Base Pricing Analysis and Price Elasticity Estimation

Ben Heim, Marcell Milo-Sidlo, Pierre Chan, Julia Luo, Alicia Soosai

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.

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

3 Data overview

The data are located in this folder:

```
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
```

*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**.

```
data <- data.frame(
  product_id = c(653791, 623280, 557775, 507562, 503626),
  product_name = c('TIDE - H-D LIQ', 'PUREX - H-D LIQ', 'GAIN - H-D LIQ', 'ARM & HAMMER - H-D LIQ', 'ALL
  category = c('DETERGENTS - HEAVY DUTY - LIQUID', 'DETERGENTS - HEAVY DUTY - LIQUID', 'DETERGENTS - HEA
  quantity = c(7012, 4956, 1201, 1010, 4412),
  price = c(7, 9, 10, 15, 12),
  promotion = c(1, 0, 1, 1, 0)
)

selected_brands <- data[data$product_name %in% c('TIDE - H-D LIQ', 'PUREX - H-D LIQ', 'GAIN - H-D LIQ',

selected_brands$brand_name <- c('Tide', 'Purex', 'Gain', 'ArmHammer')

print(selected_brands)
```

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).

```
load("~/Desktop/data_driven_marketing/ddm_assignments/hw2/data/brand_move_7012.RData")
library(data.table)
setnames(move, old = "promo_dummy", new = "promotion", skip_absent=TRUE)
setnames(move, old = "units", new = "quantity", skip_absent=TRUE)
setnames(move, old = "brand_code_uc", new = "product_id", skip_absent=TRUE)
move$promotion <- as.numeric(move$promotion)
head(move)
```

```r
final_data <- merge(move, selected_brands, by = "product_id", all = FALSE)
print(final_data)
```

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.

```r
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.

```r
threshold_bottom <- 0.35
threshold_top <- 2.5
outliers <- isOutlier(final_data$price.x, threshold_bottom, threshold_top)
print(outliers)
```

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
move <- move[, .(store_code_uc, week_end, product_id, quantity, price, promotion)]

wide_data <- dcast(
  move,
  store_code_uc + week_end ~ product_id,
  value.var = c("quantity", "price", "promotion")
)

head(wide_data)
```

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.

```
load("~/Desktop/data_driven_marketing/ddm_assignments/hw2/data/Stores.RData")
ls()
print(stores)
```

```
valid_store_meta <- stores[!is.na(stores$retailer_code)]
final_data_with_store <- merge(wide_data, valid_store_meta, by = "store_code_uc", all = FALSE)
head(final_data_with_store)
```

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:*

```
library(lubridate)
final_data_with_store$week_end <- as.Date(final_data_with_store$week_end)
final_data_with_store[, year := year(week_end)]
final_data_with_store[, month := month(week_end)]
final_data_with_store[, month_trend := 12*(year - min(year)) + month]
head(final_data_with_store[, .(week_end, year, month, month_trend)])
```

4.7 Remove missing values

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

```
final_data_with_store = final_data_with_store[complete.cases(final_data_with_store)]
final_data_with_store
```

5 Data inspection

5.1 Observations and geographic coverage

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

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:

```
market_coverage = move[, .(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.*

```
kable(market_coverage, col.names = c("Scantrack market", "No. obs."))
```

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!

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.*
- *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.*

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.

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.

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):

```
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"))
```

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:

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

6.1 Controlling for competitor prices

Now add the competitor prices to the demand model.

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?

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
```

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.

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 = move[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)
```

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)
```

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))
```

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:

```
for (i in 1:nrow(profit_DT)) {
   # Perform profit calculations for the price changes indicated in row i of the profit_DT table
}
```

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.

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?