Data Analysis to Support Paper "Identifiying Consumer Welfare Changes when Online Search Platforms Change their List of Search Results"

Ryan Martin

May 21, 2019

1 Initial Setup and Refining the Data from Kaggle to Largest Market

This software program uses the R package tidyverse. If it is not already installed in your workspace, remove the comment (the # sign) from the first line and run it before running the rest of the code. Also, set `my_folder` in the 3rd line of code to the location you saved the `SearchListingWelfareSoftware` folder.

This first code chunk takes the raw Kaggle data (`train.csv`) as input and refines it to the data's most-searched US market. The final output is labeled `market1search.csv` and is included as part of the software package. That is, this first code chunk may be skipped with no consequence to the rest of the software and is only included for completeness of replicability.

```
#install.packages("tidyverse")
library(tidyverse)
#my_folder <- "/Path/To/Directory/SearchListingWelfareSoftware"

my_file <- paste(my_folder, "train.csv", sep = "/")
dat <- read.csv(my_file) #may take several minutes
# dim(dat) #9917530 by 54
dest_table <- table(dat$srch_destination_id)
head(sort(dest_table, decreasing=TRUE), n = 20)

# reduce to US data
country_count <- table(dat$prop_country_id)
US_marker = names(which(country_count == max(country_count)))
dat_US <- dat[dat$prop_country_id == US_marker,]
rm(dat)

head(sort(table(dat_US$srch_destination_id), decreasing=TRUE), n = 30)
major_market <- as.integer(names(head(sort(table(dat_US$srch_destination_id),
  decreasing=TRUE), n = 1)))
major_market

datmarket1 <-  dat_US[dat_US$srch_destination_id== major_market,]
write_csv(datmarket1,path =
  paste(my_folder,"market1search.csv", sep = "/"))
```

2 Data Cleaning

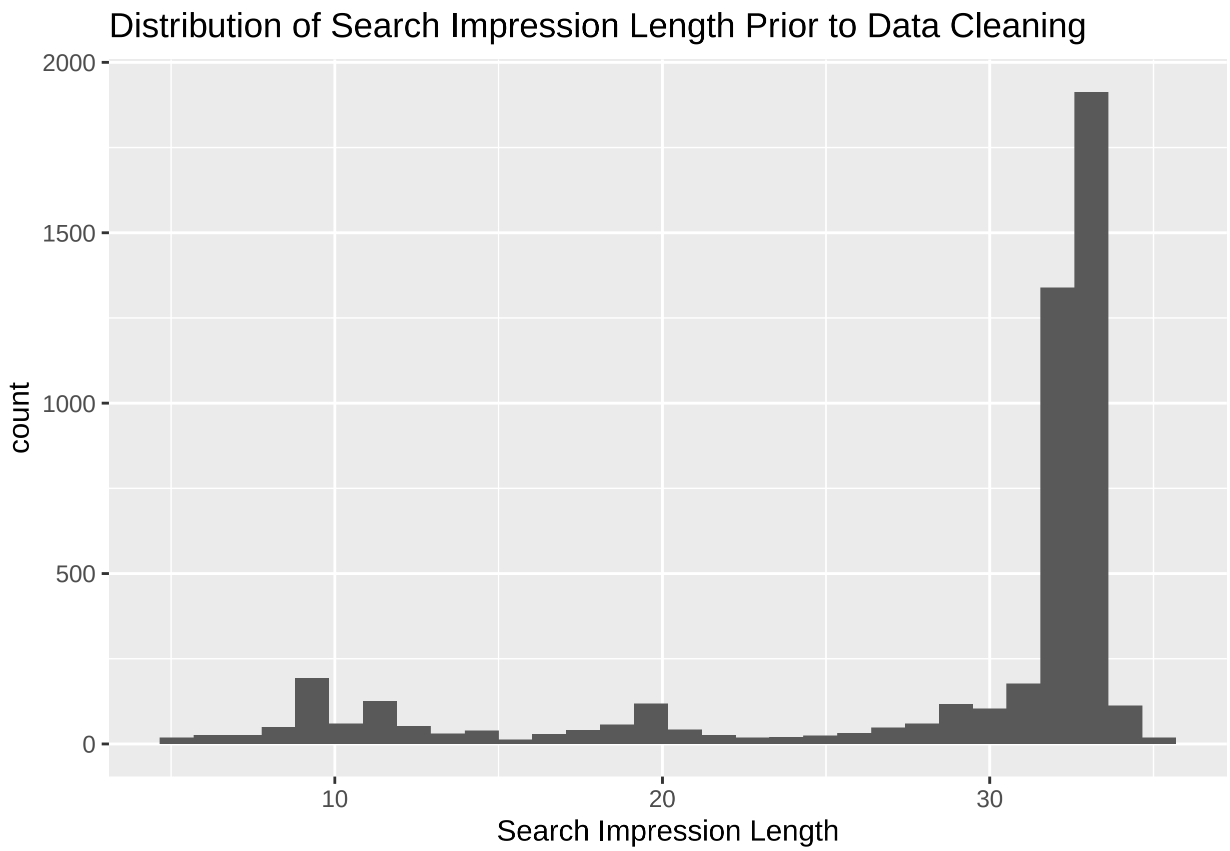
The code chunk in this section takes the data file `market1search.csv` as input. You can either construct `market1search.csv` for yourself from the raw kaggle data using the code in the previous section or use the file provided in the `SearchListingWelfareSoftware` folder.

If you did not run the first code chunk above, you will need to uncomment the first two lines of the code chunk below. You will also need to set the path in the second line to the `SearchListingWelfareSoftware` directory's location on your workspace.

```
#install.packages("tidyverse")
#my_folder <- "/Path/To/Directory/SearchListingWelfareSoftware"
library(tidyverse)
datmarket1 <- read_csv(file =
  paste(my_folder,"market1search.csv", sep = "/") )

sort(table(datmarket1$prop_id),decreasing = TRUE) #bookings by property
```

```
##
## 104517 124342  60846  40279 137997  59781  35223  38419 116942  68420  77089
##   4407   4381   4231   4208   4196   4193   4175   4137   4128   4087   4083
##  14082 134154  46274  94455  77795  21018  70177  59657  60468  78500  90845
##   4080   4069   4068   4056   4050   4040   4033   4031   4028   4007   3980
##  49656  37818  24545 122112 131892 114932 117294 125083  90809  26248  77914
##   3977   3967   3958   3931   3839   3692   3654   3505   3503   3354   3113
## 134323  38904  57952  29633 102130 100882  54850 137610  89699   9972  15443
##   2215   1869   1791   1445    763    390    336     70      9      8      8
##  11032  86042  18110  93970  46443 115624 136936  10860  29942  35720  51545
##      6      6      5      5      3      3      3      2      2      2      2
##  94534 105377 115022 127201 127791 128101 132451   4204   4548   5853   6973
##      2      2      2      2      2      2      2      1      1      1      1
##  10538  11800  12651  15207  16121  18300  22066  26711  28679  32521  34682
##      1      1      1      1      1      1      1      1      1      1      1
##  34860  38917  39849  42101  42690  43535  47102  48985  49210  54865  55230
##      1      1      1      1      1      1      1      1      1      1      1
##  56200  56255  57001  57978  59131  61179  61950  65235  65779  66535  71026
##      1      1      1      1      1      1      1      1      1      1      1
##  73820  74077  75716  79187  80548  80742  81089  86277  89203  91844  94178
##      1      1      1      1      1      1      1      1      1      1      1
## 105227 105312 105449 110263 111343 112575 113298 116470 118604 119349 121089
##      1      1      1      1      1      1      1      1      1      1      1
## 132626 133321 139310 139626 140331
##      1      1      1      1      1
```

```
full_listings <- names(table(datmarket1$prop_id)) #property names
top_listings_names <- names(table(datmarket1$prop_id))[
  table(datmarket1$prop_id)>=2500] #top property names
top_listings <- table(datmarket1$prop_id)[
  table(datmarket1$prop_id)>=2500] #bookings for top properties



srch_impression_dat = datmarket1 %>%
  group_by(srch_id) %>% dplyr::summarise(
    srch_impression_length = length(srch_id))

ggplot(srch_impression_dat) +
  geom_histogram(aes(x = srch_impression_length)) +
  ggtitle("Distribution of Search Impression Length Prior to Data Cleaning") +
  labs(x = "Search Impression Length")
```




```
#############################################
## Cleaning Step 1: removing those never booked
#############################################

# total clicks and bookings for each property
click_book_count = datmarket1 %>%
  group_by(prop_id) %>%
  dplyr::summarize(clicks = sum(click_bool),
    books = sum(booking_bool))

never_booked <- full_listings[
  click_book_count$books==0]

#filtering off unbooked locations
datmarket1filter <- datmarket1
for (troubleid in never_booked){
  datmarket1filter <- filter(datmarket1filter,
    prop_id != troubleid)
}

#############################################
## Cleaning Step 2: removing those Infrequently Booked
## i.e. booked less than 50 times
#############################################
```

```r
infrequent_booked <- full_listings[
  (click_book_count$books>0) & (
    (click_book_count$books)<50)
]

# Goes through search impressions and removes only the infrequently
# booked property from the impression if it wasn't booked in that
# impression, but deletes the entire search impression if it was
# booked in that search impression

for (troubleid in infrequent_booked){
  case1 = datmarket1filter$prop_id == troubleid
  case2 = datmarket1filter$booking_bool == 1
  case3 = !case2

  # Drop row if case 1 and case 3
  # So keep row if not case 1 or not case 3
  # this removes infrequent_booked id from search
  # impression where not booked
  if (sum(case1*(!case2)) > 0) {
    datmarket1filter =
      datmarket1filter[(!case1)|(case2), ]
  }

  # Have to re-get cases, because datmarket1filter
  # Changed now from above action
  # now deleting the search_ids that had booking from
  # infrequently_booked
  case1 = datmarket1filter$prop_id == troubleid
  case2 = datmarket1filter$booking_bool == 1

  srch_id_to_delete =
    datmarket1filter$srch_id[case1&case2]
  if (sum(case1*case2)>0) {
    for (myids in srch_id_to_delete) {
      datmarket1filter = filter(datmarket1filter,
      srch_id != myids)
    }
  }
}

srchs_removed_bc_infrequent_booked =
  length(unique(datmarket1$srch_id)) -
  length(unique(datmarket1filter$srch_id))
# removing the infrequently booked only reduced
# total search ids considered by 227, or 4.6%
srchs_removed_bc_infrequent_booked
```

```
## [1] 227
```

```r
srchs_removed_bc_infrequent_booked/length(unique(datmarket1$srch_id))
```

```
## [1] 0.04609137
```

3 Filtering Extreme Prices

In this section, I remove a small fraction of price outliers that appear misreported. This section requires the package `lubridate`, which I use for date and time data manipulation. If you do not already have lubridate installed, uncomment and execute the first line of code in the code chunk below before running the rest of the code chunk. This section is the last section with data cleaning. The plot of the distribution of search impression sizes in this section reproduces the plot included in my paper.

```r
#install.packages("lubridate")
library(lubridate) #for date time work

datmarket1filter = mutate(datmarket1filter,
  book_date = date(date_time),
  check_in_date = date(date_time) + days(srch_booking_window))

datmarket1filter <- mutate(datmarket1filter,
  book_day_of_week = wday(book_date))


min(datmarket1filter$book_date) #Nov 1, 2012
```

```
## [1] "2012-11-01"
```

```r
max(datmarket1filter$book_date) #June 30, 2013
```

```
## [1] "2013-06-30"
```

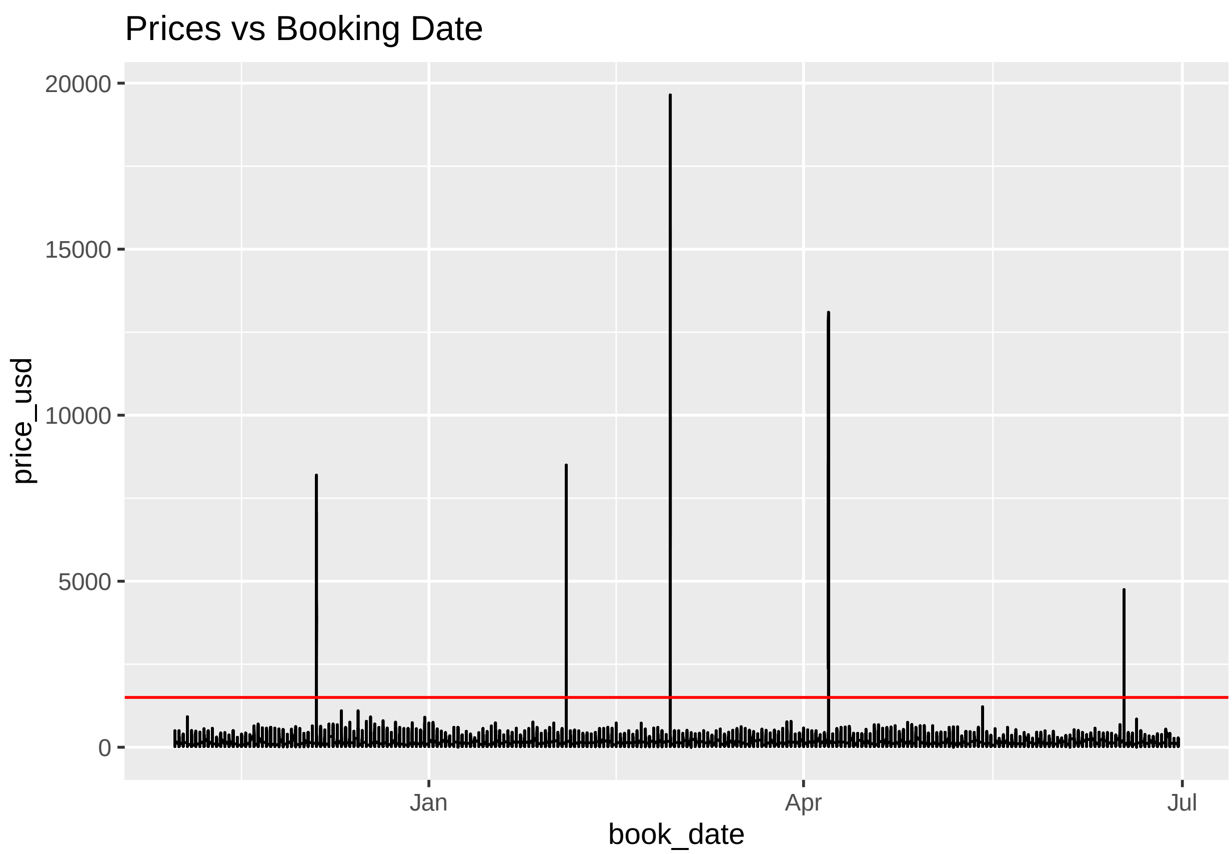
```r
min(datmarket1filter$check_in_date) #Nov 1, 2012
```

```
## [1] "2012-11-01"
```

```r
max(datmarket1filter$check_in_date) #June 2, 2014
```

```
## [1] "2014-06-02"
```

```r
# Plot of prices by booking date
# Some clear outliers.
# Dropping prices above 1500 cuts them

ggplot(data = datmarket1filter, aes(x = book_date,
    y = price_usd)) + geom_line() +
  geom_hline(yintercept = 1500, color="red") +
  ggtitle("Prices vs Booking Date")
```




```r
sum(datmarket1filter$price_usd>1500)
```

```
## [1] 59
```

```r
sum(datmarket1filter$price_usd>1500)/
  length(datmarket1filter$price_usd)
```

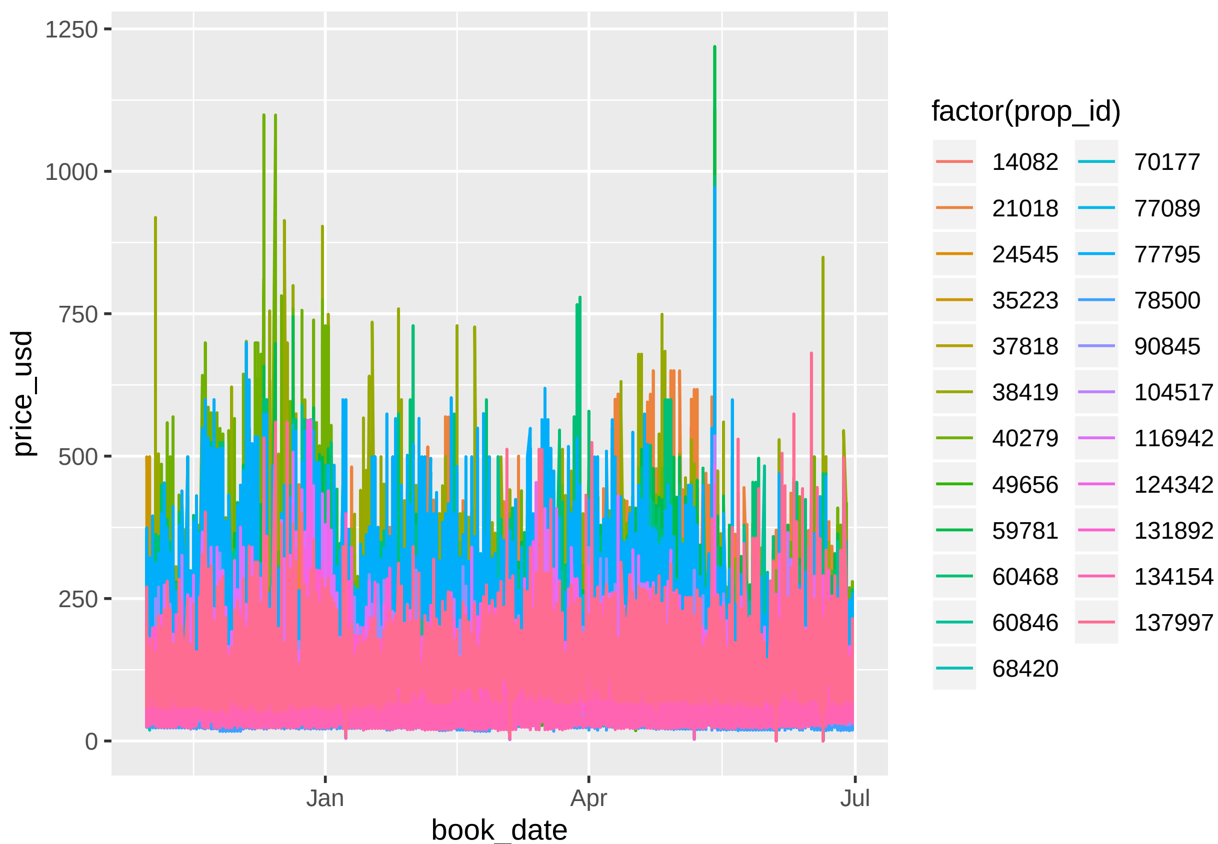
```
## [1] 0.0006516961
```

```r
# 59 prices above $1500, out of 90533 price observations
# note, some of the 59 prices are the same product on the same
# day showing up in different consumers search impressions.
# The above graph overlaps the repeats, showing
# it's just a few products on a few days

sum(
  datmarket1filter$booking_bool[
    datmarket1filter$price_usd>1500])
```

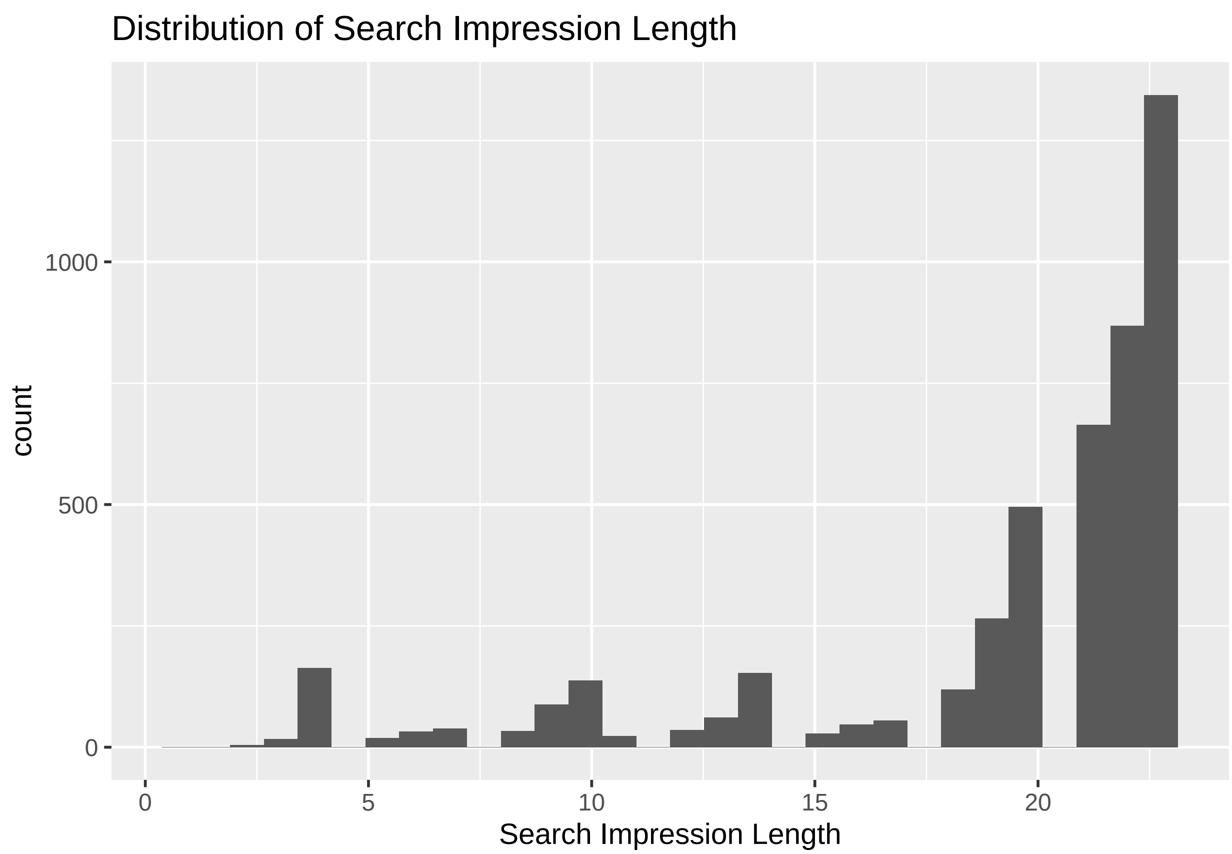
```
## [1] 3
```

```r
# 3 of these prices are observed books
# so these entire search impressions will need to be removed
# remove them first

high_price_purchase_srch <-
  datmarket1filter$srch_id[
    (datmarket1filter$booking_bool==1) &
```

```
      (datmarket1filter$price_usd>1500)]

for (troubleid in high_price_purchase_srch){
  darmarket1filter <- filter(datmarket1filter,
    srch_id != troubleid)
}

# Removing the rest of the high priced data
# So only those with prices under 1500 remain
datmarket1filter <- filter(datmarket1filter,
    price_usd<1500)

# checking remaining price plots
ggplot(data = datmarket1filter,
  aes(x = book_date, y = price_usd,
  color = factor(prop_id))) + geom_line()
```




```
temp = datmarket1filter %>% group_by(srch_id) %>%
  dplyr::summarise(
    srch_impression_length = length(srch_id),
    purchase_hotel = sum(booking_bool))

ggplot(temp) +
  geom_histogram(aes(x = srch_impression_length)) +
  ggtitle("Distribution of Search Impression Length") +
  labs(x = "Search Impression Length")
```




```
nrow(temp) # 4694 searches
```

```
## [1] 4694
```

```
# book, vs no book
summary(temp$purchase_hotel)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0000  0.0000  1.0000  0.6131  1.0000  1.0000
```

```
# 61% of remaining book a hotel!
```

4 Estimating Demand

In this section, I estimate demand from the data. This section's code depends on the package `mlogit`. If it is not already installed on your computer, uncomment and execute the first line of code below before executing the rest of the code in the code chunk.

```
#install.packages("mlogit")
library(mlogit)

dat4reg <- dplyr::select(datmarket1filter,
  srch_id,
  booking_bool,
  prop_id,
  prop_starrating,
  prop_review_score,
```

```
  prop_brand_bool,
  prop_location_score1,
  prop_location_score2,
  position,
  price_usd,
  promotion_flag,
  random_bool,
  book_day_of_week,
  book_date
)

# Adding outside gooods to all groups
# The following code first determines if
# The outside good is booked or not and adds
# in all information then binds row with data
# frame ( the . is place holder for all )
# then groups by srch_id
dat4reg = dat4reg %>% group_by(srch_id) %>%
  summarise(booking_bool = 1 - sum(booking_bool),
        random_bool = last(random_bool),
        book_day_of_week = last(book_day_of_week),
  book_date = last(book_date) )%>%
    mutate( prop_id = 0,
      prop_starrating = 0,
      prop_review_score = "2.5",
      prop_brand_bool = 0,
      prop_location_score1 = 0,
      prop_location_score2 = "0.111",
      position = 1,
      price_usd = 0,
      promotion_flag =0) %>%
    bind_rows(dat4reg,.) %>%
      arrange(srch_id)



mdat = mlogit.data(dat4reg,
  alt.var = "prop_id",
  choice= "booking_bool",
  shape = "long", chid.var = "srch_id",
  varying = 4:11)


f1 <-  mFormula(booking_bool ~
  prop_starrating +
  #prop_review_score +
  prop_brand_bool +
  prop_location_score1 +
  #prop_location_score2 +
  price_usd +
  promotion_flag|0)
```

```r
temp = sort(unique(mdat$prop_id))
regout <- mlogit(f1, mdat)
summary(regout)
```

```
##
## Call:
## mlogit(formula = booking_bool ~ prop_starrating + prop_brand_bool +
##     prop_location_score1 + price_usd + promotion_flag | 0, data = mdat,
##     method = "nr")
##
## Frequencies of alternatives:
##        0    14082    21018    24545    35223    37818    38419    40279
## 0.386877 0.019386 0.023860 0.015552 0.012569 0.042608 0.024499 0.021304
##    49656    59781    60468    60846    68420    70177    77089    77795
## 0.013634 0.016191 0.018534 0.023221 0.025138 0.034938 0.047720 0.012569
##    78500    90845   104517   116942   124342   131892   134154   137997
## 0.023434 0.018108 0.039625 0.056455 0.026417 0.019386 0.040264 0.037708
##
## nr method
## 5 iterations, 0h:0m:16s
## g'(-H)^-1g = 0.00501
## successive function values within tolerance limits
##
## Coefficients :
##                        Estimate  Std. Error  z-value  Pr(>|z|)
## prop_starrating       0.51342482  0.04820760  10.6503 < 2.2e-16 ***
## prop_brand_bool       0.41816822  0.05266617   7.9400 1.998e-15 ***
## prop_location_score1 -0.92183516  0.04267656 -21.6005 < 2.2e-16 ***
## price_usd            -0.01027078  0.00050739 -20.2424 < 2.2e-16 ***
## promotion_flag        0.26645073  0.04691611   5.6793 1.352e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Log-Likelihood: -11235
```

```r
# This is demand included in text

Num_Impression <- length(unique(dat4reg$srch_id))
Num_Books <- as.numeric(dat4reg %>% filter (prop_id != 0) %>%
  summarize( sum(booking_bool)))
Total_Sales <- dat4reg %>% group_by(prop_id) %>%
  dplyr::summarise(Market_Share =
  sum(booking_bool)/Num_Impression)                     #################################
# Checking fitted vs predicted
Total_Sales
```

```
## # A tibble: 24 x 2
##    prop_id Market_Share
##      <dbl>        <dbl>
##  1       0       0.387
##  2   14082       0.0194
##  3   21018       0.0239
##  4   24545       0.0156
##  5   35223       0.0126
```

```
## 6   37818       0.0426
## 7   38419       0.0245
## 8   40279       0.0213
## 9   49656       0.0136
## 10  59781       0.0162
## # ... with 14 more rows
```

```r
apply(fitted(regout, outcome = F), 2, mean)
```

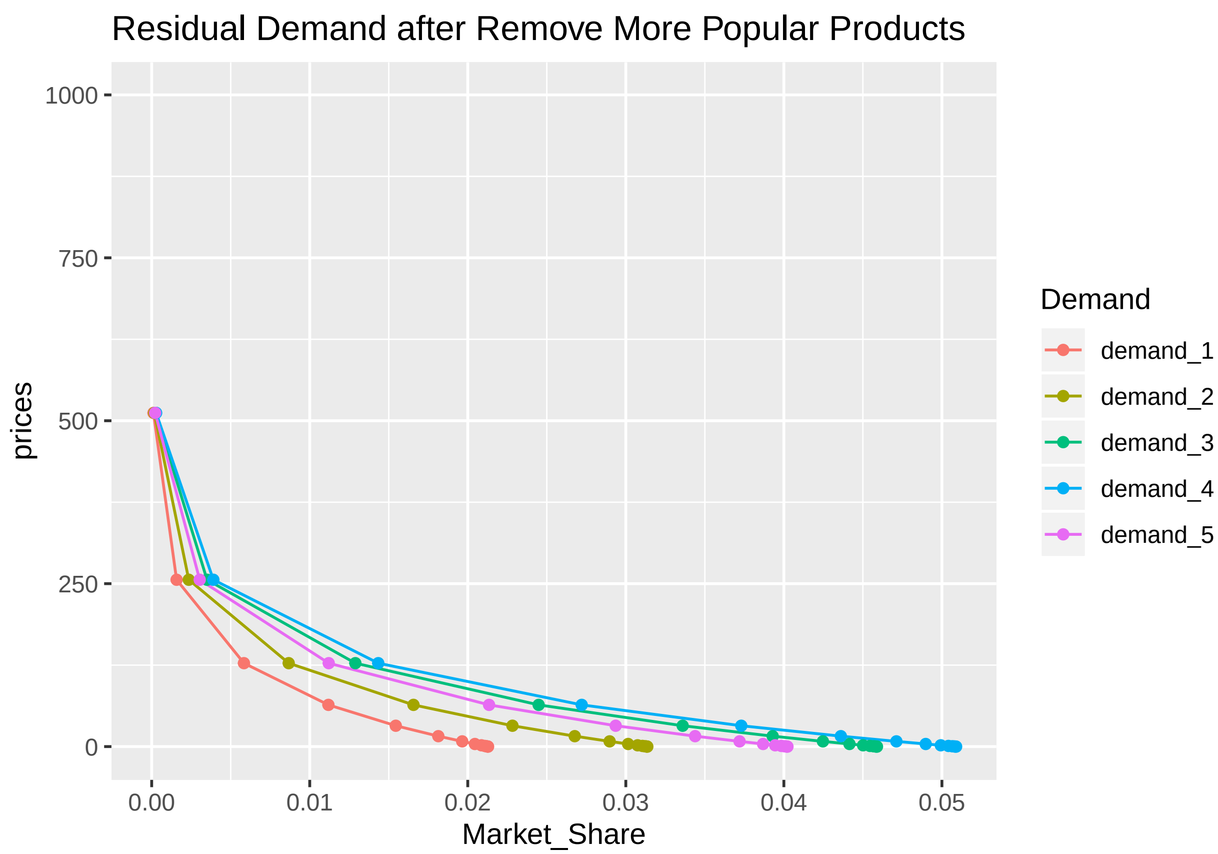
```
##          0      14082      21018      24545      35223      37818      38419
## 0.37672133 0.01642854 0.03823893 0.02011191 0.01754596 0.04322914 0.02197970
##      40279      49656      59781      60468      60846      68420      70177
## 0.01427079 0.02652776 0.02010580 0.01391068 0.02158838 0.05301220 0.02970190
##      77089      77795      78500      90845     104517     116942     124342
## 0.03069516 0.01400265 0.02630753 0.03161574 0.03439952 0.02128339 0.03056023
##     131892     134154     137997
## 0.02496667 0.04544327 0.02735284
```

5 Counterfactual Product Removal

In this section, I estimate the counterfactual welfare consequences of removing the 5 most booked hotels from all search impressions. I use the demand estimates from the previous section's code chunk and I use the Trapezoidal Riemann rule to approximate the area under the relevant (residual) demand curves. This code will take several minutes to execute.

```r
# trapezoidal rule for Riemann sum
WelfareCalc <- function(steps, height){
  area = 0
  for(k in 1:(length(steps)-1)) {
    dx = steps[k+1] - steps[k]
    area = area + (height[k+1] + height[k])/2*dx
  }
  area
}

# Finding 5 most booked products
Ordered_Prod <- Total_Sales %>% arrange(desc(Market_Share))
Top_5_Prod <- Ordered_Prod[-c(1,7:nrow(Ordered_Prod)),]
prod_names <-  colnames(predict(regout, mdat))
predicted_purchase = apply(fitted(regout, outcome = F), 2, mean)

delta_p_vec = c(0,.1,.25,.5,1,2,4,8,16,
  32,64, 128, 256, 512, 1024, 2048, 4096,
  8192, 16384, 32768)
steps_p_vec = c(0, delta_p_vec[-1] -
  delta_p_vec[-length(delta_p_vec)])


demand_mat <- matrix(0, nrow = length(steps_p_vec),
  ncol = 5)
mdattemp = mdat #initialize otside loop so price changes stay in effect

for (j in 1:5){
  current_prod = as.character(Top_5_Prod$prop_id[j])
```

```r
  for (k in 1:length(steps_p_vec)){
    # print( (j-1)*length(steps_p_vec) + k )
    mdattemp$price_usd =
      ifelse(mdattemp$prop_id==current_prod,
        mdattemp$price_usd + steps_p_vec[k],
        mdattemp$price_usd)
    force(mdattemp)
    demand_mat[k,j]  =
      apply(predict(regout, newdata =
        mdattemp), 2, mean)[current_prod]
  }
}

demand_list <- list(
  prices = delta_p_vec,
  demand_1 = demand_mat[,1],
  demand_2 = demand_mat[,2],
  demand_3 = demand_mat[,3],
  demand_4 = demand_mat[,4],
  demand_5 = demand_mat[,5]
)

demand_top5 <- as_tibble(demand_list)
demand_top5_long <- gather(demand_top5, demand_1,
  demand_2, demand_3, demand_4, demand_5,
  key = "Demand", value = "Market_Share")

ggplot(demand_top5_long, aes(x = Market_Share, y = prices,
    col = Demand)) + geom_line() +
  geom_point() + ylim(-1, 1000) +
  ggtitle("Residual Demand after Remove More Popular Products")
```




```
#Verify demand gets to 0 by end of price range
demand_top5[nrow(demand_top5),]  #does
```

```
## # A tibble: 1 x 6
##   prices  demand_1  demand_2  demand_3  demand_4  demand_5
##    <dbl>     <dbl>     <dbl>     <dbl>     <dbl>     <dbl>
## 1  32768 1.51e-148 2.24e-148 3.35e-148 3.73e-148 2.91e-148
```

```
# Calculating Welfare Loss
welf_lose5 = rep(0,5)
welf_lose5[1] = WelfareCalc(demand_list$demand_1,
  demand_list$prices)
welf_lose5[2] = WelfareCalc(demand_list$demand_2,
  demand_list$prices)
welf_lose5[3] = WelfareCalc(demand_list$demand_3,
  demand_list$prices)
welf_lose5[4] = WelfareCalc(demand_list$demand_4,
  demand_list$prices)
welf_lose5[5] = WelfareCalc(demand_list$demand_5,
  demand_list$prices)

# Welf_lose5 is reported in paper as welfare loss
# from sequentially removing goods 1 through 5 from
# consideration sets
welf_lose5
```

```
## [1] -2.274739 -3.370144 -4.984527 -5.540072 -4.344517
```

```r
sum(welf_lose5)
```

```
## [1] -20.514
```

```r
cumsum(welf_lose5)
```

```
## [1]  -2.274739  -5.644883 -10.629410 -16.169483 -20.514000
```

6 Welfare Change from Random to NonRandom Search

In this section, I estimate the welfare change in going from the OTA's random listing rule to the OTA's proprietary listing rule. This section depends on the demand estimates calculated in the code chunk in Section 4 and on some welfare functions defined in the code chunk from Section 5. This code will take several minutes to execute.

```r
# Partition data
dat4regrando = mdat[mdat$random_bool==1,]
dat4regnorando = mdat[mdat$random_bool!=1,]
predicted_share_rando <- apply(predict(regout, newdata =
        dat4regrando), 2, mean)
predicted_share_norando <- apply(predict(regout, newdata =
        dat4regnorando), 2, mean)


Total_Welfare_Calc <- function(data){
  mdattemp = data
  demand_mat_loc = matrix(0, nrow = length(steps_p_vec),
    ncol = (length(prod_names) -1) )
  for (j in 2:length(prod_names)){
  current_prod = prod_names[j] #1st is outside good
      # starts at 2 because welfare formula looks at area under all
      # gooods but the outside good
    for (k in 1:length(steps_p_vec)){
      # remove comment on print to track steps of analysis
      # print( (j-2)*length(steps_p_vec) + k )
      mdattemp$price_usd =
        ifelse(mdattemp$prop_id==current_prod,
          mdattemp$price_usd + steps_p_vec[k],
          mdattemp$price_usd)
      force(mdattemp)
      demand_mat_loc[k,j-1]  =
        apply(predict(regout, newdata =
          mdattemp), 2, mean)[current_prod]
    }
  }
  demand_mat_loc
}

Total_Welfare_Rando = Total_Welfare_Calc(dat4regrando)
Total_Welfare_NoRando = Total_Welfare_Calc(dat4regnorando)

# Initializing vectors to store areas under residual
# demand curve for goods 1 through 5, as in formulas from paper
Welfare_Vec_NoRando <- rep(0, length(prod_names)-1)
```

```r
Welfare_Vec_Rando = Welfare_Vec_NoRando

# calculating areas under residual demand curves with
# Trapezoidal Riemann Sum
for (k in 1:length(Welfare_Vec_NoRando)){
  Welfare_Vec_NoRando[k] =
    WelfareCalc(Total_Welfare_NoRando[,k],
      demand_list$prices)
  Welfare_Vec_Rando[k] =
    WelfareCalc( Total_Welfare_Rando[,k],
      demand_list$prices)

}

# Total Consumer Welfare under different experiments
# Reported in paper
sum(Welfare_Vec_NoRando)
```

```
## [1] -113.6468
```

```r
sum(Welfare_Vec_Rando)
```

```
## [1] -104.8138
```

```r
# change in welfare across experiments
welfare_benefit_norando = round(abs(sum(Welfare_Vec_NoRando)), digits=2) -
  round(abs(sum(Welfare_Vec_Rando)), digits=2)

# result reported in paper
welfare_benefit_norando
```

```
## [1] 8.84
```

7 Predicted Change in Ranking for Price Increase of 3 Standard Deviations

In this section, I look at the relationship between rankings, price and the other demand covariates from Section 4. I run a simple regression of rankings on price for the listings ranked by the OTA's proprietary listing rule. For robustness, I also run the regression for the listings ranked by OTA's random ranking system. The results show price is a much weaker predictor of ranking in the random case than in the proprietarily ranked case, as expected. Finally, I use the regression output in the former case to predict the ranking effect of a price increase of 3 standard deviations for each of the 5 most popular goods in the data.

```r
library(lubridate) #for date time work

#Top K products
K = 5
Ordered_Prod <- Total_Sales %>% arrange(desc(Market_Share))
Top_K_Prod <- Ordered_Prod[-c(1,(K+2):nrow(Ordered_Prod)),]
  #product 1 is outside good
Top_K_prod_names <-  Top_K_Prod$prop_id

Top_K_Prod_List <- list()
```

```r
for (counter in 1:length(Top_K_prod_names)){
  force (counter)
  Top_K_Prod_List[[counter]] <- datmarket1filter %>%
    filter(prop_id==Top_K_prod_names[counter])
}

market1rando = datmarket1filter %>% filter(random_bool == 1)
market1norando = datmarket1filter %>% filter(random_bool != 1)
outrando = lm(position ~ price_usd, market1rando)
outnorando = lm(position ~ price_usd, market1norando)
summary(outrando)
```

```
##
## Call:
## lm(formula = position ~ price_usd, data = market1rando)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -17.7177  -9.9044   0.0077   9.5973  20.1218
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 1.784e+01  1.151e-01  154.94   <2e-16 ***
## price_usd   1.563e-03  7.408e-04    2.11   0.0348 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 10.76 on 32696 degrees of freedom
## Multiple R-squared:  0.0001362,  Adjusted R-squared:  0.0001056
## F-statistic: 4.454 on 1 and 32696 DF,  p-value: 0.03484
```

```r
summary(outnorando)
```

```
##
## Call:
## lm(formula = position ~ price_usd, data = market1norando)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -31.258  -7.758  -1.026   7.259  23.572
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 13.045770   0.070863  184.10   <2e-16 ***
## price_usd    0.017371   0.000466   37.28   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 9.325 on 57774 degrees of freedom
## Multiple R-squared:  0.02349,    Adjusted R-squared:  0.02347
## F-statistic:  1390 on 1 and 57774 DF,  p-value: < 2.2e-16
```

```r
# Price is much weaker predictor of ranking
# for rando, than no rando, as expected
# Repeat with covariates
```

```r
outrando_cov = lm(position ~ price_usd + prop_starrating + prop_brand_bool+
  promotion_flag + prop_location_score1, market1rando)
outnorando_cov = lm(position ~ price_usd + prop_starrating +
  prop_brand_bool+
  promotion_flag + prop_location_score1, market1norando)
summary(outrando_cov)
```

```
##
## Call:
## lm(formula = position ~ price_usd + prop_starrating + prop_brand_bool +
##     promotion_flag + prop_location_score1, data = market1rando)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -17.7814  -9.6134  -0.0542   9.4960  20.8544
##
## Coefficients:
##                        Estimate Std. Error t value Pr(>|t|)
## (Intercept)          16.5967360  0.9439399  17.582  < 2e-16 ***
## price_usd            -0.0021565  0.0009943  -2.169 0.030100 *
## prop_starrating       0.5889761  0.1346269   4.375 1.22e-05 ***
## prop_brand_bool       0.6122396  0.1839096   3.329 0.000872 ***
## promotion_flag       -0.1006733  0.1388044  -0.725 0.468280
## prop_location_score1 -0.2627163  0.2655910  -0.989 0.322584
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 10.75 on 32692 degrees of freedom
## Multiple R-squared:  0.001405,   Adjusted R-squared:  0.001252
## F-statistic:   9.2 on 5 and 32692 DF,  p-value: 9.204e-09
```

```r
summary(outnorando_cov)
```

```
##
## Call:
## lm(formula = position ~ price_usd + prop_starrating + prop_brand_bool +
##     promotion_flag + prop_location_score1, data = market1norando)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -24.4678  -7.2703  -0.6283   6.9555  24.8035
##
## Coefficients:
##                        Estimate Std. Error t value Pr(>|t|)
## (Intercept)          16.1503777  0.5946602  27.159  < 2e-16 ***
## price_usd             0.0067546  0.0005849  11.548  < 2e-16 ***
## prop_starrating       0.6665611  0.0830996   8.021 1.07e-15 ***
## prop_brand_bool      -0.8034319  0.1169385  -6.871 6.46e-12 ***
## promotion_flag       -4.6436665  0.0871619 -53.276  < 2e-16 ***
## prop_location_score1 -0.1591026  0.1672941  -0.951    0.342
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 9.076 on 57770 degrees of freedom
```

```
## Multiple R-squared:  0.07494,    Adjusted R-squared:  0.07486
## F-statistic:   936 on 5 and 57770 DF,  p-value: < 2.2e-16
```

```
# Measure ranking effect of 3 sd price increase given goods average
# characteristics
predict_simple <- list(K)
predict_cov <- list(K)
meangoodvec <- rep(0,K)
sdgoodvec <- rep(0,K)
for (k in 1:K) {
  sdgoodvec[k] <- sd(Top_K_Prod_List[[k]]$price_usd)
  meangoodvec[k] <- mean(Top_K_Prod_List[[k]]$price_usd)
  newdat = tibble(price_usd= c(meangoodvec[k]-2*sdgoodvec[k],
    meangoodvec[k]- sdgoodvec[k], meangoodvec[k],
    meangoodvec[k]+sdgoodvec[k], meangoodvec[k]+2*sdgoodvec[k],
    meangoodvec[k]+3*sdgoodvec[k]))
  predict_simple[[k]] = predict(outnorando, newdat)
  newdatgoodtemp =  Top_K_Prod_List[[k]] %>%
    select(price_usd, prop_starrating,
    prop_brand_bool, promotion_flag, prop_location_score1)  %>%
    mutate(count = rep(6,nrow(.))) %>% dplyr::summarise_all(mean) %>%
    uncount(count) %>% mutate(price_usd = c(price_usd[1] - 2*sdgoodvec[k],
    price_usd[1] - sdgoodvec[k], price_usd[1], price_usd[1] +
    sdgoodvec[k], price_usd[1]+2*sdgoodvec[k], price_usd[1]+3*sdgoodvec[k]))
  predict_cov[[k]] <- predict(outnorando_cov, newdatgoodtemp )
}
predict_simple
```

```
## [[1]]
##        1        2        3        4        5        6
## 13.10880 14.10006 15.09132 16.08258 17.07384 18.06510
##
## [[2]]
##        1        2        3        4        5        6
## 12.86697 13.57972 14.29246 15.00521 15.71796 16.43070
##
## [[3]]
##        1        2        3        4        5        6
## 12.97126 13.78631 14.60136 15.41640 16.23145 17.04650
##
## [[4]]
##        1        2        3        4        5        6
## 12.83074 13.40639 13.98204 14.55769 15.13334 15.70899
##
## [[5]]
##        1        2        3        4        5        6
## 13.10190 14.00586 14.90982 15.81378 16.71774 17.62170
```

```
predict_cov
```

```
## [[1]]
##        1        2        3        4        5        6
## 13.70727 14.09272 14.47817 14.86361 15.24906 15.63451
##
## [[2]]
##        1        2        3        4        5        6
```

```
## 12.04122 12.31837 12.59552 12.87266 13.14981 13.42696
##
## [[3]]
##        1        2        3        4        5        6
## 12.75591 13.07283 13.38976 13.70669 14.02361 14.34054
##
## [[4]]
##        1        2        3        4        5        6
## 12.03181 12.25565 12.47949 12.70333 12.92716 13.15100
##
## [[5]]
##        1        2        3        4        5        6
## 13.12249 13.47399 13.82549 14.17699 14.52849 14.87999
```

```r
# Predicted ranking change of 3 standard deviation
# price increase for each of the top 5 goods
# in proprietary listing data. Regression without covariates
rankmatrixsimple = matrix(as.numeric(unlist(predict_simple)), 5,6, byrow=T)
rankmatrixsimple[,6] - rankmatrixsimple[,3]
```

```
## [1] 2.973779 2.138239 2.445141 1.726950 2.711875
```

```r
# Predicted ranking change of 3 standard deviation
# price increase for each of the top 5 goods
# in proprietary listing data. Regression with covariates.
# Numbers reported in paper.
rankmatrixcov = matrix(as.numeric(unlist(predict_cov)), 5,6, byrow=T)
rankmatrixcov[,6] - rankmatrixcov[,3]
```

```
## [1] 1.1563401 0.8314443 0.9507816 0.6715163 1.0544999
```

8 Predicted demand when prices are 3 standard deviations above their inital level

In this section, I use Section 4's demand estimates to predict demand changes when prices are increased 3 standard deviations above their market level. I do this for each of the 5 most popular goods.

```r
#Comparing against predicted demand at 3 price

# Finding 5 most booked products
Ordered_Prod <- Total_Sales %>% arrange(desc(Market_Share))
Top_5_Prod <- Ordered_Prod[-c(1,7:nrow(Ordered_Prod)),]
prod_names <-  colnames(predict(regout, mdat))
predicted_purchase = apply(fitted(regout, outcome = F), 2, mean)

demand_mat <- matrix(0, nrow = 4,
  ncol = 5)


for (j in 1:K){
  delta_p_vec = c(0,sdgoodvec[j], 2*sdgoodvec[j], 3*sdgoodvec[j])
  steps_p_vec = c(0, delta_p_vec[-1] -
    delta_p_vec[-length(delta_p_vec)])
```

```r
  current_prod = as.character(Top_5_Prod$prop_id[j])
  mdattemp = mdat #initialize inside loop so price changes removed
  for (k in 1:length(steps_p_vec)){
    mdattemp$price_usd =
      ifelse(mdattemp$prop_id==current_prod,
        mdattemp$price_usd + steps_p_vec[k],
        mdattemp$price_usd)
    force(mdattemp)
    demand_mat[k,j]  =
      apply(predict(regout, newdata =
        mdattemp), 2, mean)[current_prod]
  }
}

#change in demand from 3 standard deviatino price increase
demand_mat[4,] - demand_mat[1,]
```

```
## [1] -0.01752678 -0.02178065 -0.03259002 -0.02845890 -0.02723714
```

```r
#percent change in demand, reported in data
(demand_mat[4,] - demand_mat[1,])/demand_mat[1,]
```

```
## [1] -0.8234958 -0.7095791 -0.7538903 -0.6262515 -0.7917883
```

Bonus Robustness Check - Repeat Product Removal Analysis Using Demand Estimates on Randomly Ordered Data

In this section, I repeat the analysis from Section 5, Counterfactual Product Removal, using only the randomly ranked data to predict demand. This is a robustness check on the results of Section 5, where the entire cleaned data set was used to predict welfare changes. Results are very similar to the results of Section 5.

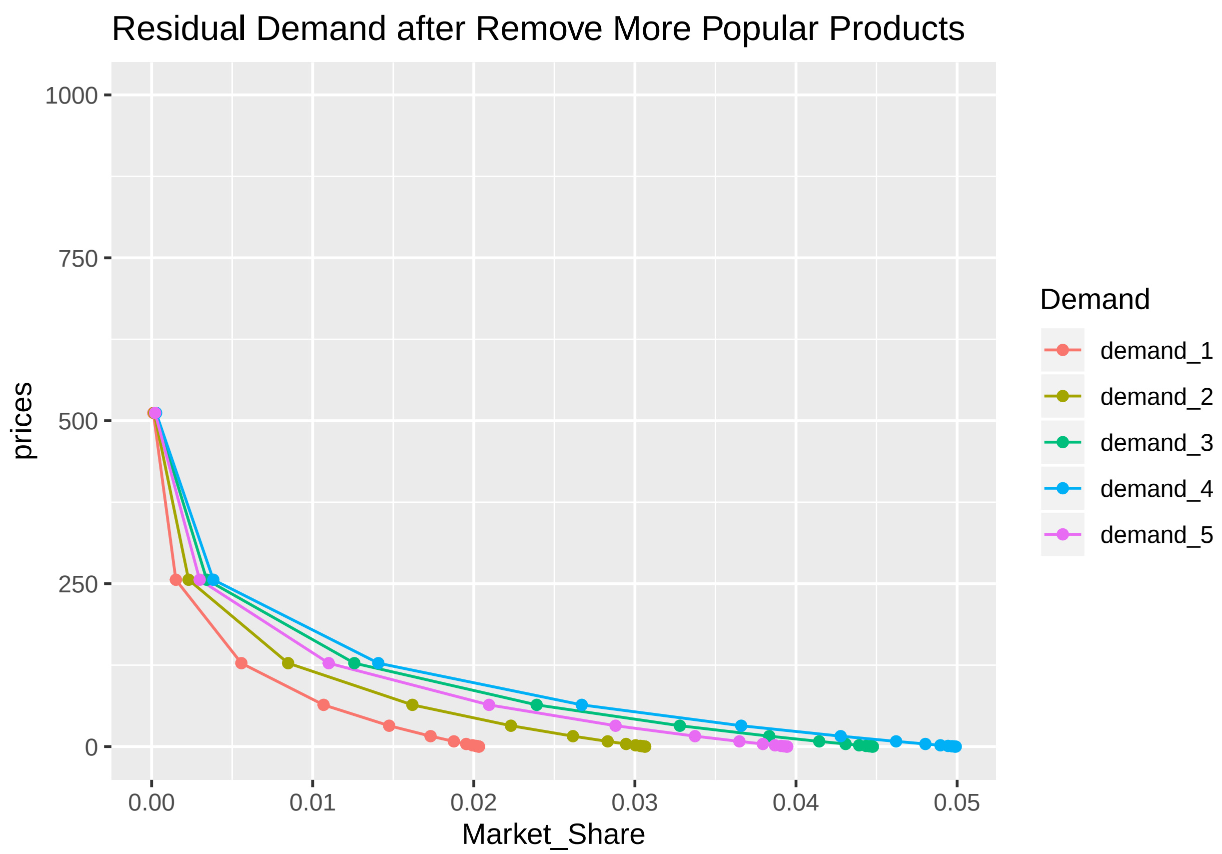
```r
mdatrando = mlogit.data(dat4regrando,
  alt.var = "prop_id",
  choice= "booking_bool",
  shape = "long", chid.var = "srch_id",
  varying = 4:11)


f1 <-  mFormula(booking_bool ~
  prop_starrating +
  #prop_review_score +
  prop_brand_bool +
  prop_location_score1 +
  #prop_location_score2 +
  price_usd +
  promotion_flag|0)


temp = sort(unique(mdatrando$prop_id))
regoutrando <- mlogit(f1, mdatrando)
summary(regoutrando)
```

```
##
## Call:
```

```
## mlogit(formula = booking_bool ~ prop_starrating + prop_brand_bool +
##     prop_location_score1 + price_usd + promotion_flag | 0, data = mdatrando,
##     method = "nr")
##
## Frequencies of alternatives:
##          0      14082      21018      24545      35223      37818      38419
## 0.90028169 0.00281690 0.00394366 0.00169014 0.00225352 0.01126761 0.00338028
##      40279      49656      59781      60468      60846      68420      70177
## 0.00225352 0.00112676 0.00112676 0.00338028 0.00225352 0.00169014 0.00507042
##      77089      77795      78500      90845     104517     116942     124342
## 0.01126761 0.00056338 0.00338028 0.00507042 0.00507042 0.01070423 0.00450704
##     131892     134154     137997
## 0.00450704 0.00394366 0.00845070
##
## nr method
## 9 iterations, 0h:0m:3s
## g'(-H)^-1g = 4.07E-07
## gradient close to zero
##
## Coefficients :
##                        Estimate Std. Error z-value  Pr(>|z|)
## prop_starrating       0.2616566  0.1784249  1.4665    0.1425
## prop_brand_bool       0.9064251  0.2208917  4.1035 4.070e-05 ***
## prop_location_score1 -1.4829458  0.1630654 -9.0942 < 2.2e-16 ***
## price_usd            -0.0087506  0.0017669 -4.9525 7.326e-07 ***
## promotion_flag        0.0984321  0.1828226  0.5384    0.5903
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Log-Likelihood: -1082.4
```

```r
# trapezoidal rule for Riemann sum
WelfareCalc <- function(steps, height){
  area = 0
  for(k in 1:(length(steps)-1)) {
    dx = steps[k+1] - steps[k]
    area = area + (height[k+1] + height[k])/2*dx
  }
  area
}

# Finding 5 most booked products
Ordered_Prod <- Total_Sales %>% arrange(desc(Market_Share))
Top_5_Prod <- Ordered_Prod[-c(1,7:nrow(Ordered_Prod)),]
prod_names <-  colnames(predict(regoutrando, mdatrando))
predicted_purchase = apply(fitted(regoutrando, outcome = F), 2, mean)

delta_p_vec = c(0,.1,.25,.5,1,2,4,8,16,
  32,64, 128, 256, 512, 1024, 2048, 4096,
  8192, 16384, 32768)
steps_p_vec = c(0, delta_p_vec[-1] -
  delta_p_vec[-length(delta_p_vec)])
```

```r
demand_mat <- matrix(0, nrow = length(steps_p_vec),
  ncol = 5)
mdattemp = mdatrando #initialize

for (j in 1:5){
  current_prod = as.character(Top_5_Prod$prop_id[j])
  for (k in 1:length(steps_p_vec)){
    # print( (j-1)*length(steps_p_vec) + k )
    mdattemp$price_usd =
      ifelse(mdattemp$prop_id==current_prod,
        mdattemp$price_usd + steps_p_vec[k],
        mdattemp$price_usd)
    force(mdattemp)
    demand_mat[k,j]  =
      apply(predict(regout, newdata =
        mdattemp), 2, mean)[current_prod]
  }
}

demand_list <- list(
  prices = delta_p_vec,
  demand_1 = demand_mat[,1],
  demand_2 = demand_mat[,2],
  demand_3 = demand_mat[,3],
  demand_4 = demand_mat[,4],
  demand_5 = demand_mat[,5]
)

demand_top5 <- as_tibble(demand_list)
demand_top5_long <- gather(demand_top5, demand_1,
  demand_2, demand_3, demand_4, demand_5,
  key = "Demand", value = "Market_Share")

ggplot(demand_top5_long, aes(x = Market_Share, y = prices,
    col = Demand)) + geom_line() +
  geom_point() + ylim(-1, 1000) +
  ggtitle("Residual Demand after Remove More Popular Products")
```




```r
#Verify demand gets to 0 by end of price range
demand_top5[nrow(demand_top5),]  #does
```

```
## # A tibble: 1 x 6
##   prices  demand_1  demand_2  demand_3  demand_4  demand_5
##    <dbl>     <dbl>     <dbl>     <dbl>     <dbl>     <dbl>
## 1  32768 1.44e-148 2.19e-148 3.28e-148 3.67e-148 2.85e-148
```

```r
# Calculating Welfare Loss
welf_lose5 = rep(0,5)
welf_lose5[1] = WelfareCalc(demand_list$demand_1,
  demand_list$prices)
welf_lose5[2] = WelfareCalc(demand_list$demand_2,
  demand_list$prices)
welf_lose5[3] = WelfareCalc(demand_list$demand_3,
  demand_list$prices)
welf_lose5[4] = WelfareCalc(demand_list$demand_4,
  demand_list$prices)
welf_lose5[5] = WelfareCalc(demand_list$demand_5,
  demand_list$prices)

# Welf_lose5 is reported in paper as welfare loss
# from sequentially removing goods 1 through 5 from
# consideration sets
welf_lose5
```

```
## [1] -2.171634 -3.293838 -4.866668 -5.437225 -4.262800
```

```r
sum(welf_lose5)
```

```
## [1] -20.03217
```

```r
cumsum(welf_lose5)
```

```
## [1]  -2.171634  -5.465472 -10.332140 -15.769365 -20.032165
```

```r
# similar to previous
```