Written Report Part

For this project, I obtained two files from kaggle, namely analysisData.csv and scoringData.csv. ScoringData.csv is a prediction file. Compared with analysisData.csv, it lacks the price column, and we observe that analysisData contains 46 columns and the scoringData package contains 45 columns. In order to better observe the status of these columns, I decided to use the following code to determine the class status of each column for better group analysis.

lis <- as.data.frame(sapply(data,class))

From this, I found that the data consists of three major categories, numeric, integer and character. Observing the situation of the dataframe, most characters are relatively complex. Next comes a more critical step, which is to find and fill in the NA values. My initial idea was to fill in the NA values in the entire data directly through the recipe method. However, when I actually ran the code, I found that because there were too many NA values in the analysisData The huge amount of data and the excessive number of missing columns lead to frequent prediction problems, so that the recipe cannot complete the task well. I started to change my mind. Since I just found out that all columns are composed of three basic classes, I decided to separate them and fill them in the most appropriate method.

First, I separated the columns containing only numeric and integer, and prepared to use “bagimpute” in *preprocess* to fill them. However, when I looked around all the columns again, I found that the two more important columns, power and torque, are expressed in the form of characters. So I extracted the numeric items in power and torque before the impute step. This is done through the *gsub()* function in the following code.

data$power <- as.numeric(gsub(" hp.*", "", data$power))

data$torque <- as.numeric(gsub(" lb-ft.*", "", data$torque))

Subsequently, I can start extracting the numeric column for impute, below is the detailed code of the process.

numeric_columns <- which(sapply(data, function(x) is.numeric(x) || is.integer(x)))

numedata <- data[,numeric_columns]

newnum <- predict(preProcess(numedata,method = 'bagImpute'),newdata = numedata)

We first use *sapply* and *is.numeric* and *is.integer* to extract the names of all numeric columns, then extract them from the data and name them as new data frame *numedata*, and then name the data frame predicted by preprocess as *newnum*. This data frame is an all-numeric column that does not contain NA values. Next, I am going to start processing non-numeric columns. Through analysis on Kaggle, I found that some character columns contain a lot of missing items, which makes it difficult to complete, so I think it would be better to directly assign the encoding value in this case. Good choice. However, when I tried to use the definition of NA for encoding, I found that these character columns contained too many categories, which caused overload when I used random forest prediction later, that is, too many categories. Therefore, I decided to select columns with fewer categories for analysis. Here I choose the following columns *isCab, is_cpo, is_new, has_accidents, engine_type, transmission_display* for encoding. I expected to convert the NA values into the text "No Data", but when I actually operated it, I found that the NA values had not been reduced, so I started looking at the definition of each column on kaggle again. At this time, I discovered that these columns did not contain NA values, but null values in the form of "", so I used the following method to perform fill-in encoding. Taking *isCab* as an example, the code is as follows.

newnum$isCab <- ifelse(newnum$isCab=="", "No Data", newnum$isCab)

When completing the filling, I also added the character column to *newnum*, so that the data cleaning and merging were completed.

Next is the selection and prediction of the data model. First, I tried to use the simplest linear regression model by *lm()* to predict the data, and only used numeric parameters, but the effect was not very satisfactory. The *rmse* was obviously too large, so I decided to adjust the model. Observing the data, for multi-class and composite category parameters, I decided to use randomforest to match the data. But I did not use randomforest directly. Before starting to predict, I found that using tune train can better adjust the model parameters, so I chose to use *mtry* as the parameter to be adjusted, and used 5 folds cross validation to adjust the data. Finally, the *train()* function is used to repeatedly verify the model to select the best *mtry* value. Following is the code.

```
trControl = trainControl(method = 'cv', number = 5)
tuneGrid = expand.grid(mtry = 1:ncol(newnum)-1)
tm=train(price~.,data=newnum,method="ranger",num.trees=200,trControl=trControl,tuneGrid=tuneGrid)
mtry=tm$bestTune$mtry
```

After obtaining the best *mtry* through *train()*, I brought the parameters into the *randomforest* of 200 trees and used this model to match the data. Then we start the final prediction step, but before that, we first process the *scoringData* data in the same way. This is used to avoid the impact of NA values and problems caused by mismatched column numbers. Finally, name the final *scoringData* as *newnumscore* and take it into *pred()* for the predicted value. Following is the code.

```
model <- randomForest(price~.,newnum,mtry=mtry,ntree=200)
pred = predict(model,data=newnumscore)
```

The above are the overall analysis steps, but during this period I developed many other prediction methods such as bag and NLP, but the results were not as good as random forest, so I finally decided to use this version.