Used Car Project / Yeoman Yoon

Defining Problem:

Unlike new cars, used cars have very unique features on top of its brand and models. It is well known that mileage, year, and brand effects the used car a lot. Yet, it is not as easy to predict how people perceive some cars to be more reliant used car. New cars' pricing may determine people's affordability. Or, any other factors can contribute.

Only if we could determine how market price of used cars will be, not only can we sell used cars at more beneficial price, but we can also control purchasing the used car at more reasonable price as dealershop. Find out the market price of used cars in India!

Import necessary libraries for analysis:

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import warnings
pd.set_option('display.max_rows', 200)

#scaler
from sklearn.preprocessing import StandardScaler, MinMaxScaler
#Linear Reg
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import linear_model
```

Read and store data in Pandas:

In [2]:
```
data = pd.read_csv('used_cars_data.csv')
```

Quick data overview:

In [3]:
```
print(f"Rows: {data.shape[0]}, Columns: {data.shape[1]}")
np.random.seed(1) # save the randomness in case of future reference
data.sample(n=8) # use random samples to see variation of the data
```

```
Rows: 7253, Columns: 14
```

Out[3]:

	S.No.	Name	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Ty
2397	2397	Ford EcoSport 1.5 Petrol Trend	Kolkata	2016	21460	Petrol	Manual	F
3777	3777	Maruti Wagon R VXI 1.2	Kochi	2015	49818	Petrol	Manual	F
4425	4425	Ford Endeavour 4x2 XLT	Hyderabad	2007	130000	Diesel	Manual	F
3661	3661	Mercedes-Benz E-Class E250 CDI Avantgrade	Coimbatore	2016	39753	Diesel	Automatic	F
4514	4514	Hyundai Xcent 1.2 Kappa AT SX Option	Kochi	2016	45560	Petrol	Automatic	F
599	599	Toyota Innova Crysta 2.8 ZX AT	Coimbatore	2019	40674	Diesel	Automatic	F
186	186	Mercedes-Benz E-Class E250 CDI Avantgrade	Bangalore	2014	37382	Diesel	Automatic	F
305	305	Audi A6 2011-2015 2.0 TDI Premium Plus	Kochi	2014	61726	Diesel	Automatic	F

Data Dictionary:

S.No.: Serial Number
Name: Name of the car which includes Brand name and Model name
Location: The location in which the car is being sold or is available for purchase Cities
Year: Manufacturing year of the car
Kilometers_driven: The total kilometers driven in the car by the previous owner(s) in KM.
Fuel_Type: The type of fuel used by the car. (Petrol, Diesel, Electric, CNG, LPG)
Transmission: The type of transmission used by the car. (Automatic / Manual)
Owner: Type of ownership
Mileage: The standard mileage offered by the car company in kmpl or km/kg
Engine: The displacement volume of the engine in CC.
Power: The maximum power of the engine in bhp.
Seats: The number of seats in the car.
New_Price: The price of a new car of the same model in INR Lakhs.(1 Lakh = 100, 000)
Price: The price of the used car in INR Lakhs (1 Lakh = 100, 000)

In [4]:
```
data.drop(['S.No.'],axis=1,inplace=True) # drop Serial number, because we are not using it in this particular project.
```

In [5]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7253 entries, 0 to 7252
Data columns (total 13 columns):
Name                 7253 non-null object
Location             7253 non-null object
Year                 7253 non-null int64
Kilometers_Driven    7253 non-null int64
Fuel_Type            7253 non-null object
Transmission         7253 non-null object
Owner_Type           7253 non-null object
Mileage              7251 non-null object
Engine               7207 non-null object
Power                7207 non-null object
Seats                7200 non-null float64
New_Price            1006 non-null object
Price                6019 non-null float64
dtypes: float64(2), int64(2), object(9)
memory usage: 736.8+ KB
```

In [6]:
```
data.isnull().sum().sort_values(ascending = False) #cheking null values
```

Out[6]:
```
New_Price            6247
Price                1234
Seats                  53
Power                  46
Engine                 46
Mileage                 2
Owner_Type              0
Transmission            0
Fuel_Type               0
Kilometers_Driven       0
Year                    0
Location                0
Name                    0
dtype: int64
```

Checking for non-numeric values:

Looking for Unique values and consistency.

In [7]:
```
for indexName in data.dtypes[data.dtypes == 'object'].index:
    if (len(data[indexName].value_counts(dropna=False)) < 10):
        print(data[indexName].value_counts(dropna=False))
    else:
        print(data[indexName].value_counts(dropna=False)[:10])
        print(f"Number of unique value: {len(data[indexName].value_counts(drop
na=False))}  ")
    print('\n\n')
```

```
Mahindra XUV500 W8 2WD        55
Maruti Swift VDI              49
Maruti Swift Dzire VDI        42
Honda City 1.5 S MT           39
Maruti Swift VDI BSIV         37
Toyota Fortuner 3.0 Diesel    35
Maruti Ritz VDi               35
Honda Amaze S i-Dtech         32
Honda City 1.5 V MT           32
Hyundai Grand i10 Sportz      32
Name: Name, dtype: int64
Number of unique value: 2041
```

```
Mumbai        949
Hyderabad     876
Kochi         772
Coimbatore    772
Pune          765
Delhi         660
Kolkata       654
Chennai       591
Jaipur        499
Bangalore     440
Name: Location, dtype: int64
Number of unique value: 11
```

```
Diesel      3852
Petrol      3325
CNG           62
LPG           12
Electric       2
Name: Fuel_Type, dtype: int64
```

```
Manual       5204
Automatic    2049
Name: Transmission, dtype: int64
```

```
First             5952
Second            1152
Third              137
Fourth & Above      12
Name: Owner_Type, dtype: int64
```

```
17.0 kmpl     207
18.9 kmpl     201
18.6 kmpl     144
21.1 kmpl     106
```

```
20.36 kmpl    105
17.8 kmpl      98
18.0 kmpl      89
12.8 kmpl      87
18.5 kmpl      86
16.0 kmpl      85
Name: Mileage, dtype: int64
Number of unique value: 451
```

```
1197 CC    732
1248 CC    610
1498 CC    370
998 CC     309
1198 CC    281
2179 CC    278
1497 CC    273
1968 CC    266
1995 CC    212
1461 CC    188
Name: Engine, dtype: int64
Number of unique value: 151
```

```
74 bhp       280
98.6 bhp     166
73.9 bhp     152
140 bhp      142
null bhp     129
78.9 bhp     128
67.1 bhp     126
67.04 bhp    125
82 bhp       124
88.5 bhp     120
Name: Power, dtype: int64
Number of unique value: 387
```

```
NaN           6247
63.71 Lakh       6
4.78 Lakh        6
33.36 Lakh       6
95.13 Lakh       6
44.28 Lakh       5
11.75 Lakh       5
11.48 Lakh       5
47.87 Lakh       5
11.26 Lakh       5
Name: New_Price, dtype: int64
Number of unique value: 626
```

Notes on Categorical Data for Cleaning:

Name: Need to divide car brand and model
Location: Change dtype to category.
Year: Consider binning. Change dtype to category.
Fuel_Type: Change dtype to category. (Petrol, Diesel, Electric, CNG, LPG)
Transmission: Change dtype to category. (Automatic / Manual)
Owner: Change dtype to numeric. Change data Name: (First = 1, Second = 2, Third = 3, Fourth & Above = 4)
Mileage: remove kmpl or km/kg and make it numeric (re-learned 1kg = 1liter)
Engine: Remove CC and make it numeric.
Power: Remove bhp and make it numeric. 129 of them are "null bhp"
New_Price: Remove INR, Cr and make it numeric.

Name:

In [9]:
```
nameAndBrand = data['Name'].str.split(' ',n = 1, expand = True) # divide by first space  (I don't know too much about car brand, so I assumed car brands are made of one word)
nameAndBrand.head(8)
```

Out[9]:

	0	1
0	Maruti	Wagon R LXI CNG
1	Hyundai	Creta 1.6 CRDi SX Option
2	Honda	Jazz V
3	Maruti	Ertiga VDI
4	Audi	A4 New 2.0 TDI Multitronic
5	Hyundai	EON LPG Era Plus Option
6	Nissan	Micra Diesel XV
7	Toyota	Innova Crysta 2.8 GX AT 8S

In [10]:
```
data.drop(['Name'], axis=1, inplace = True)
data['Name_Brand'] = nameAndBrand[0]
data['Name_Model'] = nameAndBrand[1]

del nameAndBrand
```

In [11]:
```
data['Name_Brand'] = data['Name_Brand'].astype("category")
```

Location:

In [12]:
```
data['Location'] = data['Location'].astype("category")
```

Year:

In [13]:
```
binned_year = pd.cut(data['Year'],[-np.inf, 2005, 2010, 2015, np.inf]) # bin w
ith range of 5 years
binned_year
```

Out[13]:
```
0       (2005.0, 2010.0]
1       (2010.0, 2015.0]
2       (2010.0, 2015.0]
3       (2010.0, 2015.0]
4       (2010.0, 2015.0]
               ...
7248    (2010.0, 2015.0]
7249    (2010.0, 2015.0]
7250    (2010.0, 2015.0]
7251    (2010.0, 2015.0]
7252    (2010.0, 2015.0]
Name: Year, Length: 7253, dtype: category
Categories (4, interval[float64]): [(-inf, 2005.0] < (2005.0, 2010.0] < (201
0.0, 2015.0] < (2015.0, inf]]
```

In [14]:
```
binned_year.value_counts() # this will go in the data and become category
```

Out[14]:
```
(2010.0, 2015.0]    3914
(2015.0, inf]       2075
(2005.0, 2010.0]    1103
(-inf, 2005.0]       161
Name: Year, dtype: int64
```

In [15]:
```
data['Year_bin'] = pd.cut(
    data['Year'], [-np.inf, 2005, 2010, 2015, np.inf],
    labels = ["Below 2005", "2005 to 2010", "2010 to 2015","Above 2015"]
)

data.drop(['Year'], axis =1, inplace=True)
data['Year_bin'].value_counts()
```

Out[15]:
```
2010 to 2015    3914
Above 2015      2075
2005 to 2010    1103
Below 2005       161
Name: Year_bin, dtype: int64
```

Fuel_Type:

In [16]:
```
data['Fuel_Type'] = data['Fuel_Type'].astype("category")
```

Transmission:

In [17]:
```
data['Transmission'] = data['Transmission'].astype("category")
```

Owner:

In [18]:
```
def owner_to_num(val):
    return int(val.replace('First','1').replace('Second', '2').replace('Thir
d', '3').replace('Fourth & Above','4'))

data['Owner_Type'] = data['Owner_Type'].apply(owner_to_num)
data['Owner_Type'].head()
```

Out[18]:
```
0    1
1    1
2    1
3    1
4    2
Name: Owner_Type, dtype: int64
```

In [19]:
```
data['Owner_Type'].value_counts()
```

Out[19]:
```
1    5952
2    1152
3     137
4      12
Name: Owner_Type, dtype: int64
```

Mileage:

In [20]:
```
def mile_to_num(val): #running it twice will make everything Nan.
    if isinstance(val,str):
        return float(val.replace('km/kg','').replace('kmpl',''))
    else:
        return np.nan

data['Mileage'] = data['Mileage'].apply(mile_to_num)
data['Mileage'].head()
```

Out[20]:
```
0    26.60
1    19.67
2    18.20
3    20.77
4    15.20
Name: Mileage, dtype: float64
```

Engine:

In [21]:
```
def engine_to_num(val):
    if isinstance(val,str):
        return float(val.replace('CC',''))
    else:
        return np.nan

data['Engine'] = data['Engine'].apply(engine_to_num)
data['Engine'].head()
```

Out[21]:
```
0     998.0
1    1582.0
2    1199.0
3    1248.0
4    1968.0
Name: Engine, dtype: float64
```

Power:

In [22]:
```
def power_to_num(val):
    if isinstance(val,str):
        if val.startswith('null'): #
            return np.nan
        if val.endswith('bhp'):
            return float(val.replace('bhp',''))
    else:
        return np.nan

data['Power'] = data['Power'].apply(power_to_num)
data['Power'].head()
```

Out[22]:
```
0     58.16
1    126.20
2     88.70
3     88.76
4    140.80
Name: Power, dtype: float64
```

New_Price:

In [23]:
```
def newprice_to_num(val): #running it twice will make everything Nan.
    if isinstance(val,str):
        if val.endswith('Cr'):
            return float(val.replace('Cr','')) * 100 # price conversion to Lak
h 1CR = 100Lakh
        elif val.endswith('Lakh'):
            return float(val.replace('Lakh',''))
    else:
        return np.nan

data['New_Price'] = data['New_Price'].apply(newprice_to_num)
data['New_Price'].head()
```

Out[23]:
```
0     NaN
1     NaN
2    8.61
3     NaN
4     NaN
Name: New_Price, dtype: float64
```

Before going into null handling:

it is reasonable to drop (save separately) nan value for dependant variables check to see if any errors occurred

In [24]:
```
data = data[data['Price'].notna()] #it is mostly meaningless to include data w
ith missing dependent variable

data.shape
```

Out[24]:
```
(6019, 14)
```

In [25]:
```
data.info() # confirming memory usage reduction.
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 6019 entries, 0 to 6018
Data columns (total 14 columns):
Location             6019 non-null category
Kilometers_Driven    6019 non-null int64
Fuel_Type            6019 non-null category
Transmission         6019 non-null category
Owner_Type           6019 non-null int64
Mileage              6017 non-null float64
Engine               5983 non-null float64
Power                5876 non-null float64
Seats                5977 non-null float64
New_Price            824 non-null float64
Price                6019 non-null float64
Name_Brand           6019 non-null category
Name_Model           6019 non-null object
Year_bin             6019 non-null category
dtypes: category(5), float64(6), int64(2), object(1)
memory usage: 502.0+ KB
```

In [26]:
```
# Abvious Error Correction (seat can't be 0)
data[data['Seats'] ==0] # found index number and googled seating info of this car.
data.loc[3999,'Seats'] = 5 # correct seat info
```

In [27]:
```
data.isnull().sum().sort_values(ascending = False) #cheking null values
```

Out[27]:
```
New_Price            5195
Power                 143
Seats                  42
Engine                 36
Mileage                 2
Year_bin                0
Name_Model              0
Name_Brand              0
Price                   0
Owner_Type              0
Transmission            0
Fuel_Type               0
Kilometers_Driven       0
Location                0
dtype: int64
```

Treat Nan differently to see the change in model.

try median/mean imputation or dropping Nans

Check what provides the best linear regression model.

In [28]:
```
# now using `fillna` with a numeric column

data['Mileage'].fillna(data['Mileage'].median(), inplace=True)  # median imput
ation
data['Engine'].fillna(data['Engine'].median(), inplace=True)
data['Power'].fillna(data['Power'].median(), inplace=True)
data['Seats'].fillna(data['Seats'].median(), inplace=True)

data['New_Price'].fillna(data['New_Price'].median(), inplace=True) #corr of Ne
w Price without was 0.45 to Price: 1/0.45

# data['Mileage'].fillna(data['Mileage'].mean(), inplace=True)  # mean imputat
ion
# data['Engine'].fillna(data['Engine'].mean(), inplace=True)
# data['Power'].fillna(data['Power'].mean(), inplace=True)
# data['Seats'].fillna(data['Seats'].mean(), inplace=True)
# data['New_Price'].fillna(data['New_Price'].mean(), inplace=True)

# data = data.dropna() # drop na

#mean 18.941 median: 16.921 drop:1218572290
#mean: 0.782 median: 0.784 drop: -9024648661813


data.isnull().sum().sort_values(ascending = False) #cheking null values
```

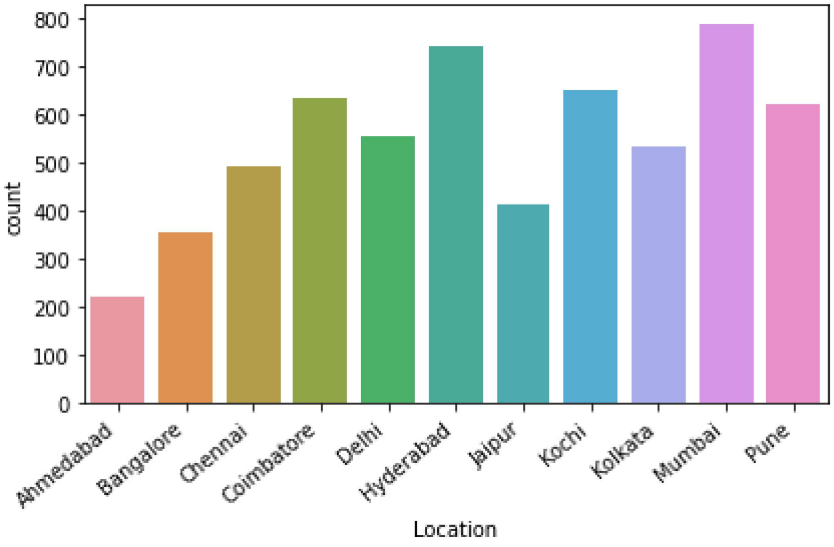
Out[28]:
```
Year_bin             0
Name_Model           0
Name_Brand           0
Price                0
New_Price            0
Seats                0
Power                0
Engine               0
Mileage              0
Owner_Type           0
Transmission         0
Fuel_Type            0
Kilometers_Driven    0
Location             0
dtype: int64
```

EDA Univariate analysis:

Categorical:

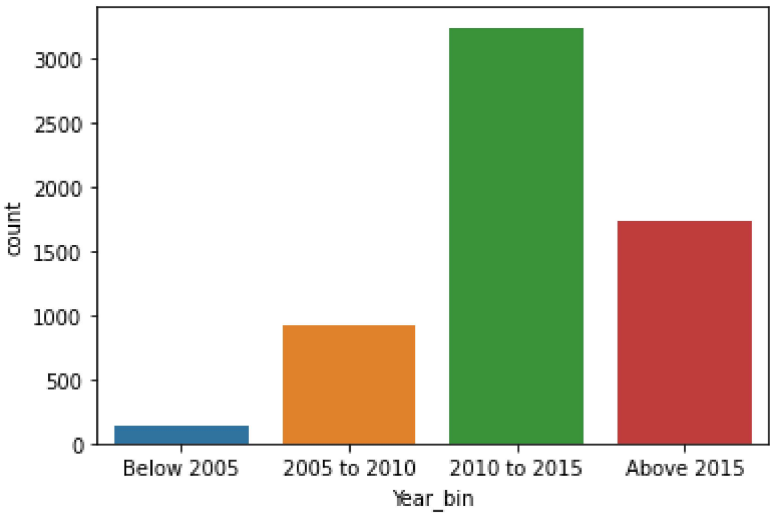
Location
Year_bin
Transmission
Fuel_Type
Name_Brand

In [29]:
```
location_plot = sns.countplot(data['Location'])
location_plot.set_xticklabels(location_plot.get_xticklabels(), rotation=40, ha
="right") # tilt text to prevent overlapping
plt.tight_layout()
plt.show()
```



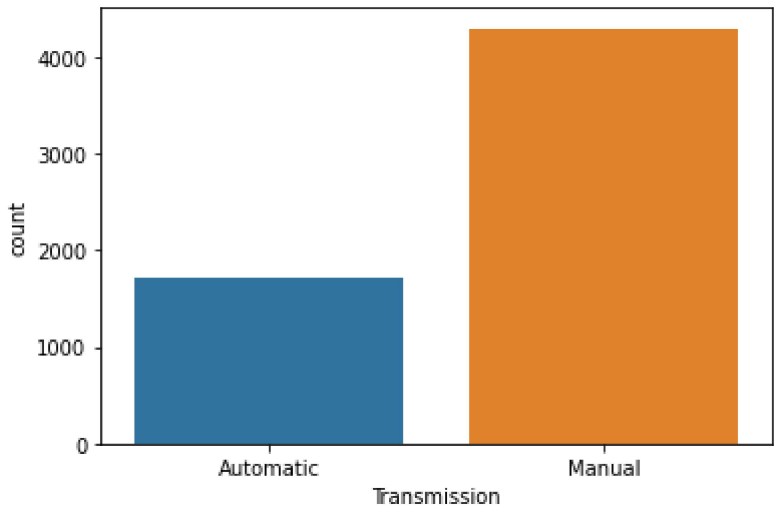
In [30]:
```
sns.countplot(data['Year_bin'])
```

Out[30]: <matplotlib.axes._subplots.AxesSubplot at 0x20c86de1d88>



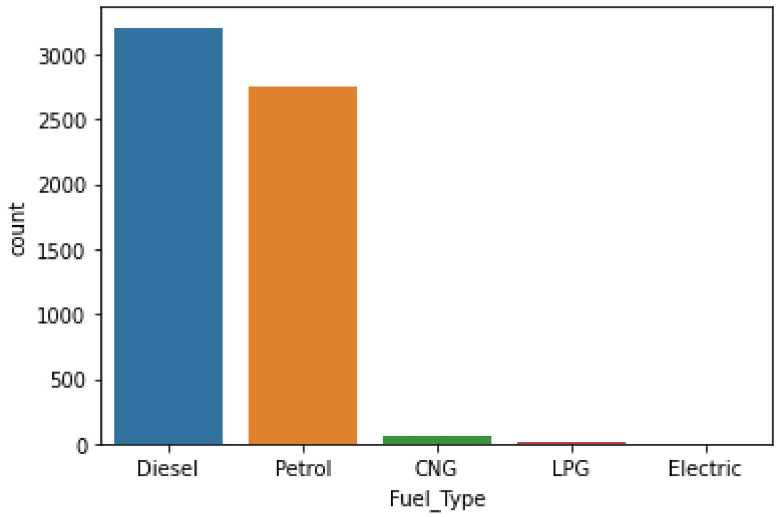
In [31]:
```
sns.countplot(data['Transmission'])
```

Out[31]: <matplotlib.axes._subplots.AxesSubplot at 0x20c86b04988>



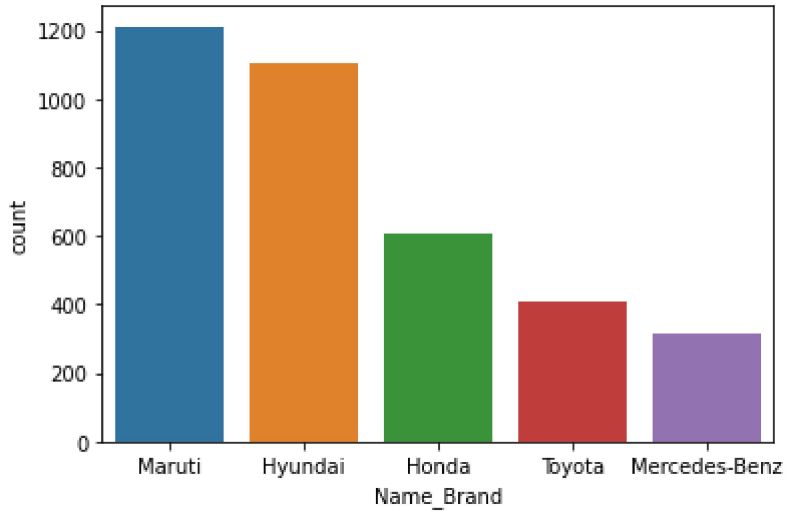
In [32]:
```
sns.countplot(data['Fuel_Type'], order=pd.value_counts(data['Fuel_Type']).inde
x)
```

Out[32]: <matplotlib.axes._subplots.AxesSubplot at 0x20c86a22a08>



In [33]:
```
sns.countplot(x='Name_Brand', data = data, order=pd.value_counts(data['Name_Brand']).iloc[:5].index) #showing top 5 brands
```

Out[33]: <matplotlib.axes._subplots.AxesSubplot at 0x20c86d3b988>



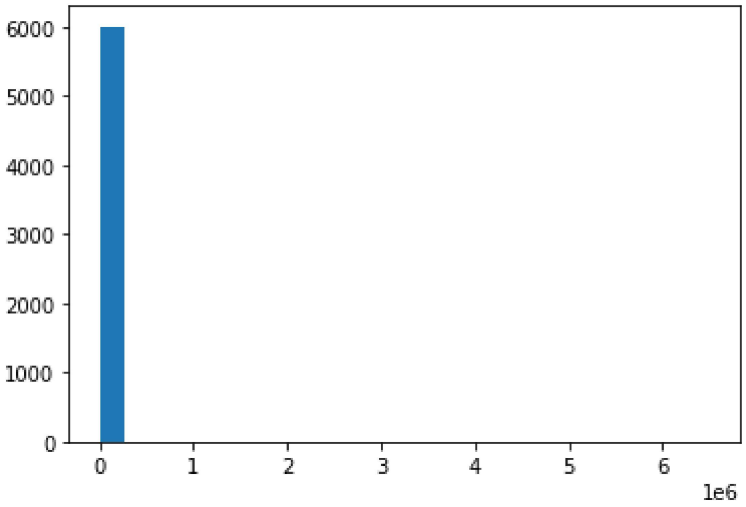
Numerical:

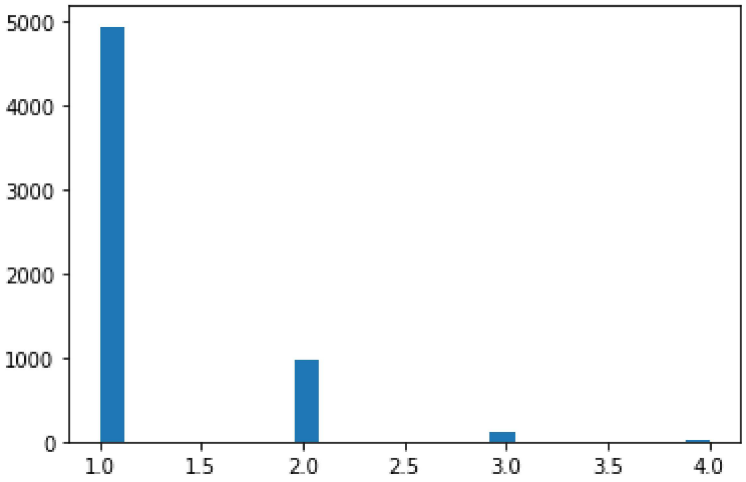
In [34]:
```
data.describe()
```

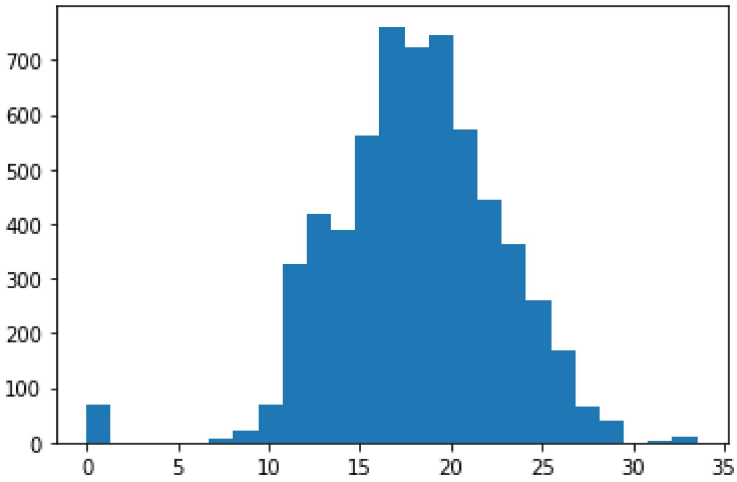
Out[34]:

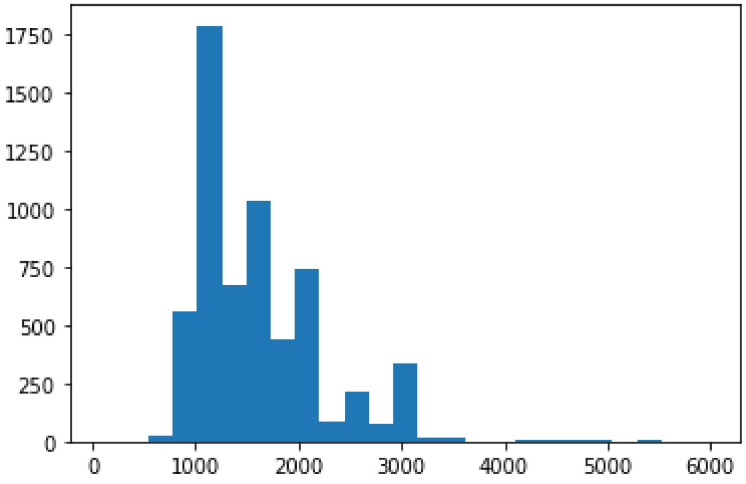
	Kilometers_Driven	Owner_Type	Mileage	Engine	Power	Seats	Ne
count	6.019000e+03	6019.000000	6019.000000	6019.000000	6019.000000	6019.000000	6019
mean	5.873838e+04	1.202858	18.134966	1620.509221	112.883539	5.277621	13
std	9.126884e+04	0.456356	4.581528	599.635458	53.283701	0.803479	10
min	1.710000e+02	1.000000	0.000000	72.000000	34.200000	2.000000	3
25%	3.400000e+04	1.000000	15.170000	1198.000000	78.000000	5.000000	11
50%	5.300000e+04	1.000000	18.150000	1493.000000	97.700000	5.000000	11
75%	7.300000e+04	1.000000	21.100000	1969.000000	138.030000	5.000000	11
max	6.500000e+06	4.000000	33.540000	5998.000000	560.000000	10.000000	230

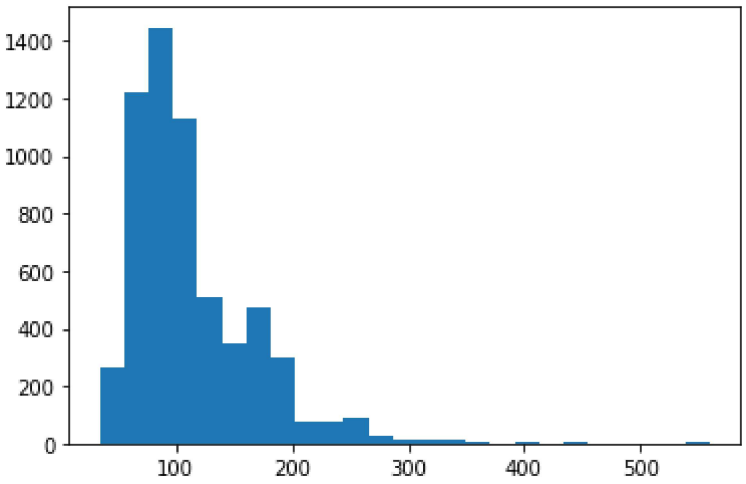
In [35]:
```
plt.hist(data['Kilometers_Driven'], bins =25)
plt.show()
plt.hist(data['Owner_Type'], bins =25)
plt.show()
plt.hist(data['Mileage'], bins =25)
plt.show()
plt.hist(data['Engine'], bins =25)
plt.show()
plt.hist(data['Power'], bins =25)
plt.show()
plt.hist(data['Seats'], bins =25)
plt.show()
plt.hist(data['New_Price'], bins =25)
plt.show()
plt.hist(data['Price'], bins =25)
plt.show()
```

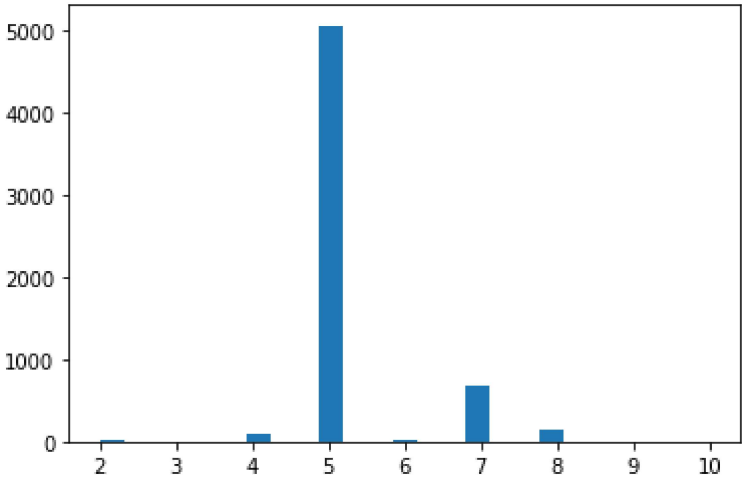

6000
5000
4000
3000
2000
1000
0
0
1
2
3
4
5
6
1e6

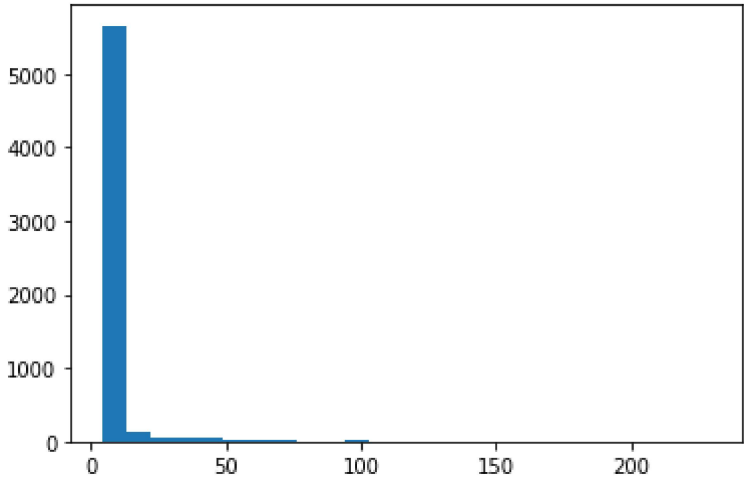

5000
4000
3000
2000
1000
0
1.0
1.5
2.0
2.5
3.0
3.5
4.0

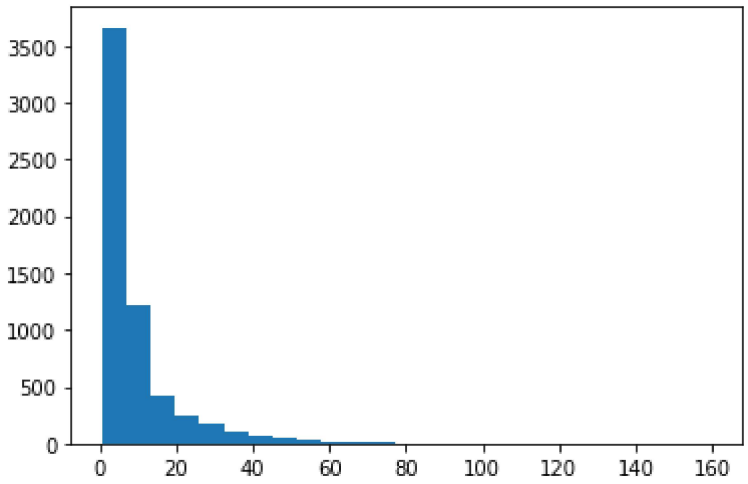

700
600
500
400
300
200
100
0
0
5
10
15
20
25
30
35


1750
1500
1250
1000
750
500
250
0
0
1000
2000
3000
4000
5000
6000


1400
1200
1000
800
600
400
200
0
100
200
300
400
500


5000
4000
3000
2000
1000
0
2
3
4
5
6
7
8
9
10

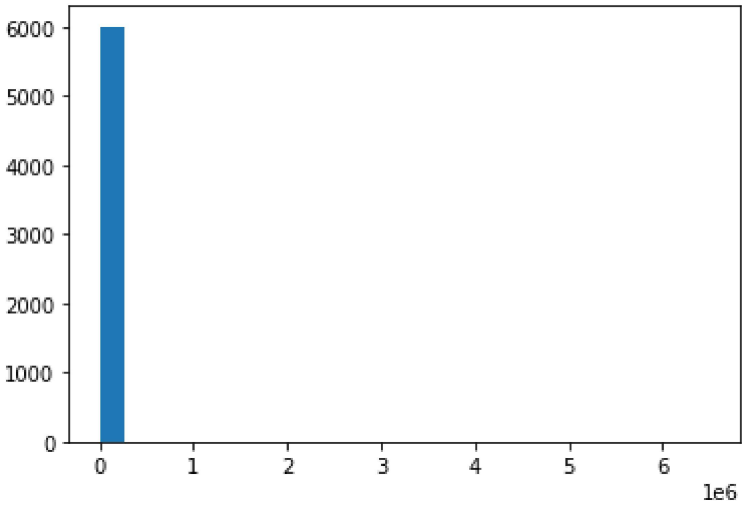


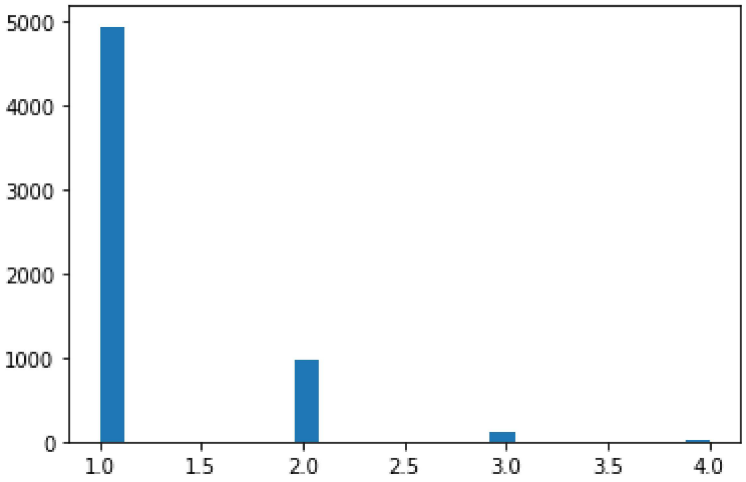


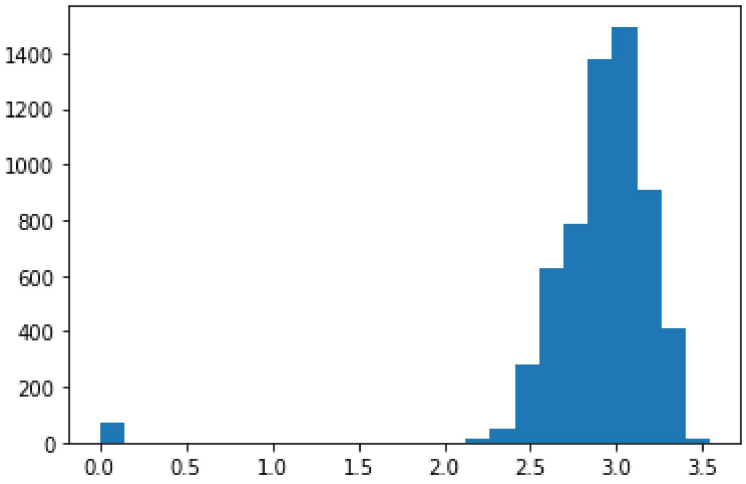
Fix Skewness

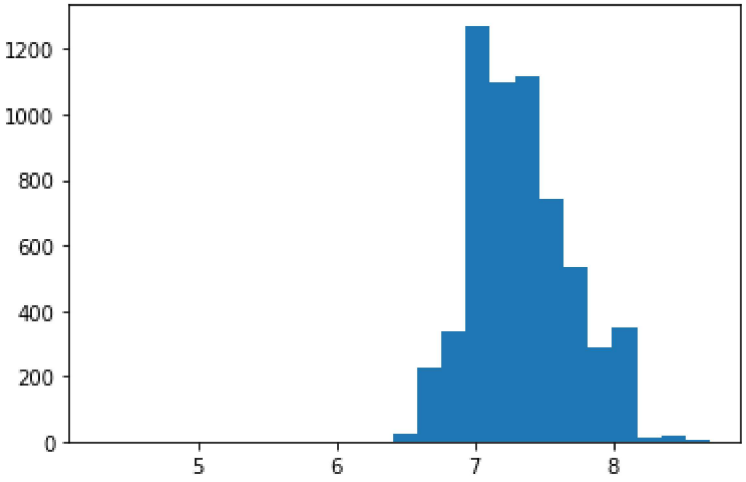
In [36]:
```
cols_to_log = ['Engine','Power','Mileage','Price']
for colname in cols_to_log:
    data[colname + '_log'] = np.log(data[colname] + 1)
data.drop(cols_to_log, axis=1, inplace=True)
```

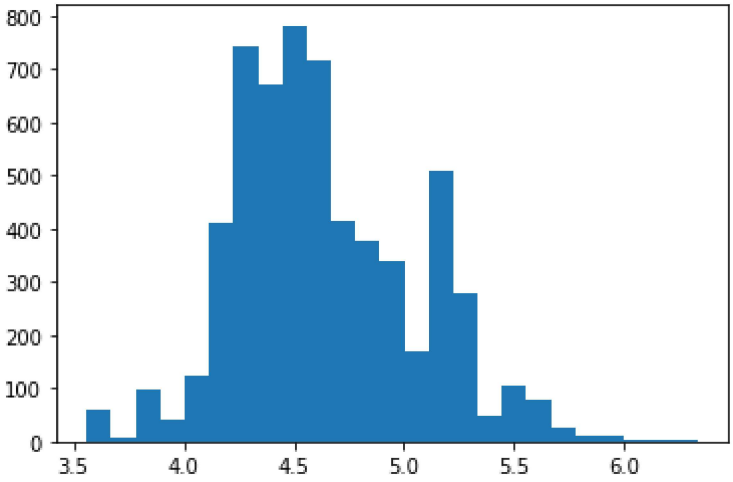
In [37]:
```
plt.hist(data['Kilometers_Driven'], bins =25)
plt.show()
plt.hist(data['Owner_Type'], bins =25)
plt.show()
plt.hist(data['Mileage_log'], bins =25)
plt.show()
plt.hist(data['Engine_log'], bins =25)
plt.show()
plt.hist(data['Power_log'], bins =25)
plt.show()
plt.hist(data['Seats'], bins =25)
plt.show()
plt.hist(data['New_Price'], bins =25)
plt.show()
plt.hist(data['Price_log'], bins =25)
plt.show()
```

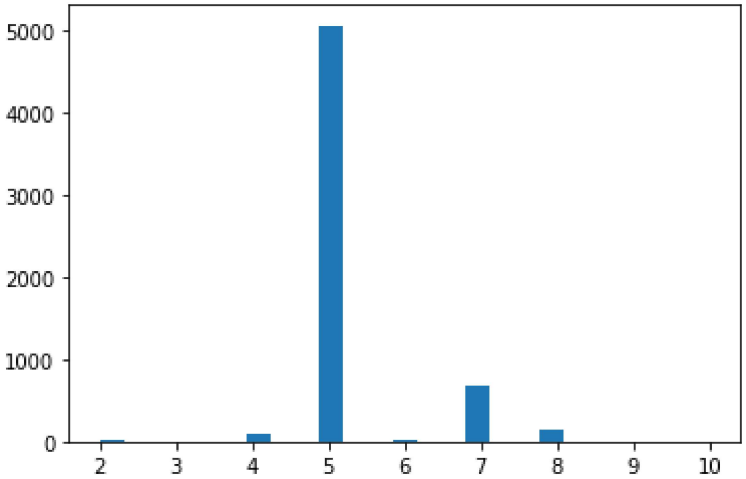

6000
5000
4000
3000
2000
1000
0
0
1
2
3
4
5
6
1e6

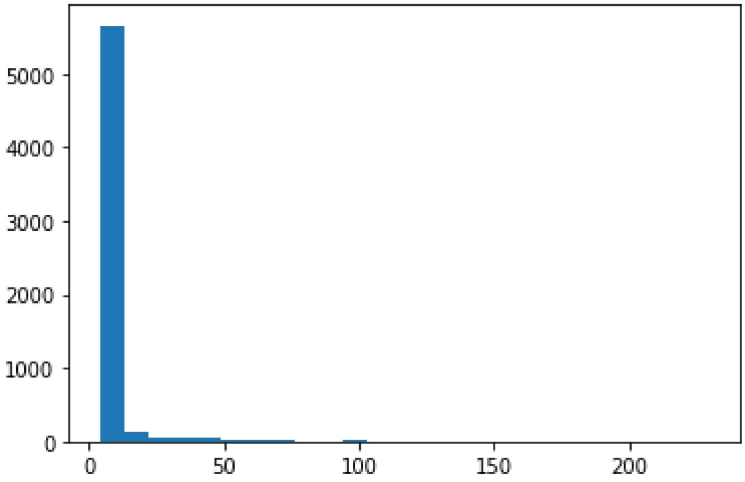

5000
4000
3000
2000
1000
0
1.0
1.5
2.0
2.5
3.0
3.5
4.0

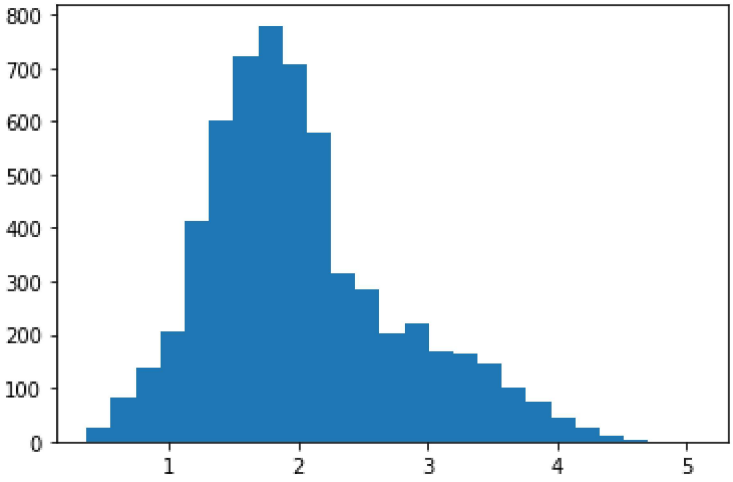

1400
1200
1000
800
600
400
200
0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5


1200
1000
800
600
400
200
0
5
6
7
8


800
700
600
500
400
300
200
100
0
3.5
4.0
4.5
5.0
5.5
6.0

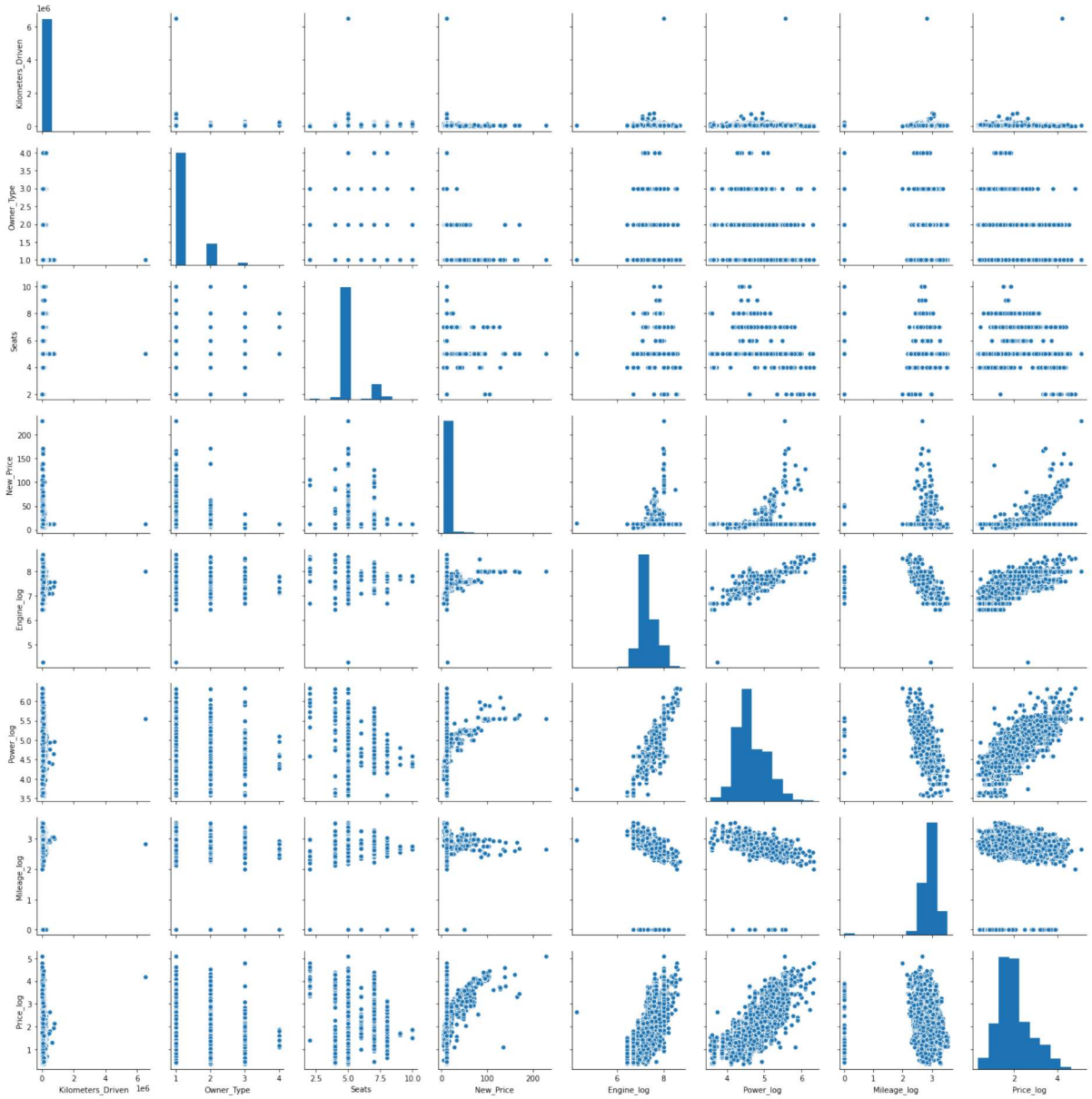

5000
4000
3000
2000
1000
0
2
3
4
5
6
7
8
9
10


5000
4000
3000
2000
1000
0
0
50
100
150
200


800
700
600
500
400
300
200
100
0
1
2
3
4
5

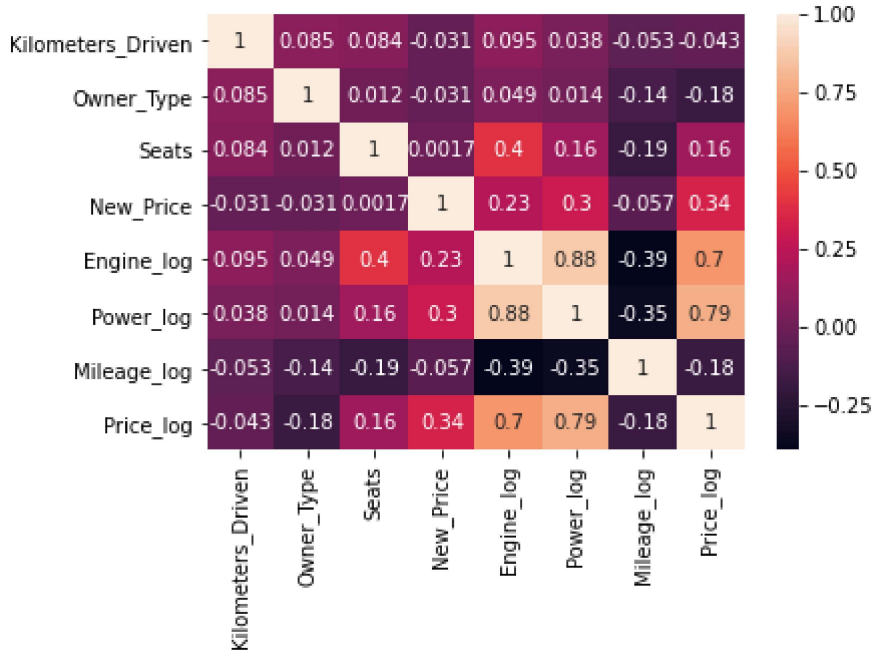
EDA Multivariable analysis:

In [38]:
```
sns.pairplot(data)
plt.show()
```



In [39]:
```
sns.heatmap(data.corr(), annot=True)
```

Out[39]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x20c8a094a08>
```



Conclusion after EDA:

Mileage, Engine, Power, New_Price are highly correlated with the Price of the Used Car. Skewness of Price, New_Price, Power was heavily right skewed. changing units can help enhancing the model. power is correlated with engine, and seats are bigger with bigger engines as well.

Linear Regression:

In [40]:
```
# Create dummy variables
data = pd.get_dummies(data, columns=['Location',
'Year_bin',
'Transmission',
'Fuel_Type', 'Name_Brand'])

data.shape
```

Out[40]:
```
(6019, 64)
```

In [41]:
```
# data.info()
```

In [42]:
```
# Separate dependent with independent
X = data.drop(['Price_log', 'Name_Model'], axis=1)
y = data[['Price_log']]
```

In [43]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, rand
om_state=1)
```

In [44]:
```
reg_mod = LinearRegression()
reg_mod.fit(X_train, y_train)
```

Out[44]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fals
e)
```

In [45]:
```
for idx, col_name in enumerate(X_train.columns):
    print("The coefficient for {} is {}".format(col_name, reg_mod.coef_[0][id
x]))
```

```
The coefficient for Kilometers_Driven is -1.6839521546657056e-06
The coefficient for Owner_Type is -0.08345600922747907
The coefficient for Seats is 0.04466235682656017
The coefficient for New_Price is 0.0018873697564917302
The coefficient for Engine_log is 0.3231778414354298
The coefficient for Power_log is 0.6013135587079671
The coefficient for Mileage_log is 0.0024395170434623054
The coefficient for Location_Ahmedabad is -0.014921467883009631
The coefficient for Location_Bangalore is 0.13465863657721872
The coefficient for Location_Chennai is -0.007560843060089528
The coefficient for Location_Coimbatore is 0.1390603375344206
The coefficient for Location_Delhi is -0.07287827611699738
The coefficient for Location_Hyderabad is 0.09979790839031988
The coefficient for Location_Jaipur is -0.02128423453624971
The coefficient for Location_Kochi is 0.011843049726440291
The coefficient for Location_Kolkata is -0.19355216447351278
The coefficient for Location_Mumbai is -0.04794961915552907
The coefficient for Location_Pune is -0.027213327002908072
The coefficient for Year_bin_Below 2005 is -0.5154707830429484
The coefficient for Year_bin_2005 to 2010 is -0.2027798608517177
The coefficient for Year_bin_2010 to 2015 is 0.20443505006026777
The coefficient for Year_bin_Above 2015 is 0.5138155938345617
The coefficient for Transmission_Automatic is 0.05502586590576142
The coefficient for Transmission_Manual is -0.0550258659057804
The coefficient for Fuel_Type_CNG is -0.12401312097734302
The coefficient for Fuel_Type_Diesel is -0.019460639389973333
The coefficient for Fuel_Type_Electric is 0.5061173765223267
The coefficient for Fuel_Type_LPG is -0.18618727535472465
The coefficient for Fuel_Type_Petrol is -0.17645634080042893
The coefficient for Name_Brand_Ambassador is 0.20071970266696093
The coefficient for Name_Brand_Audi is 0.32640118255846445
The coefficient for Name_Brand_BMW is 0.2616671922004596
The coefficient for Name_Brand_Bentley is 0.8077671745800807
The coefficient for Name_Brand_Chevrolet is -0.5408500147897137
The coefficient for Name_Brand_Datsun is -0.5711953112073261
The coefficient for Name_Brand_Fiat is -0.48898068204999945
The coefficient for Name_Brand_Force is -0.1453307421536783
The coefficient for Name_Brand_Ford is -0.2911048870056053
The coefficient for Name_Brand_Hindustan is -6.661338147750939e-16
The coefficient for Name_Brand_Honda is -0.29660500231281073
The coefficient for Name_Brand_Hyundai is -0.2829231372024167
The coefficient for Name_Brand_ISUZU is -0.6852365162695306
The coefficient for Name_Brand_Isuzu is -6.938893903907228e-17
The coefficient for Name_Brand_Jaguar is 0.4316065470771223
The coefficient for Name_Brand_Jeep is -0.004832374552820984
The coefficient for Name_Brand_Lamborghini is 1.4989080610016465
The coefficient for Name_Brand_Land is 0.6353935580788607
The coefficient for Name_Brand_Mahindra is -0.4215021761614607
The coefficient for Name_Brand_Maruti is -0.23123645477668753
The coefficient for Name_Brand_Mercedes-Benz is 0.3223705086867986
The coefficient for Name_Brand_Mini is 0.6349195118318303
The coefficient for Name_Brand_Mitsubishi is -0.05625508867694809
The coefficient for Name_Brand_Nissan is -0.3158532224208649
The coefficient for Name_Brand_OpelCorsa is 1.1102230246251565e-16
The coefficient for Name_Brand_Porsche is 0.47783701121759864
The coefficient for Name_Brand_Renault is -0.2872131014892012
The coefficient for Name_Brand_Skoda is -0.26374724963435286
```

```
The coefficient for Name_Brand_Smart is 0.027993803345557067
The coefficient for Name_Brand_Tata is -0.5421382599429363
The coefficient for Name_Brand_Toyota is -0.062021658774411986
The coefficient for Name_Brand_Volkswagen is -0.3166080806754847
The coefficient for Name_Brand_Volvo is 0.17804970685092616
```

In [46]:
```
interc = reg_mod.intercept_[0]
print(f"The intercept for our model is {interc}") #mean 18.941 median: 16.921 drop:1218572290 after scaling:-3.13
```

```
The intercept for our model is -3.1398640392136112
```

In [47]:
```
reg_mod.score(X_train, y_train) #mean: 0.965 median: 0.965 drop: 0.984 after scaling:0.906
```

Out[47]:
```
0.9063645900599933
```

In [48]:
```
#out of sample score (R^2)

reg_mod.score(X_test, y_test) #mean: 0.782 median: 0.784 drop: -9024648661813 after scaling:0.789

#(Q for discussion: Would corrlation imputation work?)
```

Out[48]:
```
0.789139042941318
```

Final Conclusion:

After Conducting EDA and linear Regression, it is clear that Year, Power, Engine, and Mileage are highly correlated with the Price. Test set R^2 is 0.789 which is well constructed model for linear regression. Outlier treatment and more scaling process to compare could increase in model R^2.

It would be very benefitial to collect more accurate data in the price of the car in new condition, since it is expected to be highly correlated with the price of used car. Considering adding more stats of purchasing the used car from the original rider to have two different dependent variables. It is also very important to keep the buying and selling data disclosed to other business.

When purchasing the used car, it is still important to firmly check the conditions of the car. the features stated in the model don't count every details of the used car. (e.g accidents).

mini brand, electric fuel types can be very popular.

In [ ]:
```
```