Advertising Effects

Giovanni Compiani

Contents

```r
# install.packages("RcppRoll")
library(bit64)
library(data.table)
library(RcppRoll)
library(ggplot2)
library(fixest)
library(knitr)
library(dplyr)
```

1 Overview

In this assignment we estimate the causal short and long-run effects of advertising on demand. The assignment is closely related to the paper "TV Advertising Effectiveness and Profitability: Generalizable Results from 288 Brands" (2001, *Econometrica*) by Shapiro, Hitsch, and Tuchman.

We first combine store-level sales data from the Nielsen RMS scanner data set with DMA-level advertising exposure data from the Nielsen Ad Intel advertising data set. We then estimate ad effects based on a within-market strategy controlling for cross-sectional heterogeneity across markets.

2 Data

2.1 Brands and product modules

Data location:

```r
data_folder = "/Users/tracy/Desktop/BUSN37105/Data Files"

#data_folder = "Data"

brands_file = "Brands_a3.RData"
stores_file = "stores_dma.RData"
move_file  = "move_8412.RData"
adv_file = "adv_8412.RData"
```

The table `brands_DT` in the file `Brands_a3.RData` provides information on the available product categories (product modules) and brands, including the "focal" brands for which we may estimate advertising effects.

product_module_code	product_module_desc	brand_code_uc	brand_descr	focal_brand
1484	SOFT DRINKS - CARBONATED	531429	COCA-COLA R	TRUE
8412	ANTACIDS	621727	PRILOSEC	TRUE
1553	SOFT DRINKS - LOW CALORIE	531433	COCA-COLA ZERO DT	TRUE

Choose Prilosec in the Antacids category for your analysis. Later, you can **optionally** repeat your analysis for the other brands.

```
selected_module = 8412
selected_brand  = 621727
```

3 Data preparation

To prepare and build the data for the main analysis, load the brand and store meta-data in `Brands_a3.RData` and `stores_dma.RData`, the RMS store-level scanner (movement) data, and the Nielsen Ad Intel DMA-level TV advertising data. The scanner data and advertising data are named according to the product module, such as `move_8412.RData` and `adv_8412.RData`.

Both the RMS scanner data and the Ad Intel advertising data include information for the top four brands in the category (product module). To make our analysis computationally more manageable we will not distinguish among all individual competing brands, but instead we will aggregate all competitors into one single brand.

```
load(paste0(data_folder, "/", brands_file))
load(paste0(data_folder, "/", stores_file))
load(paste0(data_folder, "/", move_file))
load(paste0(data_folder, "/", adv_file))
```

3.1 RMS scanner data (`move`)

Let us start manipulating the 'move' dataset.

For consistency, rename the `units` to `quantity` and `promo_percentage` to `promotion` (use the `setnames` command). The promotion variable captures promotional activity as a continuous variable with values between 0 and 1.

```
setnames(move, old = c("units", "promo_percentage"), new = c("quantity", "promotion"))
head(move)
```

```
Key: <brand_code_uc, store_code_uc, week_end>
   brand_code_uc store_code_uc   week_end quantity     price promotion
           <int>         <int>     <Date>    <num>     <num>     <num>
1:        536746          2324 2010-01-02     4520 0.2842310         0
2:        536746          2324 2010-01-09     5456 0.2864958         0
3:        536746          2324 2010-01-16     5102 0.2864958         0
4:        536746          2324 2010-01-23     4950 0.2864041         0
5:        536746          2324 2010-01-30     4697 0.2861501         0
6:        536746          2324 2010-02-06     6090 0.2863082         0
```

Create the variable `brand_name` that we will use to distinguish between the own and aggregate competitor variables. The brand name `own` corresponds to the focal brand (Prilosec in our case), and `comp` (or any other name that you prefer) corresponds to the aggregate competitor brand.

```
move[, brand_name := ifelse(brand_code_uc == selected_brand, "own", "comp")]
head(move)
```

```
Key: <brand_code_uc, store_code_uc, week_end>
   brand_code_uc store_code_uc   week_end quantity     price promotion
           <int>         <int>     <Date>    <num>     <num>     <num>
1:        536746          2324 2010-01-02     4520 0.2842310         0
2:        536746          2324 2010-01-09     5456 0.2864958         0
3:        536746          2324 2010-01-16     5102 0.2864958         0
4:        536746          2324 2010-01-23     4950 0.2864041         0
```

```
5:        536746          2324 2010-01-30     4697 0.2861501         0
6:        536746          2324 2010-02-06     6090 0.2863082         0
   brand_name
       <char>
1:       comp
2:       comp
3:       comp
4:       comp
5:       comp
6:       comp
```

We need to aggregate the data for each store/week observation, separately for the `own` and `comp` data. To aggregate prices and promotions we can take the simple arithmetic `mean` over all competitor brands (a weighted mean may be preferable but is not necessary in this analysis where prices and promotions largely serve as controls, not as the main marketing mix variables of interest). Aggregate quantities can be obtained as the `sum` over brand-level quantities.

```
move_agg <- move[, .(quantity = sum(quantity), price = mean(price),
                     promotion = mean(promotion)),
                 by = .(store_code_uc, week_end, brand_name)]
head(move_agg)
```

```
   store_code_uc   week_end brand_name quantity     price promotion
           <int>     <Date>     <char>    <num>     <num>     <num>
1:          2324 2010-01-02       comp     5048 0.4396163  0.167566
2:          2324 2010-01-09       comp     6008 0.4424408  0.000000
3:          2324 2010-01-16       comp     5898 0.4399273  0.000000
4:          2324 2010-01-23       comp     5508 0.4367092  0.167566
5:          2324 2010-01-30       comp     5375 0.4327154  0.167566
6:          2324 2010-02-06       comp     6310 0.4339731  0.167566
```

Later, when we merge the RMS scanner data with the Ad Intel advertising data, we need a common key between the two data sets. This key will be provided by the DMA code and the date. Hence, we need to merge the `dma_code` found in the `stores` table with the RMS movement data.

Now merge the `dma_code` with the movement data.

```
move_agg <- merge(move_agg, stores_dma[, .(store_code_uc, dma_code)],
                  by = "store_code_uc",
                  all.x = TRUE)
head(move_agg)
```

```
Key: <store_code_uc>
   store_code_uc   week_end brand_name quantity     price promotion dma_code
           <int>     <Date>     <char>    <num>     <num>     <num>    <int>
1:          2324 2010-01-02       comp     5048 0.4396163  0.167566      602
2:          2324 2010-01-09       comp     6008 0.4424408  0.000000      602
3:          2324 2010-01-16       comp     5898 0.4399273  0.000000      602
4:          2324 2010-01-23       comp     5508 0.4367092  0.167566      602
5:          2324 2010-01-30       comp     5375 0.4327154  0.167566      602
6:          2324 2010-02-06       comp     6310 0.4339731  0.167566      602
```

3.2 Ad Intel advertising data (`adv_DT`)

The table `adv_DT` contains information on brand-level GRPs (gross rating points) for each DMA/week combination. The original data are more disaggregated, and include individual occurrences on a specific date and at a specific time and the corresponding number of impressions. `adv_DT` is based on the original data, aggregated at the DMA/week level.

Weeks are indicated by `week_end`, where the corresponding date is always a Saturday. We use Saturdays so that the `week_end` variable in the advertising data corresponds to the date convention in the RMS scanner data, where `week_end` also corresponds to a Saturday.

The data contain two variables to measure brand-level GRPs, `grp_direct` and `grp_indirect`. `grp_direct` records GRPs for which we can create a direct, unambiguous match between the brand name in the scanner data and the name of the advertised brand. Sometimes, however, it is not entirely clear if we should associate an ad in the Ad Intel data with the brand in the RMS data. For example, should we count ads for BUD LIGHT BEER LIME when measuring the GRPs that might affect sales of BUD LIGHT BEER? As such matches are somewhat debatable, we record the corresponding GRPs in the variable `grp_indirect`.

The data do not contain observations for all DMA/week combinations during the observation period. In particular, no DMA/week record is included if there was no corresponding advertising activity. For our purposes, however, it is important to capture that the number of GRPs was 0 for such observations. Hence, we need to "fill the gaps" in the data set.

data.table makes it easy to achieve this goal. Let's illustrate using a simple example:

```r
set.seed(444)
DT = data.table(dma  = rep(LETTERS[1:2], each = 5),
                week = 1:5,
                x    = round(runif(10, min = 0, max =20)))
DT = DT[-c(2, 5, 9)]
DT
```

```
      dma  week     x
   <char> <int> <num>
1:      A     1     3
2:      A     3     8
3:      A     4     7
4:      B     1    12
5:      B     2    11
6:      B     3     1
7:      B     5     6
```

In `DT`, the observations for weeks 2 and 5 in market A and week 4 in market B are missing.

To fill the holes, we need to key the data.table to specify the dimensions—here the `dma` and `week`. Then we perform a *cross join* using `CJ` (see `?CJ`). In particular, for each of the variables along which `DT` is keyed we specify the full set of values that the final data.table should contain. In this example, we want to include the markets A and B and all weeks, 1-5.

```r
setkey(DT, dma, week)
DT = DT[CJ(c("A", "B"), 1:5)]
DT
```

```
Key: <dma, week>
```

```
       dma  week     x
    <char> <int> <num>
 1:      A     1     3
 2:      A     2    NA
 3:      A     3     8
 4:      A     4     7
 5:      A     5    NA
 6:      B     1    12
 7:      B     2    11
 8:      B     3     1
 9:      B     4    NA
10:      B     5     6
```

We can replace all missing values (`NA`) with another value, say -111, like this:

```r
DT[is.na(DT)] = -111
DT
```

```
Key: <dma, week>
       dma  week     x
    <char> <int> <num>
 1:      A     1     3
 2:      A     2  -111
 3:      A     3     8
 4:      A     4     7
 5:      A     5  -111
 6:      B     1    12
 7:      B     2    11
 8:      B     3     1
 9:      B     4  -111
10:      B     5     6
```

Use this technique to expand the advertising data in `adv_DT`, using a cross join along along all `brands`, `dma_codes`, and `weeks`:

```r
brands    = unique(adv_DT$brand_code_uc)
dma_codes = unique(adv_DT$dma_code)
weeks     = seq(from = min(adv_DT$week_end), to = max(adv_DT$week_end), by = "week")
```

Now perform the cross join and set missing values to 0.

```r
setkey(adv_DT, dma_code, week_end, brand_code_uc)
adv_DT_all <- adv_DT[CJ(dma_code = dma_codes, week_end = weeks, brand_name = brands)]

adv_DT_all[is.na(grp_direct), grp_direct := 0]
adv_DT_all[is.na(grp_indirect), grp_indirect := 0]
```

Create own and competitor names, and then aggregate the data at the DMA/week level, similar to what we did with the RMS scanner data. In particular, aggregate based on the sum of GRPs (separately for `grp_direct` and `grp_indirect`).

```r
adv_DT_all[, brand_name := ifelse(brand_code_uc == selected_brand, "own", "comp")]
adv_DT_all <- adv_DT_all[, .(grp_direct = sum(grp_direct),
                             grp_indirect = sum(grp_indirect)),
                         by = .(dma_code, week_end, brand_name)]
adv_DT_all[, grp := grp_direct + grp_indirect]
head(adv_DT_all)
```

```
   dma_code   week_end brand_name grp_direct grp_indirect      grp
      <num>     <Date>     <char>      <num>        <num>    <num>
1:      500 2010-01-02       comp   211.2821            0 211.2821
2:      500 2010-01-02        own   195.9066            0 195.9066
3:      500 2010-01-09       comp   366.3477            0 366.3477
4:      500 2010-01-09        own   195.7711            0 195.7711
5:      500 2010-01-16       comp   560.0326            0 560.0326
6:      500 2010-01-16        own   118.8251            0 118.8251
```

At this stage we need to decide if we want to measure GRPs using only `grp_direct` or also including `grp_indirect`. I propose to take the broader measure, and sum the GRPs from the two variables to create a combined `grp` measure. You can later check if your results are robust if you use `grp_direct` only (this robustness analysis is optional).

Note: In the Antacids category, `grp_indirect` only contains the value 0 and is therefore not relevant. However, if you work with the data in the other categories, `grp_indirect` contains non-zero values.

3.3 Calculate adstock/goodwill

Advertising is likely to have long-run effects on demand. Hence, we will calculate adstock or goodwill variables for own and competitor advertising. We will use the following, widely-used adstock specification (a_t is advertising in period t):

$$g_t = \sum_{l=0}^{L} \delta^l \log(1 + a_{t-l}) = \log(1 + a_t) + \delta \log(1 + a_{t-1}) + \cdots + \delta^L \log(1 + a_{t-L})$$

We add 1 to the advertising levels (GRPs) before taking the log to deal with the large number of zeros in the GRP data.

Here is a particularly easy and fast approach to calculate adstocks. First, define the adstock parameters—the number of lags and the carry-over factor δ.

```
N_lags = 52
delta  = 0.7
```

Then calculate the geometric weights based on the carry-over factor.

```
geom_weights = cumprod(c(1.0, rep(delta, times = N_lags)))
geom_weights = sort(geom_weights)
tail(geom_weights)
```

```
[1] 0.16807 0.24010 0.34300 0.49000 0.70000 1.00000
```

Now we can calculate the adstock variable using the `roll_sum` function in the `RcppRoll` package.

```
setkey(adv_DT_all, brand_name, dma_code, week_end)
adv_DT_all[, adstock := roll_sum(log(1+grp), n = N_lags+1, weights = geom_weights,
                                 normalize = FALSE, align = "right",  fill = NA),
        by = .(brand_name, dma_code)]
```

Explanations:

1. Key the table along the cross-sectional units (brand name and DMA), then along the time variable. This step is *crucial*! If the table is not correctly sorted, the time-series order of the advertising data will be incorrect.
2. Use the `roll_sum` function based on `log(1+grp)`. `n` indicates the total number of elements in the rolling sum, and `weights` indicates the weights for each element in the sum. `normalize = FALSE` tells the function to leave the `weights` untouched, `align = "right"` indicates to use all data above the current row in the data table to calculate the sum, and `fill = NA` indicates to fill in missing values for the first rows for which there are not enough elements to take the sum.

Alternatively, you could code your own weighted sum function:

```
weightedSum <- function(x, w) {
   T = length(x)
   L = length(w) - 1
   y = rep_len(NA, T)
   for (i in (L+1):T) y[i] = sum(x[(i-L):i]*w)
   return(y)
}
```

Let's compare the execution speed:

```
time_a = system.time(adv_DT_all[, stock_a := weightedSum(log(1+grp), geom_weights),
                                by = .(brand_name, dma_code)])
```

```
time_b = system.time(adv_DT_all[, stock_b := roll_sum(log(1+grp), n = N_lags+1,
                                                      weights = geom_weights,
                                                      normalize = FALSE,
                                                      align = "right",  fill = NA),
                                by = .(brand_name, dma_code)])
```

Even though the `weightedSum` function is fast, the speed difference with respect to the optimized code in `RcppRoll` is large.

```
(time_a/time_b)[3]
```

```
elapsed
     22
```

Lesson: Instead of reinventing the wheel, spend a few minutes searching the Internet to see if someone has already written a package that solves your coding problems.

3.4 Merge scanner and advertising data

Merge (join) the advertising data with the scanner data based on brand name, DMA code, and week.

```r
# Check the structure of the scanner data (move_agg)
str(move_agg)
```

```
Classes 'data.table' and 'data.frame':  7009550 obs. of  7 variables:
 $ store_code_uc: int  2324 2324 2324 2324 2324 2324 2324 2324 2324 2324 ...
 $ week_end     : Date, format: "2010-01-02" "2010-01-09" ...
 $ brand_name   : chr  "comp" "comp" "comp" "comp" ...
 $ quantity     : num  5048 6008 5898 5508 5375 ...
 $ price        : num  0.44 0.442 0.44 0.437 0.433 ...
 $ promotion    : num  0.168 0 0 0.168 0.168 ...
 $ dma_code     : int  602 602 602 602 602 602 602 602 602 602 ...
 - attr(*, ".internal.selfref")=<externalptr>
 - attr(*, "sorted")= chr "store_code_uc"
```

```r
# Check the structure of the advertising data (adv_DT_all)
str(adv_DT_all)
```

```
Classes 'data.table' and 'data.frame':  67072 obs. of  9 variables:
 $ dma_code    : num  500 500 500 500 500 500 500 500 500 500 ...
 $ week_end    : Date, format: "2010-01-02" "2010-01-09" ...
 $ brand_name  : chr  "comp" "comp" "comp" "comp" ...
 $ grp_direct  : num  211 366 560 437 339 ...
 $ grp_indirect: num  0 0 0 0 0 0 0 0 0 0 ...
 $ grp         : num  211 366 560 437 339 ...
 $ adstock     : num  NA NA NA NA NA NA NA NA NA NA ...
 $ stock_a     : num  NA NA NA NA NA NA NA NA NA NA ...
 $ stock_b     : num  NA NA NA NA NA NA NA NA NA NA ...
 - attr(*, ".internal.selfref")=<externalptr>
 - attr(*, "sorted")= chr [1:3] "brand_name" "dma_code" "week_end"
```

```r
# Ensure that the `week_end` columns in both datasets are of Date type
move_agg[, week_end := as.Date(week_end)]
adv_DT_all[, week_end := as.Date(week_end)]
```

```r
# Perform the merge
merged_data <- merge(adv_DT_all, move_agg,
                     by = c("brand_name", "dma_code", "week_end"),
                     all.x = TRUE,
                     all.y = FALSE)

# Check the result of the merge
head(merged_data)
```

```
Key: <brand_name, dma_code, week_end>
   brand_name dma_code   week_end grp_direct grp_indirect      grp adstock
       <char>    <int>     <Date>      <num>        <num>    <num>   <num>
1:       comp      500 2010-01-02   211.2821            0 211.2821      NA
2:       comp      500 2010-01-02   211.2821            0 211.2821      NA
```

```
3:       comp      500 2010-01-02   211.2821            0 211.2821      NA
4:       comp      500 2010-01-02   211.2821            0 211.2821      NA
5:       comp      500 2010-01-02   211.2821            0 211.2821      NA
6:       comp      500 2010-01-02   211.2821            0 211.2821      NA
   stock_a stock_b store_code_uc quantity     price  promotion
     <num>   <num>         <int>    <num>     <num>      <num>
1:      NA      NA         88153     1432 0.5957322 0.00000000
2:      NA      NA         95752     1946 0.5644275 0.08261606
3:      NA      NA        123214     1592 0.6173410 0.00000000
4:      NA      NA        129685     1146 0.5777817 0.20306278
5:      NA      NA        189538     1698 0.5948342 0.00000000
6:      NA      NA        366762     4388 0.4063640 0.23904801
```

```r
# Disregard the missing store_code_uc rows
merged_data <- merged_data[complete.cases(store_code_uc)]

# Find rows where the combination is duplicated
duplicates <- merged_data[duplicated(merged_data[, .(store_code_uc,
                                                     week_end, brand_name)]), ]
duplicates
```

```
Key: <brand_name, dma_code, week_end>
Empty data.table (0 rows and 13 cols): brand_name,dma_code,week_end,grp_direct,grp_indirect,grp...
```

3.5 Reshape the data

Use `dcast` to reshape the data from long to wide format. The store code and week variable are the main row identifiers. Quantity, price, promotion, and adstock are the column variables.

If you inspect the data you will see many missing `adstock` values, because the adstock variable is not defined for the first `N_lags` weeks in the data. To free memory, remove all missing values from `move` (`complete.cases`).

```r
dim(merged_data) # before merged
```

```
[1] 7009550      13
```

```r
# Reshape the data from long to wide format using dcast
reshaped_data <- dcast(merged_data, store_code_uc + week_end ~ brand_name,
                       value.var = c("quantity", "price", "promotion", "adstock", "grp"))
```

```r
# Remove rows with missing values using complete.cases
cleaned_data <- reshaped_data[complete.cases(reshaped_data)]

# Check the cleaned data
summary(cleaned_data)
```

```
 store_code_uc        week_end          quantity_comp    quantity_own
 Min.   :   2324   Min.   :2011-01-01   Min.   :    0   Min.   :   0.0
 1st Qu.:2100029   1st Qu.:2011-12-31   1st Qu.:  512   1st Qu.:  56.0
 Median :4253149   Median :2012-12-29   Median : 1143   Median : 126.0
```

```
 Mean   :4212805   Mean   :2012-12-27   Mean   : 1580   Mean   : 187.5
 3rd Qu.:6223081   3rd Qu.:2013-12-28   3rd Qu.: 2148   3rd Qu.: 252.0
 Max.   :8388364   Max.   :2014-12-27   Max.   :24414   Max.   :6006.0
   price_comp        price_own          promotion_comp      promotion_own
 Min.   :0.0166   Min.   :0.000714   Min.   :0.000000   Min.   :0.0000
 1st Qu.:0.4979   1st Qu.:0.668914   1st Qu.:0.008221   1st Qu.:0.0000
 Median :0.5554   Median :0.705233   Median :0.067028   Median :0.0000
 Mean   :0.5488   Mean   :0.714194   Mean   :0.097208   Mean   :0.1619
 3rd Qu.:0.6016   3rd Qu.:0.756818   3rd Qu.:0.146589   3rd Qu.:0.3047
 Max.   :1.2333   Max.   :4.717686   Max.   :1.000000   Max.   :1.0000
  adstock_comp         adstock_own         grp_comp           grp_own
 Min.   : 0.000139   Min.   : 2.693   Min.   :  0.0000   Min.   :  0.0
 1st Qu.: 8.947420   1st Qu.:16.544   1st Qu.:  0.3748   1st Qu.:134.9
 Median :13.987780   Median :17.111   Median :117.9095   Median :168.8
 Mean   :11.861599   Mean   :16.752   Mean   :113.2061   Mean   :176.9
 3rd Qu.:16.226524   3rd Qu.:17.535   3rd Qu.:179.2854   3rd Qu.:209.2
 Max.   :20.062026   Max.   :20.512   Max.   :654.8650   Max.   :628.5
```

```r
# Check the structure of the cleaned dataset
str(cleaned_data)
```

```
Classes 'data.table' and 'data.frame':  2800307 obs. of  12 variables:
 $ store_code_uc : int  2324 2324 2324 2324 2324 2324 2324 2324 2324 2324 ...
 $ week_end      : Date, format: "2011-01-01" "2011-01-08" ...
 $ quantity_comp : num  5104 6288 5096 5196 4118 ...
 $ quantity_own  : num  854 686 518 826 504 504 574 504 532 924 ...
 $ price_comp    : num  0.436 0.434 0.415 0.468 0.495 ...
 $ price_own     : num  0.619 0.602 0.604 0.655 0.687 ...
 $ promotion_comp: num  0.0913 0.2107 0.4246 0 0 ...
 $ promotion_own : num  0.145 0.293 0.748 0.293 0 ...
 $ adstock_comp  : num  17.3 17.1 17.2 17.1 14.8 ...
 $ adstock_own   : num  18.3 18.4 18.3 18.2 17.5 ...
 $ grp_comp      : num  69.5 153.6 190.6 155.1 16.5 ...
 $ grp_own       : num  356 269 231 210 122 ...
 - attr(*, ".internal.selfref")=<externalptr>
 - attr(*, "sorted")= chr [1:2] "store_code_uc" "week_end"
```

```r
# View a sample of the cleaned data
head(cleaned_data)
```

```
Key: <store_code_uc, week_end>
   store_code_uc   week_end quantity_comp quantity_own price_comp price_own
           <int>     <Date>         <num>        <num>      <num>     <num>
1:          2324 2011-01-01          5104          854  0.4361500 0.6190343
2:          2324 2011-01-08          6288          686  0.4335324 0.6019952
3:          2324 2011-01-15          5096          518  0.4150657 0.6042896
4:          2324 2011-01-22          5196          826  0.4679586 0.6551205
5:          2324 2011-01-29          4118          504  0.4949050 0.6874852
6:          2324 2011-02-05          4826          504  0.4955510 0.7138833
   promotion_comp promotion_own adstock_comp adstock_own  grp_comp  grp_own
            <num>         <num>        <num>       <num>     <num>    <num>
1:     0.09126372     0.1450584     17.27633    18.31968  69.49349 356.2129
2:     0.21074917     0.2934095     17.13459    18.42195 153.64872 268.9333
```

```
3:      0.42459705      0.7484034      17.24981     18.34036 190.63657 230.5960
4:      0.00000000      0.2934095      17.12539     18.19162 155.10357 210.3191
5:      0.00000000      0.0000000      14.84921     17.54406  16.48662 121.7219
6:      0.00000000      0.0000000      15.56772     17.41667 175.49243 169.0049
```

3.6 Time fixed effects

Create an index for each month/year combination in the data using the following code:

```r
library(lubridate)

# Ensure the 'week_end' column is in Date format
cleaned_data[, week_end := as.Date(week_end)]

# Create a month index based on the year and month of the 'week_end' column
cleaned_data[, month_index := 12 * (year(week_end) - 2011) + month(week_end)]

# View a sample of the cleaned dataset with the new month_index
head(cleaned_data)
```

```
Key: <store_code_uc>
   store_code_uc   week_end quantity_comp quantity_own price_comp price_own
           <int>     <Date>         <num>        <num>      <num>     <num>
1:          2324 2011-01-01          5104          854  0.4361500 0.6190343
2:          2324 2011-01-08          6288          686  0.4335324 0.6019952
3:          2324 2011-01-15          5096          518  0.4150657 0.6042896
4:          2324 2011-01-22          5196          826  0.4679586 0.6551205
5:          2324 2011-01-29          4118          504  0.4949050 0.6874852
6:          2324 2011-02-05          4826          504  0.4955510 0.7138833
   promotion_comp promotion_own adstock_comp adstock_own  grp_comp  grp_own
            <num>         <num>        <num>       <num>     <num>    <num>
1:     0.09126372     0.1450584     17.27633    18.31968  69.49349 356.2129
2:     0.21074917     0.2934095     17.13459    18.42195 153.64872 268.9333
3:     0.42459705     0.7484034     17.24981    18.34036 190.63657 230.5960
4:     0.00000000     0.2934095     17.12539    18.19162 155.10357 210.3191
5:     0.00000000     0.0000000     14.84921    17.54406  16.48662 121.7219
6:     0.00000000     0.0000000     15.56772    17.41667 175.49243 169.0049
   month_index
         <num>
1:           1
2:           1
3:           1
4:           1
5:           1
6:           2
```

```r
# Verify the range of month_index values
summary(cleaned_data$month_index)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   1.00   12.00   24.00   24.42   36.00   48.00
```

```r
# Check for a sample of the month_index column alongside the week_end column
cleaned_data[, .(week_end, month_index)][1:10]
```

```
      week_end month_index
        <Date>       <num>
 1: 2011-01-01           1
 2: 2011-01-08           1
 3: 2011-01-15           1
 4: 2011-01-22           1
 5: 2011-01-29           1
 6: 2011-02-05           2
 7: 2011-02-12           2
 8: 2011-02-19           2
 9: 2011-02-26           2
10: 2011-03-05           3
```

4 Data inspection

4.1 Time-series of advertising levels

We now take a look at the advertising data. First, pick a DMA. You can easily get a list of all DMA names and codes from the `stores` table. I picked `"CHICAGO IL"`, which corresponds to `dma_code` 602. Then plot the time-series of weekly GRPs for your chosen market, separately for the own and competitor brand.

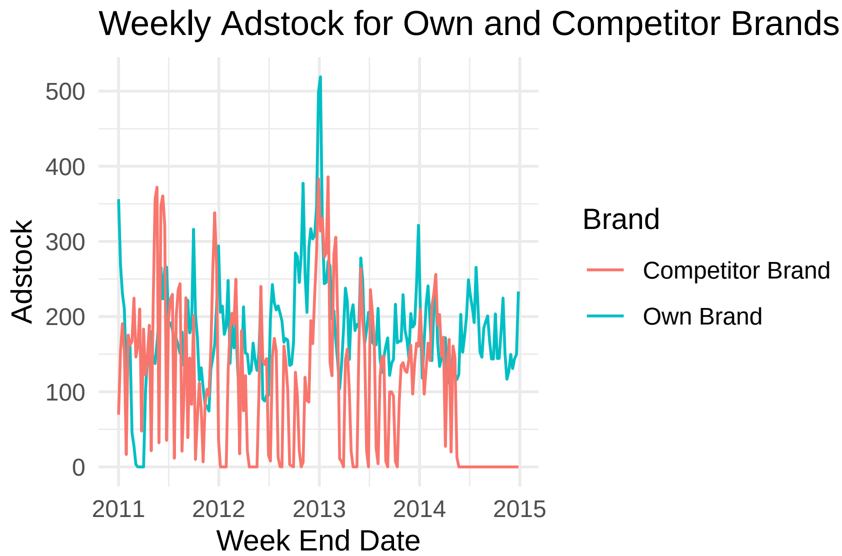
Note: I suggest you create a facet plot to display the time-series of GRPs for the two brands. Use the `facet_grid` or `facet_wrap` layer as explained in the ggplot2 guide (see "More on facetting").

```r
# check store 602
unique(stores_dma[, .(dma_code, dma_descr)])
```

```
     dma_code    dma_descr
        <int>       <char>
  1:      510 CLEVELAND OH
  2:      516      ERIE PA
  3:      602   CHICAGO IL
  4:      501  NEW YORK NY
  5:      505   DETROIT MI
 ---
201:      734 JONESBORO AR
202:      638 ST JOSEPH MO
203:      596 ZANESVILLE OH
204:      558      LIMA OH
205:      626  VICTORIA TX
```

```r
merged_dma_data <- merge(cleaned_data, stores_dma, by = "store_code_uc", all.x = TRUE)
# Filter the data for the selected DMA (Chicago with dma_code 602)
chicago_data <- merged_dma_data[dma_code == 602, ]
```

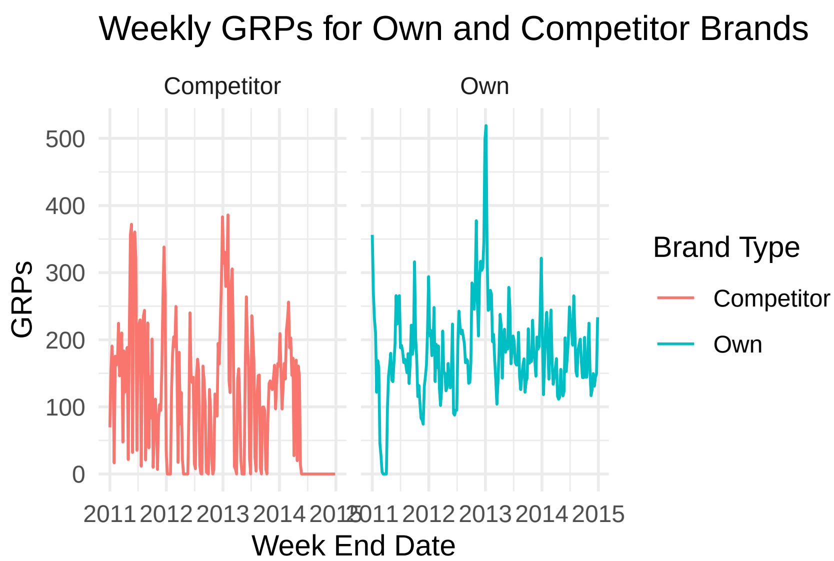
```r
# Plot time-series of weekly adstock for 'own' and 'comp'
ggplot(chicago_data, aes(x = week_end)) +
  geom_line(aes(y = grp_own, color = "Own Brand")) +
  geom_line(aes(y = grp_comp, color = "Competitor Brand")) +
  labs(
    title = "Weekly Adstock for Own and Competitor Brands",
    x = "Week End Date",
    y = "Adstock",
    color = "Brand"
  ) +
  theme_minimal()
```




```r
long_data <- melt(
  chicago_data,
  id.vars = c("store_code_uc", "week_end", "dma_code", "dma_descr", "month_index"),
  measure.vars = list(grp = c("grp_comp", "grp_own")),
  variable.name = "brand_type",
  value.name = "grp"
)
```

```
Warning in melt.data.table(chicago_data, id.vars = c("store_code_uc",
"week_end", : measure.vars is a list with length=1, which as long documented
should return integer indices in the 'variable' column, but currently returns
character column names. To increase consistency in the next release, we plan to
change 'variable' to integer, so users who were relying on this behavior should
change measure.vars=list('col_name') (output variable is column name now, but
will become column index/integer) to measure.vars='col_name' (variable is
column name before and after the planned change).
```

```r
long_data[, brand_type := ifelse(brand_type == "grp_comp", "Competitor", "Own")]

ggplot(long_data, aes(x = week_end, y = grp, color = brand_type)) +
  geom_line() +
  labs(
    title = "Weekly GRPs for Own and Competitor Brands",
    x = "Week End Date",
    y = "GRPs",
    color = "Brand Type"
  ) +
  facet_wrap(~ brand_type) +  # Facet by brand type (Own vs Competitor)
  theme_minimal()
```




```
# Calculate the DMA-level mean of grp and create normalized_grp
chicago_data <- chicago_data %>%
  group_by(dma_code) %>%
  mutate(mean_grp = mean(c(grp_own, grp_comp), na.rm = TRUE)) %>%
  ungroup() %>%
  mutate(
    normalized_grp_own = 100 * grp_own / mean_grp,
    normalized_grp_comp = 100 * grp_comp / mean_grp
  )
```

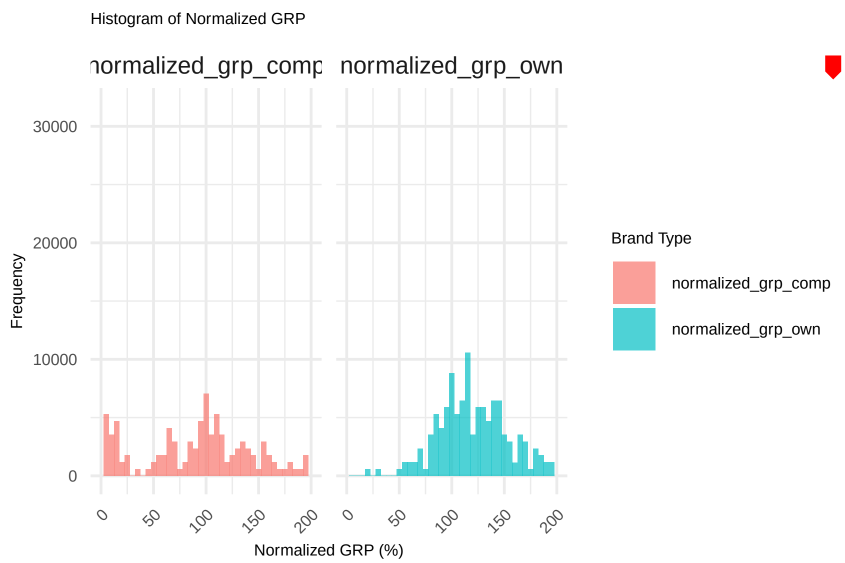
```
# Reshape to long format for plotting both variables
library(tidyverse)

long_normalized <- chicago_data %>%
  select(dma_code, normalized_grp_own, normalized_grp_comp) %>%
  pivot_longer(cols = starts_with("normalized_grp"), names_to = "brand_type",
               values_to = "normalized_grp")
```

```
# Plot histogram of normalized_grp for both own and competitor
ggplot(long_normalized, aes(x = normalized_grp, fill = brand_type)) +
  geom_histogram(binwidth = 5, alpha = 0.7, position = "identity") +
  labs(
    title = "Histogram of Normalized GRP",
    x = "Normalized GRP (%)",
    y = "Frequency",
    fill = "Brand Type"
  ) +
  scale_x_continuous(limits = c(0, 200)) +  # Set x-axis limits to exclude extreme values
  facet_wrap(~ brand_type) +
  theme_minimal() +
  theme(
    plot.title = element_text(size = 6),
    axis.title.x = element_text(size = 6),
    axis.title.y = element_text(size = 6),
```

```
    axis.text.x = element_text(size = 6, angle = 45, hjust = 1),
    axis.text.y = element_text(size = 6),
    legend.title = element_text(size = 6),
    legend.text = element_text(size = 6)
  )
```

```
Warning: Removed 14112 rows containing non-finite outside the scale range
(`stat_bin()`).
```

```
Warning: Removed 4 rows containing missing values or values outside the scale range
(`geom_bar()`).
```



4.2 Overall advertising variation

Create a new variable **at the DMA-level**, `normalized_grp`, defined as `100*grp/mean(grp)`. This variable captures the percentage deviation of the GRP observations relative to the DMA-level mean of advertising. Plot a histogram of `normalized_grp`.

Note: To visualize the data you should use the `scale_x_continuous` layer to set the axis `limits`. This data set is one of many examples where some extreme outliers distort the graph.

5 Advertising effect estimation

Estimate the following specifications:

1. Base specification that uses the log of `1+quantity` as output and the log of prices (own and competitor) and promotions as inputs. Control for store and month/year fixed effects.

```
# Load necessary library
library(fixest)

# Base specification model
base_model <- feols(
  log(1 + quantity_own) ~ log(price_own) + log(price_comp) + promotion_own |
    store_code_uc + month_index,  # Fixed effects for store and month/year
  data = chicago_data
)

# Display summary of the model results
summary(base_model)
```

```
OLS estimation, Dep. Var.: log(1 + quantity_own)
Observations: 122,844
Fixed-effects: store_code_uc: 588,  month_index: 48
Standard-errors: Clustered (store_code_uc)
                 Estimate Std. Error   t value  Pr(>|t|)
log(price_own)  -2.639497   0.153736 -17.16899 < 2.2e-16 ***
log(price_comp)  0.564884   0.407962   1.38465   0.16669
promotion_own    0.702507   0.027710  25.35213 < 2.2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
RMSE: 0.766907     Adj. R2: 0.593712
                 Within R2: 0.083416
```

2. Add the `adstock` (own and competitor) to specification 1.

```
# Specification with adstock
model_with_adstock <- feols(
  log(1 + quantity_own) ~ log(price_own) + log(price_comp) + promotion_own +
    adstock_own + adstock_comp |
    store_code_uc + month_index,  # Fixed effects for store and month/year
  data = chicago_data
)

# Display summary of the model results
summary(model_with_adstock)
```

```
OLS estimation, Dep. Var.: log(1 + quantity_own)
Observations: 122,844
Fixed-effects: store_code_uc: 588,  month_index: 48
Standard-errors: Clustered (store_code_uc)
                 Estimate Std. Error   t value   Pr(>|t|)
log(price_own)  -2.636109   0.153536 -17.16927  < 2.2e-16 ***
```

```
log(price_comp)  0.552326   0.409090   1.35013 1.7749e-01
promotion_own    0.700730   0.027713  25.28541  < 2.2e-16 ***
adstock_own      0.019531   0.002578   7.57537 1.4003e-13 ***
adstock_comp     0.001966   0.001371   1.43370 1.5219e-01
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
RMSE: 0.766749     Adj. R2: 0.593873
                 Within R2: 0.083793
```

3. Like specification 2., but not controlling for time fixed effects.

```
model_without_time_fixed_effects <- feols(
  log(1 + quantity_own) ~ log(price_own) + log(price_comp) + promotion_own + adstock_own +
    adstock_comp |
    store_code_uc,  # Fixed effects for store only
  data = chicago_data
)

# Display summary of the model results
summary(model_without_time_fixed_effects)
```

```
OLS estimation, Dep. Var.: log(1 + quantity_own)
Observations: 122,844
Fixed-effects: store_code_uc: 588
Standard-errors: Clustered (store_code_uc)
                 Estimate Std. Error    t value   Pr(>|t|)
log(price_own)  -1.666218   0.150248 -11.089798  < 2.2e-16 ***
log(price_comp)  0.392515   0.401368   0.977942 3.2851e-01
promotion_own    0.816996   0.029052  28.121422  < 2.2e-16 ***
adstock_own     -0.012863   0.001926  -6.680128 5.5504e-11 ***
adstock_comp     0.020878   0.000800  26.092908  < 2.2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
RMSE: 0.780536     Adj. R2: 0.579298
                 Within R2: 0.094045
```

Combine the results using `etable` and comment on the results.

```
etable(base_model, model_with_adstock, model_without_time_fixed_effects)
```

```
                         base_model  model_with_adstock model_without_tim..
Dependent Var.: log(1+quantity_own) log(1+quantity_own) log(1+quantity_own)

log(price_own)   -2.639*** (0.1537)  -2.636*** (0.1535)  -1.666*** (0.1502)
log(price_comp)     0.5649 (0.4080)     0.5523 (0.4091)     0.3925 (0.4014)
promotion_own    0.7025*** (0.0277)  0.7007*** (0.0277)  0.8170*** (0.0290)
adstock_own                          0.0195*** (0.0026) -0.0129*** (0.0019)
adstock_comp                            0.0020 (0.0014)  0.0209*** (0.0008)
Fixed-Effects:  ------------------- ------------------- -------------------
store_code_uc                   Yes                 Yes                 Yes
month_index                     Yes                 Yes                  No

--------------- ------------------- ------------------- -------------------
```

```
S.E.: Clustered   by: store_code_uc   by: store_code_uc   by: store_code_uc
Observations              122,844             122,844             122,844
R2                        0.59582             0.59599             0.58133
Within R2                 0.08342             0.08379             0.09405
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Examining on the weekly GRP graph, both brands experience significant fluctuations in adstock levels over time, which is expected given that advertising campaigns usually vary in intensity. In general, our brand maintains a higher adstock than our competitor brands, indicating that our brand is more likely to impact consumers' demand overtime. Specifically, around 2013, both our brand and competitor brands are experiencing a peak in adstock, implying a period with intensive advertisement and competition. Compared to competitors' brands, our distribution shape of GRP is more centered shown by the histogram, indicating that the our advertisement is impact customers more constantly which is align with the weekly GRP graph.

For the regression results, the log of our own price is significantly negatively correlated with the log(1+quantity) across all models, showing that decreasing our price is pivotal to boost sales. The negative effect of our own price on sales quantity is reduced when the time effect is not controlled. The effect of price of competitor is not significant across all models, showing that the price of competitors is not affecting our sales. The coefficient for promotion_own is positive and highly significant in all models, indicating that promotions effectively boost demand. Adstock_own is positive and significant in Model 2 (with time fixed effects), indicating that cumulative advertising (adstock) positively influences demand when time effects are controlled. This implies that own advertising has a lingering effect on demand. However, in Model 3 (without time fixed effects), adstock_own becomes negative and significant. The contradiction centers the effect of time trend for our brand. The effect of adstock of our competitor is not significant when we controlled the time effect, but it seems to increase our sales if we are not controlling the time effect. Overall, seasonality is crucial to control in this case if we want to have an accurate report on the effect of our interested variables on the sales quantity. The R-squared is relatively consistent across the models, indicating that the inclusion of adstock and fixed effects adds only a slight improvement for model fit. A R2 value around 60% generally indicates a good fit of model.