Executive Summary:

There is a huge demand for used cars in the Indian Market today. As sales of new cars have slowed down in the recent past, the pre-owned car market has continued to grow over the past few years and is now larger than the new car market. Cars4U is a budding tech start-up that aims to find foot holes in this market. In 2018-19, while new car sales were recorded at 3.6 million units, around 4million second-hand cars were bought and sold. There is a slowdown in new car sales and that could mean that the demand is shifting towards the pre-owned market. Infact, some car owners replace their old vehicles with pre-owned cars instead of buying a new automobile. Unlike new cars, where price and supply are deterministic and managed by OEMs (Original Equipment Manufacturer/except for dealership level discounts which come into play only in the last stage of the customer journey), the used car market is a very different beast, with large uncertainties in both pricing and supply. **Several factors, including mileage, brand, model, year, etc. can influence the actual worth of a car.** From the perspective of a seller, it is not an easy task to set the correct price of a used car. Keeping this in mind, the pricing scheme of these used cars becomes important in order to grow in the market. In this article, we look into this problem and develop a forecasting system (using machine learning techniques) that helps a potential seller to estimate the price of a pre-owned car. A dataset is collected and pre-processed. Exploratory data analysis has been performed. Following that, various machine learning regression algorithms, including **Linear Regression**, **Ridge / Lasso Regression**, **Decision Trees**, **Random Forest**, **XGBoost.** After evaluating the performance of each method, the best-performing model (XGBoost) was chosen. This model is capable of properly predicting prices more than 91% of the time.

Problem and Solution Summary:

Due to the unprecedented number of cars being purchased and sold, used car price prediction is a topic of high interest. Because of the affordability of used cars in developing countries, people tend more purchase used cars. A primary objective of this project is to estimate used car prices by using attributes that are highly correlated with a label (Price). An effective pricing model that can **effectively predict the price of used cars** and can help the business in devising profitable strategies using differential pricing.

The key questions:

1. Why are they selling the car?
2. How old is the car?
3. What's the car's mileage?
4. How long have they owned the car?
5. What is the Brand and Model of the car?
6. What is the engine and power of the car?

The problem formulation:

Predict the price of cars in the Indian market. Predictions will be based on vehicle features such as **power, engine, name, Location, age, make, model, mileage**

Methodology

Load libraries

Load the data

Understand the data by observing a few rows

Observations and Insights:
There are both numerical and categorical variables in the dataset as such the categorical variables will have to be encoded. There are missing values in the new_price and price columns The name column contains data that may be able to be separated into different columns eg. "Mercedes-Benz E-Class 2009-2013 E 220 CDI Avan" contains year, brand name and model which may or may not be relevant Serial No. Also, may not be necessary as is too unique and maybe dropped later We also see both categorical and numerical variables which may influence the methods we use

Let us check the data types and and missing values of each column

Observations and Insights:
Both categorical and numerical data can be seen. We will need to come up with ways to address the missing values as well, however al missing values appear to be numerical which will have to be addressed by some statistical means (mean, meadian, ect). The missing data seems minial compared to the number of rows in the dataset and a large missing value within the target variable is as expected
We can observe that S.No. has no null values. Also the number of unique values are equal to the number of observations. So, S.No. looks like an index for the data entry and such a column would not be useful in providing any predictive power for our analysis. Hence, it can be dropped.

Exploratory Data Analysis

Observations and Insights:
The average year seems to be at 2013 Mileage minimum is at 0 which means that there is either outliers or incorrect data

Let us also explore the statistics of all categorical variables

Observations and Insights:
most cars use either desisel or petrol. The least amount of cars are being sold in Ahmedabad. Most cars are manual transmission. Mahindra seems to be the top brand

Check Kilometers_Driven extreme values

Observations and Insights:
In the first row, a car manufactured as recently as 2017 having been driven 6500000 km is almost impossible. It can be considered as data entry error and so we can remove this value/entry from data.

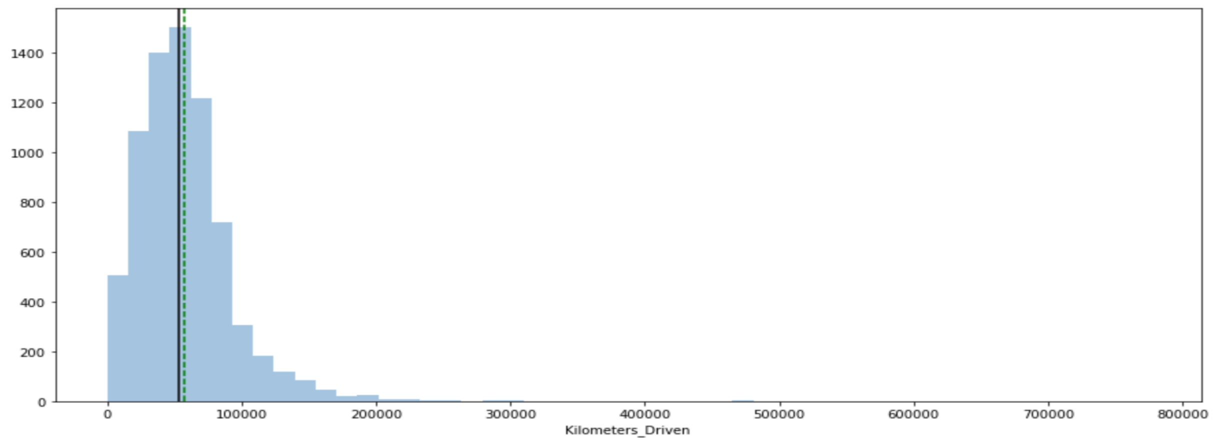
Check Mileage extreme values

Observations and Insights:

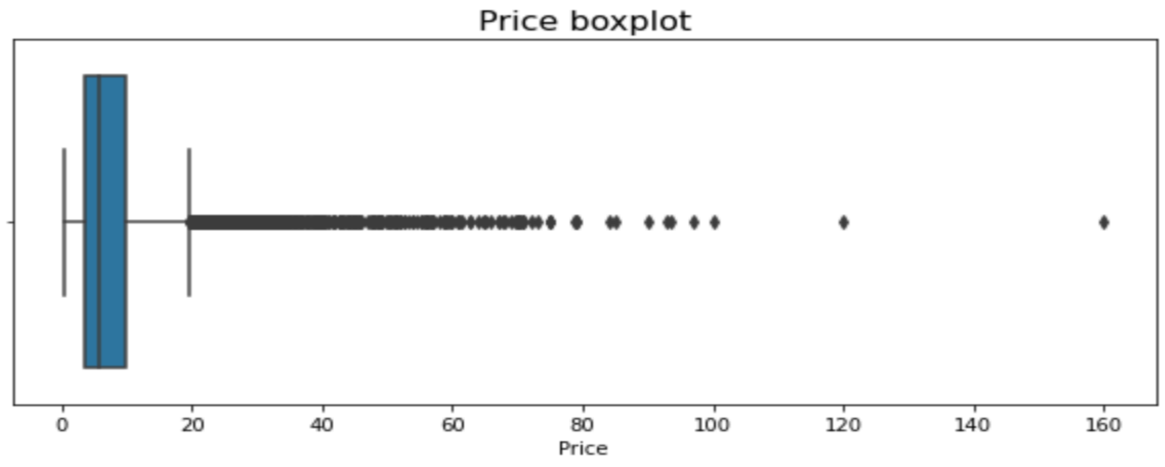
Mileage of cars can not be 0, so we should treat 0's as missing values. We will do it in the Feature Engineering part.

Univariate Analysis - Numerical Data

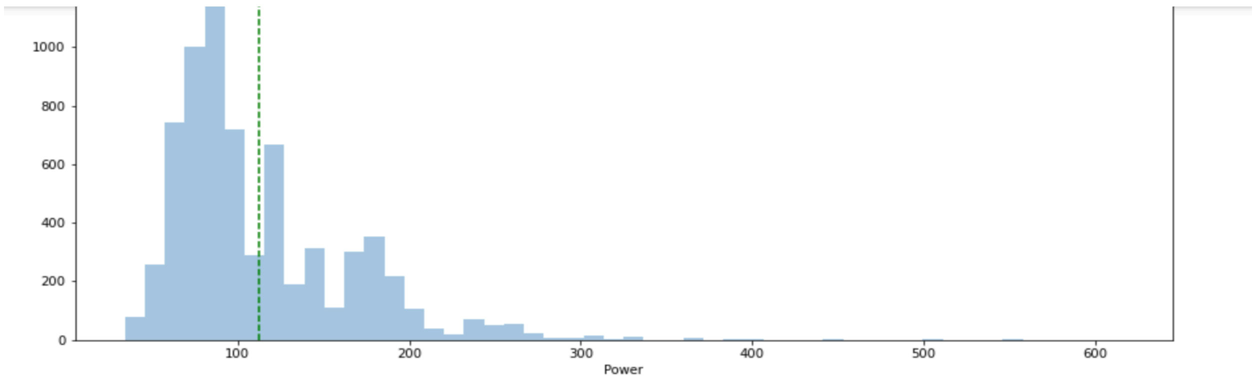
Observations and Insights:



Kilometers_driven is not normally distributed, it is right skewed. To solve this problem, the log transformation on kilometers_driven is applied when it has skewed distribution. And add the transformed variable into the dataset. You can name the variable as **' kilometers_driven_log'**.



Like Kilometers_Driven, the distribution of Price is also highly skewed, we can use log transformation on this column to see if that helps normalize the distribution. And add the transformed variable into the dataset. You can name the variable as **'price_log'**.



Like Kilometers_Driven, the distribution of Power is also highly skewed, we can use log transformation on this column to see if that helps normalize the distribution. And add the transformed variable into the dataset. You can name the variable as **'power_log'**.

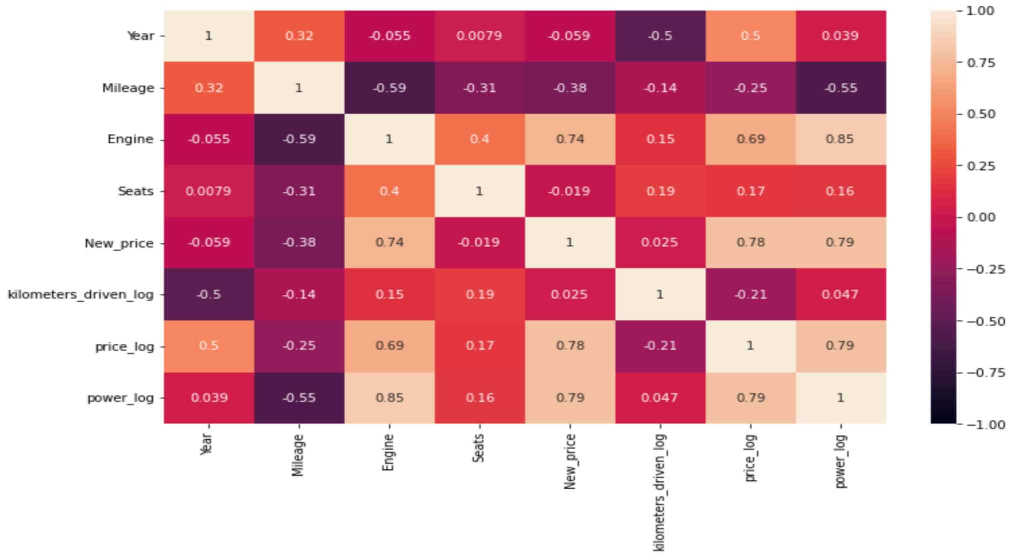
Univariate analysis - Categorical Data

Observations and Insights

Most of the owners within this dat aset are first-time owners. Most cars are manual transmission cars. Diesel and Pertol is the predominant fuel type. Most care in the data set are "newer" cars (2011 - 2015)

Bivariate Analysis

Observations and Insights from all plots:



	Year	Mileage	Engine	Seats	New_price	kilometers_driven_log	price_log	power_log
Year	1	0.32	-0.055	0.0079	-0.059	-0.5	0.5	0.039
Mileage	0.32	1	-0.59	-0.31	-0.38	-0.14	-0.25	-0.55
Engine	-0.055	-0.59	1	0.4	0.74	0.15	0.69	0.85
Seats	0.0079	-0.31	0.4	1	-0.019	0.19	0.17	0.16
New_price	-0.059	-0.38	0.74	-0.019	1	0.025	0.78	0.79
kilometers_driven_log	-0.5	-0.14	0.15	0.19	0.025	1	-0.21	0.047
price_log	0.5	-0.25	0.69	0.17	0.78	-0.21	1	0.79
power_log	0.039	-0.55	0.85	0.16	0.79	0.047	0.79	1

Positive relationships:

- price_log and year
- power and engine
- price_log and power_log

Coimbatore and Bangalore have on average the most expensive car prices

Diesel cars on average are the most expensive, electic cars although expensive are not significant in the data set. Automatic cars are the most expensive. Cars with only one previous owner are the most expensive

Feature Engineering¶

The Name column in the current format might not be very useful in our analysis. Since the name contains both the brand name and the model name of the vehicle, the column would have too many unique values to be useful in prediction. With 2041 unique names, car names are not going to be great predictors of the price in our current data. But we can process this column to extract important information for example brand name.

Missing value treatment

Missing values in Seats
extracted information from 'Name' column to impute missing values. Impute these missing values one by one, by taking median number of seats for the particular car, using the Brand and Model name.

Missing values for Mileage
replace zero to NAN as stated earlier. Impute missing Mileage. using median

Missing values for Engine
Impute these missing values one by one, by taking median number of engine for the particular car, using the Brand and Model name. Impute the rest missing Engine values using median

Missing values for Power
Impute these missing values one by one, by taking median number of power for the particular car, using the Brand and Model name. Impute the rest missing Power values using median

Missing values for New_price
Was dropped as there is no impact on the used price

Added the column age
Age based on current Year hence dropped Year column

Observation on missing values
New_Price, Price and Price_log is missing values as expected, Power_log is missing values as the log transformation needs to be done on the price column once again. To be addressed later

```
# lots of missing values, hence dropping also is not a factor is predicting the price or new cars
data.drop(columns='New_price', inplace=True)

# using Brand and Model, hence dropping name
data.drop(columns='Name', inplace=True)

# Age rather than Year
import datetime
curr_time = datetime.datetime.now()
data['Age'] = data['Year'].apply(lambda x : curr_time.year - x)

# using Age
data.drop(columns='Year', inplace=True)


# Fill Nan values for price, and reimpute Price_log and power_log
data['Price']=data.groupby(['Brand','Model'])['Price'].apply(lambda x:x.fillna(x.median()))
data.drop(columns='price_log', inplace=True)
data["price_log"] = np.log(data["Price"])
data.drop(columns='power_log', inplace=True)
data["power_log"] = np.log(data["Power"])

# drop the subset of data since these values are minnimal
data.dropna(subset=['Price', 'Mileage', 'Seats'], inplace=True)
```

Model Building

1. What we want to predict is the "Price". We will use the normalized version 'price_log' for modeling.
2. Before we proceed to the model, we'll have to encode categorical features. We will drop categorical features like Name.
3. We'll split the data into train and test, to be able to evaluate the model that we build on the train data.
4. Build Regression models using train data.
5. Evaluate the model performance.

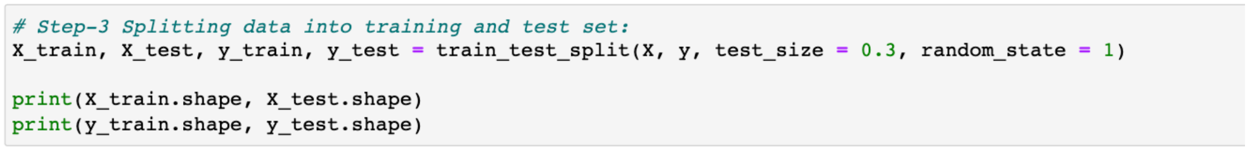
```
# Step-1
#Drop price and nad price_log since on y axis, Kilometers_Driven is already log transformed and so is Power,
#Model data needs additional insights from a domain expertise stand point

X = cars_data.drop(['Price','price_log','Kilometers_Driven','Power','Model'], axis = 1)
y = cars_data[["price_log","Price"]]
```

```
# Step-2 label encoder chosen
X = pd.get_dummies(X, drop_first = True)
X.head()
```

Encoding: In our dataset we have both categorical variables and numerical variables. To apply the ML models, we need to transform these categorical variables into numerical variables.

```
# Creating an instance of the MinMaxScaler
scaler = MinMaxScaler()

# Applying fit_transform on the training features data
X_scaled = scaler.fit_transform(X)

# The above scaler returns the data in array format, below we are converting it back to pandas DataFrame
X_scaled = pd.DataFrame(X_scaled, index = X.index, columns = X.columns)

X = X_scaled.copy()

X.head()
```

Normalization: The dataset is not normally distributed. We have used log transformation on price, power and kilometers_diven. Without normalization, the ML model will try to disregard coefficients of features that have low values because their impact will be so small compared to the big value. We have also Scaled the dataset using Min Max Scaler



```
# Step-3 Splitting data into training and test set:
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 1)

print(X_train.shape, X_test.shape)
print(y_train.shape, y_test.shape)
```

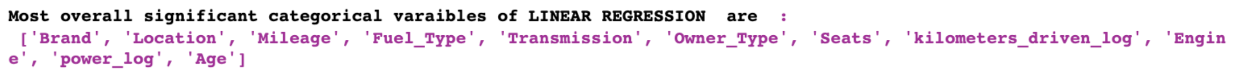
Train the data. In this process, 70% of the data can be split for the train data and 30% of the data can taken as test data.

Models For Regression Problems

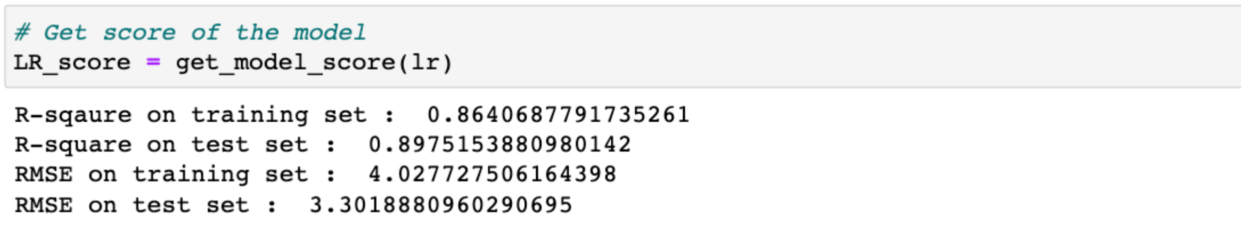
1) Linear Regression
2) Ridge / Lasso Regression
3) Decision Trees
4) Random Forest
5) XGBoost

Linear Regression

Observations from results



```
Most overall significant categorical varaibles of LINEAR REGRESSION  are  :
 ['Brand', 'Location', 'Mileage', 'Fuel_Type', 'Transmission', 'Owner_Type', 'Seats', 'kilometers_driven_log', 'Engin
e', 'power_log', 'Age']
```

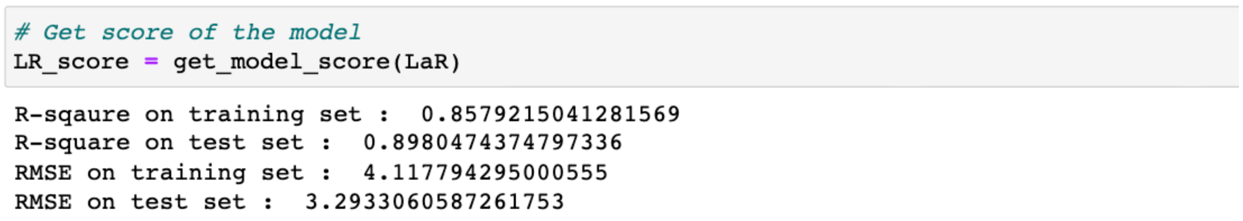


```
# Get score of the model
LR_score = get_model_score(lr)
```

```
R-sqaure on training set :  0.8640687791735261
R-square on test set :  0.8975153880980142
RMSE on training set :  4.027727506164398
RMSE on test set :  3.3018880960290695
```

Ridge / Lasso Regression

Observations from results:



```
# Get score of the model
LR_score = get_model_score(LaR)
```

```
R-sqaure on training set :  0.8579215041281569
R-square on test set :  0.8980474374797336
RMSE on training set :  4.117794295000555
RMSE on test set :  3.2933060587261753
```

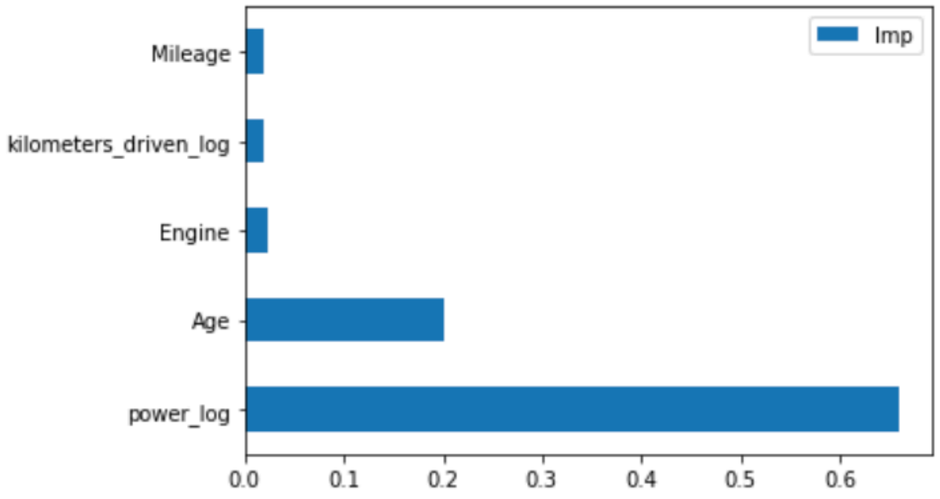
Decision Trees

Observations from results:

```
: # Get score of the model
Dtree_model = get_model_score(dtree)
```

```
R-sqaure on training set :  0.9999632188501091
R-square on test set :  0.823388407803888
RMSE on training set :  0.06625415561810441
RMSE on test set :  4.334536087358922
```

Observations from results: _____ Decision tree is also a high performing model



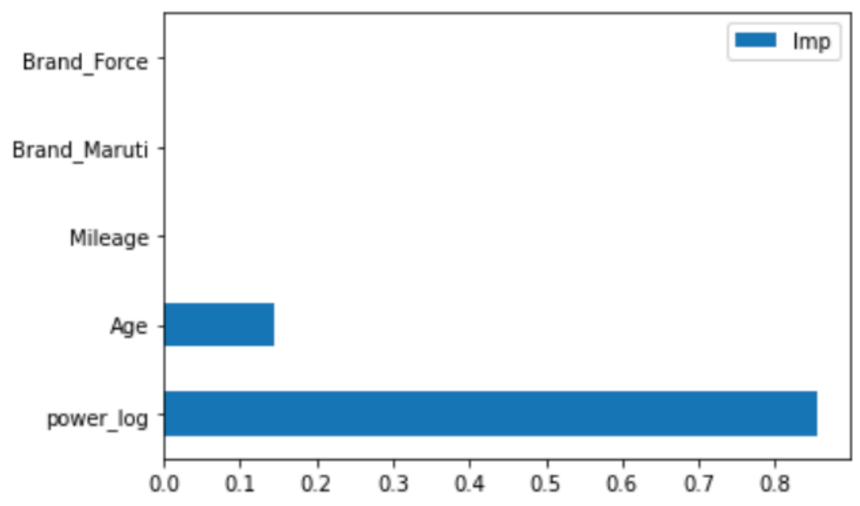
Random Forest

Observations and insights:

```
# Get score of the model
rf_model = get_model_score(regr)
```

```
R-sqaure on training set :  0.5311487222247311
R-square on test set :  0.5670889797291352
RMSE on training set :  7.480282921204805
RMSE on test set :  6.7862888490673
```

Observations and insights: _____ A very poor performing model



XGBoost

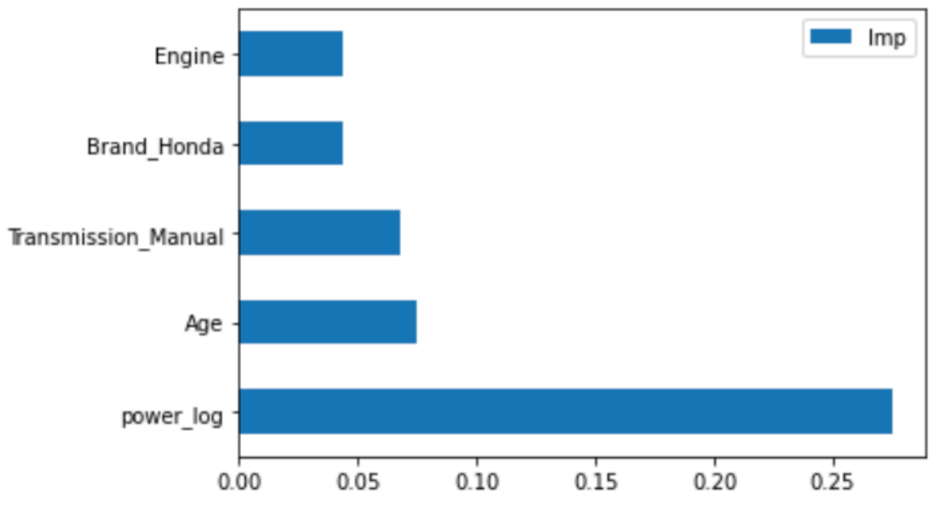
Observations and insights:

```
# Get score of the model
xg_reg_model = get_model_score(xg_reg)
```

```
R-sqaure on training set :  0.9854785810898358
R-square on test set :  0.9094489173095986
RMSE on training set :  1.3164519107931327
RMSE on test set :  3.1037007690432405
```

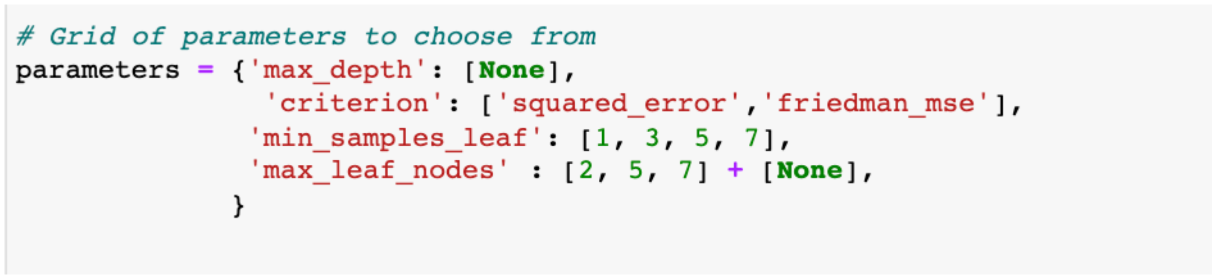
Observations and insights: _____ Xgboost seems to be the overall best performing model without overfitting

Feature Importance

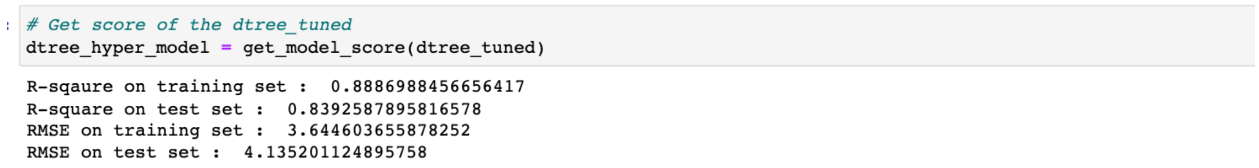


<u>Hyperparameter Tuning: Decision Tree</u>
Observations from results:

Parameters:



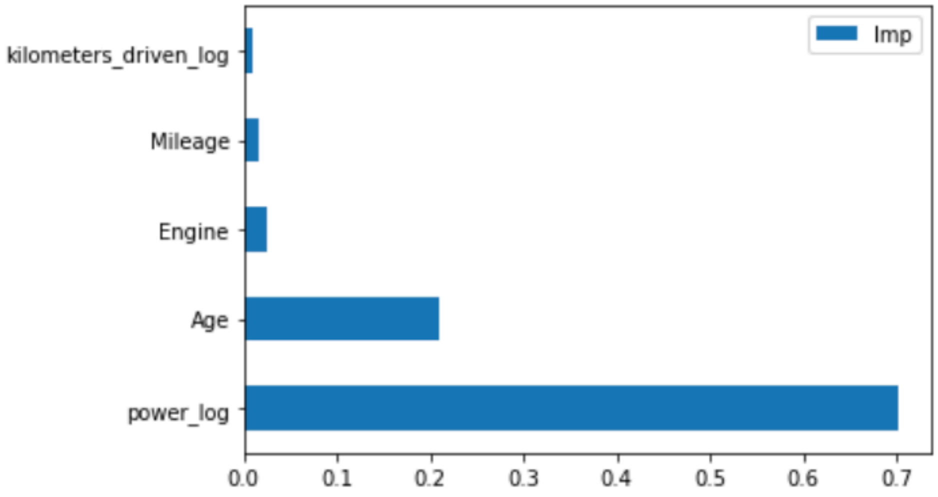
```python
# Grid of parameters to choose from
parameters = {'max_depth': [None],
               'criterion': ['squared_error','friedman_mse'],
              'min_samples_leaf': [1, 3, 5, 7],
              'max_leaf_nodes' : [2, 5, 7] + [None],
             }
```



```python
# Get score of the dtree_tuned
dtree_hyper_model = get_model_score(dtree_tuned)
```

```
R-sqaure on training set :  0.8886988456656417
R-square on test set :  0.8392587895816578
RMSE on training set :  3.644603655878252
RMSE on test set :  4.135201124895758
```

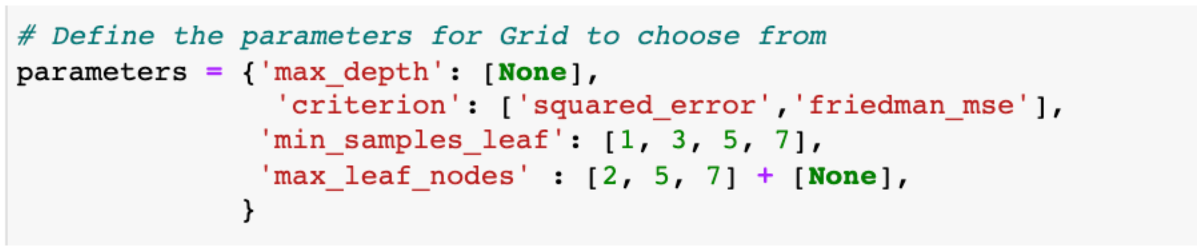
Observations and insights: _____ Hyperparameter tunning has significantly influenced the accuracy of the model



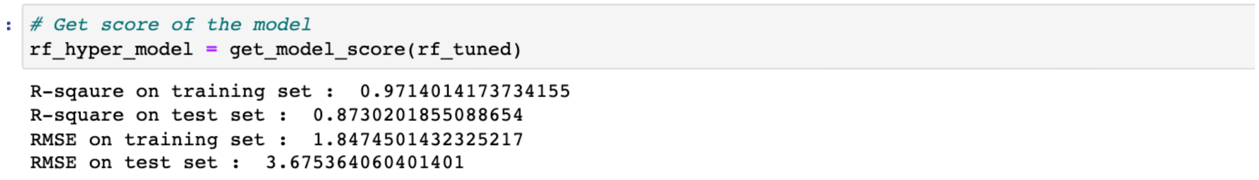
<u>Hyperparameter Tuning: Random Forest</u>

Observations from results:

Parameters:



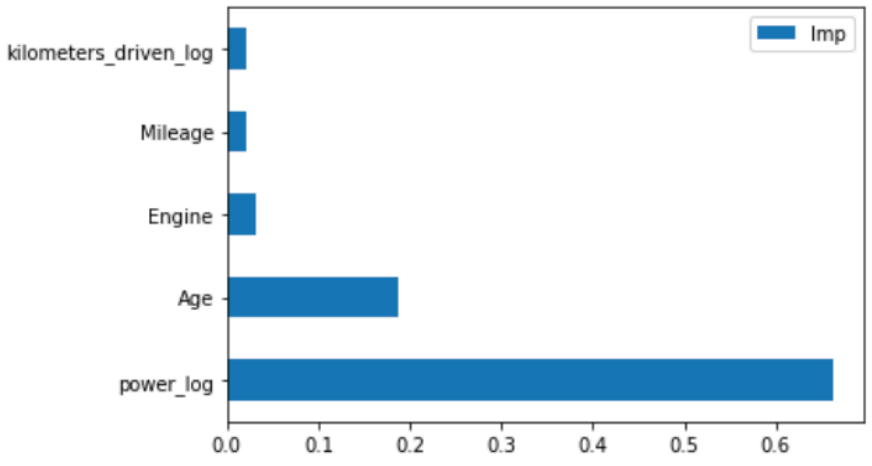
```python
# Define the parameters for Grid to choose from
parameters = {'max_depth': [None],
               'criterion': ['squared_error','friedman_mse'],
              'min_samples_leaf': [1, 3, 5, 7],
              'max_leaf_nodes' : [2, 5, 7] + [None],
             }
```



```python
# Get score of the model
rf_hyper_model = get_model_score(rf_tuned)
```

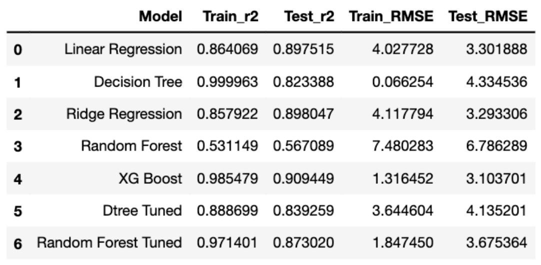
```
R-sqaure on training set :  0.9714014173734155
R-square on test set :  0.8730201855088654
RMSE on training set :  1.8474501432325217
RMSE on test set :  3.675364060401401
```

Observations and insights: _____ Hyper tuning has increated the performance and accuracy of the Random forest model



Conclusion:

The dataset was trained with the aforementioned regressors. To determine which regressors perform the best, they were assessed against a variety of performance indicators. Most important features that influence the target variable price seems to be power and age. Random Forest model performed much better using hyper parameter tuning



	Model	Train_r2	Test_r2	Train_RMSE	Test_RMSE
0	Linear Regression	0.864069	0.897515	4.027728	3.301888
1	Decision Tree	0.999963	0.823388	0.066254	4.334536
2	Ridge Regression	0.857922	0.898047	4.117794	3.293306
3	Random Forest	0.531149	0.567089	7.480283	6.786289
4	XG Boost	0.985479	0.909449	1.316452	3.103701
5	Dtree Tuned	0.888699	0.839259	3.644604	4.135201
6	Random Forest Tuned	0.971401	0.873020	1.847450	3.675364

Observations: _____ XG Boost seems to be the best overall model for accuracy and performance while Random Forest seems to have the worst performance

However after careful evaluation XGBoost is the best performing model based on the numbers shown

Improvements:

- More sophisticated way to calculate price especially relative to age
- More sophisticated way to identify the Model based on the Name column
- Taking into account “data leakeage” between test and train data
- Power_Log seems to be the most significant variable which would suggest a fabricated dataset
- To investigate how well the XGBoost model predicts price of a car, we can collect additional samples
- Label encoding the Model variable may be an option however this may prevent an unwanted bias