Samaa Nadkarni

PAC : Predict the Price of a Used Car

Introduction

This project utilises a dataset of 50,000 used cars. Each data attribute describes the features and condition of the car, to be used to build a predictive model that predicts sale price for these cars. This model is evaluated based on RMSE.

Data Attributes

The dataset contains 46 distinct attributes, with the target variable being price (list price of the car). The 45 features and condition attributes range from car mechanic descriptors (model/make name, fuel type, mileage, horsepower etc), aesthetic descriptors (height, width, length, colour etc) and conditional descriptors (has_accidents, frame_damaged etc).

Data Distribution & Plotting Variable Correlation to Price

Before starting the cleaning process, different graphs have been made to assess the relationships between all the variables in our dataset against price to find interesting patterns that can help with model construction. The most expensive cars are just shy of $35,000, with most car prices falling in the $7000-$20,000 range.

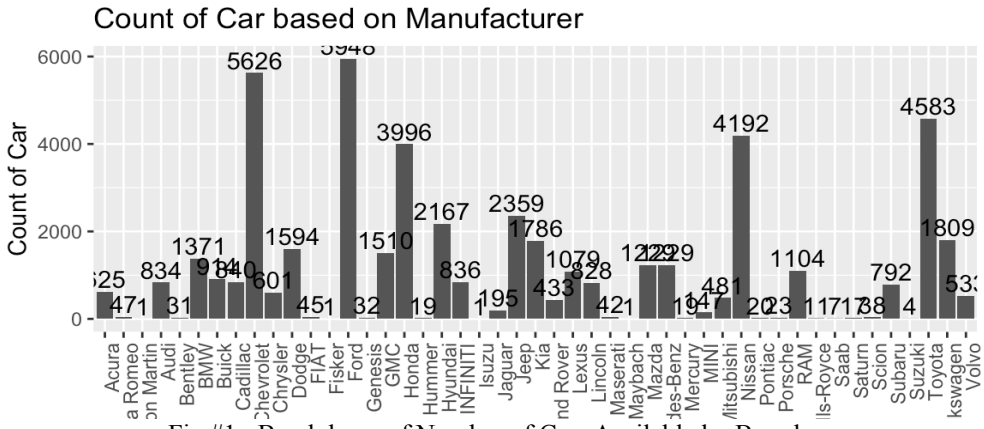

Fig #1 : Breakdown of Number of Cars Available by Brand

In Fig #1, of the four most common car brands being sold in our data set, two are traditionally considered high-end brands while the other two are affordable. This information was used as the basis of creating an is_high_end and is_affordable boolean variable based on identifying cars that fell into the luxury brand category versus the affordable, and taking a closer look at the relationship between brand name and price.

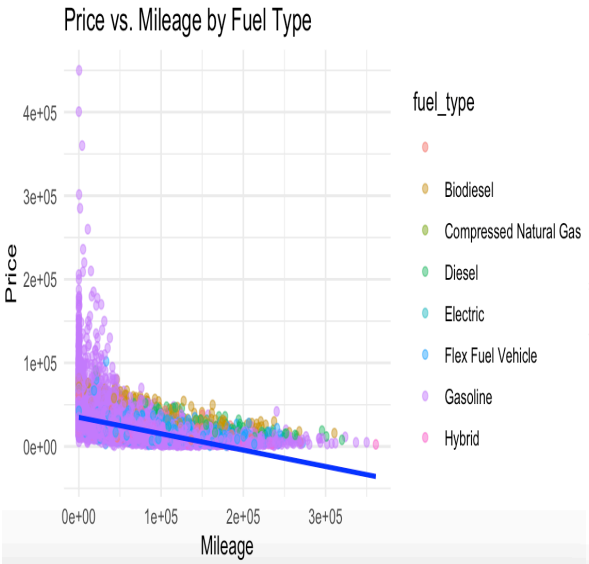

Fig #2 : Scatterplot of Mileage by Fuel Type against Price

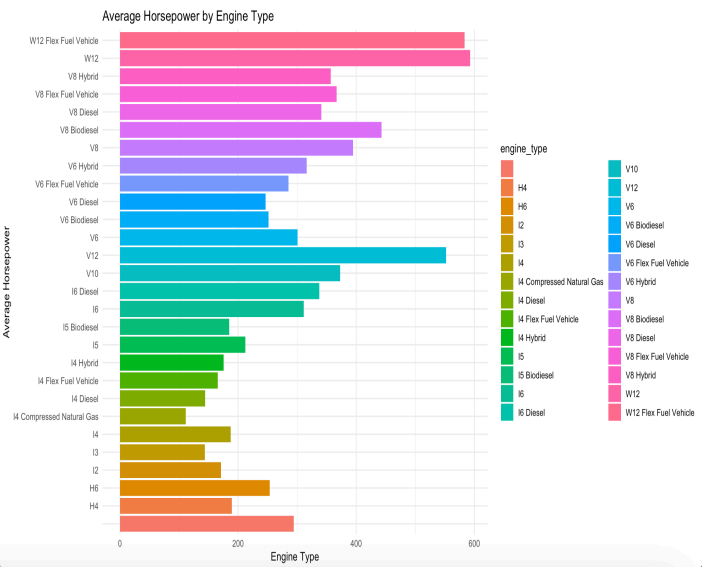

Fig #3 : Horsepower by Engine Type

The relationship between variables like mileage, fuel_type and horsepower was also investigated as these seem like important factors about a cars performance that would significantly impact pricing. Fuel_type and horsepower were also fields that had a lot of NULL and blank (“ ”) values so studying the relationship helped with finding solutions for cleaning the data. From the above graphs, we see that Gasoline is the most commonly used fuel_type across the board for mileage. This is echoed in Fig #3 as most of the engine types with high horsepower utilise Gasoline. This information was helpful in cleaning the fuel_type, high_way_fuel_economy, city_fuel_economy, and fuel_tank_volume_gallons variables.

Data Exploration, Cleaning & Transformation

Upon initial inspection, more than 5% of the dataset was found to contain NULL values. Seeming too high a percentage to ignore, the NULLs are replaced with means (if variable column is type numeric) and modes (if variable column is type char). After this, each column in the dataset is checked for blank (“ ”) values. Depending on the column, these blanks are also replaced with the mode of that column.

```
#replacing NULLs in num columns with the mean and NULLs in char columns with mode
for (i in 1:ncol(data)) {
  if (is.numeric(data[, i])) {
    data[is.na(data[, i]), i] <- mean(data[, i], na.rm = TRUE)
  } else {
    data[is.na(data[, i]), i] <- names(sort(table(data[, i]), decreasing = TRUE))[1]
  }}
sum(is.na(data))
```

For certain variables, whose values are dependent on other variables, blanks were handled differently. For example, the fuel_type column has values that depend on information in the fuel_tank_volume_gallons column. So for all blank values in fuel_type, this is replaced with “Gasoline” if fuel_tank_volume_gallons has a value greater than 0, and replaced with “Electric” if fuel_tank_volume_gallons is equal to 0. The power variable has a string containing horsepower and revolutions per minute. Since the dataset already has a separate horsepower column, it seemed prudent to parse out the RPM value into its own column. A lot of variables contain boolean “True” and “False” values that are stored in the dataset as character type variables. These were cleaned up and transformed into numeric type variables containing 1s and 0s. Similarly, the listed_date variable is converted from a character type to a date type and then a numeric type variable.

```
#fuel type
#Add "Gasoline' when fuel_tank_volume_gallons > 0 and fuel_type is blank
data$fuel_type <- ifelse(data$fuel_tank_volume_gallons > 0 & data$fuel_type == "",
                  "Gasoline", data$fuel_type)
#Add 'Electric' when fuel_tank_volume_gallons = 0
data$fuel_type <- ifelse(data$fuel_tank_volume_gallons == 0 & data$fuel_type =="",
                  "Electric", data$fuel_type)
sum(data$fuel_type == "")
data$fuel_type = as.factor(data$fuel_type)
```

Grep function is utilised to create new columns using values present in existing columns that might be helpful for creating our model. is_high_end, parses out high end brands such as Mercedes-Benz, Audi, Rolls-Royce, Porsche from the make_name column and assigns a boolean of 1 or 0 if a car meets this characteristic. Similarly, is_affordable is created where car brands like Ford, Toyota, Nissan are given a value of 1 if they fall under the affordable car umbrella. This is done to examine the relationship between brand and car price, as it is commonly theorised that luxury car brands sell for more on average than their budget-friendly counterparts.

From the major_options and description column, features such as backup camera, sunroof, navigation system, bluetooth, maintained, warranty amongst others are parsed and used to create their own boolean columns.

```
#figure out grep
unique(data$make_name)
data <- data %>%mutate(is_high_end = if_else(  grepl("Mercedes-Benz|BMW|Audi|Lexus|LandRover|
Jaguar|Porsche|Maserati|Alfa Romero|Bentley|Rolls-Royce|Aston Martin",make_name, ignore.case =
TRUE), 1, 0))
data <- data %>%mutate(is_affordable = if_else(   grepl("Chevrolet|Ford|Toyota|Nissan|Dodge|Hyundai|
Subaru|Chrysler|Honda|Jeep|Kia|Volkswagen|Mazda|Buick|Mitsubishi|Genesis|FIAT",make_name,
ignore.case = TRUE), 1,0))
data <- data %>%mutate(Backup_Camera = ifelse(grepl("backup camera", major_options, ignore.case =
TRUE), 1, 0),Alloy_Wheels = ifelse(grepl("alloy wheels", major_options, ignore.case = TRUE), 1,
0),Bluetooth = ifelse(grepl("bluetooth", major_options, ignore.case = TRUE), 1, 0),Heated_Seats =
ifelse(grepl("heated seats", major_options, ignore.case = TRUE), 1, 0),Navigation_System =
ifelse(grepl("navigation", major_options, ignore.case = TRUE), 1, 0),Remote_Start = ifelse(grepl("remote
start", major_options, ignore.case = TRUE), 1, 0),Sunroof = ifelse(grepl("sunroof", major_options,
ignore.case = TRUE), 1, 0),CarPlay = ifelse(grepl("car play|carplay|apple carplay|android auto",
major_options, ignore.case = TRUE), 1, 0),Test_Drive = ifelse(grepl("test", description, ignore.case =
TRUE), 1, 0),Leather = ifelse(grepl("leather", description, ignore.case = TRUE), 1, 0),Maintained =
ifelse(grepl("maintained", description, ignore.case = TRUE), 1, 0))
```

All of the above transformations are also applied to the Scoring Data to handle NULLS, blanks and other data messiness as well as to create the new columns we will be using for our prediction. The data is then split into a train and test set with a 70:30 ratio and a seed of 1031.

Random Forest

From the insights derived during our above exploration and their impact on price, a Random Forest model was initially chosen to predict the price of the cars. This model was iteratively refined to evaluate the significance of specific variables in predicting price. This model was also the least efficient as it took the longest to run.

```
forest <- randomForest(price~ model_name+body_type+fuel_tank_volume_gallons+fuel_type+
highway_fuel_economy+
city_fuel_economy+torque+transmission+ wheel_system+ back_legroom_inches+
length_inches+width_inches+height_inches+ engine_type + engine_displacement+
horsepower+daysonmarket+ exterior_color+maximum_seating+ year+ fleet+ frame_damaged+
franchise_dealer+has_accidents+ isCab+is_cpo+ is_new+ mileage+ salvage+ seller_rating+ RPM+
is_high_end+ is_affordable+ Backup_Camera+ Alloy_Wheels+ Bluetooth+Heated_Seats+
Navigation_System+ Remote_Start+ Sunroof+ CarPlay+ Test_Drive + Leather + Maintained,
data=train_data, ntree = 1000)
```

RMSE with Test Data and Scoring Data

The final calculated RMSE for the train sample is 1824.563, which seems respectable. When assessed using the test_data this RMSE increases to 3450.38 which, while still a good first model RMSE, is not ideal and requires improvement. Upon submission, the Scoring Data generated an RMSE of 2429.981 on the public leaderboard which was much lower than the test data RMSE. After much fine tuning, the RMSE did not seem to improve which warranted the use of a new model.

XG Boost

To further enhance our predictive accuracy and manage the complexity of the dataset, an XGBoost model was also employed. XGBoost was chosen due to its ability to handle large datasets efficiently and its superior performance in capturing nonlinear relationships and interactions

between features. Additionally, XGBoost includes built-in regularisation features which help prevent overfitting, making it an excellent complement to the Random Forest model. This model was tweaked quite a bit from its first implementation to the final submission. Initially, a correlation matrix was used to examine the relationship between the numeric type variables with price. This was used as the basis of the first XGBoost model, where without much cleaning only numeric variables were used to train the first model. This was used with an nrounds of 100. This yielded an RMSE of 3726. 082 with the test data and 2478.738 using Scoring Data.

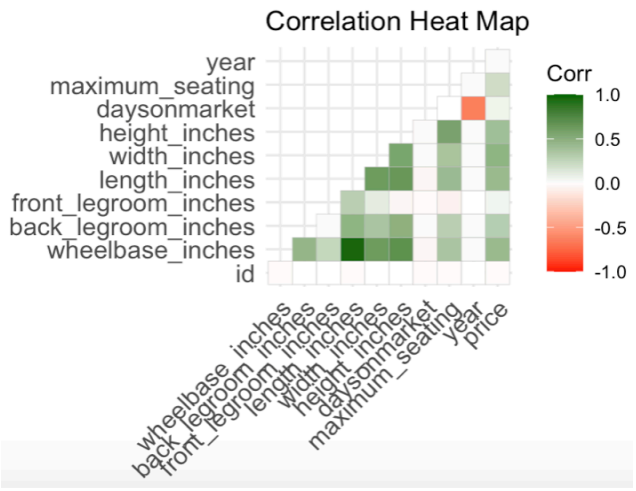

Fig #4 : Heatmap of Numeric Variables Correlation to Price

```
#Lasso
lasso_model <- cv.glmnet( x = as.matrix(train_data), y = train_data$price, alpha = 1,nfolds = 5)
optimal_lambda <- lasso_model$lambda.min #optimal lambda
selected_features <- predict(lasso_model, newx = as.matrix(X_train), s = optimal_lambda, type =
"nonzero")
selected_features <- as.numeric(unlist(selected_features))
#XG Boost
dtrain <- xgb.DMatrix(data = as.matrix(train_selected), label = y_train)
dtest <- xgb.DMatrix(data = as.matrix(test_elected), label = y_test)
xgb_model <- xgboost(data = dtrain, label = train$price, nrounds = 5000, params = params,
early_stopping_rounds = 5, nthread = 8)
y_pred <- predict(xgb_model, newdata = dtest)
rmse_xg <- sqrt(mean((y_test - y_pred)^2))
```

RMSE with Test Data and Scoring Data

The data was then cleaned further, converting variables to factors and then numerics as well as transforming any boolean variables in character format to numeric format to assist with model creation. This additional cleaning coupled with increasing the nrounds to 1000 for the train data helped reduce the RMSE of the Scoring Data to 1848.988. A lasso model was then built to aid in features selection and find the most useful variables for the xgboost model. These new features were then used to create a new training set for the model, with nrounds set as 5000. This final tuning and cleaning culminated in an RMSE of 1335.422 with the Scoring Data. This was the lowest RMSE achieved using the above cleaning and modelling techniques.

Results & Conclusion

Throughout this project, predictions were alternated between random forest modelling and xgboost modelling, to observe if findings and knowledge gathered from one model could help improve the other. The lasso model was implemented towards the end of the project for fine tuning the features being used in the xgboost model, but due to time constraints could not be improved upon as much as random forest and xgboost were worked on.

Overall, the xgboost model, in its many forms and iterations, always proved to be superior to the random forest model in generating predictions that had the lowest RMSE. All values could have been improved if further cleaning, clever utilisation of grep on the description variable, better manipulation and filling of NULL and blank values had been incorporated, and the xgboost model had been tuned. Nevertheless, this project was a great exercise in creativity and visualisation of the many ways in which data can be manipulated and improved for predictive modelling.