“Predictive Analysis of Used Car Prices” Project Report by Dave John

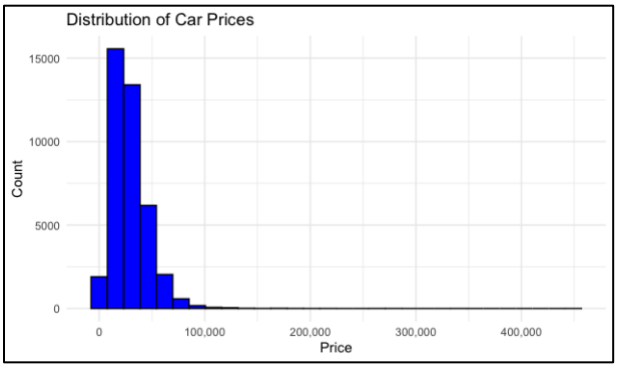
In the evolving landscape of data-driven decision-making, our recent project embarked on an analytical journey to predict used car prices. The endeavor utilized two datasets: ‘analysisData.csv’ and ‘scoringData.csv’, encompassing a diverse range of car features. Our objective was to extract meaningful insights from these datasets and build a predictive model using advanced statistical methods to determine the price of any car.

Data Exploration and Initial Observations

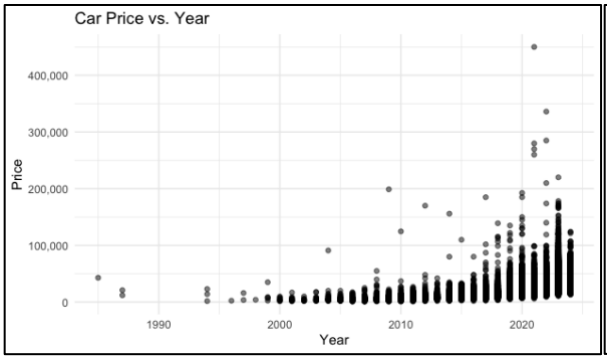
My initial step involved a comprehensive exploration of the dataset. Utilizing R's capabilities, I initially started with basic commands to understand the data's structure. The "str(data)” command provided a snapshot of the dataset's composition, revealing a mix of numerical, logical, and character variables. Similarly, “summary(data)” offered a statistical summary, highlighting key variables like the target ‘price'. This stage was crucial in identifying the range and nature of variables at our disposal.

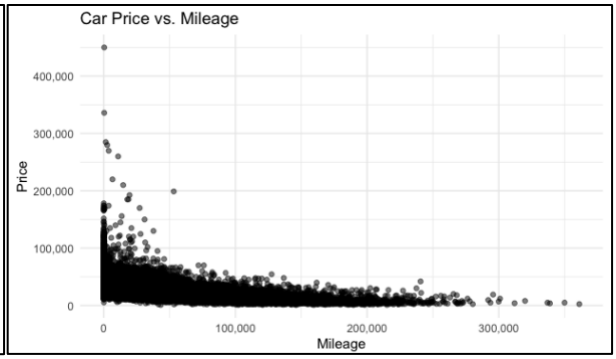
Delving into the data, my thought was to visually explore the variables. Using R, I created a series of plots that brought the data to life. Here are some of them:

1. *Distribution of Car Prices*: I started by plotting a histogram of car prices. To my intrigue, the distribution was right-skewed, indicating a higher concentration of cars in the lower price range. This skewness led me to ponder the economic factors influencing car pricing.



2. *Scatter Plots*: Next, I plotted price against numerical variables like mileage, year and horsepower. These scatter plots were revealing – they showed a clear trend of depreciation where prices tended to decrease with increasing mileage and age.





Data Preprocessing

For starters, I would play around with only a few variables and clean them to make sure I have no 'NA' or blank fields in those specific columns. While running linear models on these variables I noticed that I recognized the importance of data quality. I then embarked on an extensive data cleaning process. My approach was methodical, addressing missing values and ensuring data type integrity for the whole dataset. A significant step I took was the imputation of missing values. For numerical columns, I employed a mean imputation strategy. This choice was driven by the need to maintain statistical balance in the dataset.

Character columns posed a unique challenge. We observed that some character columns were essentially logical, containing only 'TRUE' or 'FALSE' values. To streamline these columns, I transformed those columns to logical class. This transformation simplified the dataset and prepared it for more advanced analytical techniques.

For other character columns, we addressed missing values by imputing the mode. Here, a custom function 'getmode' played a pivotal role, dynamically identifying the most frequent value within each column.

One more transformation I had carried out, was to mutate the engine_type variable, ensuring it reflected only the essential information. By splitting and extracting the primary engine type , which would be the first word in each field, I aimed to enhance model clarity without sacrificing too much detail. This step, while seemingly small, was significant in tailoring the data for the analytical narrative I sought to unfold.

```
# Create a function that can calculate mode
getmode <- function(v) {
  uniqv <- unique(na.omit(v))
  return(uniqv[which.max(tabulate(match(v, uniqv)))])}

data_clean <- data %>%
  # Impute columns to not have blank fields throughout the dataset, changed to NA
  mutate_if(is.character, ~na_if(trimws(.), "")) %>%
  # Impute Numerical columns that have NA values to mean of their respective columns
  mutate_if(is.numeric, ~ifelse(is.na(.), mean(., na.rm = TRUE), .)) %>%
  # Transform the character columns to logical columns if it's only TRUE, FALSE, or NA values
  mutate_if(~is.character(.) && all(na.omit(.) %in% c("TRUE", "FALSE")), as.logical) %>%
  # Impute Logical columns that have NA values to say "FALSE"
  mutate_if(is.logical, ~ifelse(is.na(.), FALSE, .)) %>%
  # Streamlined the engine_type column to only show the first word in each field
  mutate(engine_type = str_split(engine_type, " ", simplify = TRUE)[, 1]) %>%
  # Impute Character/Categorical columns that have NA values to mode of their respective columns
  mutate_if(is.character, ~ifelse(is.na(.), getmode(.), .))
```

Data Preprocessing Code Chunk

Model Building and Selection

With a preprocessed dataset, my focus shifted to model building. I initially gravitated towards linear models. This choice was rooted in their simplicity and interpretability. My approach involved selecting variables based on my intuition and running linear regression models. I used code snippets like "lm(price ~ variable1 + variable2, data = data_clean)" to explore these relationships. However, I soon realized the limitations of linear models in capturing the complex nonlinear relationships in our dataset as my RMSE scores had plateaued in the 7000's.

Motivated by the need for a more robust solution, I turned to going through different tuning models and had chosen XGBoost as an effective model as its prowess in handling complex datasets with a mix of variable types. The transition from linear models to XGBoost marked a significant shift in my analytical strategy. My scores dropped down to the 1800's in doing so. I was motivated by its robustness in handling complex datasets. To construct the XGBoost model, I first prepared the input using vtreat, a powerful package for feature engineering and selection. The 'designTreatmentsZ' function from vtreat allowed me to identify and prepare the variables in the 'varlist' provided.

The XGBoost model was set up with a substantial number of rounds (nrounds=10000) and an early stopping mechanism to prevent overfitting. The model's training process was a delicate balance between computational efficiency and predictive accuracy.

```
#construct a  XGBOOST model
model = designTreatmentsZ(dframe = data_clean,
                          varlist =  names(data_clean)[c(2,5,7,8,9, 12,14, 16,23:25, 31,34:36,39,42,44,45)])
newvars = model$scoreFrame[model$scoreFrame$code%in% c('clean','lev'),'varName']

model_input = prepare(treatmentplan = model,
                      dframe = data_clean,
                      varRestriction = newvars)
model_boost = xgboost(data=as.matrix(model_input),
                      label = data_clean$price,
                      nrounds=10000,
                      verbose = 0,
                      early_stopping_rounds = 100)
```

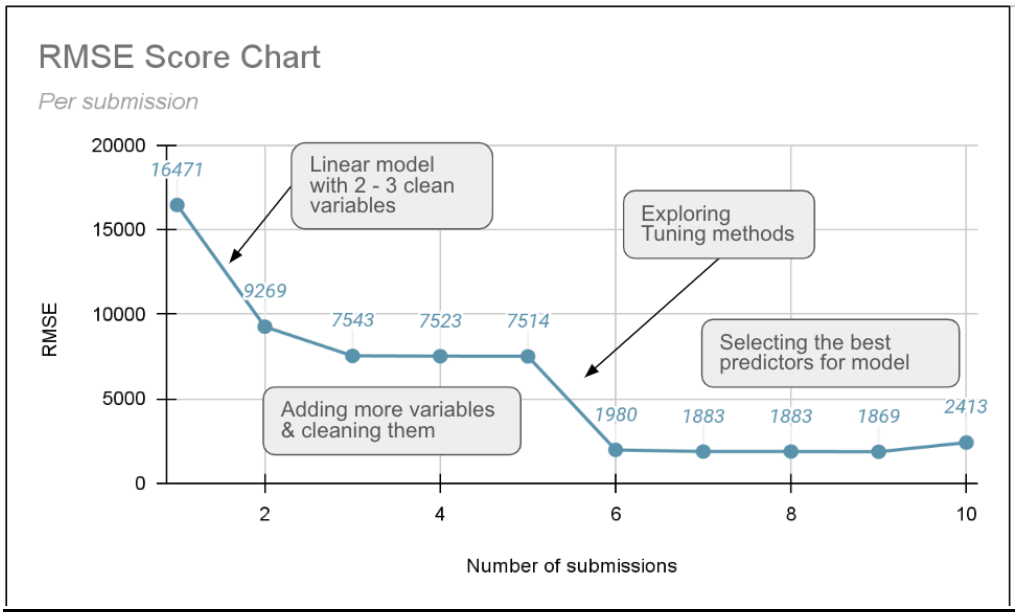
XGBoost Modeling Code Chunk

Challenges and Missteps

Throughout the project, I had faced several challenges. The computational demand was significant, especially given the large number of iterations required for the XGBoost model. I also encountered complexities in feature selection, where my initial choices did not significantly impact the model's predictive ability. This is because I had mostly chosen my variables using my own perception and bias most of the time. I had also used the p-values of some of the variables as indicators of whether I should keep them in my model or not. This occurred during the brief time I was playing around with linear models in the beginning during my earlier submission works. I realize now that a more thorough and data-driven selection of predictor variables could have significantly enhanced the model's performance. These missteps were valuable learning opportunities, highlighting areas for improvement in future projects.

Results and Performance

While the XGBoost model outperformed the initial linear models, indicating its suitability for complex datasets, the journey to this result was not without its challenges. The computational intensity of the model, coupled with a less optimized feature selection in the early stages, meant that the full potential of the model might not have been realized. However, the learning curve was steep and immensely valuable. Please see find below graph depicting the journey of this project in terms of RMSE scores.



Progression of Scores

The final model exhibited promising results. Its performance was evaluated based on the best iteration, as indicated by "model_boost$best_iteration". I would then generate predictions on the scoring dataset and construct the submission file using "write.csv(submissionFile, 'submission_9.csv', row.names = F)". The culmination of effort was reflected in the calculated Root Mean Square Error (RMSE) values as my scores, providing a quantitative measure of the model's accuracy.

Conclusion and Learnings

This assignment was not only an exercise in predictive modeling, but also an exploration of the complexities of data processing. I discovered the value of thorough data pretreatment, the strength of robust modeling techniques, and the relevance of combining computing demands with analytical depth. The experience from carrying out this project provides a solid platform for future data analytics attempts.

Appendices

The report is supplemented by commented R code, providing a transparent view of the analytical process and ensuring reproducibility of the results.