FLIP ROBO

CAR PRICE PREDICTION

Submitted by:

PURNIMA PATI

ACKNOWLEDGMENT

With the covid 19 impact in the market, we have seen lot of changes in the car market. Now some cars are in demand hence making them costly and some are not in demand hence cheaper. One of our clients works with small traders, who sell used cars. With the change in market due to covid 19 impact, our client is facing problems with their previous car price valuation machine learning models. So, they are looking for new machine learning models from new data.

The data is being scraped from www.cars.com.
This is a well-structured source for used car listings and is generous to scrapers.

To complete this project I have gone through various case studies and project how used car industry is performing post covid19 .I have gone through some articles available in google which are from ET , CNBC etc.

INTRODUCTION

- Business Problem Framing

In this article, I will analyze the used car dataset. This analysis will tell us various features that are responsible for the price of a used car. With this analysis, I will predict the price of a used car. The prices of new cars in the industry is fixed by the manufacturer with some additional costs incurred by the Government in the form of taxes. So, customers buying a new car can be assured of the money they invest to be worthy. But due to the increased price of new cars and the incapability of customers to buy new cars due to the lack of funds, used cars sales are on a global increase. There is a need for a used car price prediction system to effectively determine the worthiness of the car using a variety of features. Even though there are websites that offers this service, their prediction method may not be the best. Besides, different models and systems may contribute on predicting power for a used car's actual market value. It is important to know their actual market value while both buying and selling.

- Conceptual Background of the Domain Problem
  The price of a pre-owned car depends on various factors including model year, mileage, condition, equipment, etc. Since the price depends on so many factors, it is difficult to estimate it directly using rule-based algorithms. A more feasible strategy is to use inductive based learning to learn the price from the dataset. Hence, a machine learning approach is very suitable for this application.
  The key contribution in this article lies in collecting a real dataset pertaining to pre-owned cars and then preparing the dataset extensively, which can be used in developing an accurate predictive model. In addition to this, other key contributions include assessing how well the model predicts by comparing the predicted output on unseen data and deploying the model so that it can be used in the future by end users.
- Review of Literature

- ✓ First we have to scrap the data from the website , I scraped data from www.cars.com
- ✓ Drop duplicates rows if present in dataset.Then we check for the null values present in our dataset.
- ✓ If null values are present then fill it via mean, median or mode. Or also you can remove that rows but kindly check it properly.

- ✓ For model creation we need to replace –ve values
- ✓ After extracting the data we have to put it in data frame
- ✓ We have to encode the categorical features before we can fit a model. There are multiple ways we can do this. We will use the "one-hot encoding" method and simply dummify such features (this creates a new column for each category of each categorical column). Each of these 'dummy' columns will hold a value of 0 or 1.
- ✓ Note that the number of columns went up drastically because of the encoding.
- ✓ We will now raise the polynomial degree and compute feature interactions
- ✓ Number of columns just exploded to such a high value. This is because we raised the polynomial feature and computed every single feature interaction. We will bring this down to a reasonable number a little later.
- ✓ We will now add the new feature names and convert it back to a data frame
- ✓ We're finally ready to begin modeling (we are going to score on the training set for now). Let's test out some models before we get to reducing the absurd number of features we're dealing with:
- ✓ For modeling we will check regression method , Lasso & Ridge
- ✓ We can look at the model coefficients to get the features that are truly making a difference
- ✓ Look at the residuals will tell us how the model is doing for a range of target prices.

- Motivation for the Problem Undertaken

  There is a need for a used car price prediction system to effectively determine the worthiness of the car using a variety of features. Even though there are websites that offers this service, their prediction method may not be the best. Besides, different models and systems may contribute on predicting power for a used car’s actual market value. It is important to know their actual market value while both buying and selling.

The price of a new car in the industry is fixed by the manufacturer with some additional costs incurred by the Government in the form of taxes. So, customers buying a new car can be assured of the money they invest to be worthy. But, due to the increased prices of new cars and the financial incapability of the customers to buy them, Used Car sales are on a global increase. Therefore, there is an urgent need for a Used Car Price Prediction system which effectively determines the worthiness of the car using a variety of features. Existing System includes a process where a seller decides a price randomly and buyer has no idea about the car and it’s value in the present day scenario. In fact, seller also has no idea about the car’s existing value or the price he should be selling the car at. To overcome this problem we have developed a model which will be highly effective. Regression Algorithms are used because they provide us with continuous value as an output and not a categorized value. Because of which it will be possible to predict the actual price a car rather than the price range of a car. User Interface has also been developed which acquires input from any user and displays the Price of a car according to user’s inputs.

Analytical Problem Framing

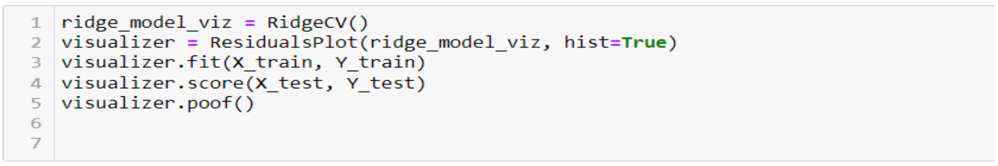
- Mathematical/ Analytical Modeling of the Problem
    - ➢ We have to encode the categorical features before we can fit a model. There are multiple ways we can do this. We will use the "one-hot encoding" method and simply dummify such features (this creates a new column for each category of each categorical column). Each of these 'dummy' columns will hold a value of 0 or 1.
    - ➢ Note that the number of columns went up drastically because of the encoding.

- ➢ We will now raise the polynomial degree and compute feature interactions
- ➢ Number of columns just exploded to such a high value. This is because we raised the polynomial feature and computed every single feature interaction. We will bring this down to a reasonable number a little later.
- ➢ We will now add the new feature names and convert it back to a data frame
- ➢ We're finally ready to begin modeling (we are going to score on the training set for now). Let's test out some models before we get to reducing the absurd number of features we're dealing with:
- ➢ For modeling we will check regression method , Lasso & Ridge
- ➢ We can look at the model coefficients to get the features that are truly making a difference
- ➢ Look at the residuals will tell us how the model is doing for a range of target prices

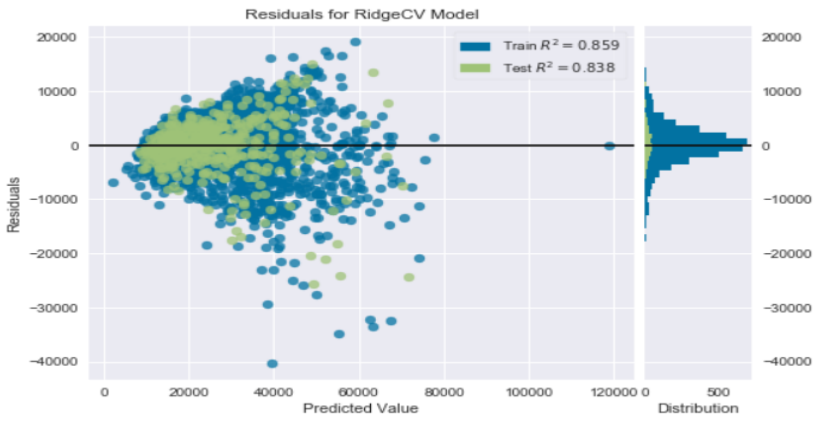
- Data Sources and their formats
- ✓ The data is being scraped from www.cars.com
- ✓ This is a well structured source for used car listings and is generous to scrapers
- ✓ Check www.cars.com/robots.txt for more information on what the website allows
  the website is hardcoded to give us only 50 pages and a maximum of 100 listings per page. To circumvent this, we'll have to use filters independent of each other to get the listings we can't access traditionally. That part is beyond the scope of the project at the moment, so let's limit our search to the first 5000 listings of sedans.
- Data Preprocessing Done
  DATA TRANSFORMATION: Before we get into implementing models, we'll have to first make sure all the data is in the right format and datatypes. Thus, we begin the no-so-fun process of data transformations.
  - ✓ Create a new df containing the final features
  - ✓ Extracting Year and converting it to the right format
  - ✓ Get brand names
  - ✓ Format miles and prices to integers
  - ✓ Adding the rest of the features that don't need formatting/transformation

  - ✓ Converting 'NaN' strings to NoneType
- Data Inputs- Logic- Output Relationships
  We will use;
  sns.pairplot(df_final.dropna(), height=2.5, aspect=1.5);
  But there are no clear linear relationships. This tells us that although the features themselves are important (by logic/domain knowledge) we have to look into polynomial transformations and/or feature interactions to make effective use of them. We have to encode the categorical features before we can fit a model. There are multiple ways we can do this. We will use the "one-hot encoding" method and simply dummify such features (this creates a new column for each category of each categorical column). Each of these 'dummy' columns will hold a value of 0 or 1.

- State the set of assumptions (if any) related to the problem under consideration



```
ridge_model_viz = RidgeCV()
visualizer = ResidualsPlot(ridge_model_viz, hist=True)
visualizer.fit(X_train, Y_train)
visualizer.score(X_test, Y_test)
visualizer.poof()
```



Although in theory, heteroscedasticity is not a good sign, in practice it is almost always visible. In the above plot, for good predictions, we must ensure that most of the data points are as close as possible to the zero line. As we can see, the greatest density of points is around the zero line and the spread increases as we move towards the more expensive cars. This is because the original dataset has fewer expensive cars as we move up in price and hence fewer data points to train the model with. In such cases, the model is expected to do poorly in such areas.

- Hardware and Software Requirements and Tools Used

Hardware requirements

Operating system- Windows 10

Processor- dual core 2.4 GHz (i5 series Intel processor or equivalent AMD)

RAM-4GB

Software Requirements

Python

PIP 2.7

Jupyter Notebook

Chrome

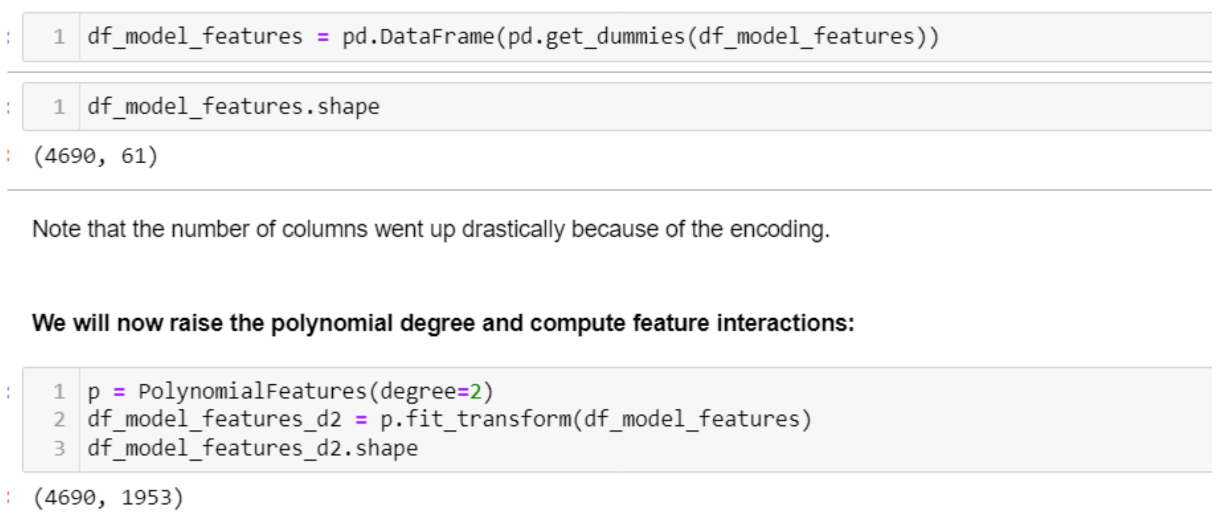
Model/s Development and Evaluation

- Identification of possible problem-solving approaches

  The first step is data collection through scrapping .The next step is to do Data Preprocessing which includes Data cleaning, Data reduction, Data Transformation. Then, using various machine learning algorithms we will predict the price. The algorithms involve Linear Regression, Ridge Regression and Lasso Regression. The best model which predicts the most accurate price is selected. After selection of the best model the predicted price is displayed to the user according to user's inputs. User can give input through website to for used car price prediction to machine learning model. Linear Regression Linear Regression attempt to model the relationship between two variables by fitting a linear equation to observed data. The other is considered to be dependent variable.
  Ridge Regression A Ridge regressor is basically a regularized version of Linear Regressor. The regularized term has the parameter 'alpha' which controls the regularization of the model i.e helps in reducing the variance of the estimates. The "LASSO" stands for Least Absolute Shrinkage and Selection Operator. Lasso regression is a regularization technique. It is used over regression methods for a more accurate prediction. This model uses shrinkage. Shrinkage is where data values are shrunk towards a central point as the mean. The lasso procedure encourages simple, sparse models (i.e. models with fewer parameters). This particular type of regression is well-suited for models showing high levels of multicollinearity or when you want to automate certain parts of model selection, like variable selection/parameter elimination

- Testing of Identified Approaches (Algorithms)
  Basic train/test split to see if the performance is consistent (though LassoCV/RidgeCV employs cross-validation, it doesn't hurt to have a holdout set)

- Run and Evaluate selected models



```
df_model_features = pd.DataFrame(pd.get_dummies(df_model_features))
```

```
df_model_features.shape
```

```
(4690, 61)
```

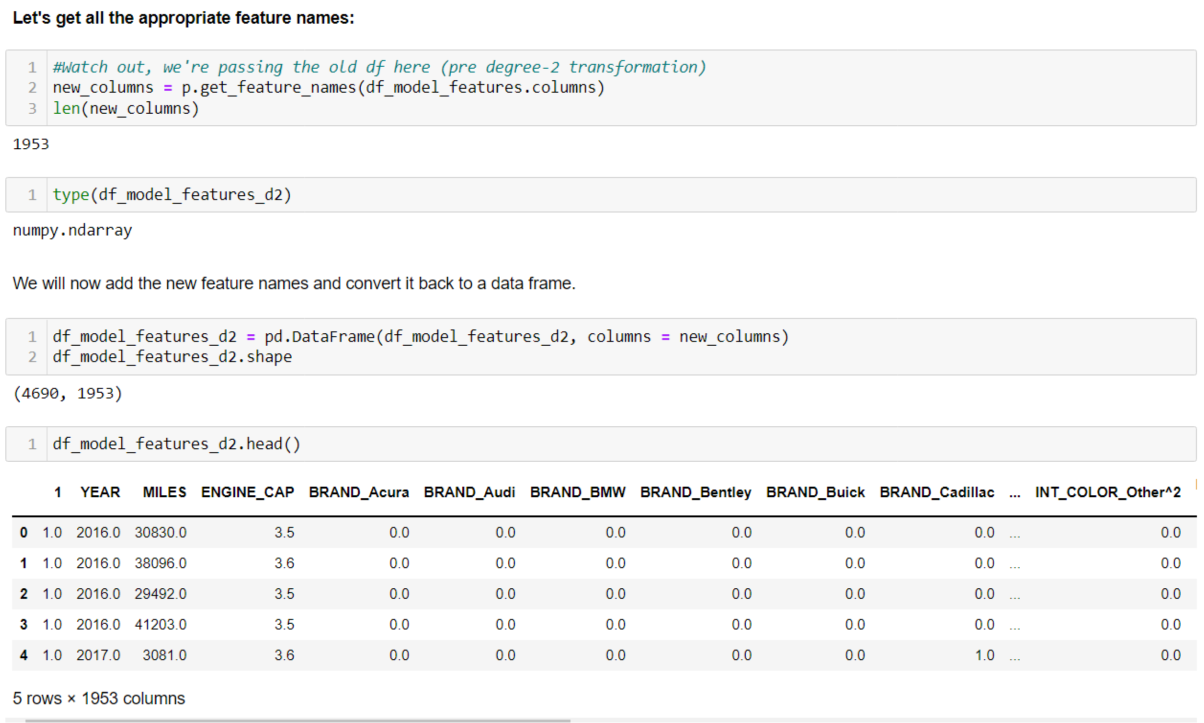
Note that the number of columns went up drastically because of the encoding.

We will now raise the polynomial degree and compute feature interactions:

```
p = PolynomialFeatures(degree=2)
df_model_features_d2 = p.fit_transform(df_model_features)
df_model_features_d2.shape
```

```
(4690, 1953)
```

the number of columns just exploded to such a high value. This is because we raised the polynomial feature and computed every single feature interaction. We will bring this down to a reasonable number a little later.



Let's get all the appropriate feature names:

```
#Watch out, we're passing the old df here (pre degree-2 transformation)
new_columns = p.get_feature_names(df_model_features.columns)
len(new_columns)
```

```
1953
```

```
type(df_model_features_d2)
```

```
numpy.ndarray
```

We will now add the new feature names and convert it back to a data frame.

```
df_model_features_d2 = pd.DataFrame(df_model_features_d2, columns = new_columns)
df_model_features_d2.shape
```

```
(4690, 1953)
```

```
df_model_features_d2.head()
```

	1	YEAR	MILES	ENGINE_CAP	BRAND_Acura	BRAND_Audi	BRAND_BMW	BRAND_Bentley	BRAND_Buick	BRAND_Cadillac	...	INT_COLOR_Other^2
0	1.0	2016.0	30830.0	3.5	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0
1	1.0	2016.0	38096.0	3.6	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0
2	1.0	2016.0	29492.0	3.5	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0
3	1.0	2016.0	41203.0	3.5	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0
4	1.0	2017.0	3081.0	3.6	0.0	0.0	0.0	0.0	0.0	1.0	...	0.0

5 rows × 1953 columns

Although we may continue to fitting a model at this point, it is advised to scale our features before doing so

```
#Scaling features first
std = StandardScaler()
df_model_features_d2 = pd.DataFrame(std.fit_transform(df_model_features_d2), columns=new_columns)
```

We're finally ready to begin modeling (we are going to score on the training set for now). Let's test out some models before we get to reducing the absurd number of features we're dealing with:

```
m = LinearRegression()
m.fit(df_model_features_d2, df_model_target)
m.score(df_model_features_d2, df_model_target)
```

```
0.8828726004825977
```

```
lasso_model = LassoCV()
lasso_model.fit(df_model_features_d2, df_model_target)
lasso_model.score(df_model_features_d2, df_model_target)
```

```
0.8526405477675314
```

```
ridge_model = RidgeCV()
ridge_model.fit(df_model_features_d2, df_model_target)
ridge_model.score(df_model_features_d2, df_model_target)
```

```
0.8671477708462859
```

We can look at the model coefficients to get the features that are truly making a difference:

```
#required_columns contains the indices of the columns with non-zero coefficients
i = 0
required_columns = []
for index, val in enumerate(lasso_model.coef_):
    if val!=0.0:
        #print('Index:%s, Value:%s'%(index, val))
        required_columns.append(index)
        i+=1
print('There are',i,'required columns.')
```

```
There are 325 required columns.
```

Now that we have the features we need, let's get rid of the rest:

```
#Filtering out the other columns
df_model_features_d2_final = df_model_features_d2[df_model_features_d2.columns[required_columns]]
```

```
df_model_features_d2_final.head()
```

	YEAR	BRAND_Bentley	BRAND_Cadillac	BRAND_Lexus	BRAND_Lincoln	BRAND_Maserati	BRAND_Mitsubishi	BRAND_Nissan	BRAND_Toyota	BRAND_
0	-0.180477	-0.014604	-0.198621	3.676972	-0.179249	-0.084179	-0.014604	-0.317709	-0.191555	-0.
1	-0.180477	-0.014604	-0.198621	-0.271963	-0.179249	-0.084179	-0.014604	-0.317709	-0.191555	-0.
2	-0.180477	-0.014604	-0.198621	3.676972	-0.179249	-0.084179	-0.014604	-0.317709	-0.191555	-0.
3	-0.180477	-0.014604	-0.198621	3.676972	-0.179249	-0.084179	-0.014604	-0.317709	-0.191555	-0.
4	0.671929	-0.014604	5.034711	-0.271963	-0.179249	-0.084179	-0.014604	-0.317709	-0.191555	-0.

5 rows × 325 columns

It may look like the data doesn't make sense, but that's because we have scaled it for consistency.

```
lasso_temp = LassoCV()
lasso_temp.fit(df_model_features_d2_final, df_model_target)
lasso_temp.score(df_model_features_d2_final, df_model_target)
```

```
0.8574714725962475
```

Let's do a basic train/test split to see if the performance is consistent (though LassoCV/RidgeCV employs cross-validation, it doesn't hurt to have a holdout set):

```
X_train, X_test, Y_train, Y_test = train_test_split(df_model_features_d2_final, df_model_target
                                                    , test_size=0.1, random_state=42)
```

```
print(X_train.shape)
print(X_test.shape)
print(Y_train.shape)
print(Y_test.shape)
```

```
(4221, 325)
(469, 325)
(4221,)
(469,)
```

```
m_2 = LinearRegression()
m_2.fit(X_train, Y_train)
m_2.score(X_train, Y_train)
```

```
0.870849432206049
```

```
lasso_model_2 = LassoCV()
lasso_model_2.fit(X_train, Y_train)
lasso_model_2.score(X_test, Y_test)
```

```
0.8351887629586059
```

```
ridge_model_2 = RidgeCV()
ridge_model_2.fit(X_train, Y_train)
ridge_model_2.score(X_test, Y_test)
```

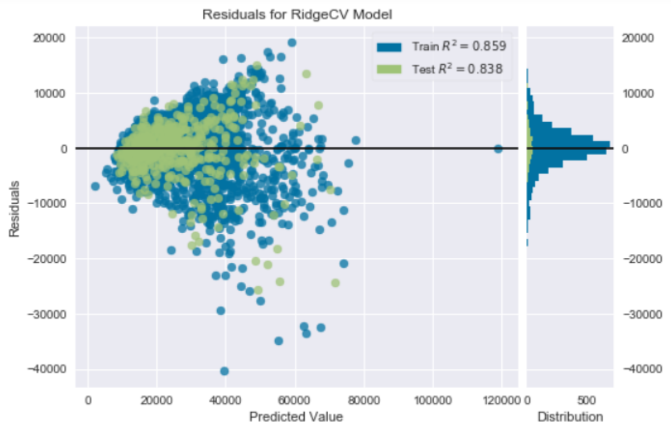
```
0.8375739909489468
```

The score is expected to fall when we score on test, but the scores look more or less consistent. We can now move on to visualize our results.

Now, let's take a moment to interpret our results and the model performance:

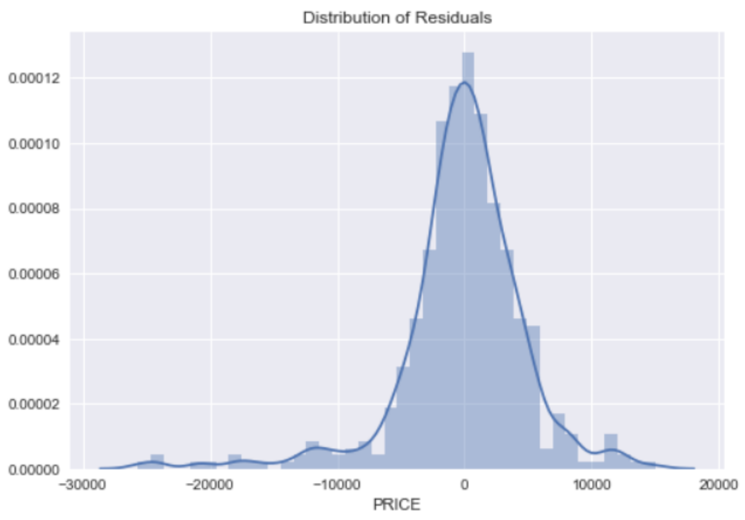
A look at the residuals will tell us how the model is doing for a range of target prices:

```
ridge_model_viz = RidgeCV()
visualizer = ResidualsPlot(ridge_model_viz, hist=True)
visualizer.fit(X_train, Y_train)
visualizer.score(X_test, Y_test)
visualizer.poof()

```



Although in theory, heteroscedasticity is not a good sign, in practice it is almost always visible. In the above plot, for good predictions, we must ensure that most of the data points are as close as possible to the zero line. As we can see, the greatest density of points is around the zero line and the spread increases as we move towards the more expensive cars. This is because the original dataset has fewer expensive cars as we move up in price and hence fewer data points to train the model with. In such cases, the model is expected to do poorly in such areas.

```python
sns.distplot((ridge_model_2.predict(X_test)-Y_test), bins=40).set_title('Distribution of Residuals');
```



Features that drive (pun intended) up the price the most:

Features that drive (pun intended) up the price the most:

```python
#Analyzing most important features
pd.DataFrame({'Column_Name': new_columns
              , 'Coefficient': lasso_model.coef_}).sort_values(['Coefficient'], ascending=False).head(5)
```

	Column_Name	Coefficient
183	ENGINE_CAP^2	4775.300741
202	ENGINE_CAP BRAND_Mercedes-Benz	4599.222229
186	ENGINE_CAP BRAND_BMW	4261.374407
1	YEAR	2407.065822
20	BRAND_Maserati	1428.566030

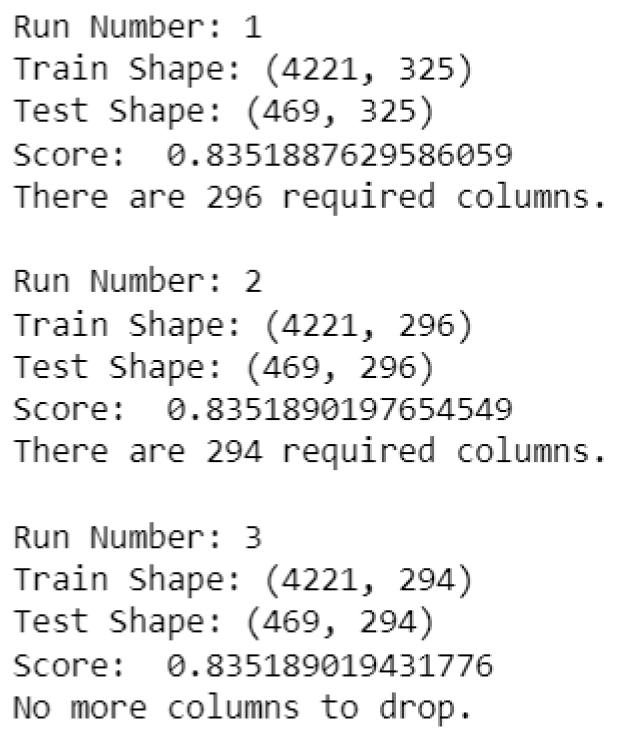
As expected, the more powerful the car, the more expensive it is. Some of the other features that drive the price up are if it is a powerful luxury car, if the manufacture year is more recent or if it is a sports car brand like Maserti.

Similarly, the features that drive the price down:

```python
pd.DataFrame({'Column_Name': new_columns
              , 'Coefficient': lasso_model.coef_}).sort_values(['Coefficient'], ascending=True).head(5)
```

	Column_Name	Coefficient
35	DRIVETRAIN_FWD	-2464.821136
143	MILES BRAND_Mercedes-Benz	-1736.885414
127	MILES BRAND_BMW	-1479.714544
124	MILES ENGINE_CAP	-1373.612307
192	ENGINE_CAP BRAND_Dodge	-1272.762751

```python
#Don't bother with this yet
xtr = X_train.copy(deep=True)
xte = X_test.copy(deep=True)
counter = 0
no_cols = 1
prev_no_cols = 0
while no_cols != 0:
    counter += 1
    print('Run Number:', counter)
    print('Train Shape:', xtr.shape)
    print('Test Shape:', xte.shape)
    las_m = LassoCV()
    las_m.fit(xtr, Y_train)
    print('Score: ', las_m.score(xte, Y_test))
    no_cols = 0
    req_columns = []
    for index, val in enumerate(las_m.coef_):
        if val!=0.0:
            #print('Index:%s, Value:%s'%(index, val))
            req_columns.append(index)
            no_cols+=1
    if no_cols == prev_no_cols:
        print('No more columns to drop.')
        break
    prev_no_cols = no_cols
    print('There are',no_cols,'required columns.\n')
    xtr = xtr[xtr.columns[req_columns]]
    xte = xte[xte.columns[req_columns]]
```



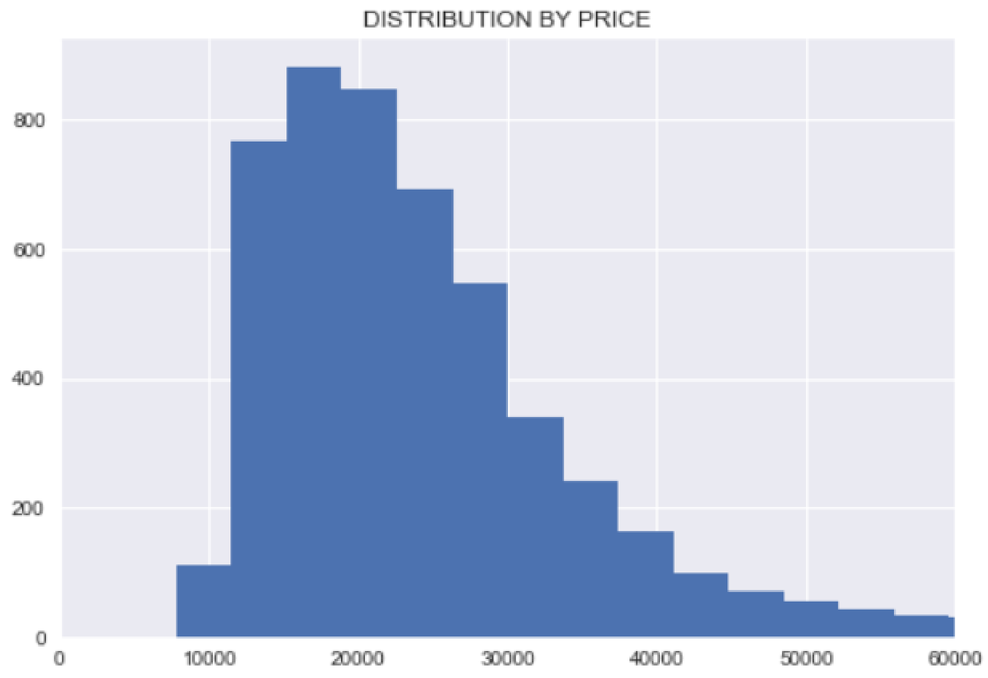
```
Run Number: 1
Train Shape: (4221, 325)
Test Shape: (469, 325)
Score:  0.8351887629586059
There are 296 required columns.

Run Number: 2
Train Shape: (4221, 296)
Test Shape: (469, 296)
Score:  0.8351890197654549
There are 294 required columns.

Run Number: 3
Train Shape: (4221, 294)
Test Shape: (469, 294)
Score:  0.835189019431776
No more columns to drop.
```

- Key Metrics for success in solving problem under consideration
  At the last stage, predictive models were applied to predict price of cars in an order: **linear regression, ridge regression, lasso**. By considering all four metrics from table 15, it can be concluded that **linear regression** is the best model for the prediction for used car prices.

- Visualizations

A look at the distribution of cars in the dataset by price:

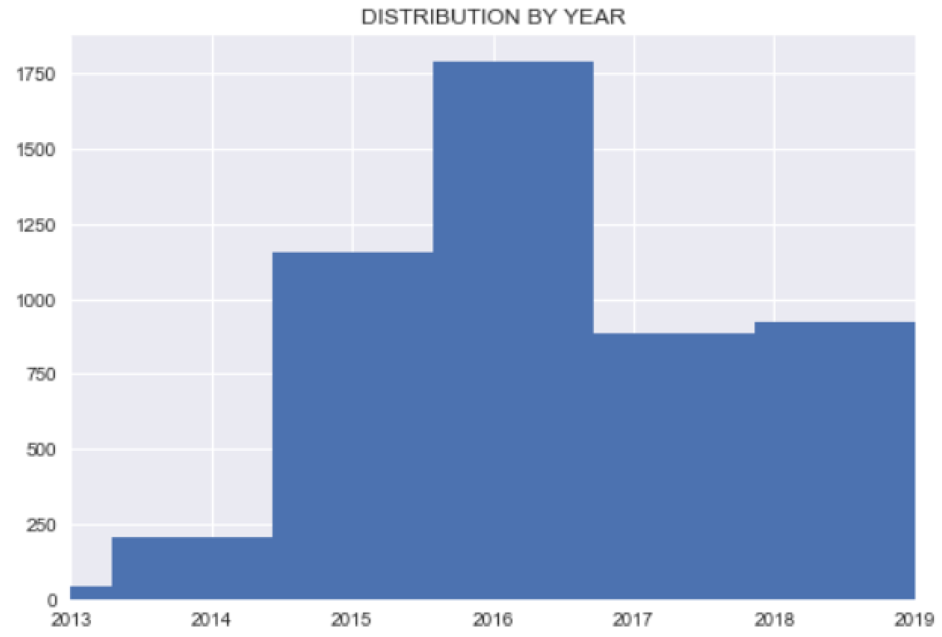
```python
#temp = df_final[(df_final['PRICE']>7000) & (df_final['PRICE']<60000)]
plt.hist(df_final['PRICE'], bins=30)
plt.xlim(0,60000)
plt.title('DISTRIBUTION BY PRICE');
```



Most cars fall under the $15,000 - $30,000 price range.

A look at the distribution of cars by year of manufacture:

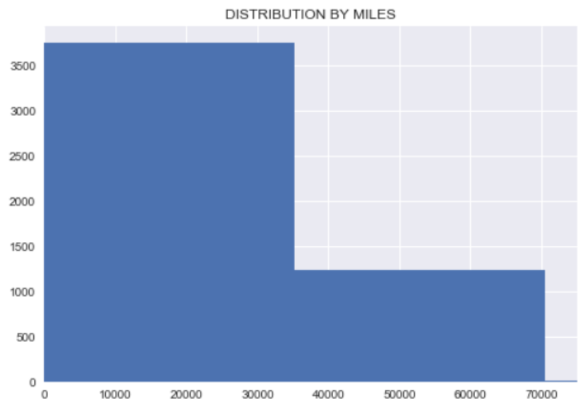
```
#temp = df_final[(df_final['PRICE']>7000) & (df_final['PRICE']<60000)]
plt.hist(df_final['YEAR'], bins = 7)
plt.xlim(2013,2019)
plt.title('DISTRIBUTION BY YEAR');
```



Most cars being sold are roughly 3 years old.

A look at the distribution of cars by the number of miles on them:

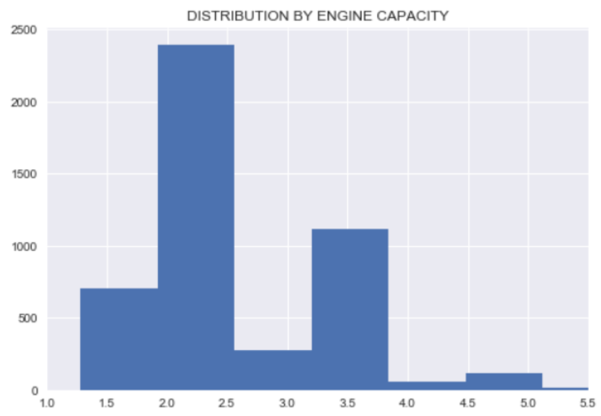
```
#temp = df_final[(df_final['PRICE']>7000) & (df_final['PRICE']<60000)]
plt.hist(df_final['MILES'])
plt.xlim(0,75000)
plt.title('DISTRIBUTION BY MILES');
```



It is interesting to note that there is a sharp drop in the number of cars with miles greater than around 30,000. Upon further investigation, I found that most car leases last 3 years with around 10,000 miles per year and then end up on the used car market after that.

A look at the distribution of cars by engine capacity:

```python
#temp = df_final[(df_final['PRICE']>7000) & (df_final['PRICE']<60000)]
plt.hist((df_final.dropna()).ENGINE_CAP)
plt.xlim(1,5.5)
plt.title('DISTRIBUTION BY ENGINE CAPACITY');
```



It is interesting to see two spikes in the above plot. This is because, for the most part, cars are divided into the commuter and the luxury types. The commuter cars come with a similar engine capacity range and the luxury cars follow the same trend (but are more powerful, hence the second spike).

- Interpretation of the Results

By performing different models, it was aimed to get different perspectives and eventually compared their performance. With this study, it purpose was to predict prices of used cars by using a dataset which we got after scraping data from website. With the help of the data visualizations and exploratory data analysis, the dataset was uncovered and features were explored deeply. The relation between features were examined. At the last stage, predictive models were applied to predict price of cars in an order: linear regression, ridge regression, lasso.

By considering all 3 metrics, it can be concluded that linear regression is the best model for the prediction for used car prices.

CONCLUSION

- Key Findings and Conclusions of the Study

  The increased prices of new cars and the financial incapability of the customers to buy them, Used Car sales are on a global increase. Therefore, there is an urgent need for a Used Car Price Prediction system which effectively determines the worthiness of the car using a variety of features. The proposed system will help to determine the accurate price of used car price prediction. This paper compares 3 different algorithms for machine learning : Linear Regression, Lasso Regression and Ridge Regression.

- Learning Outcomes of the Study in respect of Data Science

  The "LASSO" stands for Least Absolute Shrinkage and Selection Operator. Lasso regression is a regularization technique. It is used over regression methods for a more accurate prediction. This model uses shrinkage. Shrinkage is where data values are shrunk towards a central point as the mean. The lasso procedure encourages simple, sparse models (i.e. models with fewer parameters). This particular type of regression is well-suited for models showing high levels of multicollinearity or when you want to automate certain parts of model selection, like variable selection/parameter elimination

- Limitations of this work and Scope for Future Work
  In future this machine learning model may bind with various website which can provide real time data for price prediction. Also we may add large historical data of car price which can help to improve accuracy of the machine learning model. We can build an android app as user interface for interacting with user. For better performance, we plan to judiciously design deep learning network structures, use adaptive learning rates and train on clusters of data rather than the whole dataset
    - ✓ Try other models and tune using hyperparameters
    - ✓ Build out the extensive dataset using independent filters and expand to include other automobile types
    - ✓ Include data from sources other than www.cars.com
    - ✓ Perform in-depth outlier analysis