BMW Used Car Sales - Baseline Price Prediction

Nick Bultman

4/25/2021

Contents

Overview

The used car market is tricky, especially as a seller trying to price vehicles. There are so many factors that go into the selling price where it quickly becomes overwhelming when determining the acceptable price. I experienced this firsthand when selling my first moped. What year is my moped? How many miles does it have? Does it have any scratches? What did I buy it for? What have other similar mopeds sold for? These are just some of them.

Another layer of difficulty comes when you are doing this professionally. The purpose of this report seeks to answer whether or not data science can be used to assist a BMW pricing analyst in accurately and efficiently creating a baseline price for used BMW cars with historical data to better ensure company profitability.

Analysis Design

Before jumping into the analysis, it is important to outline the problem, the approach, and the criteria/metrics used to evaluate success.

Problem Statement

As a pricing analyst, one of the responsibilities is to ensure that any used inventory is properly priced based on its conditions, whatever those may be (for cars this could be mileage, body condition, year, model, etc.) (Zippia, n.d.). While they can start from scratch and deduce a price, there are a couple problems with this approach that can affect company profitability. One problem is that the pricing process takes time. Specifically for BMW, the longer it takes for a vehicle to be priced, the longer it will take for the vehicle to be sold, leaving potential profits on the table. Another problem is accuracy. Again for BMW, the more variables a pricing analyst has to wrangle with manually for a used vehicle, the more likely human error can creep in and result in profit loss (for example, buyer interest is low or nonexistent for a used vehicle because it was mispriced due to human error). If an analyst had an accurate baseline price given to them prior to their analysis, not only would this speed up their total pricing process but it would also help them be more accurate given there are less variables in total to deal with.

Given these problems as a BMW pricing analyst, this report seeks to answer the following question: can data science be used to not only establish a more accurate price baseline for used vehicles needing to be priced given the variables available through historical data but also to speed up the pricing process and better ensure company profitability?

Problem Solving Approach

The approach can be broken down into the following steps. Each step gets closer to answering the fundamental question in the problem statement.

1. Data Management

- Prepping data for analysis. This includes querying directly from GitHub through their API, parsing data appropriately, and addressing duplicates and outliers.

2. Exploratory Analysis

- After data management, attention turns to better understanding the data through various summary statistics, filtering, visualizations, and feature generation. All of this focuses around better understanding how price varies by the variables available.

3. Statistical Experimentation

- After understanding the data from the above steps, statistical tests can be explored around a hypothesis targeted towards a solution to predicting a baseline price through historical data.

4. Model Development

- Broadening the scope from simple statistical tests, attention turns to supervised learning algorithms and the overall modeling pipeline to fully address if accurate baseline price prediction can occur through historical data.

5. Prepping for Production

- After understanding price prediction through modeling, attention turns to the efficiency problem through package generation and productionalizing the modeling pipeline to fully answer the question in the problem statement.

Success Criteria/Metrics

The main question for this report has two components: price accuracy and overall efficiency.

The former can be evaluated using the root mean squared error (RMSE) since we are focused on baseline predictions. In this instance it is the average error when pricing a used BMW car. For example, if the RMSE was three thousand, this means that, on average, the baseline price predicted for a used BMW car is off by three thousand. While it is not clear how much human error skews pricing, it can intuitively be understood that a successful prediction will get closer to the final analyst price prediction than starting from nothing since the analyst will have to manually wrangle over less variables. For the sake of this analysis, given the average price of a used BMW vehicle in our historical data is about twenty two and a half thousand, it will be assumed that any RMSE under five thousand will be successful since anything over will force the analyst to question the prediction, abandon it, and start from scratch because it is greater than a fourth of the historical average sale price.[1]

The latter is more straightforward. Currently the analyst starts from scratch when pricing. Assuming baseline price predictions can be seamlessly generated, this will be a success.

Analysis

With the analysis design fully laid out, the assumptions and overall analytic workflow follow.

Assumptions

There are five key assumptions to keep in mind during the analysis:

1. As mentioned above, there is no human error data from BMW pricing analysts for poor price predictions. As a result, the assumption of a five thousand or greater RMSE is what will be deemed unacceptable given it is a fourth of the average used car price.

[1] This average was determined from the cleaned dataset, which is explained further in the data management portion of the analysis. Moreover, this assumption is documented in the "Assumptions" portion of the report.

2. To avoid confusion, it is assumed the price the vehicle was sold for is the price that an analyst came up with using the variables available in the historical data. Intuitively there is much more that can go into the used sale price of a vehicle than the variables at hand (more features of the car, bartering peculiarities by country, regional sale prices, etc.). Holding this assumption ensures the price seen is based on the available data.
3. The third assumption is that these sales all happened within a short time interval. Given vehicle prices can change quickly over time (a 2013 vehicle is priced differently in 2014 than 2015 all else equal), there is no need to worry about having a time stamp on when these vehicles were sold and it can be assumed the selling date would not affect the used car price.
4. Fourth, sales are representative of the BMW used car population and not biased in an unknown way to affect future vehicle pricing. This means historical data can be leveraged as a way to predict future BMW used car prices.
5. The last assumption is that these vehicle prices do not have human error. This ensures the separation of the analysis from the question attempting to be answered.

Now that the assumptions are outlined, the analysis can begin by jumping into the first portion of the analytic workflow.

Data Management

This portion can be broken down into the following:

- Analysis Packages
- Querying from GitHub API
- Addressing Duplicates
- Unique Factor Levels
- Numeric Variables: Missing Values and Distributions
- BMW Model Grouping Outliers

Analysis Packages

The following packages from CRAN will be used for this analysis.

```
# Load in necessary packages from CRAN for analysis
library(tidyverse)
library(gh)
library(RColorBrewer)
library(knitr)
library(caret)
library(fastDummies)
library(caretEnsemble)
library(gridExtra)
library(testthat)
```

Querying from GitHub API

It is important to work with the most updated data. As a result, while it would be simpler to download the data onto one's local machine and always load it from there, that will not catch any future updates made by the creators, which can be problematic. Instead, data will be pulled directly from GitHub through their

API.[2] This will ensure that the data is as updated as it can be. It will be stored in a data frame called `bmw_data`.

```
# Set GitHub personal access token, a temp variable to store data, and the target URL
GITHUB_PAT <- Sys.getenv("GITHUB_PAT")
temp = tempfile()
qurl = paste0("https://raw.githubusercontent.com/datacamp/careerhub-data/master",
              "/BMW Used Car Sales/bmw.csv")

# Download the data
gh(paste0('GET ', qurl), .destfile = temp, .overwrite = TRUE, .token = GITHUB_PAT)

# Read data into data frame
bmw_data <- read_csv(temp)
```

Summarizing `bmw_data`, it shows successful data extraction from GitHub's API along with what data was collected (nine variables and greater than ten thousand observations).

```
# Summarize raw data
summary(bmw_data)
```

```
##     model                year          price        transmission
##  Length:10781       Min.   :1996   Min.   :  1200   Length:10781
##  Class :character   1st Qu.:2016   1st Qu.: 14950   Class :character
##  Mode  :character   Median :2017   Median : 20462   Mode  :character
##                     Mean   :2017   Mean   : 22733
##                     3rd Qu.:2019   3rd Qu.: 27940
##                     Max.   :2020   Max.   :123456
##     mileage         fuelType              tax             mpg
##  Min.   :     1   Length:10781       Min.   :  0.0   Min.   :  5.5
##  1st Qu.:  5529   Class :character   1st Qu.:135.0   1st Qu.: 45.6
##  Median : 18347   Mode  :character   Median :145.0   Median : 53.3
##  Mean   : 25497                      Mean   :131.7   Mean   : 56.4
##  3rd Qu.: 38206                      3rd Qu.:145.0   3rd Qu.: 62.8
##  Max.   :214000                      Max.   :580.0   Max.   :470.8
##    engineSize
##  Min.   :0.000
##  1st Qu.:2.000
##  Median :2.000
##  Mean   :2.168
##  3rd Qu.:2.000
##  Max.   :6.600
```

Addressing Duplicates

To validate the second and fifth assumptions, one must check if there are duplicated observations across all the variables without price. If there are and the prices are not equal, this means that there was either human error or the selling price was affected by variables outside of our historical data.

[2]While out of the scope of this report, one needs to configure their GitHub personal access token to an environment variable to properly grab the data through GitHub's API.

```r
# Find duplicated observations across all variables without price and peek at them
bmw_data[duplicated(bmw_data %>% select(-price)),] %>%
  group_by(model, year, transmission, mileage, fuelType, tax, mpg, engineSize) %>%
  summarize(std_price = sd(price),
            n_vehicles = n()) %>%
  filter(is.na(std_price) == FALSE & std_price > 0) %>%
  glimpse()
```

```
## Rows: 24
## Columns: 10
## Groups: model, year, transmission, mileage, fuelType, tax, mpg [24]
## $ model        <chr> "1 Series", "2 Series", "2 Series", "2 Series", "2 Ser...
## $ year         <dbl> 2019, 2019, 2019, 2019, 2019, 2019, 2019, 2020, 2014, ...
## $ transmission <chr> "Manual", "Automatic", "Manual", "Manual", "Manual", "...
## $ mileage      <dbl> 123, 123, 10, 10, 101, 123, 123, 11, 67455, 123, 123, ...
## $ fuelType     <chr> "Diesel", "Diesel", "Petrol", "Petrol", "Petrol", "Die...
## $ tax          <dbl> 145, 145, 145, 145, 145, 145, 145, 145, 160, 145, 145,...
## $ mpg          <dbl> 53.3, 49.6, 42.8, 52.3, 52.3, 64.2, 52.3, 50.4, 50.4, ...
## $ engineSize   <dbl> 2.0, 2.0, 1.5, 1.5, 1.5, 2.0, 1.5, 1.5, 3.0, 2.0, 2.0,...
## $ std_price    <dbl> 723.39477, 1019.11857, 1799.58676, 2418.50946, 707.106...
## $ n_vehicles   <int> 3, 6, 2, 3, 2, 3, 16, 2, 2, 5, 2, 2, 3, 3, 4, 2, 2, 7,...
```

Since there are these duplicates in our data, for simplicity only one distinct combination of all non-price variables will be kept. Not only will this validate assumptions two and five, but it will also only drop about two percent of all historical data available.

```
## Rows: 10,526
## Columns: 9
## $ model        <chr> "5 Series", "6 Series", "5 Series", "1 Series", "7 Ser...
## $ year         <dbl> 2014, 2018, 2016, 2017, 2014, 2016, 2017, 2018, 2017, ...
## $ price        <dbl> 11200, 27000, 16000, 12750, 14500, 14900, 16000, 16250...
## $ transmission <chr> "Automatic", "Automatic", "Automatic", "Automatic", "A...
## $ mileage      <dbl> 67068, 14827, 62794, 26676, 39554, 35309, 38538, 10401...
## $ fuelType     <chr> "Diesel", "Petrol", "Diesel", "Diesel", "Diesel", "Die...
## $ tax          <dbl> 125, 145, 160, 145, 160, 125, 125, 145, 30, 20, 145, 2...
## $ mpg          <dbl> 57.6, 42.8, 51.4, 72.4, 50.4, 60.1, 60.1, 52.3, 62.8, ...
## $ engineSize   <dbl> 2.0, 2.0, 3.0, 1.5, 3.0, 2.0, 2.0, 1.5, 2.0, 2.0, 2.0,...
```

Unique Factor Levels

Upon inspection of the non-numeric variables (model, transmission, and fuel type), it appears that these should all have a finite number of values. For example, BMW does not make thousands of different vehicle models but rather a handful. Before parsing these variables to factors, the unique strings must be examined to ensure (1) there are not unknown missing values and (2) there are no name misalignments that would create different levels for the same item.

```
##  Unique BMW Models:
##  "5 Series"         "2 Series" "X5" "M4"       "Z4" "M2"
##  "6 Series"         "4 Series" "X4" "X2"       "X7" "M3"
##  "1 Series"         "X3"       "i3" "X6"       "M5" "M6"
##  "7 Series"         "3 Series" "X1" "8 Series" "i8" "Z3"
```

```
##  Unique BMW Transmission Types:
##  "Automatic" "Manual" "Semi-Auto"
```

```
##  Unique BMW Fuel Types:
##  "Diesel" "Petrol" "Other" "Hybrid" "Electric"
```

Both the model and transmission variables look good. All models hold after perusing BMW's website (BMW, n.d.) & (BMW, n.d.). Moreover, the three types of transmissions hold ground (Burbach, n.d.).

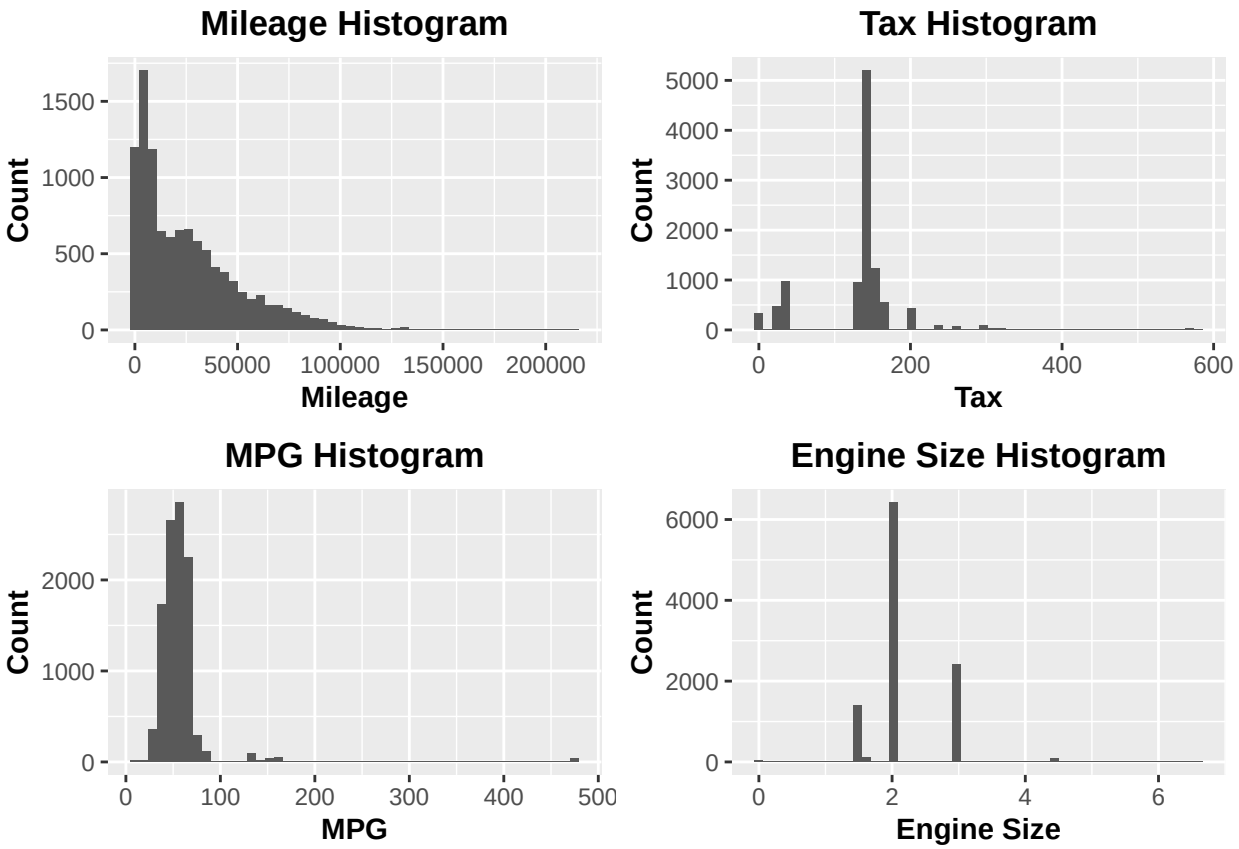
However, what does the "Other" fuel type mean? At first glance this might seem like a mistake, but after further research there are other types of fuel types that BMW has dabbled in. For example, BMW's website gives a brief overview of hydrogen fuel cells and how that differs from other fuel types (BMW, n.d.). Knowing this, the various unique fuel types appear accurate.

After verifying all of the character variables, they can be safely parsed to factors.

```
# Parse various character variables to factors
bmw_data$model <- as.factor(bmw_data$model)
bmw_data$transmission <- as.factor(bmw_data$transmission)
bmw_data$fuelType <- as.factor(bmw_data$fuelType)
bmw_data$year <- as.factor(bmw_data$year)
```

Numeric Variables: Missing Values and Distributions

From the summary of `bmw_cars` immediately after querying the data through GitHub's API, there are no missing values in the numeric variables. However, it is a good idea to look at the distribution of all the explanatory (non-price) numeric variables to ensure nothing odd is popping out that could signify a missing value coded in a unique way.



From these plots, two items stand out: miles per gallon greater than four hundred and an engine size of zero.

Upon closer inspection of the observations with greater than four hundred miles per gallon, they all belong to the BMW i3. With some more digging, it can be learned that the BMW i3 Range Extender has the specifications allowing it greater than four hundred miles per gallon, so this makes sense (ultimateSPECS, n.d.).

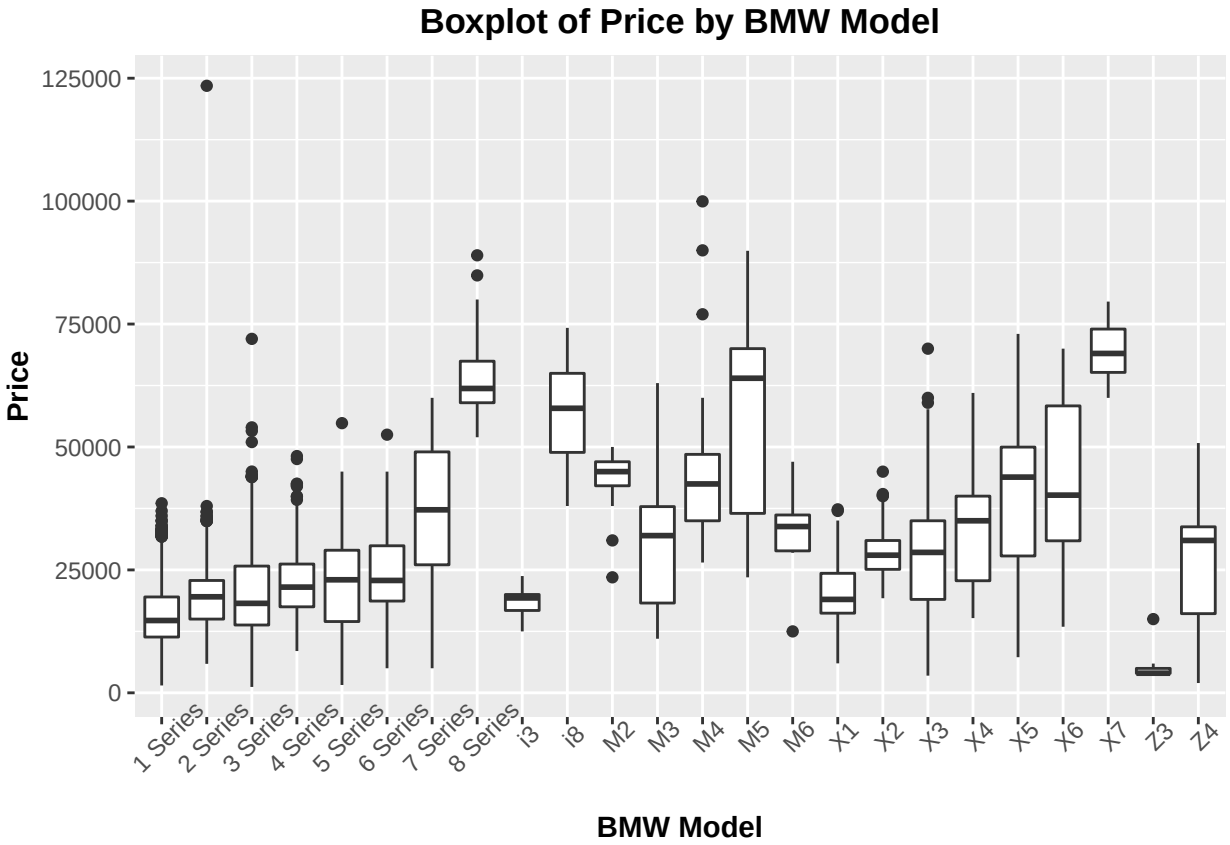
In relation to an engine size of zero, while the BMW i3 is a unique case from the large miles per gallon, the others that are seen below are not (and do not make up a large percentage of that model in the data). As a result, these will be dropped from the data (after finishing up the last step in the data management portion of the analysis).

Table 1: Percentage Engine Size Zero by BMW Model

BMW Model	NumberZero	TotalNumber	Perc
1 Series	5	1952	0.26
2 Series	2	1166	0.17
3 Series	4	2409	0.17
i3	35	43	81.40
X5	1	452	0.22

BMW Model Grouping Outliers

Lastly, before moving to the exploratory analysis, outliers must be checked when compared back to price based solely on the variables available. An intuitive view is to look at the distribution of prices grouped by model and compare further from there.



A number of outliers can be drawn from this:

- 2 Series
  - Remove the one closest to one hundred and twenty-five thousand since other prices are far lower even when considering similar mileage, fuel type, tax, miles per gallon, and engine size.
- 3 Series
  - Remove the one closest to seventy-five thousand as it is very similar to the ones for half the price.
- 5 Series
  - Remove the one closest to fifty-five thousand as there is a similar one with less miles within one and a half multiplied by the interquartile range (IQR).
- 6 Series
  - Remove the one closest to fifty-five thousand as there is a similar one with two hundred and five more miles for twenty thousand less.
- 8 Series
  - Remove ones closest to ninety and eighty-five thousand respectively since there are similar ones with prices within one and a half multiplied by the IQR.
- M4
  - Remove three points above one and a half multiplied by the IQR since there are similar ones within one and a half times the IQR.

To finish up the data management portion, the outliers noted above will be dropped along with the zero engine sized ones previously discussed to generate `bmw_data_clean`. Furthermore, below shows that the average price of a used BMW vehicle in the cleaned data is around twenty-two and a half thousand.

```
# Glimpse at final cleaned data
glimpse(bmw_data_clean)
```

```
## Rows: 10,505
## Columns: 9
## $ model        <fct> 5 Series, 6 Series, 5 Series, 1 Se...
## $ year         <fct> 2014, 2018, 2016, 2017, 2014, 2016...
## $ price        <dbl> 11200, 27000, 16000, 12750, 14500,...
## $ transmission <fct> Automatic, Automatic, Automatic, A...
## $ mileage      <dbl> 67068, 14827, 62794, 26676, 39554,...
## $ fuelType     <fct> Diesel, Petrol, Diesel, Diesel, Di...
## $ tax          <dbl> 125, 145, 160, 145, 160, 125, 125,...
## $ mpg          <dbl> 57.6, 42.8, 51.4, 72.4, 50.4, 60.1...
## $ engineSize   <dbl> 2.0, 2.0, 3.0, 1.5, 3.0, 2.0, 2.0,...
```

```
# Obtain the average price
cat("Average price of a BMW vehicle: ", mean(bmw_data_clean$price))
```

```
## Average price of a BMW vehicle:  22539.08
```

Exploratory Analysis

This section can be broken down into the following:

- Feature Generation
- Summary Statistics
  - Ungrouped and Grouped by BMW Model
- Visualizations for Data Exploration
  - Price Focused

Feature Generation

Creating more variables from the available data can draw more insight. Knowing this, the following will be created:

1. Instead of a very specific model, a new feature will be more generic.

- For example, instead of the Series 6, Series 7, Series 2, etc., all of these will be labeled as "Series" while the same goes for other ones like the X, M, Z, and i.
- This factor variable will be called `model_generic`.

2. If a vehicle has a semi-automatic or automatic transmission, it will get a one. If not, it will get a zero.

- This numeric variable will be called `is_auto_semi`.

3. If a vehicle has a semi-automatic or manual transmission, it will get a one. If not, it will get a zero.

- This numeric variable will be called `is_man_semi`.

4. If a vehicle's mileage is within the IQR as determined from the available data, it will be classified as "Medium" while "Low" will be below the IQR and "High" will be above.

- This factor variable will be called `mileage_cat`.

5. If a vehicle's miles per gallon is within the IQR as determined from the available data, it will be classified as "Medium" while "Low" will be below the IQR and "High" will be above.

- This factor variable will be called `mpg_cat`.

As seen below, the dataset with the new features will be `bmw_data_clean_full`. The five new features can be seen at the bottom.

```
# Peek at full data
glimpse(bmw_data_clean_full)
```

```
## Rows: 10,505
## Columns: 14
## $ model         <fct> 5 Series, 6 Series, 5 Series, 1 S...
## $ year          <fct> 2014, 2018, 2016, 2017, 2014, 201...
## $ price         <dbl> 11200, 27000, 16000, 12750, 14500...
## $ transmission  <fct> Automatic, Automatic, Automatic, ...
## $ mileage       <dbl> 67068, 14827, 62794, 26676, 39554...
## $ fuelType      <fct> Diesel, Petrol, Diesel, Diesel, D...
## $ tax           <dbl> 125, 145, 160, 145, 160, 125, 125...
## $ mpg           <dbl> 57.6, 42.8, 51.4, 72.4, 50.4, 60....
## $ engineSize    <dbl> 2.0, 2.0, 3.0, 1.5, 3.0, 2.0, 2.0...
## $ model_generic <fct> Series, Series, Series, Series, S...
## $ is_auto_semi  <dbl> 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, ...
## $ is_man_semi   <dbl> 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, ...
## $ mileage_cat   <fct> High, Medium, High, Medium, High,...
## $ mpg_cat       <fct> Medium, Low, Medium, High, Medium...
```

Summary Statistics

First, the mean, median, and standard deviation of the numeric variables will be examined to see if anything interesting pops out.

Table 2: Numeric Variable Summary Statistics

	price	mileage	tax	mpg	engineSize	is_auto_semi	is_man_semi
mean	22539.08	25992.95	131.43	56.57	2.17	0.77	0.67
median	19999.00	19164.00	145.00	53.30	2.00	1.00	1.00
standard_deviation	11261.17	25155.77	61.77	31.67	0.55	0.42	0.47

One interesting note is that there are more automatic or semi-automatic vehicles rather than manual or semi-automatic ones. Plus, the average miles per gallon is high at 56. However, there are legitimate i3 vehicles that push this up a bit, so the median of 53 miles per gallon is more in touch with the rest of the cars.

Calculating these summary statistics by BMW model allows a deeper analysis. Given the large output this will produce and that the main question is focused around price, the mean price, median price, and price standard deviation grouped by model specifically for the Series 1-6 will be shown.

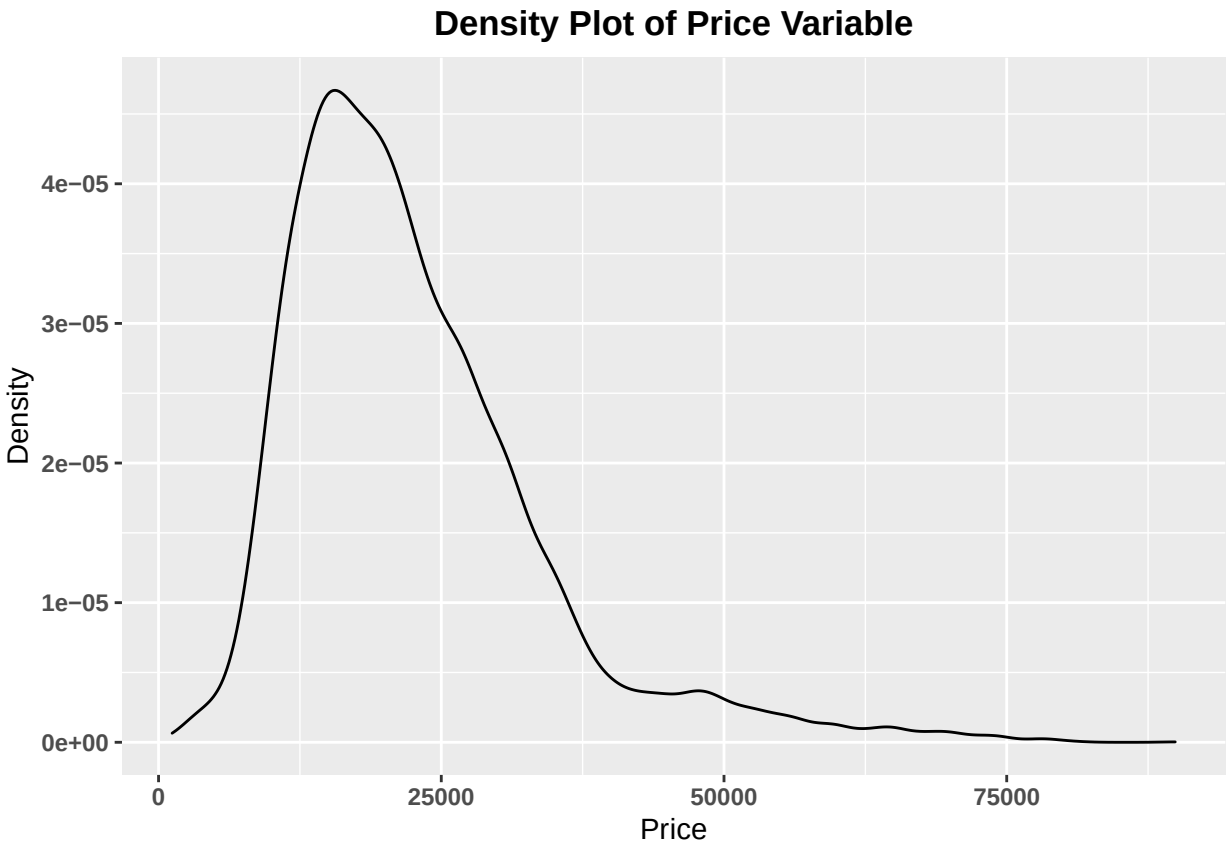
Table 3: Summary Statistics by BMW Model

model	price_mean	price_median	price_standard_deviation
1 Series	15769.42	14750	5931.197
2 Series	19289.97	19543	5357.910
3 Series	19778.72	18231	8280.839
4 Series	22245.90	21499	6264.809
5 Series	22394.93	22985	8628.154
6 Series	24091.33	22798	8181.928

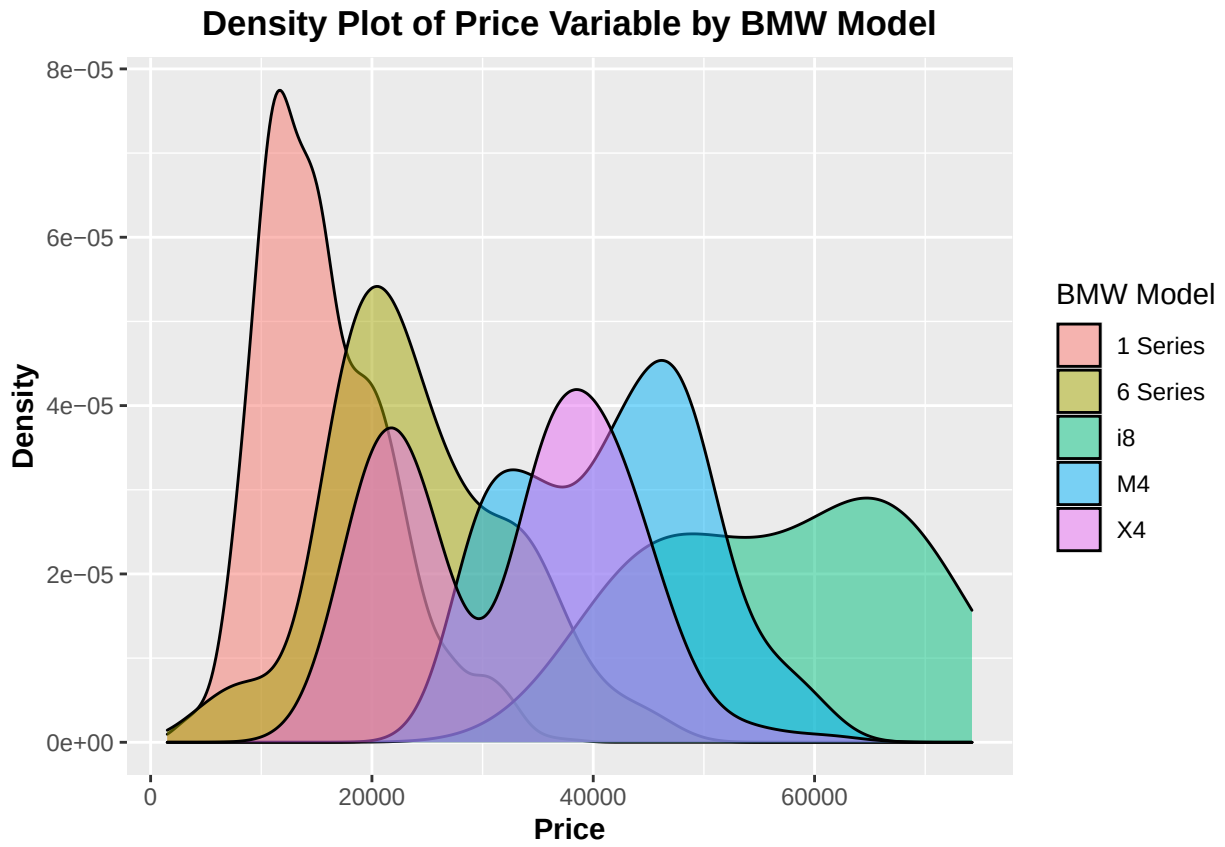
While not extremely interesting, it is reassuring to see the median price rising from 1 Series to 6 Series since this is continually stepping up in vehicle quality.

Visualizations for Data Exploration

Given the most recently explored price summary statistics by model, the distribution of the price variable as a whole is a logical next step.

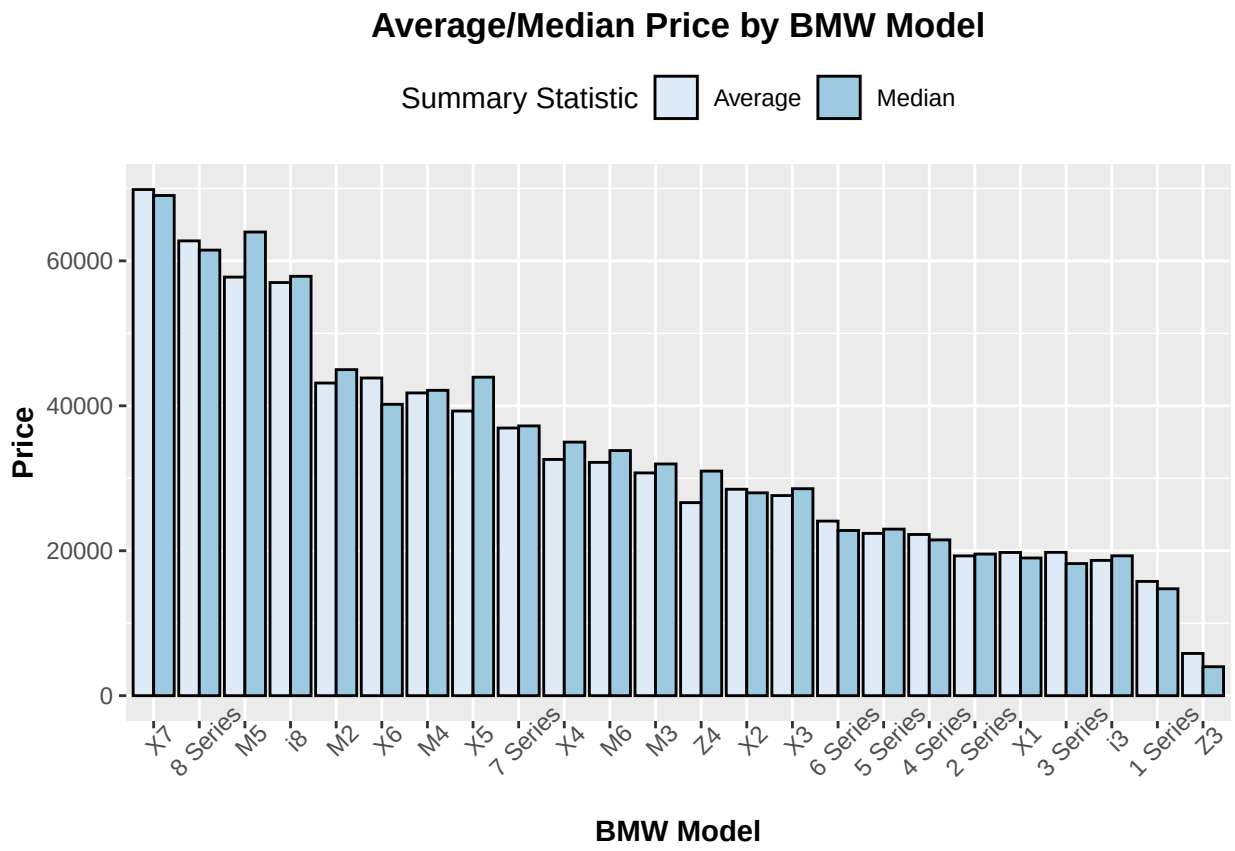


The distribution is rightly skewed, which makes sense given there are many lower priced used vehicles being sold in the world rather than higher priced ones. However, how do the distributions look when breaking it out by model? Since there are many different models, focus will be placed on the 1 Series, 6 Series, M4, X4, and i8.



While a bit more difficult to read given the many different vehicle models, the main takeaway is that simply looking at the overall distribution of price is very different than if you break out the price distributions by model. For example, the 1 Series peaks around twelve thousand while the 6 Series is around twenty thousand.

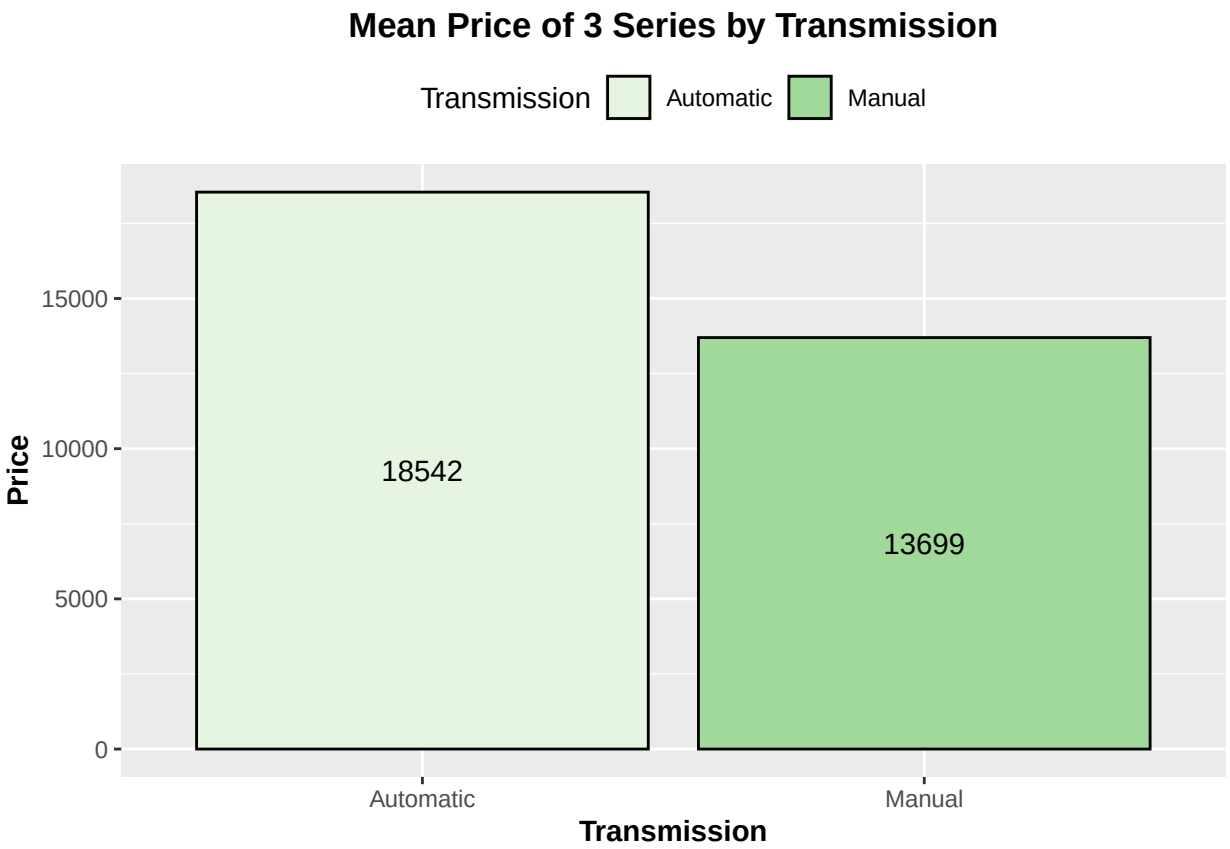
To wrap up the exploratory analysis, a plot of the average and median price of used vehicles by model is shown. From the below, one can note which vehicles are more luxurious (X7) versus more affordable (1 Series).



Statistical Experimentation

Having a good grasp of the data through the data management and exploratory analysis portions, it is time to focus more on the first part of the problem at hand: can data science be used to establish a more accurate price baseline given the variables available through historical data?

Answering this question first involves diving into a statistical experiment focused around the Series 3. More specifically, from the below plot the mean price is different between an automatic versus a manual transmission vehicle. But how different are they? Can one statistically test and determine if the mean price of a 3-Series automatic is different than 3-Series manual?



Using bootstrapping, this can be tested. The null hypothesis is that the difference in means between the two is zero (in other words, there is no difference), and the null hypothesis will only be rejected if one is 95% confident that the difference in means is not zero. The process is as follows:

1. Separate 3 Series vehicles into two data frames: one with automatic vehicles and one with manual vehicles
2. Take a sample of a hundred from each group with replacement
3. Calculate the difference between the mean price of automatic vehicles versus manual ones in the samples
4. Store the result in another data frame
5. Repeat steps two through four ten thousand times
6. Analyze results

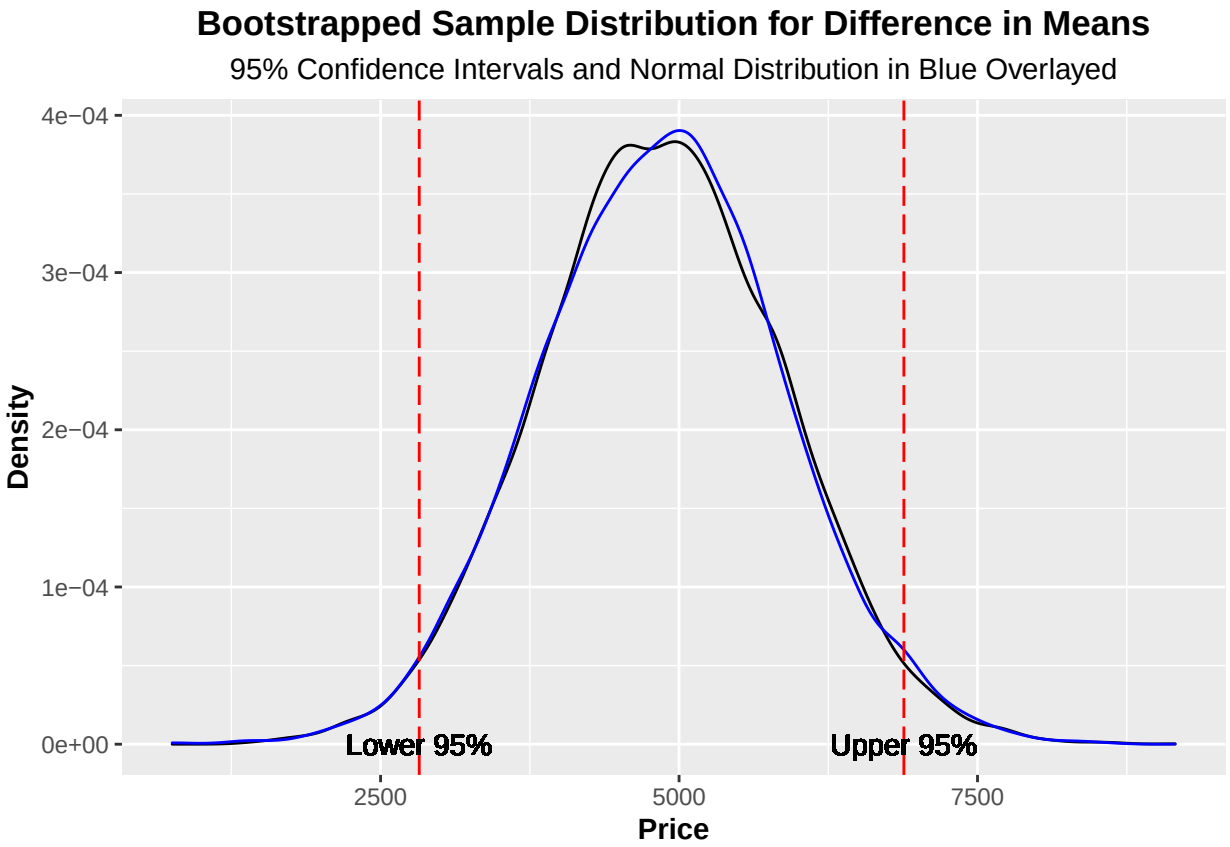
By doing this, the variation for the difference in means can be observed to see how much of the time this difference is greater than zero.

The test can be written as the code below shows.

```
# Set seed for reproducibility
set.seed(777)

# Create data frame to store results
bootstrap_samples_df <- data.frame(mean_auto = NULL,
                                   mean_man = NULL,
                                   diff = NULL)

# Data frame of automatic 3 Series
```

```
auto_bmw <- bmw_data_clean_full %>%
  filter(transmission == "Automatic" & model == "3 Series")

# Data frame of manual 3 Series
man_bmw <- bmw_data_clean_full %>%
  filter(transmission == "Manual" & model == "3 Series")

# Perform loop 10000 times
for(i in 1:10000) {

# Sample 100 of automatic and manaual 3 Seriees with replacement
sample_auto <- sample_n(auto_bmw, size = 100, replace = TRUE)
sample_man <- sample_n(man_bmw, size = 100, replace = TRUE)

# Store results in temporary data frame: mean of automatic, mean of manual, and difference
bootstrap_samples_df_temp <- data.frame(mean_auto = mean(sample_auto$price),
                                        mean_man = mean(sample_man$price),
                                        diff = mean(sample_auto$price) - mean(sample_man$price))

# Append latest iteration to final results data frame
bootstrap_samples_df <- bind_rows(bootstrap_samples_df, bootstrap_samples_df_temp)
}
```

Visualizing the results below, it shows the difference never crosses zero in the ten thousand times we sampled. Moreover, a normal distribution is overlayed to show that our bootstrapped distribution is approximately normal. If the distribution is not approximately normal, results cannot be trusted (Minitab, n.d.). Lastly, in between the vertical lines indicates the 95% confidence interval. Put another way, one is 95% confident that the difference in means falls in between the lines. Therefore, the null hypothesis can be rejected and one can assert that there is a statistical difference in the mean price between Series 3 vehicles with automatic transmissions versus manual ones. Put more specifically towards the first part of the ultimate question for this analysis, knowing whether or not a Series 3 BMW has an automatic or manual transmission (which the historical data has) helps determine an accurate baseline price for a BMW pricing analyst.



Model Development

While doing a simple statistical test is a good start to answering the first part of the overall question, more sophisticated modeling techniques can be used to fully assess whether a solid baseline price can be created for a BMW pricing analyst based on the historical data available. That is what this section focuses on, and it can be broken down into the following components:

- Data Preprocessing
- Creation of Training and Testing Data
- Initial Algorithm Fit/Validation/Selection
- Hyperparameter Tuning & Final Model Generation/Validation

Data Preprocessing

Before jumping straight into modeling, some data preprocessing needs to occur.[3]. Dummy variables for the factors will be created. After doing this, the factor variables are no longer there. Instead, they are replaced by a dummy variable for each unique level contained in the factor variables. This new dataset will be called `bmw_data_clean_full_model`, and these dummy variables for the `fuelType` variable can be seen below.

[3]Note that while this section only covers dummy variable creation, the modeling pipeline also centers and scales data since it can be beneficial for some algorithms in this analysis. However, since this is done automatically in the `caret` package (what will be used for model generation), it is omitted here for simplicity.

```r
# View fuel type portion of new data with dummy variables
glimpse(bmw_data_clean_full_model[, c("fuelType_Diesel", "fuelType_Petrol",
                                      "fuelType_Other", "fuelType_Electric",
                                      "fuelType_Hybrid")])
```

```
## Rows: 10,505
## Columns: 5
## $ fuelType_Diesel   <int> 1, 0, 1, 1, 1, 1, 1, 0, 1, 1,...
## $ fuelType_Petrol   <int> 0, 1, 0, 0, 0, 0, 0, 1, 0, 0,...
## $ fuelType_Other    <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...
## $ fuelType_Electric <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...
## $ fuelType_Hybrid   <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...
```

Creation of Training and Testing Data

Next, the data will be split into training and testing sets. To keep in line with standard naming conventions, the independent variables will be in `X` and the dependent variable (price) will be in `y`. The data will also have the indices shuffled around to ensure the ordering of the data does not bias the modeling results. Lastly, the data will be split into eighty percent training and twenty percent testing, resulting in the objects `X_train`, `y_train`, `X_test`, and `y_test`.

```r
# Generate X: independent variables
X <- bmw_data_clean_full_model %>%
  select(-price)

# Generate y: dependent variable (price)
y <- bmw_data_clean_full_model$price

# Set seed for reproducibility
set.seed(777)

# Shuffle indices and pull out random sample for training (80%)
train_index_shuffled <- sample(
  1:nrow(bmw_data_clean_full),
    size = round(nrow(bmw_data_clean_full_model) * 0.8,
            0))

# Train-test split - 80% training, 20% testing
X_train <- X[train_index_shuffled,]
y_train <- y[train_index_shuffled]
X_test <- X[-train_index_shuffled,]
y_test <- y[-train_index_shuffled]
```

Initial Algorithm Fit/Validation/Selection

Since no models have been trained up to this point, a wide variety of supervised learning algorithms will be tested to determine the next step. The selections include:

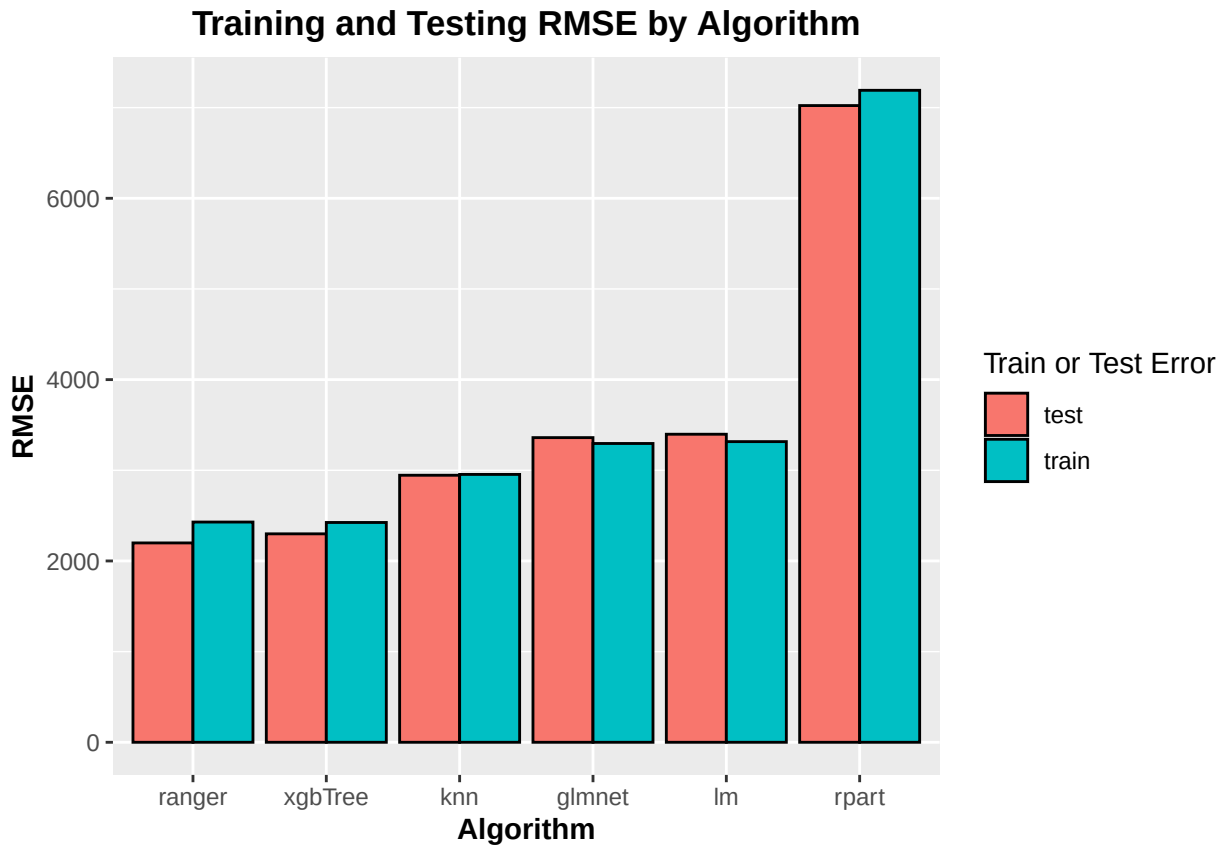
- Simple Linear Model (lm)
- Simple Decision Tree (rpart)

- Random Forest (ranger)
- Extreme Gradient Boosting (xgbTree)
- Elastic Net Regression (glmnet)
- K-Nearest Neighbors (knn)

Also, to get more robust results, five-fold cross validation and centering and scaling the data prior to fitting will be used along with a randomized assortment of initial hyperparameters that will be tuned later after determining the algorithm to proceed with. This can all be achieved in a pipeline with the code below leveraging the `caret` and `caretEnsemble` packages.

```
# Set training control argument to use five-fold cross validation
train_control <- trainControl(method = "cv",
                              number = 5)

# Set seed for reproducibility
set.seed(777)

# Train specified algorithms - explicitly indicate centering and scaling before fit
# Capture output so not printed to report
captured_output <- capture.output(model_list <- caretList(X_train,
                          y_train,
                          trControl = train_control,
                          methodList = c("lm", "rpart", "ranger",
                                         "xgbTree", "glmnet", "knn"),
                          tuneList = NULL,
                          preProcess = c("center", "scale"),
                          continue_on_fail = FALSE))
```

Below are the results by each algorithm. The random forest algorithm has the lowest RMSE on the test set and nearly the same as extreme gradient boosting on the training data (2429 for random forest versus 2424 for extreme gradient boosting). Given these results, the random forest algorithm will be used going forward. A good spot has already been reached since the RMSE for the random forest is well below five thousand, but can the RMSE be better by tuning the hyperparameters for the random forest?



Hyperparameter Tuning & Final Model Generation/Validation

The optimal parameters that the initial search found for the random forest was mtry at 74, splitrule at extratrees, and min.node.size at 5 to generate a training RMSE of 2429 and a test RMSE of 2199. A grid search will be performed by slightly varying each hyperparameter and testing out the different combinations to see if better results can be achieved.[4]

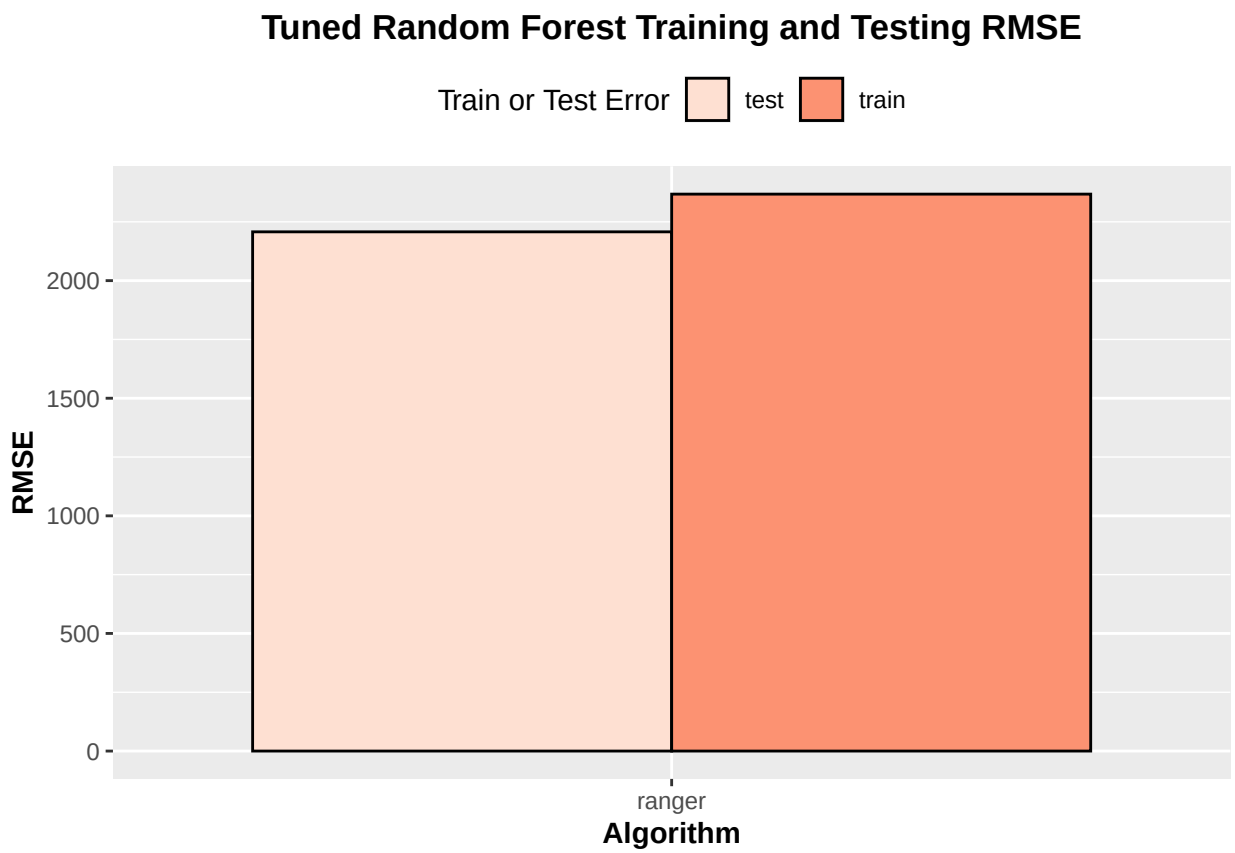
```
# Create grid that will be used when finding optimal hyperparameters
tune_grid <- expand.grid(
  mtry = c(73, 74),
  splitrule = c("extratrees"),
  min.node.size = c(2, 4, 5)
)

# Set seed for reproducibility
set.seed(777)

# Grid search to find optimal hyperparameters for random forest
# Capture output so not printed to report
captured_output <- capture.output(rf_model <- train(X_train,
                  y_train,
                  method = "ranger",
```

[4]Another hyperparameter with random forests is the number of trees. However, the `caret` package does not allow one to modify it through its `train` function. For the sake of this analysis, it will be left at the default of five hundred trees used in the `ranger` package (the package `caret` calls behind the scenes when training the random forest).

```
                  trControl = train_control,
                  tuneGrid = tune_grid,
                  preProcess = c("center", "scale")))

# Print model results
rf_model
```

```
## Random Forest
##
## 8404 samples
##   74 predictor
##
## Pre-processing: centered (74), scaled (74)
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 6723, 6722, 6724, 6723, 6724
## Resampling results across tuning parameters:
##
##   mtry  min.node.size  RMSE      Rsquared   MAE
##   73    2              2384.506  0.9554130  1559.286
##   73    4              2371.695  0.9559215  1552.740
##   73    5              2367.838  0.9560821  1552.562
##   74    2              2389.341  0.9552295  1561.458
##   74    4              2368.378  0.9560348  1554.139
##   74    5              2370.313  0.9559794  1551.863
##
## Tuning parameter 'splitrule' was held constant at a
##  value of extratrees
## RMSE was used to select the optimal model using
##  the smallest value.
## The final values used for the model were mtry =
##  73, splitrule = extratrees and min.node.size = 5.
```

From the results of training above, the optimal hyperparameters are mtry at 73, splitrule at extratrees, and min.node.size at 5. However, even though the training RMSE was slightly lower at 2368, the testing error needs to be checked as well. The below plot shows us the results.



While the test RMSE was slightly greater than the initial search, it was only by eight (2199 versus 2207), or less than a half of a percent. In other words, it essentially did not change. Moreover, having the training error closer to the testing error after tuning the hyperparameters helps assure one that there is no drastic overfitting or underfitting. Overall, hyperparameter tuning was a success and a final model was found that has the tuned hyperparameters, which will be called `rf_model_final`. More importantly, the model was successful generating accurate baseline prices for a BMW pricing analyst based on the historical data available with an RMSE of less than five thousand.

Prepping for Production

Up to this point, the analysis has focused on addressing the accuracy portion of the main question, but there is still the efficiency portion to answer. Can data science be used to speed up the pricing process?

Before beginning the analysis, specific packages from CRAN were loaded. These allowed one to leverage important functions and go from querying data to the final model for price prediction in an efficient manner. Along these lines, a package can be built with functions that will allow for efficient BMW used car baseline price predictions should those cars have variables which match the historical data variables used in this report. Moreover, packages can include documentation, unit tests, and can easily be distributed across an organization, which is perfect for a production environment.

While this report will not cover all the specifics of making a package, it will include the noteworthy points as they pertain to this analysis. These include:

- Necessary Data
- Necessary Functions
- Unit Tests

- Deployment & Version Control
- Intended Use

Necessary Data

When generating the final model, the IQR of the training data for mileage and miles per gallon were necessary to create features. To create these same features in the future with the package, this information needs to be available in the package for use along with the final model object. The below code chunk saves these objects to the appropriate place (named `historical_mileage_25_75_quantiles`, `historical_mpg_25_75_quantiles`, and `rf_model_final`). Since this is a walkthrough, the code chunks following are a demonstration of what to do and are not evaluated until the final package is loaded for unit testing and intended use.

```
# Generate historical IQRs needed for feature generation
historical_mileage_25_75_quantiles <- quantile(bmw_data_clean$mileage,
                                               c(0.25, 0.75),
                                               names = FALSE)
historical_mpg_25_75_quantiles <- quantile(bmw_data_clean$mpg,
                                           c(0.25, 0.75),
                                           names = FALSE)

# Save IQRs needed for feature generation
save(historical_mileage_25_75_quantiles,
     file = "/path/to/package/data/historical_mileage_25_75_quantiles.rda")
save(historical_mpg_25_75_quantiles,
     file = "/path/to/package/data/historical_mpg_25_75_quantiles.rda")

# Save final model for price prediction on new data
save(rf_model_final, file = "/path/to/package/data/rf_model_final.rda")
```

Necessary Functions

Now that the necessary pieces from the analysis are in place to create a pricing prediction pipeline, building a couple of functions to make it efficient comes next.

The first function will be called `auto_clean`, and this takes the raw data after querying the GitHub API and transforms it into the appropriate format intended for final model prediction.

```
auto_clean <- function(new_data_df) {

  # Load in model
  data("rf_model_final")

  # Load in historical quantiles
  data("historical_mileage_25_75_quantiles")
  data("historical_mpg_25_75_quantiles")

  # Convert variables to factor
  new_data_df$model <- as.factor(new_data_df$model)
  new_data_df$transmission <- as.factor(new_data_df$transmission)
  new_data_df$fuelType <- as.factor(new_data_df$fuelType)
  new_data_df$year <- as.factor(new_data_df$year)
```

```r
# Add categorical variables
new_data_full <- new_data_df %>%
  mutate(model_generic = as.factor(case_when(
    str_detect(model, "Series") == TRUE ~ "Series",
    str_detect(model, "X") == TRUE ~ "X",
    str_detect(model, "i") == TRUE ~ "i",
    str_detect(model, "M") == TRUE ~ "M",
    str_detect(model, "Z") == TRUE ~ "Z")),
    is_auto_semi = ifelse(transmission == "Automatic" |
                            transmission == "Semi-Auto", 1, 0),
    is_man_semi = ifelse(transmission == "Manual" |
                           transmission == "Semi-Auto", 1, 0),
    mileage_cat = as.factor(case_when(
      mileage < historical_mileage_25_75_quantiles[1] ~ "Low",
      mileage > historical_mileage_25_75_quantiles[2] ~ "High",
      mileage >= historical_mileage_25_75_quantiles[1] &
        mileage <= historical_mileage_25_75_quantiles[2] ~ "Medium"
    )),
    mpg_cat = as.factor(case_when(
      mpg < historical_mpg_25_75_quantiles[1] ~ "Low",
      mpg > historical_mpg_25_75_quantiles[2] ~ "High",
      mpg >= historical_mpg_25_75_quantiles[1] &
        mpg <= historical_mpg_25_75_quantiles[2] ~ "Medium"
    )))

# Get dummy variables
new_data_full_model <- dummy_cols(new_data_full, remove_first_dummy = FALSE) %>%
  select_if(is.numeric)

# Find difference in columns -
# Remove ones that are present in test data only
names_rm <- setdiff(names(new_data_full_model), names(rf_model_final$trainingData))

for (i in names_rm) {

  new_data_full_model[, i] <- NULL

}

# Add dummies with 0s that are not present in test data only
names_add <- setdiff(names(rf_model_final$trainingData), names(new_data_full_model))

for (i in names_add) {

  new_data_full_model[, i] <- 0

}

# Drop .outcome
new_data_full_model$.outcome <- NULL

return(new_data_full_model)
```

```
}
```

The other function is `auto_predict`, which generates baseline pricing predictions for the new data.

```
auto_predict <- function(clean_new_data_df) {

  # Load in model
  data("rf_model_final")

  predictions <- tryCatch(
    {
      # Generate predictions
      predict.train(rf_model_final, newdata = clean_new_data_df)
    },
    error = function(cond) {
      message("Pricing failed. Did you pass the data through auto_clean first?")
      message("Returning NA values as a result of error")
      message("Original error message below:")
      message(cond)
      # Return NA if error
      return(NA)
    }
  )
  return(predictions)
}
```

Lastly, one can save the scripts containing the functions into the "R" folder of the package.[5]

Unit Tests

The functions defined above must behave appropriately. This is where unit tests come in. One can embed them in a package to not only make sure the functions are working properly today but also ensure they are still working if future changes to the functions are made.

Four unit tests are below. Two are for `auto_clean` and two are for `auto_predict`. The next section will check to make sure they were successful.

- `auto_clean`
  - Returns a data frame
  - Returns the same number of rows that were in the initial data frame
- `auto_predict`
  - Returns a numeric vector
  - Length of the vector returned matches the number of rows in the initial data frame

`auto_clean` tests:

[5] Each function must be in its own separate R script. Moreover, these scripts should also contain the function documentation above it, which can be seen in the final package.

```
# Unit test to ensure it returns a data frame
test_that("Returns a data frame", {
  expect_equal(class(auto_clean(data.frame(model = "5 Series",
                                           year = 2018,
                                           transmission = "Manual",
                                           mileage = 50000,
                                           fuelType = "Diesel",
                                           tax = 100,
                                           mpg = 20,
                                           engineSize = 2))),
               "data.frame")
})

# Unit test to ensure it returns the same number of rows that were in initial data frame
test_that("Number rows in returned data frame matches number rows in initial", {
  expect_equal(nrow(auto_clean(data.frame(model = "5 Series",
                                          year = 2018,
                                          transmission = "Manual",
                                          mileage = 50000,
                                          fuelType = "Diesel",
                                          tax = 100,
                                          mpg = 20,
                                          engineSize = 2))),
               nrow(data.frame(model = "5 Series",
                               year = 2018,
                               transmission = "Manual",
                               mileage = 50000,
                               fuelType = "Diesel",
                               tax = 100,
                               mpg = 20,
                               engineSize = 2)))
})
```

`auto_predict` tests:

```
# Unit test to ensure it returns a numeric vector
test_that("Return a numeric vector", {
  expect_equal(class(auto_predict(auto_clean(data.frame(model = "5 Series",
                                                        year = 2018,
                                                        transmission = "Manual",
                                                        mileage = 50000,
                                                        fuelType = "Diesel",
                                                        tax = 100,
                                                        mpg = 20,
                                                        engineSize = 2)))),
               "numeric")
})

# Unit test to ensure vector length matches number of rows in initial data frame
test_that("Length of vector matches number of rows in initial data frame", {
  expect_equal(length(auto_predict(auto_clean(data.frame(model = "5 Series",
                                                         year = 2018,
                                                         transmission = "Manual",
                                                         mileage = 50000,
```

```
                                                    fuelType = "Diesel",
                                                    tax = 100,
                                                    mpg = 20,
                                                    engineSize = 2))),
              nrow(data.frame(model = "5 Series",
                              year = 2018,
                              transmission = "Manual",
                              mileage = 50000,
                              fuelType = "Diesel",
                              tax = 100,
                              mpg = 20,
                              engineSize = 2)))
})
```

Deployment & Version Control

Now that all the necessary items for the package are in, it can be deployed via a private GitHub repository. That way only those that have access to it can download and use it. Moreover, GitHub leverages Git, a more advanced version control system than a standard business way of having multiple versions of an item saved in the same directory (like multiple Microsoft Excel files). This package will be named `datacampBMWProd`.

When the package is deployed on GitHub and appropriate permissions have been granted, the package can easily be downloaded with the following command in the code chunk below from the `devtools` package:[6]

```
# Install package from GitHub
devtools::install_github("njbultman/datacampBMWProd")
```

Intended Use

As a wrap-up to this portion of the analysis, the unit tests will be tested with the `testthat` package and the `auto_clean` and `auto_predict` functions will be called to demonstrate that an efficient pipeline can be created to answer the latter half of the main question.

```
# Load package
library(datacampBMWProd)

# Run unit tests
testthat::test_package("datacampBMWProd")
```

```
## [ FAIL 0 | WARN 0 | SKIP 0 | PASS 4 ]
```

```
# Create test data frame
raw_df <- data.frame(model = "5 Series",
                     year = 2018,
                     transmission = "Manual",
                     mileage = 50000,
                     fuelType = "Diesel",
                     tax = 100, mpg = 20,
                     engineSize = 2)
```

[6]For the purposes of privacy, this private GitHub repository is only accessible by the creator (njbultman) currently. Please email njbultman74@gmail.com if a specific individual would like access to review with the report.

```
# Run auto_clean function
cleaned_df <- datacampBMWProd::auto_clean(raw_df)

# Run auto_predict function
datacampBMWProd::auto_predict(cleaned_df)
```

```
## [1] 13115.42
```

As seen in the output, a price prediction for the test data was successfully generated in an efficient way with the `datacampBMWProd` package, and the unit tests were successful as well.

Results & Recommendations

Overall this analysis attempted to answer the following: as a BMW pricing analyst, can data science be used to not only establish a more accurate baseline price given the variables available through historical data but also speed up the pricing process and better ensure company profitability?

After managing, exploring, and performing a statistical test on the data, a model that produced an RMSE below our five thousand threshold that was defined for success was successfully generated. Moreover, a package was constructed to efficiently create a baseline price for incoming BMW used car data instead of having the analyst start from scratch. Ultimately, this analysis showed that company profitability can be strengthened by answering the main question of this report to address the initial problem statement.

While this analysis is complete, there are five next step recommendations for diving deeper.

1. Fully evaluate what RMSE is acceptable. While this analysis chose five thousand, is there a more acceptable number?
2. Understand the structure of the data that will be flowing into the pipeline for the future.

- Will the same columns be coming through, or will there be more/less columns? The answer could potentially impact whether the model needs to be refreshed.
- Will the factor variables have new unique selections? If so, the model will need to be refreshed to accurately reflect this.

3. Understand the volume of data that will be flowing into the pipeline for the future.

- If it will be a lot of data, leveraging parallel computing could provide more efficiency gains.

4. Further explore other algorithms and their hyperparameter combinations to see if a more accurate model can be achieved.
5. Continue building out the `datacampBMWProd` package for better integration with other processes and the overall production environment.

Reproducibility

As a final note before wrapping up the report, one might have noticed in the code blocks that there was often a line with `set.seed(777)`. Before running code that involved randomness (sampling, splitting data for five-fold cross validation, index shuffling, etc.), having this piece of code ensures the results are reproducible. In other words, running the same code block many times will always return the same results.

References

BMW. n.d. “BMW: All Models.” https://www.bmw.com/en-au/models.html.

———. n.d. “Hydrogen Fuel Cell Cars: Everything You Need to Know.” https://www.bmw.com/en/innovation/how-hydrogen-fuel-cell-cars-work.html.

———. n.d. “The Bmw I8: Pioneer, Icon, and Classic of the Future.” https://www.bmw.com/en/innovation/bmw-i8-as-future-classic.html.

Burbach, David. n.d. “4 Types of Car Transmissions (and How They Work).” https://www.motorbiscuit.com/4-types-of-car-transmissions-and-how-they-work/.

Minitab. n.d. “Interpret the Key Results for Bootstrapping for 2-Sample Means.” https://support.minitab.com/en-us/minitab-express/1/help-and-how-to/basic-statistics/inference/how-to/resampling/bootstrapping-for-2-sample-means/interpret-the-results/key-results/#:~:text=The%20difference%20in%20means%20of,the%20difference%20in%20population%20means.&text=If%20a%20statistic%20has%20no,sampling%20distribution%20of%20the%20statistic.

ultimateSPECS. n.d. “BMW I3 94Ah Range Extender Specs.” https://www.ultimatespecs.com/car-specs/BMW/114719/BMW-i3-94Ah-Range-Extender.html#:~:text=With%20a%20fuel%20consumption%20of,engine%2C%20Hybrid%20%2F%20Petrol%20motor.

Zippia. n.d. “Pricing Analyst Overview.” https://www.zippia.com/pricing-analyst-jobs/.