PROJECT REPORT

SberBank Russian Housin Realty

Problem Statement :

Housing costs demand a significant investment from both consumers and developers. And when it comes to planning a budget—whether personal or corporate—the last thing anyone needs is uncertainty about one of their biggets expenses. Sberbank, Russia's oldest and largest bank, helps their customers by making predictions about realty prices so renters, developers, and lenders are more confident when they sign a lease or purchase a building.

Although the housing market is relatively stable in Russia, the country's volatile economy makes forecasting prices as a function of apartment characteristics a unique challenge. Complex interactions between housing features such as number of bedrooms and location are enough to make pricing predictions complicated. Adding an unstable economy to the mix means Sberbank and their customers need more than simple regression models in their arsenal.

In this competition, Sberbank is challenging Kagglers to develop algorithms which use a broad spectrum of features to predict realty prices. Competitors will rely on a rich dataset that includes housing data and macroeconomic patterns. An accurate forecasting model will allow Sberbank to provide more certainty to their customers in an uncertain economy.

Data :

The aim of this competition is to predict the sale price of each property. The target variable is called **price_doc** in train.csv.

The training data is from August 2011 to June 2015, and the test set is from July 2015 to May 2016. The dataset also includes information about overall conditions in Russia's economy and finance sector, so you can focus on generating accurate price forecasts for individual properties, without needing to second-guess what the business cycle will do.

Data Files :

- **train.csv, test.csv:** information about individual transactions. The rows are indexed by the "id" field, which refers to individual transactions (particular properties might appear more than once, in separate transactions). These files also include supplementary information about the local area of each property.
- **macro.csv:** data on Russia's macroeconomy and financial sector (could be joined to the train and test sets on the "timestamp" column)
- **sample_submission.csv:** an example submission file in the correct format
- **data_dictionary.txt:** explanations of the fields available in the other data files

Here is the link to Project:

https://www.kaggle.com/c/sberbank-russian-housing-market

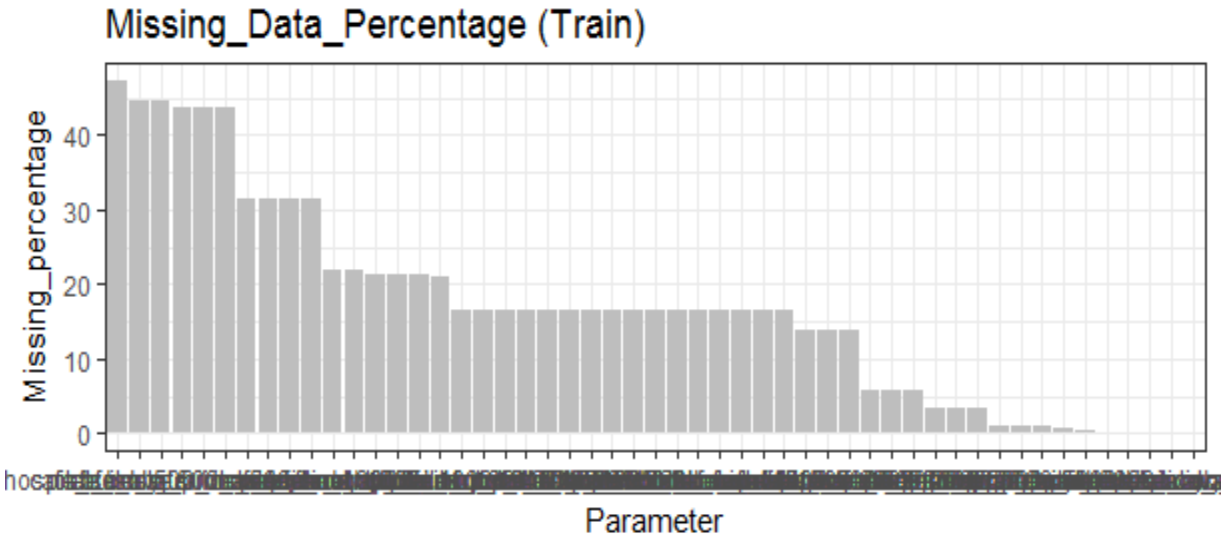
Project Description :

1.) First of all, figure out type of the Problem.
Following are the 4 possible options i.e.:
(i). Prediction/Forecasting
(ii). Classification
(iii). Optimization
(iv). Unsupervised Learning

As here in the problem statement it is given that
"In this competition, Sberbank is challenging Kagglers to develop algorithms which use a broad spectrum of features to predict realty prices. Competitors will rely on a rich dataset that includes housing data and macroeconomic patterns. An accurate **forecasting model** will allow Sberbank to provide more certainty to their customers in an uncertain economy."

which clearly shows that it is a **Prediction/Forecasting Model** as we need to predict realty prices by building a **Regression Model**.

2.) The most important and time taking part of any Analytics Project is the **Data Cleaning And Exploration Part.**
We first had a look at the data and gather some information about the number of data types and whether they are correct or not. Also, as we can see that there were a lot of NA cells which clearly shows that Missing Values need to be handled here.
So, we initiated with finding **Missing Values** in the data and visualizing them using ggplot().
The plot shows that there were 6 variables which have more than 40% Missing Value.
As we already have 292 variables, so we decided to remove all those variables with greater than 40% missing values.
Here, is the R plot of missing values.



3.) Next comes, Data Imputation, for which we have following options
(i). KNN Imputation
(ii). Central Imputation
(iii) MICE

We have tried all three of them, but **KNN** has got the maximum accuracy. The only disadvantage with KNN was the amount of time that it takes. It took nearly 2 hrs for computing all the values. So, I decide to save that imputed file as 'train_imp' in order to save time and directly upload the imputed data during next trials or iterations.

Code:
#Impute missing values
train = knnImputation(train)

4.) Imputation is followed by, Outlier detection and Removal. But directly applying outlier after imputation resulted in huge loss of Data. So, **Dimensionality Reduction** was done separately for both numerical and categorical variables.
I first tried using **corrplot()** to find out correlated independent variables but the process was to slow to perform.
Secondly, **Random Forest** was used for feature selection, but again as there are issues with accuracy of Random Forest variable importance technique and even it was taking a lot of time (waited for 6+ hours, but it didn't showed any result), so I avoided that to.
Finally, **VIF** technique solved the purpose and we removed all those variables whose VIF value was greater than 10. The total number of variables reduced to 199 from 286, which means **87 variables got removed**.
.
For categorical/factor variables, Chi-Square test was performed.
All the 16 variables have p value less than 0.05, which means all of them rejected the Null Hypothesis, and **accepted Alternate Hypothesis i.e. Dependent and Independent variables are highly related to each other.**
As a result, none of the factor variables got removed.

Code:

```
#Select all numeric data
numeric_data = train[,sapply(train,is.numeric)]
#Calculate VIF for data
vif(numeric_data)

#Chi Square test of Independence : For Factor/Categorical values
factor_data = train[,sapply(train,is.factor)]
for(i in 1:16)
{
 print(names(factor_data)[i])
 print(chisq.test(table(factor_data$product_type,factor_data[,i])))
 print("next variable")}
```

5.) As discussed, outlier removal was done, to get the best suited data for Regression Modelling.
Here, we made a for loop and applied it the whole of data and renamed all outliers as NA . And that NA was then removed using complete.cases.
So, we had a total of 854 observations and 199 variables left.

Code:

```
#Outlier detection and removal
df = train
for(i in 1:ncol(df[1:199]))
  {
  #identifying outliers using boxplot method
  val = df[,i][df[,i] %in% boxplot.stats(df[,i])$out]
  #df = df[which(!df[,i] %in% val),]
  df[,i][df[,i] %in% val] = NA #Replace outliers with NA
}

train = df[complete.cases(df),]
```

6.) We divide the train data into two parts :
Train and test.
Using the probability method, we divided Train data into 80/20 i.e, 80% as the new train data and the remaining as test data.
Train Data now has 635 observations and 199 variables, whereas Test Data has 134 observations and 199 variables. So, Random Forest Model is used for this Forecasting Problem. And in order to check the accuracy of model, MAPE was calculated.

Code:

```
#Divide the train into train and test
train.index = createDataPartition(train$price_doc, p = 0.80,list = FALSE)
train = train[train.index,]
test = train[-train.index,]
```

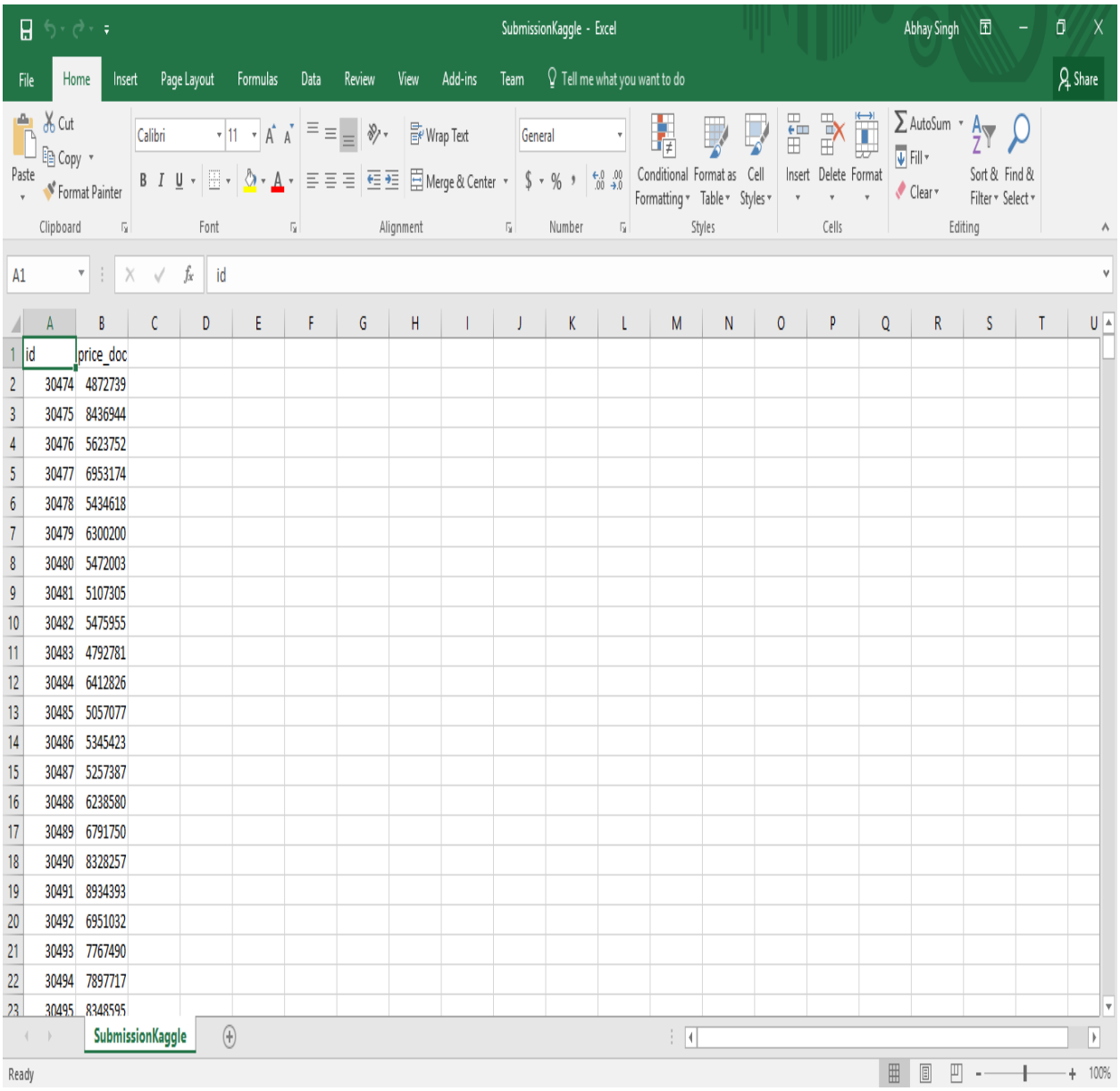
MAPE Result:

```
>
> #calculate MAPE
> mape <- function(y, yhat)
+   mean(abs((y - yhat)/y))
> mape(test[,197], pred)
[1] 0.1519304
>
```

which means that model has an accuracy of about 85%

7.) Finally, the Kaggle Test data was taken, followed by applying Pre Processing steps on the same.
We have taken the VIF data of the train and removed all those variables from test dataset to. Also, outlier was not performed on test dataset as it will cause loss of the dataset and we have to predict it for all the variables, so this is avoided.
And this test data was predicted using the Random Forest Model of ours.
Submission file was made and submitted on Kaggle.

Output :



id	price_doc
30474	4872739
30475	8436944
30476	5623752
30477	6953174
30478	5434618
30479	6300200
30480	5472003
30481	5107305
30482	5475955
30483	4792781
30484	6412826
30485	5057077
30486	5345423
30487	5257387
30488	6238580
30489	6791750
30490	8328257
30491	8934393
30492	6951032
30493	7767490
30494	7897717
30495	8348595