Predicting Used Car Prices Using Regression

Introduction and Objectives

Approximately 40 million used vehicles are sold each year. Effective pricing strategies can help any company to efficiently sell its products in a competitive market and making profit.

There are two main goals I want to achieve with this Data Science Project. First, to estimate the price of used cars by taking into account a set of features, based on historical data. Second, to get a better understanding on the most relevant features that help determine the price of a used vehicle.

Data Overview

Dataset used in this project was ‘Cars4U” data set. The dataset initially has 7253 rows and 14 columns.

The dataset has a total of 14 features namely; name, Location, Kilometers driven, Fuel type, Transmission, Owner, Mileage, Engine, Power, Seats, New Price, and Price. The continuous variable “Price” is to be predicted.

In [8]:
```
df = pd.read_csv(r'C:\Users\Odhiambo\Desktop\used_cars_data.csv', encoding= 'unicode_escape')
print(f'There are {df.shape[0]} rows and {df.shape[1]} columns.')  # f-string

# I'm now going to look at 10 random rows
# I'm setting the random seed via np.random.seed so that
# I get the same random results every time
np.random.seed(1)
df.columns
```

```
There are 7253 rows and 14 columns.
```

Out[8]:
```
Index(['S.No.', 'Name', 'Location', 'Year', 'Kilometers_Driven', 'Fuel_Type',
       'Transmission', 'Owner_Type', 'Mileage', 'Engine', 'Power', 'Seats',
       'New_Price', 'Price'],
      dtype='object')
```

Exploratory Data Analysis (EDA)

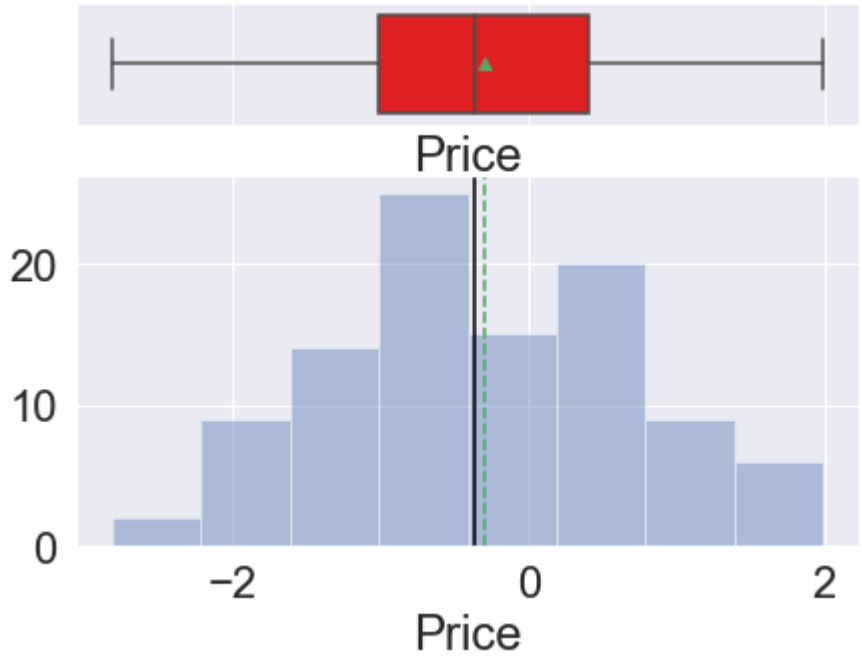
The numerical features play a big role in this Regression model, so it is important to understand well how are they distributed in the dataset.

```
df.describe().T  # quick summary of numeric features
```

	count	mean	std	min	25%	50%	75%	max
S.No.	7253.0	3626.000000	2093.905084	0.00	1813.0	3626.00	5439.00	7252.0
Year	7253.0	2013.365366	3.254421	1996.00	2011.0	2014.00	2016.00	2019.0
Kilometers_Driven	7253.0	58699.063146	84427.720583	171.00	34000.0	53416.00	73000.00	6500000.0
Seats	7200.0	5.279722	0.811660	0.00	5.0	5.00	5.00	10.0
Price	6019.0	9.479468	11.187917	0.44	3.5	5.64	9.95	160.0

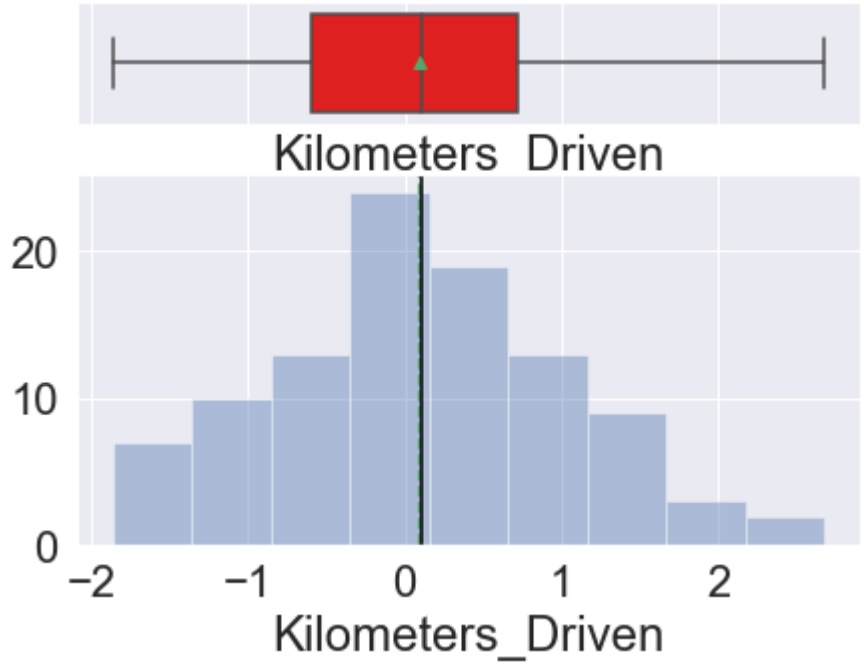
Our focus will be “price”, “year” and “Kilometers Driven”. As shown in the picture above, there is a big difference between both the maximum value/minimum value and the percentiles for each of these three features. This is an indicator of the presence of outliers, which can greatly hinder the performance of our model. They will be handled later.

Univariate Analysis



Observation

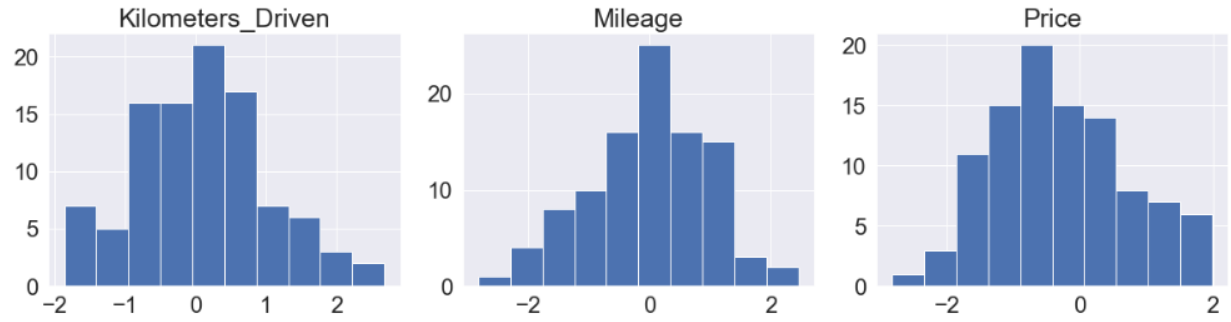
Price is right skewed which means some cars have prices greater than average prices



Observation

Kilometers driven is right skewed which means some cars have Kilometers driven greater than average prices.

Distribution of each numerical variable



Bivariate Analysis

Price is negatively correlated with Kilometers driven and Mileage which means that as Kilometers driven and Mileage (0-2) increases, Price of cars tend to decrease.

Insights based on EDA

Kilometers driven vs Price

There is negative relationship between the kilometers driven by the car and the price.

In this case, none of the variables were highly correlated so we leave them as they are.

Data pre-processing

(i) Data Preparation for modelling

In regard data cleaning process, the existing non-values and inconsistencies in the data was taken into consideration. More than 30000 cars had a selling price of 0. I started with removing those cars from the dataset. The column kilometer Driven had some missing values, I couldn't just remove all those rows.

So I replaced them with the median kilometer value of make year of that car. I dropped cars with a price less than 1000 with miles less than 60,000 and model year greater than 2010. Cars with miles more than 500000 were removed. After data cleaning, I had only 387772 rows. Dummy variables were created for all the categorical variables.

(ii) Missing values & Duplicates Treatment

After understanding the database that we will be working with, it is important to always check if there are any missing values (NaN).

```
In [12]: df.isnull().sum() # lots of columns don'thave missingnes

Out[12]: S.No.                0
         Name                 0
         Location             0
         Year                 0
         Kilometers_Driven    0
         Fuel_Type            0
         Transmission         0
         Owner_Type           0
         Mileage              2
         Engine              46
         Power               46
         Seats               53
         New_Price         6247
         Price             1234
         dtype: int64
```

Lost variables (columns) don't have missing values, but it is not practical to just delete all the rows with missing values since that would thin out the dataset too much.

For now, I will leave it at that and just drop some of the variables/columns that don't bring any added value. Keeping them will just make it harder for the model to interpret the dataset.

```
In [14]: df.drop(['S.No.', 'Name', 'Location'],axis=1,inplace=True)

In [15]: df.shape

Out[15]: (7253, 11)
```

Lastly, it is also necessary to check if there is any case of duplicate values in this dataset.
In this case, there are 6 duplicates in the dataset. They will be dealt with by dropping the existing duplicate rows except for the first one in each case, which will be kept.

```
In [17]: df.duplicated().sum()

Out[17]: 6
```

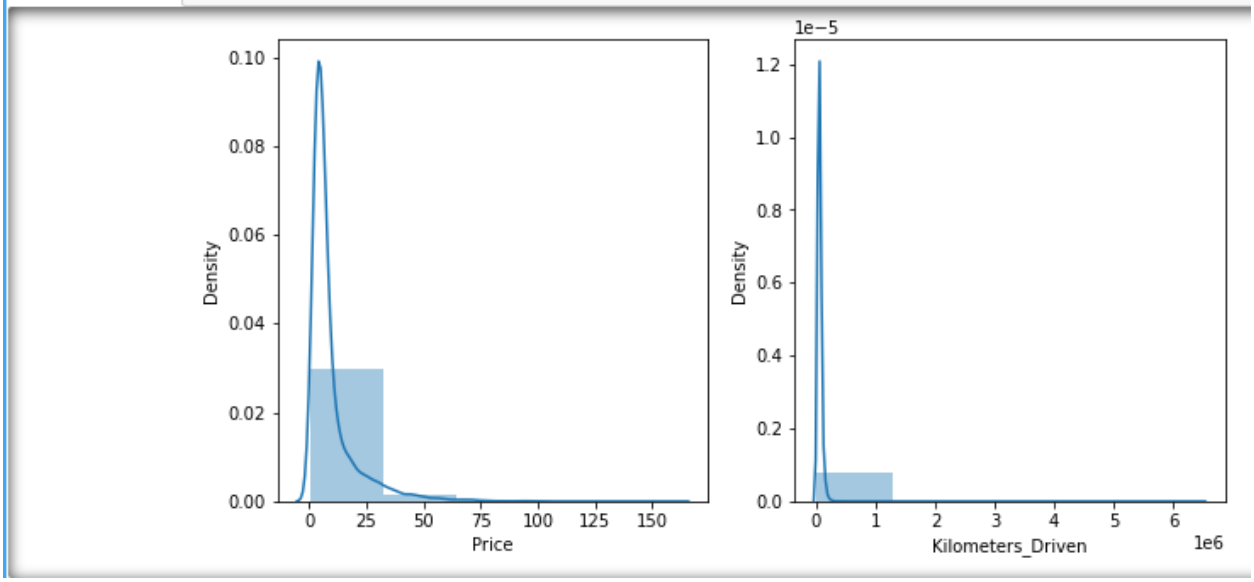
(iii) Feature engineering

First, I will be using **Regular Expressions** on the column "Name". This variable contains the text with the brand and model of each used car.
The goal is to get key information from there in order to fill the missing values of other columns in the dataset. The columns we want to fill are those with categorical values that don't have too many unique values
There are values that have been added inconsistently. To unify the strings, the strip () method will be used.

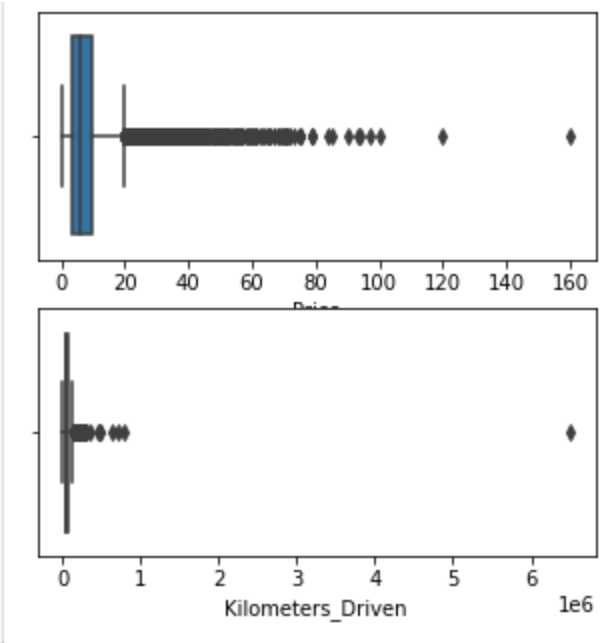
(iv) Outlier Treatment

Onto the next step, it is time to **handle the outliers** in the database. The three columns that will be taken care of are: "odometer", "price" and "year".
Let's start by visualising the two first ones:

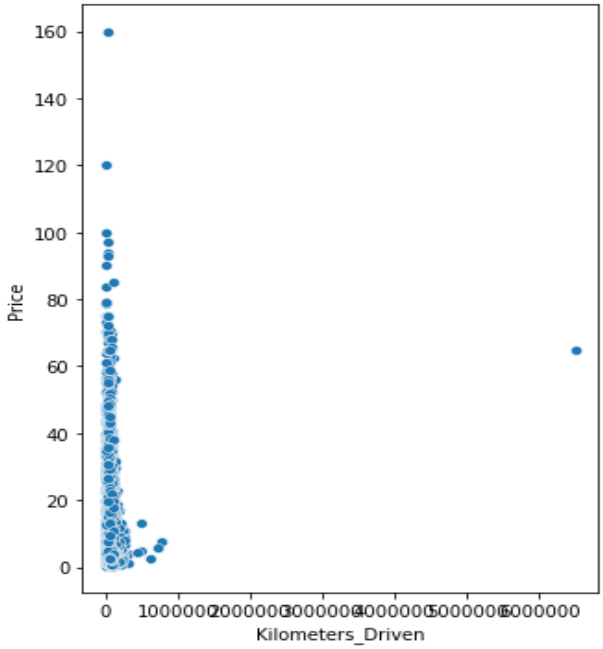
```
In [19]: plt.figure(figsize=[10,5])
         plt.subplot(121)
         sns.distplot(df["Price"],bins=5)
         plt.subplot(122)
         sns.distplot(df["Kilometers_Driven"],bins=5)
```



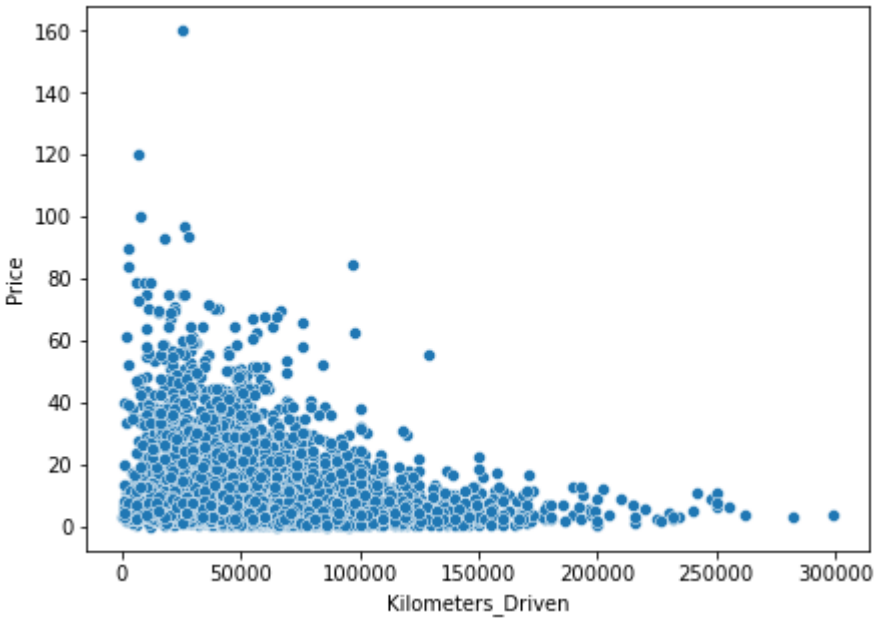
First, the column “Price” has many missing values so it is trickier to deal with. Let’s use scatterplot to visualize the outliers.



```
plt.figure(figsize=[5,7])
ax=sns.scatterplot(x=df["Kilometers_Driven"],y=df["Price"])
ax.get_xaxis().get_major_formatter().set_scientific(False)
ax.get_yaxis().get_major_formatter().set_scientific(False)
```

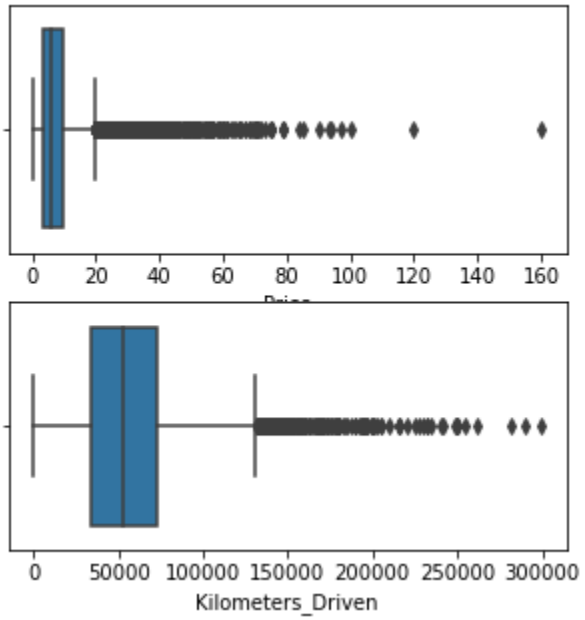


Since the outliers start at approximately 300,000; I will be dropping the values that exceeds that. Also, it is a good idea to drop the minimum value (0) since it greatly differs from the 25% percentile. With this, the feature “odometer” has also been successfully adjusted.

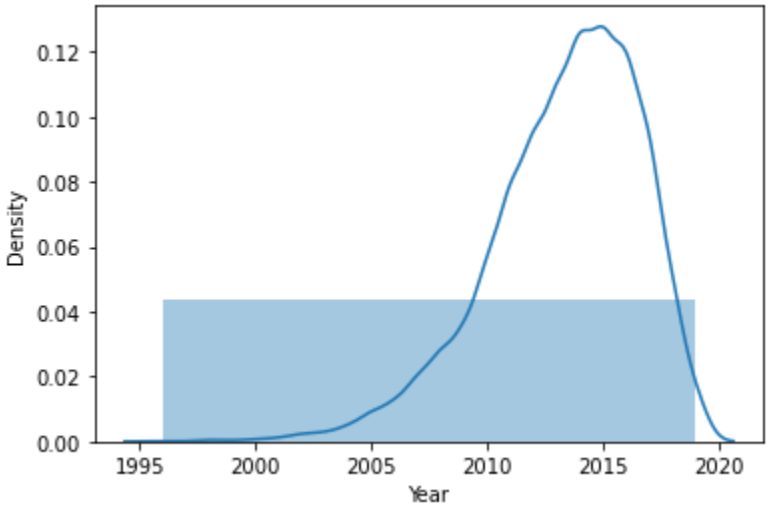


To end this step, the outliers of the feature “year” will be handled the same way it was done in “Kilometers driven”.

To that end, I will be utilising the “Kilometer driven” values. The “kilometers driven” variable is a continuous variable so it is better to group its values into several bins to serve as a reference.



Next, since “kilometers driven” had such a big difference between the minimum value and the 25% percentile as well as between the maximum value and the 75% percentile, I will be leaving out any of the values on each end and also since the price has a missing value it.



Model building - Linear Regression

While exploring the data I checked the relationship between miles on a car and the price of the car. As we all know the more miles on the car, the price will go down. Cars with fewer miles are always expensive.

I also checked the relationship between the year and the price of a car. Obviously older cars are cheaper with some exceptions to antique cars and newer cars are always expensive.

When two variables are highly correlated, they have the same effect on the dependent variable that we want to predict. In that case, it is recommendable dropping one of them.

	Metrics	Accuracy
0	R2	85.48

We got a score of more than 85.48% right off the start. Not bad at all. But we can improve it further by tuning the Hyper-parameter.

After doing the optimization of the Hyper-parameters, the results were much better

The results seem to be pretty solid, which is shown by the good performance of our model in both Train and Test Set. The fact that it got even more accurate in its predictions in the Test Set (86.34%) is especially remarkable.

	Metrics	Accuracy	Accuracy Tuned Param
0	R2	85.48	86.34

As we can see from the picture above, the score got even better after doing the optimisation of the Hyper-parameters.

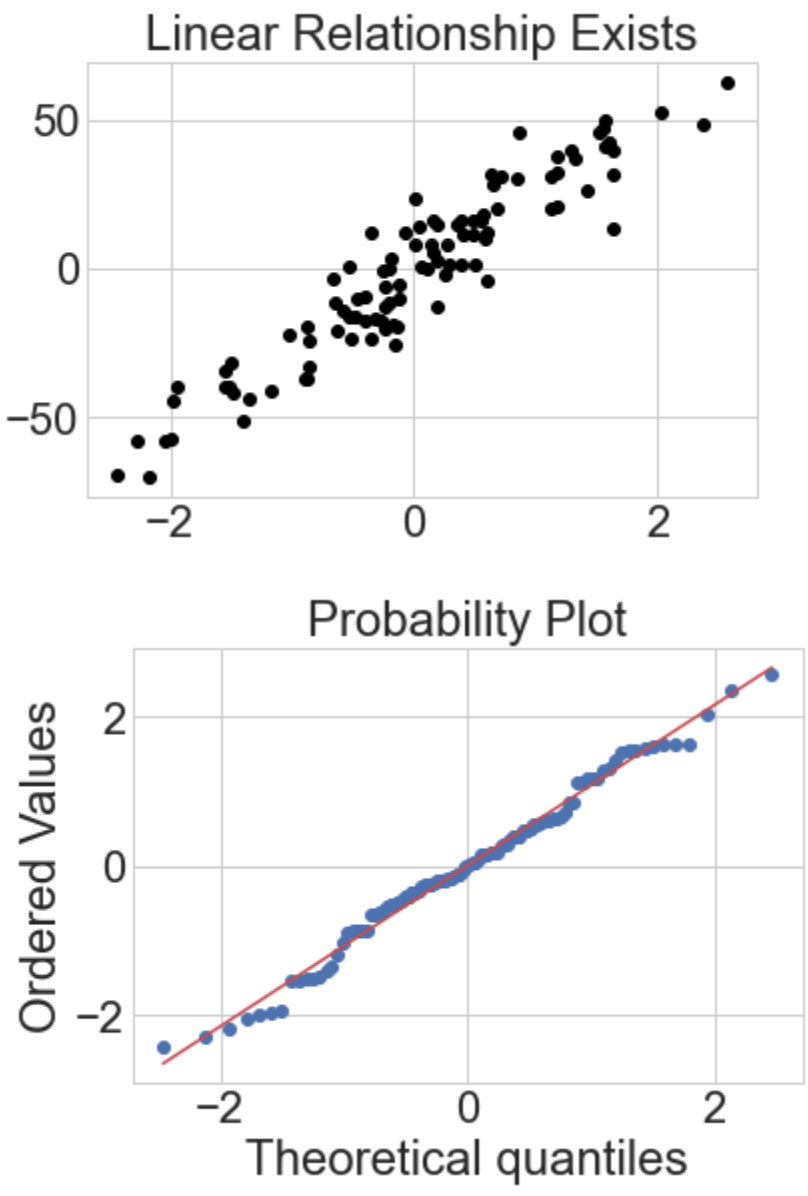
Finally, once the Training is finished and the model has learned from the training data, all that is left is to check how well does it perform in the Test Set? This will be the moment of truth for our model.

	Metrics	Accuracy	Accuracy Tuned Param	Accuracy Test Set
0	R2	85.48	86.34	87.03

Assumptions of linear regression model

Here, the test for normality was applied. The QQ plot of residuals was used to visually check the normality assumption. The normal probability plot of residuals followed a straight line suggesting that the linear relationship exists

```
Text(0.5, 1.0, 'Linear Relationship Exists')
```



Model performance evaluation

To begin the modelling, we want to divide the database into Train and Test Set. It will be done on a distribution of 80:20. As usual, we will first re-index the database and put the dependent variable “Price” as the last column for a simpler splitting.

	Metrics	Accuracy	Accuracy Tuned Param	Accuracy Test Set
0	R2	85.48	86.34	87.03

It has shown an excellent performance in such a big dataset and it has performed consistently throughout the Training and Testing process. Even more, the results of the Test Set are better than in the Training Set, with an 87.03% of accuracy in its predictions.
One of the goals of the project was to create a model that was able to estimate the price of used cars and we already achieved it.
The second goal is to find which features are the **most relevant** ones when estimating the dependent variable “Price”:

Actionable Insights & Recommendations

With this project, we have built a model that can predict with a 87.03% of accuracy the price of used cars, given a set of features. This information can have an enormous value for both companies and individuals when trying to understand how to estimate the value of a vehicle and, more importantly, the key factors that determine its pricing.
As expected, the year of the vehicle is by far the main factor when calculating the price with almost a 43%, followed by odometer. Interestingly enough, I expected the state of the car and the odometer to be deeply related but there is a big gap in the difference of relevance between both measures.
That said, it seems the region also plays a part, which totally makes sense. There might be more general vehicles that are liked everywhere but specialised cars like sport or convertibles would be a better fit in warmer areas whilst bigger trucks and SUVs would play a better role in colder places.
Mastering the art of pricing is not an easy task, but with the study of historical data it is possible to find patterns that lead to accurate results. Acquiring this knowledge can provide you with a comparative advantage before putting a vehicle on sale or buying it on the market.