Title: 64,000 images from Car Connection Dataset

Author: Usman Basharat

Date: 14th April 2020

Course: COMP-1804 Applied Machine Learning

Word Count: 3104

School of Computing and Mathematical Sciences

UNIVERSITY of GREENWICH

Table of Contents

Stage 1: Introduction

Examine Dataset

A software consultancy company are seeking to adapt and shift their business into the era of Machine Learning. Executives state they would like to draw meaning from this approach by accumulating big datasets. Therefore, they require to discover insight and draw conclusion on the dataset that has been given. Kaggle has introduced a massive 64,000 images of cars naming this The Car Connection Picture Dataset by Nicolas Gervais. As mentioned, this dataset was introduced by Nicolas Garvis. He stated that these pictures are scraped data. He collected originally around 297,000 images. However, many of the interior images were unless. Hence, he had to cut these images down to around 64,000.

This whole dataset contains numerous images of the interior, exterior of different cars with different models and years. This will be explained in detail later on in this report. As for any dataset, this requires a numerous approach of cleansing the data. Any unnecessary data will be extracted from the dataset. Any rows that contains more than four empty columns will be removed. Data will be extracted from all the image labels. Therefore, an approach would be get all this data and convert this into an excel file. Once the data has been cleansed, I will test this data using three different machine learning models. Different approaches to explore different topics can be approached by:

- Building a machine learning model by using the attributes to classify cars B
- Building a machine learning model to predict a car's price by using its attributes.
- Apply deep learning techniques to utilise the images in use. A benchmark model can be used to classify the unseen testing data

These approaches would need to be considered when approaching the dataset. Some of these suggested questions have had a good approach to exploring different options.

I have decided to go with building a machine model by predicting price using a machine learning models using factors. Within this approach, they are different stages that need to be completed in order to get the full analysis on this matter. Therefore, they are five stages that need to be questioned and they are the following:

- Stage 1 - Examine datasets and identify the key area of potential insight for clients
- Stage 2 - Data Pre-processing
- Stage 3 – Build Model
- Stage 4 – Testing and Validation
- Stage 5 – Visualisation

Each of these stages, you will see further analysis on the car dataset. These stages will determine the outcome of predicting the price for each of the results. I will be using various machine learning techniques. However, this will be further analysed during within the next stage. Please refer yourself over to Table 1 where this identifies which stages have been achieved.

Existing Research

Many questions have been suggested above within the Introduction. However, I will explore the dataset by identifying potential insight for clients in detail. Previously, I mentioned that I will be exploring the possibility of using this car dataset by predicting the price of the car. I will be exploring this further by using three different machine learning algorithms to predict the price of the cars that have been mentioned.

Enis Gegic does similar research in predicting car prices using machine learning techniques. He states that, *"Accurate car price prediction involves expert knowledge, because price usually depends on many distinctive features and factors"*. (Gegic, Isakovic, Keco, Masetic and Kevric 2019). The most valuable outcome of this key area is the potential depends on the accuracy of the prediction. Therefore, when considering any possibility of prediction; the factors and features that have been tested are the most important. Mr Gegic considers three machine learning algorithms to access the accuracy of the results. These are Artificial Neural Networks (ANN), Support Vector Machine (SVM) and lastly Random Forest. He obtained ranging results from 80% to 90%. Therefore, it selected two of the best results and did further analysis on this to make better accuracy of the prediction. Therefore, I will be using the same template by selecting three algorithms and doing further analysis on this matter.

Mr Pudaruth and also Mr Gegic, both mention in their papers, that the factors that value the most when predicting the price of the car is the age of the car (year), the make of the car (the model), origin of the car (make), the horsepower and the gas mileage. Other factors will contribute to its price, but it would not have a significant impact as these five that have been mentioned. In addition, both papers have similar machine learning algorithms. All of these factors that have been mentioned in both papers will be taken into consideration as this plays a huge part within predicting prices on cars (Pudaruth, 2014). Based upon the research that I have come across, I will be using three algorithms of Random Forest, Decision Trees and Gradient Boosting Regression

Stages Achieved

Table 1 shows which tasks have been achieved.

Stage	Task	Achieved
1.	Examine dataset and identify potential insight for client	Yes
2.	Data Analysis	Yes
3.	Build Model	Yes
4.	Testing and Validating	Yes
5.	Visualization	Yes

Stage 2: Data Analysis

The dataset will require processing before building and testing the dataset. As for any dataset, and as Nicolas mentioned, this is 'scrap' data that has been collected. Therefore, it needs to be cleansed before taking any measurements. The first step that I did is download this huge 681MB full of images and stored this so I can access this. Referring to Figure 1, my first aim was to list all of the images that have been downloaded. Therefore, this shows all of the 64,000 images.

```
import os
dir = 'C:/Users/usman/Documents/picture'
os.listdir(dir)
```

```
['Acura_ILX_2013_28_16_110_15_4_70_55_179_39_FWD_5_4_4dr_aWg.jpg',
 'Acura_ILX_2013_28_16_110_15_4_70_55_179_39_FWD_5_4_4dr_Bbw.jpg',
 'Acura_ILX_2013_28_16_110_15_4_70_55_179_39_FWD_5_4_4dr_Cvl.jpg',
 'Acura_ILX_2013_28_16_110_15_4_70_55_179_39_FWD_5_4_4dr_eeu.jpg',
```

Figure 1 shows the first step of listing all the images.

As you may notice, this is the first step of acknowledging the data. This data needs to be cleansed, therefore, you may notice unnecessary data that has been highlighted at the end. This was at the end of each of the 64,000 images that has been stored. Therefore, in order to get rid of this, it needs to be a loop. I used a *for each* loop referring it back to this array. You can see that I got rid of the last 8 of the rows by storing it in another array. You can see the following statement below that was used:

array1 = []
for i in array:
array1.append(i[:-8])

As soon as this was complete, I noticed after each word, there is an unnecessary underscore that does not need to be there. This was the same for each of the 64,000 images that had an underscore next to each word. Therefore, this was the same process that needed to be complete by putting this into another array. You can see that I used the same append by splitting the underscore. You can see the following statement below that was used:

array_two = []
for i in array1:
array_two.append(i.split('_'))

Referring to Figure 2, you may see the result of what has been obtained. This is far clearer and understandable for each column. Once this was complete, I was able to convert this data into a readable excel file. This would be easier to access, rather than keep referring to the array list above.

```
[['Acura',
  'ILX',
  '2013',
  '28',
  '16',
  '110',
  '15',
  '4',
  '70',
  '55',
  '179',
  '39',
  'FWD',
  '5',
  '4',
  '4dr'],
```

Figure 2 results after the first step of data processing

As soon as this was done, the excel file was created by simply importing the CSV file and writing to the file the array that has been created above. I needed to direct the CSV file that has been created. You may notice that has been done using the following statement:

columns = ['Make', 'Model', 'Year', 'MSRP', 'Front Wheel Size (in)', 'SAE Net Horsepower @ RPM', 'Displacement', 'Engine Type', 'Width, Max w/o mirrors (in)', 'Height, Overall (in)', 'Length, Overall (in)', 'Gas Mileage', 'Drivetrain', 'Passenger Capacity', 'Passenger Doors', 'Body Style'];
ds = pd.read_csv('C:/Users/usman/Documents/dataset.csv', names=columns)

```
empty_cols = ds[columns].isnull().sum()
print(empty_cols) #finds all empty columns
```

```
Make                              0
Model                             0
Year                              0
MSRP                            794
Front Wheel Size (in)           735
SAE Net Horsepower @ RPM        824
Displacement                   2210
Engine Type                    2322
Width, Max w/o mirrors (in)     792
Height, Overall (in)            768
Length, Overall (in)           6609
Gas Mileage                    6973
Drivetrain                      733
Passenger Capacity              727
Passenger Doors                 727
Body Style                      733
dtype: int64
```

Figure 3 shows all of the empty rows that are within the dataset.

Referring to Figure 3, this shows all of the empty rows that have been found within the dataset. As you can see, they are many that were found. However, I noticed that throughout these empty rows; I noticed many of the rows contained more than 4 columns. Many of them were empty from Front Wheel Size to Body Style. Therefore, I did a simple statement where this finds all of the empty rows and drops all of these that have been stated. I searched through empty rows using the statement that has been shown within Figure 3. I dropped these columns using the *all* statement that is shown below.

df.dropna(subset = ['selected columns'], how = 'all')

Referring to Figure 4, this shows the result after finding all of these empty rows that contained no data and it was in no use. You can compare this result to Figure 3 where this is a huge difference. Please bear in mind that some of the update statements were used above. However, this is roughly the estimate of what has been resulted.

```
Make                              0
Model                             0
Year                              0
MSRP                             31
Front Wheel Size (in)             0
SAE Net Horsepower @ RPM         89
Displacement                   1268
Engine Type                    1304
Width, Max w/o mirrors (in)       1
Height, Overall (in)              1
Length, Overall (in)           5541
Gas Mileage                    5112
Drivetrain                        0
Passenger Capacity                0
Passenger Doors                   0
Body Style                        0
```

Figure 4 shows the result of the dropping empty rows that contained no data

Referring to Figure 5, this shows the data that was individually updated. You can see that Gas Mileage for Acura does not contain anything. Therefore, a rough estimate was done using the following query:

ds.update(ds.iloc[256:266, 0:15].fillna(result))

Using the statement above, this was a routine for every column that was shown as empty for Figure 4. A rough estimate was figured and even researched if the answer was not possible. Therefore, more data to deal with by getting better results.

```
ds.iloc[256:266, 0:15]
```

	Make	Model	Year	MSRP	Front Wheel Size (in)	SAE Net Horsepower @ RPM	Displacement	Engine Type	Width, Max w/o mirrors (in)	Height, Overall (in)	Length, Overall (in)	Gas Mileage	Drivetrain	Passenger Capacity	Passenger Doors
256	Acura	MDX	2013	43.0	18.0	300	37.0	6.0	78	68	191	NaN	AWD	7.0	4.0
257	Acura	MDX	2013	43.0	18.0	300	37.0	6.0	78	68	191	NaN	AWD	7.0	4.0
258	Acura	MDX	2013	43.0	18.0	300	37.0	6.0	78	68	191	NaN	AWD	7.0	4.0
259	Acura	MDX	2013	43.0	18.0	300	37.0	6.0	78	68	191	NaN	AWD	7.0	4.0
260	Acura	MDX	2013	43.0	18.0	300	37.0	6.0	78	68	191	NaN	AWD	7.0	4.0
261	Acura	MDX	2013	43.0	18.0	300	37.0	6.0	78	68	191	NaN	AWD	7.0	4.0
262	Acura	MDX	2013	43.0	18.0	300	37.0	6.0	78	68	191	NaN	AWD	7.0	4.0
263	Acura	MDX	2013	43.0	18.0	300	37.0	6.0	78	68	191	NaN	AWD	7.0	4.0
264	Acura	MDX	2013	43.0	18.0	300	37.0	6.0	78	68	191	NaN	AWD	7.0	4.0
265	Acura	MDX	2013	43.0	18.0	300	37.0	6.0	78	68	191	NaN	AWD	7.0	4.0

Figure 5 shows the individual columns that were null

```
de[columns].isnull().sum() #all cleaned
```

```
Make                            0
Model                           0
Year                            0
MSRP                            0
Front Wheel Size (in)           0
SAE Net Horsepower @ RPM        0
Displacement                    0
Engine Type                     0
Width, Max w/o mirrors (in)     0
Height, Overall (in)            0
Length, Overall (in)            0
Gas Mileage                     0
Drivetrain                      0
Passenger Capacity              0
Passenger Doors                 0
Body Style                      0
dtype: int64
```

Figure 6 shows that all of the data has been cleansed

Referring to Figure 6, this shows that all of the data has been cleansed. As shown, the procedure for cleaning and averaging all of this data was extensive and long. However, I have made it correct by averaging better and accurate data that I felt was correct. This is going to be explained extensively later on within this report.

I also noticed many hyphens that were missing included within Height, Overall and Width, Overall. The query below changed these numeric values to *nan* values that resulted in updated these values as shown the same procedure within Figure 5.

de['Height, Overall (in)'] = pd.to_numeric(de['Height, Overall (in)'], errors='coerce')

I also noticed that one column had zeros written within the column. This was completed a similar way by updating the dataset. However, I replaced the zeros with the correct values that I thought was correct. You can see the statement below that I narrowed the query down to 6 columns to make sure no other columns were effected.

de.update(de.iloc[16994:17068, 0:6].replace(0,140))

Model_Code	Make_Code
145	0
145	0
145	0
145	0
145	0
...	...
302	40
302	40
302	40
302	40
302	40

Figure 7 shows the Model and Make code categorizing

Referring to Figure 7, this shows the categorising of each Model and Make. You may notice they are many Models and Makes that cannot get tested as strings. Therefore, when this data gets tested, and split; all of this data needs to be floats. I used a LabelEnconder where this creates both of these extra columns and does this for us automatically within one line of code. This helps big time as these two main attributes that contribute to the testing. You can see the query below of what is used for both of these columns.

ds["Model_Code"] = LabelEncoder().fit_transform(ds["Model"])

ds["Make_Code"] = LabelEncoder().fit_transform(ds["Make"])

Stage 3: Building the Model

Categorising Data

Model_Code	Make_Code
145	0
145	0
145	0
145	0
145	0
...	...
302	40
302	40
302	40
302	40
302	40

Figure 8 shows the Model and Make code categorizing

Referring to Figure 7, this shows the categorising of each Model and Make. You may notice they are many Models and Makes that cannot get tested as strings. Therefore, when this data gets tested, and split; all of this data needs to be floats. I used a LabelEnconder where this creates both of these extra columns and does this for us automatically within one line of code. This helps big time as these two main attributes that contribute to the testing. You can see the query below of what is used for both of these columns.

ds["Model_Code"] = LabelEncoder().fit_transform(ds["Model"])

ds["Make_Code"] = LabelEncoder().fit_transform(ds["Make"])

Splitting Data

Referring to Figure 8, this shows the x and y variables that were used within the model. You may notice that the variables of x are encoded with Make, Model, Year, Horsepower, and Gas Mileage. These are the five columns that were suggested by existing research that has been mentioned previously. As you may notice, these five variables are being tested against the Manufacturer Suggested Retail Price (MSRP). Referring to Figure 9, 10, 11 and 12; this shows only the first five out of all the data that is going to be tested.

```
test=['Make_Code', 'Model_Code', 'Year', 'SAE Net Horsepower @ RPM', 'Gas Mileage']
x_variables = ds[test] #independent variables
y_variables = ds['MSRP']   #dependent variables
x_training, x_testing, y_training, y_testing = model_selection.train_test_split(x_variables, y_variables, random_state=42)
```

Figure 9 shows the data being split using x and y variables.

```
y_testing.head()
```

```
33656    25
26659    66
31281    19
61475    19
28094    18
Name: MSRP, dtype: int64
```

Figure 11 shows the y variables for testing

```
y_training.head()
```

```
25607    35
51463    13
30256    29
7544     87
59774    36
Name: MSRP, dtype: int64
```

Figure 10 shows the y variables for testing

```
x_testing.head()
```

	Make_Code	Model_Code	Year	SAE Net Horsepower @ RPM	Gas Mileage
33656	20	77	2019	180	22
26659	14	312	2019	420	14
31281	17	286	2016	130	28
61475	39	137	2016	170	25
28094	16	114	2019	120	31

Figure 12 shows the x variables for testing

```
x_training.head()
```

	Make_Code	Model_Code	Year	SAE Net Horsepower @ RPM	Gas Mileage
25607	14	248	2020	280	16
51463	32	288	2012	100	30
30256	17	243	2013	260	19
7544	4	19	2016	310	20
59774	38	266	2017	270	19

Figure 13 shows the x variables for training

Stage 4: Testing and Validating Models

You may refer yourself to three of the algorithms that had been stated I was going to do. Within this section, I will explain the steps that I took to get the three different results. Each of these results are different.

Gradient Boosting

Gradient Boosting Regression is a machine learning algorithm uses "boosting" to use simple models as a combination many simple models to put a strong force. Gradient comes from the term where this minimizes lose (Dhiraj, 2019). This gives better accuracy and better results for models that have not been trained properly. Referring to Figure 13, this shows the preparation of fitting the data with the training model.

```
gb = GradientBoostingRegressor()
gb.fit(x_training, y_training)
```

```
GradientBoostingRegressor(alpha=0.9, criterion='friedman_mse', init=None,
                          learning_rate=0.1, loss='ls', max_depth=3,
                          max_features=None, max_leaf_nodes=None,
                          min_impurity_decrease=0.0, min_impurity_split=None,
                          min_samples_leaf=1, min_samples_split=2,
                          min_weight_fraction_leaf=0.0, n_estimators=100,
                          n_iter_no_change=None, presort='auto',
                          random_state=None, subsample=1.0, tol=0.0001,
                          validation_fraction=0.1, verbose=0, warm_start=False)
```

Figure 14 shows the Gradient Boosting being fit within the training values

```
print(gb.score(x_training, y_training)) #score result 75% goes to training 25% is what is actually tested.
print(gb.score(x_testing, y_testing))
```

```
0.9537910452268923
0.9563568843335911
```

Figure 15 shows the results of the training and test data

	Actual Data	Predicted Data
0	25	28.514
1	66	74.097
2	19	20.389
3	19	24.558
4	18	21.323
...	...	...
15909	25	23.847
15910	28	26.803
15911	17	21.132
15912	49	60.666
15913	124	160.088

Figure 16 shows the predicted data that has been tested for Gradient Boosting

Referring to Figure 15, this shows the accuracy and data that is being tested. Both of the tests for training and testing values have come across as 95% overall. You can see within Figure 16 of how accurate the tests are. For example, actual data is 25, therefore, the predicted outcome comes close to 28. Therefore, this is close. You can view the Visualisation within Stage 5 for this.

Random Forest

Random Forest Regression is a machine learning technique that combines multiple decision trees to determine an output rather than relying on individual decision trees. This chooses at a random point from the training set, builds decisions around the data and these two repeat until this has been completed. A benefit to using this algorithm is that this is not biased (Malik, 2020).

```
from sklearn.ensemble import RandomForestRegressor
rfRegressor = RandomForestRegressor(max_depth=5, random_state=0, n_estimators=10)
rfRegressor.fit(x_training, y_training)
print('Training Score:', np.round(rfRegressor.score(x_training, y_training), 3))
print('Testing Score:', np.round(rfRegressor.score(x_testing, y_testing), 3))
```

```
Training Score: 0.907
Testing Score: 0.904
```

Figure 17 shows the result for Random Forest.

	Actual Data	Predicted Data
0	25	30.050
1	66	55.619
2	19	21.752
3	19	21.752
4	18	21.752
...	...	...
15909	25	30.050
15910	28	30.050
15911	17	21.752
15912	49	64.683
15913	124	150.720

Figure 18 shows the predicted data for Random Forest.

Referring to Figure 17, this shows that the overall testing for both values comes at an average of 90%. As you may have noticed, this is significantly worse from Gradient Boosting. This will be explained in detail later on within this section. Figure 18 shows that the results compared to each other show that this is not as close as it seems. Therefore, the accuracy of this data has not been as hoped. You can view the Visualisation within Stage 5 for this.

Decision Trees

Decision Tree Classifier is a powerful tool that has a basic idea behind this. It has a Flowchart structure by branches represented by decision rules. The structure starts the tree by repeating until one of the conditions is met of all the tuples belong to the same attribute value, they are no more remaining attributes, and there is no more instances. This structure is easy to understand which decision has been made (Navlani, 2018).

```
from sklearn.tree import DecisionTreeClassifier
dcClassifier = DecisionTreeClassifier()
dcClassifier.fit(x_training, y_training)
print(dcClassifier.score(x_training, y_training))
print(dcClassifier.score(x_testing, y_testing))
```

```
1.0
0.9990574337061706
```

Figure 20 shows the result for Decision Trees

:

	Actual Data	Predicted Data
0	25	25
1	66	66
2	19	19
3	19	19
4	18	18
...	...	...
15909	25	25
15910	28	28
15911	17	17
15912	49	49
15913	124	124

15914 rows × 2 columns

Figure 19 shows the predicted data for decision trees

Referring to Figure 20, this shows the results for both of testing and training data. This has been be best result as the training data has been 100% and the testing is near to 99.999%. Figure 19 shows proof that the predicted outcome for each of the data has been accurate. You can view the Visualisation within Stage 5 for this.

Comparing all three results

```
Testing Dataset Mean:-
Gradient Boosting Mean: 0.9558962272347065
Random Forest Mean: 0.9185963916670715
Decision Tree Classifier Mean: 0.9902843914607177


Training Dataset Mean:
Gradient Boosting Mean: 0.9537975643812265
Random Forest Regressor Mean: 0.9127688675628901
Decision Tree Classifier Mean: 0.9992058938390753
```

Figure 21 shows the results of all three algorithms

Referring to Figure 21, this shows the average of all the tests. As you can see, the best average for both training is the Decision Tree Classifier. This has shown once again that this has proven to be the best. Hence, the reason why this is the most popular. This algorithm would be the one that I will be picking as a result of its accuracy. You can view the Visualisation within Stage 5 for this too.

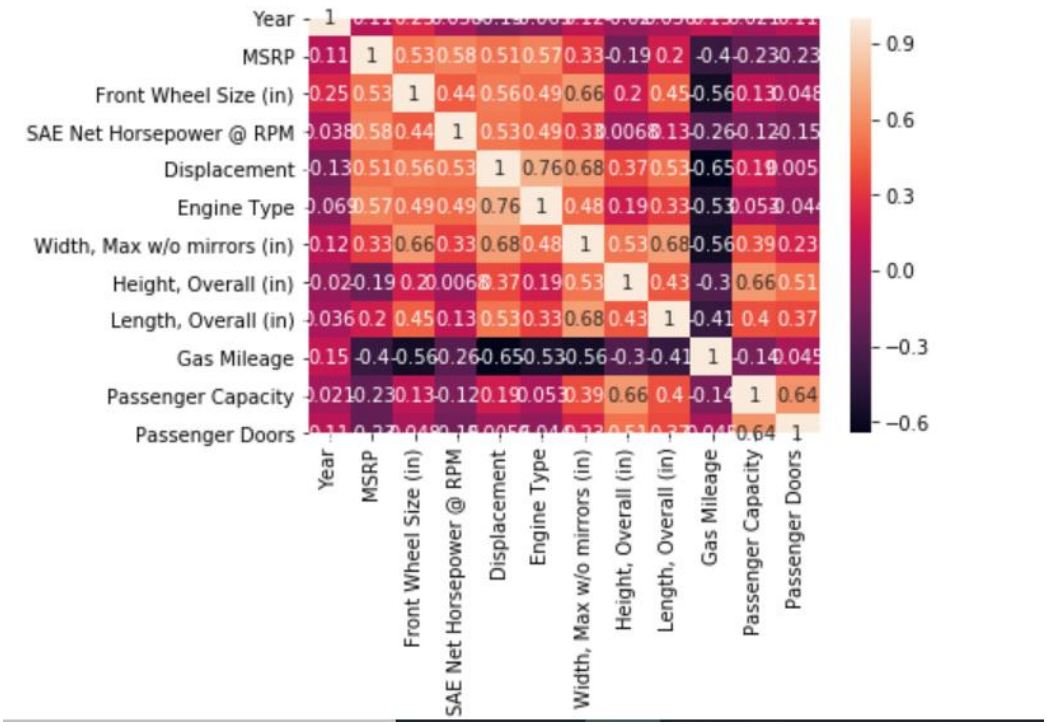
Stage 5: Visualisation

Please view the following as visualization for each stage.

Before Visualisation Overall

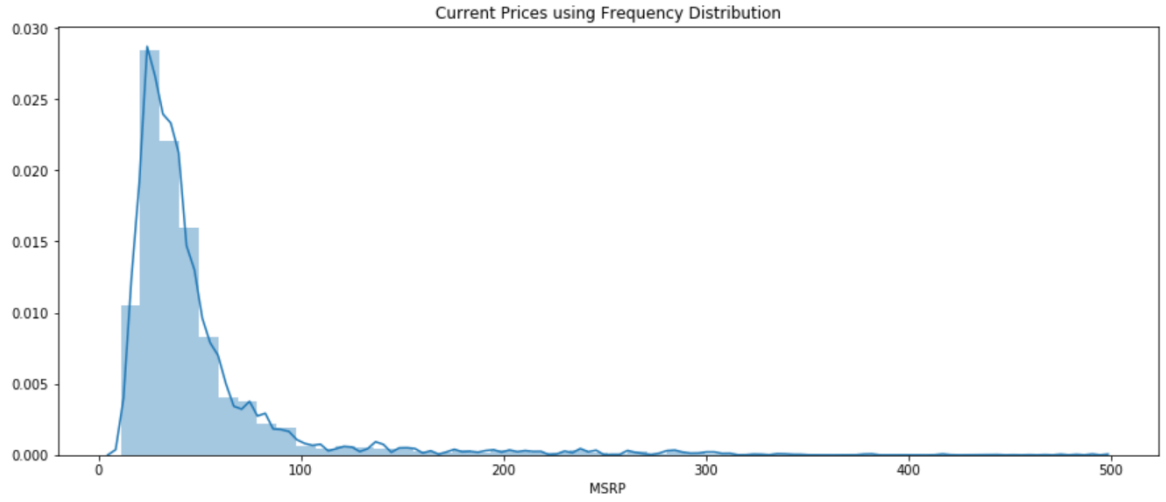
```
import seaborn as sns
sns.heatmap(ds.corr(), annot=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1dee2359188>
```



Before Visualisation only Price

```
Text(0.5, 1.0, 'Current Prices using Frequency Distribution')
```

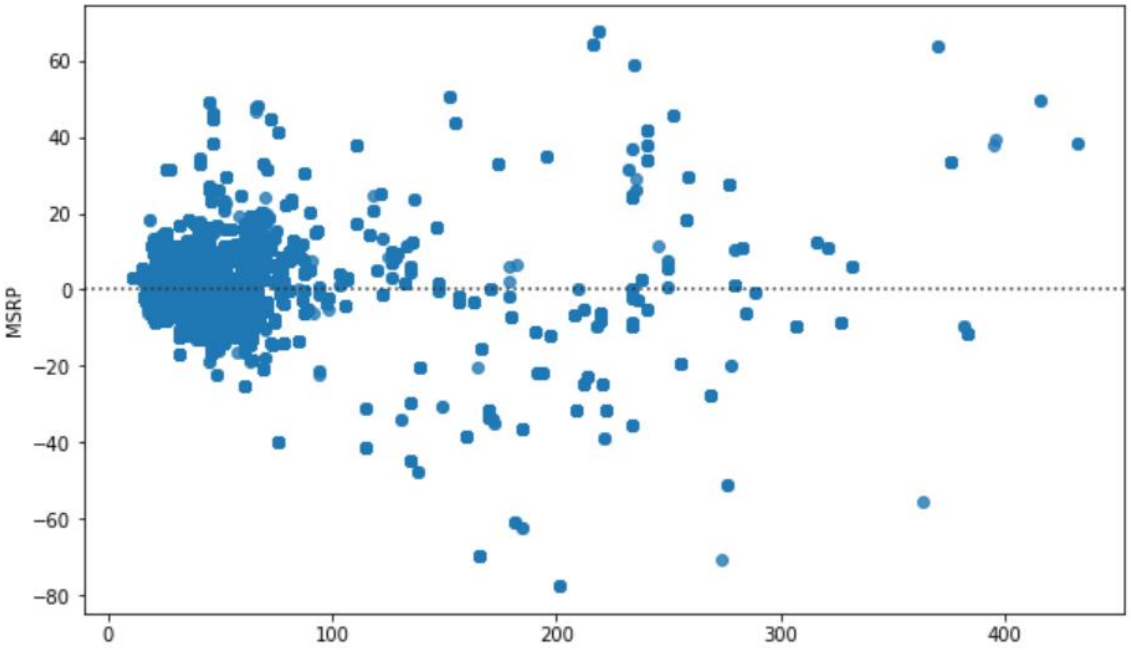


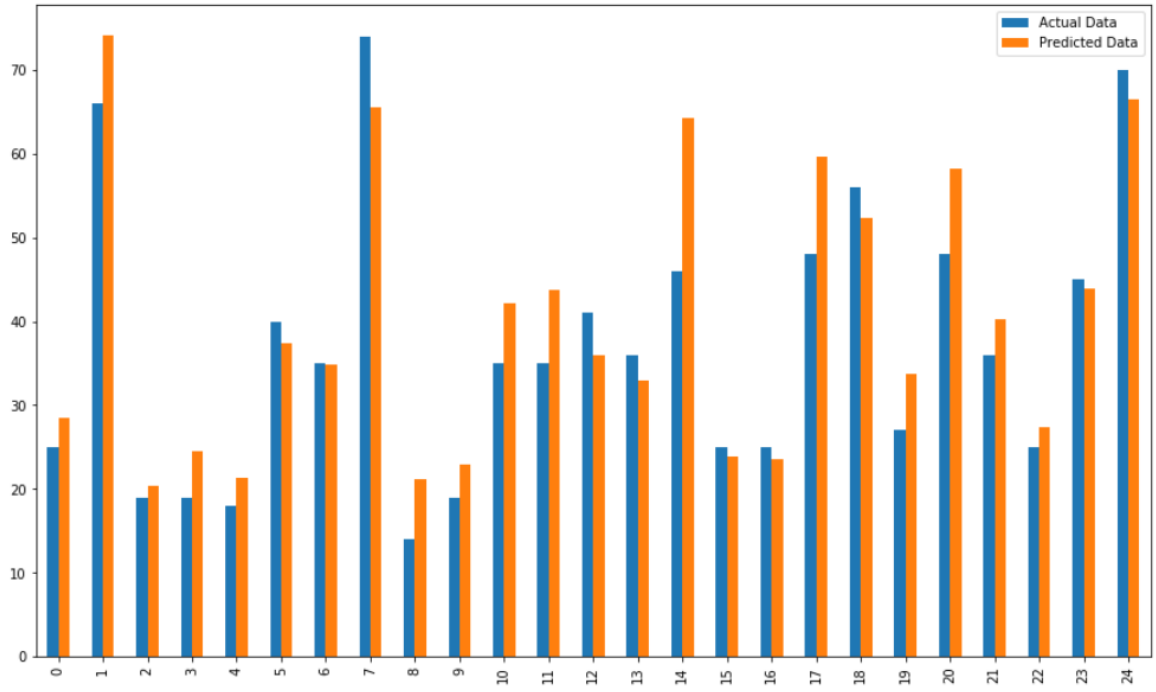

```
ds = ds[ds["Year"] < 2021]
plt.figure(figsize=(15,6))
sns.regplot(x=ds["Year"], y=ds["MSRP"], data=df).set_title("Price vs Year") #plots the price vs years
```

```
Text(0.5, 1.0, 'Price vs Year')
```

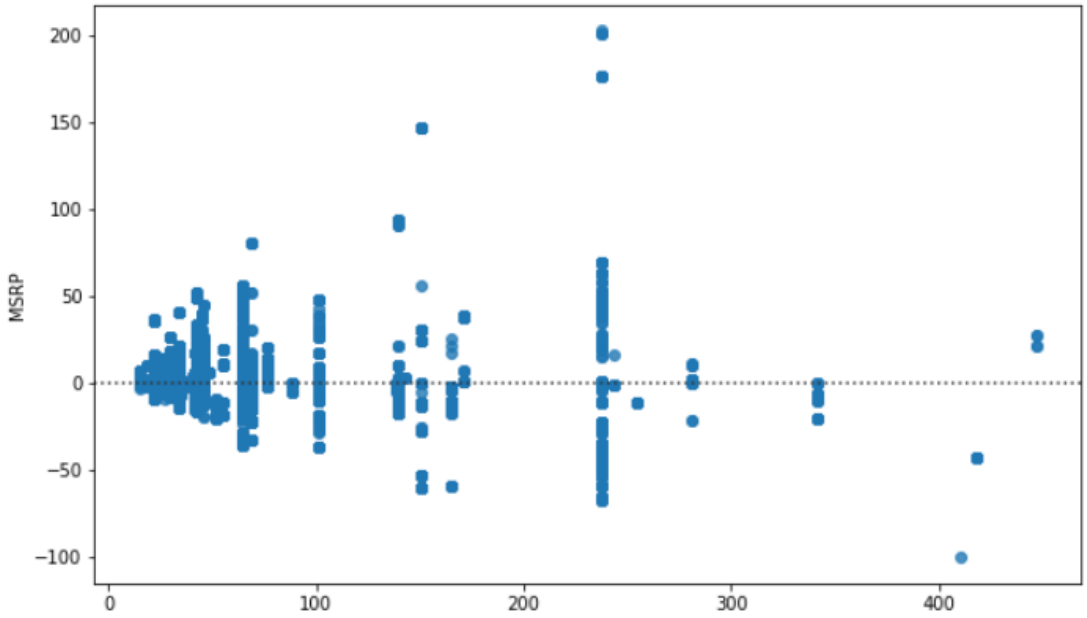
	Year	MSRP	Front Wheel Size (in)	SAE Net Horsepower @ RPM	Displacement	Engine Type	Max w/o mirrors (in)	Height, Overall (in)	Length, Overall (in)	Gas Mileage	Passenger Capacity	Passenger Doors
Year	1.000000	0.105448	0.246792	0.037901	-0.125140	-0.068836	0.119626	-0.019965	0.036007	0.150382	0.021470	0.106816
MSRP	0.105448	1.000000	0.527739	0.577222	0.508049	0.570511	0.331142	-0.187317	0.198620	-0.398118	-0.225590	-0.226558
Front Wheel Size (in)	0.246792	0.527739	1.000000	0.435377	0.555635	0.493307	0.664355	0.195831	0.449507	-0.563363	0.129900	0.048064
SAE Net Horsepower @ RPM	0.037901	0.577222	0.435377	1.000000	0.534181	0.487187	0.331930	0.006759	0.126700	-0.264273	-0.120023	-0.150922
Displacement	-0.125140	0.508049	0.555635	0.534181	1.000000	0.755394	0.675832	0.371503	0.528174	-0.645340	0.193147	0.005637
Engine Type	-0.068836	0.570511	0.493307	0.487187	0.755394	1.000000	0.483368	0.185915	0.332483	-0.526954	0.052784	-0.044350
Width, Max w/o mirrors (in)	0.119626	0.331142	0.664355	0.331930	0.675832	0.483368	1.000000	0.529538	0.682063	-0.562026	0.386026	0.225858
Height, Overall (in)	-0.019965	-0.187317	0.195831	0.006759	0.371503	0.185915	0.529538	1.000000	0.426863	-0.295503	0.658453	0.509384
Length, Overall (in)	0.036007	0.198620	0.449507	0.126700	0.528174	0.332483	0.682063	0.426863	1.000000	-0.407045	0.401824	0.373708
Gas Mileage	0.150382	-0.398118	-0.563363	-0.264273	-0.645340	-0.526954	-0.562026	-0.295503	-0.407045	1.000000	-0.136130	0.044908
Passenger Capacity	0.021470	-0.225590	0.129900	-0.120023	0.193147	0.052784	0.386026	0.658453	0.401824	-0.136130	1.000000	0.636517

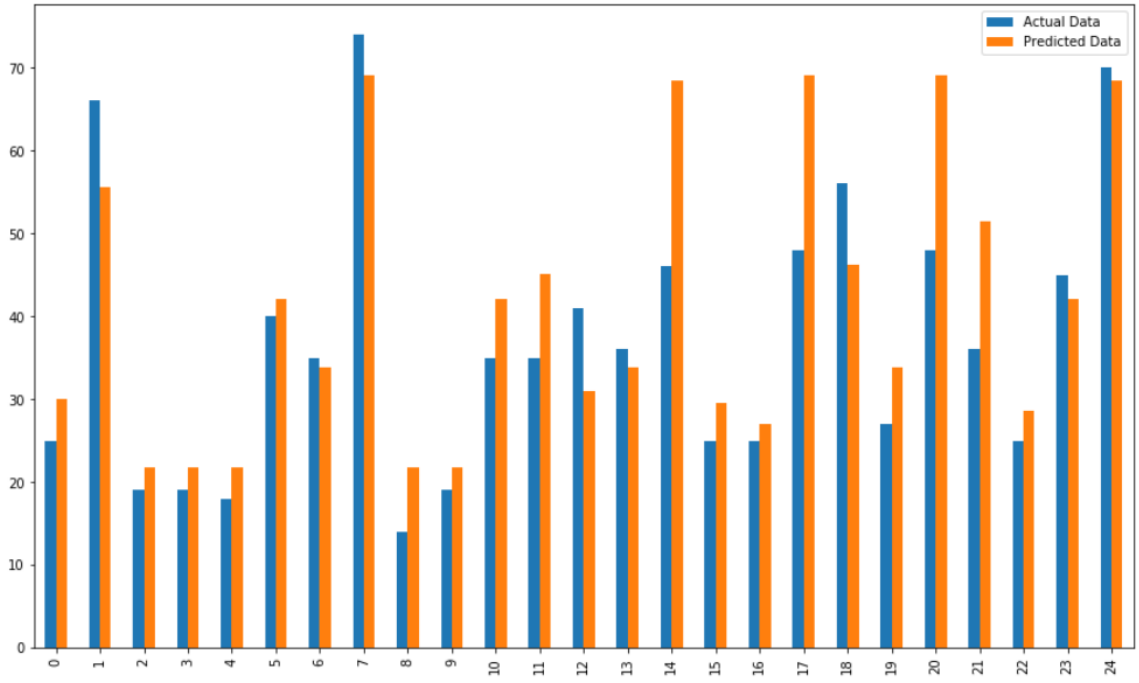
Gradient Boosting Visualisation



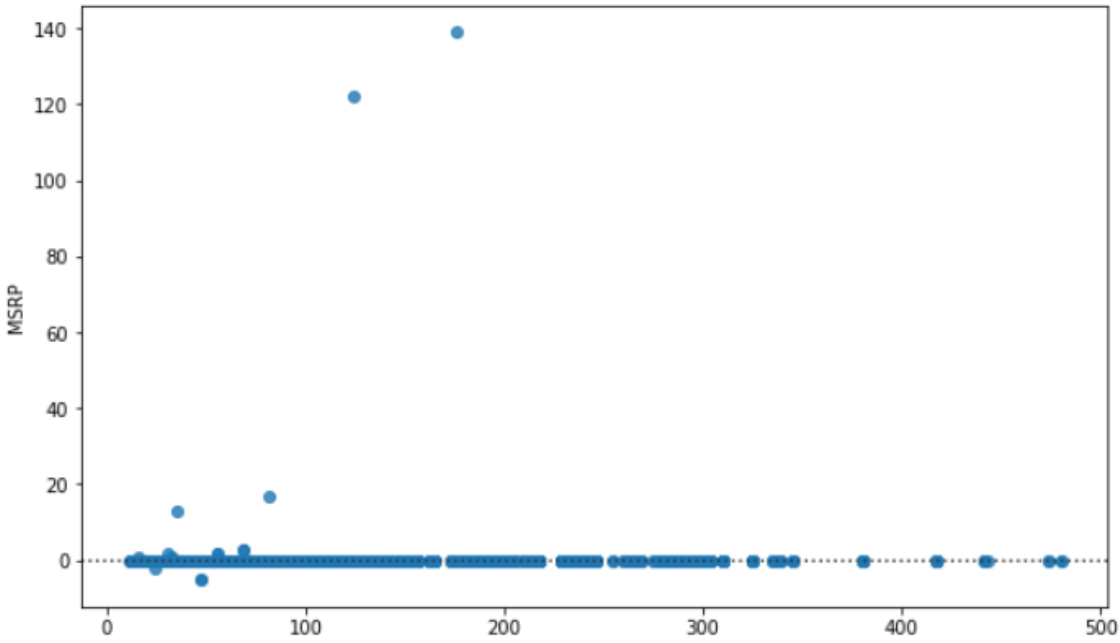


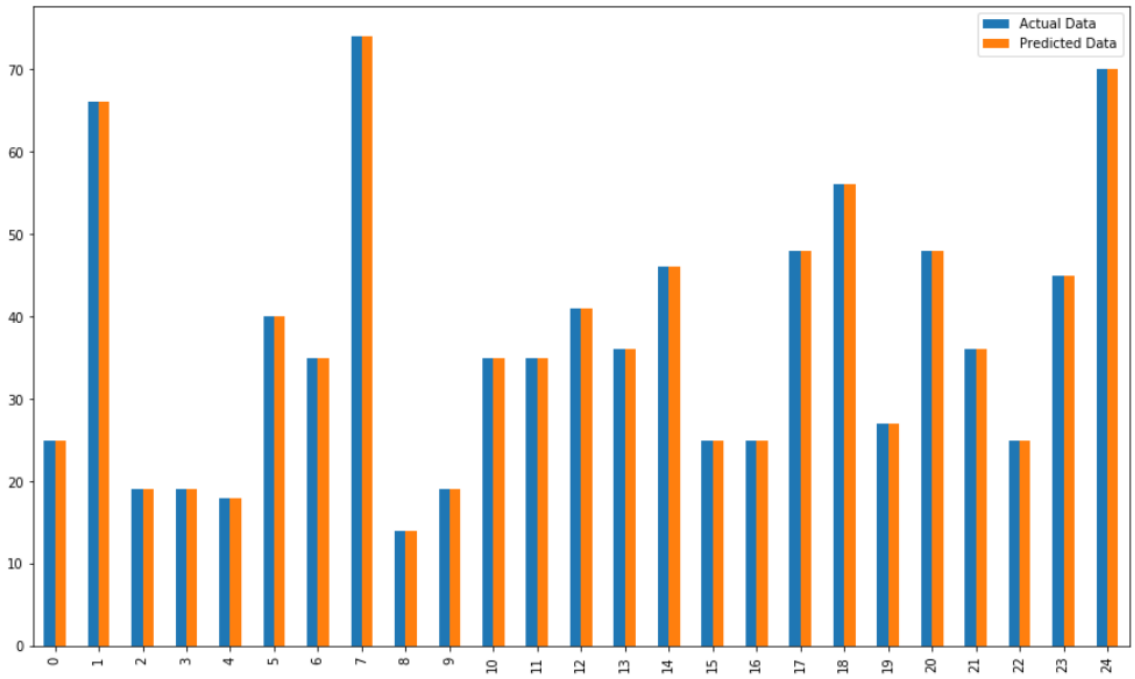
Random Forest Visualisation



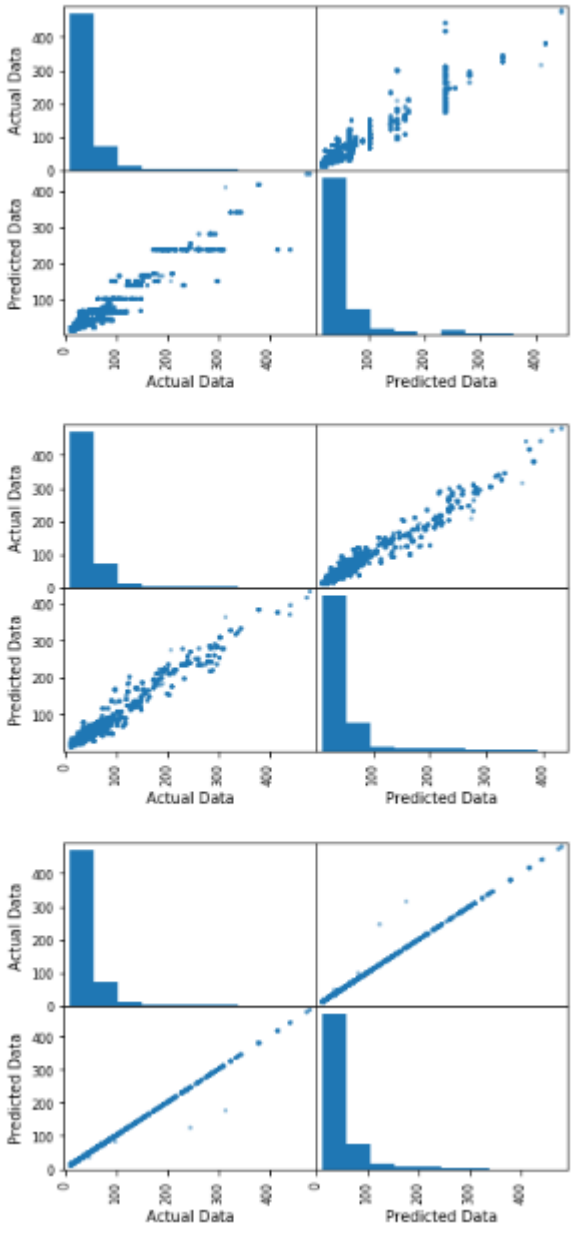


Decision Trees Visualisation





Comparing all three machine learning techniques



Bugs and Weaknesses

Feature Scaling

Another feature that I struggled is putting Feature Scaling within the data pre-processing. I put this before splitting the data. However, whenever I initialised Standard Scaler with two of my training and test variables. I ran these tests with Gradient Boosting and the score became minus and inconsistent. Therefore, for this reason, I took this out as this gave me errors that I did not see coming. This was an attempt to improve the accuracy of the variables that have been tested. This was a feature that I wanted to improve on. However, this would be one of the suggested improvements that are necessary.

Stratified Sampling

Another feature that I struggled to implement was Stratified Sampling. This is a simple feature where this distributes the classes in an attempt to have more balanced data. However, when I tried to put stratify=y_variables when I split the data, I got the error below. I tried research this error and found that it was suggested to take this out to solve the error. Therefore, this would be one of the suggested areas that need improvement.

> ***ValueError**: The least populated class in y has only 1 member, which is too few. The minimum number of groups for any class cannot be less than 2.*

Conclusion

In conclusion, this dataset has been very interesting to deal break down this data and gain information from this. I felt that the three algorithms that have been used were very interesting in terms of comparing results between all three. However, the best and most accurate would be the decision tree as compared to the rest; the result is 99.99%. One of the improvements that I would make straight away is the way that the averages that I did. I located the empty columns and rows by using iloc. However, I would want to improve this by using a *for loop* that does this automatically.

I must admit that that this coursework was an initial challenge for myself. However, breaking each barrier as it came along was a huge accomplishment. This was an interesting task to tackle a huge dataset and to get a recent dataset that just got released.

References

Dhiraj, K. (2020) Implementing Gradient Boosting in Python, *Paperspace*, [online] Available at: https://blog.paperspace.com/implementing-gradient-boosting-regression-python/ (Accessed 12 April 2020).

Gegic, E., Isakovic, B., Keco, D., Masetic, Z. and Kevric, J., 2019. Car price prediction using machine learning techniques. *TEM Journal*, *8*(1), p.113.

Malik, U. (2020) Random Forest Algorithm with Python and Scikit-Learn, *Stack Abuse*, [online] Available at: https://stackabuse.com/random-forest-algorithm-with-python-and-scikit-learn/ (Accessed 12 April 2020).

Navlani, A. (2018) Decision Tree Classification in Python, *DataCamp Community*, [online] Available at: https://www.datacamp.com/community/tutorials/decision-tree-classification-python (Accessed 12 April 2020).

Pudaruth, S., 2014. Predicting the price of used cars using machine learning techniques. *Int. J. Inf. Comput. Technol*, *4*(7), pp.753-764.