Aniket Narbariya 2019130043
Ojas Patil 2019130048
Yash Patel 2019130047

AIML CAPSTONE PROJECT

Problem Statement: To implement a car price predictor where the user can predict the prices of their used cars.

Theory:

Linear Regression:

Learning a linear regression model means estimating the values of the coefficients used in the representation with the data that we have available.Linear regression is used for finding linear relationship between target and one or more predictors. There are two types of linear regression, which are Simple and Multiple.
It performs a regression task..

One Hot Encoding:
One hot encoding is a process by which categorical variables are converted into a form that could be provided to ML algorithms to do a better job in prediction.
In one hot encoding(OHE) categorical data is encoded by transforming the categories in our dataset into columns.Here, 1 is put in the corresponding cells and the rest cells are filled with 0.
One Hot Encoding increases the dimensionality by creating multiple new columns(which are dummy variables).Hence, if the data contains n categories among which m are frequent categories (m << n) then, here 'm' separate columns are created and rest 'n-m' categories are put into a new column .

Code:

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl
%matplotlib inline
mpl.style.use('ggplot')
```

In [3]:

```
car=pd.read_csv('quikr_car.csv')
```

In [4]:

```
car.head()
```

Out[4]:

	name	company	year	Price	kms_driven	fuel_type
0	Hyundai Santro Xing XO eRLX Euro III	Hyundai	2007	80,000	45,000 kms	Petrol
1	Mahindra Jeep CL550 MDI	Mahindra	2006	4,25,000	40 kms	Diesel
2	Maruti Suzuki Alto 800 Vxi	Maruti	2018	Ask For Price	22,000 kms	Petrol
3	Hyundai Grand i10 Magna 1.2 Kappa VTVT	Hyundai	2014	3,25,000	28,000 kms	Petrol
4	Ford EcoSport Titanium 1.5L TDCi	Ford	2014	5,75,000	36,000 kms	Diesel

In [5]:

```
car.shape
```

Out[5]:

```
(892, 6)
```

In [6]:

```
car.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 892 entries, 0 to 891
Data columns (total 6 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   name        892 non-null    object
 1   company     892 non-null    object
 2   year        892 non-null    object
 3   Price       892 non-null    object
 4   kms_driven  840 non-null    object
 5   fuel_type   837 non-null    object
dtypes: object(6)
memory usage: 41.9+ KB
```

Creating backup copy

In [7]:
```
backup=car.copy()
```

Quality

- names are pretty inconsistent
- names have company names attached to it
- some names are spam like 'Maruti Ertiga showroom condition with' and 'Well mentained Tata Sumo'
- company: many of the names are not of any company like 'Used', 'URJENT', and so on.
- year has many non-year values
- year is in object. Change to integer
- Price has Ask for Price
- Price has commas in its prices and is in object
- kms_driven has object values with kms at last.
- It has nan values and two rows have 'Petrol' in them
- fuel_type has nan values

Cleaning Data

year has many non-year values

In [8]:
```
car=car[car['year'].str.isnumeric()]
```

year is in object. Change to integer

In [9]:
```
car['year']=car['year'].astype(int)
```

Price has Ask for Price

In [10]:
```
car=car[car['Price']!='Ask For Price']
```

Price has commas in its prices and is in object

In [11]:
```
car['Price']=car['Price'].str.replace(',','').astype(int)
```

kms_driven has object values with kms at last.

In [12]:
```
car['kms_driven']=car['kms_driven'].str.split().str.get(0).str.replace(',','')
```

It has nan values and two rows have 'Petrol' in them

In [13]:
```
car=car[car['kms_driven'].str.isnumeric()]
```

In [14]:
```
car['kms_driven']=car['kms_driven'].astype(int)
```

fuel_type has nan values

In [15]:
```
car=car[~car['fuel_type'].isna()]
```

In [16]:
```
car.shape
```

Out[16]:
```
(816, 6)
```

name and company had spammed data...but with the previous cleaning, those rows got removed.

Company does not need any cleaning now. Changing car names. Keeping only the first three words

In [17]:
```
car['name']=car['name'].str.split().str.slice(start=0,stop=3).str.join(' ')
```

Resetting the index of the final cleaned data

In [18]:
```
car=car.reset_index(drop=True)
```

Cleaned Data

In [19]:
```
car
```

Out[19]:

	name	company	year	Price	kms_driven	fuel_type
0	Hyundai Santro Xing	Hyundai	2007	80000	45000	Petrol
1	Mahindra Jeep CL550	Mahindra	2006	425000	40	Diesel
2	Hyundai Grand i10	Hyundai	2014	325000	28000	Petrol
3	Ford EcoSport Titanium	Ford	2014	575000	36000	Diesel
4	Ford Figo	Ford	2012	175000	41000	Diesel
...	...	...	...	...	...	...
811	Maruti Suzuki Ritz	Maruti	2011	270000	50000	Petrol
812	Tata Indica V2	Tata	2009	110000	30000	Diesel
813	Toyota Corolla Altis	Toyota	2009	300000	132000	Petrol
814	Tata Zest XM	Tata	2018	260000	27000	Diesel
815	Mahindra Quanto C8	Mahindra	2013	390000	40000	Diesel

816 rows × 6 columns

In [20]:
```
car.to_csv('Cleaned_Car_data.csv')
```

In [21]:
```
car.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 816 entries, 0 to 815
Data columns (total 6 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   name        816 non-null    object
 1   company     816 non-null    object
 2   year        816 non-null    int32 
 3   Price       816 non-null    int32 
 4   kms_driven  816 non-null    int32 
 5   fuel_type   816 non-null    object
dtypes: int32(3), object(3)
memory usage: 28.8+ KB
```

In [22]:

```
car.describe(include='all')
```

Out[22]:

	name	company	year	Price	kms_driven	fuel_type
count	816	816	816.000000	8.160000e+02	816.000000	816
unique	254	25	NaN	NaN	NaN	3
top	Maruti Suzuki Swift	Maruti	NaN	NaN	NaN	Petrol
freq	51	221	NaN	NaN	NaN	428
mean	NaN	NaN	2012.444853	4.117176e+05	46275.531863	NaN
std	NaN	NaN	4.002992	4.751844e+05	34297.428044	NaN
min	NaN	NaN	1995.000000	3.000000e+04	0.000000	NaN
25%	NaN	NaN	2010.000000	1.750000e+05	27000.000000	NaN
50%	NaN	NaN	2013.000000	2.999990e+05	41000.000000	NaN
75%	NaN	NaN	2015.000000	4.912500e+05	56818.500000	NaN
max	NaN	NaN	2019.000000	8.500003e+06	400000.000000	NaN

Out[22]:

	name	company	year	Price	kms_driven	fuel_type
count	816	816	816.000000	8.160000e+02	816.000000	816
unique	254	25	NaN	NaN	NaN	3
top	Maruti Suzuki Swift	Maruti	NaN	NaN	NaN	Petrol
freq	51	221	NaN	NaN	NaN	428
mean	NaN	NaN	2012.444853	4.117176e+05	46275.531863	NaN
std	NaN	NaN	4.002992	4.751844e+05	34297.428044	NaN
min	NaN	NaN	1995.000000	3.000000e+04	0.000000	NaN
25%	NaN	NaN	2010.000000	1.750000e+05	27000.000000	NaN
50%	NaN	NaN	2013.000000	2.999990e+05	41000.000000	NaN
75%	NaN	NaN	2015.000000	4.912500e+05	56818.500000	NaN
max	NaN	NaN	2019.000000	8.500003e+06	400000.000000	NaN

In [ ]:

In [23]:

```
car=car[car['Price']<6000000]
```

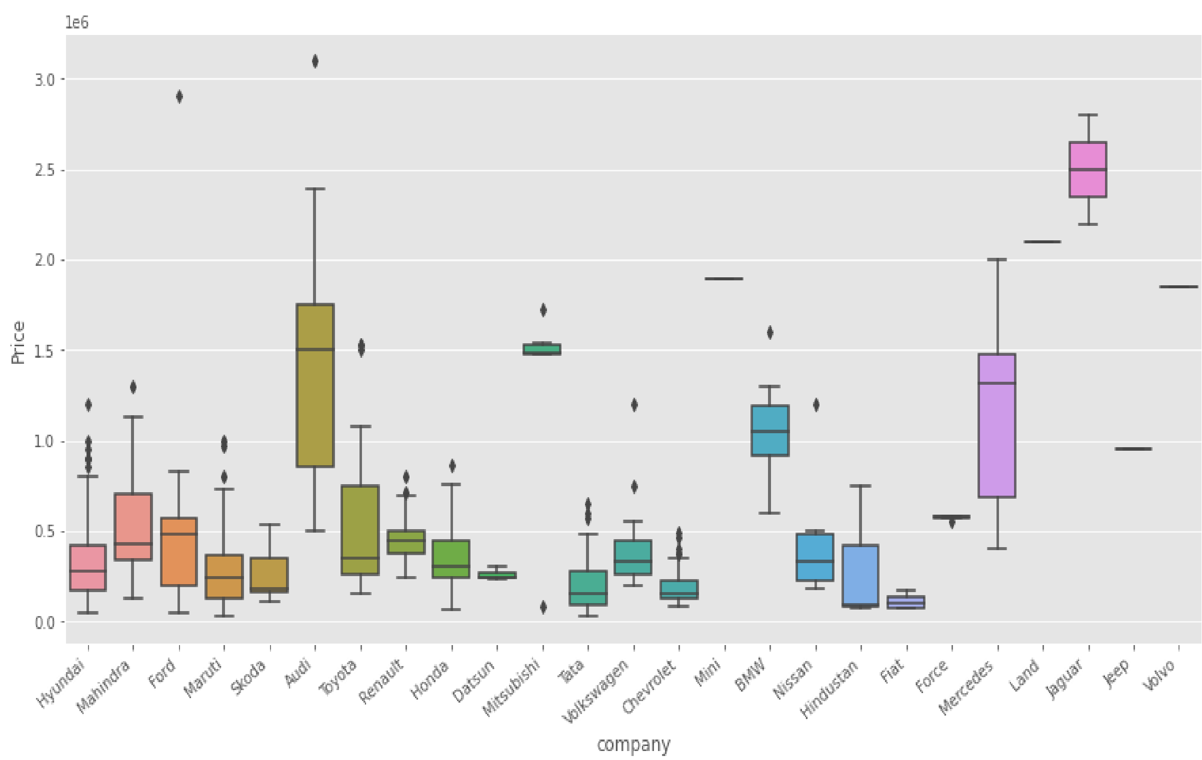
Checking relationship of Company with Price

In [24]:
```
car['company'].unique()
```

Out[24]:
```
array(['Hyundai', 'Mahindra', 'Ford', 'Maruti', 'Skoda', 'Audi', 'Toyota',
       'Renault', 'Honda', 'Datsun', 'Mitsubishi', 'Tata', 'Volkswagen',
       'Chevrolet', 'Mini', 'BMW', 'Nissan', 'Hindustan', 'Fiat', 'Force',
       'Mercedes', 'Land', 'Jaguar', 'Jeep', 'Volvo'], dtype=object)
```

In [25]:
```
import seaborn as sns
```

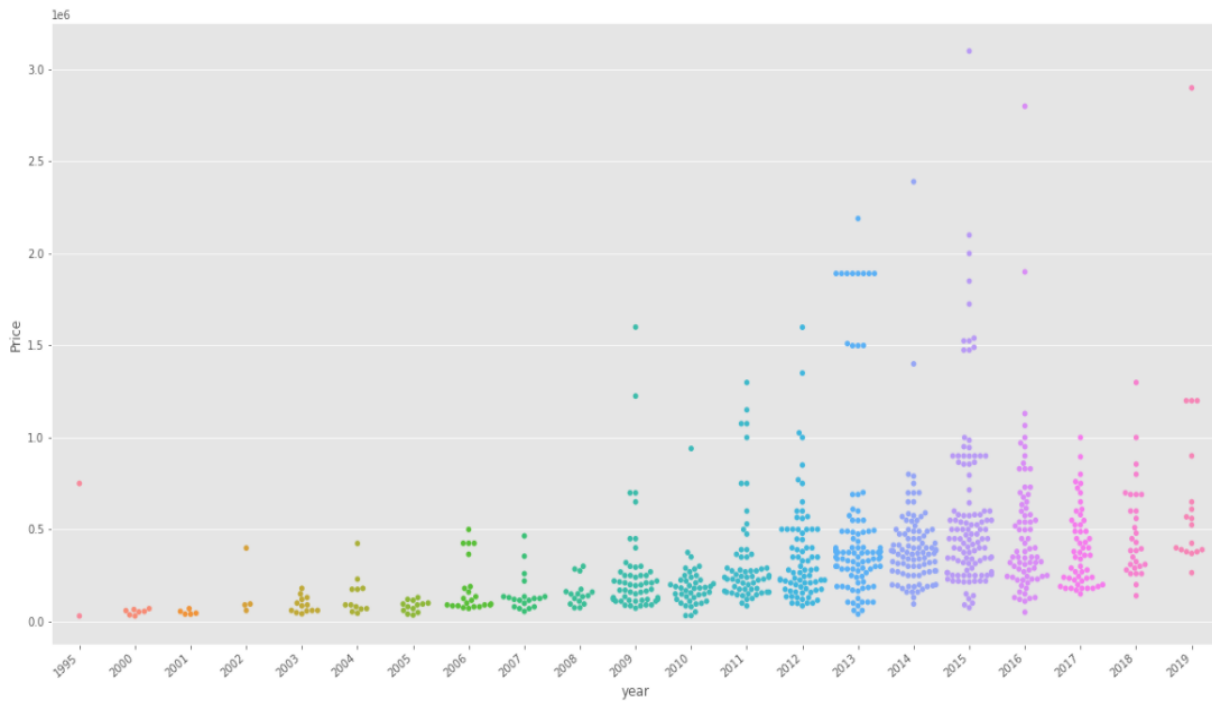
In [26]:
```
plt.subplots(figsize=(15,7))
ax=sns.boxplot(x='company',y='Price',data=car)
ax.set_xticklabels(ax.get_xticklabels(),rotation=40,ha='right')
plt.show()
```



Checking relationship of Year with Price

In [27]:

```
plt.subplots(figsize=(20,10))
ax=sns.swarmplot(x='year',y='Price',data=car)
ax.set_xticklabels(ax.get_xticklabels(),rotation=40,ha='right')
plt.show()
```



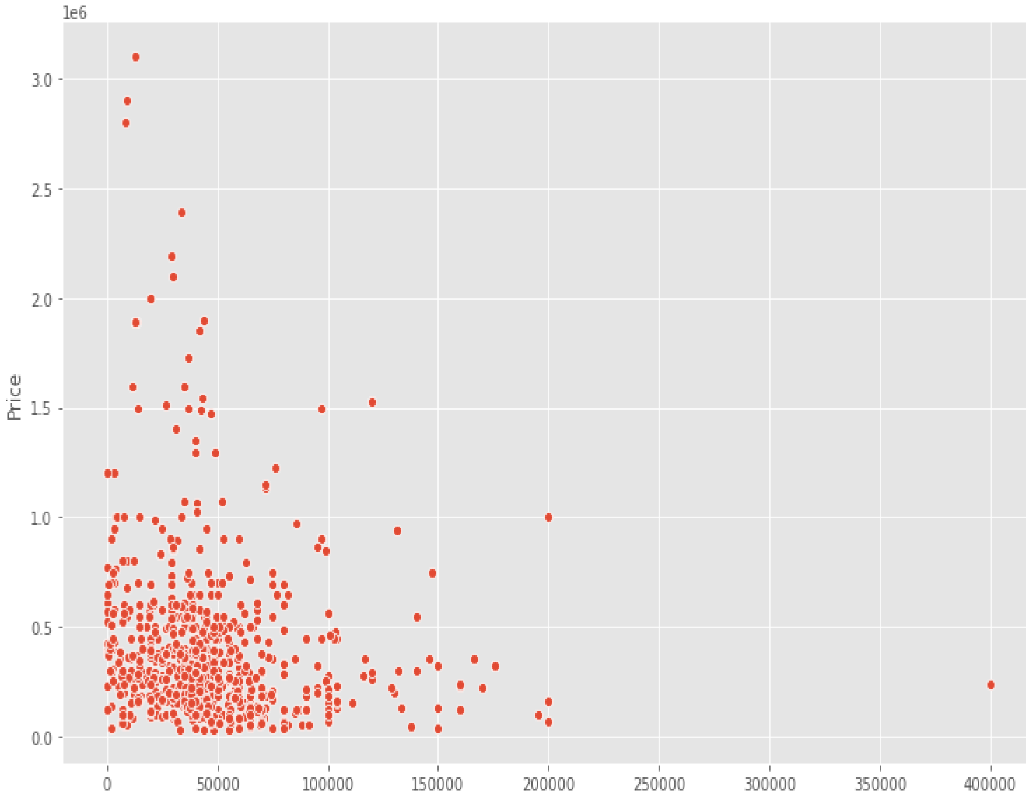
Checking relationship of kms_driven with Price

In [28]:

```
sns.relplot(x='kms_driven',y='Price',data=car,height=7,aspect=1.5)
```

Out[28]:

```
<seaborn.axisgrid.FacetGrid at 0x1d3534604c0>
```

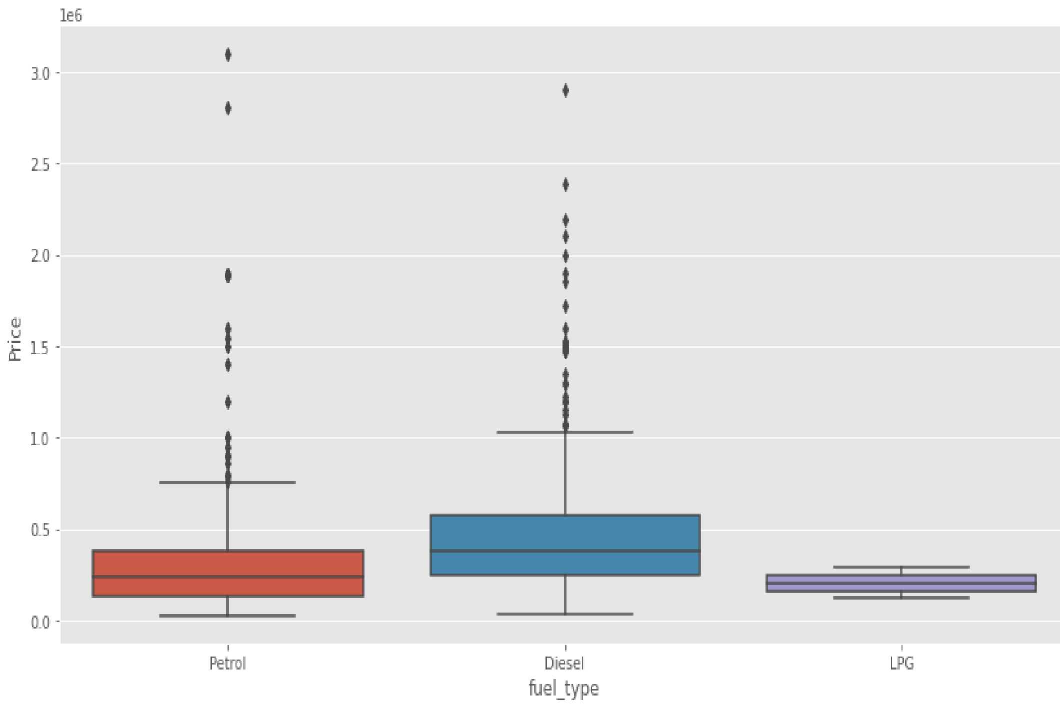


Checking relationship of Fuel Type with Price

In [29]:

```
plt.subplots(figsize=(14,7))
sns.boxplot(x='fuel_type',y='Price',data=car)
```

Out[29]: <matplotlib.axes._subplots.AxesSubplot at 0x1d353660d60>

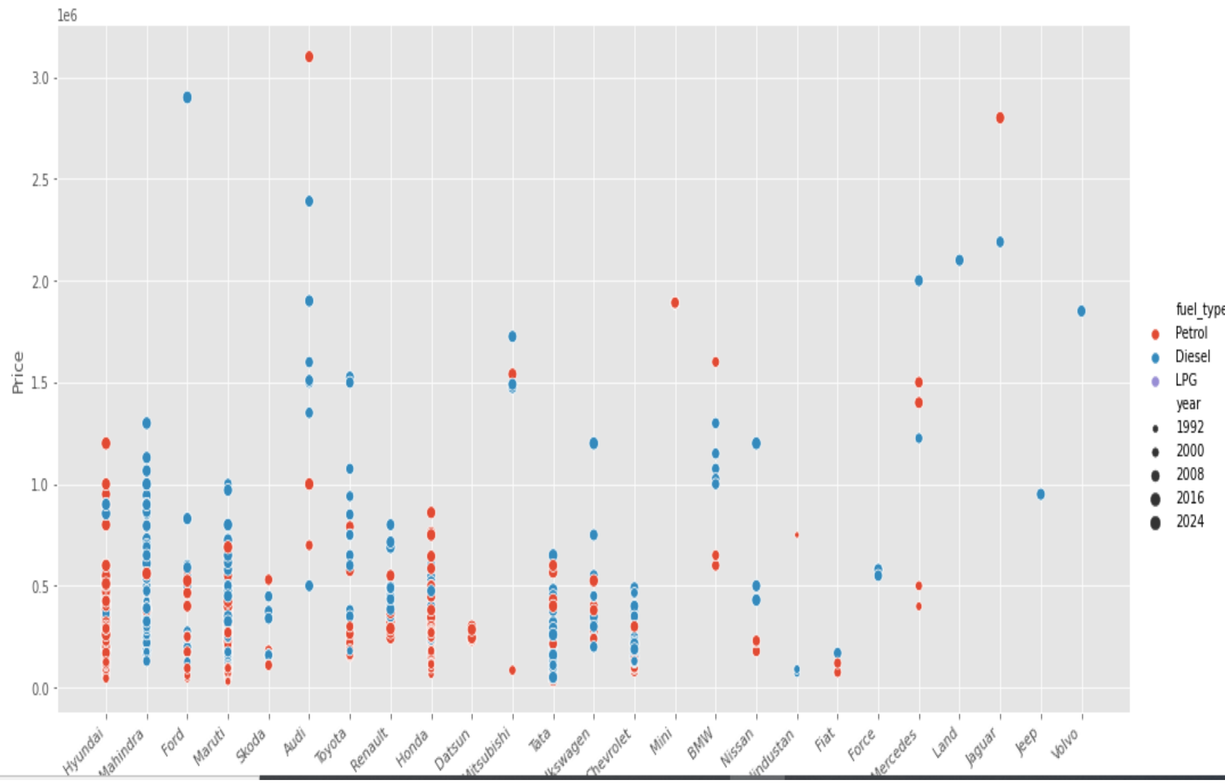


Relationship of Price with FuelType, Year and Company mixed

In [30]:

```
ax=sns.relplot(x='company',y='Price',data=car,hue='fuel_type',size='year',height=7,aspect=2)
ax.set_xticklabels(rotation=40,ha='right')
```

Out[30]: <seaborn.axisgrid.FacetGrid at 0x1d353675f40>



Extracting Training Data

In [32]:
```
X=car[['name','company','year','kms_driven','fuel_type']]
y=car['Price']
```

In [33]:
```
X
```

Out[33]:

	name	company	year	kms_driven	fuel_type
0	Hyundai Santro Xing	Hyundai	2007	45000	Petrol
1	Mahindra Jeep CL550	Mahindra	2006	40	Diesel
2	Hyundai Grand i10	Hyundai	2014	28000	Petrol
3	Ford EcoSport Titanium	Ford	2014	36000	Diesel
4	Ford Figo	Ford	2012	41000	Diesel
...	...	...	...	...	...
811	Maruti Suzuki Ritz	Maruti	2011	50000	Petrol
812	Tata Indica V2	Tata	2009	30000	Diesel
813	Toyota Corolla Altis	Toyota	2009	132000	Petrol
814	Tata Zest XM	Tata	2018	27000	Diesel
815	Mahindra Quanto C8	Mahindra	2013	40000	Diesel

815 rows × 5 columns

In [34]:
```
y.shape
```

Out[34]:
```
(815,)
```

Applying Train Test Split

In [35]:
```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2)
```

In [74]:
```
from sklearn.linear_model import LinearRegression
```

In [75]:
```
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import make_column_transformer
from sklearn.pipeline import make_pipeline
from sklearn.metrics import r2_score
```

Creating an OneHotEncoder object to contain all the possible categories

In [39]:
```
ohe=OneHotEncoder()
ohe.fit(X[['name','company','fuel_type']])
```

Out[39]:
```
OneHotEncoder()
```

Creating a column transformer to transform categorical columns

In [52]:
```
column_trans=make_column_transformer((OneHotEncoder(categories=ohe.categories_),['name','company','fuel_type']),
                                    remainder='passthrough')
```

Linear Regression Model

In [54]:
```
lr=LinearRegression()
```

Making a pipeline

In [55]:
```
pipe=make_pipeline(column_trans,lr)
```

Fitting the model

In [59]:
```
pipe.fit(X_train,y_train)
```

Out[59]:
```
Pipeline(steps=[('columntransformer',
                 ColumnTransformer(remainder='passthrough',
                                   transformers=[('onehotencoder',
                                                  OneHotEncoder(categories=[array(['Audi A3 Cabriolet', 'Audi A4 1.8', 'Audi A4 2.0', 'Audi A6 2.0',
       'Audi A8', 'Audi Q3 2.0', 'Audi Q5 2.0', 'Audi Q7', 'BMW 3 Series',
       'BMW 5 Series', 'BMW 7 Series', 'BMW X1', 'BMW X1 sDrive20d',
       'BMW X1 xDrive20d', 'Chevrolet Beat', 'Chevrolet Beat...
                                                                            array(['Audi', 'BMW', 'Chevrolet', 'Datsun', 'Fiat', 'Force', 'Ford',
       'Hindustan', 'Honda', 'Hyundai', 'Jaguar', 'Jeep', 'Land',
       'Mahindra', 'Maruti', 'Mercedes', 'Mini', 'Mitsubishi', 'Nissan',
       'Renault', 'Skoda', 'Tata', 'Toyota', 'Volkswagen', 'Volvo'],
      dtype=object),
                                                                            array(['Diesel', 'LPG', 'Petrol'], dtype=object)]),
                                                  ['name', 'company',
                                                   'fuel_type'])])),
                ('linearregression', LinearRegression())])
```

In [60]:
```
y_pred=pipe.predict(X_test)
```

Checking R2 Score

In [61]:
```
r2_score(y_test,y_pred)
```

Out[61]:
```
0.7627456237676113
```

Finding the model with a random state of TrainTestSplit where the model was found to give almost 0.92 as r2_score

```
In [62]:
scores=[]
for i in range(1000):
    X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.1,random_state=i)
    lr=LinearRegression()
    pipe=make_pipeline(column_trans,lr)
    pipe.fit(X_train,y_train)
    y_pred=pipe.predict(X_test)
    scores.append(r2_score(y_test,y_pred))
```

```
In [63]:
np.argmax(scores)
```

```
Out[63]:
655
```

```
In [64]:
scores[np.argmax(scores)]
```

```
Out[64]:
0.920088412025344
```

```
In [65]:
pipe.predict(pd.DataFrame(columns=X_test.columns,data=np.array(['Maruti Suzuki Swift','Maruti',2019,100,'Petrol']).reshape(1,5)))
```

```
Out[65]:
array([400707.28215338])
```

Conclusion:

As a result, we have successfully finished the linear regression capstone project for the automobile price predictor. We also learned how to clean our data while we were there.

In most cases, we clean our dataset by looking at the values of each column individually.

Also, during data cleaning, we convert our given data types to the required data type on which we work, as well as remove any values that do not match the required data type (for example, if the data isn't a number and we need a number, we delete such records).
To obtain a lot of precision, we additionally removed the outliers.
Then we learned about linear regression, which is a technique for determining which columns in a dataset can be used as features in a model. Then we execute train test split on our data based on the different features. We also noticed that when we run train test split on our entire test data, the accuracy is low, but when we take subsets of our datasets and run train test split on them, we get our model's accuracy scores, from which we choose our highest accuracy test data as our final test data.