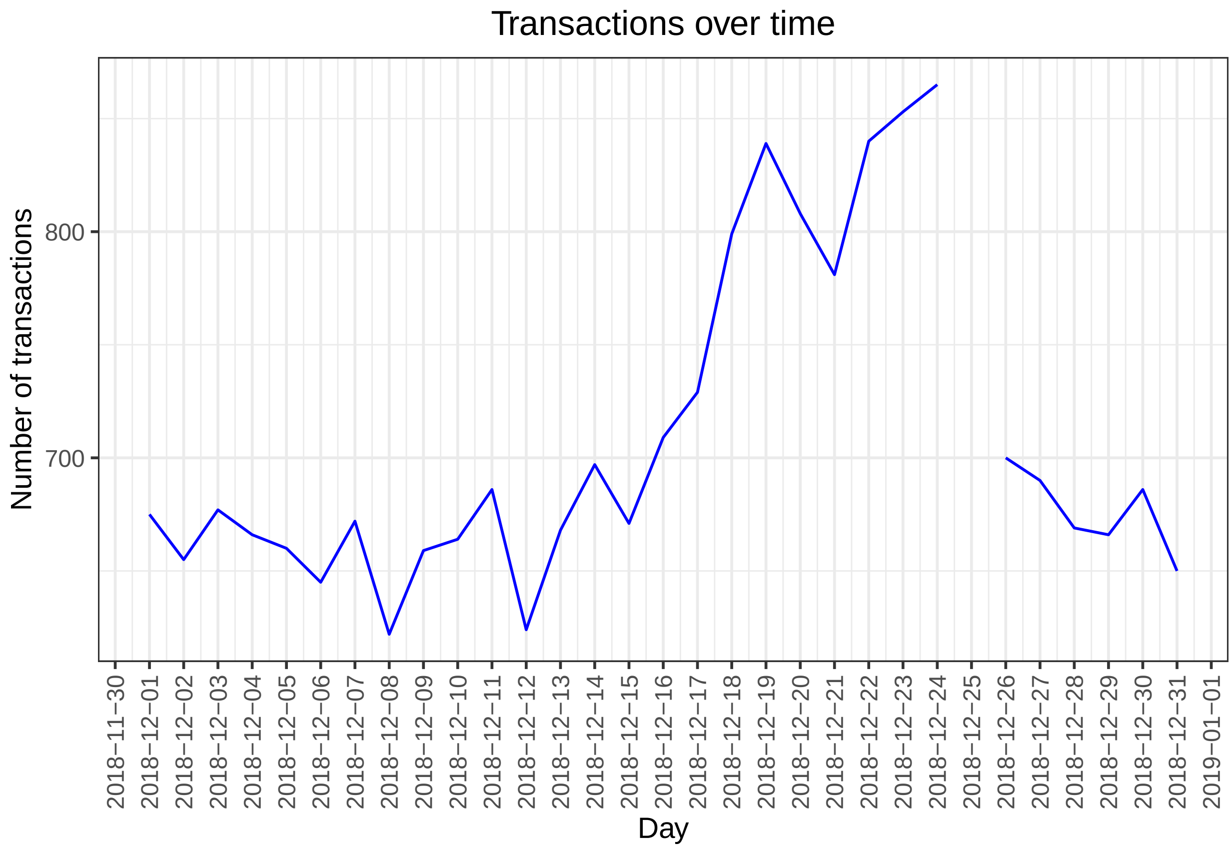
There's only 364 rows, meaning only 364 dates which indicates a missing date. Let's create a sequence of dates from 1 Jul 2018 to 30 Jun 2019 and use this to create a chart of number of transactions over time to find the missing date.

```r
#### Create a sequence of dates and join this the count of transactions by date
# 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 <- data.table(seq(as.Date("2018/07/01"), as.Date("2019/06/30"), by = "day"))
setnames(dates, "DATE")
transactions_by_day <- merge(dates, transactionData[, .N, by = DATE], all.x = TRUE)

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

#### Plot transactions over time
ggplot(transactions_by_day, aes(x = DATE, y = N)) +
 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[month(DATE) == 12], aes(x = DATE, y = N)) +
  geom_line(color = "blue") +
  labs(x = "Day", y = "Number of transactions", title = "Transactions over time") +
  scale_x_date(breaks = "1 day") +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5))
```



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.

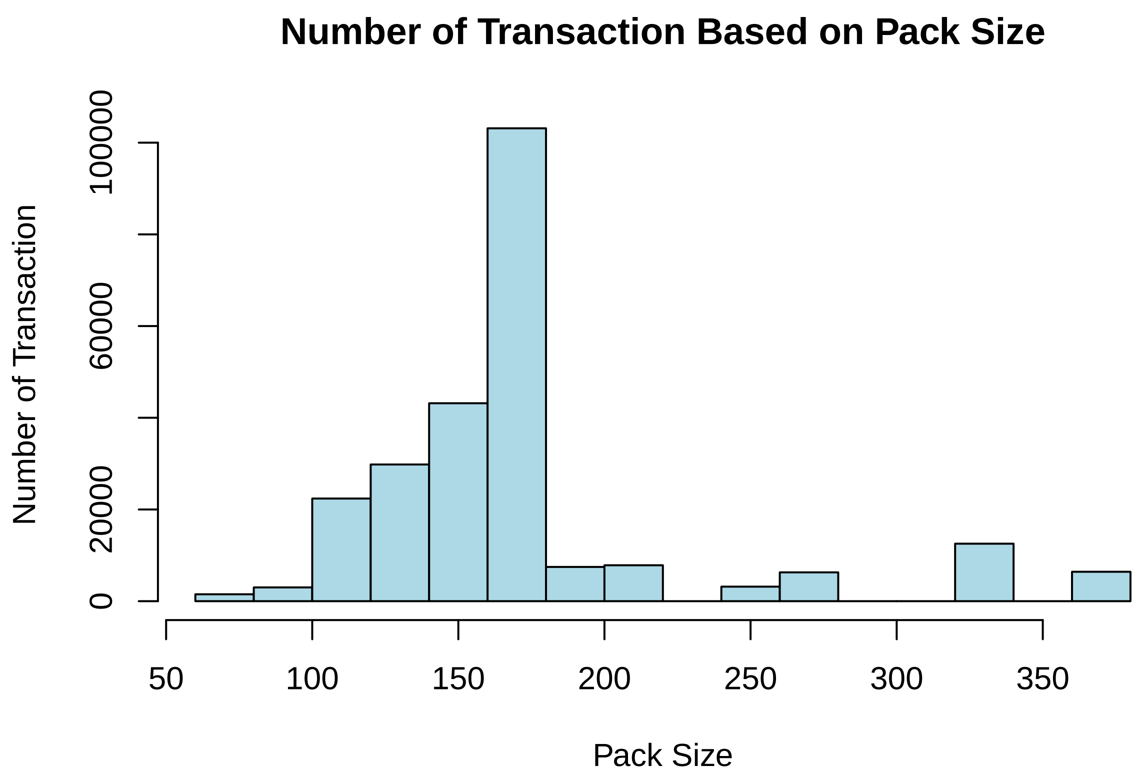
```
#### Pack size
#### We can work this out by taking the digits that are in PROD_NAME
transactionData[, PACK_SIZE := parse_number(PROD_NAME)]

#### Always check your output
#### Let's check if the pack sizes look sensible
transactionData[, .N, PACK_SIZE][order(-N)]
```

```
##     PACK_SIZE     N
##  1:       175 66390
##  2:       150 40203
##  3:       134 25102
##  4:       110 22387
##  5:       170 19983
##  6:       165 15297
##  7:       330 12540
##  8:       380  6416
##  9:       270  6285
## 10:       210  6272
## 11:       200  4473
## 12:       135  3257
## 13:       250  3169
```

```
## 14:         90   3008
## 15:        190   2995
## 16:        160   2970
## 17:        220   1564
## 18:         70   1507
## 19:        180   1468
## 20:        125   1454
```

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.
options(scipen=999)
hist(transactionData$PACK_SIZE, main="Number of Transaction Based on Pack Size", xlab =
↪  "Pack Size", ylab = "Number of Transaction", col = "lightblue")
```



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

```
#### Brands
# Over to you! Create a column which contains the brand of the product, by extracting it
↪  from the product name.
transactionData$BRAND <- toupper(word(transactionData$PROD_NAME, 1, 1))

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

```r
brandrank <- transactionData[, .N, BRAND][order(BRAND)]
brandrank
```

```
##          BRAND     N
##  1:     BURGER  1564
##  2:        CCS  4551
##  3:    CHEETOS  2927
##  4:   CHEEZELS  4603
##  5:       COBS  9693
##  6:     DORITO  3183
##  7:    DORITOS 22041
##  8:     FRENCH  1418
##  9:      GRAIN  6272
## 10:        GRN  1468
## 11:  INFUZIONS 11057
## 12:     INFZNS  3144
## 13:     KETTLE 41288
## 14:    NATURAL  6050
## 15:        NCC  1419
## 16:   PRINGLES 25102
## 17:        RED  4427
## 18:        RRD 11894
## 19:      SMITH  2963
## 20:     SMITHS 27390
## 21:      SNBTS  1576
## 22:   SUNBITES  1432
## 23:      THINS 14075
## 24:   TOSTITOS  9471
## 25:   TWISTIES  9454
## 26:   TYRRELLS  6442
## 27: WOOLWORTHS  1516
## 28:         WW 10320
##          BRAND     N
```

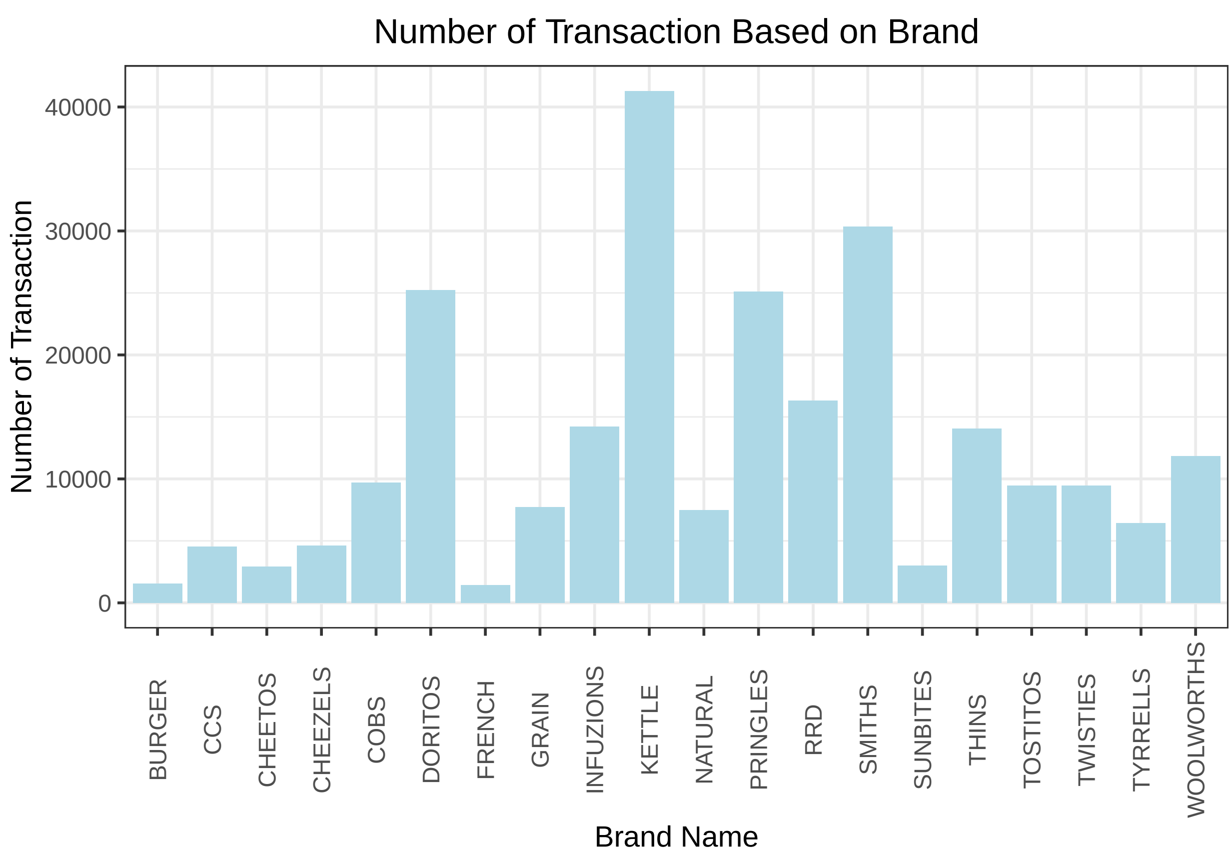
Some of the brand names look like they are of the same brands - such as RED and RRD, which are both Red Rock Deli chips. Let's combine these together.

```r
#### Clean brand names
transactionData[BRAND == "RED", BRAND := "RRD"]
# Over to you! Add any additional brand adjustments you think may be required.
transactionData[BRAND == "DORITO", BRAND := "DORITOS"]
transactionData[BRAND == "GRN", BRAND := "GRAIN"]
transactionData[BRAND == "INFZNS", BRAND := "INFUZIONS"]
transactionData[BRAND == "NCC", BRAND := "NATURAL"]
transactionData[BRAND == "SMITH", BRAND := "SMITHS"]
transactionData[BRAND == "SNBTS", BRAND := "SUNBITES"]
transactionData[BRAND == "WW", BRAND := "WOOLWORTHS"]

#### Check again
# Over to you! Check the results look reasonable.
brandrank <- transactionData[, .N, BRAND][order(-N)]
brandrank
```

```
##          BRAND     N
```

```
##  1:      KETTLE 41288
##  2:      SMITHS 30353
##  3:     DORITOS 25224
##  4:    PRINGLES 25102
##  5:         RRD 16321
##  6:   INFUZIONS 14201
##  7:       THINS 14075
##  8:  WOOLWORTHS 11836
##  9:        COBS  9693
## 10:    TOSTITOS  9471
## 11:    TWISTIES  9454
## 12:       GRAIN  7740
## 13:     NATURAL  7469
## 14:    TYRRELLS  6442
## 15:    CHEEZELS  4603
## 16:         CCS  4551
## 17:    SUNBITES  3008
## 18:     CHEETOS  2927
## 19:      BURGER  1564
## 20:      FRENCH  1418
```

```
ggplot(data = brandrank, aes(x = BRAND, y = N)) +
  geom_bar(stat = "identity", position = "dodge", fill = "lightblue") +
  labs(title = "Number of Transaction Based on Brand", x = "Brand Name", y = "Number of
  ↪   Transaction") +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5))
```



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(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
```

```
head(customerData)
```

```
##    LYLTY_CARD_NBR              LIFESTAGE PREMIUM_CUSTOMER
## 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.
colSums(is.na(data))
```

```
##   LYLTY_CARD_NBR             DATE        STORE_NBR           TXN_ID
##                0                0                0                0
##         PROD_NBR        PROD_NAME         PROD_QTY        TOT_SALES
##                0                0                0                0
##        PACK_SIZE            BRAND        LIFESTAGE PREMIUM_CUSTOMER
##                0                0                0                0
```

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

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

Data exploration is now complete!

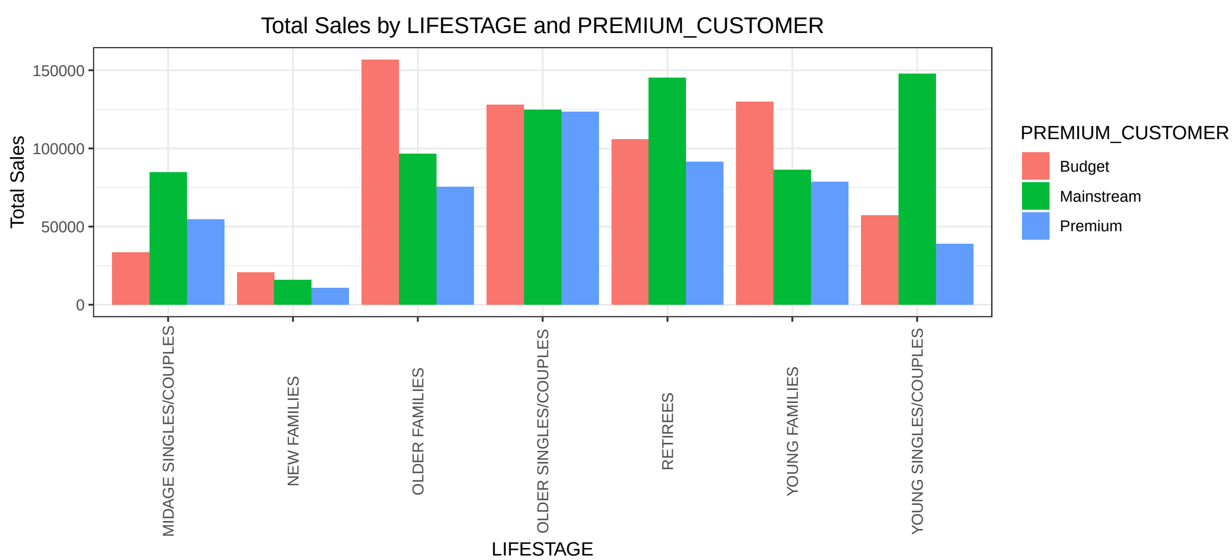
Data analysis on customer segments

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.

```
#### Total sales by LIFESTAGE and PREMIUM_CUSTOMER
# Over to you! Calculate the summary of sales by those dimensions and create a plot.
totalSales <- data %>%
  group_by(LIFESTAGE, PREMIUM_CUSTOMER) %>%
  summarise(TOT_SALES = sum(TOT_SALES))
```

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

```
ggplot(data = totalSales, aes(x = LIFESTAGE, y = TOT_SALES, fill = PREMIUM_CUSTOMER)) +
  geom_bar(stat = "identity", position = "dodge") +
  labs(title = "Total Sales by LIFESTAGE and PREMIUM_CUSTOMER", x = "LIFESTAGE", y =
  ↪  "Total Sales", fill = "PREMIUM_CUSTOMER") +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5))
```

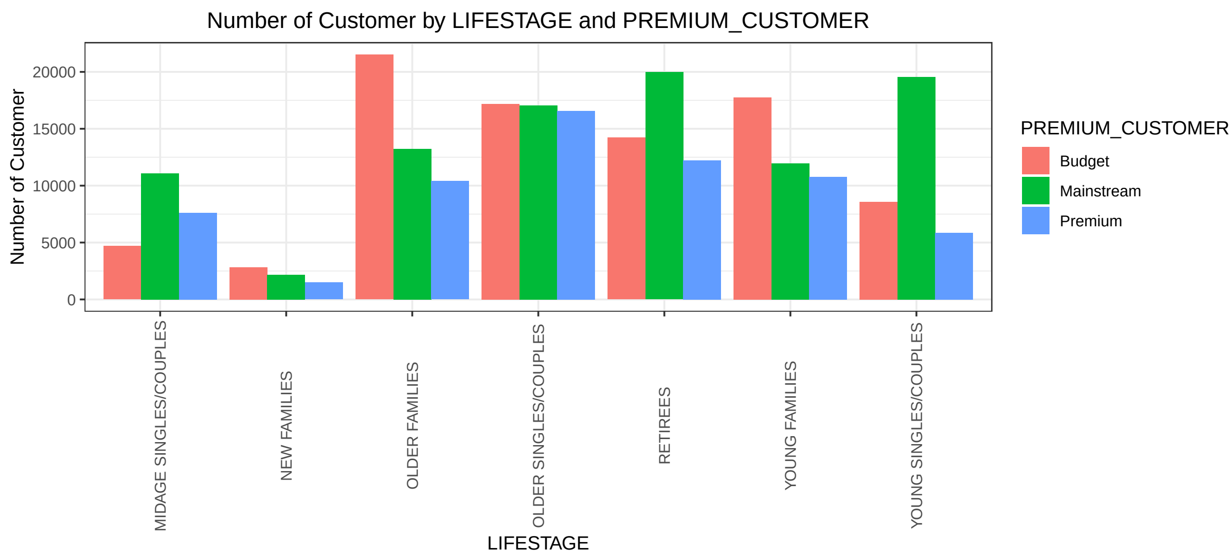


Sales are coming mainly from Budget - older families, Mainstream - young singles/couples, and Mainstream - retirees Let's see if the higher sales are due to there being more customers who buy chips.

```
#### 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.
numCustomers <- data %>%
  group_by(LIFESTAGE, PREMIUM_CUSTOMER) %>%
  summarise(NUM_CUSTOMERS = n())
```

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

```r
ggplot(data = numCustomers, aes(x = LIFESTAGE, y = NUM_CUSTOMERS, fill =
↪  PREMIUM_CUSTOMER)) +
  geom_bar(stat = "identity", position = "dodge") +
  labs(title = "Number of Customer by LIFESTAGE and PREMIUM_CUSTOMER", x = "LIFESTAGE", y
  ↪  = "Number of Customer", fill = "PREMIUM_CUSTOMER") +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5))
```

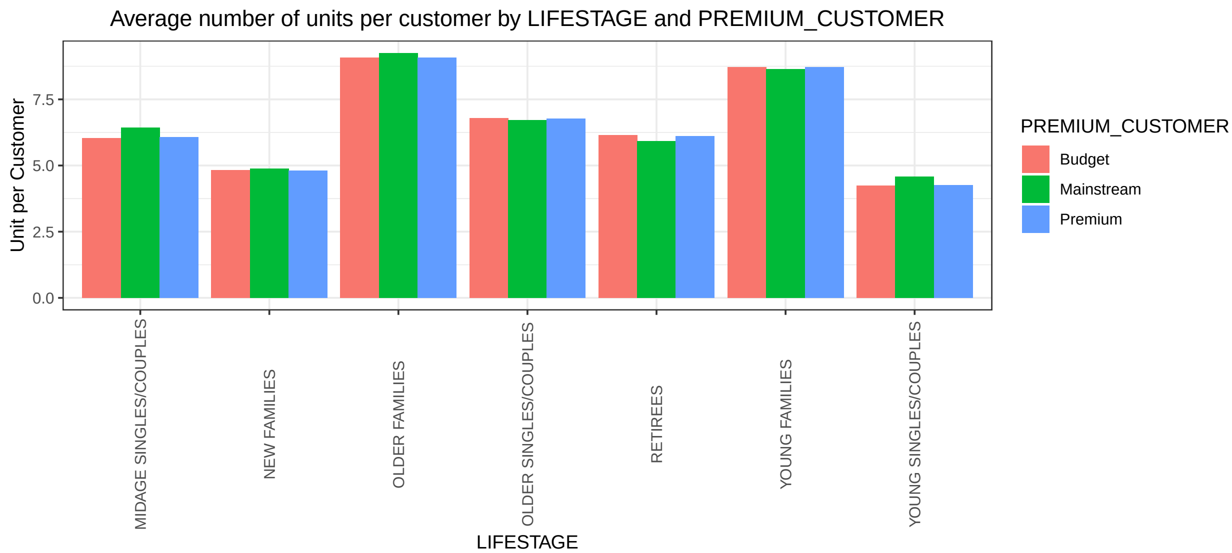


There are more Mainstream - young singles/couples and Mainstream - retirees who buy chips. This contributes to there being more sales to these customer segments but this is not a major driver for the Budget - Older families segment. Higher sales may also be driven by more units of chips being bought 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.
avgUnit <- data %>%
  group_by(LIFESTAGE, PREMIUM_CUSTOMER) %>%
  summarise(AVG_UNITS_PER_CUSTOMER = sum(PROD_QTY)/uniqueN(LYLTY_CARD_NBR))
```

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

```r
ggplot(data = avgUnit, aes(x = LIFESTAGE, y = AVG_UNITS_PER_CUSTOMER, fill =
↪  PREMIUM_CUSTOMER)) +
  geom_bar(stat = "identity", position = "dodge") +
  labs(title = "Average number of units per customer by LIFESTAGE and PREMIUM_CUSTOMER",
  ↪  x = "LIFESTAGE", y = "Unit per Customer", fill = "PREMIUM_CUSTOMER") +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5))
```

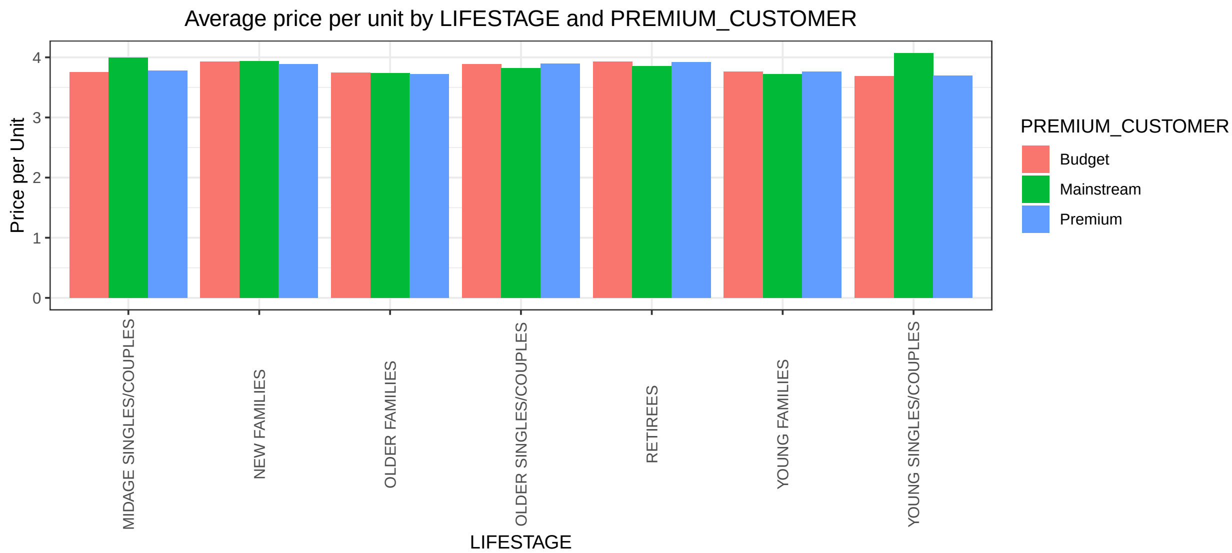


Older families and young families in general buy more chips per customer Let's also investigate the average price per unit chips bought for each customer segment as this is also a driver of total sales.

```
#### 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.
avgPrice <- data %>%
  group_by(LIFESTAGE, PREMIUM_CUSTOMER) %>%
  summarise(AVG_PRICE_PER_UNIT = sum(TOT_SALES)/sum(PROD_QTY))
```

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

```
ggplot(data = avgPrice, aes(x = LIFESTAGE, y = AVG_PRICE_PER_UNIT, fill =
↪  PREMIUM_CUSTOMER)) +
  geom_bar(stat = "identity", position = "dodge") +
  labs(title = "Average price per unit by LIFESTAGE and PREMIUM_CUSTOMER", x =
  ↪  "LIFESTAGE", y = "Price per Unit", fill = "PREMIUM_CUSTOMER") +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5))
```



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.
subsetData <- data[LIFESTAGE %in% c("MIDAGE SINGLES/COUPLES", "YOUNG SINGLES/COUPLES")]
subsetData$PRICE_PER_UNIT = subsetData$TOT_SALES/subsetData$PROD_QTY
mainstreamData <- subsetData[PREMIUM_CUSTOMER == "Mainstream"]
otherData <- subsetData[PREMIUM_CUSTOMER %in% c("Premium", "Budget")]
t.test(mainstreamData$PRICE_PER_UNIT, otherData$PRICE_PER_UNIT)
```

```
##
##  Welch Two Sample t-test
##
## data:  mainstreamData$PRICE_PER_UNIT and otherData$PRICE_PER_UNIT
## t = 37.624, df = 54791, p-value < 0.00000000000000022
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  0.3159319 0.3506572
## sample estimates:
## mean of x mean of y
##  4.039786  3.706491
```

The t-test results in a p-value of 0.00000000000000022, 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)
# Affinity Analysis for Brand
brandData <- data[PREMIUM_CUSTOMER == "Mainstream" & LIFESTAGE == "YOUNG
↪  SINGLES/COUPLES", c("LYLTY_CARD_NBR", "BRAND")]
aggData1 <- aggregate(BRAND ~ LYLTY_CARD_NBR, brandData, c)
trans1 <- as(aggData1$BRAND, "transactions")
```

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

```r
rules1 <- apriori(trans1, parameter = list(supp = 0.01, conf = 0.2, target = "rules"))
```

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.2    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: 79
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[20 item(s), 7917 transaction(s)] done [0.00s].
## sorting and recoding items ... [18 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 done [0.00s].
## writing ... [98 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

```r
rules_sorted1 <- sort(rules1, by = "confidence", decreasing = TRUE)
inspect(rules_sorted1[1:20])
```

```
##      lhs                        rhs      support    confidence coverage
## [1]  {SMITHS, WOOLWORTHS}    => {KETTLE} 0.01023115 0.4764706  0.02147278
## [2]  {KETTLE, WOOLWORTHS}    => {SMITHS} 0.01023115 0.4655172  0.02197802
## [3]  {PRINGLES, THINS}       => {KETTLE} 0.01705191 0.4560811  0.03738790
## [4]  {SMITHS, THINS}         => {KETTLE} 0.01149425 0.4550000  0.02526209
## [5]  {INFUZIONS, SMITHS}     => {KETTLE} 0.01288367 0.4513274  0.02854617
## [6]  {RRD, SMITHS}           => {KETTLE} 0.01503095 0.4507576  0.03334596
## [7]  {PRINGLES, SMITHS}      => {KETTLE} 0.02122016 0.4504021  0.04711381
## [8]  {KETTLE, RRD}           => {SMITHS} 0.01503095 0.4407407  0.03410383
## [9]  {PRINGLES, TOSTITOS}    => {KETTLE} 0.01162056 0.4259259  0.02728306
## [10] {COBS, PRINGLES}        => {KETTLE} 0.01023115 0.4218750  0.02425161
## [11] {DORITOS, RRD}          => {SMITHS} 0.01073639 0.4207921  0.02551472
## [12] {INFUZIONS, PRINGLES} => {KETTLE} 0.01717822 0.4184615  0.04105090
## [13] {WOOLWORTHS}            => {KETTLE} 0.02197802 0.4084507  0.05380826
## [14] {DORITOS, PRINGLES}     => {KETTLE} 0.02551472 0.4072581  0.06264999
```

```
## [15] {DORITOS, TOSTITOS}   => {KETTLE} 0.01035746 0.4019608  0.02576734
## [16] {DORITOS, TWISTIES}   => {KETTLE} 0.01061008 0.4019139  0.02639889
## [17] {WOOLWORTHS}          => {SMITHS} 0.02147278 0.3990610  0.05380826
## [18] {DORITOS, SMITHS}     => {KETTLE} 0.02033599 0.3975309  0.05115574
## [19] {CHEEZELS}            => {KETTLE} 0.01679929 0.3958333  0.04244032
## [20] {NATURAL}             => {KETTLE} 0.01818871 0.3913043  0.04648225
##      lift     count
## [1]  1.230740  81
## [2]  2.299127  81
## [3]  1.178073 135
## [4]  1.175281  91
## [5]  1.165794 102
## [6]  1.164322 119
## [7]  1.163404 168
## [8]  2.176759 119
## [9]  1.100181  92
## [10] 1.089718  81
## [11] 2.078235  85
## [12] 1.080900 136
## [13] 1.055042 174
## [14] 1.051962 202
## [15] 1.038278  82
## [16] 1.038157  84
## [17] 1.970908 170
## [18] 1.026836 161
## [19] 1.022451 133
## [20] 1.010753 144
```

We can see that Mainstream - young singles/couples tends to buy KETTLE and SMITHS.

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.
# Affinity Analysis for Pack Size
packData <- data[PREMIUM_CUSTOMER == "Mainstream" & LIFESTAGE == "YOUNG SINGLES/COUPLES",
↪  c("LYLTY_CARD_NBR", "PACK_SIZE")]
aggData2 <- aggregate(PACK_SIZE ~ LYLTY_CARD_NBR, packData, c)
trans2 <- as(aggData2$PACK_SIZE, "transactions")
```

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

```r
rules2 <- apriori(trans2, parameter = list(supp = 0.01, conf = 0.2, target = "rules"))
```

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.2    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: 79
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[20 item(s), 7917 transaction(s)] done [0.00s].
## sorting and recoding items ... [16 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 done [0.00s].
## writing ... [115 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

```
rules_sorted2 <- sort(rules2, by = "confidence", decreasing = TRUE)
inspect(rules_sorted2[1:20])
```

```
##      lhs                rhs   support    confidence coverage   lift     count
## [1]  {165, 330}      => {175} 0.01035746 0.6259542  0.01654667 1.365954  82
## [2]  {200}           => {175} 0.01338891 0.6057143  0.02210433 1.321786 106
## [3]  {110, 134, 150} => {175} 0.01187318 0.5987261  0.01983074 1.306537  94
## [4]  {110, 170}      => {175} 0.02362006 0.5862069  0.04029304 1.279217 187
## [5]  {165, 170}      => {175} 0.01654667 0.5822222  0.02841986 1.270522 131
## [6]  {110, 165}      => {175} 0.01654667 0.5646552  0.02930403 1.232187 131
## [7]  {150, 170}      => {175} 0.03069344 0.5612009  0.05469243 1.224649 243
## [8]  {134, 170}      => {175} 0.02336744 0.5589124  0.04180877 1.219655 185
## [9]  {150, 165}      => {175} 0.02425161 0.5565217  0.04357711 1.214438 192
## [10] {170, 330}      => {175} 0.01338891 0.5549738  0.02412530 1.211061 106
## [11] {134, 150}      => {175} 0.04029304 0.5370370  0.07502842 1.171919 319
## [12] {134, 330}      => {175} 0.01818871 0.5353160  0.03397752 1.168163 144
## [13] {134, 270}      => {175} 0.01035746 0.5222930  0.01983074 1.139745  82
## [14] {110, 134}      => {175} 0.02968296 0.5142232  0.05772389 1.122135 235
## [15] {110, 150}      => {175} 0.03334596 0.5106383  0.06530251 1.114312 264
## [16] {110, 330}      => {175} 0.01528357 0.5105485  0.02993558 1.114116 121
## [17] {150, 330}      => {175} 0.02122016 0.4970414  0.04269294 1.084641 168
## [18] {134, 165}      => {175} 0.01692560 0.4962963  0.03410383 1.083015 134
## [19] {150, 380}      => {175} 0.01199949 0.4896907  0.02450423 1.068600  95
## [20] {150, 270}      => {175} 0.01098901 0.4887640  0.02248326 1.066578  87
```

We can see that Mainstream - young singles/couples tends to buy the 175g pack size.