Base Pricing Analysis and Price Elasticity Estimation

Giovanni Compiani

Contents

1 Overview

The goal is to conduct a base pricing analysis. We estimate brand-level demand using scanner data, and then we make profitability predictions corresponding to specific base price changes. We estimate log-linear demand models that use (log) prices and promotions as inputs, and predict log quantities, `log(1+Q)`. The models predict the demand for a focal brand, and we control for (log) prices and promotions of three competitors. Obviously, this approach generalizes to an arbitrarily large number of competing products as long as the sample size is large enough.

Our focus is on the two top brands in the liquid laundry detergent category, *Tide* and *Gain*. Both are Procter & Gamble brands. The two closest competitors are *Arm & Hammer* and *Purex*.

2 Packages

Make sure to install two packages that we have not used before: fixest and knitr.

```r
library(bit64)
library(data.table)
library(fixest)
library(knitr)
library(ggplot2)
```

3 Data overview

The data are located in this folder:

```r
data_folder = "/Users/nichada/MyCode/MPCS/_BUSN_DataSci/DataSci_Mkt/hw2/data"
#data_folder = "Data"
```

The data source is an extract from the Nielsen RMS retail scanner data set. The data set captures weekly price and quantity data for all products (UPC's) sold in the stores of a large number of U.S. retail chains. The Kilts data do not include all retailers (for example, Walmart is not part of the data), and the identity of the retailers is not revealed. However, we know if two stores belong to the same retail chain.

3.1 Brand data

The data.table `brands` in `Brands.RData` includes brand information for the top five brands in three categories (product modules):

```
1036   FRUIT JUICE - LEMON/LIME
1040   FRUIT JUICE - ORANGE - OTHER CONTAINER
7012   DETERGENTS - HEAVY DUTY - LIQUID
```

The data include the brand code, brand description, and total revenue calculated across all observations. The top five brands were selected based on total brand revenue.

We will focus on the liquid laundry detergent category with corresponding `product_module_code` 7012.

3.2 Store data

Inspect the table `stores` in the file `Stores.RData`. The variable `store_code_uc` identifies each retail stores. For some (but not all) stores we know the corresponding `retailer_code` that identifies the chain (banner) that the store belongs to. The data include the Scantrack (SMM) market code and the Scantrack market description. Scantrack markets correspond to large metropolitan market areas such as *Chicago* or *Raleigh-Durham* (see the data manual for a map of the Scantrack markets). The three-digit ZIP code of each store is also included.

3.3 Movement data

The movement data (`move`) are in files of the form `brand_move_<module code>.RData`. The data are at the brand/store/week level and include prices and quantities (`units`). The data are aggregates of all UPC's that share the same brand name. Brand prices are measured as the weighted average over all store/week UPC prices in equivalent units, and quantities represent total product volume measured in equivalent units such as ounces. In the liquid laundry detergent category (module 7012), prices represent dollars per ounce and units are total product volume in ounces per store/week. The aggregation weights are based on total store-level UPC revenue across all weeks, and hence the aggregation weights are constant within each store. The movement data also include a promotion indicator (`promo_dummy`), a logical `TRUE/FALSE` variable.

The `week_end` variable date is the last day of a Nielsen week, which always starts on a Sunday and ends on a Saturday. Note that prices may change during the period, and hence even the UPC-level price may be an average over more than one posted price. The sample includes data for the 2010-2013 period.

Please consult the official Kilts Center Retail Scanner Dataset Manual for all details.

4 Prepare the data for the demand analysis

We first load the brand and store data.

```
load(paste0(data_folder, "/Brands.RData"))
load(paste0(data_folder, "/Stores.RData"))
```

4.1 Select the category and brands

*Choose the laundry detergent category (module) and select the corresponding brand-level meta data from the data table `brands`. Then sort (order) the brand data corresponding to total brand revenue, and select the **top four brands** (ranked by revenue).

```
selected_module = 7012                 # Laundry detergent
laundry_brands = brands[product_module_code == selected_module]

sorted_brands = laundry_brands[order(-laundry_brands$revenue), ]

top_four_brands = head(sorted_brands, 4)

top_four_brands
```

```
   brand_code_uc            brand_descr             product_module_descr
           <int>                 <char>                           <char>
1:        653791         TIDE - H-D LIQ DETERGENTS - HEAVY DUTY - LIQUID
2:        557775         GAIN - H-D LIQ DETERGENTS - HEAVY DUTY - LIQUID
3:        507562 ARM & HAMMER - H-D LIQ DETERGENTS - HEAVY DUTY - LIQUID
4:        623280        PUREX - H-D LIQ DETERGENTS - HEAVY DUTY - LIQUID
   product_module_code    revenue
                 <int>      <num>
1:                7012 3659669291
2:                7012 1201306647
3:                7012 1010503850
4:                7012  495632613
```

Let's assign each brand a new name using a new variable, `brand_name`, and give the four brands simple names such as `Tide`, `Gain`, `ArmHammer`, and `Purex`. These simplified brand names will make our code and the estimation output more readable. More specifically, create a new data containing the four selected brands and add to it the `brand_name` variable.

Note that we will add the brand names to the quantity, price, and promotion variables. In R, `price_ArmHammer` (as well as `price_Arm_Hammer`) are legal variable names, but `price_Arm&Hammer` and `price_Arm & Hammer` are not, and hence I do not suggest the brand names `Arm&Hammer` or `Arm & Hammer`.

```
top_four_brands$brand_name = NA

top_four_brands$brand_name[top_four_brands$brand_descr == "TIDE - H-D LIQ"] = "Tide"
top_four_brands$brand_name[top_four_brands$brand_descr == "GAIN - H-D LIQ"] = "Gain"
top_four_brands$brand_name[top_four_brands$brand_descr == "ARM & HAMMER - H-D LIQ"] = "ArmHammer"
top_four_brands$brand_name[top_four_brands$brand_descr == "PUREX - H-D LIQ"] = "Purex"

top_four_brands
```

```
   brand_code_uc            brand_descr             product_module_descr
           <int>                 <char>                           <char>
1:        653791         TIDE - H-D LIQ DETERGENTS - HEAVY DUTY - LIQUID
2:        557775         GAIN - H-D LIQ DETERGENTS - HEAVY DUTY - LIQUID
```

```
3:        507562 ARM & HAMMER - H-D LIQ DETERGENTS - HEAVY DUTY - LIQUID
4:        623280        PUREX - H-D LIQ DETERGENTS - HEAVY DUTY - LIQUID
   product_module_code    revenue brand_name
                 <int>      <num>     <char>
1:                7012 3659669291       Tide
2:                7012 1201306647       Gain
3:                7012 1010503850  ArmHammer
4:                7012  495632613      Purex
```

4.2 Prepare the movement data

*Load the movement data, and—for better readability—change the variable names from `units` to `quantity` and from `promo_dummy` to `promotion` (you can use the `setnames` command for this). Change the data type of the `promotion` variable from `logical` to `numeric` using the `as.numeric` function. Finally, merge the new `brand_name` variable with the movement table (more precisely, perform an inner join, i.e. retain all observations that are present in both the parent and child data sets).

```r
load(paste0(data_folder, "/brand_move_7012.RData"))
setnames(move, old = c("units", "promo_dummy"), new = c("quantity", "promotion"))
move$promotion = as.numeric(move$promotion)
merged_data = merge(move, top_four_brands[,c("brand_code_uc","brand_name")], by = "brand_code_uc")
merged_data
```

```
Key: <brand_code_uc>
         brand_code_uc store_code_uc   week_end      price quantity promotion
                 <int>         <int>     <Date>      <num>    <num>     <num>
      1:        507562          1123 2010-01-02 0.06528157   2000.0         1
      2:        507562          1123 2010-01-09 0.06852484   3300.0         1
      3:        507562          1123 2010-01-16 0.07700768   1500.0         0
      4:        507562          1123 2010-01-23 0.07757468   1275.0         0
      5:        507562          1123 2010-01-30 0.06442845   4025.0         1
     ---
5256704:        653791       8386077 2013-11-30 0.17189507   1000.0         0
5256705:        653791       8386077 2013-12-07 0.13699283   6000.0         1
5256706:        653791       8386077 2013-12-14 0.17348442    650.0         0
5256707:        653791       8386077 2013-12-21 0.17500879    900.0         0
5256708:        653791       8386077 2013-12-28 0.16948965   1404.8         0
         brand_name
             <char>
      1:  ArmHammer
      2:  ArmHammer
      3:  ArmHammer
      4:  ArmHammer
      5:  ArmHammer
     ---
5256704:       Tide
5256705:       Tide
5256706:       Tide
5256707:       Tide
5256708:       Tide
```

4.3 Remove outliers

Most data contain some "flaws" or outliers. Here is an easy way of removing such outliers:

First, we create a function that flags all observations in a vector `x`, for example a price series, as outliers if the ratio between a value and the median value among all `x` observations is below or above a threshold.

```
isOutlier <- function(x, threshold_bottom, threshold_top) {
   is_outlier = rep(FALSE, times = length(x))
   median_x   = median(x, na.rm = TRUE)
   is_outlier[x/median_x < threshold_bottom | x/median_x > threshold_top] = TRUE
   return(is_outlier)
}
```

Now run this function on the price data, separately for each brand and store. Then tabulate the number of outliers, and remove the corresponding observations from the data set.

I recommend to use a lower threshold (`threshold_bottom`) value of 0.35 and an upper threshold (`threshold_top`) of 2.5.

```
threshold_bottom = 0.35
threshold_top = 2.5

## not sure if needed:
## merged_data_means = merged_data[, lapply(.SD, mean),
##                                   by = .(brand_code_uc, store_code_uc),
##                                   .SDcols = c("price")]

merged_data[,isoutlier:=isOutlier(price,threshold_bottom,threshold_top), by = .(brand_code_uc, store_coo

# tabulate data
outlier_counts <- merged_data[isoutlier == TRUE, .(outlier_count = .N), by = .(brand_code_uc, brand_name
print(outlier_counts)
```

```
      brand_code_uc brand_name store_code_uc outlier_count
              <int>     <char>         <int>         <int>
   1:        507562  ArmHammer         16410             5
   2:        507562  ArmHammer         24470             7
   3:        507562  ArmHammer         38444             2
   4:        507562  ArmHammer         39046             1
   5:        507562  ArmHammer         63281             3
  ---
4504:        653791       Tide       3244237             1
4505:        653791       Tide       4100240             6
4506:        653791       Tide       4616166             2
4507:        653791       Tide       4616981             9
4508:        653791       Tide       6705250             1
```

```
merged_data_cleaned = merged_data[isoutlier=="FALSE"]

merged_data_cleaned
```

```
Key: <brand_code_uc>
         brand_code_uc store_code_uc   week_end      price quantity promotion
                 <int>         <int>     <Date>      <num>    <num>     <num>
      1:        507562          1123 2010-01-02 0.06528157   2000.0         1
      2:        507562          1123 2010-01-09 0.06852484   3300.0         1
      3:        507562          1123 2010-01-16 0.07700768   1500.0         0
      4:        507562          1123 2010-01-23 0.07757468   1275.0         0
      5:        507562          1123 2010-01-30 0.06442845   4025.0         1
     ---
5239436:        653791       8386077 2013-11-30 0.17189507   1000.0         0
```

```
5239437:        653791       8386077 2013-12-07 0.13699283   6000.0          1
5239438:        653791       8386077 2013-12-14 0.17348442    650.0          0
5239439:        653791       8386077 2013-12-21 0.17500879    900.0          0
5239440:        653791       8386077 2013-12-28 0.16948965   1404.8          0
         brand_name isoutlier
             <char>    <lgcl>
      1:  ArmHammer     FALSE
      2:  ArmHammer     FALSE
      3:  ArmHammer     FALSE
      4:  ArmHammer     FALSE
      5:  ArmHammer     FALSE
     ---
5239436:       Tide     FALSE
5239437:       Tide     FALSE
5239438:       Tide     FALSE
5239439:       Tide     FALSE
5239440:       Tide     FALSE
```

4.4 Reshape the movement data from long to wide format

To prepare the data for the regression analysis, we need to **reshape the data from long to wide format** using **dcast**.

All the details on casting and the reverse operation (melting from wide to long format using `melt`) are explained in the data.table html vignettes:

https://rdatatable.gitlab.io/data.table/articles/datatable-reshape.html

Let's be specific about the structure of the data that we need to use to estimate a demand model. We would like to obtain a table with observations, characterized by a combination of store id (`store_code_uc`) and week (`week_end`) in rows, and information on quantities, prices, and promotions in columns. Quantities, prices, and promotions are brand-specific.

```r
# Step 1: Melt the data into long format
melted_data <- melt(merged_data_cleaned,
                    id.vars = c("store_code_uc", "week_end", "brand_name"),
                    measure.vars = c("price", "quantity", "promotion"))

# Step 2: Use dcast() to reshape the data back to wide format
wide_data <- dcast(melted_data,
                   store_code_uc + week_end ~ variable + brand_name,
                   value.var = "value")

# View the result
head(wide_data)
```

```
Key: <store_code_uc, week_end>
   store_code_uc   week_end price_ArmHammer price_Gain price_Purex price_Tide
           <int>     <Date>           <num>      <num>       <num>      <num>
1:          1123 2010-01-02      0.06528157  0.1209109  0.08248828        NaN
2:          1123 2010-01-09      0.06852484  0.1209109  0.08248828  0.1299496
3:          1123 2010-01-16      0.07700768  0.1209109  0.06126852  0.1519637
4:          1123 2010-01-23      0.07757468  0.1015777  0.08214914  0.1524072
5:          1123 2010-01-30      0.06442845  0.1209109  0.08214914  0.1524981
6:          1123 2010-02-06      0.07757468  0.1209109  0.08214914  0.1519637
   quantity_ArmHammer quantity_Gain quantity_Purex quantity_Tide
```

```
                 <num>         <num>          <num>         <num>
1:                2000           400           1060          3000
2:                3300           700           1582          9600
3:                1500           600           4988          3450
4:                1275          1300            832          2700
5:                4025           300           1142          2100
6:                2275           300           1786          3300
   promotion_ArmHammer promotion_Gain promotion_Purex promotion_Tide
                 <num>          <num>           <num>          <num>
1:                   1              0               0             NA
2:                   1              0               0              1
3:                   0              0               1              0
4:                   0              1               0              0
5:                   1              0               0              0
6:                   0              0               0              0
```

4.5 Merge store information with the movement data

Now merge the movement data with the store meta data, in particular with the retailer code, the Scantrack (SMM) market code, and the Scantrack market description. But only with the store meta data where we have a valid retailer code. Hence, we need to remove store data if the retailer code is missing (NA). Use the `is.na` *function to check if* `retailer_code` *is NA or not.*

```r
# Merge movement data with store meta data
# Keep only stores with a non-NA retailer code
movement_merged <- merge(wide_data, stores[!is.na(retailer_code)],
                         by = "store_code_uc", all.x = TRUE)

# Display the merged data
head(movement_merged)
```

```
Key: <store_code_uc>
   store_code_uc   week_end price_ArmHammer price_Gain price_Purex price_Tide
           <int>     <Date>           <num>      <num>       <num>      <num>
1:          1123 2010-01-02      0.06528157  0.1209109  0.08248828        NaN
2:          1123 2010-01-09      0.06852484  0.1209109  0.08248828  0.1299496
3:          1123 2010-01-16      0.07700768  0.1209109  0.06126852  0.1519637
4:          1123 2010-01-23      0.07757468  0.1015777  0.08214914  0.1524072
5:          1123 2010-01-30      0.06442845  0.1209109  0.08214914  0.1524981
6:          1123 2010-02-06      0.07757468  0.1209109  0.08214914  0.1519637
   quantity_ArmHammer quantity_Gain quantity_Purex quantity_Tide
                 <num>         <num>          <num>         <num>
1:                2000           400           1060          3000
2:                3300           700           1582          9600
3:                1500           600           4988          3450
4:                1275          1300            832          2700
5:                4025           300           1142          2100
6:                2275           300           1786          3300
   promotion_ArmHammer promotion_Gain promotion_Purex promotion_Tide
                 <num>          <num>           <num>          <num>
1:                   1              0               0             NA
2:                   1              0               0              1
3:                   0              0               1              0
4:                   0              1               0              0
5:                   1              0               0              0
```

```
6:                  0             0              0              0
   retailer_code store_zip3 SMM_code SMM_description
           <int>      <int>    <int>          <char>
1:            89        441       16       Cleveland
2:            89        441       16       Cleveland
3:            89        441       16       Cleveland
4:            89        441       16       Cleveland
5:            89        441       16       Cleveland
6:            89        441       16       Cleveland
```

4.6 Create time variables or trends

A time trend records the progress of time. For example, a time trend at the week-level may equal 1 in the first week in the data, 2 in the second week, etc., whereas a trend at the month-level may equal 1 in the first month, 2 in the second month, etc.

I suggest you create a monthly time trend. Use the functions `year` *and* `month` *to extract the year and month components of the week (*`week_end`*) variable in the movement data (alternatively, you could use the* `week` *function if you wanted to create a time trend at the week-level). Then, use the following code to create the monthly trend:*

```
# Ensure the week_end is in Date format if it isn't already
# Ensure movement_merged is a data.table
setDT(movement_merged)

movement_merged[, week_end := as.Date(week_end)]

# Extract year and month from week_end
movement_merged[, year := year(week_end)]
movement_merged[, month := month(week_end)]

movement_merged[, month_trend := 12*(year - min(year)) + month]
```

4.7 Remove missing values

Finally, *retain only complete cases*, i.e. rows without missing values:

```
# Remove rows with any missing values
movement_merged <- movement_merged[complete.cases(movement_merged)]

# Display the cleaned data
colSums(is.na(movement_merged))
```

```
      store_code_uc          week_end    price_ArmHammer         price_Gain
                  0                 0                  0                  0
        price_Purex        price_Tide quantity_ArmHammer      quantity_Gain
                  0                 0                  0                  0
     quantity_Purex     quantity_Tide promotion_ArmHammer     promotion_Gain
                  0                 0                  0                  0
    promotion_Purex    promotion_Tide      retailer_code         store_zip3
                  0                 0                  0                  0
           SMM_code   SMM_description               year              month
                  0                 0                  0                  0
        month_trend
                  0
```

5 Data inspection

5.1 Observations and geographic coverage

First, document the number of observations and the number of unique stores in the data.

```r
#Document the number of observations and unique stores
num_observations <- nrow(movement_merged)
num_unique_stores <- uniqueN(movement_merged$store_code_uc)

cat("Number of observations:", num_observations, "\n")
```

```
Number of observations: 1259352
```

```r
cat("Number of unique stores:", num_unique_stores, "\n")
```

```
Number of unique stores: 6421
```

Second, we assesss if the included stores have broad geographic coverage. We hence create a summary table that records the number of observations for each separate Scantrack market:

```r
# Create a summary table for each Scantrack market (SMM_description)
market_coverage <- movement_merged[, .(n_obs = .N), by = SMM_description]
```

Note the use of the data.table internal `.N`: `.N` is the number of observations, either in the whole data table, or—as in this case—the number of observations within each group defined by the `by =` statement.

A convenient way to print a table is provided by the **`kable`** function that is included in the **`knitr`** package. Please consult the documentation for `kable` to see all options. Particularly useful are the options `col.names`, which is used below, and `digits`, which allows you to set the number of digits to the right of the decimal point.

Now use `kable` to document the number of observations within each Scantrack market.

```r
# Use kable to print the summary table
kable(market_coverage, col.names = c("Scantrack Market", "No. Observations"))
```

Scantrack Market	No. Observations
Cleveland	24891
Chicago	88257
Boston	42219
New Orleans - Mobile	27704
Urban NY	23363
Suburban NY	39284
Albany	8918
San Francisco	40263
Rem Omaha	6672
Nashville	5880
Rem Los Angeles - Collar	19850
Exurban NY	12285
Salt Lake City	23812
Atlanta	10049
Miami	56504
Jacksonville	16962
Oklahoma City - Tulsa	6654
Sacramento	14089
Rem Jacksonville	14786
Kansas City	7979

Scantrack Market	No. Observations
Memphis	5388
Richmond	8701
Phoenix	29709
Los Angeles	127244
Rem Atlanta	9014
Rem Richmond - Norfolk	2061
Washington DC	8636
Rem Seattle - Portland	9465
West Texas	8798
Las Vegas	19845
San Antonio	16650
Minneapolis	18830
Detroit	13109
Rem Boston	24688
Louisville	7487
St. Louis	17079
Rem Milwaukee	11217
Omaha	6903
Milwaukee	15987
Tampa	41020
Rem St. Louis	7633
Raleigh - Durham	7478
Charlotte	4166
Seattle	25835
Rem Indianapolis	5664
Pittsburgh	20966
Rem Charlotte	6884
San Diego	22955
Birmingham	11440
Orlando	29707
Portland OR	17985
Dallas	21053
Rem Denver	13596
Philadelphia	14982
Houston	22434
Baltimore	6533
Cincinnati	7528
Des Moines	5260
Rem Philadelphia	1235
Buffalo - Rochester	4172
Denver	15460
Rem New Orleans - Mobile	7358
Rem West Texas	18370
Grand Rapids	5605
Hartford - New Haven	11056
Columbus	6064
Rem Pittsburgh	2151
Indianapolis	6331
Rem North California	13171
Rem Kansas City	8380
Rem Minneapolis	3995
Rem Detroit	1771

Scantrack Market	No. Observations
Rem Greenville	2274
Little Rock	2838
Rem Memphis - Little Rock	2487
Syracuse	2283

5.2 Price variation

Before estimating the demand models we would like to understand the degree of price variation in the data. Comment on why this is important for a regression analysis such as demand estimation!

Comment :

- We need price variation to observe the changes in demand as price varies, which is crucial to derive price elasticity estimation.

We will predict demand for Tide and Gain. For each of these two brands separately, we would like to visualize the overall degree of price variation across observations, and also the variation in relative prices with respect to the competing brands.

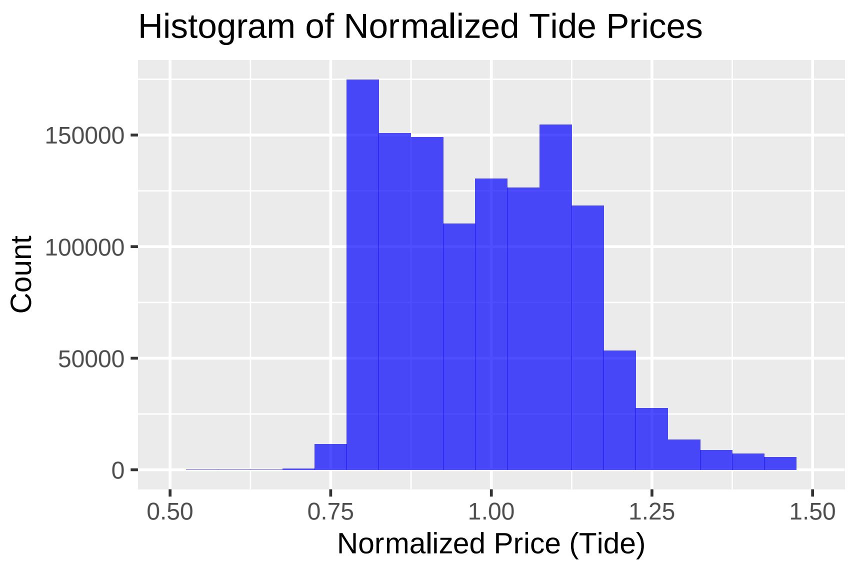
- *To visualize the (own) price variation, normalize the prices of Tide and Gain by dividing by the average of these prices, and show the histogram of normalized prices.*

```r
# Normalize prices for Tide and Gain
movement_merged[, price_Tide_normalized := price_Tide / mean(price_Tide, na.rm = TRUE)]
movement_merged[, price_Gain_normalized := price_Gain / mean(price_Gain, na.rm = TRUE)]

# Plot histograms for normalized prices
ggplot(movement_merged, aes(x = price_Tide_normalized)) +
  geom_histogram(binwidth = 0.05, fill = "blue", alpha = 0.7) +
  labs(title = "Histogram of Normalized Tide Prices", x = "Normalized Price (Tide)", y = "Count") +
  scale_x_continuous(limits = c(0.5, 1.5))  # Set limits to avoid extreme outliers
```

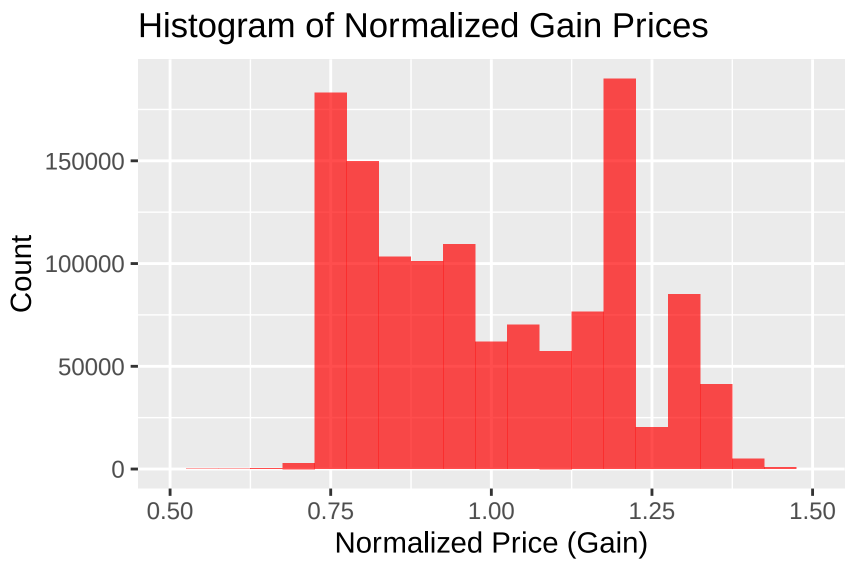
```
Warning: Removed 13193 rows containing non-finite outside the scale range
(`stat_bin()`).
```

```
Warning: Removed 2 rows containing missing values or values outside the scale range
(`geom_bar()`).
```



```r
ggplot(movement_merged, aes(x = price_Gain_normalized)) +
  geom_histogram(binwidth = 0.05, fill = "red", alpha = 0.7) +
  labs(title = "Histogram of Normalized Gain Prices", x = "Normalized Price (Gain)", y = "Count") +
  scale_x_continuous(limits = c(0.5, 1.5))  # Set limits to avoid extreme outliers
```

```
Warning: Removed 309 rows containing non-finite outside the scale range (`stat_bin()`).
Removed 2 rows containing missing values or values outside the scale range
(`geom_bar()`).
```



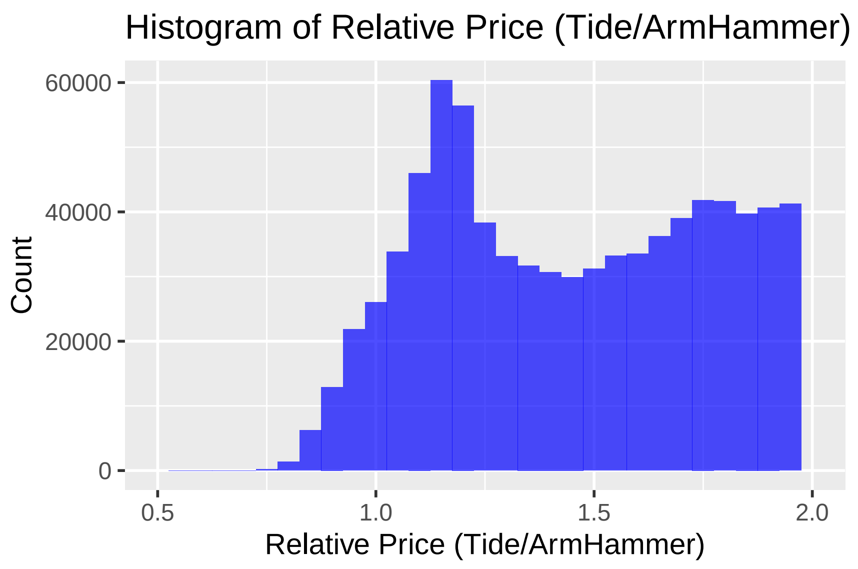
- *To visualize relative prices, calculate the ratio of Tide and Gain prices with respect to the three competing brands, and show the histogram of relative prices.*

```r
# Calculate relative prices for Tide and Gain compared to the competing brands (e.g., ArmHammer, Purex)
movement_merged[, relative_price_Tide_ArmHammer := price_Tide / price_ArmHammer]
movement_merged[, relative_price_Tide_Purex := price_Tide / price_Purex]
movement_merged[, relative_price_Gain_ArmHammer := price_Gain / price_ArmHammer]
movement_merged[, relative_price_Gain_Purex := price_Gain / price_Purex]

# Plot histograms for relative prices
ggplot(movement_merged, aes(x = relative_price_Tide_ArmHammer)) +
  geom_histogram(binwidth = 0.05, fill = "blue", alpha = 0.7) +
  labs(title = "Histogram of Relative Price (Tide/ArmHammer)", x = "Relative Price (Tide/ArmHammer)", y
  scale_x_continuous(limits = c(0.5, 2))  # Set limits to avoid extreme outliers
```

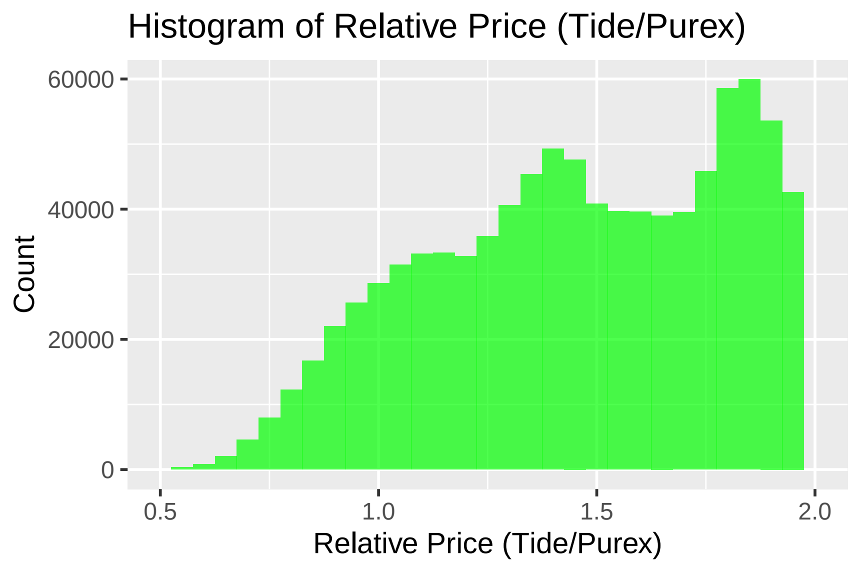
```
Warning: Removed 429795 rows containing non-finite outside the scale range
(`stat_bin()`).
```

```
Warning: Removed 2 rows containing missing values or values outside the scale range
(`geom_bar()`).
```



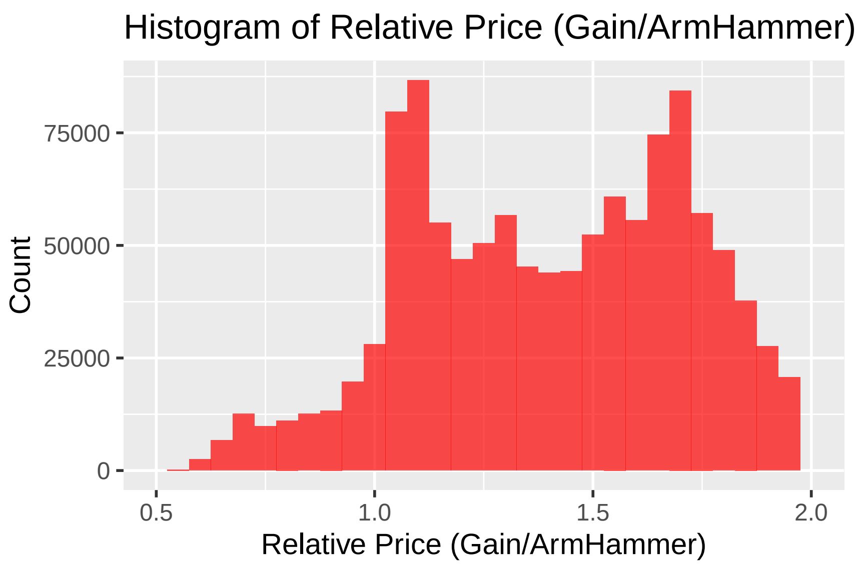
```r
ggplot(movement_merged, aes(x = relative_price_Tide_Purex)) +
  geom_histogram(binwidth = 0.05, fill = "green", alpha = 0.7) +
  labs(title = "Histogram of Relative Price (Tide/Purex)", x = "Relative Price (Tide/Purex)", y = "Coun
  scale_x_continuous(limits = c(0.5, 2))  # Set limits to avoid extreme outliers
```

```
Warning: Removed 311389 rows containing non-finite outside the scale range
(`stat_bin()`).
Removed 2 rows containing missing values or values outside the scale range
(`geom_bar()`).
```



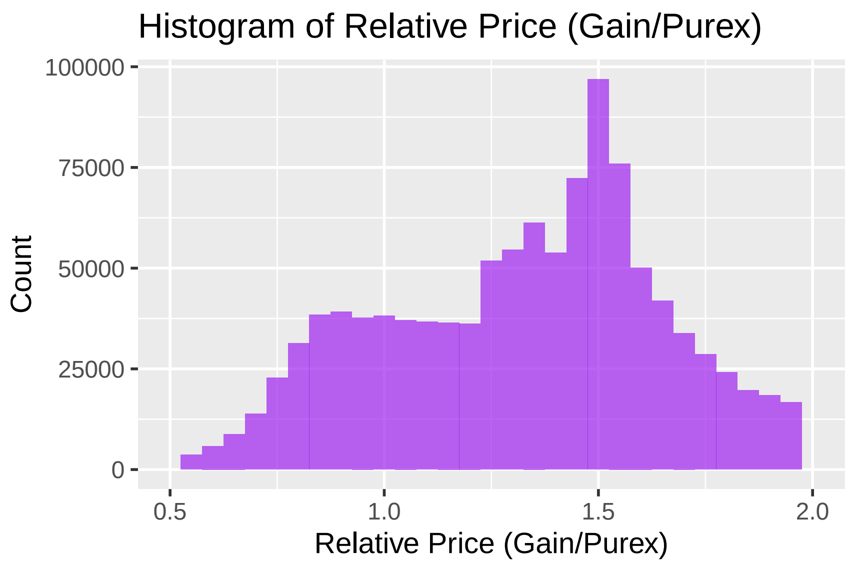
```r
ggplot(movement_merged, aes(x = relative_price_Gain_ArmHammer)) +
  geom_histogram(binwidth = 0.05, fill = "red", alpha = 0.7) +
  labs(title = "Histogram of Relative Price (Gain/ArmHammer)", x = "Relative Price (Gain/ArmHammer)", y
  scale_x_continuous(limits = c(0.5, 2))  # Set limits to avoid extreme outliers
```

```
Warning: Removed 105318 rows containing non-finite outside the scale range
(`stat_bin()`).
Removed 2 rows containing missing values or values outside the scale range
(`geom_bar()`).
```



```
ggplot(movement_merged, aes(x = relative_price_Gain_Purex)) +
  geom_histogram(binwidth = 0.05, fill = "purple", alpha = 0.7) +
  labs(title = "Histogram of Relative Price (Gain/Purex)", x = "Relative Price (Gain/Purex)", y = "Coun
  scale_x_continuous(limits = c(0.5, 2))  # Set limits to avoid extreme outliers
```

```
Warning: Removed 163462 rows containing non-finite outside the scale range
(`stat_bin()`).
Removed 2 rows containing missing values or values outside the scale range
(`geom_bar()`).
```



Note: To avoid that the scale of a graph is distorted by a few outliers, use the `limits` option in `scale_x_continuous` (see the ggplot2 introduction). This also helps to make the graphs comparable with each other.

5.3 Summary of data inspection

Discuss the data description, including sample size, geographic coverage, and the results on own and relative price variation.

1. Sample Size and Unique Stores The dataset includes 1,259,352 observations and covers 6,421 unique stores. The large sample size and number of unique stores ensure a comprehensive dataset that captures diverse pricing and demand patterns across numerous retail locations. This extensive coverage enhances the robustness of our analysis.

2. Geographic Coverage The dataset encompasses a broad range of Scantrack markets, indicating good geographic representation. The number of observations per market varies, covering major regions like Denver, Northern California, and West Texas, among others. This wide coverage enables us to study regional differences in consumer behavior, competitive strategies, and pricing variations.

3. Own Price Variation The histograms of normalized prices for Tide and Gain reveal meaningful price variation around their respective means. The histogram for Tide indicates a concentrated price range between approximately 0.75 and 1.25, with the majority of prices clustered around the mean. This distribution suggests consistent pricing with occasional adjustments, likely due to regional factors or promotions.

Similarly, the histogram for Gain shows a comparable spread of prices, but with some observable peaks, indicating periods of price adjustments or strategic promotions. These variations in own prices are essential for the demand analysis as they provide the basis for estimating the sensitivity of demand to price changes.

4. Relative Price Variation The histograms of relative prices between Tide and its competitors, Arm & Hammer and Purex, illustrate a spread of price ratios centered around 1.0. This indicates that Tide's prices are generally comparable to its competitors but occasionally deviate, suggesting pricing strategies that reflect competitive positioning.

The histograms of relative prices for Gain with respect to Arm & Hammer and Purex display similar patterns, with price ratios centered around 1.0. However, there are more pronounced peaks in the distributions, implying more aggressive pricing or promotions for Gain compared to its competitors. This relative price variation is

6 Estimation

Now we are ready to estimate demand models for Tide and Gain.

We want to estimate a sequence of models with an increasing number of controls and compare the stability of the key results across these models. In all models the output is `log(1+quantity_<brand name>)`.

To keep things simple, we will initially estimate demand for Tide only.

Let's start with the following models:

1. log of own price as only input
2. Add store fixed effects
3. Add a time trend—maybe linear, or a polynomial with higher-order terms
4. Instead of a time trend add fixed effects for each month (more precisely: for each year/month combination)

Estimate the models using the `feols` function from the fixest package (consult the corresponding fixest guide included among the R learning resources on Canvas). Store the regression outputs in some appropriately named variables (objects).

Hint: Recall that it is perfectly legitimate in R to write model formulas such as

`log(1+quantity_<brand name>) ~ log(price_<brand name>)`

Hence, there is no need to create new variables such as the logarithm of own price, etc., before estimating a demand model.

```r
# Model 1: Log of Tide's price as the only input
fit_base <- feols(log(1 + quantity_Tide) ~ log(price_Tide), data = movement_merged)

# Model 2: Add store fixed effects
fit_store_FE <- feols(log(1 + quantity_Tide) ~ log(price_Tide) | store_code_uc, data = movement_merged)

# Model 3: Add a time trend
fit_trend <- feols(log(1 + quantity_Tide) ~ log(price_Tide) + month_trend | store_code_uc, data = moveme
```

```r
# Model 4: Add year/month fixed effects

# Create a new variable that combines year and month into a single string
movement_merged[, year_month := paste(year(week_end), month(week_end), sep = "-")]

# Check the new year_month variable
head(movement_merged$year_month)
```

```
[1] "2010-1" "2010-1" "2010-1" "2010-1" "2010-2" "2010-2"
```

```r
# Model 4: Log of Tide's price with store fixed effects and year/month fixed effects
fit_month_FE <- feols(log(1 + quantity_Tide) ~ log(price_Tide) | store_code_uc + year_month, data = move
```

You can display the regression coefficients using the `summary` function. As a much more elegant solution, however, I recommend using the `etable` function in the `fixest` package, which produces nicely formatted output.

Please **consult the fixest guide on how to use `etable`**, and **go through the *Checklist for creating LaTeX tables using `etable`***!

Here is an example (note that the `fit` objects are the regression outputs—adjust the names if necessary):

```r
etable(fit_base, fit_store_FE, fit_trend, fit_month_FE,
       tex = TRUE,
       fitstat = c("n", "r2"), signif.code = NA,
       cluster = c("store_code_uc", "month_trend"))
```

Dependent Variable:	log(1+quantity_Tide)			
Model:	(1)	(2)	(3)	(4)
Variables				
Constant	-6.374			
	(0.3112)			
log(price_Tide)	-7.465	-5.612	-5.602	-5.667
	(0.1606)	(0.1375)	(0.1462)	(0.1428)
month_trend			-0.0075	
			(0.0008)	
Fixed-effects				
store_code_uc		Yes	Yes	Yes
year_month				Yes
Fit statistics				
Observations	1,259,352	1,259,352	1,259,352	1,259,352
R^2	0.49632	0.84607	0.84998	0.85242

Clustered (store_code_uc & month_trend) standard-errors in parentheses

Note the option `cluster = c("store_code_uc", "month_trend")`, which tells `etable` to show standard errors that are clustered at the store and month level. These clustered standard errors will be larger and more accurate than regular standard errors because they reflect that the error terms in the regression are likely correlated at the store and month level.

Before moving on, you may want to remove the regression output objects that are no longer used, because they take up much space in memory:

```r
rm(fit_base, fit_store_FE, fit_trend)
```

6.1 Controlling for competitor prices

Now add the competitor prices to the demand model.

```r
fit_comp <- feols(log(1 + quantity_Tide) ~ log(price_Tide) + log(price_Gain) + log(price_Purex) +  log(
```

6.2 Controlling for promotions

Now add the promotions dummies, first just for Tide, then for all brands. Compare the results. Did controlling for promotions change the own price elasticity estimate in an expected manner?

```r
# Add promotion for tide
fit_promo_comp_tide <-  feols(log(1 + quantity_Tide) ~ log(price_Tide) + log(price_Gain) + log(price_Arm

# Add promotion for all brands
fit_promo_comp <-  feols(log(1 + quantity_Tide) ~ log(price_Tide) + log(price_Gain) + log(price_ArmHamme

etable(fit_month_FE, fit_comp, fit_promo_comp_tide, fit_promo_comp,
       tex = TRUE,
       fitstat = c("n", "r2"), signif.code = NA,
       cluster = c("store_code_uc", "month_trend"))
```

Dependent Variable:	log(1+quantity_Tide)			
Model:	(1)	(2)	(3)	(4)
Variables				
log(price_Tide)	-5.667	-5.648	-4.190	-4.204
	(0.1428)	(0.1399)	(0.1439)	(0.1451)
log(price_Gain)		0.6467	0.5181	0.5741
		(0.0827)	(0.0761)	(0.0923)
log(price_Purex)		0.1938	0.1703	0.2057
		(0.0286)	(0.0252)	(0.0283)
log(price_ArmHammer)		0.1185	0.0892	0.1277
		(0.0352)	(0.0288)	(0.0308)
promotion_Tide			0.3413	0.3418
			(0.0181)	(0.0181)
promotion_Gain				0.0238
				(0.0163)
promotion_ArmHammer				0.0294
				(0.0140)
promotion_Purex				0.0376
				(0.0112)
Fixed-effects				
store_code_uc	Yes	Yes	Yes	Yes
year_month	Yes	Yes	Yes	Yes
Fit statistics				
Observations	1,259,352	1,259,352	1,259,352	1,259,352
R^2	0.85242	0.85497	0.85964	0.85978

Clustered (store_code_uc & month_trend) standard-errors in parentheses

Summarize and comment on the estimation results. Was it necessary to control for store fixed effects, time trends/fixed effects, as well as competitor prices and promotions? What do we learn from the magnitudes of the own and cross-price elasticities?

We will use the final model including all variables (I called it `fit_promo_comp`) as our preferred model. To make this final model distinguishable from the regression output for Gain we rename it:

```
fit_Tide = fit_promo_comp
```

1. Did controlling for promotions change the own price elasticity estimate in an expected manner? Yes, controlling for promotions did change the own price elasticity in an expected manner. Without promotions (Model 1 of the second table), the elasticity of Tide's price is -5.667. After controlling for promotions (Model 4 of the first table), the price elasticity becomes less negative (-4.204), which is expected since promotions tend to increase demand, making the price effect less dramatic when controlling for promotional influences.

2. Summarize and comment on the estimation results. The estimation results show that the own-price elasticity of Tide is negative across models, indicating price sensitivity. The inclusion of promotions reduces the elasticity, as the promotion increases demand directly. Cross-price elasticities with competitors show that Gain is a significant substitute (elasticity around 0.65 before promo), while Purex has little competitive effect (0.19), and Arm & Hammer also acts as a substitute with a small positive

effect (around 0.11). Promotions for Tide significantly increase demand (about 0.38), while competitor promotions have mixed effects.

3. Was it necessary to control for store fixed effects, time trends/fixed effects, as well as competitor prices and promotions? Yes, controlling for these factors was necessary to improve the accuracy of the estimates. Store fixed effects account for unique store-specific characteristics that could influence demand, while time trends and fixed effects control for temporal variations such as seasonality. Controlling for competitor prices and promotions is also critical to isolate the true effect of Tide's price and promotion on its quantity sold, ensuring that external competitive factors do not confound the results.

4. What do we learn from the magnitudes of the own and cross-price elasticities? The own-price elasticity of Tide (-4.20 in the final model) suggests that Tide is quite price-sensitive, with a 1% increase in price leading to about a 4% decrease in demand. The cross-price elasticity with Gain (around 0.65) shows a strong substitutive relationship, meaning that Gain and Tide are significant competitors. Follow by Purex and Arm & Hammer which have a small positive substitutive effect. After Promotion, the cross elasticity is softer between Tide and Gain (0.57), suggesting softer substitution effect. While stronger between Tide and Purex(increases from 0.1938 to 0.2057)/ ArmHammer(increases from 0.1185 to 0.1277). This signified that when Tide is on Promotion, consumers see Purex/ ArmHammer as possible substitute of Tide although still less than Gain. These elasticity highlight the importance of pricing and promotions in driving demand for Tide in a competitive market.

5. Promotion The coefficient for `promotion_Tide` in Models 3 and 4 is 0.3413 and 0.3418, respectively. This means that when Tide is on promotion, its demand increases by about 34%. This is a substantial effect, indicating that promotions significantly boost the demand for Tide. Model4 promotion effects are smaller but still positive, meaning that when competitors are on promotion, Tide's demand also increases slightly ie Gain is on promotion, Tide's demand increases by 2.38%.

6.3 Demand model for Gain

Now repeat the steps to estimate demand for Gain.

Briefly comment on the estimates, as you did before with Tide.

```
# Model 1: Log of Tide's price as the only input
fit_base <- feols(log(1 + quantity_Gain) ~ log(price_Gain), data = movement_merged)

# Model 2: Add store fixed effects
fit_store_FE <- feols(log(1 + quantity_Gain) ~ log(price_Gain) | store_code_uc, data = movement_merged)

# Model 3: Add a time trend
fit_trend <- feols(log(1 + quantity_Gain) ~ log(price_Gain) + month_trend | store_code_uc, data = moveme

# Model 4: Add year/month fixed effects
fit_month_FE_gain <- feols(log(1 + quantity_Gain) ~ log(price_Gain) | store_code_uc + year_month, data =

etable(fit_base, fit_store_FE, fit_trend, fit_month_FE_gain,
       tex = TRUE,
       fitstat = c("n", "r2"), signif.code = NA,
       cluster = c("store_code_uc", "month_trend"))
```

Dependent Variable:	log(1+quantity_Gain)			
Model:	(1)	(2)	(3)	(4)
Variables				
Constant	-14.55			
	(0.3124)			
log(price_Gain)	-9.967	-6.709	-6.633	-6.668
	(0.1479)	(0.1456)	(0.1685)	(0.1317)
month_trend			0.0056	
			(0.0026)	
Fixed-effects				
store_code_uc		Yes	Yes	Yes
year_month				Yes
Fit statistics				
Observations	1,259,352	1,259,352	1,259,352	1,259,352
R^2	0.56615	0.74513	0.74603	0.75024

Clustered (store_code_uc & month_trend) standard-errors in parentheses

```
rm(fit_base, fit_store_FE, fit_trend)


### Controlling for competitor prices
fit_comp_gain <- feols(log(1 + quantity_Gain) ~ log(price_Gain) + log(price_Tide) + log(price_Purex) +

## Controlling for promotions
# Add only promotion dummy for gain
fit_promo_comp_gain <- feols(log(1 + quantity_Gain) ~ log(price_Gain) + log(price_Tide) + log(price_Pur

# Add all promotion dummies
fit_promo_comp <-  feols(log(1 + quantity_Gain) ~ log(price_Gain) + log(price_Tide) + log(price_ArmHamm
```

```r
fit_Gain = fit_promo_comp

etable(fit_month_FE_gain, fit_comp_gain, fit_promo_comp_gain, fit_promo_comp,
       tex = TRUE,
       fitstat = c("n", "r2"), signif.code = NA,
       cluster = c("store_code_uc", "month_trend"))
```

Dependent Variable:	log(1+quantity_Gain)			
Model:	(1)	(2)	(3)	(4)
Variables				
log(price_Gain)	-6.668	-6.696	-4.921	-4.969
	(0.1317)	(0.1307)	(0.2871)	(0.2744)
log(price_Tide)		1.613	1.479	1.795
		(0.1902)	(0.1515)	(0.1837)
log(price_Purex)		0.1217	0.1116	0.1782
		(0.0497)	(0.0469)	(0.0445)
log(price_ArmHammer)		0.0809	0.0686	0.1481
		(0.0526)	(0.0499)	(0.0447)
promotion_Gain			0.7035	0.6993
			(0.0842)	(0.0826)
promotion_Tide				0.0795
				(0.0284)
promotion_ArmHammer				0.0647
				(0.0221)
promotion_Purex				0.0759
				(0.0206)
Fixed-effects				
store_code_uc	Yes	Yes	Yes	Yes
year_month	Yes	Yes	Yes	Yes
Fit statistics				
Observations	1,259,352	1,259,352	1,259,352	1,259,352
R^2	0.75024	0.75251	0.75845	0.75878

Clustered (store_code_uc & month_trend) standard-errors in parentheses

1. Did controlling for promotions change the own price elasticity estimate in an expected manner? Yes, controlling for promotions changed the own price elasticity estimate for Gain in an expected manner. Without controlling for promotions (Model 1 of the second table), the own price elasticity of Gain is -6.668, indicating strong price sensitivity. When promotions are controlled for (Model 4 of the second table), the elasticity becomes less negative (-4.969), as expected, since promotions increase demand and thus reduce the overall sensitivity to price changes.

2. Summarize and comment on the estimation results. The results show that the own-price elasticity of Gain is significantly negative across models, indicating that demand for Gain is highly price-sensitive. When controlling for promotions, the elasticity decreases, meaning that promotions reduce the effect of price on demand. Competitor prices, particularly Tide's price, have a positive impact on Gain's demand (elasticity around 1.613), suggesting that Tide is a significant substitute. Promotions for Gain itself have a large positive effect (0.69), while competitor promotions slightly increasing Gain demand.

3. Was it necessary to control for store fixed effects, time trends/fixed effects, as well as competitor prices and promotions? Yes, controlling for store fixed effects, time trends, and competitor prices and promotions was necessary. Store fixed effects account for store-specific characteristics, while time trends control for seasonal or time-based variations. Including competitor prices and promotions helps isolate

the true effect of Gain's price on its quantity demanded by controlling for external competitive factors that could confound the relationship between Gain's price and demand.

4. What do we learn from the magnitudes of the own and cross-price elasticities? Gain's own-price elasticity (-6.696 in the final model) shows that demand for Gain is highly price-sensitive, with a 1% increase in price leading to a 6% decrease in demand. The positive cross-price elasticity with Tide (around 1.6) suggests that Gain and Tide are strong substitutes, followed by Purex 1.2. While Arm & Hammer price effect on Gain demand is smaller, suggesting a weak substitute.
   After promotion, Tide remains a strong substitude for Gain, showing the increase from 1.613 to 1.795. This suggests the substitution is stronger when all promotions included. Tide's price plays a critical role in Gain's demand. Purex is also a substitude for Gain, price elasticity increases in model4 from 0.1217 to 0.1782. ArmHammer is a weak substitute for Gain, but becomes stronger when promotions are included (Model4) from 0.0809 to 0.1481.

7 Profitability analysis

The goal is to fine-tune prices jointly for Tide and Gain. We hence use the estimates of the preferred demand models and evaluate the product-line profits when we change the prices of the two brands.

To predict profits, let's only retain data for one year, 2013:

```
move_predict = movement_merged[year == 2013]
```

Although we have excellent demand data, we do not know the production costs of the brands (this is confidential information). We can infer the cost making an informed assumption on retail margins and the gross margin of the brand.

```
gross_margin  = 0.35
retail_margin = 0.18

cost_Tide = (1-gross_margin)*(1-retail_margin)*mean(move_predict$price_Tide)
cost_Gain = (1-gross_margin)*(1-retail_margin)*mean(move_predict$price_Gain)

cost_Tide
```

```
[1] 0.08100355
```

```
cost_Gain
```

```
[1] 0.068805
```

As prices are measured in dollars per ounce, these marginal costs are also per ounce.

Now create a vector indicating the percentage price changes that we consider within an acceptable range, up to +/- 5%.

```
percentage_delta = seq(-0.05, 0.05, 0.025)    # Identical to = c(-0.5, -0.025, 0.0, 0.025, 0.05)
percentage_delta
```

```
[1] -0.050 -0.025  0.000  0.025  0.050
```

We will consider all possible combinations of price changes for Tide and Gain. This can be easily achieved by creating a data table with the possible combinations in rows (please look at the documentation for the `rep` function):

```
L = length(percentage_delta)
profit_DT = data.table(delta_Tide = rep(percentage_delta, each = L),
                       delta_Gain = rep(percentage_delta, times = L),
                       profit     = rep(0, times = L*L))

head(profit_DT)
```

```
   delta_Tide delta_Gain profit
        <num>      <num>  <num>
1:     -0.050     -0.050      0
2:     -0.050     -0.025      0
3:     -0.050      0.000      0
4:     -0.050      0.025      0
5:     -0.050      0.050      0
6:     -0.025     -0.050      0
```

Inspect the resulting table. The `profit` column will allow us to store the predicted profits.

Now we are ready to iterate over each row in `profit_DT` and evaluate the total product-line profits of Tide and Gain for the corresponding percentage price changes. You can perform this iteration with a simple for-loop:

```r
# Store the original average price of Tide and Gain
orig_price_Tide <- mean(move_predict$price_Tide, na.rm = TRUE)
orig_price_Gain <- mean(move_predict$price_Gain, na.rm = TRUE)

# Print the original prices
orig_price_Tide # 0.1519766
```

```
[1] 0.1519766
```

```r
orig_price_Gain # 0.1290901
```

```
[1] 0.1290901
```

```r
# Function to calculate total profit for a given price, cost, and quantity
calc_profit <- function(price, cost, quantity) {
  profit = (price - cost) * quantity
  return(profit)
}
```

```r
for (i in 1:nrow(profit_DT)) {
   # Perform profit calculations for the price changes indicated in row i of the profit_DT table
  # Calculate new prices for Tide and Gain based on the percentage change
  move_predict_copy <- copy(move_predict)

  new_price_Tide <- move_predict$price_Tide * (1 + profit_DT$delta_Tide[i])
  new_price_Gain <- orig_price_Gain * (1 + profit_DT$delta_Gain[i])

  move_predict_copy$price_Tide <- new_price_Tide
  move_predict_copy$price_Gain <- new_price_Gain

  # Predict new demand for Tide and Gain using the updated prices
  move_predict_copy[, predicted_quantity_Tide := exp(predict(fit_Tide, newdata = move_predict_copy))]
  move_predict_copy[, predicted_quantity_Gain := exp(predict(fit_Gain, newdata = move_predict_copy))]

  # Calculate profits for Tide and Gain
  profit_Tide <- calc_profit(move_predict_copy$price_Tide, cost_Tide, move_predict_copy$predicted_quant
  profit_Gain <- calc_profit(move_predict_copy$price_Gain, cost_Gain, move_predict_copy$predicted_quant

  # Store total product-line profits (Tide + Gain) in profit_DT
  profit_DT$profit[i] <- sum(profit_Tide, profit_Gain, na.rm = TRUE)
  print(profit_DT$profit[i])
}
```

```
[1] 112170699
[1] 112453772
[1] 112800862
[1] 113209301
[1] 113675660
[1] 109662121
[1] 109840588
[1] 110087409
```

```
[1] 110399744
[1] 110773958
[1] 107079147
[1] 107150511
[1] 107294662
[1] 107508580
[1] 107788421
[1] 104484688
[1] 104447407
[1] 104487425
[1] 104601545
[1] 104785706
[1] 101925324
[1] 101778562
[1] 101713686
[1] 101727317
[1] 101815175
```

```r
# Base Total Profit
  move_predict_copy <- copy(move_predict)
  i = 13
  move_predict_copy$price_Tide <- move_predict$price_Tide * (1 + profit_DT$delta_Tide[i])
  move_predict_copy$price_Gain <- orig_price_Gain * (1 + profit_DT$delta_Gain[i])

  move_predict_copy[, predicted_quantity_Tide := exp(predict(fit_Tide, newdata = move_predict_copy))] #
  move_predict_copy[, predicted_quantity_Gain := exp(predict(fit_Gain, newdata = move_predict_copy))]

  # Calculate profits for Tide and Gain
  profit_Tide <- calc_profit(move_predict_copy$price_Tide, cost_Tide, move_predict_copy$predicted_quant
  profit_Gain <- calc_profit(move_predict_copy$price_Gain, cost_Gain, move_predict_copy$predicted_quant

  #sum(move_predict_copy$predicted_quantity_Tide) #1870104585
  #sum(move_predict_copy$predicted_quantity_Gain) #265303723

  # Store total product-line profits (Tide + Gain) in profit_DT
  baseline_total_profit <- sum(profit_Tide, profit_Gain, na.rm = TRUE)
  # Print baseline total profit
  print(baseline_total_profit)
```

```
[1] 107294662
```

```r
  # Calculate profit ratio relative to baseline profit
  profit_DT[, profit_ratio := profit / baseline_total_profit]
```

Some hints:

- Before you start the loop, store the original price levels of Tide and Gain.
- Update the price columns in `move_predict` and then predict demand.
- Calculate total profits at the new price levels for both brands and then store the total profit from Tide and Gain in `profit_DT`.

Show a table of profits in levels and in ratios relative to the baseline profit at current price levels, in order to assess the percent profit differences resulting from the contemplated price changes.

```r
# Display the table with profit levels and ratios
kable(profit_DT, col.names = c("Delta Tide", "Delta Gain", "Profit Level", "Profit Ratio"), digits = 3)
```

Delta Tide	Delta Gain	Profit Level	Profit Ratio
-0.050	-0.050	112170699	1.045
-0.050	-0.025	112453772	1.048
-0.050	0.000	112800862	1.051
-0.050	0.025	113209301	1.055
-0.050	0.050	113675660	1.059
-0.025	-0.050	109662121	1.022
-0.025	-0.025	109840588	1.024
-0.025	0.000	110087409	1.026
-0.025	0.025	110399744	1.029
-0.025	0.050	110773958	1.032
0.000	-0.050	107079147	0.998
0.000	-0.025	107150511	0.999
0.000	0.000	107294662	1.000
0.000	0.025	107508580	1.002
0.000	0.050	107788421	1.005
0.025	-0.050	104484688	0.974
0.025	-0.025	104447407	0.973
0.025	0.000	104487425	0.974
0.025	0.025	104601545	0.975
0.025	0.050	104785706	0.977
0.050	-0.050	101925324	0.950
0.050	-0.025	101778562	0.949
0.050	0.000	101713686	0.948
0.050	0.025	101727317	0.948
0.050	0.050	101815175	0.949

85238164

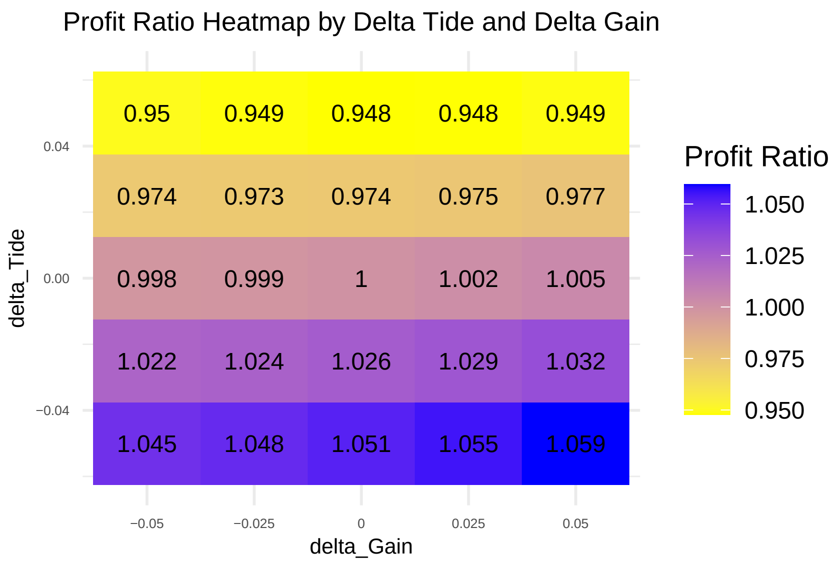
```
# Install and load necessary packages (if not already installed)
if (!require(ggplot2)) install.packages("ggplot2")
if (!require(reshape2)) install.packages("reshape2")

library(ggplot2)
library(reshape2)

# Reshape the data to wide format for heatmap
profit_wide <- dcast(profit_DT, delta_Tide ~ delta_Gain, value.var = "profit_ratio")

# Melt the data back into a long format for ggplot
profit_long <- melt(profit_wide, id.vars = "delta_Tide", variable.name = "delta_Gain", value.name = "pr

# Create the heatmap using ggplot2 with smaller text sizes
ggplot(profit_long, aes(x = delta_Gain, y = delta_Tide, fill = profit_ratio)) +
  geom_tile() +
  scale_fill_gradient(low = "yellow", high = "blue") +
  geom_text(aes(label = round(profit_ratio, 3)), color = "black", size = 3) +  # Display profit_ratio v
  labs(title = "Profit Ratio Heatmap by Delta Tide and Delta Gain",
       x = "delta_Gain",
       y = "delta_Tide",
       fill = "Profit Ratio") +
  theme_minimal() +
  theme(
    plot.title = element_text(size = 10, hjust = 0.5),  # Adjust title size
    axis.title.x = element_text(size = 8),  # Adjust x-axis title size
    axis.title.y = element_text(size = 8),  # Adjust y-axis title size
    axis.text.x = element_text(size = 5),   # Adjust x-axis tick label size
    axis.text.y = element_text(size = 5)    # Adjust y-axis tick label size
  )
```



Discuss the profitability predictions and how prices should be changed, if at all. How do you reconcile the recommended price changes with the own-price elasticity estimates?

1. Data :

- Baseline scenario : Profit Ratio: 1.
  - Ratio is 1, the baseline profit when no price changes.
- Best Profit : Tide increases by 5% and Gain Price increase by 5% (delta_Tide = -0.050 and delta_Gain = +0.050)
  - Profit ratio is 1.059, signifies 5.9% increase in profit relative to the baseline. Suggesting that consumers buys more Tide and steal demand from Gain. Overall increasing total profit of both products. This combination is better than reduceing price for both products.
- Most Loss : Both Tide and Gain Price increase 5% (delta_Tide = +0.050 and delta_Gain = +0.050)
  - Ratio 0.949, which is 5.1% reduction in profit compared to the baseline.
- When Tide's price is reduced by 5% (delta Tide = -0.05) and Gain's price remains unchanged (delta Gain = 0.00), the profit ratio is 1.051, indicating a 5.1% increase in profit. This shows that customers buy more Tide in response to the price reduction, but it does not appear to significantly reduce Gain's sales.
  - When Tide's price is reduced, customers tend to buy more of Tide, but this doesn't necessarily mean they are switching from Gain. This suggests that the demand for Tide is highly elastic—meaning (~ -5 own-elasticity) that consumers are very responsive to price reductions for Tide, and they buy more of it when the price goes down.
  - However, Gain's demand doesn't drop significantly when Tide's price is reduced, indicating that Gain's customer base is less sensitive to Tide's price changes. Gain and Tide may serve slightly different customer segments, or Gain's customers may be less inclined to switch products based on Tide's pricing alone.

2. Suggestion :

   - Reduce Prices: The most profitable strategy is to implement a promotion to reduce prices of Tide

5% and increase price of Gain by 5%. This yields in a 5.9% increase in profit, the highest observed.

- Avoid Price Hikes: Increasing prices, even slightly, results in profit losses due to the strong price elasticity of both brands.