Stat 432 Homework 2

Giang Le (gianghl2)

Assigned: Aug 30, 2021; Due: 11:59 PM CT, Sep 7, 2021

Contents

Question 1 (linear regression review)

Let's used the real estate data as an example. The data can be obtained from the course website.

a. Construct a new categorical variable called `season` into the real estate dataset. You should utilize the original variable `date` to perform this task and read the definition provided in our lecture notes. The `season` variable should be defined as: spring (Mar - May), summer (Jun - Aug), fall (Sep - Nov), and winter (Dec - Feb). Show a summary table to demonstrate that your variable conversion is correct.

In the code below, I calculated the difference between `date` and the whole number part of the variable's value. I multiplied the result by 12 to get the month number and converted them to names of months as string. After that I created a variable called `season` based on string matches with the months (spring (Mar - May), summer (Jun - Aug), fall (Sep - Nov), and winter (Dec - Feb)).

```
realestate = read.csv("realestate.csv", row.names = 1)
head(realestate, 10)
```

```
##        date  age   distance stores latitude longitude price
## 1  2012.917 32.0   84.87882     10 24.98298  121.5402  37.9
## 2  2012.917 19.5  306.59470      9 24.98034  121.5395  42.2
## 3  2013.583 13.3  561.98450      5 24.98746  121.5439  47.3
## 4  2013.500 13.3  561.98450      5 24.98746  121.5439  54.8
## 5  2012.833  5.0  390.56840      5 24.97937  121.5425  43.1
## 6  2012.667  7.1 2175.03000      3 24.96305  121.5125  32.1
## 7  2012.667 34.5  623.47310      7 24.97933  121.5364  40.3
## 8  2013.417 20.3  287.60250      6 24.98042  121.5423  46.7
## 9  2013.500 31.7 5512.03800      1 24.95095  121.4846  18.8
## 10 2013.417 17.9 1783.18000      3 24.96731  121.5149  22.1
```

```
# Define a new factor variable for month
realestate$month = as.factor(as.numeric((format(round((realestate$date
                  - trunc(realestate$date)),3)))) * 12)
table(realestate$month)
```

```
##
##      0  0.996  2.004      3  3.996  5.004      6  6.996  8.004      9  9.996
##     28     46     25     32     29     58     47     23     30     27     31
## 11.004
##     38
```

```r
levels(realestate$month) = c("Dec", "Jan", "Feb", "Mar", "Apr", "May",
                             "Jun", "Jul", "Aug", "Sep", "Oct", "Nov", "Dec")
head(realestate$month)
```

```
## [1] Nov Nov Jul Jun Oct Aug
## Levels: Dec Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov
```

```r
# Define a new factor variable for season
realestate$season[realestate$month == "Dec" | realestate$month == "Jan" |
                    realestate$month == "Feb"]  <- "winter"
realestate$season[realestate$month == "Mar" | realestate$month == "Apr" |
                    realestate$month == "May"]  <- "spring"
realestate$season[realestate$month == "Jun" | realestate$month == "Jul" |
                    realestate$month == "Aug"]  <- "summer"
realestate$season[realestate$month == "Sep" | realestate$month == "Oct" |
                    realestate$month == "Nov"]  <- "fall"
class(realestate$season)
```

```
## [1] "character"
```

```r
realestate$season <- as.factor(realestate$season)
```

In this summary, we can see that there are 4 levels of `season` and the number of examples looks correct.

```r
summary(realestate)
```

```
##       date           age           distance           stores      
##  Min.   :2013   Min.   : 0.000   Min.   :  23.38   Min.   : 0.000  
##  1st Qu.:2013   1st Qu.: 9.025   1st Qu.: 289.32   1st Qu.: 1.000  
##  Median :2013   Median :16.100   Median : 492.23   Median : 4.000  
##  Mean   :2013   Mean   :17.713   Mean   :1083.89   Mean   : 4.094  
##  3rd Qu.:2013   3rd Qu.:28.150   3rd Qu.:1454.28   3rd Qu.: 6.000  
##  Max.   :2014   Max.   :43.800   Max.   :6488.02   Max.   :10.000  
##                                                                     
##     latitude       longitude         price            month        season   
##  Min.   :24.93   Min.   :121.5   Min.   :  7.60   May    : 58   fall  : 96  
##  1st Qu.:24.96   1st Qu.:121.5   1st Qu.: 27.70   Jun    : 47   spring:119  
##  Median :24.97   Median :121.5   Median : 38.45   Jan    : 46   summer:100  
##  Mean   :24.97   Mean   :121.5   Mean   : 37.98   Nov    : 38   winter: 99  
##  3rd Qu.:24.98   3rd Qu.:121.5   3rd Qu.: 46.60   Mar    : 32               
##  Max.   :25.01   Max.   :121.6   Max.   :117.50   Oct    : 31               
##                                                   (Other):162               
```

b. Split your data into two parts: a testing data that contains 100 observations, and the rest as training data. For this question, you need to set a random seed while generating this split so that the result can be replicated. **Use your UIN as the random seed**. Report the mean `price` of your testing data and training data, respectively.

I set the seed as my UIN and split the data into train and test (100 observations) by randomly sampling 100 indices to select the observations.

```r
set.seed(662095561)
sample_index <- sample.int(n = nrow(realestate), size = 100, replace = F)
sample_index
```

```
##   [1] 284 173  19 381 227 158 281  47 111 188 221 228  54 267  57 183 186 177
##  [19] 391 261  37 124  40 398  30   3   4 355 219  43 335  59 136 414 151   2
##  [37] 258 187 193 110 301 300  46 239 116 240  10 142 408 358 276  88 359  96
```

```
## [55]   7 236 331  22 232 327  32  55  75 337 103 220 169 338 270 202 154 294
## [73] 137 196 214 395 345 354 348  83  11 198 156 201 208 278 285 190 104  21
## [91] 246 314 308   9 271 363 340  63 178 266
```

```r
train <- realestate[-sample_index, ]
test  <- realestate[sample_index, ]
```

Here are means of `price` in the train and test data.

```r
mean(train$price)
```

```
## [1] 37.96019
```

```r
mean(test$price)
```

```
## [1] 38.043
```

c. Use the training dataset to perform a linear regression. The goal is to model `price` with `season`, `age`, `distance` and `stores`. Then use this model to predict the testing data using the `predict()` function. Calculate the training data mean squared error (**training error**):

$$\text{Training Error} = \frac{1}{n_{\text{train}}} \sum_{i \in \text{Train}} (y_i - \hat{y}_i)^2$$

and prediction mean squared error (**testing error**) using the testing data, defined as:

$$\text{Testing Error} = \frac{1}{n_{\text{test}}} \sum_{i \in \text{Test}} (y_i - \hat{y}_i)^2$$

I performed a linear regression by calling lm(). After that I called predict.lm() to obtain predictions on the test data.

```r
re_lm = lm(price ~ season + age + distance + stores, data=train)
predict.lm(re_lm, test)
```

```
##       284       173        19       381       227       158       281        47
## 26.040554 54.489266 47.213217 44.409773 16.461320 39.684750 49.477395 46.731338
##       111       188       221       228        54       267        57       183
## 46.956754 23.954494 44.074765 41.896224 43.455949 32.647218 42.710589 33.497800
##       186       177       391       261        37       124        40       398
## 29.104996 12.842312 45.891083 35.808821 29.614079 42.122099 43.778961 38.595685
##        30         3         4       355       219        43       335        59
## 45.343907 44.817361 44.817361 33.793207 43.442107 37.250586 34.493471 13.496344
##       136       414       151         2       258       187       193       110
## 30.858031 54.516127 41.880801 46.463529 37.398068 29.564853 40.279676 36.049558
##       301       300        46       239       116       240        10       142
## 46.853760 46.762139 41.199519 35.644191 36.177272 35.391271 32.620357 37.271527
##       408       358       276        88       359        96         7       236
## 28.582615 53.238923 53.567720 18.177901 49.682723 45.336961 41.444679 45.884557
##       331        22       232       327        32        55        75       337
## 28.184657 48.087597 14.934780 52.689628 38.581425 43.805821 50.806424 32.560783
##       103       220       169       338       270       202       154       294
## 49.682723 41.291251 39.490619 36.384432 33.785338 43.197586 47.307378 48.631618
##       137       196       214       395       345       354       348        83
## 42.871771 43.181401 52.856474 14.910191 18.753570 36.022697  6.220840 47.992071
##        11       198       156       201       208       278       285       190
## 32.842094 43.823677 17.186557 33.047540 33.092733 29.551011 44.600075 14.952176
##       104        21       246       314       308         9       271       363
```

```
## 48.251126 33.543881 44.280149 46.969772 21.922484  7.680258 40.184306 38.593530
##       340        63       178       266
## 47.519722 28.962283 45.165679 38.078799
```

The mean train and test errors are calculated below. The mean train error is much lower than the mean test error.

```
mean_test_error <- mean((test$price - predict.lm(re_lm, test)) ^ 2)
mean_train_error <- mean((train$price - predict.lm(re_lm, train)) ^ 2)
mean_test_error
```

```
## [1] 115.1524
```

```
mean_train_error
```

```
## [1] 72.30836
```

d. For this last part, we will explicitly calculate the parameter estimates using the linear regression solution (for details, see our lecture notes):

$$\widehat{\boldsymbol{\beta}} = (\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y}$$

To perform this calculation, you need to properly define the data matrix $\mathbf{X}$ and the outcome vector $\mathbf{y}$ **from just your training data**. One thing to be careful here is that the data matrix $\mathbf{X}$ should contain a column of 1 to represent the intercept term. Construct such a data matrix with `season`, `age`, `distance` and `stores`, while making sure that the `season` variable is using a dummy coding. Should your dummy variable be three columns or four columns if an intercept is already included? and Why? After obtaining the parameter estimates, validate your results by calculating the training error of your model, and compare it with the value obtained from the previous question.

I defined the X matrix and the outcome vector using the train data. The dummy variable should have three columns instead of four columns because we already included an intercept term. The reason is because the model in R defaults one variable as the intercept so if we did not have the intercept term we could have included all four variables for the seasons.

I calculated b_nought using the normal equation.

```
X = cbind("intercept" = rep(1, length(train$age)),
          "spring" = train$season == "spring",
          "summer" = train$season == "summer",
          "winter" = train$season == "winter",
  #        "fall" = train$season == "fall",
          "age" = train$age,
          "distance"= train$distance,
          "stores" = train$stores)
y = cbind("price" = train$price)
b_nought = solve(t(X) %*% X) %*% t(X) %*% y
b_nought
```

```
##                 price
## intercept 41.415758540
## spring     1.713393096
## summer     3.386888402
## winter     1.646653341
## age       -0.268606032
## distance  -0.005430919
## stores     1.327853202
```

Find the training error and compare it with the value obtained from the previous question. The difference is very small to the point of negligible.

```r
mean((train$price - X %*% b_nought) ^ 2) - mean_train_error
```

```
## [1] -1.421085e-14
```

Question 2 (model selection)

For this question, use the original six variables defined in the `realestate` data, and **treat all of them as continuous variables**. However, you should keep your training/testing split. Fit models using the training data, and when validating, use the testing data.

a. Calculate the Marrows' C_p criterion using the full model, i.e., with all variables included. Compare this result with a model that contains only `age`, `distance` and `stores`. Which is the better model based on this criterion? Compare their corresponding testing errors. Does that match your expectation? If yes, explain why you expect this to happen. If not, what could be the causes?

Here I fit a full model using 6 variables and calculated the Cp criterion.

```r
realestate_orig = read.csv("realestate.csv", row.names = 1)
train_orig <- realestate_orig[-sample_index, ]
test_orig <- realestate_orig[sample_index, ]
re_lm_full = lm(price ~ ., data=train_orig)
summary(re_lm_full)
```

```
##
## Call:
## lm(formula = price ~ ., data = train_orig)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -35.355  -5.355  -1.035   4.236  33.571
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -1.586e+04  7.469e+03  -2.124   0.0345 *
## date         4.080e+00  1.637e+00   2.493   0.0132 *
## age         -2.720e-01  4.075e-02  -6.674 1.16e-10 ***
## distance    -3.981e-03  8.082e-04  -4.926 1.38e-06 ***
## stores       1.207e+00  1.999e-01   6.040 4.45e-09 ***
## latitude     2.333e+02  4.793e+01   4.867 1.81e-06 ***
## longitude    1.536e+01  5.560e+01   0.276   0.7826
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 8.26 on 307 degrees of freedom
## Multiple R-squared:  0.6101, Adjusted R-squared:  0.6025
## F-statistic: 80.06 on 6 and 307 DF,  p-value: < 2.2e-16
```

```r
predict.lm(re_lm_full, test_orig)
```

```
##       284       173        19       381       227       158       281        47
## 32.606573 54.050606 46.687821 47.172575 13.973688 42.111073 47.844103 46.862340
##       111       188       221       228        54       267        57       183
## 44.908777 24.049901 44.949157 43.824822 41.530629 33.592679 43.249275 33.293506
##       186       177       391       261        37       124        40       398
## 30.520915 13.568268 44.790887 37.034786 30.456230 42.997127 45.924107 45.173074
##        30         3         4       355       219        43       335        59
```

```
## 45.971683 48.598073 48.259393 32.020221 42.811921 36.097831 40.005427 13.430171
##       136       414       151         2       258       187       193       110
## 31.054171 53.739122 40.344882 48.311163 38.632598 29.818369 38.322583 36.215772
##       301       300        46       239       116       240        10       142
## 45.059620 47.901125 40.111737 37.175397 38.778496 34.251552 34.246923 39.420467
##       408       358       276        88       359        96         7       236
## 27.714067 52.841600 47.978675 16.630510 47.772061 44.258614 39.252790 43.955050
##       331        22       232       327        32        55        75       337
## 23.971705 49.404103 13.585751 48.080721 41.118960 45.608543 53.373491 37.753190
##       103       220       169       338       270       202       154       294
## 47.429301 42.982049 37.136217 36.153677 32.347292 47.975105 43.167817 43.496440
##       137       196       214       395       345       354       348        83
## 44.488533 39.654077 52.119151 15.751481 28.408243 35.849897 10.919779 45.899857
##        11       198       156       201       208       278       285       190
## 33.090884 42.458218 15.389410 33.429174 33.872843 30.756901 43.863864 14.076087
##       104        21       246       314       308         9       271       363
## 44.049931 34.905419 45.782757 45.874505 22.971567  9.291696 41.454475 44.209071
##       340        63       178       266
## 50.509193 29.799607 45.078429 40.518298
```

```r
# validating
mean((test_orig$price - predict.lm(re_lm_full, test_orig)) ^ 2)
```

```
## [1] 110.6082
```

```r
mean((train_orig$price - predict.lm(re_lm_full, train_orig)) ^ 2)
```

```
## [1] 66.70402
```

```r
# Calculate the Cp criterion
p = 7
RSS = sum(residuals(re_lm_full)^2)
Cp_full = RSS + 2*p*summary(re_lm_full)$sigma^2
Cp_full
```

```
## [1] 21900.21
```

Model that contains only `age`, `distance`, and `stores` is below, I calculated the Cp criterion.

```r
train_small <- train_orig[, c('price', 'age', 'distance', 'stores')]
test_small <- test_orig[, c('price', 'age', 'distance', 'stores')]
re_lm_small = lm(price ~ ., data=train_small)
summary(re_lm_small)
```

```
##
## Call:
## lm(formula = price ~ ., data = train_small)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -37.397  -5.313  -1.202   4.391  33.872
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) 42.7113131  1.4737032  28.982  < 2e-16 ***
## age         -0.2618103  0.0425602  -6.152 2.37e-09 ***
## distance    -0.0053194  0.0004931 -10.787  < 2e-16 ***
```

```
## stores       1.3754134  0.2064437   6.662 1.23e-10 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 8.64 on 310 degrees of freedom
## Multiple R-squared:  0.5692, Adjusted R-squared:  0.565
## F-statistic: 136.5 on 3 and 310 DF,  p-value: < 2.2e-16
```

```r
# validating
mean((test_small$price - predict.lm(re_lm_small, test_small)) ^ 2)
```

```
## [1] 119.683
```

```r
mean((train_small$price - predict.lm(re_lm_small, train_small)) ^ 2)
```

```
## [1] 73.70066
```

```r
# Calculate the Cp criterion
p = 4
RSS = sum(residuals(re_lm_small)^2)
Cp_small = RSS + 2*p*summary(re_lm_small)$sigma^2
Cp_small
```

```
## [1] 23739.22
```

The full model is a better model based on the Cp criterion. The mean squared errors on the test data is also smaller for the full model. This matches my expectation because the full model has a much better R_squared value than the smaller model.

b. Use the best subset selection to obtain the best model of each model size. Perform the following:

   - Report the matrix that indicates the best model with each model size.
   - Use the AIC and BIC criteria to compare these different models and select the best one respectively. Use a plot to intuitively demonstrate the comparison of different model sizes.
   - Report the best model for each criteria. Are they the same?
   - Based on the selected variables of these two best models, calculate and report their respective prediction errors on the testing data.
   - Which one is better? Is this what you expected? If yes, explain why you expect this to happen. If not, what could be the causes?

Here's the matrix that indicates the best model with each model size.

```r
install.packages("leaps",repos = "http://cran.us.r-project.org")
```

```
##
## The downloaded binary packages are in
##  /var/folders/9c/3_mgdyf12z7dvb8rt4d60nt80000gn/T//RtmpM9YncD/downloaded_packages
```

```r
library("leaps")
# The package specifies the X matrix and outcome y vector
RSSleaps = regsubsets(x = as.matrix(train_orig[,c(1:6)]), y = train_orig[,c("price")])
summary(RSSleaps, matrix=T)
```

```
## Subset selection object
## 6 Variables  (and intercept)
##           Forced in Forced out
## date          FALSE      FALSE
## age           FALSE      FALSE
## distance      FALSE      FALSE
## stores        FALSE      FALSE
```

```
## latitude      FALSE      FALSE
## longitude     FALSE      FALSE
## 1 subsets of each size up to 6
## Selection Algorithm: exhaustive
##          date age distance stores latitude longitude
## 1  ( 1 ) " "  " " "*"      " "    " "      " "
## 2  ( 1 ) " "  " " "*"      "*"    " "      " "
## 3  ( 1 ) " "  "*" "*"      "*"    " "      " "
## 4  ( 1 ) " "  "*" "*"      "*"    "*"      " "
## 5  ( 1 ) "*"  "*" "*"      "*"    "*"      " "
## 6  ( 1 ) "*"  "*" "*"      "*"    "*"      "*"
```

```r
sumleaps = summary(RSSleaps, matrix = T)
modelsize=apply(sumleaps$which,1,sum)
sumleaps$which
```

```
##   (Intercept)  date   age distance stores latitude longitude
## 1        TRUE FALSE FALSE     TRUE  FALSE    FALSE     FALSE
## 2        TRUE FALSE FALSE     TRUE   TRUE    FALSE     FALSE
## 3        TRUE FALSE  TRUE     TRUE   TRUE    FALSE     FALSE
## 4        TRUE FALSE  TRUE     TRUE   TRUE     TRUE     FALSE
## 5        TRUE  TRUE  TRUE     TRUE   TRUE     TRUE     FALSE
## 6        TRUE  TRUE  TRUE     TRUE   TRUE     TRUE      TRUE
```

```r
apply(sumleaps$which,1,sum)
```

```
## 1 2 3 4 5 6
## 2 3 4 5 6 7
```

```r
# Comparing AIC and BIC criteria
n = nrow(train_orig)
AIC = n*log(sumleaps$rss/n) + 2*modelsize;
BIC = n*log(sumleaps$rss/n) + modelsize*log(n);
cbind("model size" = modelsize, "Our BIC" = BIC, "leaps BIC" = sumleaps$bic,
        "Difference" = BIC-sumleaps$bic)
```

```
##   model size  Our BIC leaps BIC Difference
## 1          2 1431.752 -182.8785    1614.63
## 2          3 1403.617 -211.0138    1614.63
## 3          4 1373.201 -241.4291    1614.63
## 4          5 1354.163 -260.4675    1614.63
## 5          6 1353.458 -261.1727    1614.63
## 6          7 1359.129 -255.5013    1614.63
```

```r
cbind("model size" = modelsize, "Our AIC" = AIC)
```

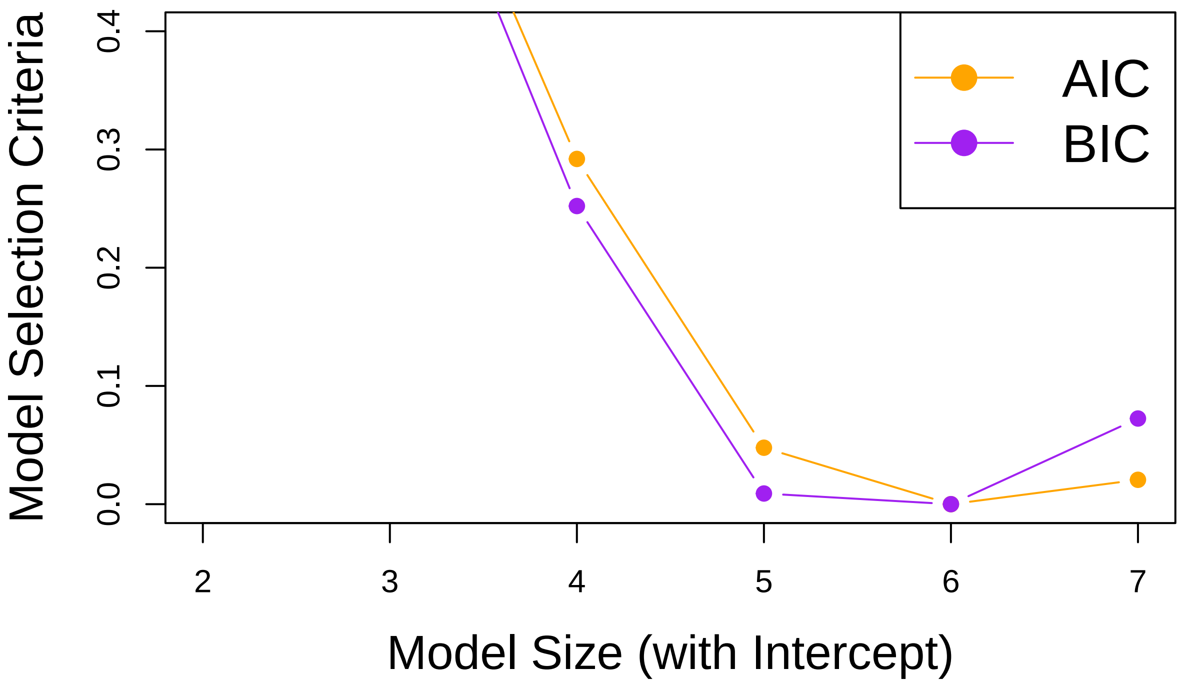
```
##   model size  Our AIC
## 1          2 1424.253
## 2          3 1392.368
## 3          4 1358.204
## 4          5 1335.416
## 5          6 1330.961
## 6          7 1332.883
```

Both criteria suggest that at p = 6 (5 variables), we have the best model (all variables) except for longitude. The best models are the same using these two criteria.

```r
inrange <- function(x) { (x - min(x)) / (max(x) - min(x)) }
    BIC = inrange(BIC)
    AIC = inrange(AIC)

    plot(range(modelsize), c(0, 0.4), type="n",
         xlab="Model Size (with Intercept)",
         ylab="Model Selection Criteria", cex.lab = 1.5)

    points(modelsize, AIC, col = "orange", type = "b", pch = 19)
    points(modelsize, BIC, col = "purple", type = "b", pch = 19)
    legend("topright", legend=c("AIC", "BIC"),
           col=c("orange", "purple"),
           lty = rep(1, 3), pch = 19, cex = 1.7)
```



We include all variables except for longitude. The testing error is reported below (~ 110.37). This is better than the small model and only slightly better than the full model.

```r
train_best <- train_orig[, c('price', 'age', 'distance', 'stores', 'date', 'latitude')]
test_best <- test_orig[, c('price', 'age', 'distance', 'stores', 'date', 'latitude')]
re_lm_best = lm(price ~ ., data=train_best)

# validating
mean((test_best$price - predict.lm(re_lm_best, test_best)) ^ 2)
```

```
## [1] 110.337
```

```r
mean((train_best$price - predict.lm(re_lm_best, train_best)) ^ 2)
```

```
## [1] 66.7206
```

c. Use a step-wise regression with AIC to select the best model. Clearly state:

- What is your initial model?
- What is the upper/lower limit of the model?

- Are you doing forward or backward?

Is your result the same as question b)? Provide a brief discussion about their similarity or dissimilarity and the reason for that.

My initial model is the model with only the intercept and the upper limit of the model is a model that includes all variables. I'm doing a forward regression. The result best model is the same as the result in question (b), where only longitude should be dropped. Best subset selection is more exhaustive than stepwise regression, as it considers all models but it is more expensive computationally. Stepwise regression is more efficient but could miss an optimal model.

```
step(lm(price ~ 1, data=train_orig), scope=list(upper=re_lm_full, lower=~1),
         direction="forward", trace=0)
```

```
##
## Call:
## lm(formula = price ~ distance + stores + age + latitude + date,
##     data = train_orig)
##
## Coefficients:
## (Intercept)     distance       stores          age     latitude         date
##  -1.403e+04   -4.148e-03    1.205e+00   -2.723e-01    2.314e+02    4.119e+00
```