Regression Project

Mitchell Marfinetz

Context

There is a huge demand for used cars in the Indian Market today.

Unlike new cars, where price and supply are fairly deterministic and managed by OEMs, the used car market is a very different beast, with large uncertainties in both pricing and supply.

From the perspective of a seller, it is not an easy task to set the correct price of a used car.

Objectives

Come up with a pricing model that can effectively predict the price of used cars.

The model should help the business in devising profitable strategies using differential pricing.

Problem Statement and Formulation:

We are trying to solve a problem with the use case of regression, which uses training data to make predictions on unseen testing data.

Exploratory Data Analysis

Observations:

The median value for car year in the data set was

A 2014. The model year for cars in the data set

Ranges from 1996 - 2019.

Let us now explore the summary statistics of numerical variables

```
# Explore basic summary statistics of numeric variables. Hint: Use describe() method.
data.describe()
```

	Year	Kilometers_Driven	Mileage	Engine	Power	Seats	New_price	Price
count	7253.000000	7.253000e+03	7251.000000	7207.000000	7078.000000	7200.000000	1006.000000	6019.000000
mean	2013.365366	5.869906e+04	18.141580	1616.573470	112.765214	5.280417	22.779692	9.479468
std	3.254421	8.442772e+04	4.562197	595.285137	53.493553	0.809277	27.759344	11.187917
min	1996.000000	1.710000e+02	0.000000	72.000000	34.200000	2.000000	3.910000	0.440000
25%	2011.000000	3.400000e+04	15.170000	1198.000000	75.000000	5.000000	7.885000	3.500000
50%	2014.000000	5.341600e+04	18.160000	1493.000000	94.000000	5.000000	11.570000	5.640000
75%	2016.000000	7.300000e+04	21.100000	1968.000000	138.100000	5.000000	26.042500	9.950000
max	2019.000000	6.500000e+06	33.540000	5998.000000	616.000000	10.000000	375.000000	160.000000

Observations and Insights: The median value for car year in this data set was a 2014. The mode

Let us also explore the summary statistics of all categorical variables and the n

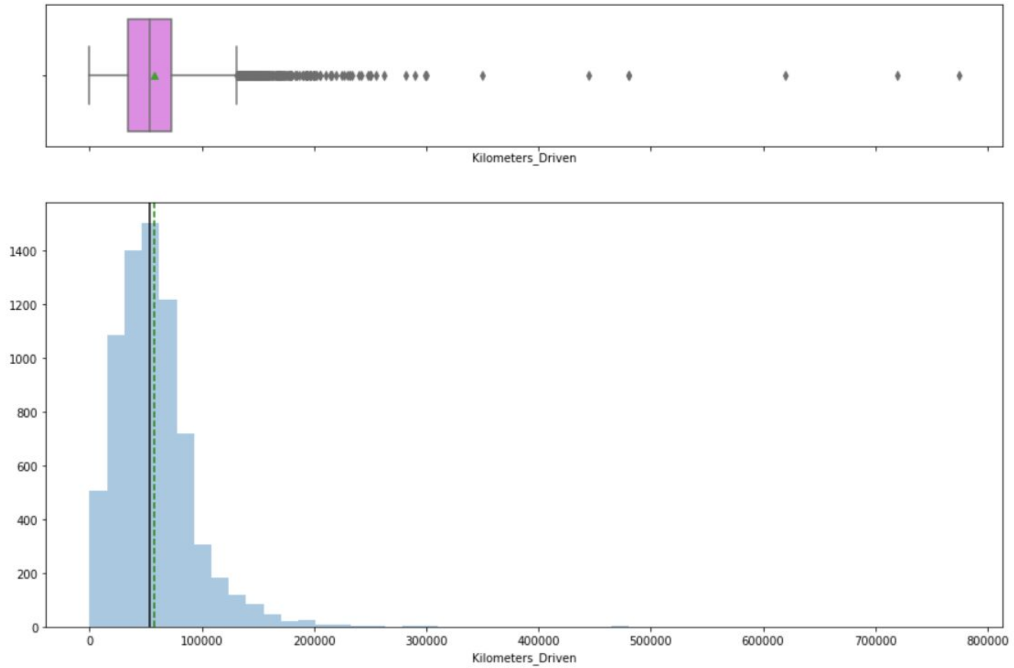
```
# Explore basic summary statistics of categorical variables. Hint: Use the argument include=['object']
data.describe(include = ['object'])
```

	Name	Location	Fuel_Type	Transmission	Owner_Type
count	7253	7253	7253	7253	7253
unique	2041	11	5	2	4
top	Mahindra XUV500 W8 2WD	Mumbai	Diesel	Manual	First
freq	55	949	3852	5204	5952

Univariate Analysis

Let us plot histogram and box-plot for the feature 'Kilometers_Driven' to understand the distribution and outliers, if any.

```
#Plot histogram and box-plot for 'Kilometers_Driven'
histogram_boxplot(data['Kilometers_Driven'])
```

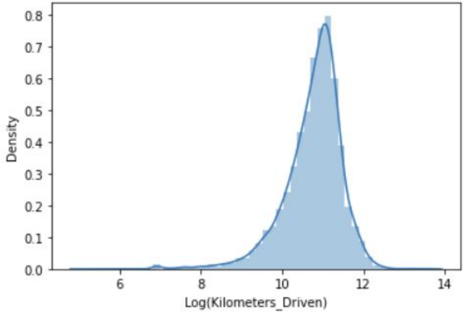


Univariate analysis

Applying log transformations

to normally distribute the data.

```
#Log transformation of the feature 'Kilometers_Driven'
sns.distplot(np.log(data["Kilometers_Driven"]), axlabel="Log(Kilometers_Driven)");
```



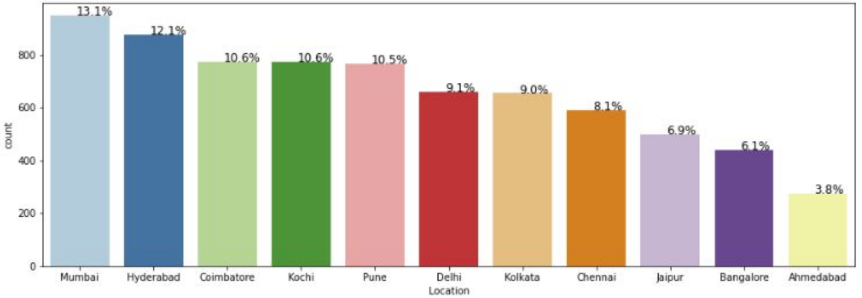
Observations and Insights: after applying the log transformation to the data it distributes normally.

```
### We can add a transformed kilometers_driven feature in data
data["kilometers_driven_log"] = np.log(data["Kilometers_Driven"])
```

Note: Like Kilometers_Driven, the distribution of Price is also highly skewed, we can use log transformation on this column to see if that helps no
can name the variable as **'price_log'**

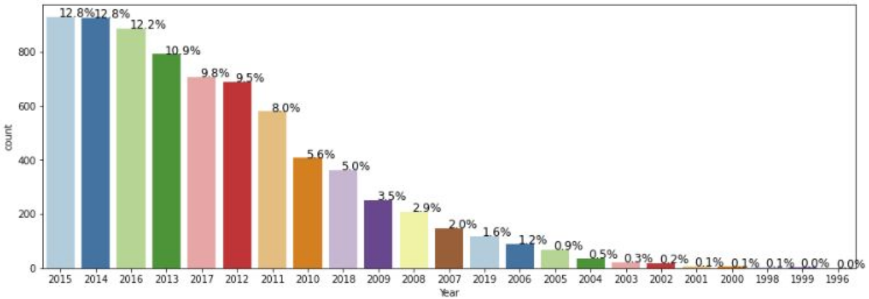
```
# Plot histogram and box-plot for 'Price'
histogram_boxplot(data['Price'])
```

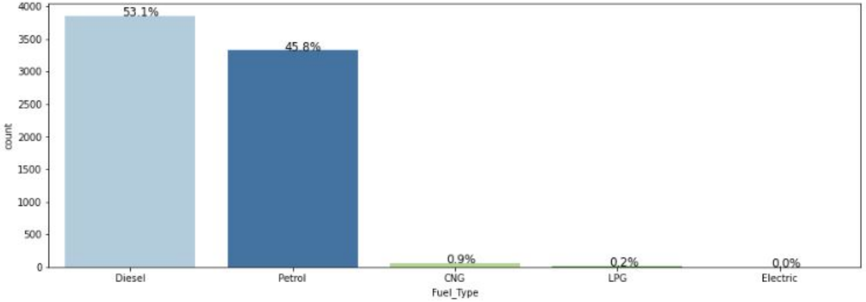
Categorical Data



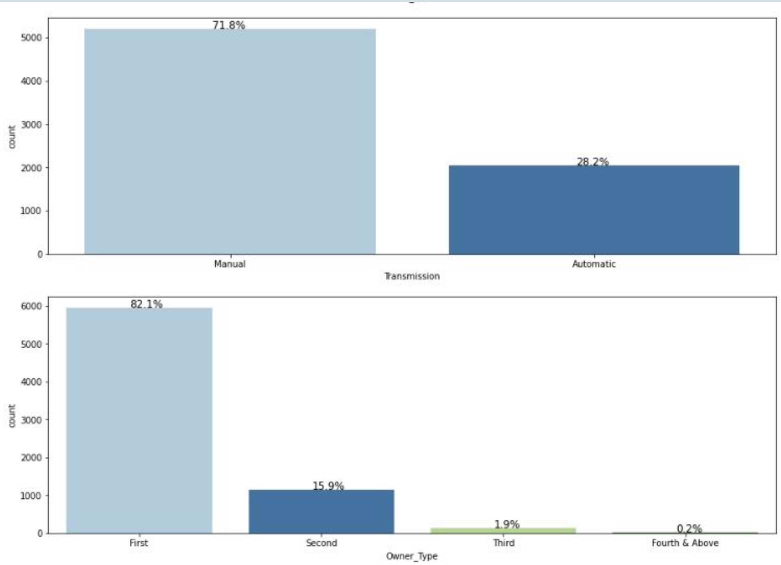
Note: Explore for other variables like Year, Fuel_Type, Transmission, Owner_Type`

```
perc_on_bar('Year')
perc_on_bar('Fuel_Type')
perc_on_bar('Transmission')
perc_on_bar('Owner_Type')
```





Categorical Data



Observations and Insights from all plots: The majority of cars from the database had one owner, was from 2012 or newer, was a manual transmission, and was from either mumbai, hyderbad, coimbatore, kochi, or pune.

The majority of cars from the database had one owner, was from 2012 or newer, was a manual transmission, and was from either mumbai, hyderbad, coimbatore, kochi, or pune.

```
#We can include the log transformation values and drop the original skewed data columns
plt.figure(figsize=(12, 7))
sns.heatmap(data.drop(['Kilometers_Driven', 'Price'],axis=1).corr(), annot = True, vmin = -1, vmax = 1)
plt.show()
```

Heat map

Engine and price have a positive correlation, as well as power and price, year and mileage have a slightly weaker correlation. Kilometers driven and year have a negative correlation, as well as engine and mileage and power and mileage.

Imputing Missing Values

```
# Impute missing Mileage. For example use can use median or any other methods.
data['Mileage'].fillna(data.Mileage.median(), inplace = True)
```

```
# Now check total number of missing values of the seat column to verify if they are impu
data['Mileage'].isnull().sum()
```

```
0
```

Missing values for Engine

```
data['Engine'].fillna(data.Engine.median(), inplace = True)
data['Engine'].isnull().sum()
```

```
0
```

Missing values for Power

```
data['Power'].fillna(data.Power.median(), inplace = True)
data['Power'].isnull().sum()
```

```
0
```

Missing values for New_price

```
data['New_price'].fillna(data.New_price.median(), inplace = True)
data['New_price'].isnull().sum()
```

```
0
```

Observations for missing values after imputing: Log_price still has missing values

```
data['Price_log'].fillna(data.Price_log.median(), inplace = True)
data['Price_log'].isnull().sum()
```

```
0
```

Model Building: Linear Regression

Most overall significant categorical variables of linear regression are: Model, New_price, Location, Fuel_Type, Engine, Owner_Type, Power, Transmission, kilometers_driven_log, Brand, Year

```
# import Linear Regression from sklearn
from sklearn.linear_model import LinearRegression
import sklearn.metrics as metrics
# Create a linear regression model
lr = LinearRegression()
# Fit linear regression model
lr.fit(X_train,y_train['log_price'])
```

```
LinearRegression()
```

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

```
R-sqaure on training set :  0.9399395114403349
R-square on test set :  0.8687919879680566
RMSE on training set :  2.738077260682623
RMSE on test set :  4.037008046078894
```

Ridge Regression

```
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.linear_model import RidgeClassifier

# Create a Ridge regression model
ridge = Ridge(alpha = 0.062, normalize = True, random_state = True)
# Fit Ridge regression model.
ridge.fit(X_train, y_train['log_price'])
```

```
Ridge(alpha=0.062, normalize=True, random_state=True)
```

```
# Get score of the model
pred = ridge.predict(X_test)
print(pd.Series(ridge.coef_, index = X.columns))
get_model_score(ridge)
```

```
Year          0.096633
Mileage      -0.002994
Engine        0.000092
Power         0.002419
Seats         0.025717
                 ...
Model_xylo   -0.432047
Model_yeti    0.229240
Model_z4      0.804154
Model_zen    -0.388004
Model_zest    0.054142
Length: 264, dtype: float64
R-sqaure on training set :  0.9329854032987761
R-square on test set :  0.8878171070038233
RMSE on training set :  2.8922509497597493
RMSE on test set :  3.7328690499677024
[0.9329854032987761,
 0.8878171070038233,
 2.8922509497597493,
 3.7328690499677024]
```

Decision Tree

Important Features:

Power

Year

Engine

Km_driven_log

```
# Create a decision tree regression model
dtree = tree.DecisionTreeRegressor(random_state=1, max_depth = 18)
# Fit decision tree regression model.
dtree.fit(X_train, y_train['log_price'])
```

```
DecisionTreeRegressor(max_depth=18, random_state=1)
```

```
get_model_score(dtree)
```

```
R-sqaure on training set :  0.9990762001530377
R-square on test set :  0.8045286900704055
RMSE on training set :  0.33957851999173394
RMSE on test set :  4.927435509872185
[0.9990762001530377,
 0.8045286900704055,
 0.33957851999173394,
 4.927435509872185]
```

```
print(pd.DataFrame(dtree.feature_importances_, columns = ["Imp"], i
```

```
                           Imp
Power                 0.608971
Year                  0.231887
Engine                0.045536
Mileage               0.016152
kilometers_driven_log 0.013779
...                        ...
Model_7               0.000000
Model_hexa            0.000000
Model_grande          0.000000
Model_800             0.000000
Model_zest            0.000000

[264 rows x 1 columns]
```

Decision Tree Tuning

```
# Choose the type of estimator.
dtree_tuned = DecisionTreeRegressor(random_state = 1)

# Grid of parameters to choose from.
dtree_tuned.get_params().keys()
# Check documentation for all the parametrs that the model takes and play with those.
parameters = {'random_state': [1],
              'min_samples_leaf': [5],
              'max_leaf_nodes' : [77],
              'max_features': [264],
              'max_depth': [None],

             }



# Type of scoring used to compare parameter combinations
scorer = metrics.make_scorer(recall_score, pos_label=1)

# Run the grid search
grid_obj = GridSearchCV(dtree_tuned, parameters, scoring=scorer, cv=5)
grid_obj = grid_obj.fit(X_train, y_train['log_price'])

# Set the clf to the best combination of parameters
dtree_tuned = grid_obj.best_estimator_

# Fit the best algorithm to the data.
dtree_tuned.fit(X_test, y_test['log_price'])
```

```
DecisionTreeRegressor(max_features=264, max_leaf_nodes=77, min_samples_leaf=5,
                      random_state=1)
```

Random Forest Regression

```
# Create a Randomforest regression model
rand_regr = RandomForestRegressor(random_state=1, n_estimators = 2000, max_depth = 15)
# Fit Randomforest regression model.
rand_regr.fit(X_train, y_train['log_price'])

# Get score of the model.
get_model_score(rand_regr)
```

```
R-sqaure on training set :  0.9755988992589022
R-square on test set :  0.8567551327677465
RMSE on training set :  1.7452432909450755
RMSE on test set :  4.218120400190681
[0.9755988992589022, 0.8567551327677465, 1.7452432909450755, 4.218120400190681]
```

```
# Print important features similar to decision trees
print (pd.DataFrame(rand_regr.feature_importances_, columns = ["Imp"], index = X_train.columns).sort_values(by = 'Imp', ascending = False).head(11))
```

```
                          Imp
Power                0.614083
Year                 0.232897
Engine               0.037036
kilometers_driven_log 0.015625
Mileage              0.013022
Transmission_Manual  0.005417
New_price            0.005215
Location_Kolkata     0.004848
Brand_tata           0.004409
Seats                0.003467
Model_rover          0.002873
```

Tuning The Decision Tree

```
# Choose the type of estimator.
dtree_tuned = DecisionTreeRegressor(random_state = 1)

# Grid of parameters to choose from.
dtree_tuned.get_params().keys()
# Check documentation for all the parametrs that the model takes and play with those.
parameters = {'random_state': [1],
              'min_samples_leaf': [5],
              'max_leaf_nodes' : [77],
              'max_features': [264],
              'max_depth': [None],

             }



# Type of scoring used to compare parameter combinations
scorer = metrics.make_scorer(recall_score, pos_label=1)

# Run the grid search
grid_obj = GridSearchCV(dtree_tuned, parameters, scoring=scorer, cv=5)
grid_obj = grid_obj.fit(X_train, y_train['log_price'])

# Set the clf to the best combination of parameters
dtree_tuned = grid_obj.best_estimator_

# Fit the best algorithm to the data.
dtree_tuned.fit(X_test, y_test['log_price'])
```

```
DecisionTreeRegressor(max_features=264, max_leaf_nodes=77, min_samples_leaf=5,
                      random_state=1)
```

Tuned Performance

The tuned model

Does not perform as well.

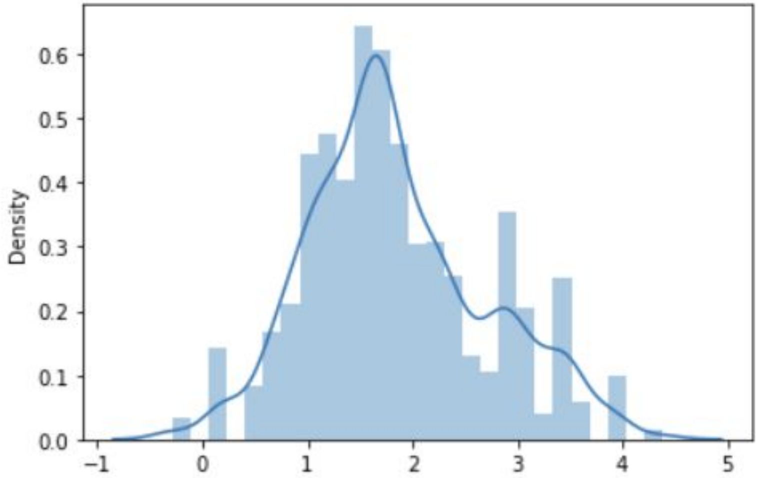
Other models should be

explored.

```
prediction = dtree_tuned.predict(X_test)
sns.distplot(prediction)

# Get score of the dtree_tuned
get_model_score(dtree_tuned)
```

```
R-sqaure on training set :  0.8121845371109644
R-square on test set :  0.8781119148775478
RMSE on training set :  4.841912542396703
RMSE on test set :  3.890989549955709
[0.8121845371109644, 0.8781119148775478, 4.841912542396703, 3.890989549955709]
```



Random Forest Tuning

The tree has is tuned to have

A minimum of 5 samples,

77 max leaf nodes, and 2000

estimators.

```
# Choose the type of Regressor.
rand_regr_tuned = RandomForestRegressor()

# Define the parameters for Grid to choose from
rand_regr.get_params().keys()

# Check documentation for all the parametrs that the model takes and play with those
parameters = {'random_state': [1],
              'min_samples_leaf': [5],
              'max_leaf_nodes' : [77],
              'max_features': [264],
              'n_estimators': [2000],
              'criterion': ['friedman_mse'],
             }


# Type of scoring used to compare parameter combinations
scorer = metrics.make_scorer(recall_score, pos_label=1)




# Run the grid search
grid_obj = GridSearchCV(rand_regr_tuned, parameters, scoring=scorer, cv=5)
grid_obj = grid_obj.fit(X_train, y_train)



# Set the clf to the best combination of parameters
rand_regr_tuned = grid_obj.best_estimator_

# Fit the best algorithm to the data.
rand_regr_tuned.fit(X_test, y_test['log_price'])
```

```
RandomForestRegressor(criterion='friedman_mse', max_features=264,
                      max_leaf_nodes=77, min_samples_leaf=5, n_estimators=2000,
                      random_state=1)
```

Tuned Performance

The model performs better in

Comparison to the decision tree,

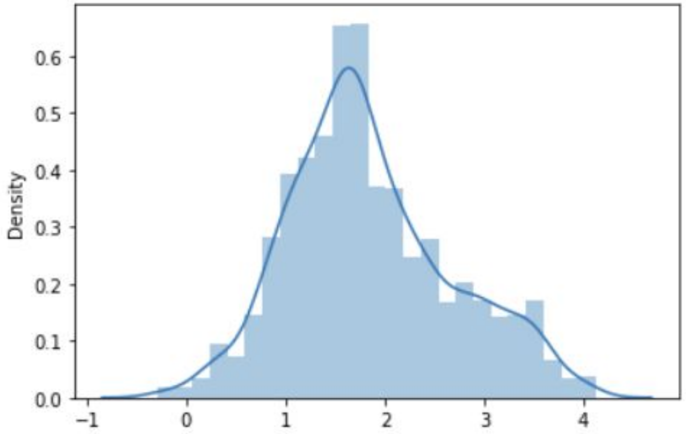
But not as well as the linear

regression.

```
prediction = rand_regr_tuned.predict(X_test)
sns.distplot(prediction)

# Get score of the model.
get_model_score(rand_regr_tuned)
```

```
R-sqaure on training set :  0.8375219020376733
R-square on test set :  0.8730932769670771
RMSE on training set :  4.5034845939419945
RMSE on test set :  3.970285633601398
[0.8375219020376733, 0.8730932769670771, 4.5034845939419945, 3.970285633601398]
```



Overall Model Performance

	Model	Train_r2	Test_r2	Train_RMSE	Test_RMSE
0	lr	0.939940	0.868792	2.738077	4.037008
1	dtree	0.999076	0.804529	0.339579	4.927436
2	rand_regr	0.975599	0.856755	1.745243	4.218120
3	dtree_tuned	0.812185	0.878112	4.841913	3.890990
4	rand_regr_tuned	0.837522	0.873093	4.503485	3.970286

As you can see, the linear regression model performs most accurately compared to all other models.

Proposal and Recommendations

I propose to adopt the linear regression model. I purpose this because it performs the best out of all the other models. it has similar r2 values compared to the other models, while maintaining a lower rmse as well. Also this is helpful because linear regression results in an output of a value which is useful for pricing.

Recommendations: Businesses looking to participate in the used car market of India, should implement a linear regression model in order to effectively and accurate predict the price of used car.

Important Variables to note are power, year, engine, and km driven. The linear regression model is the most accurately performing model on the given data, as demonstrated above.