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 [2]:
```
car=pd.read_csv('quikr_car.csv')
```

In [3]:
```
car.head()
```

Out[3]:

	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 [4]:
```
car.shape
```

Out[4]:
```
(892, 6)
```

In [5]:
```
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 [6]:
```
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 [7]:
```
car=car[car['year'].str.isnumeric()]
```

year is in object. Change to integer

In [8]:
```
car['year']=car['year'].astype(int)
```

Price has Ask for Price

In [9]:
```
car=car[car['Price']!='Ask For Price']
```

Price has commas in its prices and is in object

In [10]:
```
car['Price']=car['Price'].str.replace(',','').astype(int)
```

kms_driven has object values with kms at last.

In [11]:
```
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 [12]:
```
car=car[car['kms_driven'].str.isnumeric()]
```

In [13]:
```
car['kms_driven']=car['kms_driven'].astype(int)
```

fuel_type has nan values

In [14]:
```
car=car[~car['fuel_type'].isna()]
```

In [15]:
```
car.shape
```

Out[15]:
```
(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 [16]:
```
car['name']=car['name'].str.split().str.slice(start=0,stop=3).str.join(' ')
```

Resetting the index of the final cleaned data

In [17]:
```
car=car.reset_index(drop=True)
```

Cleaned Data

In [18]:
```
car
```

Out[18]:

	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 [19]:
```
car.to_csv('Cleaned_Car_data.csv')
```

In [20]:
```
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 [21]:
```
car.describe()
```

Out[21]:

	year	Price	kms_driven
count	816.000000	8.160000e+02	816.000000
mean	2012.444853	4.117176e+05	46275.531863
std	4.002992	4.751844e+05	34297.428044
min	1995.000000	3.000000e+04	0.000000
25%	2010.000000	1.750000e+05	27000.000000
50%	2013.000000	2.999990e+05	41000.000000
75%	2015.000000	4.912500e+05	56818.500000
max	2019.000000	8.500003e+06	400000.000000

In [ ]:
```

```

In [22]:
```
car=car[car['Price']<6000000]
print(car)
```

```
                      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

[815 rows x 6 columns]
```

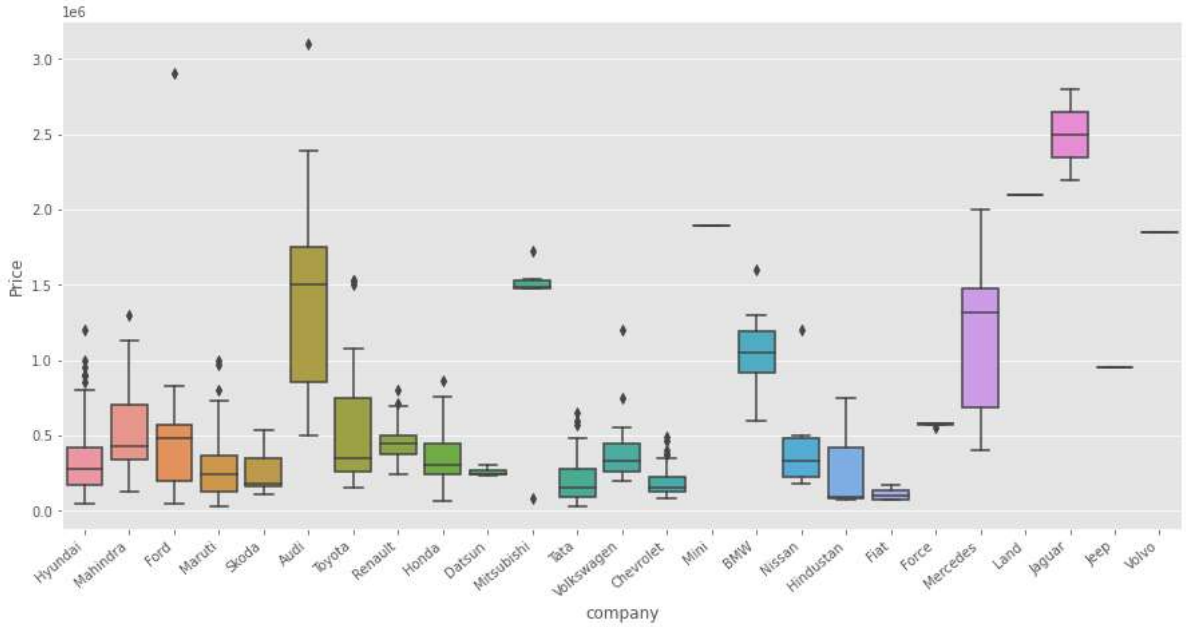
Checking relationship of Company with Price

In [23]:
```
car['company'].unique()
```

Out[23]:
```
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 [24]:
```
import seaborn as sns
```

In [25]:
```
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()
```

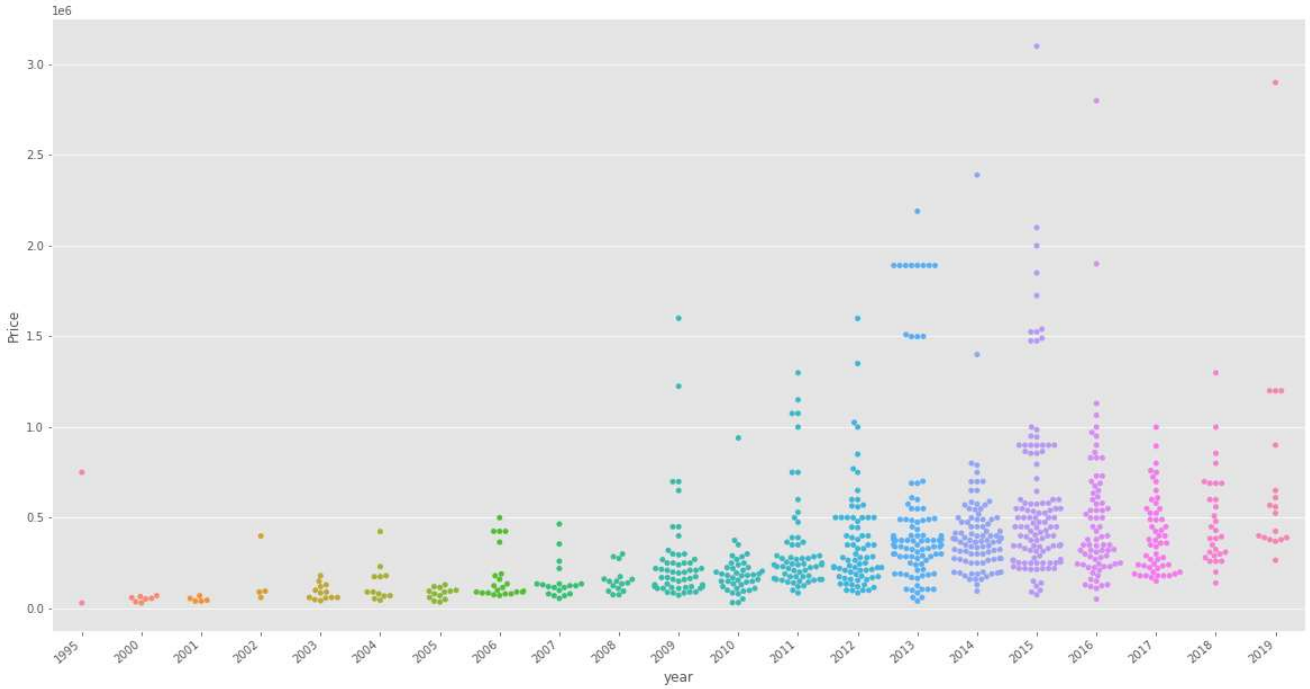


In [26]:
```
import warnings

warnings.simplefilter("ignore", UserWarning)
```

Checking relationship of Year with Price

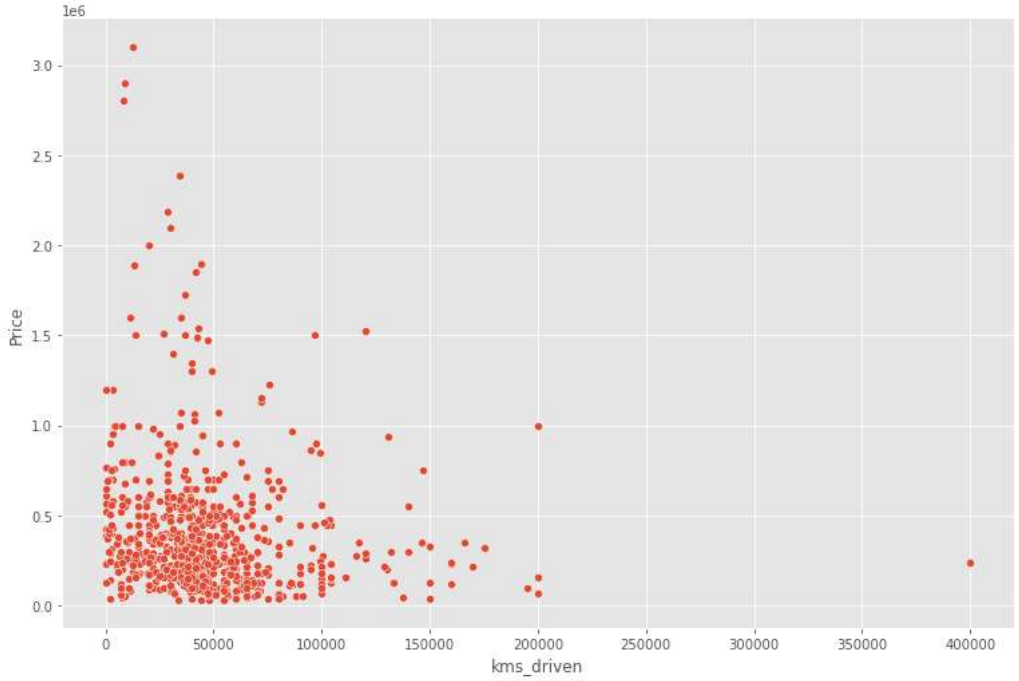
In [27]:
```
plt.subplots(figsize=(20,10))
ax=sns.swarmplot(x='year',y='Price',data=car)
ax.set_xticklabels(ax.get_xticklabels(),rotation=40,ha='right')
plt.show()
```



Checking relationship of kms_driven with Price

In [28]:
```
sns.relplot(x='kms_driven',y='Price',data=car,height=7,aspect=1.5)
```

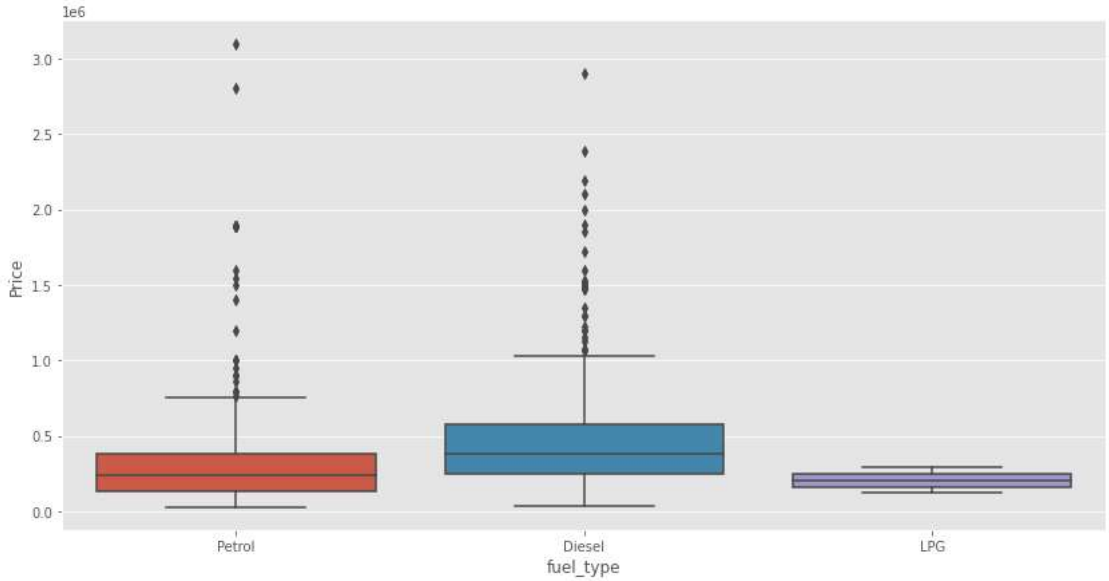
Out[28]:
```
<seaborn.axisgrid.FacetGrid at 0x1f61c36a8e0>
```



Checking relationship of Fuel Type with Price

In [29]:
```
plt.subplots(figsize=(14,7))
sns.boxplot(x='fuel_type',y='Price',data=car)
```

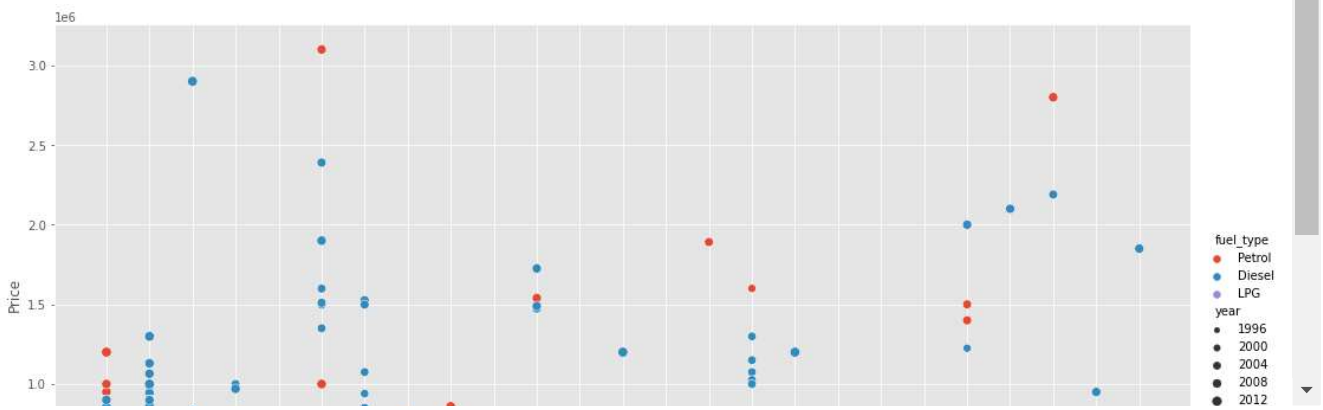
Out[29]:
```
<AxesSubplot:xlabel='fuel_type', ylabel='Price'>
```



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 0x1f617bc48b0>
```



Extracting Training Data

In [31]:
```
X=car[['name','company','year','kms_driven','fuel_type']]
y=car['Price']
```

In [32]:
```
X
```

Out[32]:

	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 [33]:
```
y.shape
```

Out[33]:
```
(815,)
```

Applying Train Test Split

In [34]:
```
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 [35]:
```
from sklearn.linear_model import LinearRegression
```

In [36]:
```
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 [37]:
```
ohe=OneHotEncoder()
ohe.fit(X[['name','company','fuel_type']])
```

Out[37]:
```
OneHotEncoder
OneHotEncoder()
```

Creating a column transformer to transform categorical columns

In [38]:
```
column_trans=make_column_transformer((OneHotEncoder(categories=ohe.categories_),['name','company','fuel_type']),
                                     remainder='passthrough')
```

Linear Regression Model

In [39]:
```
lr=LinearRegression()
```

Making a pipeline

In [40]:
```
pipe=make_pipeline(column_trans,lr)
```

Fitting the model

In [41]:
```
pipe.fit(X_train,y_train)
```

Out[41]:
```
Pipeline
columntransformer: ColumnTransformer
onehotencoder
remainder
OneHotEncoder
passthrough
LinearRegression
```

In [42]:
```
y_pred=pipe.predict(X_test)
```

Checking R2 Score

In [43]:
```
r2_score(y_test,y_pred)
```

Out[43]:
```
0.6056087185113774
```

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

In [44]:
```
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 [45]:
```
np.argmax(scores)
```

Out[45]:
```
655
```

In [46]:
```
scores[np.argmax(scores)]
```

Out[46]:
```
0.9200949144714073
```

In [47]:
```
pipe.predict(pd.DataFrame(columns=X_test.columns,data=np.array(['Maruti Suzuki Swift','Maruti',2019,100,'Petrol']).reshape(1,5)))
```

Out[47]:
```
array([400871.07693607])
```

The best model is found at a certain random state

In [48]:
```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.1,random_state=np.argmax(scores))
lr=LinearRegression()
pipe=make_pipeline(column_trans,lr)
pipe.fit(X_train,y_train)
y_pred=pipe.predict(X_test)
r2_score(y_test,y_pred)
```

Out[48]:
```
0.9200949144714073
```