quantium

2023-03-25

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. Page 1 20200128__InsideSherpa__Task1__DraftSolutions - Template (1).Rmd 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")
```

```
## Installing package into '/cloud/lib/x86_64-pc-linux-gnu-library/4.2'
## (as 'lib' is unspecified)
```

```
install.packages("ggplot2")
```

```
## Installing package into '/cloud/lib/x86_64-pc-linux-gnu-library/4.2'
## (as 'lib' is unspecified)
```

```
install.packages("ggmosaic")
```

```
## Installing package into '/cloud/lib/x86_64-pc-linux-gnu-library/4.2'
## (as 'lib' is unspecified)
```

```
install.packages("readr")
```

```
## Installing package into '/cloud/lib/x86_64-pc-linux-gnu-library/4.2'
## (as 'lib' is unspecified)
```

```
#### Load required libraries
library(data.table)
library(ggplot2)
library(ggmosaic)
library(readr)
#### Point the filePath to where you have downloaded the datasets to and
#### assign the data files to data.tables
# over to you! fill in the path to your working directory. If you are on a Windows machine, you will ne
filePath <- "/cloud/project/"
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")
str(transactionData)
```

```
## Classes 'data.table' and 'data.frame': 264836 obs. of 8 variables:
## $ DATE : Date, format: "2018-10-17" "2019-05-14" ...
## $ 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 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.
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 summarising the individual words in the product name.

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

```
##                                       words
## 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
```

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()`.

```
# Over to you! Remove digits, and special characters, and then sort the distinct words by frequency of
#### Removing digits

#### Removing special characters
productWords$words <- gsub("[^[:alpha:][:space:]]", "", productWords$words)
head(productWords)
```

```
##                                    words
## 1:   Natural Chip        Compny SeaSaltg
## 2:                 CCs Nacho Cheese    g
## 3:   Smiths Crinkle Cut  Chips Chicken g
## 4:     Smiths Chip Thinly  SCreamOnion g
## 5:  Kettle Tortilla ChpsHnyJlpno Chili g
## 6: Old El Paso Salsa   Dip Tomato Mild g
```

```
#### 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
library(dplyr)
```

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

## 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
# Group the data by words and count the frequency of occurrence
word_freq <- productWords %>%
  group_by(words) %>%
  summarize(count = n())
# Sort the data in descending order of frequency
word_freq <- word_freq %>%
  arrange(desc(count))
# View the top 10 most frequent words
head(word_freq, 10)
```

```
## # A tibble: 10 x 2
##    words                              count
##    <chr>                              <int>
##  1 Burger Rings g                         1
##  2 CCs Nacho Cheese    g                  1
##  3 CCs Original g                         1
##  4 CCs Tasty Cheese    g                  1
##  5 Cheetos Chs  Bacon Balls g             1
##  6 Cheetos Puffs g                        1
##  7 Cheezels Cheese Box g                  1
##  8 Cheezels Cheese g                      1
##  9 Cobs Popd Sea Salt  Chips g            1
## 10 Cobs Popd Sour Crm  Chives Chips g     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]
```

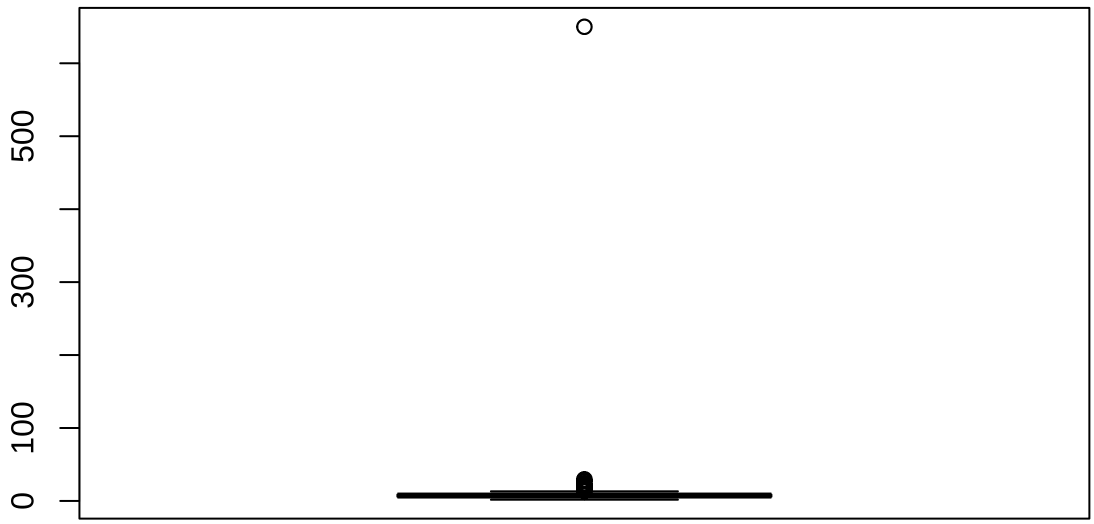
Next, we can use `summary()` to check summary statistics such as mean, min and max values for each feature to see if there are any obvious outliers in the data and if there are any nulls in any of the columns (`NA's : number of nulls` will appear in the output if there are any nulls).

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

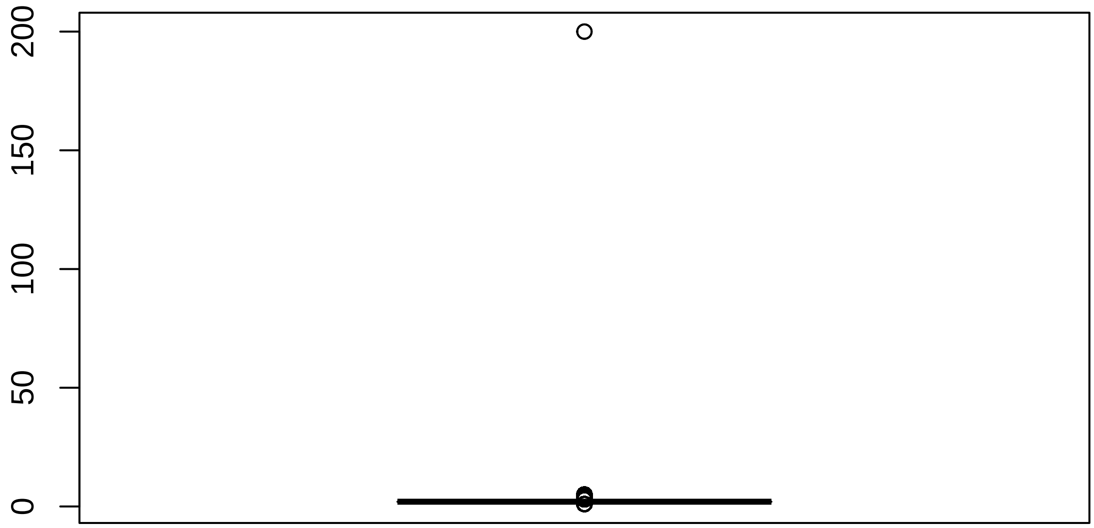
```
##       DATE              STORE_NBR     LYLTY_CARD_NBR        TXN_ID
##  Min.   :2018-07-01   Min.   :  1.0   Min.   :   1000   Min.   :      1
##  1st Qu.:2018-09-30   1st Qu.: 70.0   1st Qu.:  70015   1st Qu.:  67569
##  Median :2018-12-30   Median :130.0   Median : 130367   Median : 135183
##  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
```

```r
# Create a boxplot to check for outliers
boxplot(transactionData$TOT_SALES)
```



```r
boxplot(transactionData$PROD_QTY)
```



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_filtered <- transactionData[transactionData$PROD_QTY == 200, ]
```

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_filtered
```

```
##          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
```

```
transactionData_filtered1 <- transactionData[transactionData$LYLTY_CARD_NBR == 226000, ]
transactionData_filtered1
```

```
##          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
# Over to you!
#### Re-examine transaction data
# Over to you!
# Remove rows where LYLTY_CARD_NBR is equal to 226000
transactionData <- transactionData[transactionData$LYLTY_CARD_NBR != 226000, ]
# View the modified data frame
str(transactionData)
```

```
## Classes 'data.table' and 'data.frame': 246740 obs. of 8 variables:
## $ DATE : Date, format: "2018-10-17" "2019-05-14" ...
## $ STORE_NBR : int 1 1 1 2 2 4 4 5 7 7 ...
## $ LYLTY_CARD_NBR: int 1000 1307 1343 2373 2426 4149 4196 5026 7150 7215 ...
## $ TXN_ID : int 1 348 383 974 1038 3333 3539 4525 6900 7176 ...
## $ PROD_NBR : int 5 66 61 69 108 16 24 42 52 16 ...
## $ 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 2 1 ...
## $ TOT_SALES : num 6 6.3 2.9 15 13.8 5.7 3.6 3.9 7.2 5.7 ...
## - attr(*, ".internal.selfref")=<externalptr>
```

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
# Over to you! Create a summary of transaction count by date.
# Convert date column to a Date object
transactionData$DATE <- as.Date(transactionData$DATE, format = "%Y-%m-%d")

# Aggregate transaction count by date
transactions_by_date <- aggregate(x = list(count = transactionData$TXN_ID),
                                  by = list(date = transactionData$DATE),
                                  FUN = length)

# Print the summary
print(transactions_by_date)
```

```
##          date count
## 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
## 11 2018-07-11   701
## 12 2018-07-12   717
## 13 2018-07-13   727
## 14 2018-07-14   661
## 15 2018-07-15   712
## 16 2018-07-16   678
## 17 2018-07-17   694
## 18 2018-07-18   689
## 19 2018-07-19   637
## 20 2018-07-20   684
## 21 2018-07-21   683
## 22 2018-07-22   673
## 23 2018-07-23   673
## 24 2018-07-24   648
## 25 2018-07-25   674
## 26 2018-07-26   672
## 27 2018-07-27   697
## 28 2018-07-28   640
## 29 2018-07-29   659
## 30 2018-07-30   692
## 31 2018-07-31   688
## 32 2018-08-01   680
## 33 2018-08-02   669
## 34 2018-08-03   662
## 35 2018-08-04   665
## 36 2018-08-05   705
## 37 2018-08-06   706
## 38 2018-08-07   668
## 39 2018-08-08   695
## 40 2018-08-09   652
## 41 2018-08-10   675
## 42 2018-08-11   678
## 43 2018-08-12   642
## 44 2018-08-13   703
## 45 2018-08-14   702
## 46 2018-08-15   702
## 47 2018-08-16   690
## 48 2018-08-17   663
## 49 2018-08-18   683
## 50 2018-08-19   670
## 51 2018-08-20   644
## 52 2018-08-21   653
## 53 2018-08-22   689
```

```
## 54  2018-08-23   696
## 55  2018-08-24   647
## 56  2018-08-25   657
## 57  2018-08-26   685
## 58  2018-08-27   670
## 59  2018-08-28   636
## 60  2018-08-29   666
## 61  2018-08-30   653
## 62  2018-08-31   658
## 63  2018-09-01   687
## 64  2018-09-02   671
## 65  2018-09-03   661
## 66  2018-09-04   718
## 67  2018-09-05   685
## 68  2018-09-06   745
## 69  2018-09-07   663
## 70  2018-09-08   666
## 71  2018-09-09   705
## 72  2018-09-10   645
## 73  2018-09-11   647
## 74  2018-09-12   661
## 75  2018-09-13   646
## 76  2018-09-14   688
## 77  2018-09-15   636
## 78  2018-09-16   669
## 79  2018-09-17   660
## 80  2018-09-18   717
## 81  2018-09-19   670
## 82  2018-09-20   656
## 83  2018-09-21   699
## 84  2018-09-22   609
## 85  2018-09-23   738
## 86  2018-09-24   672
## 87  2018-09-25   729
## 88  2018-09-26   652
## 89  2018-09-27   632
## 90  2018-09-28   694
## 91  2018-09-29   671
## 92  2018-09-30   704
## 93  2018-10-01   662
## 94  2018-10-02   650
## 95  2018-10-03   658
## 96  2018-10-04   684
## 97  2018-10-05   651
## 98  2018-10-06   702
## 99  2018-10-07   644
## 100 2018-10-08   676
## 101 2018-10-09   724
## 102 2018-10-10   700
## 103 2018-10-11   706
## 104 2018-10-12   658
## 105 2018-10-13   663
## 106 2018-10-14   636
## 107 2018-10-15   674
```

```
## 108 2018-10-16   675
## 109 2018-10-17   682
## 110 2018-10-18   611
## 111 2018-10-19   699
## 112 2018-10-20   679
## 113 2018-10-21   677
## 114 2018-10-22   684
## 115 2018-10-23   659
## 116 2018-10-24   672
## 117 2018-10-25   655
## 118 2018-10-26   716
## 119 2018-10-27   643
## 120 2018-10-28   649
## 121 2018-10-29   666
## 122 2018-10-30   665
## 123 2018-10-31   652
## 124 2018-11-01   695
## 125 2018-11-02   670
## 126 2018-11-03   680
## 127 2018-11-04   697
## 128 2018-11-05   642
## 129 2018-11-06   673
## 130 2018-11-07   679
## 131 2018-11-08   662
## 132 2018-11-09   710
## 133 2018-11-10   713
## 134 2018-11-11   731
## 135 2018-11-12   678
## 136 2018-11-13   653
## 137 2018-11-14   681
## 138 2018-11-15   689
## 139 2018-11-16   679
## 140 2018-11-17   701
## 141 2018-11-18   690
## 142 2018-11-19   722
## 143 2018-11-20   732
## 144 2018-11-21   651
## 145 2018-11-22   626
## 146 2018-11-23   702
## 147 2018-11-24   670
## 148 2018-11-25   610
## 149 2018-11-26   642
## 150 2018-11-27   680
## 151 2018-11-28   640
## 152 2018-11-29   685
## 153 2018-11-30   670
## 154 2018-12-01   675
## 155 2018-12-02   655
## 156 2018-12-03   677
## 157 2018-12-04   666
## 158 2018-12-05   660
## 159 2018-12-06   645
## 160 2018-12-07   672
## 161 2018-12-08   622
```

```
## 162 2018-12-09   659
## 163 2018-12-10   664
## 164 2018-12-11   686
## 165 2018-12-12   624
## 166 2018-12-13   668
## 167 2018-12-14   697
## 168 2018-12-15   671
## 169 2018-12-16   709
## 170 2018-12-17   729
## 171 2018-12-18   799
## 172 2018-12-19   839
## 173 2018-12-20   808
## 174 2018-12-21   781
## 175 2018-12-22   840
## 176 2018-12-23   853
## 177 2018-12-24   865
## 178 2018-12-26   700
## 179 2018-12-27   690
## 180 2018-12-28   669
## 181 2018-12-29   666
## 182 2018-12-30   686
## 183 2018-12-31   650
## 184 2019-01-01   634
## 185 2019-01-02   674
## 186 2019-01-03   637
## 187 2019-01-04   704
## 188 2019-01-05   636
## 189 2019-01-06   673
## 190 2019-01-07   668
## 191 2019-01-08   669
## 192 2019-01-09   686
## 193 2019-01-10   685
## 194 2019-01-11   631
## 195 2019-01-12   687
## 196 2019-01-13   628
## 197 2019-01-14   663
## 198 2019-01-15   657
## 199 2019-01-16   674
## 200 2019-01-17   677
## 201 2019-01-18   658
## 202 2019-01-19   696
## 203 2019-01-20   683
## 204 2019-01-21   637
## 205 2019-01-22   689
## 206 2019-01-23   647
## 207 2019-01-24   619
## 208 2019-01-25   671
## 209 2019-01-26   672
## 210 2019-01-27   648
## 211 2019-01-28   661
## 212 2019-01-29   667
## 213 2019-01-30   689
## 214 2019-01-31   690
## 215 2019-02-01   708
```

```
## 216 2019-02-02   692
## 217 2019-02-03   690
## 218 2019-02-04   659
## 219 2019-02-05   691
## 220 2019-02-06   666
## 221 2019-02-07   663
## 222 2019-02-08   714
## 223 2019-02-09   671
## 224 2019-02-10   697
## 225 2019-02-11   683
## 226 2019-02-12   684
## 227 2019-02-13   693
## 228 2019-02-14   664
## 229 2019-02-15   667
## 230 2019-02-16   670
## 231 2019-02-17   682
## 232 2019-02-18   619
## 233 2019-02-19   664
## 234 2019-02-20   695
## 235 2019-02-21   646
## 236 2019-02-22   692
## 237 2019-02-23   689
## 238 2019-02-24   682
## 239 2019-02-25   693
## 240 2019-02-26   662
## 241 2019-02-27   687
## 242 2019-02-28   682
## 243 2019-03-01   670
## 244 2019-03-02   677
## 245 2019-03-03   674
## 246 2019-03-04   670
## 247 2019-03-05   724
## 248 2019-03-06   661
## 249 2019-03-07   658
## 250 2019-03-08   701
## 251 2019-03-09   637
## 252 2019-03-10   651
## 253 2019-03-11   690
## 254 2019-03-12   711
## 255 2019-03-13   702
## 256 2019-03-14   640
## 257 2019-03-15   724
## 258 2019-03-16   664
## 259 2019-03-17   715
## 260 2019-03-18   644
## 261 2019-03-19   688
## 262 2019-03-20   692
## 263 2019-03-21   672
## 264 2019-03-22   725
## 265 2019-03-23   680
## 266 2019-03-24   710
## 267 2019-03-25   688
## 268 2019-03-26   689
## 269 2019-03-27   660
```

```
## 270 2019-03-28    677
## 271 2019-03-29    699
## 272 2019-03-30    684
## 273 2019-03-31    647
## 274 2019-04-01    635
## 275 2019-04-02    700
## 276 2019-04-03    718
## 277 2019-04-04    709
## 278 2019-04-05    664
## 279 2019-04-06    681
## 280 2019-04-07    640
## 281 2019-04-08    650
## 282 2019-04-09    646
## 283 2019-04-10    699
## 284 2019-04-11    632
## 285 2019-04-12    672
## 286 2019-04-13    664
## 287 2019-04-14    685
## 288 2019-04-15    651
## 289 2019-04-16    674
## 290 2019-04-17    649
## 291 2019-04-18    667
## 292 2019-04-19    655
## 293 2019-04-20    738
## 294 2019-04-21    710
## 295 2019-04-22    671
## 296 2019-04-23    663
## 297 2019-04-24    702
## 298 2019-04-25    701
## 299 2019-04-26    684
## 300 2019-04-27    667
## 301 2019-04-28    677
## 302 2019-04-29    697
## 303 2019-04-30    680
## 304 2019-05-01    643
## 305 2019-05-02    667
## 306 2019-05-03    657
## 307 2019-05-04    630
## 308 2019-05-05    680
## 309 2019-05-06    707
## 310 2019-05-07    667
## 311 2019-05-08    698
## 312 2019-05-09    668
## 313 2019-05-10    665
## 314 2019-05-11    679
## 315 2019-05-12    687
## 316 2019-05-13    638
## 317 2019-05-14    705
## 318 2019-05-15    632
## 319 2019-05-16    664
## 320 2019-05-17    652
## 321 2019-05-18    626
## 322 2019-05-19    730
## 323 2019-05-20    707
```

```
## 324 2019-05-21   671
## 325 2019-05-22   687
## 326 2019-05-23   633
## 327 2019-05-24   691
## 328 2019-05-25   705
## 329 2019-05-26   648
## 330 2019-05-27   665
## 331 2019-05-28   683
## 332 2019-05-29   714
## 333 2019-05-30   669
## 334 2019-05-31   664
## 335 2019-06-01   682
## 336 2019-06-02   662
## 337 2019-06-03   656
## 338 2019-06-04   637
## 339 2019-06-05   680
## 340 2019-06-06   700
## 341 2019-06-07   762
## 342 2019-06-08   699
## 343 2019-06-09   718
## 344 2019-06-10   676
## 345 2019-06-11   634
## 346 2019-06-12   709
## 347 2019-06-13   607
## 348 2019-06-14   743
## 349 2019-06-15   724
## 350 2019-06-16   690
## 351 2019-06-17   658
## 352 2019-06-18   639
## 353 2019-06-19   662
## 354 2019-06-20   698
## 355 2019-06-21   716
## 356 2019-06-22   643
## 357 2019-06-23   653
## 358 2019-06-24   612
## 359 2019-06-25   696
## 360 2019-06-26   657
## 361 2019-06-27   669
## 362 2019-06-28   673
## 363 2019-06-29   703
## 364 2019-06-30   704
```

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.

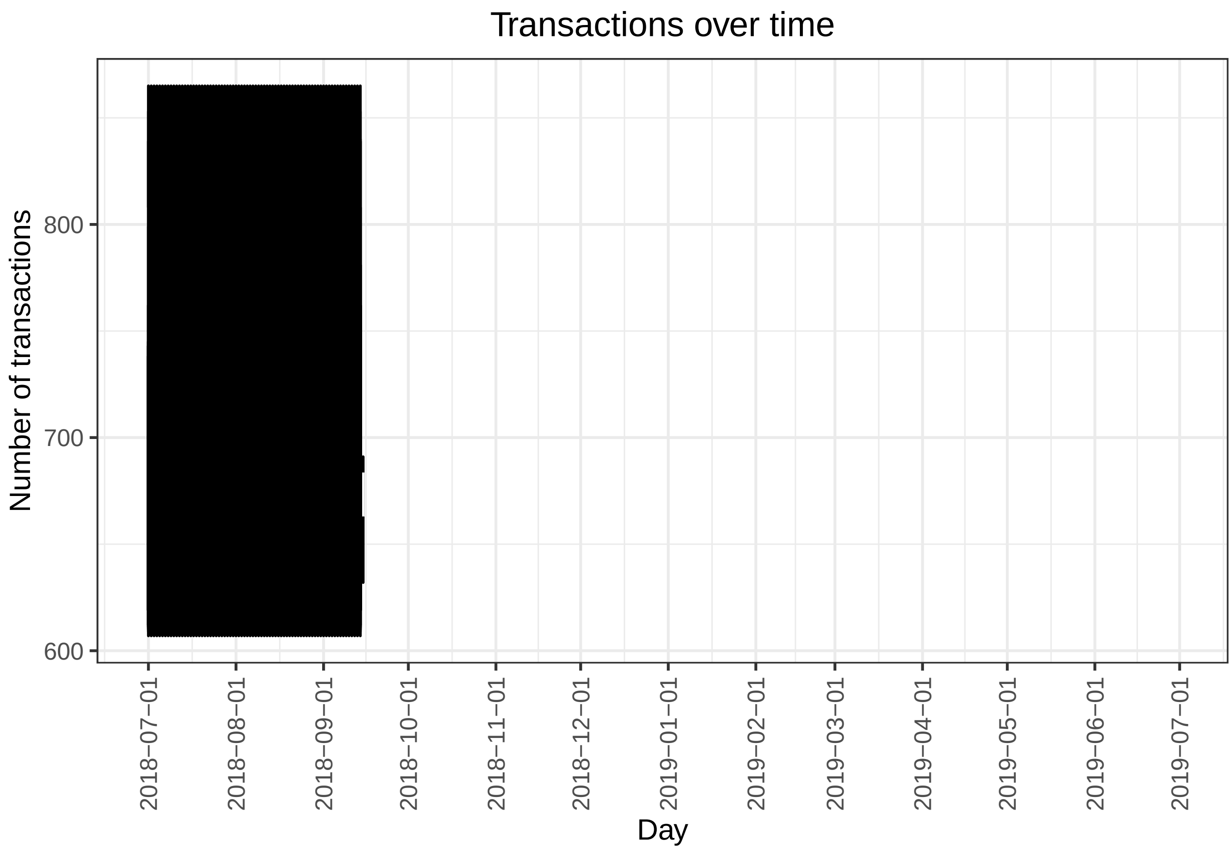
```r
#### Create a sequence of dates and join this the count of transactions by date
# Over to you - create a column of dates that includes every day from 1 Jul 2018 to 30 Jun 2019, and jo
# Generate a sequence of dates from 1 Jul 2018 to 30 Jun 2019
date_seq <- seq(from = as.Date("2018-07-01"), to = as.Date("2019-06-30"), by = "day")

# Create a data frame with the date sequence
date_df <- data.frame(DATE = date_seq)

# Join the date_df to transaction_by_date to fill in the missing days
```

```r
transactions_by_day <- merge(date_df, transactions_by_date, all = TRUE)
```

```r
library(dplyr)

# Rename the "date" column to "transaction_date" in transaction_by_day
transactions_by_day <- transactions_by_day %>% rename(transaction_date = date)

# Merge transaction_by_day with date_df to fill in the missing dates
transactions_by_day_filled <- merge(date_df, transactions_by_day, all.x = TRUE)

# Print the summary of transaction count by date with missing days filled in
head(transactions_by_day_filled)
```

```
##         DATE transaction_date count
## 1 2018-07-01       2018-07-01   663
## 2 2018-07-01       2018-07-02   650
## 3 2018-07-01       2018-07-04   669
## 4 2018-07-01       2018-07-05   660
## 5 2018-07-01       2018-07-07   695
## 6 2018-07-01       2018-07-10   650
```

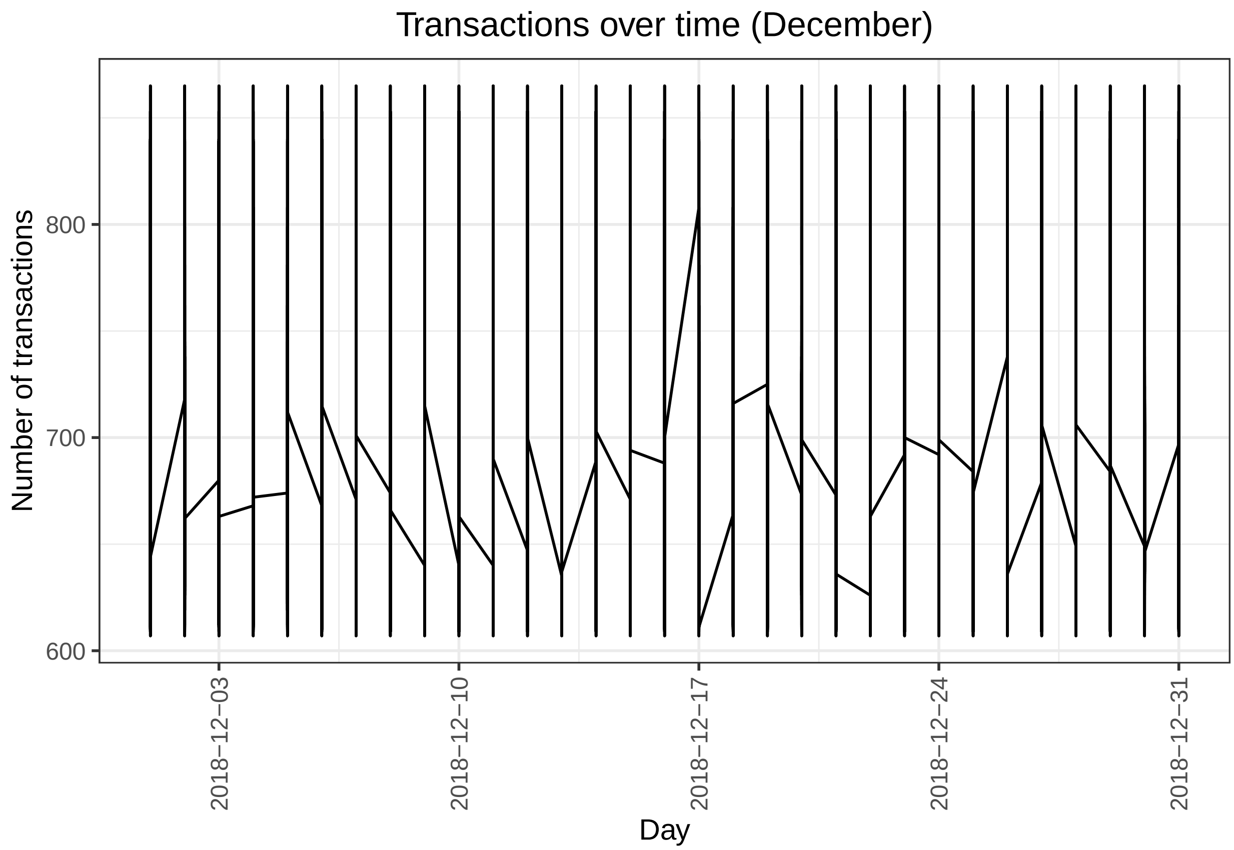
```r
transactions_by_day_filled <- transactions_by_day_filled %>% select(-transaction_date)


#### 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_filled, 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.
# Set the plot themes
theme_set(theme_bw())
theme_update(plot.title = element_text(hjust = 0.5))

# Plot transactions over time, zoomed in to December
ggplot(transactions_by_day_filled, aes(x = DATE, y = count)) +
  geom_line() +
  labs(x = "Day", y = "Number of transactions", title = "Transactions over time (December)") +
  scale_x_date(breaks = "1 week", limits = as.Date(c("2018-12-01", "2018-12-31"))) +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5))
```

```
## Warning: Removed 121576 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.

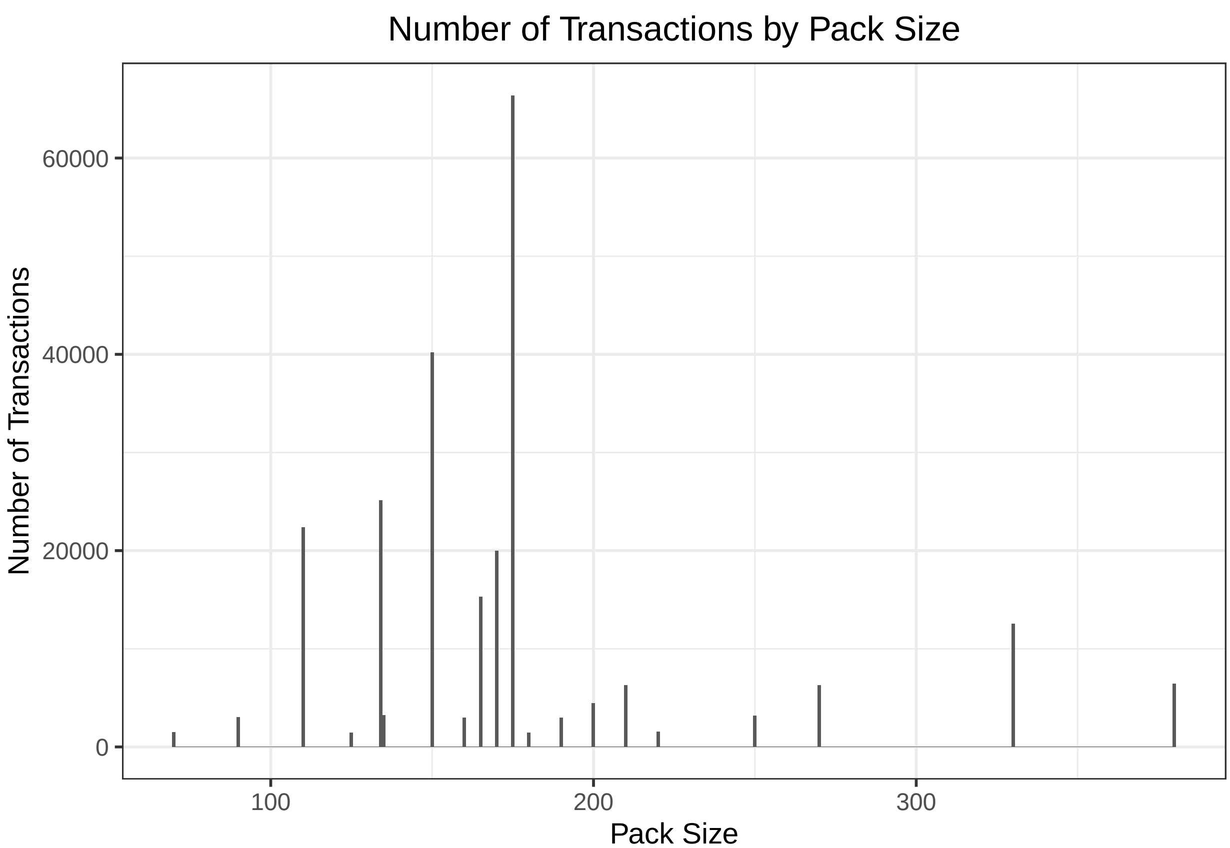
```
#### Pack size
#### We can work this out by taking the digits that are in PROD_NAME
transactionData[, PACK_SIZE := parse_number(PROD_NAME)]
#### Always check your output
#### Let's check if the pack sizes look sensible
transactionData[, .N, PACK_SIZE][order(PACK_SIZE)]
```

```
##     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 cont
# Over to you! Plot a histogram showing the number of transactions by pack size.
library(ggplot2)

# Plot histogram of transactions by pack size
ggplot(transactionData, aes(x = PACK_SIZE)) +
  geom_histogram(binwidth = 1) +
  labs(x = "Pack Size", y = "Number of Transactions", title = "Number of Transactions by Pack Size")
```



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 produ
#### Checking brands
# Over to you! Check the results look reasonable.
### Brands
library(stringr)

transactionData$brand_name <- str_to_title(word(transactionData$PROD_NAME, 1))
```

```r
# view the first 10 rows
head(transactionData$brand_name, 10)
```

```
##  [1] "Natural" "Ccs"     "Smiths"  "Smiths"  "Kettle"  "Smiths"  "Grain"
##  [8] "Doritos" "Grain"   "Smiths"
```

```r
unique(transactionData$brand_name)
```

```
##  [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_name == "Red", brand_name := "Rrd"]
transactionData[brand_name == "Dorito", brand_name := "Doritos"]
# Over to you! Add any additional brand adjustments you think may be required.
#### Check again
# Over to you! Check the results look reasonable.
```

Examining customer data

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

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

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

```r
tibble(customerData)
```

```
## # A tibble: 72,637 x 3
##    LYLTY_CARD_NBR LIFESTAGE              PREMIUM_CUSTOMER
##             <int> <chr>                  <chr>
##  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
##  7           1009 NEW FAMILIES           Premium
##  8           1010 YOUNG SINGLES/COUPLES  Mainstream
##  9           1011 OLDER SINGLES/COUPLES  Mainstream
## 10           1012 OLDER FAMILIES         Mainstream
## # i 72,627 more rows
```

```
# Group the data by words and count the frequency of occurrence
word_freq <- customerData %>%
  group_by(LIFESTAGE) %>%
  summarize(count = n())
# Sort the data in descending order of frequency
word_freq <- word_freq %>%
  arrange(desc(count))
# View the top 10 most frequent words
head(word_freq, 10)
```

```
## # A tibble: 7 x 2
##   LIFESTAGE              count
##   <chr>                  <int>
## 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
```

```
# Group the data by words and count the frequency of occurrence
word_freq <- customerData %>%
  group_by(PREMIUM_CUSTOMER) %>%
  summarize(count = n())
# Sort the data in descending order of frequency
word_freq <- word_freq %>%
  arrange(desc(count))
# View the top 10 most frequent words
head(word_freq, 10)
```

```
## # A tibble: 3 x 2
##   PREMIUM_CUSTOMER count
##   <chr>            <int>
## 1 Mainstream       29245
## 2 Budget           24470
## 3 Premium          18922
```

```
#### 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. Page 7 20200128_InsideSherpa_Task1_DraftSolutions - Template (1).Rmd 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.
# Find transactions without a matched customer
unmatched_transactions <- anti_join(transactionData, customerData, by = "LYLTY_CARD_NBR")

# Check if there are any unmatched transactions
if (nrow(unmatched_transactions) == 0) {
  cat("All transactions have a matched customer.\n")
} else {
```

```
  cat(paste0(nrow(unmatched_transactions), " transactions did not have a matched customer.\n"))
}
```

```
## All transactions have a matched customer.
```

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.
sales_summary <- data %>%
  group_by(LIFESTAGE, PREMIUM_CUSTOMER) %>%
  summarise(total_sales = sum(TOT_SALES)) %>%
  arrange(desc(total_sales))
```

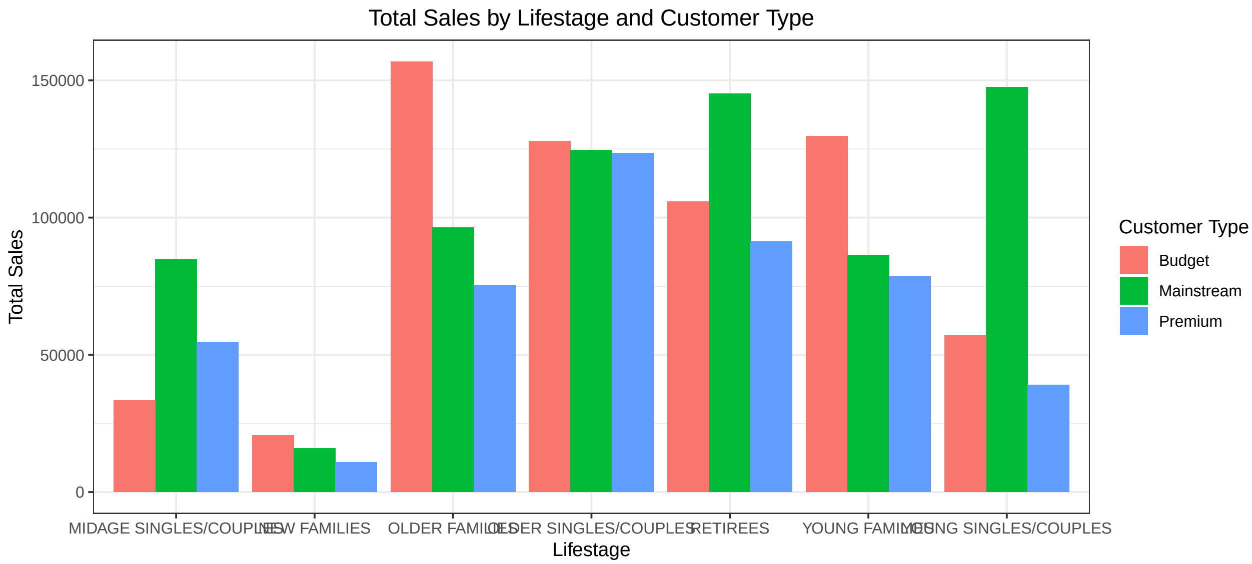
```
## `summarise()` has grouped output by 'LIFESTAGE'. You can override using the
## `.groups` argument.
```

```
# View the summary
sales_summary
```

```
## # A tibble: 21 x 3
## # Groups:   LIFESTAGE [7]
##    LIFESTAGE              PREMIUM_CUSTOMER total_sales
##    <chr>                  <chr>                  <dbl>
##  1 OLDER FAMILIES         Budget               156864.
##  2 YOUNG SINGLES/COUPLES Mainstream           147582.
##  3 RETIREES               Mainstream           145169.
##  4 YOUNG FAMILIES         Budget               129718.
##  5 OLDER SINGLES/COUPLES Budget               127834.
##  6 OLDER SINGLES/COUPLES Mainstream           124648.
##  7 OLDER SINGLES/COUPLES Premium              123538.
##  8 RETIREES               Budget               105916.
##  9 OLDER FAMILIES         Mainstream            96414.
## 10 RETIREES               Premium               91297.
## # i 11 more rows
```

```
library(ggplot2)

# Create the plot
ggplot(sales_summary, aes(x = LIFESTAGE, y = total_sales, fill = PREMIUM_CUSTOMER)) +
  geom_col(position = "dodge") +
  labs(title = "Total Sales by Lifestage and Customer Type",
```

```r
      x = "Lifestage",
      y = "Total Sales",
      fill = "Customer Type") +
  theme(plot.title = element_text(hjust = 0.5))
```



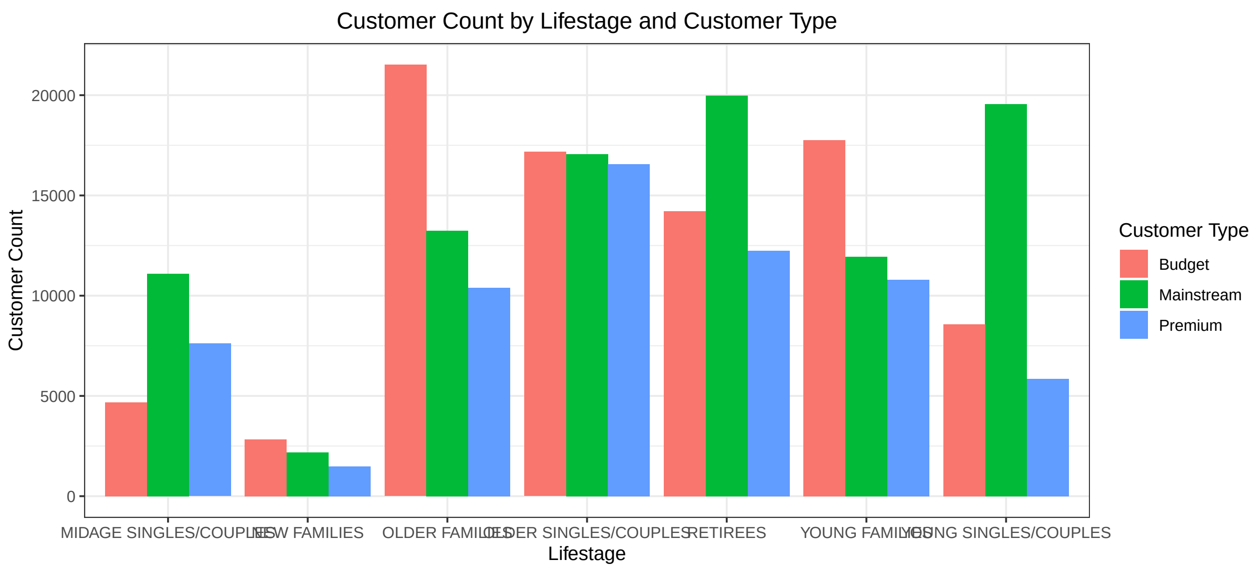
Page 8 20200128_InsideSherpa_Task1_DraftSolutions - Template (1).Rmd Sales are coming mainly from Budget - older families, Mainstream - young singles/couples, and Mainstream - retirees Let's see if the higher sales are due to there being more customers who buy chips.

```r
#### 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.
customer_count <- data %>%
  count(LIFESTAGE, PREMIUM_CUSTOMER, name = "customer_count") %>%
  arrange(desc(customer_count))

# View the results
customer_count
```

```
##                  LIFESTAGE PREMIUM_CUSTOMER customer_count
##  1:         OLDER FAMILIES           Budget          21514
##  2:               RETIREES       Mainstream          19970
##  3:  YOUNG SINGLES/COUPLES       Mainstream          19544
##  4:         YOUNG FAMILIES           Budget          17763
##  5:  OLDER SINGLES/COUPLES           Budget          17172
##  6:  OLDER SINGLES/COUPLES       Mainstream          17061
##  7:  OLDER SINGLES/COUPLES          Premium          16560
##  8:               RETIREES           Budget          14225
##  9:         OLDER FAMILIES       Mainstream          13241
## 10:               RETIREES          Premium          12236
## 11:         YOUNG FAMILIES       Mainstream          11947
## 12: MIDAGE SINGLES/COUPLES       Mainstream          11095
## 13:         YOUNG FAMILIES          Premium          10784
## 14:         OLDER FAMILIES          Premium          10403
## 15:  YOUNG SINGLES/COUPLES           Budget           8573
## 16: MIDAGE SINGLES/COUPLES          Premium           7612
## 17:  YOUNG SINGLES/COUPLES          Premium           5852
```

```
## 18: MIDAGE SINGLES/COUPLES          Budget           4691
## 19:           NEW FAMILIES          Budget           2824
## 20:           NEW FAMILIES      Mainstream           2185
## 21:           NEW FAMILIES         Premium           1488
##                  LIFESTAGE PREMIUM_CUSTOMER customer_count
```

```r
library(ggplot2)

# Create the plot
ggplot(customer_count, aes(x = LIFESTAGE, y = customer_count, fill = PREMIUM_CUSTOMER)) +
  geom_col(position = "dodge") +
  labs(title = "Customer Count by Lifestage and Customer Type",
       x = "Lifestage",
       y = "Customer Count",
       fill = "Customer Type") +
  theme(plot.title = element_text(hjust = 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.
prod_qty_avg <- data %>%
  group_by(LIFESTAGE, PREMIUM_CUSTOMER) %>%
  summarize(avg_qty = mean(PROD_QTY))
```

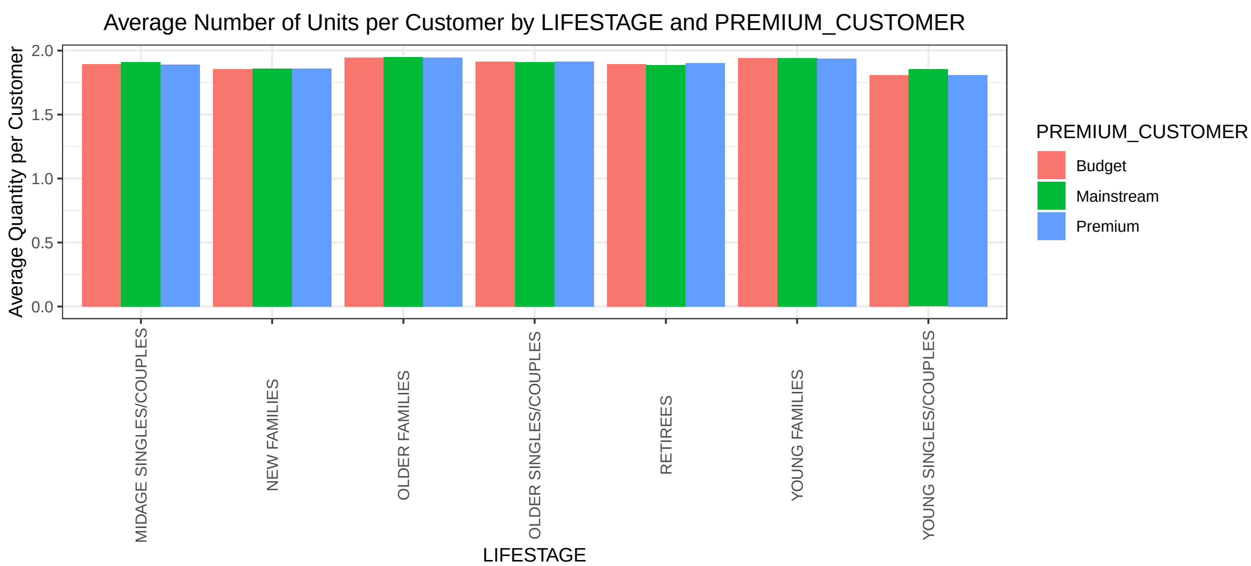
```
## `summarise()` has grouped output by 'LIFESTAGE'. You can override using the
## `.groups` argument.
```

```r
# View the summary
prod_qty_avg
```

```
## # A tibble: 21 x 3
## # Groups:   LIFESTAGE [7]
##    LIFESTAGE              PREMIUM_CUSTOMER avg_qty
```

```
##    <chr>                   <chr>              <dbl>
##  1 MIDAGE SINGLES/COUPLES Budget              1.89
##  2 MIDAGE SINGLES/COUPLES Mainstream          1.91
##  3 MIDAGE SINGLES/COUPLES Premium             1.89
##  4 NEW FAMILIES           Budget              1.86
##  5 NEW FAMILIES           Mainstream          1.86
##  6 NEW FAMILIES           Premium             1.86
##  7 OLDER FAMILIES         Budget              1.95
##  8 OLDER FAMILIES         Mainstream          1.95
##  9 OLDER FAMILIES         Premium             1.95
## 10 OLDER SINGLES/COUPLES  Budget              1.91
## # i 11 more rows
```

```r
library(ggplot2)

# Create the plot
ggplot(prod_qty_avg, aes(x = LIFESTAGE, y = avg_qty, fill = PREMIUM_CUSTOMER)) +
  geom_col(position = "dodge") +
  labs(x = "LIFESTAGE", y = "Average Quantity per Customer",
       title = "Average Number of Units per Customer by LIFESTAGE and PREMIUM_CUSTOMER") +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5))
```



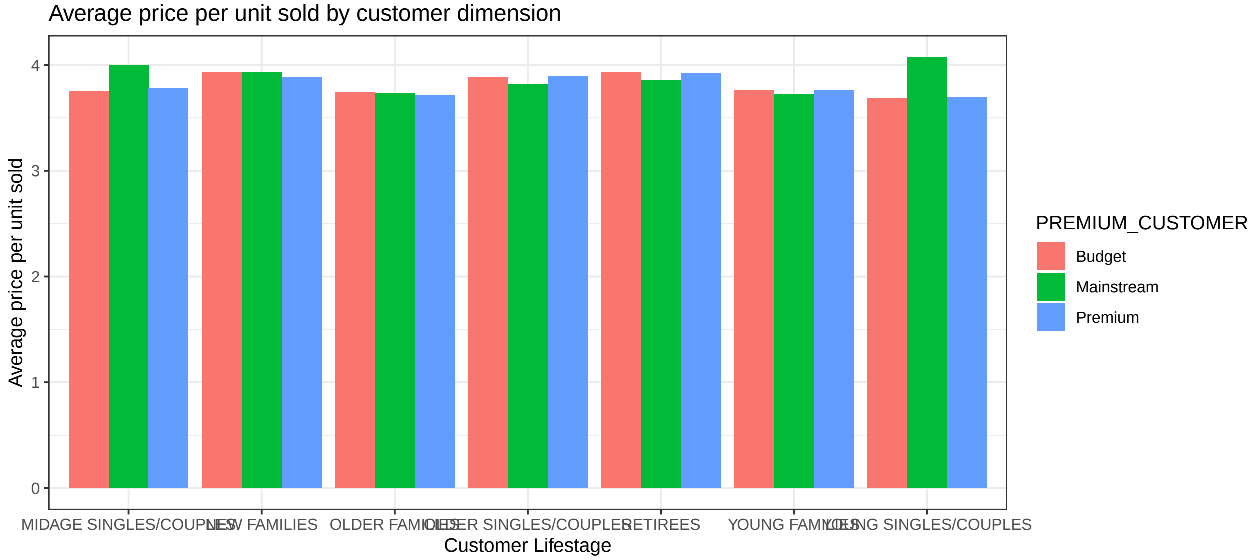
Older families and young families in general buy more chips per customer Let's also investigate the average price per unit chips bought for each customer segment as this is also a driver of total sales.

```r
#### Average price per unit by LIFESTAGE and PREMIUM_CUSTOMER
# Over to you! Calculate and plot the average price per unit sold (average sale price) by those two cus
price_per_unit <- data %>%
  group_by(LIFESTAGE, PREMIUM_CUSTOMER) %>%
  summarize(avg_price_per_unit = sum(TOT_SALES) / sum(PROD_QTY))
```

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

```r
# Plot average price per unit sold
ggplot(price_per_unit, aes(x = LIFESTAGE, y = avg_price_per_unit, fill = PREMIUM_CUSTOMER)) +
  geom_bar(stat = "identity", position = "dodge") +
```

```
  labs(x = "Customer Lifestage", y = "Average price per unit sold", title = "Average price per unit sold
  theme_bw()
```



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 Page 9 20200128_InsideSherpa_Task1_DraftSolutions - Template (1).Rmd 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.
#### Perform an independent t-test between mainstream vs premium and budget midage and
#### young singles and couples


# Filter the relevant data

t_test_data <- price_per_unit %>%
  filter((LIFESTAGE %in% c("MIDAGE SINGLES/COUPLES", "YOUNG SINGLES/COUPLES")) &
           (PREMIUM_CUSTOMER %in% c("Budget", "Premium")) |
           (LIFESTAGE %in% c("MIDAGE SINGLES/COUPLES", "YOUNG SINGLES/COUPLES")) &
           (PREMIUM_CUSTOMER == "Mainstream"))

group1 <- t_test_data %>%
  filter(LIFESTAGE %in% c("MIDAGE SINGLES/COUPLES", "YOUNG SINGLES/COUPLES") &
         PREMIUM_CUSTOMER == "Mainstream") %>%
  pull(avg_price_per_unit)

group2 <- t_test_data %>%
  filter(LIFESTAGE %in% c("MIDAGE SINGLES/COUPLES", "YOUNG SINGLES/COUPLES") &
         PREMIUM_CUSTOMER %in% c("Budget", "Premium")) %>%
  pull(avg_price_per_unit)
```

```r
# Perform independent t-test
t_test_result <- t.test(group1, group2)

# View the test results
t_test_result
```

```
##
##  Welch Two Sample t-test
##
## data:  group1 and group2
## t = 6.6358, df = 1.7353, p-value = 0.03113
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  0.07556858 0.53647998
## sample estimates:
## mean of x mean of y
##  4.034246  3.728222
```

The t-test results in a p-value of XXXXXXX, i.e. the unit price for mainstream, young and mid-age singles and couples [ARE / ARE NOT] 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. Y
#install.packages("arules")
#install.packages("tidyverse")
#library(tidyverse)
#library(arules)
#library(dplyr)

# Create a subset of the data with only Mainstream and Young Singles/Couples customers
#customer_subset <- data %>%
 ## filter(LIFESTAGE %in% c("YOUNG SINGLES/COUPLES", "MIDAGE SINGLES/COUPLES") & PREMIUM_CUSTOMER == "M

# Create a transaction matrix
#transactions <- customer_subset %>%
 # select(LYLTY_CARD_NBR, brand_name) %>%
#  distinct() %>%
 # group_by(LYLTY_CARD_NBR) %>%
#  summarize(items = list(brand_name))

# Convert the transaction matrix to a binary matrix
#binary_matrix <- as.data.frame.matrix(table(transactions$LYLTY_CARD_NBR, #unlist(transactions$items)))


# Convert binary_matrix to transactions
#transactions <- as(transpose(binary_matrix), "transactions")

# Run the apriori algorithm with a minimum support of 0.01 and confidence of 0.5
#rules <- apriori(transactions, parameter = list(supp = 0.01, conf = 0.5))
```

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

```r
#### Preferred pack size compared to the rest of the population
# Over to you! Do the same for pack size.
```