Advertising Effects

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

Contents

1 Overview **2**

2 Data **2**
2.1 Brands and product modules . . . 2

3 3 Data preparation **3**
3.1 3.1 RMS scanner data (`move`) . . . 3
3.2 3.2 Ad Intel advertising data (`adv_DT`) . . . 4
3.3 3.3 Calculate adstock/goodwill . . . 7
3.4 Merge scanner and advertising data . . . 9
3.5 Reshape the data . . . 9
3.6 Time fixed effects . . . 10

4 Data inspection **11**
4.1 Time-series of advertising levels . . . 11
4.2 Overall advertising variation . . . 12

5 Advertising effect estimation **14**

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

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 = "data"
```

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.

```r
brands = load(paste0(data_folder, "/Brands_a3.RData"))
```

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.

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

3 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.

```
brands = load(paste0(data_folder, "/Brands_a3.RData"))
stores = load(paste0(data_folder, "/stores_dma.RData"))
move_data = load(paste0(data_folder, "/move_8412.RData"))
advertising_data = load(paste0(data_folder, "/adv_8412.RData"))
```

3.1 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.

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.

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.

```
#Step 1: Renaming (use set names)
#Step 1a: Units -> Quantity
#Step 1b: promo_percentage -> promotion
library(data.table)
setnames(move, old = "units", new = "quantity", skip_absent=TRUE)
setnames(move, old = "promo_percentage", new = "promotion", skip_absent=TRUE)
```

```
#Step 2: Identifying 'own' (Prilosec) vs. 'comp' (non-Prilosec)
move$brand_name = ifelse(move$brand_code_uc == selected_brand, "own", "comp")
colnames(move)
```

```
[1] "brand_code_uc" "store_code_uc" "week_end"      "quantity"
[5] "price"         "promotion"     "brand_name"
```

```
#Step 3: Aggregating the data for each store/observation
#For 'comp'
move <- move[, .(
  quantity = sum(quantity, na.rm = TRUE),       # Sum quantities
  price = mean(price, na.rm = TRUE),            # Mean price
  promotion = mean(promotion, na.rm = TRUE)     # Mean promotion
), by = .(store_code_uc, week_end, brand_name)]
```

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.

```r
#Merge stores_dma w/move
#Can you check this one because I only see one dma value when I merge both together
move = merge(move, stores_dma, by = "store_code_uc")
print(move)
```

```
Key: <store_code_uc>
         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 0.1675660
      2:          2324 2010-01-09       comp     6008 0.4424408 0.0000000
      3:          2324 2010-01-16       comp     5898 0.4399273 0.0000000
      4:          2324 2010-01-23       comp     5508 0.4367092 0.1675660
      5:          2324 2010-01-30       comp     5375 0.4327154 0.1675660
     ---
7009546:       8388364 2014-11-29        own      196 0.7003856 0.8740923
7009547:       8388364 2014-12-06        own      112 0.7003856 0.8740923
7009548:       8388364 2014-12-13        own       56 0.7317053 0.4356169
7009549:       8388364 2014-12-20        own       56 0.7575497 0.1467626
7009550:       8388364 2014-12-27        own       56 0.7712573 0.0000000
         dma_code                         dma_descr
            <int>                            <char>
      1:      602                        CHICAGO IL
      2:      602                        CHICAGO IL
      3:      602                        CHICAGO IL
      4:      602                        CHICAGO IL
      5:      602                        CHICAGO IL
     ---
7009546:      518 GREENSBORO-HIGH POINT-WINSTON SALEM NC
7009547:      518 GREENSBORO-HIGH POINT-WINSTON SALEM NC
7009548:      518 GREENSBORO-HIGH POINT-WINSTON SALEM NC
7009549:      518 GREENSBORO-HIGH POINT-WINSTON SALEM NC
7009550:      518 GREENSBORO-HIGH POINT-WINSTON SALEM NC
```

3.2 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:

```
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.

```
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:

```
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`:

```
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.

```
#Perfoming Cross Join (Double Check)
library(data.table)
setDT(adv_DT)
complete_set = CJ(brand_code_uc = brands, dma_code = dma_codes, week_end = weeks)
adv_DT = complete_set[adv_DT, on = .(brand_code_uc, dma_code, week_end)]
adv_DT[is.na(adv_DT)] = 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`).

```
#Locating own and comps within advertising datatable...
selected_brand  = 621727
adv_DT[, brand_name := ifelse(adv_DT$brand_code_uc == selected_brand, "own", "comp")]
```

```
adv_DT <- adv_DT[, .(
  grp_direct = sum(grp_direct, na.rm = TRUE),       # Sum of direct GRPs
  grp_indirect = sum(grp_indirect, na.rm = TRUE)    # Sum of indirect GRPs
), by = .(dma_code, week_end, brand_name)]
```

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).

```
#Direct & Indirect GRPs
adv_DT$grp = adv_DT$grp_direct + adv_DT$grp_indirect
```

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 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.

```
adv_DT[, grp := grp_direct + grp_indirect]
```

```
setkey(adv_DT, brand_name, dma_code, week_end)

adv_DT[, 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[, stock_a := weightedSum(log(1+grp), geom_weights),
                             by = .(brand_name, dma_code)])
```

```
time_b = system.time(adv_DT[, 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
     19
```

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.

```
head(move)
```

```
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
    dma_descr
       <char>
1: CHICAGO IL
2: CHICAGO IL
3: CHICAGO IL
4: CHICAGO IL
5: CHICAGO IL
6: CHICAGO IL
```

```
adv_DT[, stock_a := NULL]
adv_DT[, stock_b := NULL]
move <- merge(move, adv_DT, by = c("brand_name", "dma_code", "week_end"))
head(move)
```

```
Key: <brand_name, dma_code, week_end>
   brand_name dma_code   week_end store_code_uc quantity     price  promotion
       <char>    <int>     <Date>         <int>    <num>     <num>      <num>
1:       comp      500 2010-01-02         88153     1432 0.5957322 0.00000000
2:       comp      500 2010-01-02         95752     1946 0.5644275 0.08261606
3:       comp      500 2010-01-02        123214     1592 0.6173410 0.00000000
4:       comp      500 2010-01-02        129685     1146 0.5777817 0.20306278
5:       comp      500 2010-01-02        189538     1698 0.5948342 0.00000000
6:       comp      500 2010-01-02        366762     4388 0.4063640 0.23904801
            dma_descr grp_direct grp_indirect      grp adstock
               <char>      <num>        <num>    <num>   <num>
1: PORTLAND-AUBURN ME   211.2821            0 211.2821      NA
2: PORTLAND-AUBURN ME   211.2821            0 211.2821      NA
3: PORTLAND-AUBURN ME   211.2821            0 211.2821      NA
4: PORTLAND-AUBURN ME   211.2821            0 211.2821      NA
5: PORTLAND-AUBURN ME   211.2821            0 211.2821      NA
6: PORTLAND-AUBURN ME   211.2821            0 211.2821      NA
```

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`).

```
wide_data <- dcast(move, dma_code + store_code_uc + week_end ~ brand_name, value.var = c("quantity", "p
wide_data <- wide_data[complete.cases(wide_data), ]
```

3.6 Time fixed effects

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

```
wide_data[, week_end := as.Date(week_end, format = "%Y-%m-%d")]
wide_data[, month_index := 12 * (as.integer(format(week_end, "%Y")) - 2011) + as.integer(format(week_en
head(wide_data[, .(week_end, month_index)]) #sanity check
```

```
     week_end month_index
       <Date>       <num>
1: 2011-02-05           2
2: 2011-02-12           2
3: 2011-02-19           2
4: 2011-02-26           2
5: 2011-03-05           3
6: 2011-04-09           4
```

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.

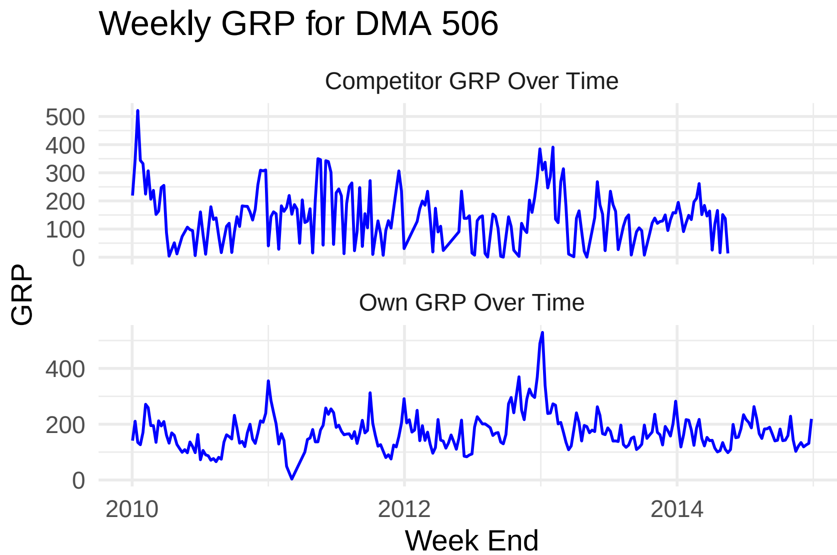
```
chosen_code = 506
move_filtered <- move[dma_code == chosen_code]
head(move_filtered)
```

```
Key: <brand_name, dma_code, week_end>
   brand_name dma_code   week_end store_code_uc quantity     price promotion
       <char>    <int>     <Date>         <int>    <num>     <num>     <num>
1:       comp      506 2010-01-02         12869      230 0.4862497 0.0000000
2:       comp      506 2010-01-02         18073     4660 0.5466057 0.0000000
3:       comp      506 2010-01-02         69801     3570 0.5652718 0.3868085
4:       comp      506 2010-01-02         73428     1036 0.6423019 0.4711488
5:       comp      506 2010-01-02        202709     2260 0.5431721 0.0000000
6:       comp      506 2010-01-02        208754      730 0.6484600 0.0000000
                    dma_descr grp_direct grp_indirect      grp adstock
                       <char>      <num>        <num>    <num>   <num>
1: BOSTON (MANCHESTER) MA-NH   219.0284            0 219.0284      NA
2: BOSTON (MANCHESTER) MA-NH   219.0284            0 219.0284      NA
3: BOSTON (MANCHESTER) MA-NH   219.0284            0 219.0284      NA
4: BOSTON (MANCHESTER) MA-NH   219.0284            0 219.0284      NA
5: BOSTON (MANCHESTER) MA-NH   219.0284            0 219.0284      NA
6: BOSTON (MANCHESTER) MA-NH   219.0284            0 219.0284      NA
```

```
# Aggregate grp by week_end and brand
move_filtered <- move_filtered[, .(
  grp = mean(grp, na.rm = TRUE)
), by = .(week_end, brand_name)]
```

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").

```
library(ggplot2)

# Create the facet plot
facet_labels <- c("own" = "Own GRP Over Time", "comp" = "Competitor GRP Over Time")

ggplot(move_filtered, aes(x = week_end, y = grp)) +
  geom_line(color = "blue") +
  facet_wrap(~ brand_name, ncol = 1, scales = "free_y",
             labeller = labeller(brand_name = facet_labels)) +
  labs(
    title = paste("Weekly GRP for DMA", chosen_code),
    x = "Week End",
    y = "GRP"
  ) +
  theme_minimal()
```

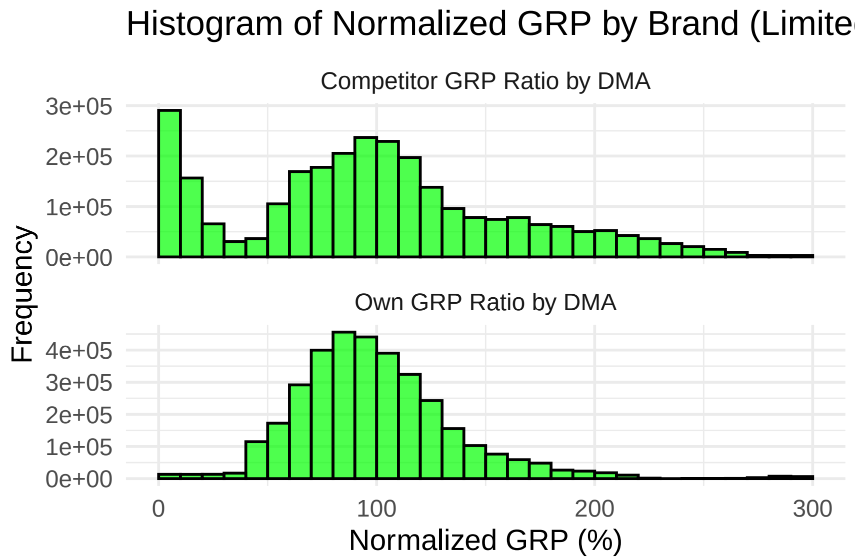


4.2 Overall advertising variation

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`.

```
mean_grp_dma_brand <- move[, .(mean_grp = mean(grp, na.rm = TRUE)), by = .(dma_code, brand_name)]

# Step 2: Merge the mean back into the 'move' data table and calculate 'normalized_grp'
move <- merge(move, mean_grp_dma_brand, by = c("dma_code", "brand_name"), all.x = TRUE)
move[, normalized_grp := 100 * grp / mean_grp]

facet_labels <- c("own" = "Own GRP Ratio by DMA", "comp" = "Competitor GRP Ratio by DMA")

# Set limits to exclude outliers
lower_limit <- 0
upper_limit <- 300

# Plot the histogram using 'move' without modifying it
ggplot(move, aes(x = normalized_grp)) +
  geom_histogram(binwidth = 10, fill = "green", color = "black", alpha = 0.7, boundary = 0) +
  facet_wrap(~ brand_name, ncol = 1, scales = "free_y",
             labeller = as_labeller(facet_labels)) +
  scale_x_continuous(limits = c(lower_limit, upper_limit)) +  # Or use coord_cartesian()
  labs(
    title = "Histogram of Normalized GRP by Brand (Limited Range)",
    x = "Normalized GRP (%)",
    y = "Frequency"
  ) +
  theme_minimal()
```

```
Warning: Removed 23750 rows containing non-finite outside the scale range
(`stat_bin()`).
```



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.
2. Add the `adstock` (own and competitor) to specification 1.
3. Like specification 2., but not controlling for time fixed effects.

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

```
head(wide_data)
```

```
Key: <dma_code, store_code_uc>
   dma_code store_code_uc   week_end quantity_comp quantity_own price_comp
      <int>         <int>     <Date>         <num>        <num>      <num>
1:      500         88153 2011-02-05           620          532  0.5834591
2:      500         88153 2011-02-12           870          210  0.5834591
3:      500         88153 2011-02-19          1432          238  0.5794119
4:      500         88153 2011-02-26           862          154  0.5860128
5:      500         88153 2011-03-05          1880          350  0.5860128
6:      500         88153 2011-04-09           762          210  0.5732326
   price_own promotion_comp promotion_own adstock_comp adstock_own month_index
       <num>          <num>         <num>        <num>       <num>       <num>
1: 0.6667948              0             0     15.28524    17.20203           2
2: 0.6667948              0             0     16.10376    16.93872           2
3: 0.6667948              0             0     16.28562    15.65049           2
4: 0.6667948              0             0     16.55701    14.50100           2
5: 0.6667948              0             0     16.84442    11.60743           3
6: 0.6667948              0             0     16.35107    12.84814           4
```

```
library(fixest)
basic_model <- feols(
  log(1 + quantity_own) ~ log(price_own) + log(price_comp) + promotion_own + promotion_comp |
    store_code_uc + month_index,
  data = wide_data
)

model_with_adstock <- feols(
  log(1 + quantity_own) ~ log(price_own) + log(price_comp) + promotion_own + promotion_comp + adstock_o
    store_code_uc + month_index,
  data = wide_data
)

model_with_adstock_no_time <- feols(
  log(1 + quantity_own) ~ log(price_own) + log(price_comp) + promotion_own + promotion_comp + adstock_o
    store_code_uc,
  data = wide_data
)


etable(basic_model, model_with_adstock, model_with_adstock_no_time)
```

```
                          basic_model  model_with_adstock model_with_adstoc..
Dependent Var.: log(1+quantity_own) log(1+quantity_own) log(1+quantity_own)
```

```
log(price_own)   -2.116*** (0.0819)  -2.113*** (0.0817)  -1.913*** (0.0770)
log(price_comp)    0.1436* (0.0579)    0.1466* (0.0578) -0.4477*** (0.0787)
promotion_own    0.8782*** (0.0116)  0.8773*** (0.0116)  0.9286*** (0.0109)
promotion_comp    0.0346** (0.0113)   0.0349** (0.0113)     0.0075 (0.0147)
adstock_own                          0.0307*** (0.0017) -0.0182*** (0.0008)
adstock_comp                        -0.0036*** (0.0007)  0.0113*** (0.0004)
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              2,005,858           2,005,858           2,005,858
R2                          0.64314             0.64321             0.63844
Within R2                   0.07012             0.07030             0.07444
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The results from the basic model shows predominantly expected results. We see a negative correlation between own price and quantity sold, with a 2.1% decrease in quantity with each percentage point increase in the price of own products. There is a slight positive correlation between competitor price and own quantity sold, with own quantity going up by 0.14% for each percentage point competitors raise their prices. We see a positive correlation between own promotions and quantity, with the dummy variable having a coefficient of 0.8782. Slightly surprisingly, we also see a positive coefficient associated with competitor promotions (0.0346). This could be explained by a phenomenon where competitors' promotions draw some additional attention to own products as well, or causing customers to realize their need for a product within the category. This category spill-over means that there is a boost in sales of own products when competitors run a promotion on their own products.

The results from the adstock model with time fixed-effects show similar results to the basic model. It shows a slightly weaker negative correlation between own price and quantity sold, and a slightly stronger positive correlation between competitors' pricing and own quantity sold. The effects of running promotions is also similar to the basic model, with results showing a large positive effect of own promotions on quantities sold, and a small, but still positive effect of competitors running promotions. The adstock model also includes on the adstock, or the weighted average of current and past advertising, and these coefficients show the effects running an advertising campaign on the quantities sold. We can see that there is a slight positive correlation in own products sold when running an advertising campaign, while during and after a competitor's advertising, we see a decrease in the quantity of our own products sold.

In the model where we do not control for time fixed effects, our results deviate significantly from those in the previous models. While the coefficient on own price is still negative, indicating that an increase in the price of our products leads to a decrease in the number of units sold, the effect of competitors' price also has a negative coefficient. This would mean that an increase in a competing product's price lead to a decrease in sales for our own product as well. This is likely due to the model possibly overestimating the cross-price elasticity when time fixed effects are excluded. The effects of running promotions does not vary much in this model compared to the previous ones, though we do see that the coefficient on competitors' promotions is not statistically significant, indicating that the effect of a competitor running a promotion does not significantly change the sales metrics of our own product. The effect of adstock in this model is the reverse of what we observed in the model containing time fixed effects. Our results show a negative correlation between own advertising efforts and the quantity of own products sold, while it indicates a positive correlation between competitors' ad campaigns and our own products. This logically does not make sense and highlights the importance of including fixed effects into the regression model whenever possible.

Controlling for time fixed effects results in a higher coefficient in the effect of own adstock on the quantity of own product sold when compared to the model without time fixed effects. From this, we can conclude that the company is focusing on advertising in markets where they believe they are not as strong as their competitors. The effect of competitor adstock, however, increases in the model without time fixed effects, which is in line with what we would expect when we do not control for time effects when the competition is focusing its advertising in markets where they already have a strong presence.