Quantium VI

Rolly Mougoue

2022-04-19

Task1- Data preparation and customer analytics

Load required libraries and datasets

We'll start with setting up our working environment by installing and loading all the necessary packages (Tidyverse, Lubridate, dplyr, pad...) and setting the working directory for easy upload of our datasets.

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --
```

```
## v ggplot2 3.3.5     v purrr   0.3.4
## v tibble  3.1.6     v dplyr   1.0.8
## v tidyr   1.2.0     v stringr 1.4.0
## v readr   2.1.2     v forcats 0.5.1
```

```
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(data.table)
```

```
##
## Attaching package: 'data.table'
```

```
## The following objects are masked from 'package:dplyr':
##
##     between, first, last
```

```
## The following object is masked from 'package:purrr':
##
##     transpose
```

```
library(ggplot2)
library(ggmosaic)
library(readr)
library(readxl)
```

```r
## Setting the work directory and uploading our data sets


setwd(dir = "D:/Google Data Analytics/The Forage Projects/Quantium/Datasets")


purchase_behaviour <- read_csv("QVI_purchase_behaviour.csv")
```

```
## Rows: 72637 Columns: 3

## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (2): LIFESTAGE, PREMIUM_CUSTOMER
## dbl (1): LYLTY_CARD_NBR
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```r
Transactions <- read_excel("QVI_transaction_data.xlsx")
```

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()` or `glimpse()` 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 `Transactions` in the console to see a sample of the data or use `head(Transactions)` to look at the first 10 rows. Let's check if columns we would expect to be numeric are in numeric form and date columns are in date format.

```r
str(Transactions)
```

```
## tibble [264,836 x 8] (S3: tbl_df/tbl/data.frame)
##  $ DATE          : num [1:264836] 43390 43599 43605 43329 43330 ...
##  $ STORE_NBR     : num [1:264836] 1 1 1 2 2 4 4 4 5 7 ...
##  $ LYLTY_CARD_NBR: num [1:264836] 1000 1307 1343 2373 2426 ...
##  $ TXN_ID        : num [1:264836] 1 348 383 974 1038 ...
##  $ PROD_NBR      : num [1:264836] 5 66 61 69 108 57 16 24 42 52 ...
##  $ PROD_NAME     : chr [1:264836] "Natural Chip        Compny SeaSalt175g" "CCs Nacho Cheese    175g
##  $ PROD_QTY      : num [1:264836] 2 3 2 5 3 1 1 1 1 2 ...
##  $ TOT_SALES     : num [1:264836] 6 6.3 2.9 15 13.8 5.1 5.7 3.6 3.9 7.2 ...
```

```r
Transactions <- data.table(Transactions)
purchase_behaviour <- data.table(purchase_behaviour)
```

Examining the DATE variable

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
Transactions$DATE <- as.Date(Transactions$DATE, origin = "1899-12-30")
```

Examining the PROD_NAME variable

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

```
#### Examine PROD_NAME
Prod_Names <- as.data.frame(table(Transactions$PROD_NAME))
```

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.

```
#### Examine the words in PROD_NAME to see if there are any incorrect entries
productWords <- data.table(unlist(strsplit(unique(Transactions$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 ourset of product words. We can do this using `grepl()`.

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

#### Removing special characters
productWords <- productWords[grepl("[:alpha:]", words), ]

#### Let's look at the most common words by counting the number of times a word appears and sorting the
productWords <- 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.

```
#### remove the salsa product
Transactions <- Transactions %>%
  dplyr::filter(!grepl("Salsa", Transactions$PROD_NAME))

#### summarizing the data
summary(Transactions)
```

```
##       DATE                STORE_NBR     LYLTY_CARD_NBR        TXN_ID
##  Min.   :2018-07-01   Min.   :  1.0   Min.   :   1000   Min.   :      1
##  1st Qu.:2018-09-30   1st Qu.: 70.0   1st Qu.:  70015   1st Qu.:  67569
##  Median :2018-12-30   Median :130.0   Median : 130367   Median : 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  
```

In product quantity it is visible that in PROD_QTY column the max value is of 200 i.e 200 quantity were purchased at once.. hence this can be considered as a outlier to our data.. there are no nulls as all the summary statistics have a numerical value # finding the outlier in PROD_QTY

```
#### Filter the dataset to find the outlier
filter(Transactions, 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
```

We can observe that there were two transactions done that had product quantity > 10. Both the transaction were done using the same loyalty card number and the same store. Let's see if the customer has had other transactions

```
filter(Transactions, 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.

```
# Filter out the customer based on the loyalty card number
Transactions <- filter(Transactions, LYLTY_CARD_NBR != 226000)
#### Re-examine transaction data
summary(Transactions)
```

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

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.

```
#### Count the number of transactions by date
Transactions %>%
  group_by(DATE) %>%
  summarise(num_trans = n())
```

```
## # A tibble: 364 x 2
##    DATE       num_trans
##    <date>         <int>
##  1 2018-07-01       663
##  2 2018-07-02       650
##  3 2018-07-03       674
##  4 2018-07-04       669
##  5 2018-07-05       660
##  6 2018-07-06       711
##  7 2018-07-07       695
##  8 2018-07-08       653
##  9 2018-07-09       692
## 10 2018-07-10       650
## # ... with 354 more rows
```

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.

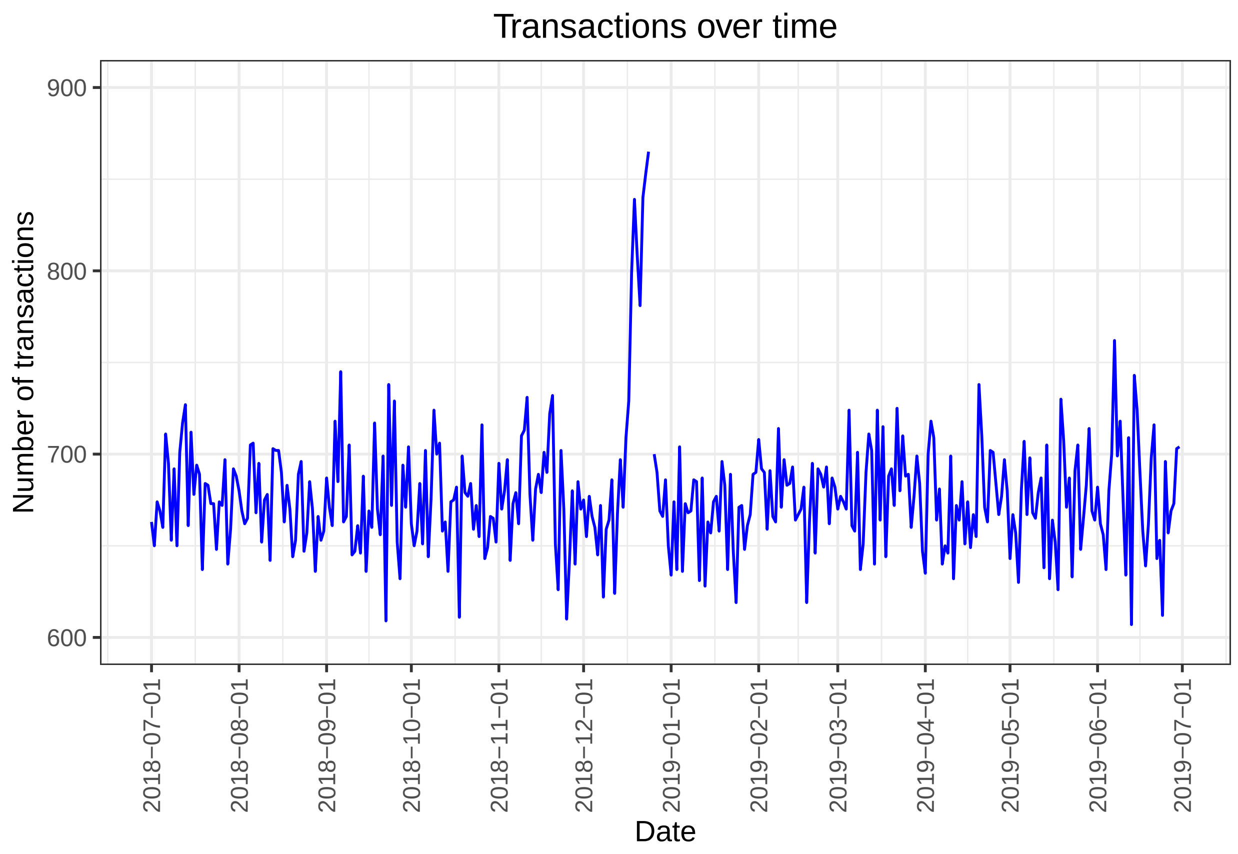
```
#### Create a sequence of dates and join this the count of transactions by date
allDates <- data.table(seq(as.Date("2018/07/01"), as.Date("2019/06/30"), by = "day"))
setnames(allDates, "DATE")
transactions_by_day <- merge(allDates, Transactions, by = "DATE", all.y = TRUE, all.x = TRUE) %>%
  group_by(DATE) %>%
  summarise(num_trans = n())
```

Setting plot themes to format graphs

```
theme_set(theme_bw())
theme_update(plot.title = element_text(hjust = 0.5))
```

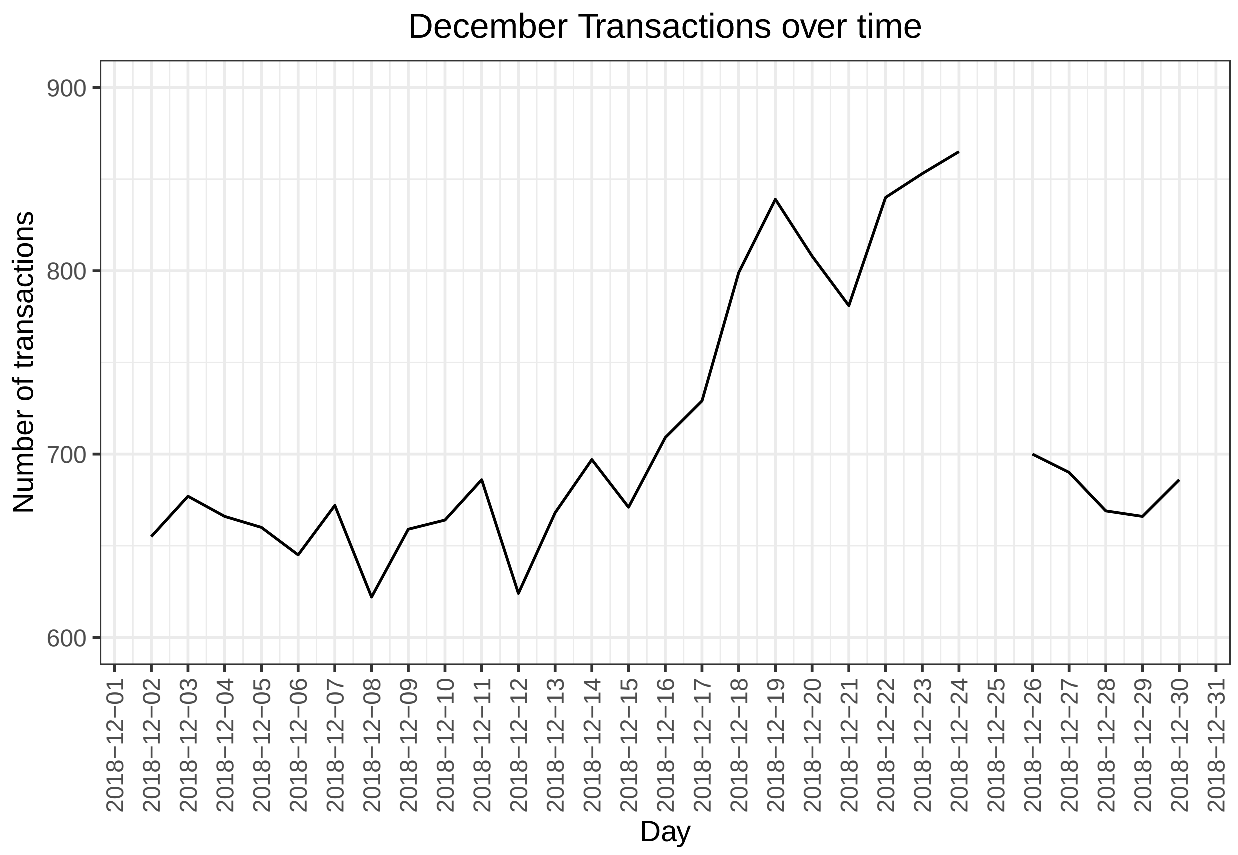
Plot transactions over time

```
ggplot(transactions_by_day, aes(x = DATE, y = num_trans)) +
  geom_line(col = "Blue") +
  labs(x = "Date", y = "Number of transactions", title = "Transactions over time") +
  scale_x_date(breaks = "1 month") + ylim(600, 900) +
  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
ggplot(transactions_by_day[transactions_by_day$DATE > "2018-12-01" & transactions_by_day$DATE < "2018-1
  geom_line( aes(x = DATE, y = num_trans)) +
  labs(x = "Day", y = "Number of transactions", title = "December Transactions over time") +
  scale_x_date(breaks = "1 day") + ylim(600, 900) +
  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.

Examining the Chips Pack Sizes and Brand

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.

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

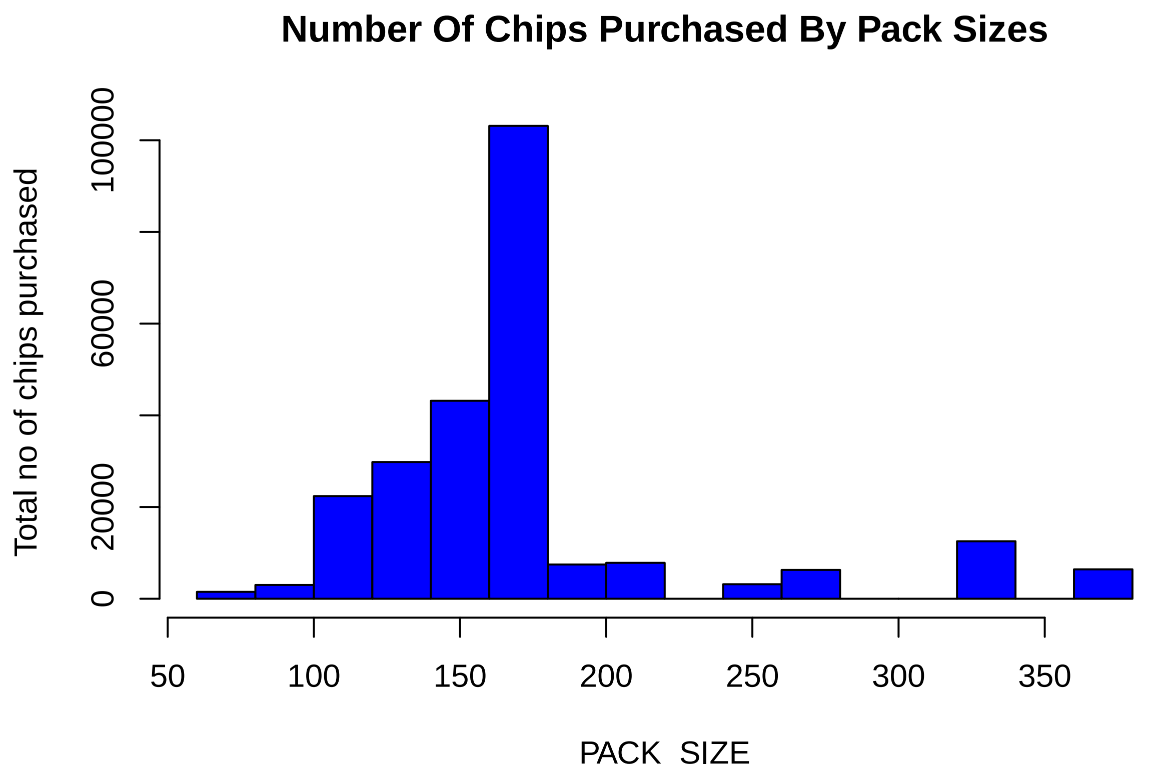
```
## Warning in ‘[.data.table‘(Transactions, , ‘:=‘(PACK_SIZE,
## parse_number(PROD_NAME))): Invalid .internal.selfref detected and fixed by
## taking a (shallow) copy of the data.table so that := can add this new column
## by reference. At an earlier point, this data.table has been copied by R (or
## was created manually using structure() or similar). Avoid names<- and attr<-
## which in R currently (and oddly) may copy the whole data.table. Use set* syntax
## instead to avoid copying: ?set, ?setnames and ?setattr. If this message doesn’t
## help, please report your use case to the data.table issue tracker so the root
## cause can be fixed or this message improved.
```

```
#### Let's check if the pack sizes look sensible
Transactions[, .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! Now 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.

```
#### plotting histogram of the packsize
options(scipen=999) # turn off scientific notations like 1e+05
hist(Transactions[, PACK_SIZE], col = "blue",border = "black" , xlab = "PACK  SIZE", ylab = "Total no o
```



the plot looks reasonable with no outliers and from the plot it can be seen that the the packs of size 170-180 was purchased the most. Now to create brands, we can use the first word in PROD_NAME to work out the brand name...

```
#### we'll use the first word of the PROD_NAME to create our data of brand
Transactions[, Brands := toupper(substr(PROD_NAME, 1, regexpr(pattern = ' ', PROD_NAME) - 1))]


#### Checking brands
Transactions[, .N, by = Brands][order(-N)]
```

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

```
Transactions[Brands == "RED", Brands := "RRD"]
Transactions[Brands == "SNBTS", Brands := "SUNBITES"]
Transactions[Brands == "INFZNS", Brands := "INFUZIONS"]
Transactions[Brands == "WW", Brands := "WOOLWORTHS"]
Transactions[Brands == "SMITH", Brands := "SMITHS"]
Transactions[Brands == "NCC", Brands := "NATURAL"]
Transactions[Brands == "DORITO", Brands := "DORITOS"]
Transactions[Brands == "GRAIN", Brands := "GRNWVES"]

#### Checking again brand names
Transactions[, .N, by = Brands][order(Brands)]
```

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

Now 8 of our rows that had similar brand has been merged and now we can finally stop our data exploration and continue further..

Checking the customer data

Examining customer data

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

```
str(purchase_behaviour)
```

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

```
summary(purchase_behaviour)
```

```
##  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
```

we can see that the data is mostly descriptive. It gives the description of the customer who purchased the chip, the loyalty card number is a numeric vector while lifestage and premium_customer are character vectors

Let's have a closer look at the LIFESTAGE and PREMIUM_CUSTOMER columns.

```
##### Examining the values of lifestage and premium_customer
purchase_behaviour[, .N, by = LIFESTAGE][order(-N)]
```

```
##                 LIFESTAGE     N
## 1:               RETIREES 14805
## 2:  OLDER SINGLES/COUPLES 14609
## 3:  YOUNG SINGLES/COUPLES 14441
## 4:         OLDER FAMILIES  9780
## 5:         YOUNG FAMILIES  9178
## 6: MIDAGE SINGLES/COUPLES  7275
## 7:           NEW FAMILIES  2549
```

```
purchase_behaviour[, .N, by = PREMIUM_CUSTOMER][order(-N)]
```

```
##    PREMIUM_CUSTOMER     N
## 1:       Mainstream 29245
## 2:           Budget 24470
## 3:          Premium 18922
```

Merge transaction data to customer data

```r
data <- merge(Transactions, purchase_behaviour , all.x = TRUE)
```

As the number of rows in `data` is the same as that of `Transactions`, 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 `Transactions` 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
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            Brands         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. Hence we can proceed further since all the data was matched properly

```r
#### saving the cleaned file for further analysis
write.csv(data, "QVI_Cleaned_data.csv")
```

We've now completed data exploration from different datasets.

Data analysis on customer segments

Now that the data is ready for analysis, we can define some metrics of interest to the client such as: 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

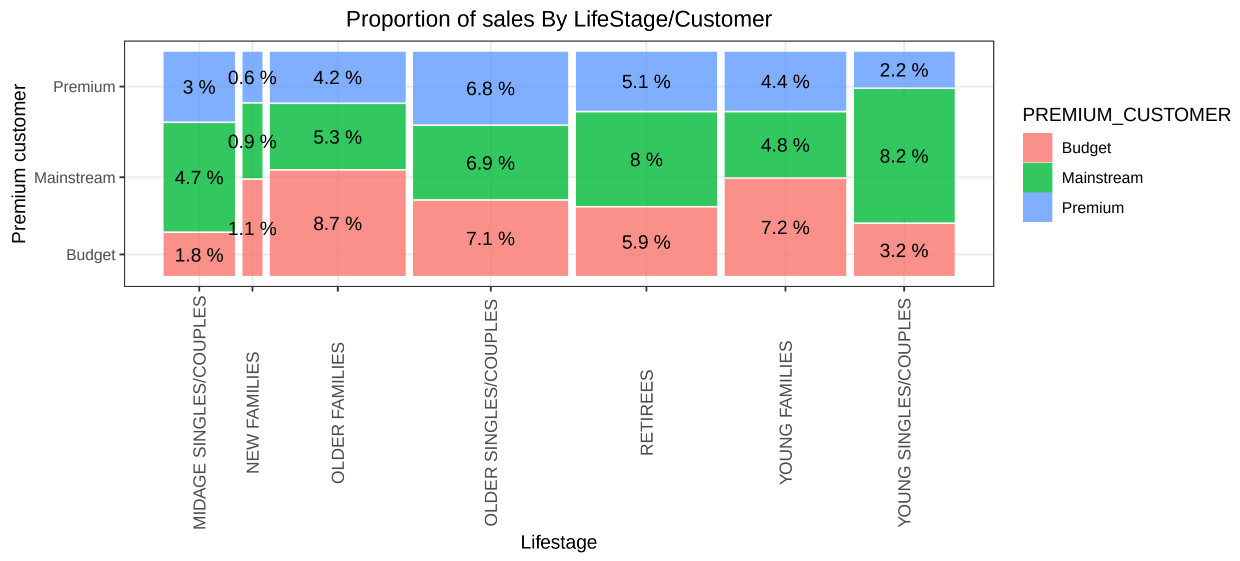
Let's start with calculating total sales by LIFESTAGE and PREMIUM_CUSTOMER and plotting the split by these segments to describe which customer segment contribute most to chip sales.

```r
# Total sales by LIFESTAGE and PREMIUM_CUSTOMER
Total_Sales <- data %>%
  group_by(LIFESTAGE, PREMIUM_CUSTOMER) %>%
  summarise(Sales = sum(TOT_SALES, na.rm = TRUE)) %>%
  na.omit()
```

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

```r
# create plot
plot <- ggplot(data = Total_Sales) +
  geom_mosaic(aes(weight = Sales, x = product(PREMIUM_CUSTOMER, LIFESTAGE) , fill = PREMIUM_CUSTOMER))
  labs(x = "Lifestage", y = "Premium customer", title = "Proportion of sales By LifeStage/Customer") +
# Plot and label with proportion of sales
plot +
  geom_text(data = ggplot_build(plot)$data[[1]], aes(x = (xmin + xmax)/2 , y = (ymin + ymax)/2, label =
```

```
## Warning: `unite_()` was deprecated in tidyr 1.2.0.
## Please use `unite()` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was generated.
```

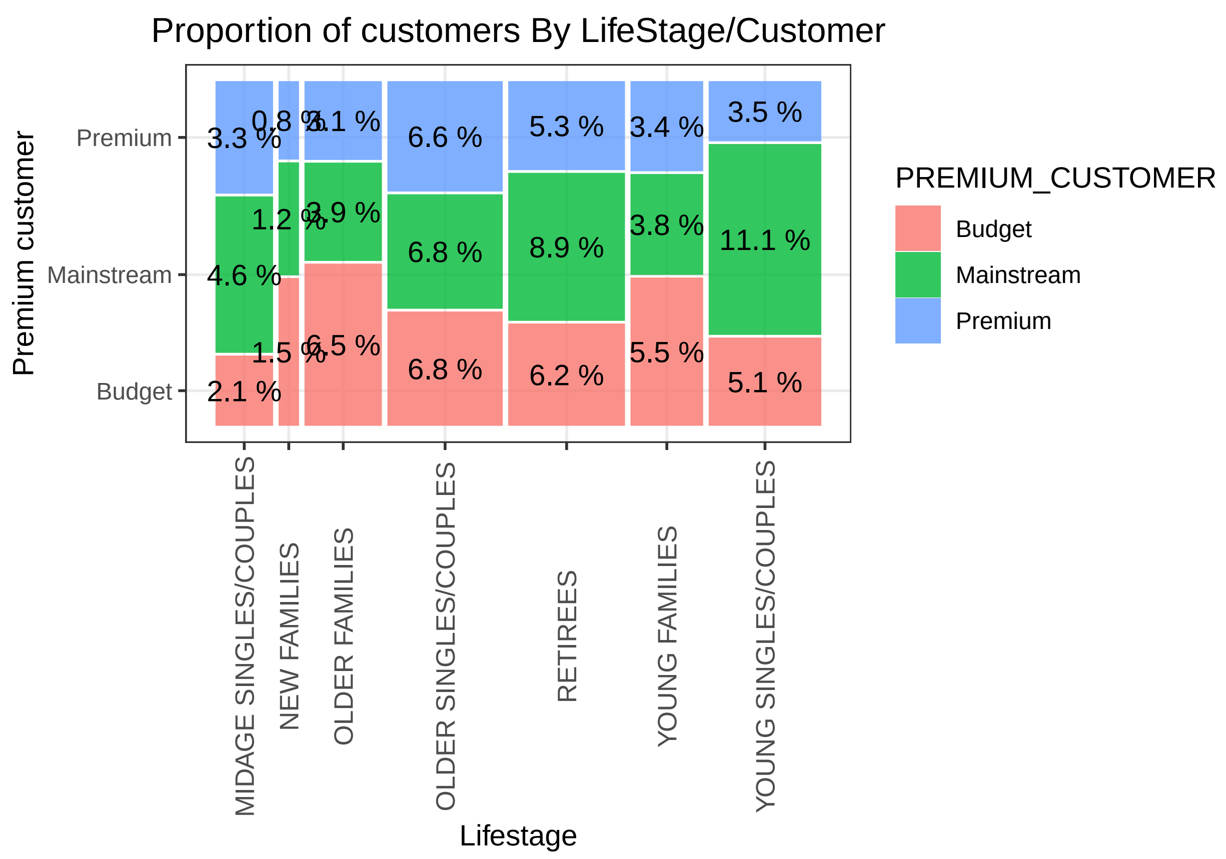


We can observe from the plot that mainstream- young/single couples and mainstream retirees contribute the most to there being higher sales of chips which isn't the case for budget- older families segment.

```
## Number of customers by LIFESTAGE and PREMIUM_CUSTOMER
Customers <- data[, .(customers = uniqueN(LYLTY_CARD_NBR)), .(LIFESTAGE, PREMIUM_CUSTOMER)][order(-custo


# Create plot
Plot <- ggplot(data = Customers) +
  geom_mosaic(aes(weight = customers, x = product(PREMIUM_CUSTOMER, LIFESTAGE), fill = PREMIUM_CUSTOMER)
  labs(x = "Lifestage", y = "Premium customer", title = "Proportion of customers By LifeStage/Customer")

Plot + geom_text(data = ggplot_build(Plot)$data[[1]], aes(x = (xmin + xmax)/2 , y = (ymin + ymax)/2, lab
```



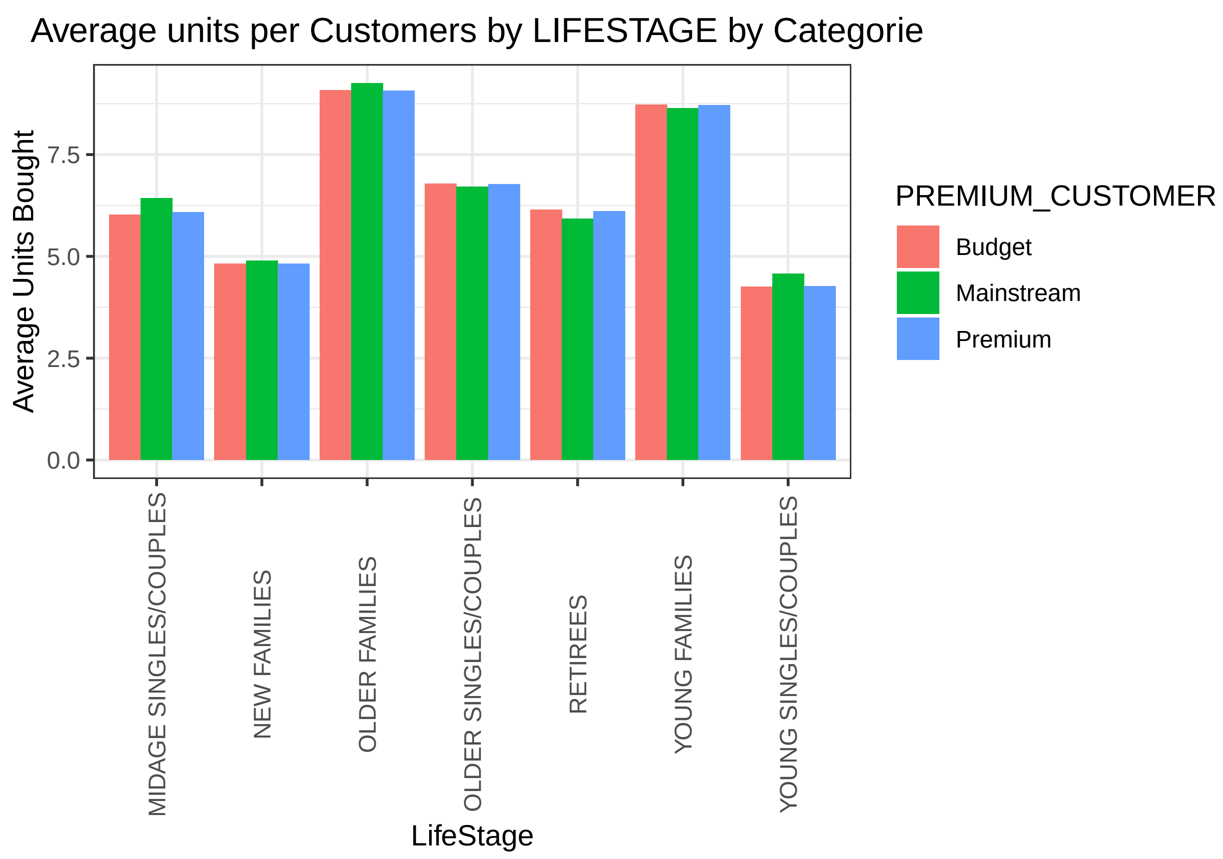
There are more Mainstream - young singles/couples and Mainstream - retirees who buy chips. This contributes to there being more sales to these customer segments but this is not a major driver for the Budget-Older families segment.

Higher sales may also be driven by number of chips bought by each customer.. hence we'll try to plot average number of chips(average number of PROD_QTY) by lifestage and premium_customer. Let's have a look at this next

```
#### Average number of units per customer by LIFESTAGE and PREMIUM_CUSTOMER
Average_Units <- data %>%
  group_by(LIFESTAGE, PREMIUM_CUSTOMER) %>%
  summarise(Quantity = sum(PROD_QTY)/uniqueN(LYLTY_CARD_NBR)) %>%
  na.omit()
```

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

```
ggplot(data = Average_Units, aes(weight = Quantity , x = LIFESTAGE, fill = PREMIUM_CUSTOMER)) +
  geom_bar(position = position_dodge()) +
  labs(title = " Average units per Customers by LIFESTAGE by Categorie", x = "LifeStage", y = "Average 
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5))
```

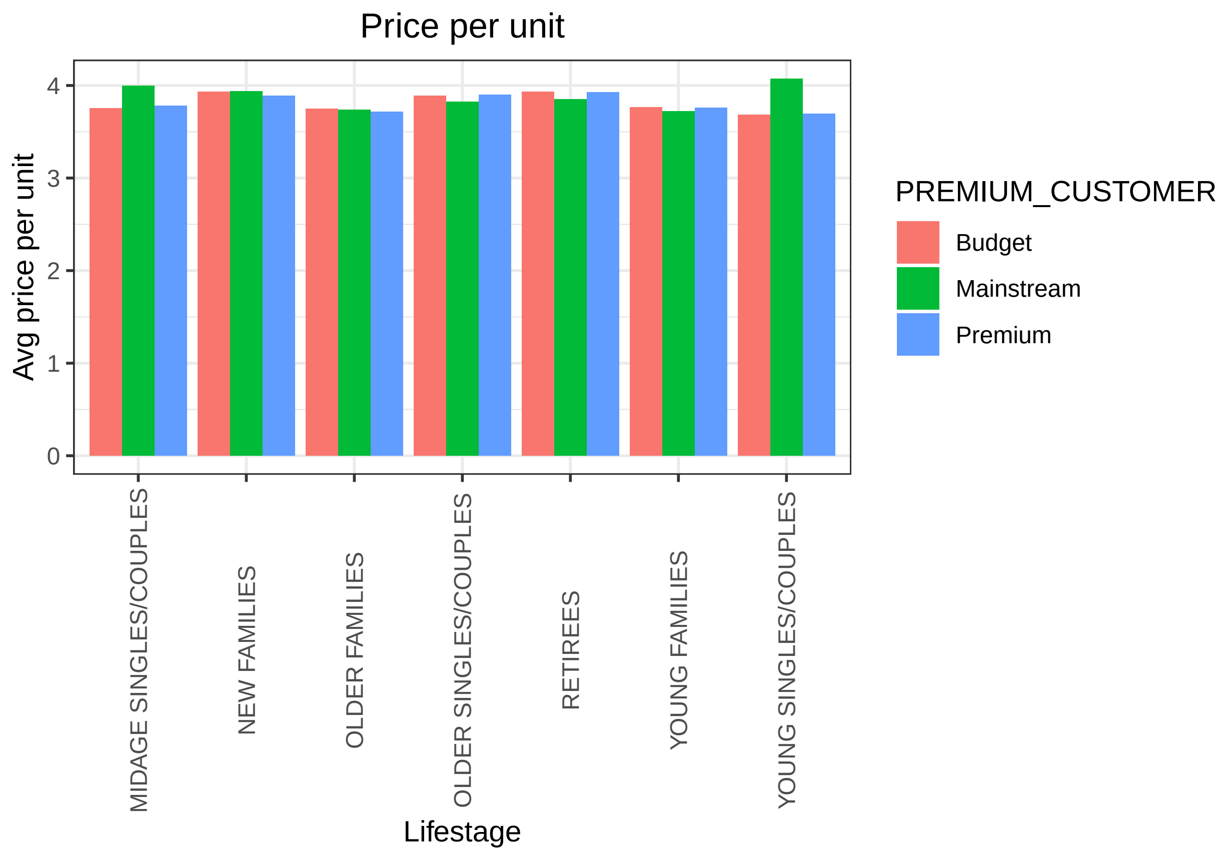


Older families and young families in general buy more chips per customer Let's also investigate the average price per unit chips bought for each customer segment as this is also a driver of total sales.

```
#### Average price per unit by LIFESTAGE and PREMIUM_CUSTOMER
avg_price <- data[, .(AVG = sum(TOT_SALES)/sum(PROD_QTY)), .(LIFESTAGE, PREMIUM_CUSTOMER)][order(-AVG)]
avg_price <- data %>%
  group_by(LIFESTAGE, PREMIUM_CUSTOMER) %>%
  summarise(AVG = sum(TOT_SALES)/sum(PROD_QTY)) %>%
  na.omit()
```

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

```
#### Create plot
ggplot(data = avg_price, aes(weight = AVG, x = LIFESTAGE, fill = PREMIUM_CUSTOMER)) + geom_bar(position
```



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 between the average price per unit is not same we can check if this difference is stastically significant or not by performing independent t-test between mainstream vs premium and budget midage and young young single couples.

```
#### young singles and couples
pricePerUnit <- data[, price := TOT_SALES/PROD_QTY]
t.test(data[LIFESTAGE %in% c("YOUNG SINGLES/COUPLES", "MIDAGE SINGLES/COUPLES") & PREMIUM_CUSTOMER == "I
, data[LIFESTAGE %in% c("YOUNG SINGLES/COUPLES", "MIDAGE SINGLES/COUPLES") & PREMIUM_CUSTOMER != "Mains
, alternative = "greater")
```

```
##
##  Welch Two Sample t-test
##
## data:  data[LIFESTAGE %in% c("YOUNG SINGLES/COUPLES", "MIDAGE SINGLES/COUPLES") & PREMIUM_CUSTOMER ==
## t = 37.624, df = 54791, p-value < 0.000000000000000022
## alternative hypothesis: true difference in means is greater than 0
## 95 percent confidence interval:
##  0.3187234       Inf
## sample estimates:
## mean of x mean of y
##  4.039786  3.706491
```

The t-test results in a p-value of 2.2e-16. Meaning 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.

```
# Deep dive into Mainstream, young singles/couples
Customer_Type1 <- data[LIFESTAGE == "YOUNG SINGLES/COUPLES" & PREMIUM_CUSTOMER == "Mainstream",]
Customer_Type2<- data[!(LIFESTAGE == "YOUNG SINGLES/COUPLES" & PREMIUM_CUSTOMER == "Mainstream"),]

## Brand affinity compared to the rest of the population
quantity_type1 <- Customer_Type1[, sum(PROD_QTY)]

quantity_type2 <- Customer_Type2[, sum(PROD_QTY)]

quantity_type1_by_brand <- Customer_Type1[, .(targetSegment = sum(PROD_QTY)/quantity_type1), by = Brand

quantity_type2_by_brand <- Customer_Type2[, .(other = sum(PROD_QTY)/quantity_type2), by = Brands]

brand_proportions <- merge(quantity_type1_by_brand, quantity_type2_by_brand)[, affinityToBrand := targe

brand_proportions[order(-affinityToBrand)]
```

```
##         Brands targetSegment       other affinityToBrand
##  1:   TYRRELLS   0.031552795 0.025692464       1.2280953
##  2:   TWISTIES   0.046183575 0.037876520       1.2193194
##  3:    DORITOS   0.122760524 0.101074684       1.2145526
##  4:     KETTLE   0.197984817 0.165553442       1.1958967
##  5:   TOSTITOS   0.045410628 0.037977861       1.1957131
##  6:   PRINGLES   0.119420290 0.100634769       1.1866703
##  7:       COBS   0.044637681 0.039048861       1.1431238
##  8:  INFUZIONS   0.064679089 0.057064679       1.1334347
##  9:      THINS   0.060372671 0.056986370       1.0594230
## 10:    GRNWVES   0.032712215 0.031187957       1.0488733
## 11:   CHEEZELS   0.017971014 0.018646902       0.9637534
## 12:     SMITHS   0.096369910 0.124583692       0.7735355
## 13:     FRENCH   0.003947550 0.005758060       0.6855694
## 14:    CHEETOS   0.008033126 0.012066591       0.6657329
## 15:        RRD   0.043809524 0.067493678       0.6490908
## 16:    NATURAL   0.019599724 0.030853989       0.6352412
## 17:        CCS   0.011180124 0.018895650       0.5916771
## 18:   SUNBITES   0.006349206 0.012580210       0.5046980
## 19: WOOLWORTHS   0.024099379 0.049427188       0.4875733
## 20:     BURGER   0.002926156 0.006596434       0.4435967
```

We can see that : - Mainstream young singles/couples are 23% more likely to purchase Tyrrells chips compared to the rest of the population - Mainstream young singles/couples are 56% less likely to purchase Burger Rings compared to the rest of the population

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
quantity_type1_by_pack <- Customer_Type1[, .(targetSegment = sum(PROD_QTY)/quantity_type1), by = PACK_SI
quantity_type2_by_pack <- Customer_Type2[, .(other = sum(PROD_QTY)/quantity_type2), by = PACK_SIZE]

pack_proportions <- merge(quantity_type1_by_pack, quantity_type2_by_pack)[, affinityToPack := targetSegm

pack_proportions[order(-affinityToPack)]
```

```
##     PACK_SIZE targetSegment       other affinityToPack
##  1:       270   0.031828847 0.025095929      1.2682873
##  2:       380   0.032160110 0.025584213      1.2570295
##  3:       330   0.061283644 0.050161917      1.2217166
##  4:       134   0.119420290 0.100634769      1.1866703
##  5:       110   0.106280193 0.089791190      1.1836372
##  6:       210   0.029123533 0.025121265      1.1593180
##  7:       135   0.014768806 0.013075403      1.1295106
##  8:       250   0.014354727 0.012780590      1.1231662
##  9:       170   0.080772947 0.080985964      0.9973697
## 10:       150   0.157598344 0.163420656      0.9643722
## 11:       175   0.254989648 0.270006956      0.9443818
## 12:       165   0.055652174 0.062267662      0.8937572
## 13:       190   0.007481021 0.012442016      0.6012708
## 14:       180   0.003588682 0.006066692      0.5915385
## 15:       160   0.006404417 0.012372920      0.5176157
## 16:        90   0.006349206 0.012580210      0.5046980
## 17:       125   0.003008972 0.006036750      0.4984423
## 18:       200   0.008971705 0.018656115      0.4808989
## 19:        70   0.003036577 0.006322350      0.4802924
## 20:       220   0.002926156 0.006596434      0.4435967
```

It looks like Mainstream young singles/couples are 27% more likely to purchase a 270g pack of chips compared to the rest of the population but let's dive into what brands sell this pack size.

```r
data[PACK_SIZE == 270,unique(PROD_NAME)]
```

```
## [1] "Twisties Cheese     270g" "Twisties Chicken270g"
```

The only brand offering 270g packs are Twisties and this may instead be reflecting a higher likelihood of purchasing them.

Conclusion // Insights

Let's recap what we've found during this first analysis!

Sales have mainly been due to Budget - older families, Mainstream - young singles/couples, and Mainstream retirees shoppers. We noticed a correlation between high spend in chips for mainstream young singles/couples and retirees and their number in the sample as compared to other buyers. Mainstream, midage and young singles and couples are also more likely to pay more per packet of chips. This is indicative of impulse buying behaviour. **We've also found that Mainstream young singles and couples are 23% more likely to purchase Tyrrells chips compared to the rest of the population