CAR PRICE PREDICTION PROJECT

Submitted by:

SURAJ CHAKRABORTY

ACKNOWLEDGMENT

First and foremost, praises and thanks to the God, the Almighty, for his Shower of blessings throughout my project work to complete the project successfully.

I would like to express my deep and sincere gratitude to my SME Mr. Sajid Choudhary of Flip Robo technology, for giving me the opportunity to make this project and providing guidance on how to make the project. It was a great privilege to work under his mentorship. I would also like to thank my datatrained institute mentors for giving me all the knowledge in data science, which I am able to implement now for making the project. The projects which I had made while learning with datatrained helped me a lot in providing reference to make this project. The files attached with the data file of this project also helped me to get a better understanding of the project and steps that I needed to follow. Also the internet played a key role in helping me make the project as whenever I got stuck, I looked up on the internet for possible solutions that I could use and also to get rid of errors.

I am extremely grateful to my mother for her love, prayers and care and sacrifices for educating and preparing me for my future.

INTRODUCTION

- ## Business Problem Framing

  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. The client is facing problems with the previous car valuation model, this is because of the impact of covid 19 which has changed the market for nearly all businesses. Now, using new data and machine learning we need to deliver a fresh new valuation model to our client so that our client can resume business as before in the new market.

  This project will help our client to get a better understanding of the new trends in the automobile business and the factors that contribute to the price of the vehicle.

- ## Conceptual Background of the Domain Problem

  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. We have to make car price valuation model.

  The project needs to be complete in two phases, the data collection phase and the model building phase. In the data collection phase, we have to scrape at least 5000 records of data. In this section we need to scrape the data of different used cars from different websites.

  After collecting the data, we need to build a machine learning model and before model building we need to do all data pre-processing steps then try different models with different hyper parameters and select the best model.

- ## Review of Literature

  I have received lot of important information regarding the topic form the internet. I also found some articles regarding the topic in different websites like towardsdatascience.com, kaggle.com, medium.com etc. Those articles helped me get a better understanding of the use case and how to approach and deal with the problems. And also the data description provided to us helped me understand the steps I needed to follow in order to complete the project and get a better model as result.

- ## Motivation for the Problem Undertaken

  The objective is to provide our client with the best model, which can then be used to understand how exactly the prices vary with the features. They can accordingly manipulate the strategy of the firm. And further, the model will be a good way for the management to understand the pricing dynamics of the new market.

  What motivated me to take the project is the fact that the project is a fully fledged project that involves both scraping the data and creating the ML model using the scraped data. And also the fact that we get to decide which features we want and don't want and scrape according to that, got us more involved in the project and gave us more knowledge on feature selection before scraping data.

Analytical Problem Framing

- Mathematical/ Analytical Modeling of the Problem

  In this project, we build a regression model which can be used to predict the price of the used cars based on the features provided. We performed different steps including exploratory data analysis, data cleaning, data pre-processing and model building. We used different models for training and with the Machine Learning models we determined the optimal values of Hyper Parameters and selected the final model based on their scores.

- Data Sources and their formats

  The project consists of two phases, one for scraping the data and the other for building the model. In the first phase we had to scrape the data for different used cars from different websites such as cardekho, cars24, carwale, etc. and hence these websites acted as the data sources for this project. We preferred using multiple websites for collecting data as it would help our model in avoiding over fitting. We scraped different features of the cars, some of which were object data and some numeric i.e. integer format and the target i.e. price was of integer data type which contained the price of the cars.

- Data Pre-processing Done

  - At first we scraped the data using a jupyter notebook and saved the dataset in an excel sheet. And then we loaded the dataset into a new jupyter notebook.

  - We dropped the unwanted column from the dataset which was the index column that had no role to play.

```
df.drop(['Unnamed: 0'], axis=1, inplace=True)
```

- Checked for missing values in the dataset and found two columns containing missing data. In the first column, only one missing value was present, hence we dropped the record and in the other column, we replaced the missing values with the most frequent value as it was of object data type.

```
df[df['Location'].isnull()]
```

	Year of manufacture	Company	Model	Variant	Fuel type	Location	Kilometers driven	No. of Owners	Transmission	Mileage	Seats	Price
3621	2017	Maruti	Wagon R 1.0	VXI	Petrol	NaN	35,421 km	1st Owner	MANUAL	20.5	5	4,07,299

Only one record can be seen having null value for location column, hence we can drop the record.

```
df.drop([3621], inplace = True )
```

Screenshot of the code for first column

```
: # Replacing the missing values

  from sklearn.impute import SimpleImputer

  imp=SimpleImputer(strategy= 'most_frequent')
  df['Transmission']= imp.fit_transform(df['Transmission'].values.reshape(-1,1))
```

Screenshot of the code for second column

- We then cleaned the data for all the columns in the dataset. In the company column, there we few same values but written differently, we fixed that. We also found some data slipped into another column, we fixed that too.

```
df["Company"].replace("Maruti", "Maruti Suzuki", inplace=True)
```

Screenshot of the code

```
: df["Company"].replace("New", "Skoda", inplace=True)
  df["Model"].replace("Skoda", "Rapid", inplace=True)
  df["Variant"].replace("Rapid 1.0 TSI Ambition AT", "1.0 TSI Ambition AT", inplace=True)
```

```
: df["Company"].replace("Land", "Land Rover", inplace=True)
  df.at[[408,451],'Model']= 'Range Rover'
  df.at[[1827,1854],'Model']= 'Freelander'
  df["Variant"].replace("Range Rover Evoque HSE Dynamic", "Evoque HSE Dynamic", inplace=True)
  df["Variant"].replace("Range Rover Evoque 2.0 TD4 SE Dynamic", "Evoque 2.0 TD4 SE Dynamic", inplace=True)
  df["Variant"].replace("Freelander 2 SE", "2 SE", inplace=True)
```

Screenshot of the code

- In the Variant column, we checked the number of unique values and found it to be very high, hence we dropped the column as it may affect the model performance.

```
df['Variant'].nunique()
```

```
1767
```

We can see that the variant column contains nearly 1800 unique values. This can have a negative effect on our model. Hence we will remove the column.

```
df.drop(['Variant'], axis=1, inplace=True)
```

Screenshot of the code

- In the fuel type column, we found some categories having only one or two records, we merged those categories into one as the data showed imbalance.

```
# Merging the multiple fuel types to hybrid.

df["Fuel type"].replace("LPG + Lpg", "Hybrid", inplace=True)
df["Fuel type"].replace("Petrol + CNG", "Hybrid", inplace=True)
df["Fuel type"].replace("Petrol + LPG", "Hybrid", inplace=True)
df["Fuel type"].replace("LPG", "Hybrid", inplace=True)
df["Fuel type"].replace("Diesel + Lpg", "Hybrid", inplace=True)
```

Screenshot of the code

- In the location column, some values were same but written in different cases, hence we converted all the values to lower case and also merged two same values with differently called names.

```
# Converting all the Location data to lowercase.

df['Location']=df['Location'].str.lower()
```

```
df['Location'].unique()
```

```
array(['ahmedabad', 'bangalore', 'chennai', 'delhi-ncr', 'gurgaon',
       'hyderabad', 'jaipur', 'kolkata', 'mumbai', 'new-delhi', 'noida',
       'pune', 'delhi', 'bengaluru', 'navi mumbai', 'chandigarh'],
      dtype=object)
```

Again we can see there are two unique values for bangalore. lets correct it.

```
# Replacing and making two unique values as one.

df["Location"].replace("bengaluru", "bangalore", inplace=True)
```

Screenshot of the code

- In the kilometres column, there were many unwanted strings for which the data was of object type. We removed the unwanted strings and then converted the column to integer.

```
# Removing the unwanted strings.

df['Kilometers driven']= df['Kilometers driven'].str.replace(' Kms', '') \
    .str.replace(' kms', '') \
        .str.replace(',', '') \
            .str.replace(' km', '')
```

```
df['Kilometers driven']= df['Kilometers driven'].astype(int)
```

Screenshot of the code

- In the num of owners columns too there were unwanted strings which made the column as object type and some numeric values were written in string format. We removed the unwanted strings, changes the numeric values from strings to numbers and converted the column to integer.

```
df['No. of Owners']= df['No. of Owners'].str.replace(' Owner', '')
```

```
# Converting all the values to the same format.

replacement = {
    "1st": "1",
    "2nd": "2",
    "3rd": "3",
    "4th": "4",
    "First": "1",
    "Second": "2",
    "Third": "3",
    "Fourth": "4",
    "Test Drive Car": "0",
    "UnRegistered Car": "0"
}

df['No. of Owners'].replace(replacement, inplace=True)
```

```
# Converting the datatype to integer.

df['No. of Owners']= df['No. of Owners'].astype(int)
```

Screenshot of the code

- In the transmission column, there were some wrong inputs in the data, we replaced them with null and then filled the null values with the most frequent value.

```
# Replacing the wrong inputs with nan value.

replacement = {
    "KA01": np.nan,"KA53": np.nan,"KA02": np.nan,"KA04": np.nan,"KA50": np.nan,"DL9C": np.nan,"DL6C": np.nan,"DL4C": np.nan,
    "HR51": np.nan,"DL3C": np.nan,"HR26": np.nan,"HR87": np.nan,"UP14": np.nan,"AP23": np.nan,"TS08": np.nan,"TS10": np.nan,"AP3
    "RJ14": np.nan,"RJ45": np.nan,"RJ19": np.nan,"MH01": np.nan,"MH04": np.nan,"MH02": np.nan,"MH11": np.nan,"MH14": np.nan,
    "MH12": np.nan,"MH13": np.nan,"MANUAL": "Manual","AUTOMATIC": 'Automatic'
}

df['Transmission'].replace(replacement, inplace=True)
```

```
df['Transmission']= imp.fit_transform(df['Transmission'].values.reshape(-1,1))
```

Screenshot of the code

- In the mileage column, there were some unwanted strings and also some missing values denoted by a dash. We removed the unwanted strings and filled the missing data with the median of the values and rounded off all the data.

```
# Removing the unwanted strings.

df['Mileage']= df['Mileage'].str.replace(' kmpl', '')
df['Mileage']= df['Mileage'].str.replace(' km/kg', '')
```

Screenshot of the code

```
# Replacing the empty and unwanted data with null value.

df["Mileage"].replace("-", np.nan, inplace=True)
df["Mileage"].replace("2393 CC", np.nan, inplace=True)
df["Mileage"].replace("0.0", np.nan, inplace=True)
```

Here we can see the the data in the column has some skewness present, hence we wil be replacing the null values with their median.

```
# Replacing the null values with their median.

imp=SimpleImputer(strategy= 'median')
df['Mileage']= imp.fit_transform(df['Mileage'].values.reshape(-1,1))
```

```
# Rounding off the values.

df['Mileage']= round(df["Mileage"],2)
```

Screenshot of the code

- In the seats column as well, there were unwanted strings and some missing values denoted by a dash. We removed the

strings, replaced the missing data with the most frequent value and then converted the column to integer.

```
# Removing the unwanted strings.

df['Seats']= df['Seats'].str.replace(' Person', '')
df['Seats']= df['Seats'].str.replace('Person', '')
df['Seats']= df['Seats'].str.replace('7 & ', '')

# The column contains some missing values, we are converting them to nan.
df['Seats']= df['Seats'].replace('-', np.nan)
```

```
# Replacing the null values with the most frequent value.

imp=SimpleImputer(strategy= 'most_frequent')
df['Seats']= imp.fit_transform(df['Seats'].values.reshape(-1,1))
```

```
# Converting the datatype to integer.

df['Seats']= df['Seats'].astype(int)
```

Screenshot of the code

- In our target column i.e. price, we had punctuations and also many records contain data in word form. We converted all the records to the same format and then removed the punctuations and finally converted the column to integer data type.

```
# Removing the unwanted character.

df['Price']= df['Price'].str.replace('*', '')
```

```
df['Price']= df['Price'].apply(lambda x: x.replace(" Lakh", '0000') if len(x.split('.')[-1])== 6 else x.replace(" Lakh", '000') i
                    else x)
```

Here we are converting the price in lakhs given in alphabetic form into numeric form.

```
df['Price']= df['Price'].apply(lambda x: x.replace(" Crore", '000000') if len(x.split('.')[-1])== 7 else x.replace(" Crore", '00
                    else x)
```

Here we are converting the price in crores given in alphabetic form into numeric form.

```
df['Price']= df['Price'].apply(lambda x: x.replace("000", '00000') if len(x)< 6 else x)
```

Screenshot of the code

```
# Removing the now unwanted characters.

df['Price']= df['Price'].str.replace(',', '')
df['Price']= df['Price'].str.replace('.', '')
```

```
# Converting the datatype to integer.

df['Price']= df['Price'].astype(int)
```

Screenshot of the code

- Encoded the categorical object type data in the dataset.

```
# Encoding

from sklearn.preprocessing import LabelEncoder

enc= LabelEncoder()
for i in df.columns:
    if df[i].dtypes== 'O':
        df[i]= enc.fit_transform(df[i].values.reshape(-1,1))

# encoding the year column as well.

df['Year of manufacture']= enc.fit_transform(df['Year of manufacture'].values.reshape(-1,1))
```

Screenshot of the code

- Visualized the distribution and count of data values in different features of the dataset.

```
df1.plot(kind='density', subplots= True, layout=(4,3),sharex=False, legend=True, figsize=[20,10])
plt.show
```

Screenshot of one of the code

- Checked the correlation among all the columns in the dataset and also checked the correlation of all the features with the target using different visualization techniques.

```
cor= df.corr()
cor
```

Screenshot of one of the code

```
plt.figure(figsize=(15,5))
cor['Price'].sort_values(ascending=False).drop(['Price']).plot(kind='bar')
plt.xlabel('Feature')
plt.ylabel('Correlation with target')
plt.title('Correlation')
plt.show()
```

Screenshot of the code

- Checked for outliers in the numeric columns in the dataset. We found few outliers but they were very close to the threshold, hence to did not remove them.

```
sns.boxplot(df['Kilometers driven']);
```

Screenshot of the code

- Treated skewness in the dataset. We found some skewness present in the continuous numeric data, we them minimised the skewness using power transformation.

```
from sklearn.preprocessing import power_transform

df['Kilometers driven']=power_transform(df['Kilometers driven'].values.reshape(-1,1))
```

Screenshot of the code

- We then split the dataset into features and target and then scaled the features using standard scaler.

```
# First lets split the data into target and features.

x= df.drop(['Price','No. of Owners'], axis=1) # By hit and trial we found the 'no of owners' column effecting the model performar
y= df['Price']
```

```
from sklearn.preprocessing import StandardScaler

sc=StandardScaler()
x=pd.DataFrame(sc.fit_transform(x), columns=x.columns)
x
```

Screenshot of the code

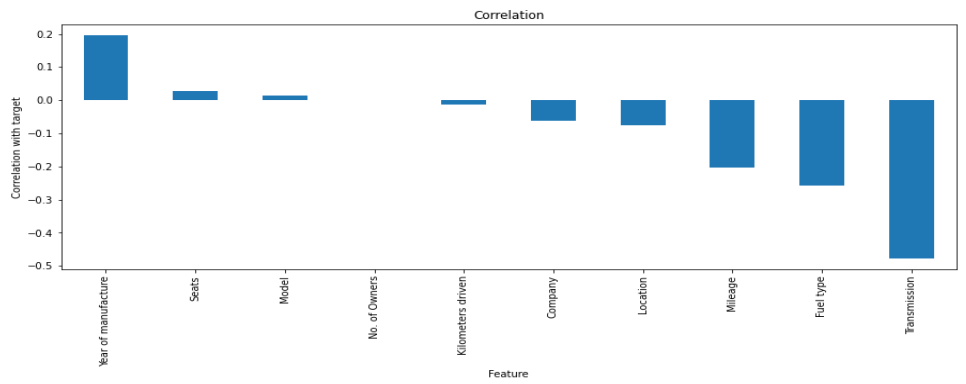
- Data Inputs- Logic- Output Relationships

  We checked the data distribution of the numeric independent columns and the counts of categorical columns in the dataset and found some numeric columns being normally distributed and some having skewness in the data. We then checked the correlation among all the columns of the dataset and then visualized it.

  We also visualized the correlation of independent columns with the target separately, where we found the feature 'Year of manufacture' showing a positive correlation with the target, which meant that more new the car, the more value it has. Also we found the features 'Transmission, 'Fuel type', 'Mileage' showing a high negative correlation with the target which meant that the features impact the price significantly.

  But the feature 'No of owners' showed us no correlation with the target value, a near zero correlation was found between them.

```
plt.figure(figsize=(15,5))
cor['Price'].sort_values(ascending=False).drop(['Price']).plot(kind='bar')
plt.xlabel('Feature')
plt.ylabel('Correlation with target')
plt.title('Correlation')
plt.show()
```



Screenshot of correlation of target with all features

- Hardware and Software Requirements and Tools Used

  Hardware required: Minimum Intel i3 processor with 8 GB of RAM.

  Operating system: Windows 7, 8, or 10 or LINUX or Mac OS

  Tools used:

  - Jupyter notebook: For the coding and creating the project.

- Microsoft edge: For hosting.
- Chrome driver: For scrapping data from websites.
- Microsoft word: For making the project report.
- Microsoft PowerPoint: For making the PPT.

Libraries & Packages used:

- Numpy: Used to perform log transformation on the dataset for treating the skewness. Also used to find the exponent of the final predicted result to inverse the logarithm and round up the values.
- Pandas: Used to load the dataset into the jupyter notebook and controlling the viewing pattern of the data.
- Selenium: Used for controlling web browser through programs and performing browser automation.
    - Webdriver: Used for initializing the web driver to a variable.
    - Exceptions: Used to deal with any unwanted exception during scraping.
    - Keys: To perform click operation of keyboard keys.
- Time: Used to halt the scrapping process for a certain time to let a page load successfully
- Matplotlib: Uses to visualize and plot the distribution of data in columns as well as the relation between different columns.
- Seaborn: Used to visualize the distribution and correlation between data using different types of figures and graphs.
- Sklearn: Used to perform different operations using different modules in it.
    - r2_score: To measure how close the predicted data are to the actual data.
    - mean_squared_error: Gives the average squared difference between the estimated values and true value.
    - mean_absolute_error: This measure the absolute average distance between the real data and the predicted data.

        - SimpleImputer: Used to replace missing values present in the dataset.
        - StandardScaler: Used to scale the dataset.
        - LabelEncoder: Used for encoding the object type data.
        - power_transform: Used to reduce the skewness in the columns.
        - Train_test_split: To split the data in train and test data.
        - Using sklearn we also imported different models for the training the data e.g. Linear Regression, Lasso etc.
        - Cross_val_score: Used to check the cross validation scores for different models.
        - RandomizedSearchCV: Used to perform hyper-parameter tuning to find the best parameters for the models.
    - xgboost: Used to call the XGBRegressor model for training.
    - Joblib: Used to save the model that we created in .pkl format.

Model/s Development and Evaluation

- Identification of possible problem-solving approaches (methods)

  The approach that we followed is:

  - Loaded the dataset into the jupyter notebook
  - Performed extensive EDA to get better insights of the dataset.
  - Checked for missing values and treated them accordingly.
  - Performed data cleaning by cleaning and converting all columns to the format they can be used for model building.
  - Encoded the categorical types of data.
  - Performed visualization to check the distribution and counts of the data.
  - Checked the correlation among all the columns in the dataset and performed visualization to see the relationship between the features and the target column.
  - Checked for outliers in the dataset.
  - Handled the skewness in the dataset.
  - Split the dataset into target and features.
  - Performed scaling on the features.
  - Found the best random state and preformed train test split using it.
  - Trained different models.
  - Checked CV score for all the models for any overfitting or underfitting.
  - Hyper-parameter tuned the better performing models to find the best parameters for those models.
  - Trained the models using those parameters found to be performing best.
  - Saved the best performing model as the final model.

- Testing of Identified Approaches (Algorithms)

  The algorithms that we used are:

  - Linear Regression
  - Decision Tree Regressor
  - Random Forest Regressor
  - Support Vector Regressor
  - Lasso regularization
  - XGBRegressor

- Run and Evaluate selected models

  - Linear Regression: It measures the relationship between continuous numeric dependent variable and the independent variables by estimating probabilities.

```
lr.fit(x_train,y_train)
predlr= lr.predict(x_test)

print('Score: ',lr.score(x_train,y_train))
print('r2 score: ', r2_score(y_test,predlr))
print('Mean absolute error:', mean_absolute_error(y_test,predlr))
print('Mean squared error:', mean_squared_error(y_test,predlr))
print('Root mean squared error:', np.sqrt(mean_squared_error(y_test,predlr)))
```

```
Score:  0.297691391667149
r2 score:  0.4164264596460484
Mean absolute error: 512584.93508970994
Mean squared error: 897283410475.0093
Root mean squared error: 947250.4475982101
```

Screenshot of the code

Using linear regression, we found the training score to be 30%, the R2 score to be 42% and the errors being very high.

  - Decision tree regressor: Builds regression models in the form of a tree structure. It breaks down the dataset into smaller subsets.

```
from sklearn.tree import DecisionTreeRegressor

dt= DecisionTreeRegressor()
dt.fit(x_train,y_train)
preddt= dt.predict(x_test)

print('Score: ',dt.score(x_train,y_train))
print('r2 score: ', r2_score(y_test,preddt))
print('Mean absolute error:', mean_absolute_error(y_test,preddt))
print('Mean squared error:', mean_squared_error(y_test,preddt))
print('Root mean squared error:', np.sqrt(mean_squared_error(y_test,preddt)))
```

```
Score:  0.9999959689917485
r2 score:  0.4518090849178026
Mean absolute error: 247367.3648757017
Mean squared error: 842880253923.1499
Root mean squared error: 918085.1016780252
```

Screenshot of the code

Using Decision tree regressor, we found the training score to be 99%, the R2 score to be 45%, and high errors.

- Random forest regressor: It builds multiple decision trees and merges them together to get a more accurate prediction.

```
from sklearn.ensemble import RandomForestRegressor

fr=RandomForestRegressor()
fr.fit(x_train,y_train)
predfr=fr.predict(x_test)

print('Score: ',fr.score(x_train,y_train))
print('r2 score: ', r2_score(y_test,predfr))
print('Mean absolute error:', mean_absolute_error(y_test,predfr))
print('Mean squared error:', mean_squared_error(y_test,predfr))
print('Root mean squared error:', np.sqrt(mean_squared_error(y_test,predfr)))
```

```
Score:  0.9466800819880374
r2 score:  0.6746131683805578
Mean absolute error: 253163.77121357928
Mean squared error: 500304050492.19196
Root mean squared error: 707321.7446764888
```

Screenshot of the code

Using Random forest regressor, we found the training score to be 94%, the R2 score to be 67%, and comparative less errors.

- Support vector regressor: It looks at data and sorts it into one of two categories. It helps in determining the closest match between the data points and the function which is used to represent them.

```
from sklearn.svm import SVR

svr= SVR()
svr.fit(x_train,y_train)
preds= svr.predict(x_test)

print('Score: ',svr.score(x_train,y_train))
print('r2 score: ', r2_score(y_test,preds))
print('Mean absolute error:', mean_absolute_error(y_test,preds))
print('Mean squared error:', mean_squared_error(y_test,preds))
print('Root mean squared error:', np.sqrt(mean_squared_error(y_test,preds)))
```

```
Score:  -0.09261032543837877
r2 score:  -0.10369442985354094
Mean absolute error: 555402.9385904536
Mean squared error: 1697004119721.7385
Root mean squared error: 1302691.1067945994
```

Screenshot of the code

Using support vector regressor, we found the training score to be -9%, the R2 score to be -10%, and with very high errors.

- Lasso regularization: It uses shrinkage. Shrinkage is where data values are shrunk towards a central point, like the mean. The lasso procedure encourages simple, sparse models i.e. models with fewer parameters.

```
# Regularization

from sklearn.linear_model import Lasso

rd= Lasso()
rd.fit(x_train,y_train)
predrd= rd.predict(x_test)

print('Score: ',rd.score(x_train,y_train))
print('r2 score: ', r2_score(y_test,predrd))
print('Mean absolute error:', mean_absolute_error(y_test,predrd))
print('Mean squared error:', mean_squared_error(y_test,predrd))
print('Root mean squared error:', np.sqrt(mean_squared_error(y_test,predrd)))
```

```
Score:  0.29769139166243375
r2 score:  0.4164265397797099
Mean absolute error: 512583.80858675845
Mean squared error: 897283287264.1367
Root mean squared error: 947250.3825621485
```

Screenshot of the code

Using Lasso regressor, we found the training score to be 30%, the R2 score to be 41%, and with high errors.

- XGB regressor: It is an efficient implementation of gradient boosting that can be used for regression predictive modelling.

```
from xgboost import XGBRegressor

xg = XGBRegressor()
xg.fit(x_train,y_train)
predx= xg.predict(x_test)

print('Score: ',xg.score(x_train,y_train))
print('r2 score: ', r2_score(y_test,predx))
print('Mean absolute error:', mean_absolute_error(y_test,predx))
print('Mean squared error:', mean_squared_error(y_test,predx))
print('Root mean squared error:', np.sqrt(mean_squared_error(y_test,predx)))
```

```
Score:  0.9623558762231919
r2 score:  0.6543959671140425
Mean absolute error: 284389.1507947198
Mean squared error: 531389351740.90155
Root mean squared error: 728964.5750932631
```

Screenshot of the code

Using XGB regressor, we found the training score to be 96%, the R2 score to be 65%, with the errors being comparatively less.

- Key Metrics for success in solving problem under consideration

  Key metrics used are:

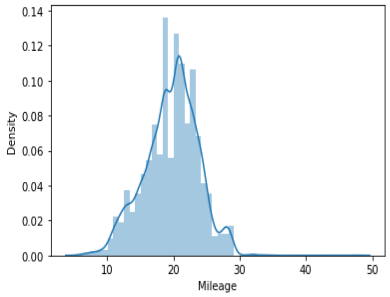
  - R2 score: R-squared is used to gives the measure of how close the data are to the fitted regression line. It is also known as the coefficient of determination. In general, it is used to know the model performance. As, the higher the R-squared, the better the model fits the data.
  - Mean absolute error: This is used because in statistics, the mean absolute error (MAE) is a way to measure the accuracy of a given model. This tells us that the average difference between the actual data value and the value predicted by the model. The lower the MAE for a given model, the more closely the model is able to predict the actual values.
  - Mean squared error: The mean squared error (MSE) is used to determine the performance of the algorithm. The mean square error gives measure of the average of error squares i.e. the average of the square of the difference between the observed and predicted values of a variable.
  - Root mean squared error: The RMSE is used, as it is a good measure of accuracy. It is the square root of the mean of the square of all of the errors**.** Basically it gives the square root of the mean squared error.

- Visualizations

  The plots that are used are:

  - Distribution plot:

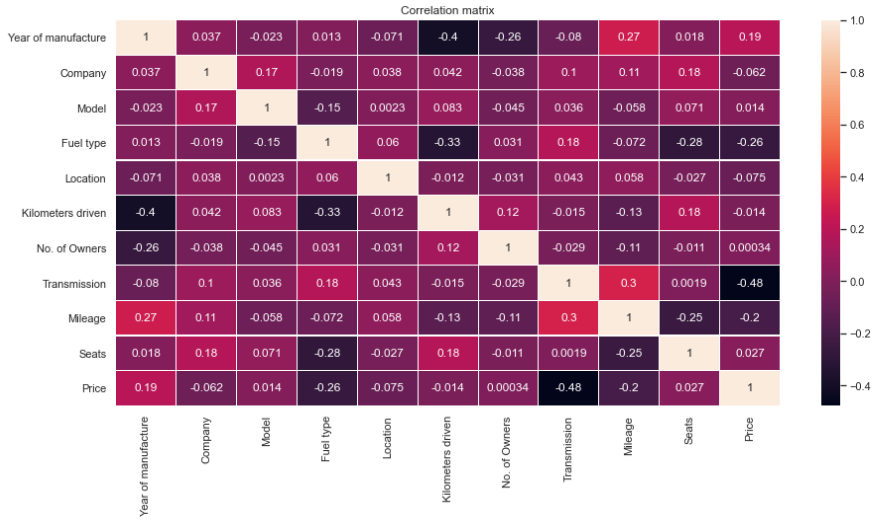
```
sns.distplot(df['Mileage']);
```



Screenshot of the plot

The distribution plot was used to check the distribution of data in the columns that contained missing values. We found that the data distribution had some skewness in it and therefore we replaced the missing values with the median of the column data.

- Heatmap:

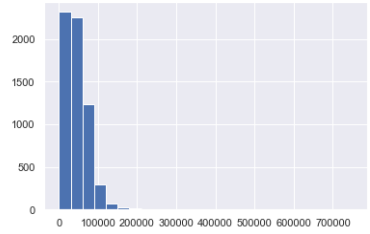


Screenshot of the plot

Heat map was used at two occasions in the project, once for checking any missing values in the dataset and the next time to plot the correlation matrix of the dataset. In the first case, we found some missing values and then treating them. And in the second case we found many types of correlation between the columns. There were some positive as well as negative relations in the matrix, and also some columns showed no correlation.

- Histogram:

```
sns.set(style='darkgrid')
plt.hist(df['Kilometers driven'], bins=25)
plt.show()
```

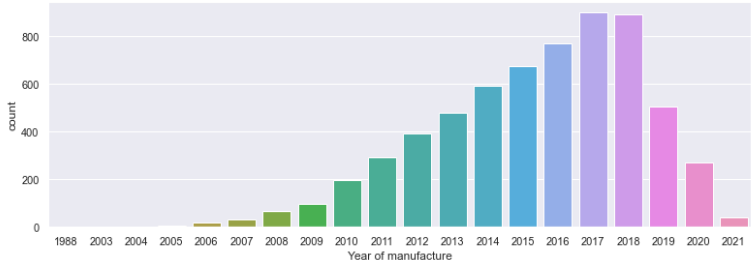


Screenshot of the plot

We used the histogram to see the distribution of data in the continuous numeric columns. We found a columns being close to normal distribution, and the other column was found to have some skewness in the data.

- Count Plot:

```
plt.figure(figsize=[13,4])
sns.countplot(df1['Year of manufacture'])
plt.show()
```

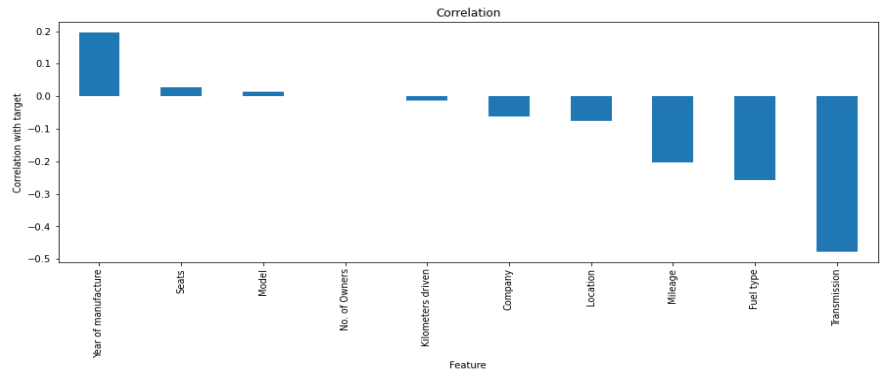


Screenshot of the plot

We used the count plot to check the count of values in different columns of the dataset. And we found some values being more and some being less in different columns of the dataset.

- Bar graph:

```
plt.figure(figsize=(15,5))
cor['Price'].sort_values(ascending=False).drop(['Price']).plot(kind='bar')
plt.xlabel('Feature')
plt.ylabel('Correlation with target')
plt.title('Correlation')
plt.show()
```

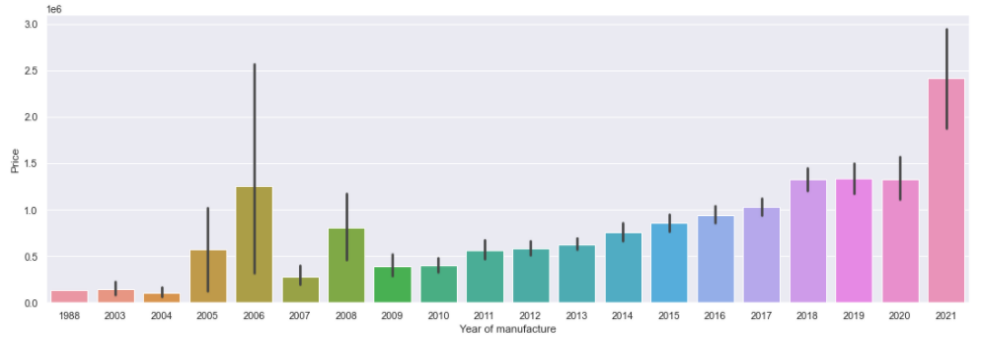


Screenshot of the plot

We plotted this bar graph to visualize the correlation of all the independent columns with the target column. We found one positive relations in between the columns. But there were mostly negative relations and a near zero correlations was also found.

- Catplot:

```
sns.catplot(x='Year of manufacture',y='Price', data=df1, kind='bar', aspect=3);
```

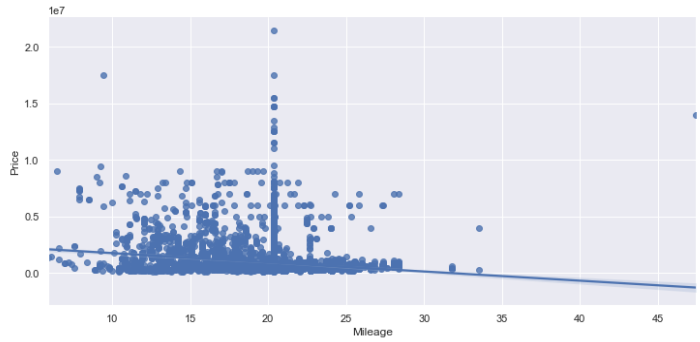


Screenshot of the plot

We used the catplot multiple times to visualize the relationship between an independent columns and the target. Here in this plot we found a good correlation between the year feature and the price. This meant that more new the car is, the more is the price.

- Lineplot:

```
sns.lmplot(x ='Mileage',y ='Price', data = df, aspect=2);
```

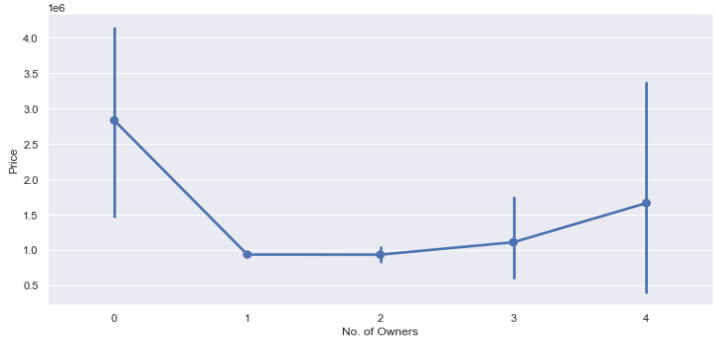


Screenshot of the plot

We used the lineplot multiple times to visualize the relationship between an independent columns and the target. Here in this plot we found a slight negative correlation between the target and the mileage column, which showed that as the mileage increased the price of car decreases.

- Factorplot:

```
sns.factorplot(x ='No. of Owners', y ='Price', data = df, aspect=2);
```

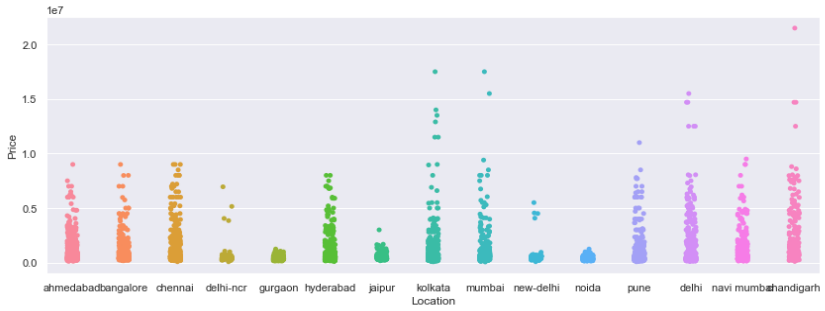


Screenshot of the plot

We used the factorplot multiple times to visualize the relationship between an independent columns and the target. Here in this plot we can see the line of the graph going down, then up, this shows no significant correlation between the columns.

- Stripplot:

```
plt.figure(figsize=[14,5])
sns.stripplot(data= df1, x='Location', y='Price')
plt.show()
```

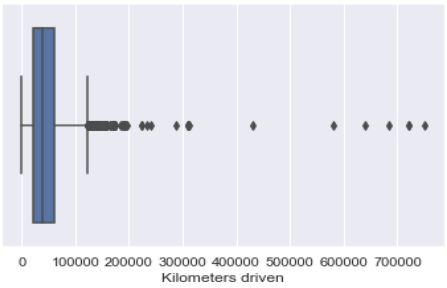


Screenshot of the plot

We used the Stripplot multiple times to visualize the relationship between an independent columns and the target. The plot here showed very little correlation between the columns. Here we can say that the cars with different price are quite equally distributed between the cities.

- Boxplot:

```
sns.boxplot(df['Kilometers driven']);
```



Screenshot of the plot

We used the boxplot to check for any outliers in the dataset. In this plot we can see some outliers in the column, some are very close to the threshold and a few can be seen far away.

- Interpretation of the Results

From the pre-processing we interpreted:

- Some unwanted columns were present in the dataset, which had no role to play in building the model.

- There were some missing values in certain columns of the dataset.
- The dataset contained much junk info along with the actual data in all the column, which needed through cleaning.
- Majority of features were categorical in nature which needed encoding.
- Removing outliers can lead to data loss, which can affect the model.
- Features which have near zero correlation with the target can affect the model performance.
- A numeric column showed skewness in the data, which was then treated.
- Scaling the features is very important in order to get a good model performance.

From the visualization we interpreted:

- The distribution of data in one column was near to normal distribution and the other column had some skewness in it.
- Among the independent columns there were all kinds of relation present, some showed positive relation and some negative.
- The features like ‘Year of manufacture’ gave a good positive correlation with the target.
- The features like ‘Transmission’, ‘Fuel type’ had a negative relationship with the target data.
- The features such as ‘No of owners’ had no correlation with the target data.
- There were few outliers present in the numeric features in the dataset, but were close to the threshold.

From the modelling we interpreted:

- Finding the best performing random state can be very useful in getting a good performance.
- Scaled data give a higher model performance as compared to non-scaled data.

- Building multiple models helps us to find and choose the best fit model for our dataset.
- Performing cross validation test is very important to detect the overfitted or underfitted models.
- The model that has the least difference between the r2 score and the cv score can be considered the best performing model.
- Hyper parameter tuning play a very crucial role in finding the best parameters for the model and selecting a final model.
- In this project the XGB regressor model performed the best among all the models, which was then saved.

CONCLUSION

- Key Findings and Conclusions of the Study

  We found that while scraping data, it is important to fetch the data from different websites and also use different locations so that there is variance in the dataset. We fetched nearly 6300 records, which formed the dataset and since the target was the price, we took the regression approach. We found an index column in the dataset, which did not have any role to play in the modelling, also some missing values were found in two columns of the dataset. We found that all the columns in the dataset contained junk data along with the actual value and hence cleansing of the data played a very crucial role in building the model. When cleaning the data, we also found few missing values in certain columns which were denoted by a dash, which needed to be treated. Also the target column had some values in word form and we had to convert all the values to the same format.

We found some features being positively correlated to the target, some being negatively and some showed very less correlation with the target, and by hit and trial we found that the feature showing near zero correlation was affecting our model negatively and hence we had to remove that feature.

We found few outliers in certain columns but they were very close to the threshold, so we decided to keep them and avoid data loss. We also found certain skewness in data in a column which we then treated. The checking of cross validation score and performing the hyper parameter tuning helped us to find the best parameters for the best models and select the final model for our project.

- Learning Outcomes of the Study in respect of Data Science

  From this project we learnt that EDA and Data cleaning play a very crucial role in the outcome of the model. The EDA when done thoroughly gives us many insights into the data and without cleaning the data we could not have proceeded further. The visualization of the dataset is very powerful in conveying the importance of different features in the dataset, through visualization we can identify the important features in the dataset, as well as features that do not have any significant importance. We learnt that the features which do not have any correlation with the target data should not be used in building the model as it may affect the model performance. Also we got the idea that scaling the dataset can be very useful in improving the model performance. We also learnt how checking cross validation and hyper parameter tuning can help us find the best model for out project.

- Limitations of this work and Scope for Future Work

  The model that we build was built by taking the data of different used cars from several websites, but since some company's cars are very few for sale, the model may perform slightly weak in case of cars belonging to those companies. Also the solution provided can

perform weak if large number of outliers and high skewness are present in the given data.

The model created can be used by our client to understand the new trend in the market and can familiarize themselves with the changes after covid 19 and also update themselves from the older model to the new one.

To further improve the results, more data of different cars can be fetched to train the model also more features can be added to the scraping list, which will help our model perform even better. We can also use more different websites for fetching data, this will help us avoid overfitting. Also different other models can be used to check which model performs best. And for tuning the models we can use more number of parameters in order to increase the model score.