STA 141C Final Project

Group 13

Analysis on New and Used Cars (1990 - 2023)

Group Members:

Tracy Yu
Kayla Pham
Constanza Widel
Rachel How

March 23rd, 2023

Cover Letter

(a) **Main Findings**

Our goal in this project is to determine variables that can influence or be significant in predicting the final price of a car. After cleaning and removing outliers from the data, we ran Multiple Linear Regression on four different models to predict the price of a car. Using the R-squared and Adjusted R-squared values, we determined that the model with the predictor variables Mileage, Brand, and Year had the best fit.

We then used this model to explore LU decomposition as an alternative and computationally efficient way to obtain the regression coefficients.

Finally, we used a k-NN algorithm to predict car price and determined that car brand is significant to predicting a car's price.

(b) **Methods**

The main technique from this course that we applied to our project was LU decomposition, which is a method of solving a linear equation such as $Ax = b$. In big data programming, LU decomposition has the advantage over the "solve()" function in terms of flop counts and computation time. We used the Python SciPy "lu" function to decompose a given square matrix into P, L, and U. This step requires a significant amount of multiplications and additions since our dataset is quite large and takes $O(n^3)$ flops.

To keep flop count as low as possible, we utilized forward and backward substitution to minimize flop counts. In doing so, the overall flop count for the LU decomposition algorithm to solve for the regression coefficients was $O(n^2)$ and our computational time was also quite low.

Additional techniques that were utilized were Multiple Linear Regression and k-NN algorithms, as mentioned above.

Data Preparation

(a) **Introduction**

The main question we seek to answer with this project is: what are the factor(s) that influence the price of a car?

Undoubtedly, cars influence our way of life on a daily basis. For many Americans, having a vehicle has become a norm in order to perform many of the tasks at hand. Within the last few years, car prices have been going through major ebbs and flows. This makes analyses like this important for an average consumer because they should understand whether or not they are getting a justifiable deal on their car.

The *Used and New Cars* data set was obtained from Kaggle (https://www.kaggle.com/datasets/georgejnr/used-and-new-cars-datasets). The original data set contained 115,762 data points and contained six different variables:

a. **Model**: This column provides information on the brand of the car and specific type, as well as when the car was manufactured.

b. **Status**: This information provides information on whether or not the car was new or used. If the car was used, then it is specified whether or not it was Certified Used (meaning the dealership purchased the car from a third-party and then resold at the dealership after it went through their certification process) or if it was purely used and sold independently.

c. **Mileage**: This column provides information on the mileage of the vehicle (in miles).

d. **Price**: This column provides information on how much the car is being offered for sale.

e. **MSRP**: This column provides information on the MSRP (i.e. Manufacturer Suggested Retail Price) of the car. This is the price that the manufacturer itself believes the car should be sold for.

f. **Year**: This column provides the year that the car was manufactured.

(b) **Data Cleaning and Methodologies**

The first step in our data cleaning process was to examine the data where there were N/As or "Not Specified" was present. Although there were no instances of missing data within our data set, there were several instances where there were pieces of data that were not specified one way or another.

a. **Mileage Column**:
Within the **Mileage** column, there were several entries were "Not available" was placed. Investigating this, we discovered this was only applicable for new cars, so we replaced "Not available" with 0, as new vehicles will have 0 miles on the odometer.

The column was then changed to an integer data type so that future analyses can be run using this variable.

b. **MSRP Column**:
Next, we took a look at the **MSRP** column. Because there were a mix of words and prices within this column (ex: "MSRP $49,445" vs. "$49,445"), we had to clean this column such that there were only numbers available. Some MSRP values were "Not specified." To address this, we made the decision to use the corresponding **Price** value as the MSRP value. The reason for this is because we found that 62% of the **MSRP** values contained this value.

For the purposes of this project, we made the decision to assign the MSRP and Price values to be the same because it would take too much time to find the correct MSRP values for over 70,000 values. In order to create a more robust analysis, it would be advisable to have this data available so that the results of the analysis can be more accurate.

There were also some entries that were listed as "Not Priced." This made up less than 1% of our data set, so we removed those entirely.

This column did not always have a straightforward number for the MSRP value. Instead, most of these entries contained a price, followed by "price drop." This indicated that there was a discounted value of the specified amount from the **Price** value.

For example, an entry of "$600 price drop" within this column meant that the MSRP value for that car was $600 off the Price value listed in the **Price** column. This is what was present for the majority of entries within the **MSRP** column.

In order to address this, we subsetted the data set into two different data sets: one data set that contained "price drop" within the **MSRP** column and another that contained every other entry. From there, we dropped the text "price drop" from the subsetted data set and subtracted the amount within the **MSRP** column from the value in the corresponding **Price** column. Then, we assigned that new value for the **MSRP** column.

Afterwards, we concatenated the two subsetted data sets back together to continue our cleaning process.

The last part of cleaning the **MSRP** column was to remove any characters that were not numbers and convert the column to a float data type so we would be able to run our analyses with it.

c. **Creating the Brand and Model columns**:
We thought it would be useful for our analyses to be able to group each car by its brand and see if we would be able to pull meaningful insights from individual car manufacturers. To do this, we created a new column in our data set called the **Brand** that took the information from the original **Model** column.

After that, we were able to remove the year and the brand name from the original **Model** column so that this column now consisted of purely the car model (otherwise known as trim).

d. **Issue of Consumer Relevance**:
Because this data set that contained vehicles from 1966 to 2023, we decided to only run our analyses on vehicles from 1990 to 2023. This is because we anticipated most consumers in 2023 are only going to seriously consider purchasing cars from this time period. Cars that are beyond this range are more than likely either not going to be looked at, or they will be rare collector's vehicles that are more expensive than the cars from that time period (which will skew results).

e. **Removal of Outliers**:
In order to properly prepare our data for the future analyses we would conduct, we had to remove any egregious outliers that were present within the data. Using the Interquartile Rule to find our outliers, we determined that any **Price** value greater than $96,621.50 was an outlier, as this was our upper fence value. Our lower fence value was calculated to be a negative number, but since we could not have any negative **Price** values, our lower fence was defined as $0.

This resulted in a removal of 9,890 data points that were deemed outliers.

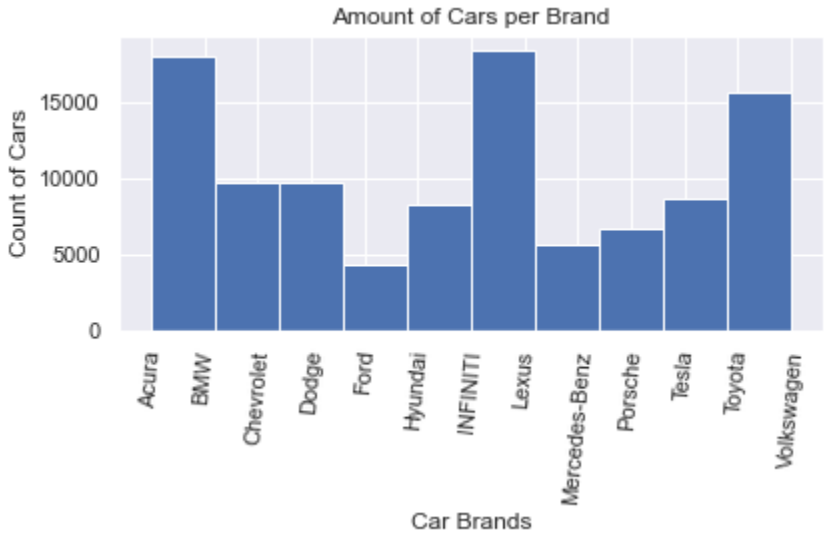
(c) **Interesting Insights on Data Set**

There were some interesting finds as we began to explore the Cars data set. Below are a few notable insights.

a. **Car Brands**

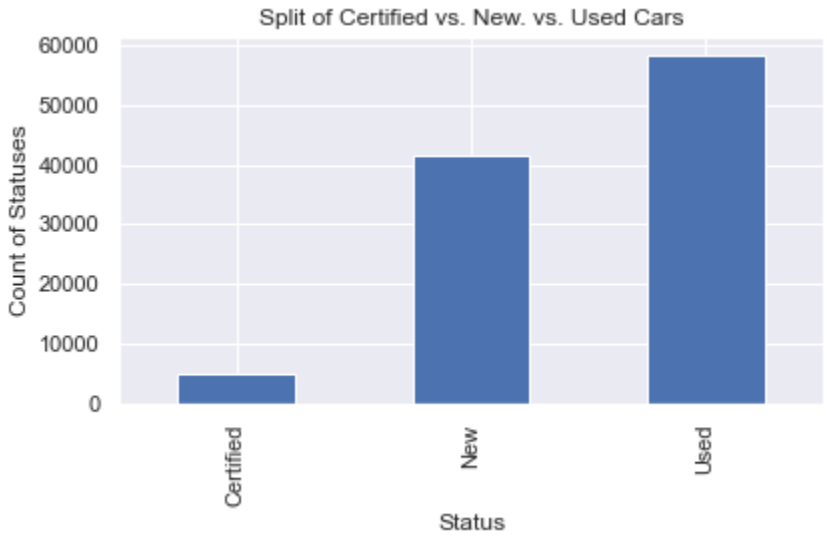
The top 3 car brands that were present within our data set were Acura, Hyundai, and Volkswagen.

Conversely, the lowest 3 car brands present were Dodge, Porsche, and Tesla.



b. **Status Split**

The Cars data set was dominated by Used vehicles, followed by New vehicles, and then Certified vehicles. It is worth noting that this may be due in part to how this data was collected and the fact that the older vehicles may not have had as good of record keeping when it came to car sales.



Linear Regression Analysis

(a) **Preparation**

For our dataset, we want to determine factors that could affect car prices. We will be using Multiple Linear Regression to predict price.

As stated earlier, our Cars data set contains the variables **Model**, **Status**, **Mileage**, **Brand**, **Price**, **MSRP**, and **Year**. We found that we have both numerical and categorical variables (e.g. Status, Brand, Year). To better perform linear regression tests we first have to clean the data. Turning all the categorical variables into dummy variables in order to run regression analysis. Because we have more than 100,000 data points, we decided to check for outliers and drop them.

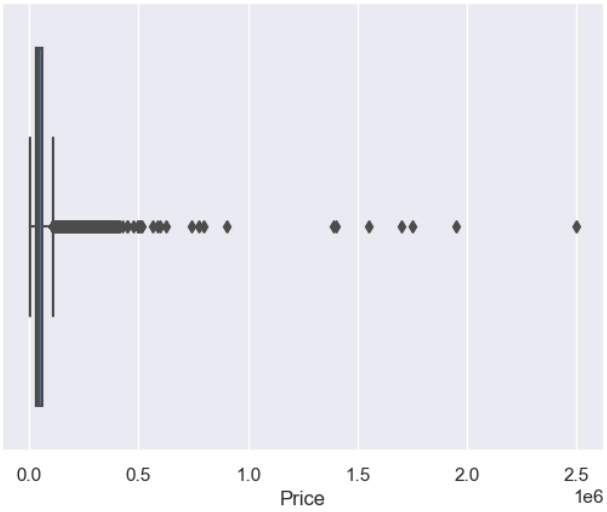


Figure 1: Price before removing outliers

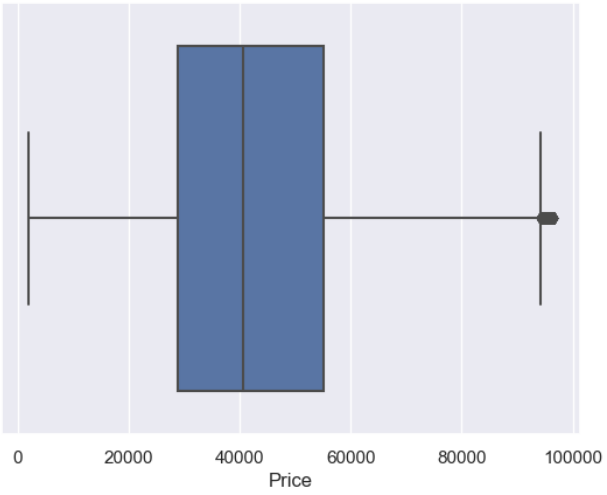


Figure 2: Price after removing outliers

We first ran a regression model with all 5 predictors and found that **Status** and **Model** are not significant to Price. Therefore, we will not be using MSRP as our variable since it is 62% similar to the

Price column.

(b) **Different Models**

Model 1: Regression Analysis of first predictive model with Brands, Mileage, and Year

$$\begin{aligned}\text{Price} = \beta_0 - \beta_1(Mileage) + \beta_2(BMW) + \beta_3(Chevrolet) + \beta_4(Dodge) + \beta_5(Ford) - \beta_6(Hyundai) + \\ \beta_7(INFINITI) - \beta_8(Lexus) + \beta_9(Mercedes - Benz) + \beta_{10}(Porsche) + \beta_{11}(Tesla) - \\ \beta_{12}(Toyota) - \beta_{13}(Volkswagen) + \beta_{14}(1990) + \beta_{15}(1991) - \beta_{16}(1992) - \\ \beta_{17}(1993) + ... + \beta_{45}(2021) + \beta_{46}(2022) + \beta_{47}(2023) + \varepsilon\end{aligned} \tag{1}$$

OLS Regression Results

Dep. Variable:	Price	R-squared:	0.604
Model:	OLS	Adj. R-squared:	0.604
Method:	Least Squares	F-statistic:	3478.
Date:	Tue, 21 Mar 2023	Prob (F-statistic):	0.00
Time:	17:47:16	Log-Likelihood:	-1.1374e+06
No. Observations:	104973	AIC:	2.275e+06
Df Residuals:	104926	BIC:	2.275e+06
Df Model:	46		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	3.011e+04	326.620	92.186	0.000	2.95e+04	3.07e+04
Mileage	-0.1267	0.002	-67.183	0.000	-0.130	-0.123
BMW	1.307e+04	184.168	70.975	0.000	1.27e+04	1.34e+04
Chevrolet	3785.0991	180.851	20.929	0.000	3430.634	4139.564
Dodge	2306.7638	180.759	12.762	0.000	1952.478	2661.050
Ford	9742.6659	234.567	41.535	0.000	9282.918	1.02e+04
Hyundai	-9934.0482	186.192	-53.354	0.000	-1.03e+04	-9569.115
INFINITI	5680.9204	183.686	30.927	0.000	5320.898	6040.942
Lexus	6640.3707	179.209	37.054	0.000	6289.124	6991.617
Mercedes-Benz	1.802e+04	214.458	84.021	0.000	1.76e+04	1.84e+04
Porsche	2.432e+04	204.475	118.957	0.000	2.39e+04	2.47e+04
Tesla	1.34e+04	187.825	71.357	0.000	1.3e+04	1.38e+04
Toyota	-3121.3663	209.624	-14.890	0.000	-3532.227	-2710.505
Volkswagen	-6861.7137	177.916	-38.567	0.000	-7210.426	-6513.002
1990	4652.2320	3607.133	1.290	0.197	-2417.701	1.17e+04
1991	1.424e+04	3199.584	4.452	0.000	7973.656	2.05e+04
1992	-1588.9675	3454.105	-0.460	0.646	-8358.967	5181.032
1993	-2216.7418	2616.702	-0.847	0.397	-7345.443	2911.959
1994	2360.7888	2616.703	0.902	0.367	-2767.913	7489.491
1995	1.802e+04	2401.298	7.503	0.000	1.33e+04	2.27e+04
1996	1.165e+04	2400.380	4.851	0.000	6940.621	1.64e+04

1997	588.1502	2093.239	0.281	0.779	-3514.570	4690.871
1998	-2751.1420	1928.018	-1.427	0.154	-6530.032	1027.748
1999	-1.469e+04	1394.877	-10.529	0.000	-1.74e+04	-1.2e+04
2000	-1.309e+04	1278.402	-10.240	0.000	-1.56e+04	-1.06e+04
2001	-1.036e+04	1121.054	-9.239	0.000	-1.26e+04	-8160.339
2002	-8095.5707	957.064	-8.459	0.000	-9971.404	-6219.737
2003	-4174.1395	927.469	-4.501	0.000	-5991.966	-2356.313
2004	-5809.8395	822.662	-7.062	0.000	-7422.246	-4197.433
2005	-4818.9127	759.520	-6.345	0.000	-6307.562	-3330.264
2006	-3943.6322	627.507	-6.285	0.000	-5173.537	-2713.727
2007	-3049.9975	611.676	-4.986	0.000	-4248.875	-1851.120
2008	-4174.2381	554.935	-7.522	0.000	-5261.903	-3086.573
2009	-2605.1121	606.275	-4.297	0.000	-3793.402	-1416.822
2010	-4058.3685	535.211	-7.583	0.000	-5107.375	-3009.362
2011	-4181.1551	459.850	-9.092	0.000	-5082.456	-3279.854
2012	-2531.7142	426.304	-5.939	0.000	-3367.264	-1696.164
2013	-2941.0406	368.651	-7.978	0.000	-3663.592	-2218.489
2014	-2179.4380	356.728	-6.110	0.000	-2878.621	-1480.255
2015	-486.4566	332.018	-1.465	0.143	-1137.207	164.294
2016	1799.3658	322.537	5.579	0.000	1167.198	2431.534
2017	3391.5288	314.639	10.779	0.000	2774.840	4008.217
2018	4894.1217	305.300	16.031	0.000	4295.738	5492.506
2019	6055.1261	297.107	20.380	0.000	5472.801	6637.451
2020	9069.4247	300.406	30.191	0.000	8480.634	9658.216
2021	1.257e+04	308.039	40.800	0.000	1.2e+04	1.32e+04
2022	1.667e+04	313.507	53.187	0.000	1.61e+04	1.73e+04
2023	2.189e+04	312.264	70.104	0.000	2.13e+04	2.25e+04

Omnibus:	8557.605	Durbin-Watson:	1.915
Prob(Omnibus):	0.000	Jarque-Bera (JB):	11734.289
Skew:	0.688	Prob(JB):	0.00
Kurtosis:	3.887	Cond. No.	9.83e+15

Figure 3: Model 1: Price = Mileage + Brand + Year + ε

From the above model, we can see that both R-squared value and the Adjusted R-squared value is 0.604, and both AIC and BIC value is approximately 2,275,000. Some of the p-values are greater than 0.05 and some are 0. However, the R-squared and Adjusted R-squared values are more than 0.5, which looks promising.

Model 2: Regression Analysis of second predictive model with Mileage and Brand

$$\begin{aligned}\text{Price} = \beta_0 - \beta_1(Mileage) + \beta_2(BMW) + \beta_3(Chevrolet) + \beta_4(Dodge) + \beta_5(Ford) - \beta_6(Hyundai) + \\ \beta_7(INFINITI) - \beta_8(Lexus) + \beta_9(Mercedes - Benz) + \beta_{10}(Porsche) + \beta_{11}(Tesla) - \\ \beta_{12}(Toyota) - \beta_{13}(Volkswagen) + \varepsilon\end{aligned} \tag{2}$$

OLS Regression Results

Dep. Variable:	Price	R-squared:	0.548
Model:	OLS	Adj. R-squared:	0.548
Method:	Least Squares	F-statistic:	9796.
Date:	Tue, 21 Mar 2023	Prob (F-statistic):	0.00
Time:	17:47:24	Log-Likelihood:	-1.1443e+06
No. Observations:	104973	AIC:	2.289e+06
Df Residuals:	104959	BIC:	2.289e+06
Df Model:	13		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	4.76e+04	141.950	335.319	0.000	4.73e+04	4.79e+04
Mileage	-0.2989	0.001	-271.958	0.000	-0.301	-0.297
BMW	1.16e+04	195.872	59.228	0.000	1.12e+04	1.2e+04
Chevrolet	5189.1561	191.348	27.119	0.000	4814.117	5564.195
Dodge	1696.2041	190.507	8.904	0.000	1322.813	2069.595
Ford	1.033e+04	244.089	42.310	0.000	9848.894	1.08e+04
Hyundai	-8965.9047	198.451	-45.179	0.000	-9354.867	-8576.943
INFINITI	5711.8016	195.890	29.158	0.000	5327.861	6095.742
Lexus	4894.7663	190.307	25.720	0.000	4521.768	5267.765
Mercedes-Benz	1.993e+04	225.918	88.231	0.000	1.95e+04	2.04e+04
Porsche	1.883e+04	210.605	89.401	0.000	1.84e+04	1.92e+04
Tesla	9981.4452	195.865	50.961	0.000	9597.552	1.04e+04
Toyota	-730.8289	221.913	-3.293	0.001	-1165.774	-295.883
Volkswagen	-7012.4202	189.301	-37.044	0.000	-7383.447	-6641.393

Omnibus:	11751.545	Durbin-Watson:	1.900
Prob(Omnibus):	0.000	Jarque-Bera (JB):	24016.750
Skew:	0.715	Prob(JB):	0.00
Kurtosis:	4.856	Cond. No.	6.26e+05

Figure 4: Model 2: Price = Mileage + Brand + ε

From the above model, we can see that both R-squared value and the Adjusted R-squared value is 0.548, and both AIC and BIC value is approximately 2,289,000. The p-values in this model are mostly 0's except for Toyota.

Model 3: Regression Analysis of third predictive model with Mileage and Year

$$\begin{aligned} \text{Price} = \beta_0 - \beta_1(Mileage) + \beta_2(1990) + \beta_3(1991) - \beta_4(1992) - \\ \beta_5(1993) + ... + \beta_{33}(2021) + \beta_{34}(2022) + \beta_{35}(2023) + \varepsilon \end{aligned} \tag{3}$$

OLS Regression Results

Dep. Variable:	Price	R-squared:	0.402
Model:	OLS	Adj. R-squared:	0.402
Method:	Least Squares	F-statistic:	2075.
Date:	Tue, 21 Mar 2023	Prob (F-statistic):	0.00
Time:	17:47:25	Log-Likelihood:	-1.1590e+06
No. Observations:	104973	AIC:	2.318e+06
Df Residuals:	104938	BIC:	2.318e+06
Df Model:	34		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	4.458e+04	362.648	122.928	0.000	4.39e+04	4.53e+04
Mileage	-0.2079	0.002	-92.256	0.000	-0.212	-0.204
1990	8469.9773	4431.049	1.912	0.056	-214.820	1.72e+04
1991	2.068e+04	3930.480	5.261	0.000	1.3e+04	2.84e+04
1992	-3943.1249	4243.306	-0.929	0.353	-1.23e+04	4373.698
1993	-6821.4469	3214.370	-2.122	0.034	-1.31e+04	-521.325
1994	3814.5397	3214.652	1.187	0.235	-2486.134	1.01e+04
1995	2.505e+04	2949.106	8.495	0.000	1.93e+04	3.08e+04
1996	1.453e+04	2948.759	4.929	0.000	8754.611	2.03e+04
1997	3872.1774	2571.526	1.506	0.132	-1167.979	8912.334
1998	163.4414	2368.561	0.069	0.945	-4478.906	4805.789
1999	-5283.4194	1711.132	-3.088	0.002	-8637.216	-1929.623
2000	-5406.6503	1569.162	-3.446	0.001	-8482.187	-2331.114
2001	-2662.6753	1375.502	-1.936	0.053	-5358.640	33.289
2002	-4343.8902	1175.234	-3.696	0.000	-6647.334	-2040.447
2003	-17.9967	1139.019	-0.016	0.987	-2250.459	2214.466
2004	-2229.3402	1010.318	-2.207	0.027	-4209.549	-249.131
2005	-1495.1239	932.674	-1.603	0.109	-3323.153	332.905
2006	-541.9357	770.411	-0.703	0.482	-2051.931	968.060
2007	-709.5030	751.061	-0.945	0.345	-2181.572	762.566
2008	-1328.2555	681.406	-1.949	0.051	-2663.803	7.292
2009	-2757.5092	744.743	-3.703	0.000	-4217.195	-1297.824
2010	-4887.0515	657.117	-7.437	0.000	-6174.993	-3599.110
2011	-3876.5904	564.626	-6.866	0.000	-4983.250	-2769.931
2012	-5410.0846	523.127	-10.342	0.000	-6435.407	-4384.762
2013	-4283.9924	452.354	-9.470	0.000	-5170.601	-3397.384
2014	-3074.4277	437.801	-7.022	0.000	-3932.512	-2216.343
2015	-2678.6723	406.766	-6.585	0.000	-3475.928	-1881.417
2016	118.8276	394.665	0.301	0.763	-654.711	892.366
2017	485.1777	385.219	1.259	0.208	-269.847	1240.203
2018	2477.9454	372.541	6.651	0.000	1747.770	3208.121
2019	494.8006	361.182	1.370	0.171	-213.111	1202.712
2020	2163.2141	364.347	5.937	0.000	1449.099	2877.329
2021	5244.7065	373.819	14.030	0.000	4512.026	5977.387
2022	8334.5029	375.175	22.215	0.000	7599.164	9069.842
2023	1.043e+04	372.290	28.008	0.000	9697.284	1.12e+04

Omnibus:	9625.246	Durbin-Watson:	1.309
Prob(Omnibus):	0.000	Jarque-Bera (JB):	12769.930
Skew:	0.782	Prob(JB):	0.00
Kurtosis:	3.690	Cond. No.	1.09e+16

Figure 5: Model 2: Price = Mileage + Year + ε

From the above model, we can see that both R-squared value and the Adjusted R-squared value is

0.402, and both AIC and BIC value is approximately 2,318,000. Some of the p-values are greater than 0.05 and some appear to be 0's.

Model 4: Regression Analysis of fourth predictive model with only Mileage

$$\text{Price} = \beta_0 - \beta_1(Mileage) + \varepsilon \tag{4}$$

OLS Regression Results

Dep. Variable:	Price	R-squared:	0.379
Model:	OLS	Adj. R-squared:	0.379
Method:	Least Squares	F-statistic:	6.397e+04
Date:	Tue, 21 Mar 2023	Prob (F-statistic):	0.00
Time:	17:47:30	Log-Likelihood:	-1.1610e+06
No. Observations:	104973	AIC:	2.322e+06
Df Residuals:	104971	BIC:	2.322e+06
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	5.253e+04	60.315	870.985	0.000	5.24e+04	5.27e+04
Mileage	-0.3094	0.001	-252.926	0.000	-0.312	-0.307

Omnibus:	11344.851	Durbin-Watson:	1.385
Prob(Omnibus):	0.000	Jarque-Bera (JB):	16957.944
Skew:	0.813	Prob(JB):	0.00
Kurtosis:	4.110	Cond. No.	6.26e+04

Figure 6: Model 2: Price = Mileage + ε

From the above model, we can see that the R - squared value is 0.379, the Adjusted R - squared value is 0.379, and both AIC and BIC value is approximately 2,322,000. All p-values appear to be 0.

(c) **Final Model Selection - Techniques**

We will be using R-squared and Adjusted R-squared values to pick the best model. The reason is that R-squared tells us an estimate of the relationship between dependent and independent variables. R-squared range between 0 and 1; R-squared = 0 means there is no linear relationship between the variables versus an R-squared = 1 shows a perfect linear relationship.

In our case, out of all the models, Model 1 has the highest value. We got R-squared score of 0.604 which means 60.4% of our dependent variable can be explained using our independent variables and has a linear relationship.

(d) **Final Model - Conclusion**

After performing Multiple Linear Regression on all four models, we conclude that Mileage, Brand and Year of a car are significant in predicting car prices. Because we have more than 100,000 data points and very few predictors in our data set, for further analysis to predict the price of a car, we could find other data sets with more predictors and try random forest regression.

However, our model only has R-squared value of 60.4 %, which could mean that about 40% unknown factors could affect the price of a car.

LU Decomposition

(a) **What is LU Decomposition?**

LU decomposition is a computationally efficient method to obtain linear regression coefficients. When programming, it is important to consider the flop count—a measure of how computationally expensive an algorithm can be. In real life, datasets can be extremely large, so choosing the appropriate algorithm is imperative to reduce computational time and expense.

The goal in linear regression is to solve $(X^TX)^{-1}\beta = X^Ty$ for β, where X is our dataset matrix, y is the response, and β is the coefficients of the predictor variables of X. LU decomposition decomposes the square matrix (X^TX) into L (an upper triangular matrix), and U (a lower triangular matrix).In doing so, the linear equation becomes $(LU)\beta = X^Ty$ (1).

In Python, the SciPy function "lu()" can decompose a given square matrix into L and U. This process takes n^3 flops. We decided to test LU decomposition on our Model 1 to try and obtain the coefficients from seen in Figure 3. In this case, X was the data frame we used to fit the linear model and y was the response we were modeling, which was **Price**.

(b) **Forward and Backward Substitution**

Instead of using the Python SciPy function "solve()" to solve the linear equation (1) for β, we used forward and backward substitution. Each of these algorithms takes $O(n^2)$ operations. Using this, the entire LU decomposition algorithm to obtain the coefficients would take n^2 flops, which is lower and more efficient than the "solve()" function.

Before doing so, we need to ensure that the matrices L and U are in the correct form; L must have every diagonal value of 1 and U can have no diagonal entry of 0. Once confirmed, we perform forward substitution on L and X^Ty and then do backward substitution on U and the result from forward substitution. This results in an array of β, which are the linear regression coefficients. However, the resulting array was significantly different than the regression coefficients obtained by fitting the linear model. Because of this, we decided to try to solve the equation using the permutation matrix P, which was also obtained in the decomposition of (X^TX).

We confirmed that $PA = LU$, and thus, the permutation matrix P allows us to rewrite the linear equation from $(X^TX)\beta = X^Ty$ into $(PLU)\beta = X^Ty$ into $(LU)\beta = PX^Ty$, since $P^{-1} = P$. To solve for β, we can use forward and backward substitution like before, only the result matrix is PX^Ty instead of X^Ty. As before, each forward and backward substitution algorithm takes $O(n^2)$ flops, so the total flops for the LU algorithm is $O(n^2)$. The resulting array of coefficients was also significantly different than the coefficients obtained through the linear regression model fit.

(c) **Conclusion**

Using the Python "time" package, we timed how long the LU decomposition algorithm took to run—from decomposing the matrix to the final step of backward substitution. The overall time was 0.002030134 seconds, which is sufficiently good given that our original data set had more than 100,000 entries, and was faster than the time it took to run the OLS regression, which was around 0.215165376 seconds.

We also solved $(LU)\beta = PX^Ty$ without using forward and backward substitution (by using the "solve()" function) and we obtained the same β coefficients as we did using forward and backward substitution. This indicates that our code for forward and backward substitution worked. However, our final LU decomposition results did not match the regression coefficients found for Model 1 above. The difference between our OLS coefficients and LU coefficients ranged widely, with some variables have just a 1.00 value difference, while others had close to -9.31e+04 value difference. Figure 7 shows the two arrays of coefficients; the left side shows the coefficients obtained through fitting an OLS model and the right array are the coefficients obtained through our LU decomposition.

Mileage	-0.126665
BMW	13071.251594
Chevrolet	3785.099118
Dodge	2306.763778
Ford	9742.665933
Hyundai	-9934.048208
INFINITI	5680.920363
Lexus	6640.370719
Mercedes-Benz	18018.911544
Porsche	24323.627144
Tesla	13402.608433
Toyota	-3121.366289
Volkswagen	-6861.713707
1990	4652.232038
1991	14244.797185
1992	-1588.967478
1993	-2216.741815
1994	2360.788798
1995	18016.929863
1996	11645.333142
1997	588.150236
1998	-2751.142019
1999	-14687.200745
2000	-13090.197062
2001	-10357.589642
2002	-8095.570702
2003	-4174.139541
2004	-5809.839463
2005	-4818.912723
2006	-3943.632176
2007	-3049.997485
2008	-4174.238094
2009	-2605.112071
2010	-4058.368470
2011	-4181.155145
2012	-2531.714189
2013	-2941.040610
2014	-2179.438017
2015	-486.456583
2016	1799.365850
2017	3391.528773
2018	4894.121700
2019	6055.126149
2020	9069.424723
2021	12567.879773
2022	16674.488822
2023	21890.957827
intercept	30109.670849

```
array([[ 1.12175736e+00],
       [-1.42886961e+05],
       [-1.53193743e+05],
       [-1.62355958e+05],
       [-1.43255845e+05],
       [-1.63888496e+05],
       [-1.52300994e+05],
       [-1.94993994e+05],
       [-1.99366414e+05],
       [-1.26097601e+05],
       [-1.27427392e+05],
       [-1.58873666e+05],
       [-1.60920862e+05],
       [ 1.36183561e+05],
       [ 3.21678594e+05],
       [ 1.01748771e+05],
       [ 8.43450923e+04],
       [ 2.53895773e+05],
       [ 9.44223748e+04],
       [ 1.03099189e+05],
       [ 8.39197924e+04],
       [ 9.35515883e+04],
       [ 8.38102009e+04],
       [ 7.73108821e+04],
       [ 7.85261880e+04],
       [ 7.72367763e+04],
       [ 8.86429342e+04],
       [ 1.02335823e+06],
       [ 5.35792546e+04],
       [ 5.66373013e+04],
       [ 9.43047240e+05],
       [ 6.22244108e+04],
       [ 5.25282759e+04],
       [ 5.60387567e+04],
       [ 5.79871620e+04],
       [ 5.52508767e+04],
       [ 7.12841597e+04],
       [ 7.88181280e+04],
       [ 9.26864091e+04],
       [ 1.03069532e+05],
       [ 1.16599409e+05],
       [ 1.33324925e+05],
       [ 1.43194925e+05],
       [ 1.46007068e+05],
       [ 1.78483535e+05],
       [ 2.15747436e+05],
       [ 2.16337522e+05],
       [-1.45921819e+04]])
```


Figure 7: Left: OLS coefficients; Right: LU coefficients

k-NN Regression with Cross-Validation

(a) **Preparation**

Our goal with k-NN regression is to use the predictor variables given to us by the dataset to predict the car price and maximize that prediction accuracy.

To do so, we first cleaned the data to be able to run through the k-NN algorithm. **Status** has to be turned into binary values so that the k-NN algorithm could read the predictor in quantitative values; therefore, we turned any entries that said "Used" to 0 and anything else to 1.

Then, we used cross-validation to resample the data set into training and testing sets. To decide how many neighbors to consider when predicting the car price, we ran the algorithm through the intial dataset using all predictors and minimized mean squared error. From there, we decided to use 50 neighbors.

(b) **k-NN Algorithm**

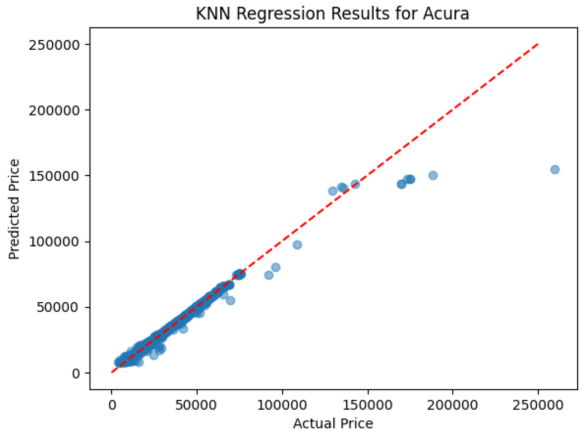
Using the KNeighborsRegressor function from the sklearn package, we initialized the k-NN regressor and fit the model on the training data.

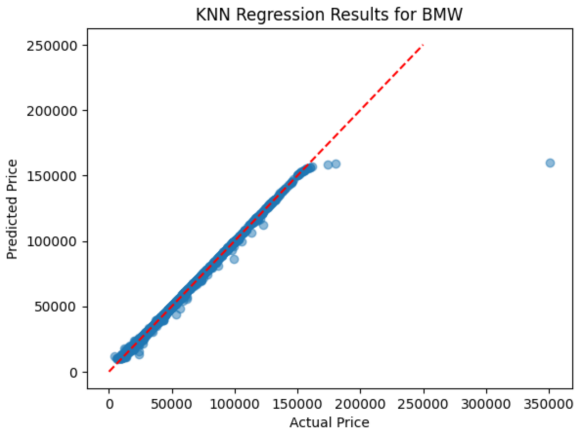
Then, we used the information to predict the values for the testing set. To analyze how accurate the prediction of car prices were on the testing set, we calculated the mean squared error of the predictions and created visualizations that displayed the accuracy of the regression compared to a perfectly accurate k-NN line. When running the algorithm through our preliminary dataset that removed car brand and used the rest of the predictors (**Year, Status, Mileage**), we realized that our algorithm did not give such accurate results.

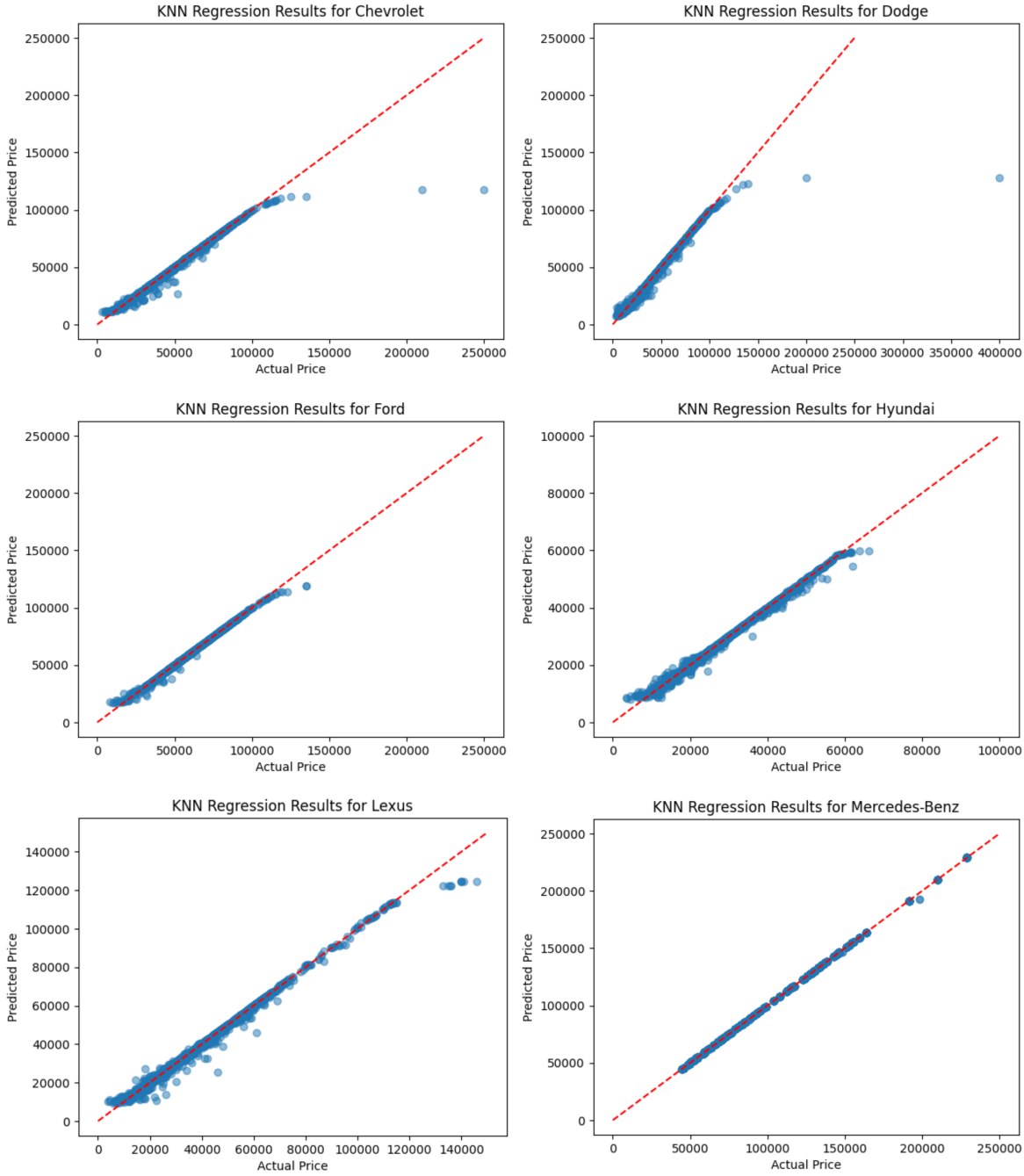
Therefore, we subsetted the data set by car brand which we hypothesized would be a very important predictor for predicting car price but could not initially include in the algorithm. we also created visualizations for these data sets so that we can easily compare the algorithm from car brand to car brand.

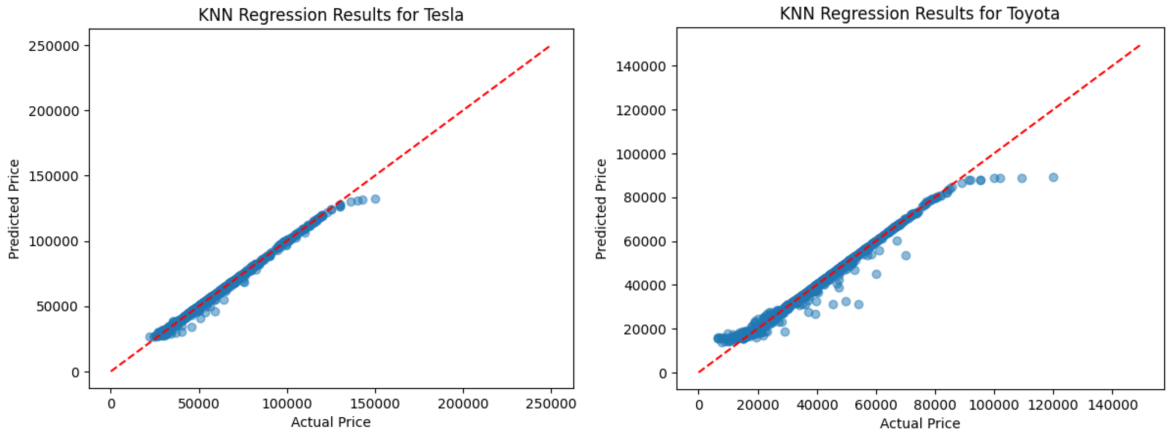
(c) **Analysis**

The initial results using all predictors and disregarding car brand did not show any significant results. After subsetting the data set by car brand and fitting a k-NN regression specific to each brand, the graphs show a significant increase in accuracy compared to our preliminary results with the dataset that did not include car brands.






KNN Regression Results for Chevrolet
Predicted Price
Actual Price
KNN Regression Results for Dodge
Predicted Price
Actual Price
KNN Regression Results for Ford
Predicted Price
Actual Price
KNN Regression Results for Hyundai
Predicted Price
Actual Price
KNN Regression Results for Lexus
Predicted Price
Actual Price
KNN Regression Results for Mercedes-Benz
Predicted Price
Actual Price



Above are the plots that shows each of the k-NN regression specific to the car brands provided in the dataset. The red dotted line represents a perfectly accurate k-NN line and the farther the dots deviate from the line show the less accurate k-NN was at predicting that car price. The regression predictions seem to be close to accurate.

A common trend amongst the car brand is the the dots follow the dotted line closely near zero and then have some dots that plateau. This could possibly be due to the fact that there are cars that are priced more than what is expected in the algorithm based on the predictors (Year, Status, Mileage). Some luxury cars may be priced significantly more than other cars despite it having similar features, explaining the dots that have a high actual price and deviate from the regression line.

(d) **Conclusion**

From the k-NN regression, we conclude that car brand is highly significant when it comes to predicting car price. The other predictors (Year, Status, Mileage) also were fairly accurate when we subset the data by car brands; however, we conclude that there is not enough evidence to draw a causal relationship between these predictors and car price based off the k-NN regression.

Simulation Study

(a) **Setting Up the Simulation**

In order to prepare our simulation, we needed to create a data frame containing random values for our X values. This simulation data frame contained 46 columns and 1 intercept column for a total of 47 columns in order to match how many β values we had in our true model.

Next, we created a simulation ϵ by generating 48 normally distributed random values.

Finally, we created a simulation Y using the simulation data frame, our true values of β, and our simulation error ϵ.

(b) **Outcome and Reflection**

Unfortunately, the outcome of our simulation study did not yield our expected results. A majority of the simulation values of β were off from our true values of β by a relatively large amount.

There are a few reasons why this could be the case. First, the simulation data contained random values that were generated by NumPy's random.randint() function. In a different situation, this method of generating data would be acceptable if the original data set only contained quantitative variables.

Our Cars data set, however, had a plethora of categorical variables that far outnumbered the amount of quantitative variables we had available. We used one-hot encoding with our categorical variables, and some of these categorical variables were linked with each other. For example, a used car sold in 2012 would have 1's in the **Status** and **Year** category and 0's everywhere else. This further complicated the process of simulating the correct types of data.

For future simulation studies, we would have to find an alternative way to generate a test data set that would be able to accurately replicate all the categorical data in order to yield more expected results.

Conclusion
Throughout this project, we sought to answer the question: what factor(s) influence the price of a car?

Through our Multiple Linear Regression analysis, we found that a car's mileage, brand, and manufactured year are all relevant and significant predictors of the price of a car.

Using our k-NN algorithm, we found that car brand is highly significant when predicting a price of a car. Contrary to our linear regression conclusion, however, we found that while the other predictors of Year, Status, and Mileage are also important, we cannot conclude a relationship between these specific predictors and a car's price—at least with the use of our kNN algorithm.

With this, the ultimate conclusion that we draw is that a car's brand is a reasonable predictor of what a car's price is going to be. This makes sense, as there are certain car brands that are notorious for being expensive (e.g. luxury brands such as Mercedes-Benz) and other car brands that seek to be more affordable for the average consumer (e.g. Toyota).

This conclusion should be taken with a grain of salt. There were big limitations that hindered some of our analyses. The biggest limitations was the fact our data set only had seven variables to use, most of which were categorical variables. In hindsight, it would be advisable to use a data set that has more predictors available, such as number of cylinders or if the car is a specialty vehicle (like a sports car). This would make for a more precise analysis of what actually influences the price of the car the most.

Additionally, this data set did not tell us whether the car was actually sold or not. It only included information on the sale price and the MSRP price of the vehicle. Understanding which cars were actually sold versus which cars were not would paint a better picture of what influences a car sale, as we could focus more of our attention on the sold cars and run a separate analysis on the unsold cars to see if there was some relationship for those two groups.

References and Acknowledgements

(a) **Data Preparation References**
1. https://www.statology.org/interquartile-range-python/
2. https://seaborn.pydata.org/generated/seaborn.histplot.html

(b) **LU Decomposition References**
1. https://courses.physics.illinois.edu/cs357/sp2020/notes/ref-9-linsys.html
2. $https://johnfoster.pge.utexas.edu/numerical-methods-book/LinearAlgebra_LU.html$
3. https://zief0002.github.io/matrix-algebra/statistical-application-estimating-regression-coefficients-with-lu-decomposition.html
4. STA 141C TA: Eunseong Bae

(c) **Linear Regression References**
1. $https://www.sfu.ca/\ mjbrydon/tutorials/BAinPy/10_multiple_regression.html$
2. https://www.geeksforgeeks.org/ordinary-least-squares-ols-using-statsmodels/
3. https://www.statology.org/ols-regression-python/

(d) **k-NN Algorithm References**
1. https://realpython.com/knn-python/
2. https://towardsdatascience.com/building-a-k-nearest-neighbors-k-nn-model-with-scikit-learn-51209555453a
3. https://www.datacamp.com/tutorial/k-nearest-neighbor-classification-scikit-learn

(e) **Simulation Study References**
1. https://datascience.stackexchange.com/questions/26555/valueerror-shapes-1-10-and-2-not-aligned-10-dim-1-2-dim-0
2. STA 141C TA: Eunseong Bae

Python Code

(1) Import Python Libraries

```
import pandas as pd
import numpy as np
import random
import matplotlib.pyplot as plt
import scipy
from scipy import stats
import statsmodels.api as sm
import pylab as py
import seaborn as sns
```

(2) Data Cleaning and Preparation

```
cars_1 = pd.read_csv('car_data.csv')
cars_1 = cars_1[['Model', 'Year', 'Status', 'Mileage', 'Price', 'MSRP']]

# If Mileage is "Not available", replace it with "0 mi" as the "Not available" is only
# applicable with New cars.
cars_1['Mileage'] = cars_1['Mileage'].replace('Not available','0 mi.')

# Check for unique values.
cars_1['Status'].value_counts()
cars_1['Mileage'].value_counts()
cars_1['MSRP'].value_counts()
cars_1['Year'].value_counts()

# Remove the years that are below 1990.
cars_1 = cars_1[cars_1.Year >= 1990]
cars_1['Year'].value_counts()

# Now we only have years 1990 and above.

cars_1[cars_1['MSRP'] == 'Not specified'].sample(n = 25)

# Factors to take into consideration when evaluating new vs. used MSRP:

# Used vehicles will have different mileage, which will make the actual sale price vary greatly.
# With that said, some of these vehicles have much better retention in price than others.
# For example: The Porsche 911 will retain their value better than a BMW 750.
#(Both are within the luxury segment).

# Most (but not all) of the cars with unspecified MSRPs have a mark-up within the Price.

len(cars_1[cars_1["MSRP"].str.contains("Not specified")]) / len(cars_1)*100

# About 62% of the data set have unspecified prices within the MSRP column.
# It would ultimately take too much time to find the MSRP for each individual car.
# It is in our best interest to replace the "Not specified" with the sell price so we can focus on
# the analysis.

# Where the MSRP is "Not specified", this will change into whatever
#the Price category has stated.
```

```
cars_1.loc[cars_1['MSRP'] == 'Not specified', 'MSRP'] = cars_1['Price']

# Get rid of unnecessry symbols in columns so we can convert to int for data analysis.
cars_1["Price"] = cars_1["Price"].str.replace('[$,]','', regex = True)
# Before we can convert to int, we need to address "Not Priced"
cars_1["MSRP"] = cars_1['MSRP'].str.replace('[$,MSRP]', '', regex = True)
cars_1['Mileage'] = cars_1['Mileage'].str.replace('[mi.,]', '', regex = True)

# Because the "Not priced" values only make up less than a percent of the data set,
# we will remove them entirely.
len(cars_1.loc[cars_1['Price'] == 'Not Priced']) / len(cars_1)*100

cars_1 = cars_1[cars_1['Price'].str.contains("Not Priced") == False]
pd.to_numeric(cars_1['Price'])

# Create a function that will take the price drop number and subtract that from the price.
# That will become the new "MSRP" number.

cars_2 = cars_1[cars_1['MSRP'].str.contains('price drop') == True]
cars_2['MSRP'] = cars_2['MSRP'].str.replace('price drop', '', regex = True)
cars_2['Price'] = pd.to_numeric(cars_2['Price'])
cars_2['MSRP'] = pd.to_numeric(cars_2['MSRP'])

# The new MSRP column will be the Price - MSRP price drop.
cars_2['MSRP'] = cars_2['Price'] - cars_2['MSRP']

cars_1 = cars_1[cars_1['MSRP'].str.contains("price drop") == False]

merge_cars = [cars_1, cars_2]
cars = pd.concat(merge_cars)

# Now we want a column that is just the brand of the vehicle.

def get_brand(model):
    mod = model.split(" ")
    brand = mod[1]
    return brand

cars['Brand'] = cars['Model'].apply(get_brand)

# Similarly, we want to change the "Model" column to just the trim.
# This is because we now have the brand and the year.

def get_trim(model):
    mod = model.split(" ")
    return " ".join(mod[2:4])

cars['Model'] = cars['Model'].apply(get_trim)

# There are different levels of certification in Status, so we are going to change
# it to just "Certified"
cars.loc[cars['Status'].str.contains('Certified'), 'Status'] = 'Certified'
```

```
# Final data frame to do analysis on.
cars = cars[['Brand', 'Model', 'Year', 'Status', 'Mileage', 'Price', 'MSRP']]

# Changing all the necessary int variables to int.
cars = cars.astype({"Mileage": "int", "Price": "float", "MSRP": "int"})
# We needed to reset the index because when we merged the data sets,
# we did not account for the indices.
cars = cars.reset_index(drop = True)
display(cars)
```

	Brand	Model	Year	Status	Mileage	Price	MSRP
0	Acura	TLX A-Spec	2022	New	0	49445	49445
1	Acura	RDX A-Spec	2023	New	0	50895	50895
2	Acura	TLX Type	2023	New	0	57745	57745
3	Acura	TLX Type	2023	New	0	57545	57545
5	Acura	TLX A-Spec	2023	New	0	50195	50195
...	...	...	...	...	...	...	...
115722	Volkswagen	Jetta 1.4T	2019	Certified	48496	22789	22662
115723	Volkswagen	Golf GTI	2015	Used	70646	16994	16489
115755	Volkswagen	Passat 3.6L	2018	Used	58535	22000	21800
115756	Volkswagen	Arteon 2.0T	2022	New	0	41090	39090
115759	Volkswagen	Beetle	2012	Used	100395	9994	9742

114863 rows × 7 columns

(3) Removing Outliers

```
Q1 = cars.Price.quantile([0.25]) #28994
Q3 = cars.Price.quantile([.75]) #56045
IQR = 56045 - 28994 #27,051

upper_fence = Q3 + (1.5*IQR) # £96,621.50
lower_fence = Q1 - (1.5*IQR) # £0 because our lower fence is a negative number (-£11,582).

print(lower_fence)
# Output: 0.25   -10592.5

outlier_upper = len([i for i in cars['Price'] if i > 96621.5])
#9,890 entries are greater than our upper fence.

cars = cars[cars['Price'] < 96621.5]
display(outlier_upper)
#Output: 9890

#Drop Model column since it is not significant
cars.drop('Model', axis = 1)
```

New Dataframe with outliers removed

	Brand	Year	Status	Mileage	Price	MSRP
0	Acura	2022	New	0	49445.0	49445
1	Acura	2023	New	0	50895.0	50895
2	Acura	2023	New	0	57745.0	57745
3	Acura	2023	New	0	57545.0	57545
4	Acura	2023	New	0	50195.0	50195
...	...	...	...	...	...	...
114858	Volkswagen	2019	Certified	48496	22789.0	22662
114859	Volkswagen	2015	Used	70646	16994.0	16489
114860	Volkswagen	2018	Used	58535	22000.0	21800
114861	Volkswagen	2022	New	0	41090.0	39090
114862	Volkswagen	2012	Used	100395	9994.0	9742

104973 rows × 6 columns

(4) Linear Regression Analysis

```
# Turning categorical variables into Dummy Variables using one hot encoding
brand = pd.get_dummies(cars['Brand'])
year = pd.get_dummies(cars['Year'])

#Selecting numerical variables
df = cars.select_dtypes(exclude=["object"])
df = df[['Price', 'Mileage']]

#Join dummy variable column to dataframe
df1 = df.join(brand)
df2 = df1.join(year)

#Drop dummy variable of Brand to avoid multicollinearity
final_df = df2.drop('Acura', axis = 1)

#Add column for intercept of 1's
final_df = sm.add_constant(final_df, has_constant = 'add')

#drop Price column
col = "Price"
x = final_df.loc[:, final_df.columns != col]
x_col = x.columns.tolist()
y = final_df['Price']
```

```
# Model 1: Price ~ Mileage + Brand + Year
## creating function to get model statistics
def stats():
    x = final_df[x_col]
    results = sm.OLS(y, x).fit()
    print(results.summary())
stats()

# Model 2: Price ~ Mileage + Brand

# Drop Year columns
df_mil_brd = x.drop(x.iloc[:, 14:48],axis = 1)
x_col = df_mil_brd.columns.tolist()

def stats():
    x = df_mil_brd[x_col]
    results = sm.OLS(y, x).fit()
    print(results.summary())
stats()


# Model 3: Price ~ Mileage + Year

# Drop Brands columns
df_mil_yr = x.drop(x.iloc[:, 2:14],axis = 1)
x_col = df_mil_yr.columns.tolist()

def stats():
    x = df_mil_yr[x_col]
    results = sm.OLS(y, x).fit()
    print(results.summary())
stats()


# Model 4: Price ~ Mileage

#Select only Mileage column
df_mil = x.drop(x.iloc[:, 2:48],axis = 1)
x_col = df_mil.columns.tolist()

def stats():
    x = df_mil_yr[x_col]
    results = sm.OLS(y, x).fit()
    print(results.summary())
stats()
```

(5) LU Decomposition

```
#use dataframe "no_out_cars" that has no outliers
no_out_cars = cars

#use predictor variables of Model 1 (brand, price, mileage, year)
A_11 = no_out_cars[['Brand', 'Price', 'Mileage', 'Year']]
```

```
#Because Brand is qualitative and nominal, we need to do one-hot encoding
dummies_brand = pd.get_dummies(A_11['Brand'])

#Year is quantitative ordinal and not continuous so it also must be encoded
dummies_year = pd.get_dummies(A_11['Year'])

#Join dummy variable column to dataframe
A_merged11 = A_11.join(dummies_brand)
A_merged11 = A_merged11.join(dummies_year)

#drop original Brand and Year column bc those are dummy variables now
A_final11 = A_merged11.drop('Brand', axis = 1)
A_final11 = A_final11.drop('Year', axis = 1)

#drop 1 dummy variable column for Brandto avoid multicollinearity
A_final11 = A_final11.drop('Acura', axis = 1)

#add intercept column of 1s
A_final11["intercept"] = '1'

#determine matrix A and b (to solve Ax = b)
A = A_final11.drop('Price', axis = 1).to_numpy()
A = A.astype('i')
b = A_final11[['Price']]
b = b.to_numpy()
b = b.astype('i')

#matrix needs to be square for LU decomposition so we use normal equation definition (X'X)
A_LU = A.transpose() @ A

#Using the definition of Beta_hat, our result vector is A'b
A_b = A.transpose() @ b

#determine start time for LU computation time
import time
start_time = time.time()

#scipy function to find LU decomposition of A'A
P, L, U = scipy.linalg.lu(A_LU)

#Confirm PA = LU
print('PA =', P @ A_LU)
print('LU =', L @ U)

#Confirm A = PLU
print('A = ', A)
print('PLU =', P @ L @ U)

#function for forward substitution on L and b
def forward_substitution(L, b):

    #get number of rows
    n = L.shape[0]
```

```
    #allocate space for the solution vector y
    y = np.zeros_like(b, dtype=np.double);

    #initialize with the first row.
    y[0] = b[0] / L[0, 0]

    #Looping over rows in reverse starting from the second to last row bc the last row
    #was done right above
    for i in range(1, n):
        y[i] = (b[i] - np.dot(L[i,:i], y[:i])) / L[i,i] #forward substitution formula

    return y

y_lu = forward_substitution(L, (P @ A_b))

#function for backward substitution on U and y
def back_substitution(U, y):

    #get number of rows
    n = U.shape[0]

    #allocate space for the solution vector y
    x = np.zeros_like(y, dtype=np.double);

    #initialize with the last row.
    x[-1] = y[-1] / U[-1, -1]

    #Looping over rows in reverse starting from second to last row bc the last row
    #was done right above
    for i in range(n-2, -1, -1):
        x[i] = (y[i] - np.dot(U[i,i:], x[i:])) / U[i,i] #formula for backward substitution

    return x
x_lu = back_substitution(U, y_lu)
x_lu
#end time for LU computation
end_time = time.time()

#solve without forward/backward substitution to check our results
coef_LU = np.linalg.inv(L @ U) @ (P @ A_b)
coef_LU #this array matches the x_lu array above

#find difference between our OLS coefficients and LU coefficients
#start time for OLS to compare
start_ols = time.time()
#OLS Coefficients of Model 1:
model11_coef = sm.OLS(y11.astype(float), x11.astype(float)).fit().params
model11_coef = model11_coef.to_numpy()
end_ols = time.time()

#reshape model11_coef to match x_lu to find difference
model11_coef = model11_coef.reshape(48, 1)

#difference between OLS coefficients and LU coefficients
```

```
model11_coef - x_lu

#get final time count for LU
end_time - start_time

#get final time count for OLS
end_ols - start_ols
```

(6) k-NN Regression

```
from sklearn.neighbors import KNeighborsRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.impute import SimpleImputer

imputer = SimpleImputer()
x = imputer.fit_transform(cars.drop(['Brand','Model', 'MSRP'], axis=1))
y = cars['Price']

# Split the dataset into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)

y_train = pd.to_numeric(y_train)
y_test = pd.to_numeric(y_test)

# Initialize the KNN regressor and fit the model on the training data
knn = KNeighborsRegressor(n_neighbors=5)
knn.fit(x_train, y_train)

# Predict the values for the testing set
y_pred = knn.predict(x_test)

# Calculate the mean squared error of the predictions
mse = mean_squared_error(y_test, y_pred)
print('Mean squared error:', mse)

# If we increase k = 50

# k-NN
from sklearn.neighbors import KNeighborsRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.impute import SimpleImputer

imputer = SimpleImputer()
x = imputer.fit_transform(cars.drop(['Brand','Model','MSRP'], axis=1))
y = cars['Price']

# Split the dataset into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)

y_train = pd.to_numeric(y_train)
y_test = pd.to_numeric(y_test)
```

```
# Initialize the KNN regressor and fit the model on the training data
knn = KNeighborsRegressor(n_neighbors=50)
knn.fit(x_train, y_train)

# Predict the values for the testing set
y_pred = knn.predict(x_test)

# Calculate the mean squared error of the predictions
mse = mean_squared_error(y_test, y_pred)
print('Mean squared error:', mse)

import matplotlib.pyplot as plt

# Plot the predicted vs. actual values
plt.scatter(y_test, y_pred, alpha=0.5)
plt.plot([0, 1750000], [0, 1750000], 'r--')
plt.xlabel('Actual Price')
plt.ylabel('Predicted Price')
plt.title('KNN Regression Results')
plt.show()

acura = cars[(cars['Brand'] == 'Acura')]

imputer = SimpleImputer()
x = imputer.fit_transform(acura.drop(['Brand','Model', 'MSRP'], axis=1))
y = acura['Price']

# Split the dataset into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)

y_train = pd.to_numeric(y_train)
y_test = pd.to_numeric(y_test)

# Initialize the KNN regressor and fit the model on the training data
knn = KNeighborsRegressor(n_neighbors=50)
knn.fit(x_train, y_train)

# Predict the values for the testing set
y_pred = knn.predict(x_test)

# Calculate the mean squared error of the predictions
mse = mean_squared_error(y_test, y_pred)
print('Mean squared error:', mse)

# Plot the predicted vs. actual values
plt.scatter(y_test, y_pred, alpha=0.5)
plt.plot([0, 250000], [0, 250000], 'r--')
plt.xlabel('Actual Price')
plt.ylabel('Predicted Price')
plt.title('KNN Regression Results for Acura')
plt.show()
```

(6) Simulation Study

```
from sklearn import linear_model
random.seed(19)

# Assigning true beta values to an array.
true_betas1 = model11.params
true_betas1 = true_betas1.to_frame().reset_index()
true_betas1.columns = ['Variables', 'Betas']

true_betas = np.asarray(true_betas1['Betas'])

# Create our simulation data frame X
sim_x = pd.DataFrame(np.random.randint(0, 100, size = (100000, 47)), columns = list('ABCDEFGHIJKLMN
sim_x.insert(47, 'Intercept', 1)

# Our error term for simulation.
sim_epsilon = np.random.randn(48)

# Simulation Y made with the true beta values.
sim_y = np.asarray((sim_x * true_betas) + sim_epsilon)
sim_x.to_numpy()
display(sim_x)

# OLS regression for simulation data.
regr = linear_model.LinearRegression()
regr.fit(sim_x, sim_y)

sim_beta = regr.coef_.tolist()
sim_intercept = regr.intercept_.tolist()
display(sim_beta)
display(sim_intercept)
```