Homework Four

Akshay Tandon, Nathan Matare, Linda Xie

2/8/2017

RMS Scanner Data

We prepare the RMS scanner data by calculating average price, promototion and total quantity by brand name, store code, and week-end.

```
move[brand_code_uc == selected_brand, brand_name := 'own']
move[brand_code_uc != selected_brand, brand_name := 'comp']
move[ ,brand_code_uc := NULL] # no longer needed
setkey(move, brand_name, store_code_uc, week_end)

move[ ,':='(price = mean(price), promotion = mean(promotion), quantity = sum(quantity)), by = .(bran
move <- unique(move[ ,.(brand_name, store_code_uc, week_end, price, promotion, quantity)]) # remove

setkey(stores, store_code_uc)
stores_dma <- unique(stores[, .(store_code_uc, dma_code)])
move <- merge(move, stores_dma) # uniquely merge dma_code with move data
```

Cross Join

Next we cross-join the adv_DT data.table and impute the NAs

```
adv_DT <- adv_DT[CJ(brands, dma_codes, weeks)]
adv_DT[is.na(adv_DT)] <- 0
```

Create Group Data

Finally, we average the national group data, sum the local group data, and create a 'grp' variable.

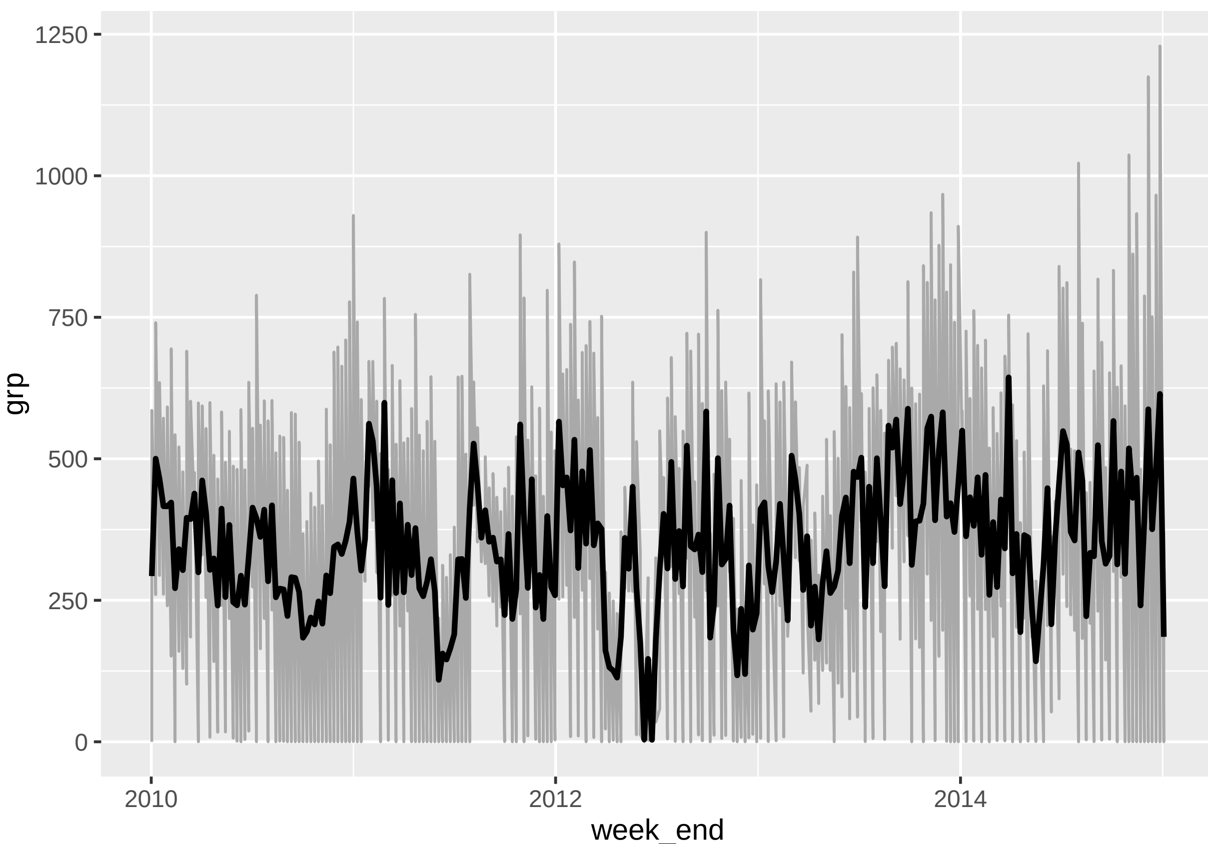
```
adv_DT[ ,':='(  national_grp = sum(national_grp), local_grp = sum(local_grp),
                grp = sum(national_grp, local_grp)), by = .(brand_name, dma_code, week_start)] # me
adv_DT[ ,local_occ := NULL]
adv_DT <- unique(adv_DT[ ,.(brand_name, dma_code, week_start, grp)]) # remove duplicated comptitors
```

Plots

We select Chicago IL at code 602.

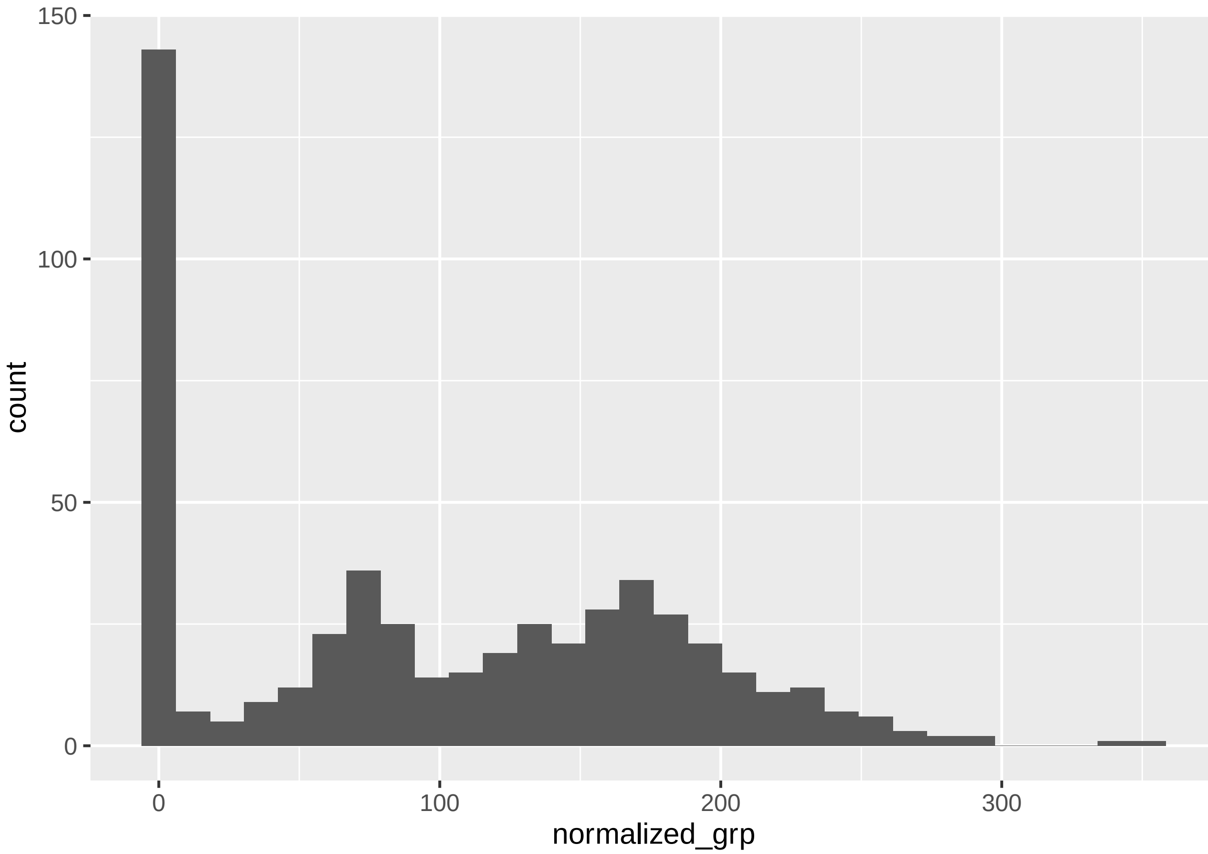
When inspecting the group level data for Chicago IL, we observe an oscillating pattern. There appears to be a mean-reverting trend centered around 300. Notably, in mid-2012 advertising levels drop significantly before returning to the mean-level; afterwards the oscillation is less apparent. Perhaps a change in strategy altered the advertising levels.

```
code = 602
name = 'CHIAGO IL'
ggplot(data = adv_DT[dma_code == code], aes(y = grp, x = week_end)) + geom_line(color = 'darkgrey')
  stat_summary(fun.y=mean,geom="line", lwd=1, aes(group=1)) # plot
```




```
adv_DT[ ,normalized_grp := 100 * grp / mean(grp), by = .(dma_code)]      # normalize
ggplot(data = adv_DT[dma_code == code], aes(normalized_grp)) + geom_histogram()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



We observe a bi-modal distribution. After normalization, the advertising levels for Chicago appear to be clustered around 80 and 180. There is also a large number of zeros. The bi-modal distribution ostensibly mirrors the change in time-series data from the previous plot. That is, the event in mid-2012 may have altered advertising levels so as to produce two data-generating processes.

Advertising Effect Estimation

```r
fit_base = felm(log(1+quantity_own) ~ log(price_own) + log(price_comp) + promotion_own + # Base spec
                  promotion_comp | as.factor(store_code_uc) + month_index, data = move)

fit_adstock = felm(log(1+quantity_own) ~ log(price_own) + log(price_comp) + promotion_own +
                     promotion_comp + adstock_own + adstock_comp | as.factor(store_code_uc) +
                     month_index, data = move) # Ad Stock

fit_adstock_noFE = felm(log(1+quantity_own) ~ log(price_own) + log(price_comp) + promotion_own +
                          promotion_comp + adstock_own + adstock_comp | as.factor(store_code_uc),
                        data = move) # Ad Stock w/o Fixed Effect

#Stargazer
stargazer(fit_base, fit_adstock, fit_adstock_noFE,
          column.labels = c("Base", "Ad Stock", "Ad Stock no Time FE"),
          type = "text", dep.var.labels.include = FALSE)
```

```
##
## ===================================================================================
##                                           Dependent variable:
##                     ---------------------------------------------------------------
##                              Base                 Ad Stock        Ad Stock no Time FE
##                              (1)                    (2)                   (3)
## -----------------------------------------------------------------------------------
## log(price_own)            -1.365***              -1.365***             -1.221***
##                            (0.004)                (0.004)               (0.004)
##
## log(price_comp)            0.798***               0.798***              0.675***
##                            (0.006)                (0.006)               (0.006)
##
## promotion_own              0.865***               0.865***              0.896***
##                            (0.001)                (0.001)               (0.001)
##
## promotion_comp             0.021***               0.020***             -0.022***
##                            (0.002)                (0.002)               (0.002)
##
## adstock_own                                       0.002***              0.015***
##                                                   (0.0003)              (0.0001)
##
## adstock_comp                                     -0.002***              0.004***
##                                                   (0.0001)             (0.00003)
##
## -----------------------------------------------------------------------------------
## Observations              3,308,681              3,308,681             3,308,681
## R2                          0.872                  0.872                 0.870
## Adjusted R2                 0.871                  0.871                 0.869
## Residual Std. Error 0.527 (df = 3292121) 0.527 (df = 3292119) 0.531 (df = 3292166)
## ===================================================================================
## Note:                                               *p<0.1; **p<0.05; ***p<0.01
```

After fitting the base model, we observe four highly significant coefficients; indicating that own/comp price and own/comp promotion do, in fact, impact own_demand.

Intuitively, the inverse sign relationship between log(price_own) and log(price_comp) makes sense. For example consider a 50% increase in price_own versus price_comp a 50% increase in price_own suggests a 68.25 decrease in own_demand (-1.365 * 50) whereas a 50% increase in price_comp suggests a 39.9 increase in own_demand (0.798 * 50). Given basic economics, one would expect that as one lowers own prices, ceteris paribus, demand should increase.

A similar analysis follows for promotion_own; that is, promoting one owns product naturally increases demand. However, when inspecting the relative size of the coefficients, we observe a small promotion_comp relative to promotion_own. It appears that, although competitor promotion is significant, the effect is rather small (0.021)

After controlling for adstock_own and adstock_comp our coefficients do not change dramatically. Interestingly, however, is the minuscule size of the coefficients on adstock own/comp. We would expect advertising to work; that is, yield large coefficients! What we observe is significant but tiny coefficients. And although the intuition remains, a negative adstock_comp decreases own_demand; the effect is too minuscule to potentially warrant participation (0.002).

Tellingly, whatever effect advertising does have on own_demand; it seems largely explained by our fixed time effect. That is, when we stop controlling for time-fixed effects, advertising coefficients change in magnitude—suggesting an omitted variable bias. Our price_own and price_comp coefficients change as well.

Border Strategy

```
stores[, border_name := as.factor(border_name)]
setkey(move, store_code_uc, dma_code, week_end)

move = merge(move, stores[on_border == TRUE, .(store_code_uc, border_name)], allow.cartesian = TRUE)
move[, month_index := as.numeric(month_index)] #Convert index to numeric for below analysis

fit_border = felm(log(1+quantity_own) ~ log(price_own) + log(price_comp) + promotion_own +  #Advertisin
                    promotion_comp | as.factor(store_code_uc) + border_name:month_index, data = move)

fit_border_SE = felm(log(1+quantity_own) ~ log(price_own) + log(price_comp) + promotion_own + #Advertis
                       promotion_comp | as.factor(store_code_uc) + border_name:month_index | 0 |
                       dma_code, data = move)

#Stargazer
stargazer(fit_border, fit_border_SE,
          column.labels = c("Border", "Border w SE Cluster"),
          type = "text", dep.var.labels.include = FALSE)
```

```
##
## ===============================================================
##                                      Dependent variable:
##                                 -----------------------------
##                                  Border   Border w SE Cluster
##                                    (1)            (2)
## ---------------------------------------------------------------
## log(price_own)                  -2.386***      -2.386***
##                                  (0.213)        (0.621)
##
## log(price_comp)                  0.565**         0.565
##                                  (0.276)        (1.016)
##
## promotion_own                   0.610***        0.610**
##                                  (0.066)        (0.282)
##
## promotion_comp                   -0.105         -0.105
##                                  (0.094)        (0.139)
##
## ---------------------------------------------------------------
## Observations                      2,299          2,299
## R2                                0.850          0.850
## Adjusted R2                       0.849          0.849
## Residual Std. Error (df = 2281)   0.605          0.605
## ===============================================================
## Note:                             *p<0.1; **p<0.05; ***p<0.01
```

Interestingly, when controlling for border effects, both price_own and price_comp increase significantly in magnitude; border stores are highly susceptible to price changes! However, our standard errors increase significantly when compared to the base model. It would appear that border stores, on average, do help explain the variation in own_demand, but the extent to which is non-uniform—indicative of the larger standard-errors. Naturally, promotion_own decreases and similarly displays larger stand-errors. Here, competitor competition is insignificant. Finally, our coefficients stay the same when controlling for standard-error clustering, but the clustering control increases the relative size of our standard errors—making price_comp completely insignificant.