# Exercise

*Ran Zhou*

*3/12/2018*

## Introduction

The goal of this exercise is to build a model to predict a home's current market value. This report will discuss the steps I used to clean data, use external data and perform feature engineering. The model I have tried are linear regression, regularized linear regression, and xgboost. After comparing the results, I have chosen linear regression.

## Data Exploration

### Part I: Overview

24 variables are given. *SaleDollarCnt* is our response variable. 3 variables are not useful for the predictive model. *PropertyID* is the identifier of each record; *censusblockgroup* and *Usecode* have the same value for all the records. The rest of the 20 variables can be grouped into four types of descriptive information.

- ***Time***: *TransDate* is from April to September in the training data, and October to next Jamuary in the data we want to make prediction. The price might have seasonal trend, so I extracted the month and other date related variables from *TransDate*. *TransQ*, *TransMonth*, and *TransWD* are generated from this variable, which means transaction season, transaction month, transaction weekday, and tranaction day from the standard day respectively.
- ***Physical Porperties***: *BathroomCnt*, *BedroomCnt*, *BuiltYear*, *FinishedSquareFeet*, *GarageSquareFeet*, *LotsizeSquarefeet*, *StoryCnt*, *ViewType*
- ***Location***: *Longitude*, *Latitude*. These two variables can be also used to impute missing values for the variables that are related to geographic location. It can be also used to gather other geographic informaiton, such as city&zipcode, nearby entertainment facilities, nearby transportation convenience, or nearby safety. I used Google Geocode API to get *City*, *Neighborhood*,*Zipcode*, and *Location Type*. Please refer to the external data part for more details.
- ***Location Porperties***: *BGMedHomeValue*, *BGMedRent*, *BGMedYearBuilt*, *BGPctVacant*, *BGPctOwn*, *BGPctKids*, *BGMedIncome*, *BGMedAge*, *ViewType*, *ZoneCodeCounty*

### Part II: External Data

***Longitude*** and ***Latitude*** can be used to gather other geographic informaiton, such as city&zipcode, nearby entertainment facilities, nearby transportation convenience, or nearby safety.

For this exercise, I only used Google Geocoding API as my external data resource because of the difficulty of finding data from the open source Yelp data for Washington region to get the nearby entertainment facilities, and the difficulty of finding the local crime statistics in King County. The variables I added to the original dataset are *City*, *Neighborhood*, *Zipcode*, *Location Type*. I used Python to get the information. The Python code are included below.

After I bring in the external data, the number of variables becomes 32.

```
import googlemaps
from datetime import datetime
import pandas as pd
import numpy as np

myapi = "AIzaSyDqBB9Y-cicN0SpVqupKyexslwp6wJEfSU"
gmaps = googlemaps.Client(key=myapi)

Train = pd.read_csv("~/Desktop/Zillow/DataScienceZExercise_TRAINING_CONFIDENTIAL1.csv")
Test = pd.read_csv("~/Desktop/Zillow/DataScienceZExercise_TEST_CONFIDENTIAL2.csv")

def get_info(df):

    # Initialize the variables
    city = []
    loc_type = []
    zipcode = []
    neighborhood = []
    for i in range(len(df)):

        # Get the information from the Google api
        loc = df[["Latitude","Longitude"]].loc[i]/1000000
        info = gmaps.reverse_geocode((loc[0], loc[1]))[0]
        add = np.array(info['address_components'])
        add_types = np.array(list(map(lambda x: x["types"][0], add)))
        c = add[add_types == 'locality']
        z = add[add_types == 'postal_code']
        n = add[add_types == 'neighborhood']

        # append the info to the initialized variable
        if len(c) == 0:
            city.append('')
        else:
            city.append(c[0]['long_name'])
        if len(z) == 0:
            zipcode.append('')
        else:
            zipcode.append(z[0]['long_name'])
        if len(n)==0:
            neighborhood.append('')
        else:
            neighborhood.append(n[0]['long_name'])
        loc_type.append(info['geometry']['location_type'])

        return (city, loc_type, zipcode, neighborhood)

### Train

city, loc_type, zipcode, neighbohood = get_info(Train)

Train['City'] = city
Train['Neighborhood'] = neighborhood
Train['Zipcode'] = zipcode
```

```
Train['LocType'] = loc_type

### Test

city, loc_type, zipcode, neighbohood = get_info(Test)

Test['City'] = city
Test['Neighborhood'] = neighborhood
Test['Zipcode'] = zipcode
Test['LocType'] = loc_type
```

### Part III: Missing Values

After I combined the information of *City*, *Neighborhood*, *Zipcode*, and *Location Type* from the external data, the figures below show the statistics of the missing data in both training set and test set.

Since a lot of variables are related to the geographic location. I can use ***K-Nearest Neighbors(KNN)*** with *Longitude* and *Latitude* as the predictors to predict and impute the missing values.

- *BGMedHomeValue* and *BGMedYearBuilt* are location properties. Impute the missing values using KNN.
- *BGMedRent* has too many missing values. It is expensive to use KNN. I will replace the missing values with the mean value of the others.
- *GarageSquareFeet* has missing values because some houses have no garage. Replace them with 0.
- *Neighborhood* has a lot of missing values. Create a new level "Other".
- *ViewType* has missing values. Create a new level "No view" since NA indicates no view.
- *Zipcode* only has one missing value. Use KNN to impute it.

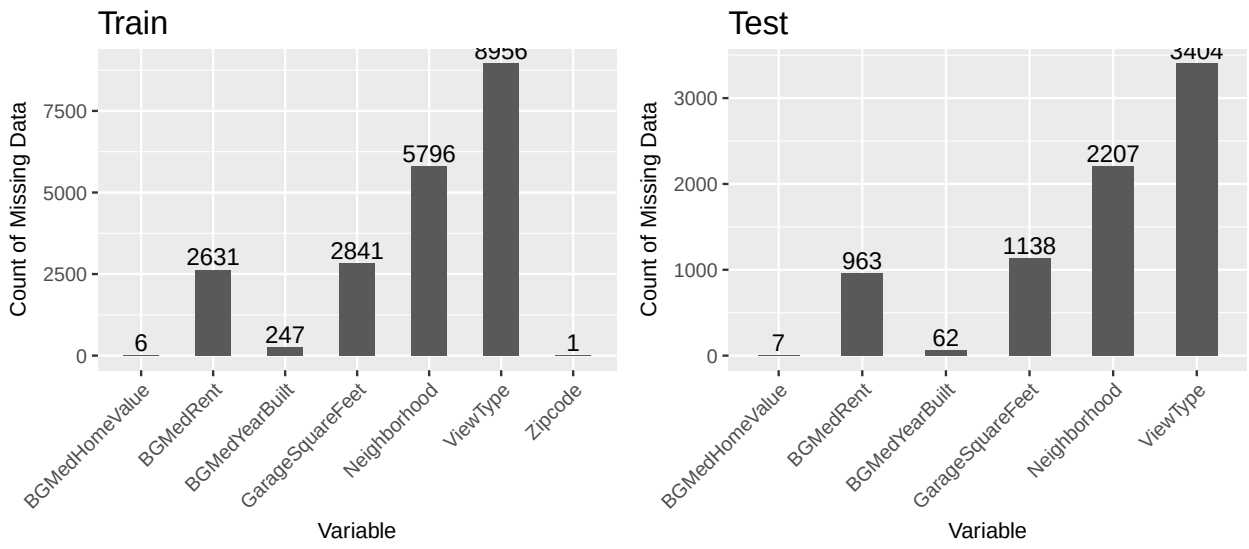
***KNN*** can be also used to clean the categorical variables that are realated to their geographic locations. For example, 'Milton' is a city only appear 6 times in the variable *City*, we can replace 'Milton' with another nearby city name to reduce the number of categories. Categorical variables that can be cleaned by KNN are: *Zipcode*, *City*, *ZoneCodeCounty*, *Neighborhood*

Since the geographic distance is not the same as euclidean distance, I will write my own KNN function using a distance calculated from package ***geosphere***. Value k will be chosen based on different variables.

```
library(ggplot2)
library(ggpubr)
library(dplyr)
library(geosphere)
library(caret)
library(reshape2)
library(knitr)

# load data with geocoded infomation
Train = read.csv("~/Desktop/Zillow/TRAIN1.csv")%>%

  # extract info from Transdate
  mutate(TransDate = as.Date(as.character(TransDate), "%m/%d/%Y"))%>%
  mutate(TransQ = quarters(TransDate))%>%
  mutate(TransMonth = months(TransDate))%>%
  mutate(TransWD = weekdays(TransDate))%>%

  # change City, ZoneCodeCounty, and Neighborhood to character for the convenience of the KNN
```

```
  mutate(City = as.character(City))%>%
  mutate(ZoneCodeCounty = as.character(ZoneCodeCounty))%>%
  mutate(Neighborhood = as.character(Neighborhood))%>%
  mutate(Neighborhood = ifelse(Neighborhood=="", "Other", Neighborhood ))

Test = read.csv("~/Desktop/Zillow/TEST1.csv")%>%
  mutate(TransDate = as.Date(as.character(TransDate), "%m/%d/%Y"))%>%
  mutate(TransQ = quarters(TransDate))%>%
  mutate(TransMonth = months(TransDate))%>%
  mutate(TransWD = weekdays(TransDate))%>%
  mutate(City = as.character(City))%>%
  mutate(ZoneCodeCounty = as.character(ZoneCodeCounty))%>%
  mutate(Neighborhood = as.character(Neighborhood))%>%
  mutate(Neighborhood = ifelse(Neighborhood=="", "Other", Neighborhood ))
```

```
# Check Missing Values
check_na = function(df,  title){
  na_cnt = apply(is.na(df%>%select(-SaleDollarCnt)) , 2, sum)

  cnt = na_cnt[na_cnt>0]
  cnt = data.frame(Variable = names(cnt), Count = cnt)
  ggplot(cnt, aes(x = Variable, y = Count))+
    geom_bar(stat = "identity", width = 0.5)+
    geom_text(aes(label=Count), vjust=-0.25)+
    ylab("Count of Missing Data")+
    ggtitle(title)+
    theme(plot.title = element_text(size=15),
          axis.title.x = element_text(size = 10),
          axis.title.y = element_text(size = 10),
          axis.text.x = element_text(angle = 45, hjust = 1))
}
p1 = check_na(Train,'Train')
p2 = check_na(Test,'Test')
ggarrange(p1,p2,ncol = 2, nrow = 1)
```



```r
# KNN for Imputing Missing Values
knn = function(df1, df2, impute_col, k, type){
  df = rbind(df1[,c("PropertyID","Longitude","Latitude", impute_col)],
             df2[,c("PropertyID","Longitude","Latitude", impute_col)])
  y = df[,impute_col]
  Train_df = df[!is.na(y),]
  Test_df = df[is.na(y),]
  # get the distance matrix
  mat = distm(Train_df[,c('Longitude','Latitude')]/1000000,
              Test_df[,c('Longitude','Latitude')]/1000000,
              fun=distVincentyEllipsoid)
  # get the estimation
  for (i in 1:dim(mat)[2]){
    idx = order(mat[,i])[1:30]
    freq = table(Train_df[idx,impute_col])
    #print(freq)
    if (type == "Num"){
      pred = as.numeric(names(freq)[which(freq == max(freq))])
    }else{
      pred = names(freq)[which(freq == max(freq))]
    }
    if (length(pred)>1) pred = pred[1]
    id = Test_df$PropertyID[i]
    if (sum(df1$PropertyID==id)>0){
      df1[df1$PropertyID==id,impute_col] = pred
    }else{
      df2[df2$PropertyID==id,impute_col] = pred
    }
  }
  return (list(df1, df2))
}

# impute missing value for BGMedYearBuilt
df = knn(Train, Test, "BGMedYearBuilt", 31, "Num")
Train = df[[1]]
Test = df[[2]]

# impute missing value for Zipcode
freq = table(Train$Zipcode)
freq_1 = as.numeric(names(freq)[freq == 1])
Train[Train$Zipcode %in% freq_1,"Zipcode"] = NA
freq = table(Test$Zipcode)
freq_1 = as.numeric(names(freq)[freq == 1])
Test[Test$Zipcode %in% freq_1,"Zipcode"] = NA
df = knn(Train, Test, "Zipcode", 13, "Num")
Train = df[[1]]
Test = df[[2]]

# deal with City
freq = table(Train$City)
freq_10 = names(freq)[freq == 10]
Train[Train$City %in% freq_10,"City"] = NA
Test[Test$City %in% freq_10,"City"] = NA
```

```
df = knn(Train, Test, "City", 30, "Cat")
Train = df[[1]]
Test = df[[2]]

# deal with ZoneCodeCounty
freq = table(c(Train$ZoneCodeCounty,Test$ZoneCodeCounty))
freq_15 = names(freq)[freq <= 15]
Train[Train$ZoneCodeCounty %in% freq_15,"ZoneCodeCounty"] = NA
Test[Test$ZoneCodeCounty %in% freq_15,"ZoneCodeCounty"] = NA
df = knn(Train, Test, "ZoneCodeCounty", 13, "Cat")
Train = df[[1]]
Test = df[[2]]

# deal with Neighborhood
freq = table(Train$Neighborhood)
freq_5 = names(freq)[freq <= 5]
Train[Train$Neighborhood %in% freq_5,"Neighborhood"] = NA
Test[Test$Neighborhood %in% freq_5,"Neighborhood"] = NA
df = knn(Train, Test, "Neighborhood", 13, "Cat")
Train = df[[1]]
Test = df[[2]]

# impute missing for GarageSquareFeet, ViewType & BGMedRent
mean_rent = mean(c(Train$BGMedRent,Test$BGMedRent), na.rm = T)
Train = Train %>%
  mutate(GarageSquareFeet = ifelse(is.na(GarageSquareFeet), 0, GarageSquareFeet))%>%
  mutate(ViewType = ifelse(is.na(ViewType), "0", as.character(ViewType)))%>%
  mutate(BGMedRent = ifelse(is.na(BGMedRent), mean_rent, BGMedRent))
Test = Test %>%
  mutate(GarageSquareFeet = ifelse(is.na(GarageSquareFeet), 0, GarageSquareFeet))%>%
  mutate(ViewType = ifelse(is.na(ViewType), "0", as.character(ViewType)))%>%
  mutate(BGMedRent = ifelse(is.na(BGMedRent), mean_rent, BGMedRent))
```

### Part III: Variable Relationships

1. ***Histograms***:

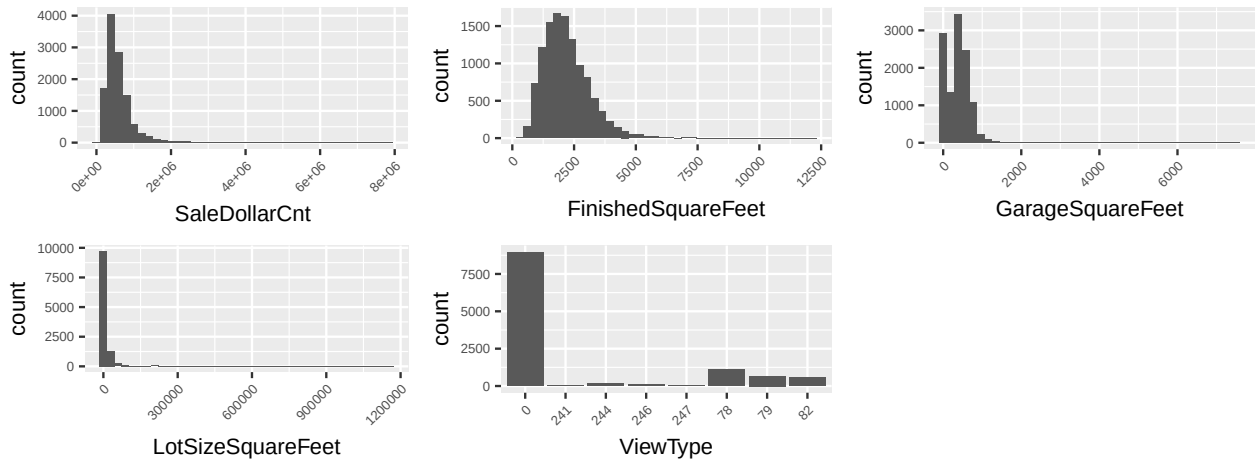
Issues identified:

- *SaleDollarCnt* and the three size-related variables, *FinishedSquareFeet*, *GarageSquareFeet*, and *LotSizeSqaureFeet* are left skewed. I performed log transformation on *SaleDollarCnt*, and I will look at the pair plots first before considering log transformation for the size-related variables.
- *ViewType* : 241 and 247 has very few data points. Classify them to a closer class 244.

```
# histograms
plot_hist = function(df, coln, continuous){
  if (continuous == "Y"){
    ggplot(df)+
      geom_histogram(aes_string(coln), bins = 40)+
      theme(axis.title.x = element_text(size = 10),
            axis.title.y = element_text(size = 10),
            axis.text.x = element_text(size = 6, angle = 45, hjust = 1),
            axis.text.y = element_text(size = 6))
  }else{
    ggplot(df)+
```

```
      geom_histogram(aes_string(coln), stat = "count")+
      theme(axis.title.x = element_text(size = 10),
            axis.title.y = element_text(size = 10),
            axis.text.x = element_text(size = 6, angle = 45, hjust = 1),
            axis.text.y = element_text(size = 6))
  }
}
p1 = plot_hist(Train, "SaleDollarCnt","Y")
p2 = plot_hist(Train, "FinishedSquareFeet","Y")
p3 = plot_hist(Train, "GarageSquareFeet","Y")
p4 = plot_hist(Train, "LotSizeSquareFeet","Y")
p5 = plot_hist(Train, "ViewType","N")
```

```
ggarrange(p1,p2,p3,p4,p5, ncol = 3, nrow = 2)
```




2. ***Pair plot*** with log transformation for *SaleDollarCnt*:

Issues identified:

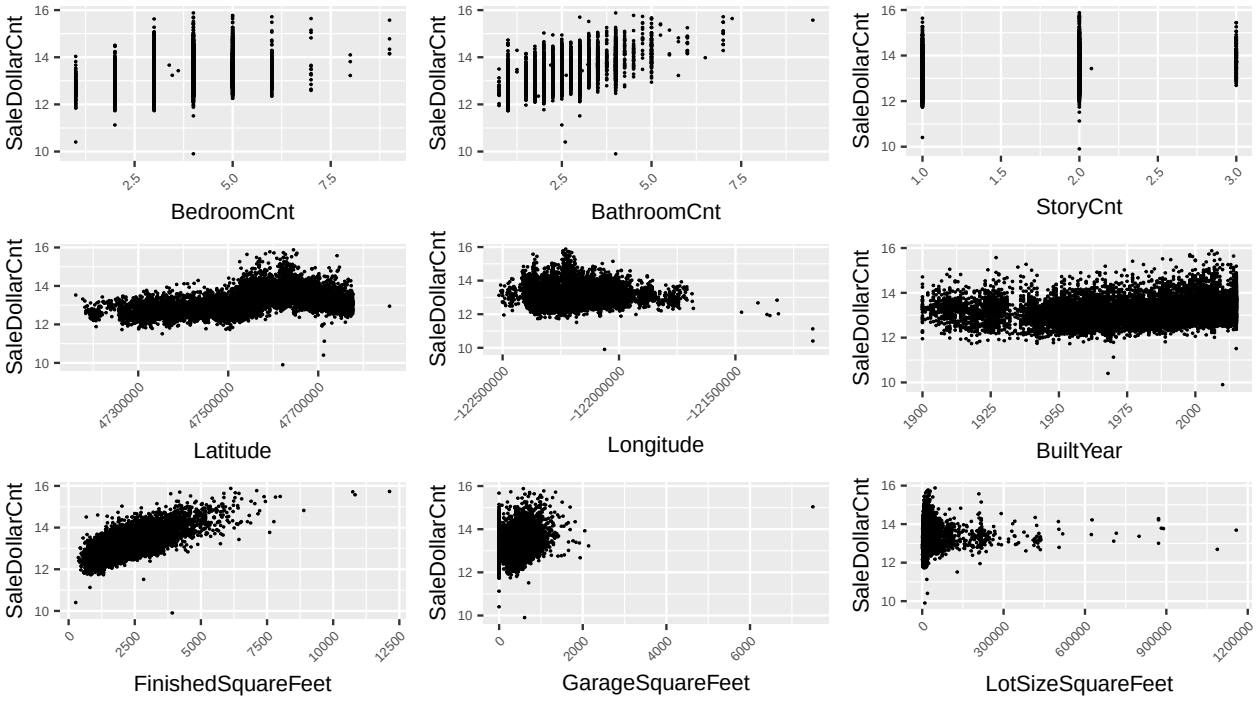
- Overall: Four data points having low house price always fall out of the main trend. Remove the outliers.
- *BedroomCnt* : a few data points has non-integer values. Correct them to be integer.
- *BathroomCnt* : a few data points having values appearing only once. Classify those to a closer value.
- *GarageSquareFeet*: one data point has a huge garage. Remove the outlier.
- *StoryCnt*: one data point has a non-integer value. Correct it to be integer.
- *Latitude*: one data point is out of the King County area. Remove the ourlier.

Variable Relationships:

- *BuiltYear*: non-linear relationship. Use second order or third order ***polynomial***.
- *FinishedSquareFeet*: relationship with price is stronger after ***log transformation***.
- *GarageSquareFeet*: no need to do log transformation after removing the outlier.
- *LotSquaredFeet*: relationship with price is stronger after ***log transformation***.
- *Latitude*: non-linear relationship. Use ***spline***.

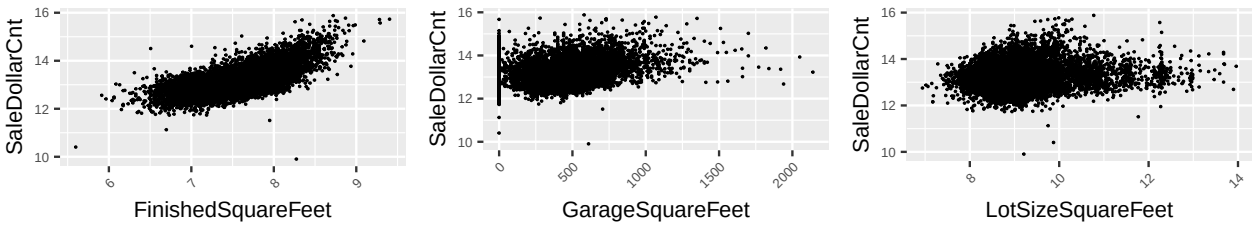
```
# pair plots
plot_price = function(df, coln, tt){
  ggplot(df %>%mutate(SaleDollarCnt = log(SaleDollarCnt)),
       aes_string(x = coln, y = "SaleDollarCnt"))+
  geom_point(size = 0.1) +
  theme(
    axis.title.x = element_text(size = 10),
```

```
    axis.title.y = element_text(size = 10),
    axis.text.x = element_text(angle = 45, hjust = 1, size = 6),
    axis.text.y = element_text(size = 6))
}
p1 = plot_price(Train, "BedroomCnt")
p2 = plot_price(Train, "BathroomCnt")
p3 = plot_price(Train, "StoryCnt")
p4 = plot_price(Train, "Latitude")
p5 = plot_price(Train, "Longitude")
p6 = plot_price(Train, "BuiltYear")
p7 = plot_price(Train, "FinishedSquareFeet")
p8 = plot_price(Train, "GarageSquareFeet")
p9 = plot_price(Train, "LotSizeSquareFeet")
ggarrange(p1,p2,p3,p4,p5,p6,p7,p8,p9,ncol = 3, nrow = 3)
```




*FinishedSquareFeet*, *GarageSquareFeet*, *LotSizeSquareFeet* after cleaning or transformation

```
p7 = plot_price(Train%>%mutate(FinishedSquareFeet = log(FinishedSquareFeet)), "FinishedSquareFeet")
p8 = plot_price(Train%>%filter(GarageSquareFeet < 6000), "GarageSquareFeet")
p9 = plot_price(Train%>%mutate(LotSizeSquareFeet = log(LotSizeSquareFeet)), "LotSizeSquareFeet")
ggarrange(p7,p8,p9,ncol = 3, nrow = 1)
```

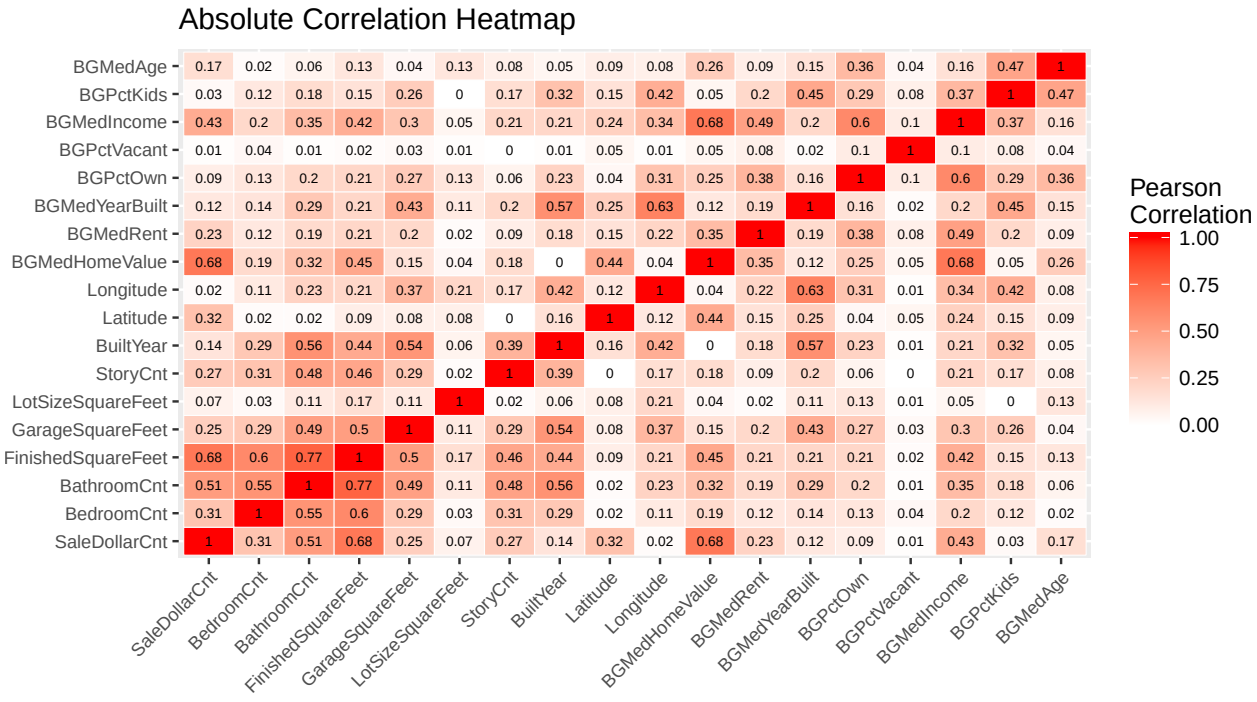



3. ***Pairwise Absolute Correlation Heatmap*** for numerical variables:

The heatmap gives me a sense of variable importance and possible collinearity.

Variables with strong correlation with *SaleDollarCnt*: * *FinishedSquareFeet* * *BGMedHomeValue* * *BathroomCnt* * *BGMedIncome*: this variable has strong correlation with BGMedHomeValue. Be careful of collinearity.

```
Train_Temp = Train%>%
  select(-c(PropertyID, Usecode, censusblockgroup,TransDate))

# list of colum names for numerical features
num_features = c()
features = colnames(Train_Temp)
for (f in features){
  if( !((class(Train_Temp[[f]]) == "character") || (class(Train_Temp[[f]]) == "factor")))
    num_features = c(num_features, f)
}

# absolute correlation matrix and heatmap
cormat = abs(round(cor(Train_Temp[, num_features]),2))
melted_cormat = melt(cormat)
ggplot(data = melted_cormat, aes(x=Var1, y=Var2, fill=value)) +
  geom_tile(color = "white")+xlab("")+ylab("")+ggtitle("Absolute Correlation Heatmap")+
  scale_fill_gradient2(low = "white", high = "red", mid = "white", midpoint = 0,
                       limit = c(0,1), space = "Lab", name="Pearson\nCorrelation")+
  geom_text(aes(Var2, Var1, label = value), color = "black", size = 2) +
  theme(axis.text.x = element_text(angle = 45, hjust = 1, size = 8),
        axis.text.y = element_text(size = 8),axis.title = element_text(size = 9))
```




### Part IV: Data Cleaning and Transformation

Based on Part III, the histograms and pair plots, the cleaning of the data is the following.

```
Train = Train %>%
  select(-c(censusblockgroup, Usecode))%>%
  filter(GarageSquareFeet < 6000)%>%
  filter(SaleDollarCnt > 100000) %>%
  filter(Latitude<47.85*10^6)%>%
  mutate(StoryCnt = round(StoryCnt))%>%
  mutate(ViewType = ifelse(ViewType %in% c("241","247"), "244", ViewType))%>%
  mutate(BedroomCnt = round(BedroomCnt))%>%
  mutate(BathroomCnt = ifelse(BathroomCnt == 1.25, 1.5,
                        ifelse(BathroomCnt>1.75 & BathroomCnt<2.125, 2,
                        ifelse(BathroomCnt>2.125 & BathroomCnt<2.375, 2.25,
                        ifelse(BathroomCnt>2.375 & BathroomCnt<2.625, 2.5,
                        ifelse(BathroomCnt>2.625 & BathroomCnt<2.875, 2.75,
                        ifelse(BathroomCnt>3 & BathroomCnt<3.25, 3.25,
                               BathroomCnt)))))))
Test = Test%>%
  select(-c(censusblockgroup, Usecode))%>%
  mutate(StoryCnt = round(StoryCnt))%>%
  mutate(ViewType = ifelse(ViewType %in% c("241","247"), "244", ViewType))%>%
  mutate(BedroomCnt = round(BedroomCnt))%>%
  mutate(BathroomCnt = ifelse(BathroomCnt == 1.25, 1.5,
                        ifelse(BathroomCnt>1.75 & BathroomCnt<2.125, 2,
                        ifelse(BathroomCnt>2.125 & BathroomCnt<2.375, 2.25,
                        ifelse(BathroomCnt>2.375 & BathroomCnt<2.625, 2.5,
                        ifelse(BathroomCnt>2.625 & BathroomCnt<2.875, 2.75,
                        ifelse(BathroomCnt>3 & BathroomCnt<3.25, 3.25,
                               BathroomCnt)))))))
```

# Model

I tried ***linear regression*** and ***regularized linear regression***. Linear regression is able to model non-linear relationships using both polynomial and splines, and it is also allow me to add variable interactions to model different relationships for different categories. However, this data have some categorical variables with a lot of levels, linear regression is not able to make selection on them. Regularized linear regression is able to make selection or shrink the parameters, but it is not able to capture the variable interactions and using non-linear relationship like splines. After I compare the results from the two type of models, I decided to use linear regression.

I have also tried xgboost, it is a model able to capture the variable interaction, non-linear relationship, and perform variable selection. However, the variable importance plot for xgboost shows that it only selects the first few levels for categorical variables. When a categorical variable has a lot of levels, it discards all the other levels which I think are somewhat important.

By all these experiments, I have chosen linear regression over xgboost and regularized linear regression. This model section will discuss the details of the modeling. For xgboost, the way to do cross validation is different from the linear regression. Since I did not use this model. I will put the code I have tried to produce the variable importance plot in the appendix.

### Part I: Linear regression

### Variable Selection

In this part, I used several ways to help me find the important vairbales They are correlation heatmap, variable importance plot of random forest, and cross-validation. Since *ZoneCodeCity* and *Neighborhood* have more than 50 categories and some levels have very few data points, which may cause rank-deficient issue, I will also further clean my data and reduce the number of levels based on the variable significance of the model.

1. ***Correlation heatmap***

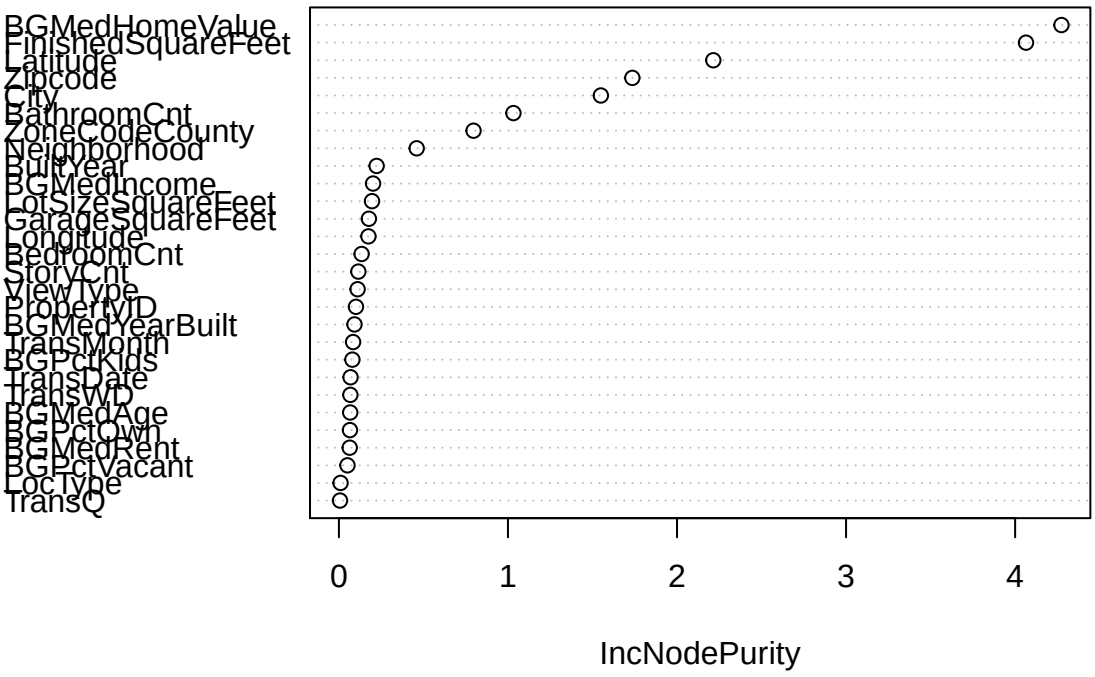
Correlation heatmap ranks the correlation for ***numeric variables***, the plot has been shown above.

2. ***Variable importance plot*** from ***random forest***

Varible importance plot is able to rank the importance for ***both numerical variables and categorical variables***. In order to run random forest, I have to reduce the number of levels to less than 53 for the categorical variables. I grouped the levels with less data points into one to reduce the categories, and created a temporary dataframe.

```
# find the levels that need to be grouped together
freq = table(Train$Neighborhood)
nei_name = c("",names(freq[freq<35]))

freq = table(Train$ZoneCodeCounty)
zone_name = names(freq[freq<29])

freq = table(Train$Zipcode)
zip_name = names(freq[freq<107])

# create temporary data set
Train_Temp = Train%>%
  mutate(Zipcode = as.factor(ifelse(as.character(Zipcode) %in% zip_name,
                                    "other", as.character(Zipcode))))%>%
  mutate(ZoneCodeCounty = as.factor(ifelse(as.character(ZoneCodeCounty) %in% zone_name,
                                    "other", as.character(ZoneCodeCounty))))%>%
  mutate(Neighborhood = as.factor(ifelse(as.character(Neighborhood) %in% nei_name,
                                    "other", as.character(Neighborhood))))%>%
  mutate(SaleDollarCnt = log(SaleDollarCnt))%>%
  mutate(FinishedSquareFeet = log(FinishedSquareFeet))%>%
  mutate(LotSizeSquareFeet = log(LotSizeSquareFeet))%>%
  mutate(ViewType = as.factor(ViewType))%>%
  mutate(TransQ = as.factor(TransQ))
```

```
library(randomForest)
fit.rf = randomForest(log(SaleDollarCnt) ~ ., data = Train_Temp)
varImpPlot(fit.rf, main = "Variable Importance")
```




3. Varible selection with ***cross-validation***

I started with the initial model. Then I try to remove the redundant variables, add interactions, and add non-linear relationships using 10-fold cross-validation. I ended up with a model with an average cross-validation AAPE around 0.114. I have also identified other outliers(1628, 7541 in the cleaned data set from previous step) using residual plots and removed them.

```
library(splines)

# Initial Model
lm_fit = function(df){
  knots.try = c( 47400000, 47500000, 47580000, 47610000, 47640000, 47680000,47700000)
  lm(log(SaleDollarCnt) ~ factor(BedroomCnt)+BathroomCnt+factor(StoryCnt)+
       log(FinishedSquareFeet)+GarageSquareFeet+log(LotSizeSquareFeet)+
       poly(BuiltYear,2)+ZoneCodeCounty+Neighborhood+Zipcode+ViewType+
       bs(Latitude,knots = knots.try)+Longitude+TransMonth+
       BGMedHomeValue+BGMedYearBuilt+BGMedAge+BGMedIncome+
       BGPctKids+BGPctVacant+BGPctOwn+BGMedRent, data = df)
}

# metric
AAPE = function(pred, actual){
  mean(abs(pred-actual)/actual)
}
MAPE = function(pred, actual){
  median(abs(pred-actual)/actual)
}
# cross-validation function
cv = function(df, model, nfold){
  acc = rep(0,nfold)
```

```
  acc_t = rep(0,nfold)
  p = list()
  fit_summary = list()
  i = 1
  folds = createFolds(1:nrow(df),nfold)
  for (idx in folds){
    my_train = df[-idx,]
    my_test = df[idx,]

    fit = model(my_train)
    pred = predict(fit, my_test)
    pred_t = predict(fit, my_train)

    acc[i] = AAPE(exp(pred), my_test$SaleDollarCnt)
    acc_t[i] = AAPE(exp(pred_t), my_train$SaleDollarCnt)
    fit_summary[[i]] = summary(fit)
    p[[i]] = data.frame(p = exp(pred), t = my_test$SaleDollarCnt)
    i = i+1
  }
  return (list(acc,acc_t,fit_summary,p))
}
```

4. Further reduce number of levels for *ZoneCodeCounty* and *Neighborhood* while training

During the process of refining my model, I also used the p-value of each coefficient to identify the levels that are not very important. I either gourped them as one, or used KNN to cluster them to a level that nearby points belong to.

```
# KNN to clean ZoneCodeCounty
Train_Temp = Train%>%mutate(ZoneCodeCounty = as.character(ZoneCodeCounty))
Test_Temp = Test%>%mutate(ZoneCodeCounty = as.character(ZoneCodeCounty))
rm_list = c("PUD","R10","R 7200","RSA 4","RSA 8","SR8")
Train_Temp[Train_Temp$ZoneCodeCounty %in% rm_list,"ZoneCodeCounty"] = NA
Test_Temp[Test_Temp$ZoneCodeCounty %in% rm_list,"ZoneCodeCounty"] = NA
df = knn(Train_Temp, Test_Temp, "ZoneCodeCounty", 13, "Cat")
Train_Temp = df[[1]]
Test_Temp = df[[2]]

# Group levels
Train_Temp = Train_Temp%>%
    mutate(Neighborhood = as.character(Neighborhood))%>%
    mutate(Neighborhood = ifelse(Neighborhood == "Maple Leaf", "Northwest Seattle",
                                 ifelse(Neighborhood %in% c("Meridian Park", "North City", "Parkwood",
                                                            "Richmond Highlands"), "Ridgecrest",
                                        ifelse(Neighborhood == "Magnolia", "Ballard",
                                               ifelse(Neighborhood == "Willow - Rose Hill", "North Rose Hi
                                                      Neighborhood)))))%>%
    mutate(ZoneCodeCounty = ifelse(ZoneCodeCounty == "R 9600","R 8400",
                                   ifelse(ZoneCodeCounty == "RS 9600","RS 7200",
                                          ifelse(ZoneCodeCounty == "UVEV","UV",
                                                 ifelse(ZoneCodeCounty == "A35","A10",ZoneCodeCounty)))
Test_Temp = Test_Temp%>%
    mutate(Neighborhood = as.character(Neighborhood))%>%
    mutate(Neighborhood = ifelse(Neighborhood == "Maple Leaf", "Northwest Seattle",
                                 ifelse(Neighborhood %in% c("Meridian Park", "North City", "Parkwood",
```

```
                                                          "Richmond Highlands"), "Ridgecrest",
                                    ifelse(Neighborhood == "Magnolia", "Ballard",
                                           ifelse(Neighborhood == "Willow - Rose Hill", "North Rose Hi
                                                  Neighborhood)))))%>%
    mutate(ZoneCodeCounty = ifelse(ZoneCodeCounty == "R 9600","R 8400",
                                   ifelse(ZoneCodeCounty == "RS 9600","RS 7200",
                                          ifelse(ZoneCodeCounty == "UVEV","UV",
                                                 ifelse(ZoneCodeCounty == "A35","A10",ZoneCodeCounty))))
```

**Final Model**

- *BedroomCnt*, *StoryCnt*: change to categorical
- *FinishedSquareFeet*: works better without log transformation
- *Zipcode*, *ViewType*: relationship between house price and house area is different for each zipcode and each view type. Add interactions with *FinishedSquareFeet*. *Zipcode* also has interaction with *BGHomeValue*.
- *LotSizeSquareFeet*: log transformation
- *BuiltYear*: third order polynomial
- *Latitude*: spline
- *City*: have coliearity with zipcode. Remove.
- *BGMedIncome*: it has high correlation with *SaleDollarCnt*, but it is also highly correlated with *BGMedHomeValue*, which is an very significant variable. Remove.
- Vraibles removed: *TransQ*, *TransWD*, *TransDate*, *TransMonth*, *LocType*, *BGPctVacant*, *BGPctKids*, *BGPctOwn*, *BGMedRent*, *BGMedAge*

```
Train = Train[-c(1628, 7541),]
lm_fit = function(df){
  knots.try = c( 47400000, 47500000, 47580000, 47610000, 47640000, 47680000,47700000)
  lm(log(SaleDollarCnt) ~ factor(BedroomCnt)+BathroomCnt+factor(StoryCnt)+
       FinishedSquareFeet*(Zipcode+ViewType)+GarageSquareFeet+log(LotSizeSquareFeet)+
       poly(BuiltYear,3)+ZoneCodeCounty+Neighborhood+
       bs(Latitude,knots = knots.try)+Longitude+
       BGMedHomeValue*Zipcode+BGMedYearBuilt, data = df)
}
results_lm = cv(Train, lm_fit, 10)
data.frame(CV = c("Mean AAPE", "Standard Deviation"),
           Test = c(mean(results_lm[[1]]),sd(results_lm[[1]])),
           Train = c(mean(results_lm[[2]]),sd(results_lm[[2]])))%>%
  kable(caption = "Linear Regression Result")
```

Table 1: Linear Regression Result

| CV | Test | Train |
|---|---|---|
| Mean AAPE | 0.1149898 | 0.1091813 |
| Standard Deviation | 0.0034200 | 0.0004470 |

**Part II: Regularized Linear Regression**

I used all the variables with some transformation as what I did for linear regression before normalizing.

- *BedroomCnt*, *StoryCnt*: change to categorical
- *LotSizeSquareFeet*: log transformation

- *SaleDollarCnt*: log transformation

```
library(glmnet)
data = rbind(Train,Test)%>%
  select(-c(PropertyID, TransDate))%>%
  mutate(StoryCnt = factor(StoryCnt))%>%
  mutate(BedroomCnt = factor(BedroomCnt))%>%
  mutate(LotSizeSquareFeet = log(LotSizeSquareFeet))%>%
  mutate(SaleDollarCnt = log(SaleDollarCnt))

# seperate numerical features and categorical features
features = colnames(data)
ohe_feats = c()
for (f in features){
  if( (class(data[[f]]) == "character") || (class(data[[f]]) == "factor")){
    ohe_feats = c(ohe_feats, f)
    levels = unique(data[[f]])
    data[[f]] = factor(data[[f]], level = levels)
  }
}

# normalizing
num_feats = features[!features %in% ohe_feats]
data[, num_feats[2:16]] = scale(data[,num_feats[2:16]])

# one-hot-encoding features
dummies = dummyVars(~ ZoneCodeCounty + BedroomCnt + StoryCnt + ViewType  + City  + Neighborhood +
                      Zipcode + LocType + TransQ + TransMonth + TransWD  , data = data)
df_all_ohe = as.data.frame(predict(dummies, newdata = data))
df_all_combined = cbind(data[,-c(which(colnames(data) %in% ohe_feats))],df_all_ohe)

# construct data matrix
train = df_all_combined[1:dim(Train)[1],]
x = model.matrix(SaleDollarCnt~., train)[,-1]
y = train$SaleDollarCnt

# cross-validation
cv.regularized = function(x, y, model, nfold){
  acc = rep(0,nfold)
  acc_t = rep(0,nfold)
  p = list()
  fit_summary = list()
  i = 1
  folds = createFolds(1:length(y),nfold)
  for (idx in folds){
    my_train_x = x[-idx,]
    my_train_y = y[-idx]
    my_test_x = x[idx,]
    my_test_y = y[idx]

    fit = model(my_train_x, my_train_y)
    pred = predict(fit[[1]], s = fit[[2]], my_test_x)
    pred_t = predict(fit[[1]], s = fit[[2]], my_train_x)
```

```
    cfs = predict(fit[[1]], type = 'coefficients', s = fit[[2]])

    acc[i] = AAPE(exp(pred), exp(my_test_y))
    acc_t[i] = AAPE(exp(pred_t), exp(my_train_y))
    p[[i]] = data.frame(p = exp(pred), t = exp(my_test_y))
    i = i+1
  }
  return (list(acc,acc_t,cfs, p))
}

# ridge
rg_fit = function(x,y){
  lambda <- 10^seq(10, -2, length = 100)
  fit = glmnet(x, y, alpha = 0, lambda = lambda)
  cv.out = cv.glmnet(x, y, alpha = 0)
  bestlam = cv.out$lambda.min
  return (list(fit, bestlam))
}
results = cv.regularized(x,y,rg_fit,10)
data.frame(CV = c("Mean AAPE", "Standard Deviation"),
           Test = c(mean(results[[1]]),sd(results[[1]])),
           Train = c(mean(results[[2]]),sd(results[[2]])))%>%
  kable(caption = "Ridge Result")
```

Table 2: Ridge Result

| CV | Test | Train |
|---|---|---|
| Mean AAPE | 0.1213614 | 0.1181853 |
| Standard Deviation | 0.0038798 | 0.0004611 |

```
# lasso
lasso_fit = function(x,y){
  lambda <- 10^seq(10, -2, length = 100)
  fit = glmnet(x, y, alpha = 1, lambda = lambda)
  cv.out = cv.glmnet(x, y, alpha = 1)
  bestlam = cv.out$lambda.min
  return (list(fit, bestlam))
}
results = cv.regularized(x,y,lasso_fit,10)
data.frame(CV = c("Mean AAPE", "Standard Deviation"),
           Test = c(mean(results[[1]]),sd(results[[1]])),
           Train = c(mean(results[[2]]),sd(results[[2]])))%>%
  kable(caption = "Lasso Regression Result")
```

Table 3: Lasso Regression Result

| CV | Test | Train |
|---|---|---|
| Mean AAPE | 0.1402478 | 0.1395279 |
| Standard Deviation | 0.0036659 | 0.0004128 |

# Appendix

### xgboost

```
library(xgboost)

# use the 'train' dataframe from regularized linear regression part
y_train = train$SaleDollarCnt
train = train%>%select(-c(SaleDollarCnt))
train_sparse = data.matrix(train)

dtrain = xgb.DMatrix(data=train_sparse, label=y_train)

gc()
```

```
##            used  (Mb) gc trigger  (Mb) max used  (Mb)
## Ncells  2495088 133.3    4703850 251.3  4703850 251.3
## Vcells 26227091 200.1   56781400 433.3 56781400 433.3
```

```
evalerror = function(preds, dtrain) {
  true_val = as.vector(getinfo(dtrain, "label"))
  aape = mean(abs(exp(preds)-exp(true_val))/exp(true_val))
  return(list(metric = "AAPE", value = aape))
}

# Params for xgboost
param = list(booster = "gbtree",
             eval_metric=evalerror,
             objective = "reg:linear",
             eta = 0.1,
             gamma = 5,
             max_depth = 15,
             min_child_weight = 1,
             subsample = .8,
             colsample_bytree = .8
             )

xgbcv = xgb.cv(params = param, data = dtrain, nrounds = 150, nfold = 5, showsd = T, print_every_n = 10,
               early_stopping_rounds = 20, maximize = F)
```

```
## [1]  train-AAPE:0.999987+0.000000    test-AAPE:0.999987+0.000000
## Multiple eval metrics are present. Will use test_AAPE for early stopping.
## Will train until test_AAPE hasn't improved in 20 rounds.
##
## [11] train-AAPE:0.980421+0.000027    test-AAPE:0.980418+0.000071
## [21] train-AAPE:0.744178+0.000087    test-AAPE:0.744025+0.000712
## [31] train-AAPE:0.373634+0.000133    test-AAPE:0.373455+0.001698
## [41] train-AAPE:0.176955+0.000369    test-AAPE:0.178136+0.003172
## [51] train-AAPE:0.138424+0.000345    test-AAPE:0.140964+0.003877
## [61] train-AAPE:0.137142+0.000347    test-AAPE:0.140079+0.004239
## [71] train-AAPE:0.138228+0.000243    test-AAPE:0.141316+0.004232
## Stopping. Best iteration:
## [56] train-AAPE:0.136659+0.000354    test-AAPE:0.139397+0.004060
```

```
rounds = which.min(xgbcv$evaluation_log$test_AAPE_mean)
xgb = xgb.train(params = param, data = dtrain, nrounds = rounds, watchlist = list(train=dtrain),
                print_every_n = 10, early_stop_round = 10, maximize = F)
```

```
## [1]  train-AAPE:0.999987
## [11] train-AAPE:0.980435
## [21] train-AAPE:0.744341
## [31] train-AAPE:0.373592
## [41] train-AAPE:0.176146
## [51] train-AAPE:0.137084
## [56] train-AAPE:0.134818
```

```
mat = xgb.importance (feature_names = colnames(train_sparse), model = xgb)
xgb.plot.importance (importance_matrix = mat)
```