The goal is to create a pricing model that can effectively predict the price of a used car.

In [1]:
```
%load_ext nb_black
```

In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

sns.set()

import warnings

warnings.filterwarnings("ignore")

pd.set_option("display.max_columns", None)
pd.set_option("display.max_rows", 200)

from sklearn.linear_model import LinearRegression

from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
```

In [3]:
```
data = pd.read_csv("used_cars_data.csv")

df = data.copy()
```

Viewing a sample of the dataset.

In [4]:
```
np.random.seed(2)
df.sample(10)
```

Out[4]:

	S.No.	Name	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_T
4584	4584	Tata Tigor 1.05 Revotorq XT	Kochi	2018	28973	Diesel	Manual	
6505	6505	Volkswagen Vento Diesel Highline	Chennai	2011	76041	Diesel	Manual	
3675	3675	Maruti Swift VDI	Ahmedabad	2012	65000	Diesel	Manual	
5654	5654	Hyundai i20 Magna Optional 1.2	Kochi	2014	42315	Petrol	Manual	
4297	4297	Toyota Camry 2.5	Mumbai	2014	68400	Petrol	Automatic	

	S.No.	Name	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_T
		G						
2603	2603	Mercedes-Benz New C-Class 220 CDI AT	Jaipur	2010	74213	Diesel	Automatic	
4337	4337	Volkswagen Vento Petrol Highline AT	Kochi	2014	32283	Petrol	Automatic	Sec
6625	6625	Maruti Swift VDI BSIV	Kolkata	2012	72000	Diesel	Manual	
2846	2846	Skoda Superb Elegance 1.8 TSI AT	Kochi	2011	73783	Petrol	Automatic	Sec
1237	1237	Audi Q3 2.0 TDI Quattro	Hyderabad	2013	60000	Diesel	Automatic	

Checking the shape of the data.

In [5]:
```
print(f"There are {df.shape[0]} rows and {df.shape[1]} columns.")
```

```
There are 7253 rows and 14 columns.
```

Checking for duplicate rows in the data.

In [6]:
```
df.duplicated().sum()
```

Out[6]: 0

Checking the columns in the data.

In [7]:
```
print(df.columns)
```

```
Index(['S.No.', 'Name', 'Location', 'Year', 'Kilometers_Driven', 'Fuel_Type',
       'Transmission', 'Owner_Type', 'Mileage', 'Engine', 'Power', 'Seats',
       'New_Price', 'Price'],
      dtype='object')
```

Checking the data types of the columns.

In [8]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7253 entries, 0 to 7252
Data columns (total 14 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
```

```
 0   S.No.              7253 non-null   int64
 1   Name               7253 non-null   object
 2   Location           7253 non-null   object
 3   Year               7253 non-null   int64
 4   Kilometers_Driven  7253 non-null   int64
 5   Fuel_Type          7253 non-null   object
 6   Transmission       7253 non-null   object
 7   Owner_Type         7253 non-null   object
 8   Mileage            7251 non-null   object
 9   Engine             7207 non-null   object
 10  Power              7207 non-null   object
 11  Seats              7200 non-null   float64
 12  New_Price          1006 non-null   object
 13  Price              6019 non-null   float64
dtypes: float64(2), int64(3), object(9)
memory usage: 793.4+ KB
```

Checking for missing values in the data.

In [9]:
```
df.isnull().sum()
```

Out[9]:
```
S.No.                   0
Name                    0
Location                0
Year                    0
Kilometers_Driven       0
Fuel_Type               0
Transmission            0
Owner_Type              0
Mileage                 2
Engine                 46
Power                  46
Seats                  53
New_Price            6247
Price                1234
dtype: int64
```

Summary of the dataset

In [10]:
```
df.describe(include="all").T
```

Out[10]:

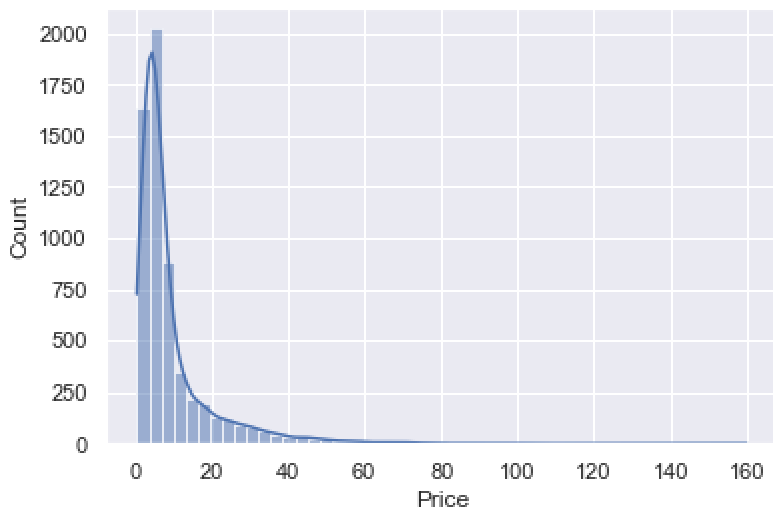
	count	unique	top	freq	mean	std	min	25%	50%	75%
S.No.	7253	NaN	NaN	NaN	3626	2093.91	0	1813	3626	5439
Name	7253	2041	Mahindra XUV500 W8 2WD	55	NaN	NaN	NaN	NaN	NaN	NaN
Location	7253	11	Mumbai	949	NaN	NaN	NaN	NaN	NaN	NaN
Year	7253	NaN	NaN	NaN	2013.37	3.25442	1996	2011	2014	2016
Kilometers_Driven	7253	NaN	NaN	NaN	58699.1	84427.7	171	34000	53416	73000
Fuel_Type	7253	5	Diesel	3852	NaN	NaN	NaN	NaN	NaN	NaN
Transmission	7253	2	Manual	5204	NaN	NaN	NaN	NaN	NaN	NaN
Owner_Type	7253	4	First	5952	NaN	NaN	NaN	NaN	NaN	NaN
Mileage	7251	450	17.0 kmpl	207	NaN	NaN	NaN	NaN	NaN	NaN

	count	unique	top	freq	mean	std	min	25%	50%	75%
Engine	7207	150	1197 CC	732	NaN	NaN	NaN	NaN	NaN	NaN
Power	7207	386	74 bhp	280	NaN	NaN	NaN	NaN	NaN	NaN
Seats	7200	NaN	NaN	NaN	5.27972	0.81166	0	5	5	5
New_Price	1006	625	33.36 Lakh	6	NaN	NaN	NaN	NaN	NaN	NaN
Price	6019	NaN	NaN	NaN	9.47947	11.1879	0.44	3.5	5.64	9.95

Initial Data Visualization

In [11]:
```
sns.histplot(data=df, x="Price", bins=50, kde=True)
```

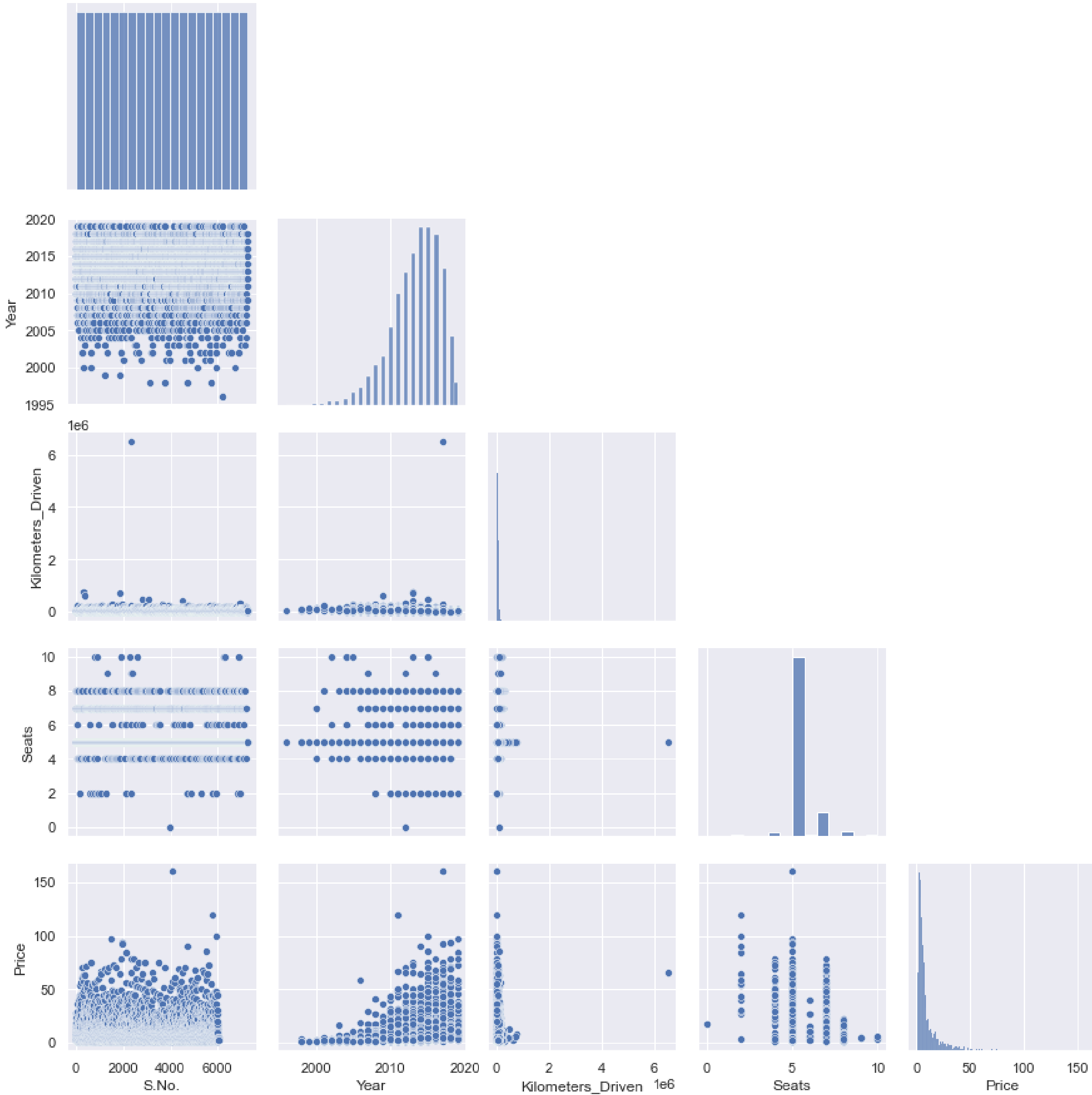
Out[11]: <AxesSubplot:xlabel='Price', ylabel='Count'>



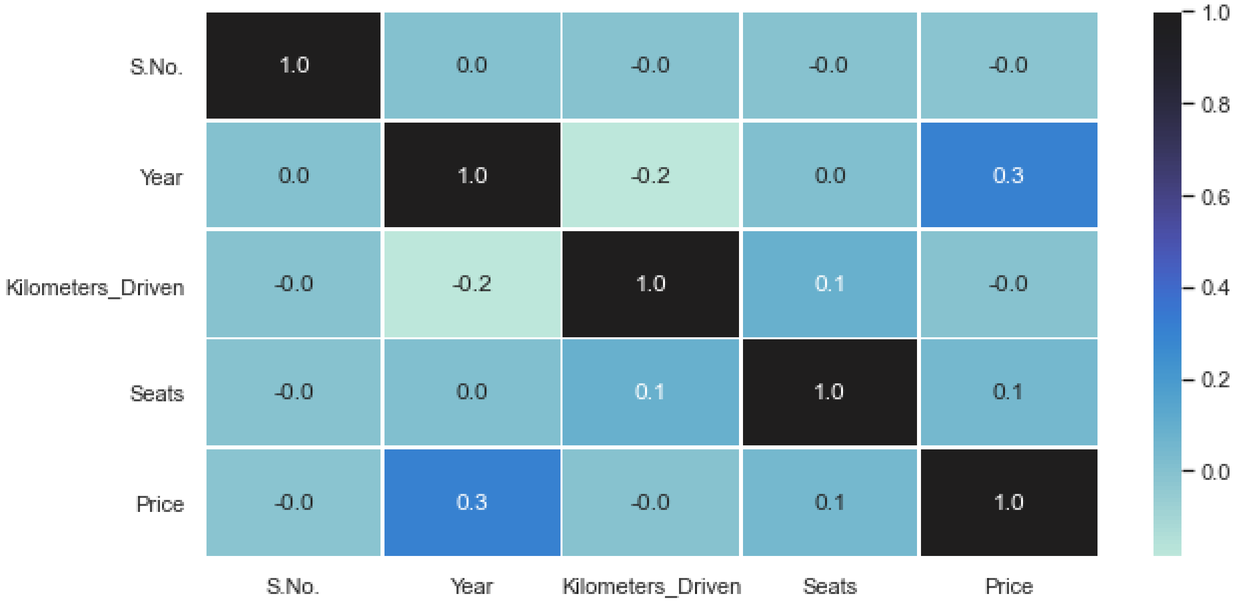
- Most of the cars sell around 9.5 Lakh.

In [12]:
```
sns.pairplot(df, corner=True)
```

Out[12]: <seaborn.axisgrid.PairGrid at 0x28f2b2d6520>



In [13]:
```
plt.figure(figsize=(10, 5))
sns.heatmap(data.corr(), annot=True, linewidths=0.5, fmt=".1f", center=1)
plt.show()
```



Data Preprocessing

Droping missing values (rows) in 'Price' column (dependent variable/target variable).

In [14]:
```
df.dropna(subset=["Price"], inplace=True)
```

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

Out[15]:
```
(6019, 14)
```

Checking missing values in rest of the data.

In [16]:
```
df.isnull().sum()
```

Out[16]:
```
S.No.                   0
Name                    0
Location                0
Year                    0
Kilometers_Driven       0
Fuel_Type               0
Transmission            0
Owner_Type              0
Mileage                 2
Engine                 36
Power                  36
Seats                  42
New_Price            5195
Price                   0
dtype: int64
```

Going to drop the 'S.No.' column as it seems pointless to know.

In [17]:
```
df.drop(["S.No."], axis=1, inplace=True)
df.head()
```

Out[17]:

	Name	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Mileage
0	Maruti Wagon R LXI CNG	Mumbai	2010	72000	CNG	Manual	First	26.6 km/kg
1	Hyundai Creta 1.6 CRDi SX Option	Pune	2015	41000	Diesel	Manual	First	19.67 kmp
2	Honda Jazz V	Chennai	2011	46000	Petrol	Manual	First	18.2 kmp
3	Maruti Ertiga VDI	Chennai	2012	87000	Diesel	Manual	First	20.77 kmp
4	Audi A4 New 2.0 TDI Multitronic	Coimbatore	2013	40670	Diesel	Automatic	Second	15.2 kmp

Going to work on columns with missing values first.

Mileage - km/kg

In [18]:
```
def mileage_to_num(Mileage_val):
    if isinstance(Mileage_val, str):
        if Mileage_val.endswith(" km/kg") or (" kmpl"):
            return float(Mileage_val.strip(" km/kg").strip(" kmpl"))
    else:
        return np.nan
```

In [19]:
```
for colname in ["Mileage"]:
    df[colname] = df[colname].apply(mileage_to_num)
```

In [20]:
```
df["Mileage"].head()
```

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

- Stripped the 'km/kg' and 'kmpl' and converted 'Mileage' to a float dtype.

In [21]:
```
df["Mileage"].mean()
```

Out[21]:
```
18.134960943992073
```

In [22]:
```
print(df["Mileage"].isnull().sum())
df["Mileage"].fillna(df["Mileage"].mean(), inplace=True)
df["Mileage"].isnull().sum()
```

```
2
```

Out[22]:
```
0
```

- There were 2 missing values, so I replaced them with the mean value. Now there are 0 missing values.

In [23]:
```
print(df[df["Mileage"] == 0.0].index.values)
```

```
[  14   67   79  194  229  262  307  424  443  544  631  647  707  749
  915  962  996 1059 1259 1271 1308 1345 1354 1385 1419 1460 1764 1857
 2053 2096 2130 2267 2343 2542 2597 2681 2780 2842 3033 3044 3061 3093
 3189 3210 3271 3516 3522 3645 4152 4234 4302 4412 4629 4687 4704 5016
 5022 5119 5270 5311 5374 5426 5529 5647 5875 5943 5972 6011]
```

In [24]:
```
df["Mileage"].median()
```

Out[24]:
```
18.15
```

In [25]:
```
df["Mileage"].replace(0.0, 18.15, inplace=True)
```

In [26]:
```
print(df[df["Mileage"] == 0.0].index.values)
```

```
[]
```

In [27]:
```
print(df[df["Mileage"] == 18.15].index.values)
```

```
[  14   37   67   79  194  229  262  307  424  443  544  631  647  707
  749  915  962  996 1059 1259 1271 1308 1345 1354 1385 1419 1460 1764
 1857 2053 2096 2130 2267 2343 2542 2597 2681 2696 2780 2842 2994 3033
 3044 3061 3093 3189 3210 3271 3503 3513 3516 3522 3569 3645 4152 4234
 4302 4412 4629 4687 4704 5016 5022 5119 5270 5311 5374 5426 5529 5647
 5736 5807 5875 5943 5972 6011]
```

- Noticed there were quite a few '0.0' values, so I replaced them with the median.

Checking columns with missing values again.

In [28]:
```
df.isnull().sum()
```

Out[28]:
```
Name                  0
Location              0
Year                  0
Kilometers_Driven     0
Fuel_Type             0
Transmission          0
Owner_Type            0
Mileage               0
Engine               36
Power                36
```

```
Seats               42
New_Price         5195
Price                0
dtype: int64
```

Engine - CC

In [29]:
```
def engine_to_num(Engine_val):
    if isinstance(Engine_val, str):
        if Engine_val.endswith(" CC"):
            return float(Engine_val.strip(" CC"))
    else:
        return np.nan
```

In [30]:
```
for colname in ["Engine"]:
    df[colname] = df[colname].apply(engine_to_num)
```

In [31]:
```
df["Engine"].head()
```

Out[31]:
```
0     998.0
1    1582.0
2    1199.0
3    1248.0
4    1968.0
Name: Engine, dtype: float64
```

- Stripped the 'CC' and converted 'Engine' to a float dtype.

In [32]:
```
df["Engine"].mean()
```

Out[32]:
```
1621.276449941501
```

In [33]:
```
print(df["Engine"].isnull().sum())
df["Engine"].fillna(df["Engine"].mean(), inplace=True)
df["Engine"].isnull().sum()
```

```
36
```

Out[33]:
```
0
```

- There were 36 missing values, so I replaced them with the mean value. Now there are 0 missing values.

Checking columns with missing values again.

In [34]:
```
df.isnull().sum()
```

Out[34]:
```
Name                 0
Location             0
Year                 0
Kilometers_Driven    0
Fuel_Type            0
```

```
Transmission          0
Owner_Type            0
Mileage               0
Engine                0
Power                36
Seats                42
New_Price          5195
Price                 0
dtype: int64
```

Power - bhp

I see that there are some values that are 'null bhp', so I'm going to replace them with '0 bhp. That way I can strip bhp off of the values with numbers. Will replace the 0s with NaNs, then NaNs with the mean.

In [35]:
```
df["Power"].replace("null bhp", "0 bhp", inplace=True)
```

In [36]:
```
print(df[df["Power"] == "0 bhp"].index.values)
```

```
[  76   79   89  120  143  227  245  262  307  308  386  424  428  443
  472  575  631  647  648  739  748  829  915  926  934 1068 1143 1153
 1271 1319 1345 1388 1419 1555 1578 1649 1672 1857 1999 2053 2130 2164
 2262 2267 2305 2343 2369 2393 2441 2450 2497 2501 2527 2579 2597 2635
 2640 2891 3033 3061 3104 3189 3247 3290 3439 3516 3533 3589 3628 3638
 3645 3669 3733 3800 3882 3898 3930 3999 4077 4080 4351 4354 4629 4709
 4714 4744 4830 4886 4900 4954 5065 5119 5228 5426 5438 5458 5529 5533
 5647 5755 5759 5861 5873 5893 5925 5943 5985]
```

In [37]:
```
def power_to_num(Power_val):
    if isinstance(Power_val, str):
        if Power_val.endswith(" bhp"):
            return float(Power_val.strip(" bhp"))
    else:
        return np.nan
```

In [38]:
```
for colname in ["Power"]:
    df[colname] = df[colname].apply(power_to_num)
```

In [39]:
```
print(df[df["Power"] == 0].index.values)
```

```
[  76   79   89  120  143  227  245  262  307  308  386  424  428  443
  472  575  631  647  648  739  748  829  915  926  934 1068 1143 1153
 1271 1319 1345 1388 1419 1555 1578 1649 1672 1857 1999 2053 2130 2164
 2262 2267 2305 2343 2369 2393 2441 2450 2497 2501 2527 2579 2597 2635
 2640 2891 3033 3061 3104 3189 3247 3290 3439 3516 3533 3589 3628 3638
 3645 3669 3733 3800 3882 3898 3930 3999 4077 4080 4351 4354 4629 4709
 4714 4744 4830 4886 4900 4954 5065 5119 5228 5426 5438 5458 5529 5533
 5647 5755 5759 5861 5873 5893 5925 5943 5985]
```

In [40]:
```
df["Power"].replace(0, np.nan, inplace=True)
```

In [41]:
```
print(df[df["Power"] == 0].index.values)
```

```
[]
```

In [42]:
```
df.isnull().sum()
```

Out[42]:
```
Name                    0
Location                0
Year                    0
Kilometers_Driven       0
Fuel_Type               0
Transmission            0
Owner_Type              0
Mileage                 0
Engine                  0
Power                 143
Seats                  42
New_Price            5195
Price                   0
dtype: int64
```

In [43]:
```
df["Power"].mean()
```

Out[43]:
```
113.25304969366827
```

In [44]:
```
print(df["Power"].isnull().sum())
df["Power"].fillna(df["Power"].mean(), inplace=True)
df["Power"].isnull().sum()
```

```
143
```

Out[44]:
```
0
```

- There were 143 missing values, so I replaced them with the mean value. Now there are 0 missing values.

Checking columns with missing values again.

In [45]:
```
df.isnull().sum()
```

Out[45]:
```
Name                    0
Location                0
Year                    0
Kilometers_Driven       0
Fuel_Type               0
Transmission            0
Owner_Type              0
Mileage                 0
Engine                  0
Power                   0
Seats                  42
New_Price            5195
Price                   0
dtype: int64
```

Seats

In [46]:
```
df["Seats"].mean()
```
Out[46]:
```
5.278735151413753
```

In [47]:
```
print(df["Seats"].isnull().sum())
df["Seats"].fillna(5, inplace=True)
df["Seats"].isnull().sum()
```
```
42
```
Out[47]:
```
0
```

- There were 42 missing values, so I replaced them with a value of 5. Now there are 0 missing values.

Checking columns with missing values again.

In [48]:
```
df.isnull().sum()
```
Out[48]:
```
Name                    0
Location                0
Year                    0
Kilometers_Driven       0
Fuel_Type               0
Transmission            0
Owner_Type              0
Mileage                 0
Engine                  0
Power                   0
Seats                   0
New_Price            5195
Price                   0
dtype: int64
```

New_Price - Lakh

Given there are a lot of missing values, and that the new price of a car can depend on the make, location being sold, and possibly the economy, I will drop this column. Most cars anyway depreciate soon after leaving the dealer. We are interested in what a used car will sell at and I assume the used cars sell in relating to their original price.

In [49]:
```
df.drop(["New_Price"], axis=1, inplace=True)
df.head()
```
Out[49]:

	Name	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Mileag
0	Maruti Wagon R LXI CNG	Mumbai	2010	72000	CNG	Manual	First	26.6(

	Name	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Mileag
1	Hyundai Creta 1.6 CRDi SX Option	Pune	2015	41000	Diesel	Manual	First	19.6
2	Honda Jazz V	Chennai	2011	46000	Petrol	Manual	First	18.2
3	Maruti Ertiga VDI	Chennai	2012	87000	Diesel	Manual	First	20.7
4	Audi A4 New 2.0 TDI Multitronic	Coimbatore	2013	40670	Diesel	Automatic	Second	15.2

In [50]:
```
df.isnull().sum()
```

Out[50]:
```
Name                 0
Location             0
Year                 0
Kilometers_Driven    0
Fuel_Type            0
Transmission         0
Owner_Type           0
Mileage              0
Engine               0
Power                0
Seats                0
Price                0
dtype: int64
```

Updated Visualization

In [51]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 6019 entries, 0 to 6018
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Name               6019 non-null   object 
 1   Location           6019 non-null   object 
 2   Year               6019 non-null   int64  
 3   Kilometers_Driven  6019 non-null   int64  
 4   Fuel_Type          6019 non-null   object 
 5   Transmission       6019 non-null   object 
 6   Owner_Type         6019 non-null   object 
 7   Mileage            6019 non-null   float64
 8   Engine             6019 non-null   float64
 9   Power              6019 non-null   float64
 10  Seats              6019 non-null   float64
 11  Price              6019 non-null   float64
dtypes: float64(5), int64(2), object(5)
memory usage: 611.3+ KB
```

In [52]:
```
to_convert = ["Name", "Location", "Fuel_Type", "Transmission", "Owner_Type"]
df[to_convert] = df[to_convert].astype("category")
```

In [53]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 6019 entries, 0 to 6018
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype   
---  ------             --------------  -----   
 0   Name               6019 non-null   category
 1   Location           6019 non-null   category
 2   Year               6019 non-null   int64   
 3   Kilometers_Driven  6019 non-null   int64   
 4   Fuel_Type          6019 non-null   category
 5   Transmission       6019 non-null   category
 6   Owner_Type         6019 non-null   category
 7   Mileage            6019 non-null   float64 
 8   Engine             6019 non-null   float64 
 9   Power              6019 non-null   float64 
 10  Seats              6019 non-null   float64 
 11  Price              6019 non-null   float64 
dtypes: category(5), float64(5), int64(2)
memory usage: 507.0 KB
```

- Coverted the 'object' dtypes to 'category'.

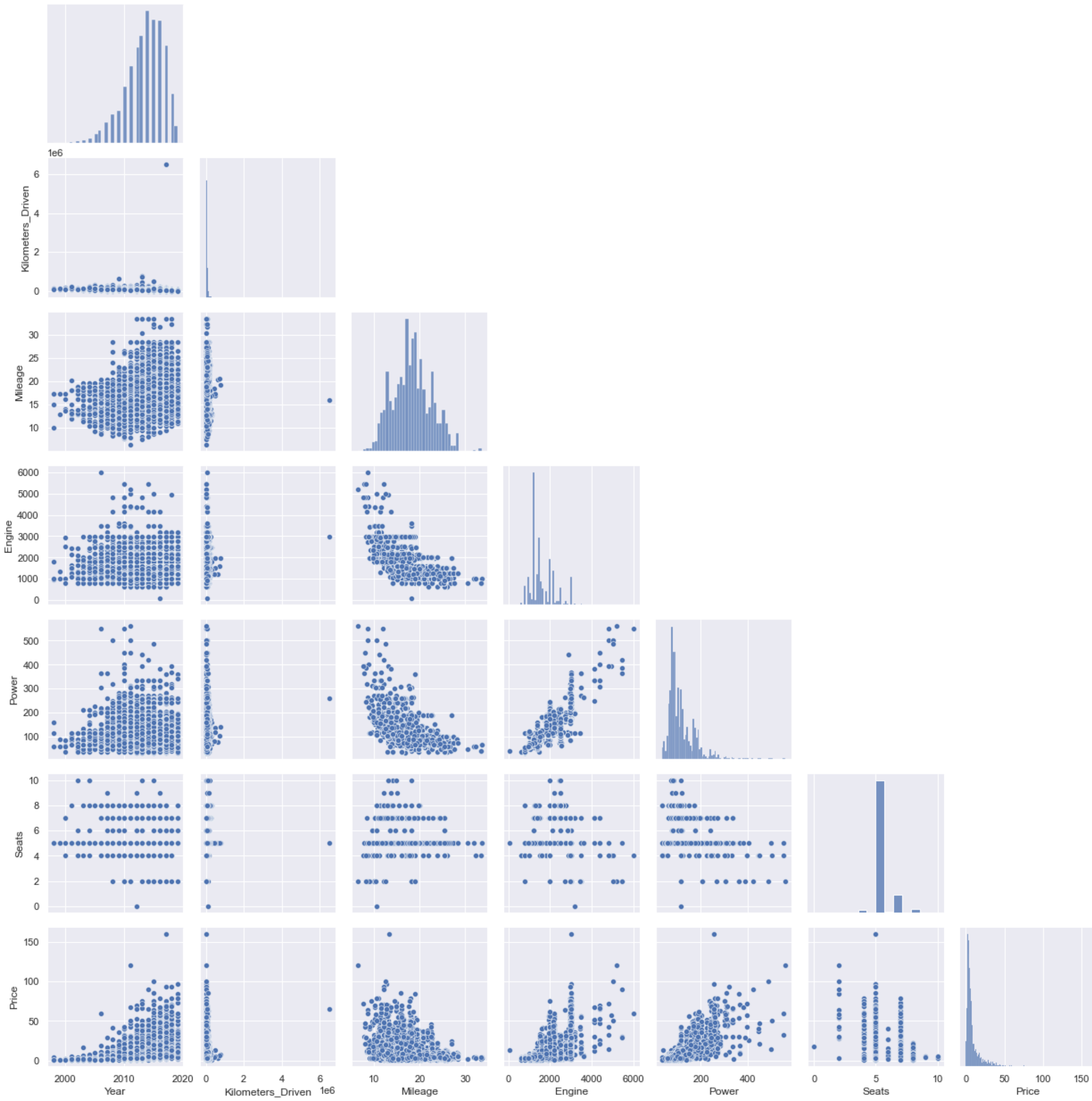
In [54]:
```
df.describe(include="all").T
```

Out[54]:

	count	unique	top	freq	mean	std	min	25%	50%	75%
Name	6019	1876	Mahindra XUV500 W8 2WD	49	NaN	NaN	NaN	NaN	NaN	NaI
Location	6019	11	Mumbai	790	NaN	NaN	NaN	NaN	NaN	NaI
Year	6019	NaN	NaN	NaN	2013.36	3.26974	1998	2011	2014	201
Kilometers_Driven	6019	NaN	NaN	NaN	58738.4	91268.8	171	34000	53000	7300
Fuel_Type	6019	5	Diesel	3205	NaN	NaN	NaN	NaN	NaN	NaI
Transmission	6019	2	Manual	4299	NaN	NaN	NaN	NaN	NaN	NaI
Owner_Type	6019	4	First	4929	NaN	NaN	NaN	NaN	NaN	NaI
Mileage	6019	NaN	NaN	NaN	18.34	4.15117	6.4	15.4	18.15	21.
Engine	6019	NaN	NaN	NaN	1621.28	599.554	72	1198	1493	196
Power	6019	NaN	NaN	NaN	113.253	53.231	34.2	78	98.6	138.0
Seats	6019	NaN	NaN	NaN	5.27679	0.806346	0	5	5	
Price	6019	NaN	NaN	NaN	9.47947	11.1879	0.44	3.5	5.64	9.9

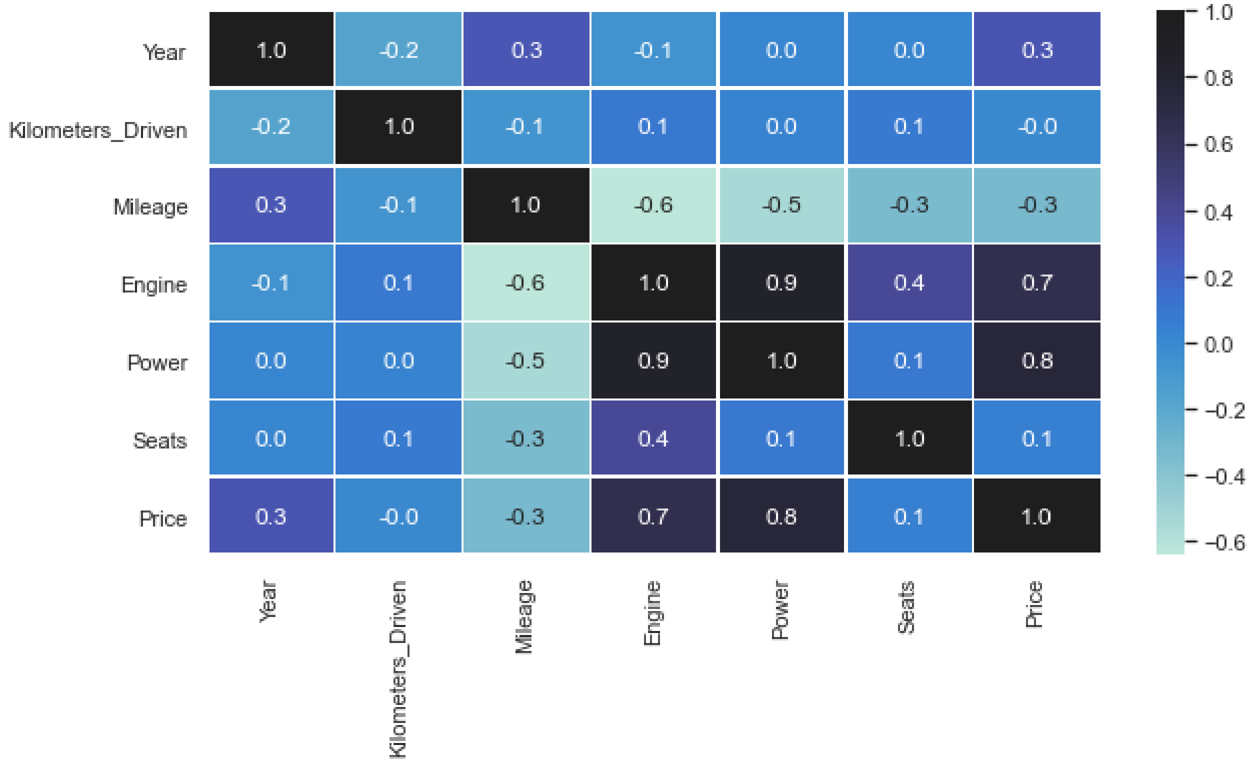
In [55]:
```
sns.pairplot(df, corner=True)
```

Out[55]: <seaborn.axisgrid.PairGrid at 0x28f2e509d30>



- Newer cars generally fetch higher prices along with 5 seat cars.
- After a certain threshold of kilometers driven, the price seems to have a limit on how much it can be sold for.
- As power increases, the price potential increases.
- Engine and power look highly correlated, so I will drop either one before building the model.

In [56]:
```
plt.figure(figsize=(10, 5))
sns.heatmap(df.corr(), annot=True, linewidths=0.5, fmt=".1f", center=1)
plt.show()
```



In [57]:

```
my_tab = pd.crosstab(index=df["Year"], columns="count")
my_tab
```

Out[57]:

col_0	count
Year	
1998	4
1999	2
2000	4
2001	8
2002	15
2003	17
2004	31
2005	57
2006	78
2007	125
2008	174
2009	198
2010	342
2011	466
2012	580
2013	649
2014	797

col_0	count
Year	
2015	744
2016	741
2017	587
2018	298
2019	102

In [58]:
```
my_tab = pd.crosstab(index=df["Fuel_Type"], columns="count")
my_tab
```

Out[58]:

col_0	count
Fuel_Type	
CNG	56
Diesel	3205
Electric	2
LPG	10
Petrol	2746

In [59]:
```
my_tab = pd.crosstab(index=df["Owner_Type"], columns="count")
my_tab
```

Out[59]:

col_0	count
Owner_Type	
First	4929
Fourth & Above	9
Second	968
Third	113

In [60]:
```
my_tab = pd.crosstab(index=df["Transmission"], columns="count")
my_tab
```

Out[60]:

col_0	count
Transmission	
Automatic	1720
Manual	4299

In [61]:
```
my_tab = pd.crosstab(index=df["Seats"], columns="count")
my_tab
```

Out[61]:

col_0	count
Seats	
0.0	1
2.0	16
4.0	99
5.0	5056
6.0	31
7.0	674
8.0	134
9.0	3
10.0	5

In [62]:

```
df["Name2"] = df["Name"].str.split(" ").str[0]
```

In [63]:

```
df["Name2"] = df["Name2"].astype("category")
```

In [64]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 6019 entries, 0 to 6018
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Name               6019 non-null   category
 1   Location           6019 non-null   category
 2   Year               6019 non-null   int64
 3   Kilometers_Driven  6019 non-null   int64
 4   Fuel_Type          6019 non-null   category
 5   Transmission       6019 non-null   category
 6   Owner_Type         6019 non-null   category
 7   Mileage            6019 non-null   float64
 8   Engine             6019 non-null   float64
 9   Power              6019 non-null   float64
 10  Seats              6019 non-null   float64
 11  Price              6019 non-null   float64
 12  Name2              6019 non-null   category
dtypes: category(6), float64(5), int64(2)
memory usage: 514.4 KB
```

In [65]:

```
df.head()
```

Out[65]:

	Name	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Mileag
0	Maruti Wagon R LXI CNG	Mumbai	2010	72000	CNG	Manual	First	26.6(
1	Hyundai Creta 1.6	Pune	2015	41000	Diesel	Manual	First	19.6

	Name	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Mileag
	CRDi SX Option							
2	Honda Jazz V	Chennai	2011	46000	Petrol	Manual	First	18.2(
3	Maruti Ertiga VDI	Chennai	2012	87000	Diesel	Manual	First	20.7
4	Audi A4 New 2.0 TDI Multitronic	Coimbatore	2013	40670	Diesel	Automatic	Second	15.2(

- Created a new column of just the car manufacturer, so I can visualize the 'Name' column better.

In [66]:
```
my_tab = pd.crosstab(index=df["Name2"], columns="count")
my_tab
```

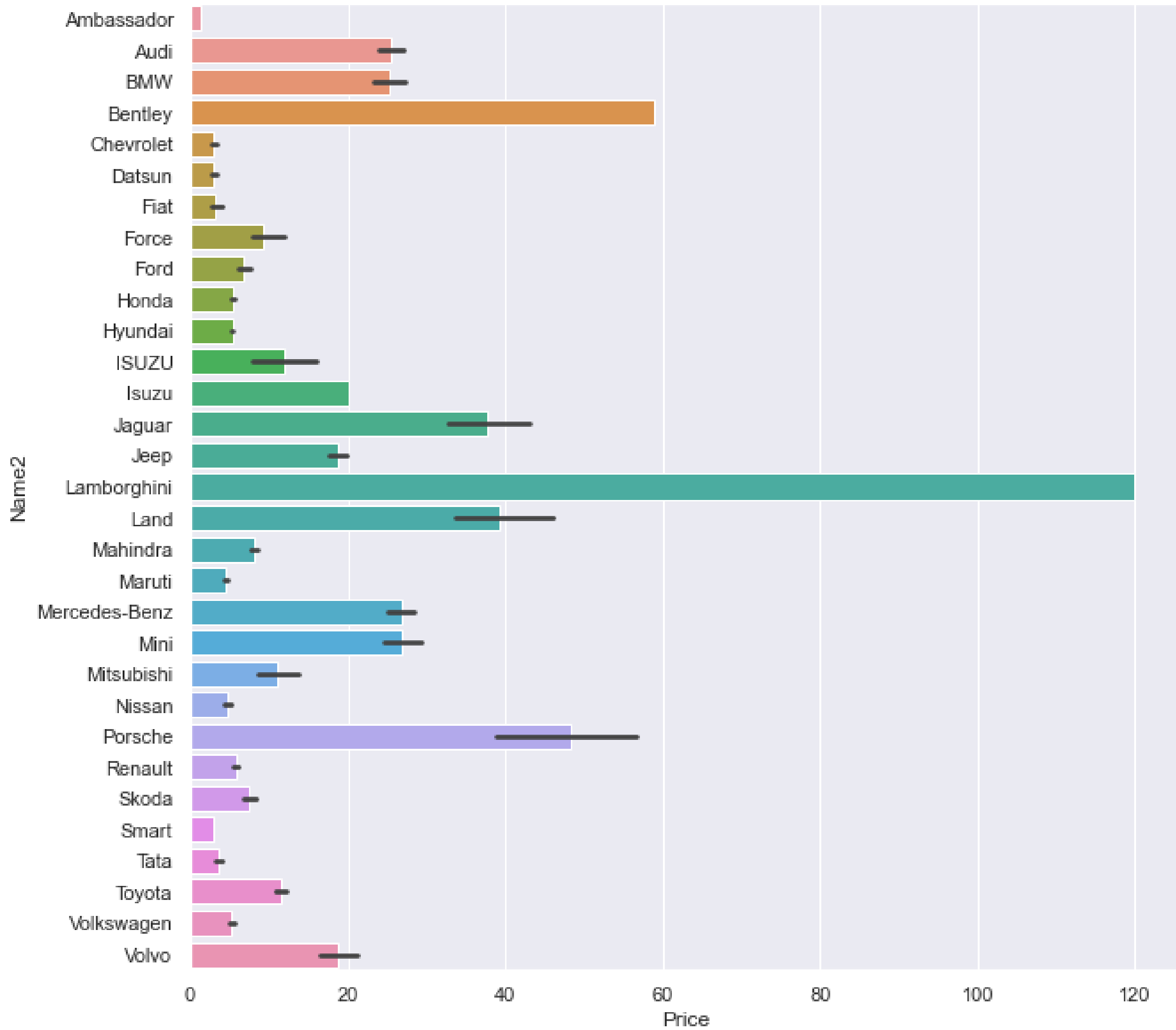
Out[66]:

col_0	count
Name2	
Ambassador	1
Audi	236
BMW	267
Bentley	1
Chevrolet	121
Datsun	13
Fiat	28
Force	3
Ford	300
Honda	608
Hyundai	1107
ISUZU	2
Isuzu	1
Jaguar	40
Jeep	15
Lamborghini	1
Land	60
Mahindra	272
Maruti	1211
Mercedes-Benz	318

col_0	count
Name2	
Mini	26
Mitsubishi	27
Nissan	91
Porsche	18
Renault	145
Skoda	173
Smart	1
Tata	186
Toyota	411
Volkswagen	315
Volvo	21

In [67]:
```
plt.figure(figsize=(10, 10))
sns.barplot(df["Price"], df["Name2"])
```

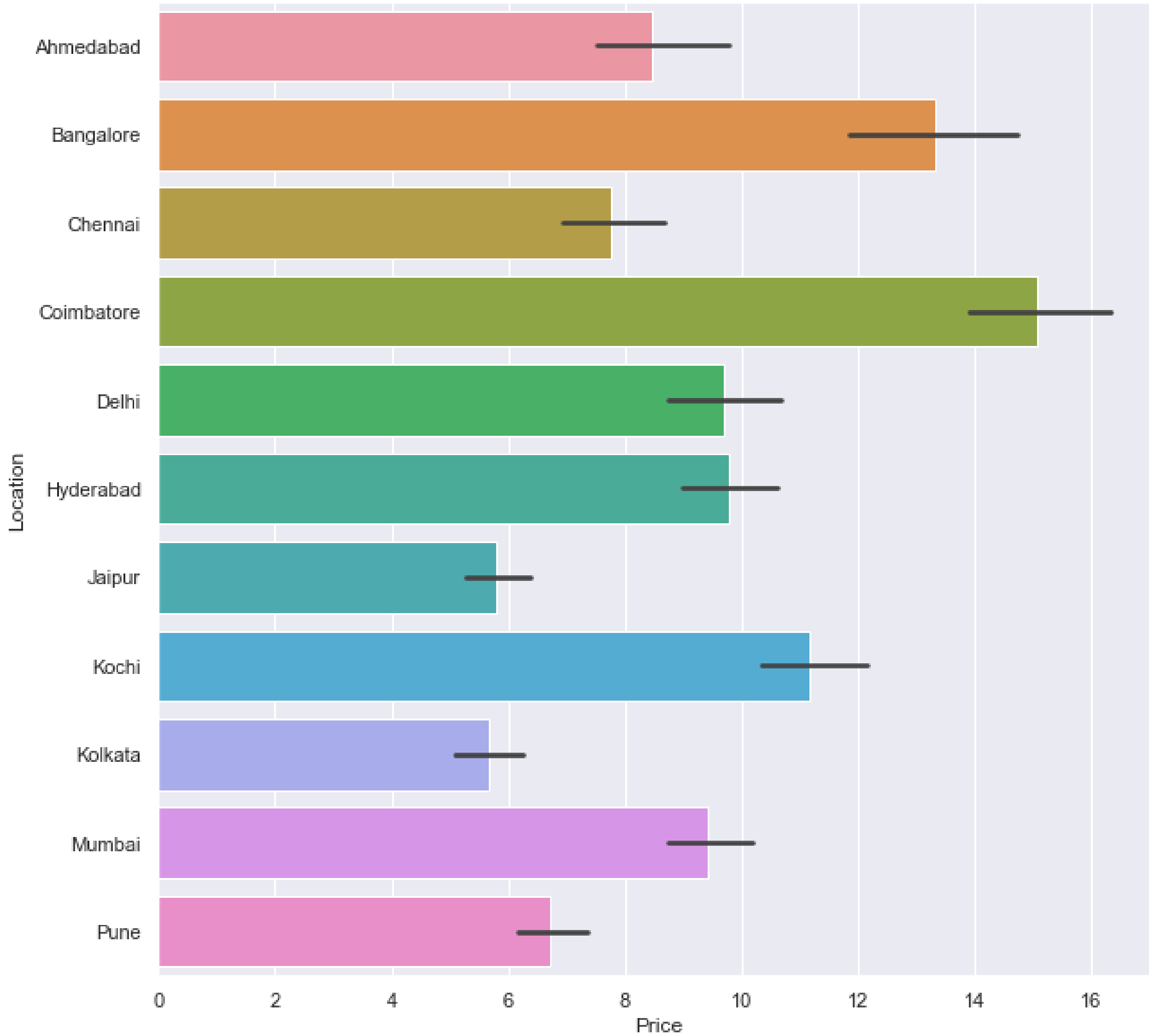
Out[67]: <AxesSubplot:xlabel='Price', ylabel='Name2'>



- Bentley, Jaguar, Lamborghini, Land Rover, and Porsche all sell at high prices.

In [68]:
```
plt.figure(figsize=(10, 10))
sns.barplot(df["Price"], df["Location"])
```

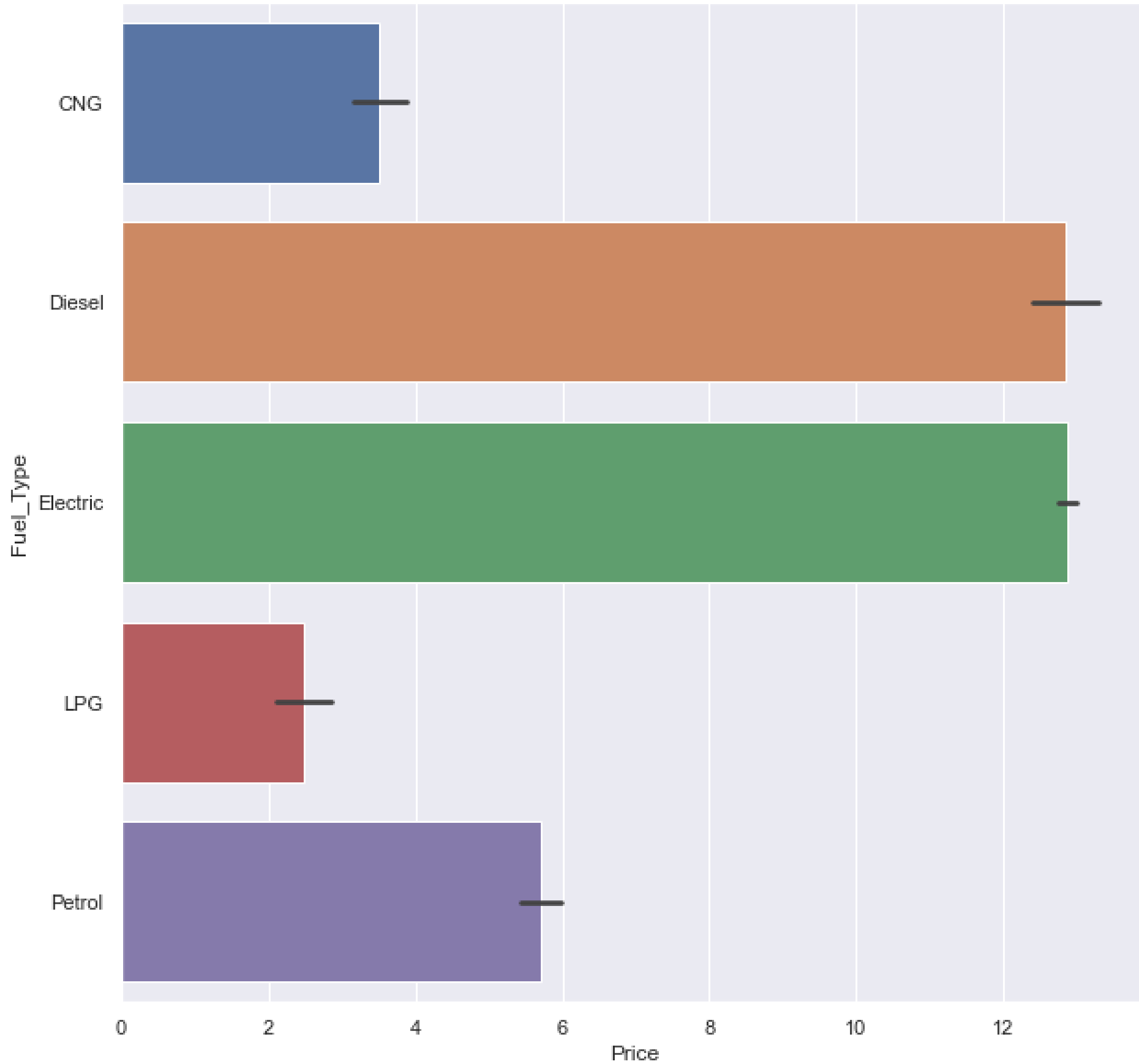
Out[68]: <AxesSubplot:xlabel='Price', ylabel='Location'>



- Cars from Coimbatore and Bangalore generally sell for more.

In [69]:
```
plt.figure(figsize=(10, 10))
sns.barplot(df["Price"], df["Fuel_Type"])
```

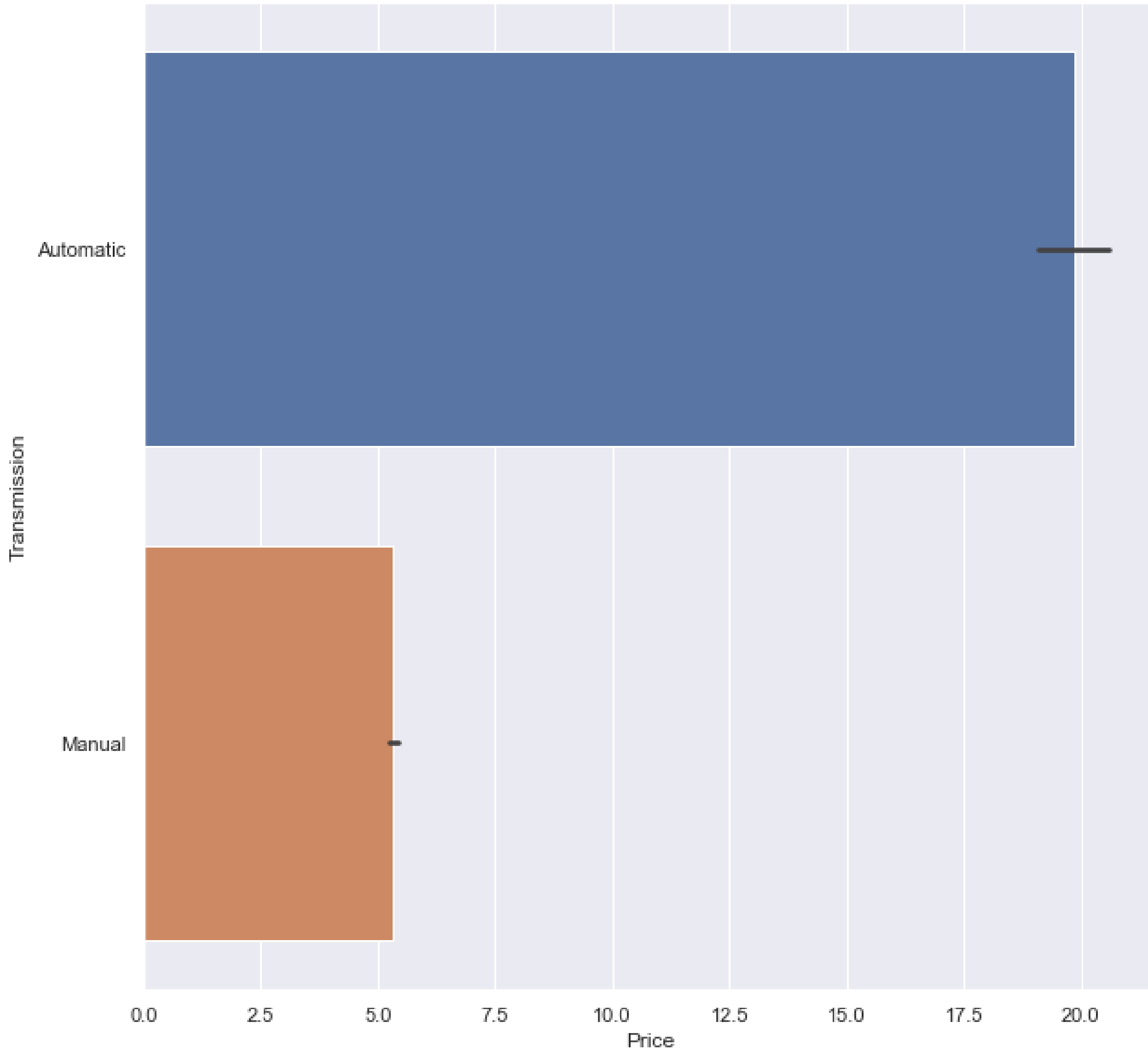
Out[69]: <AxesSubplot:xlabel='Price', ylabel='Fuel_Type'>



- Electric and Diesel fetch higher prices.

In [70]:
```
plt.figure(figsize=(10, 10))
sns.barplot(df["Price"], df["Transmission"])
```

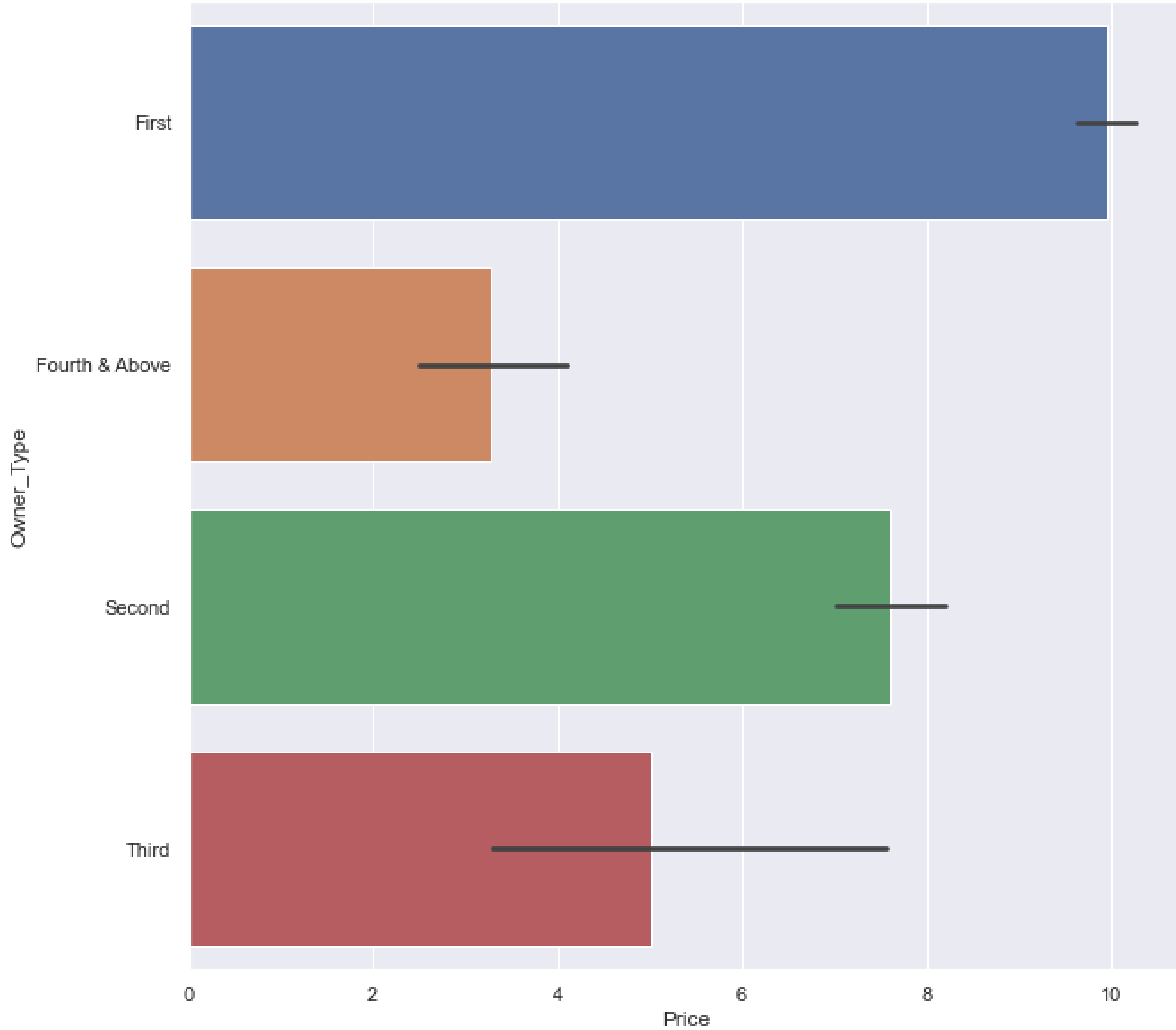
Out[70]: <AxesSubplot:xlabel='Price', ylabel='Transmission'>



- Automatic cars sell at a higher price than Manual cars.

In [71]:
```
plt.figure(figsize=(10, 10))
sns.barplot(df["Price"], df["Owner_Type"])
```

Out[71]: <AxesSubplot:xlabel='Price', ylabel='Owner_Type'>

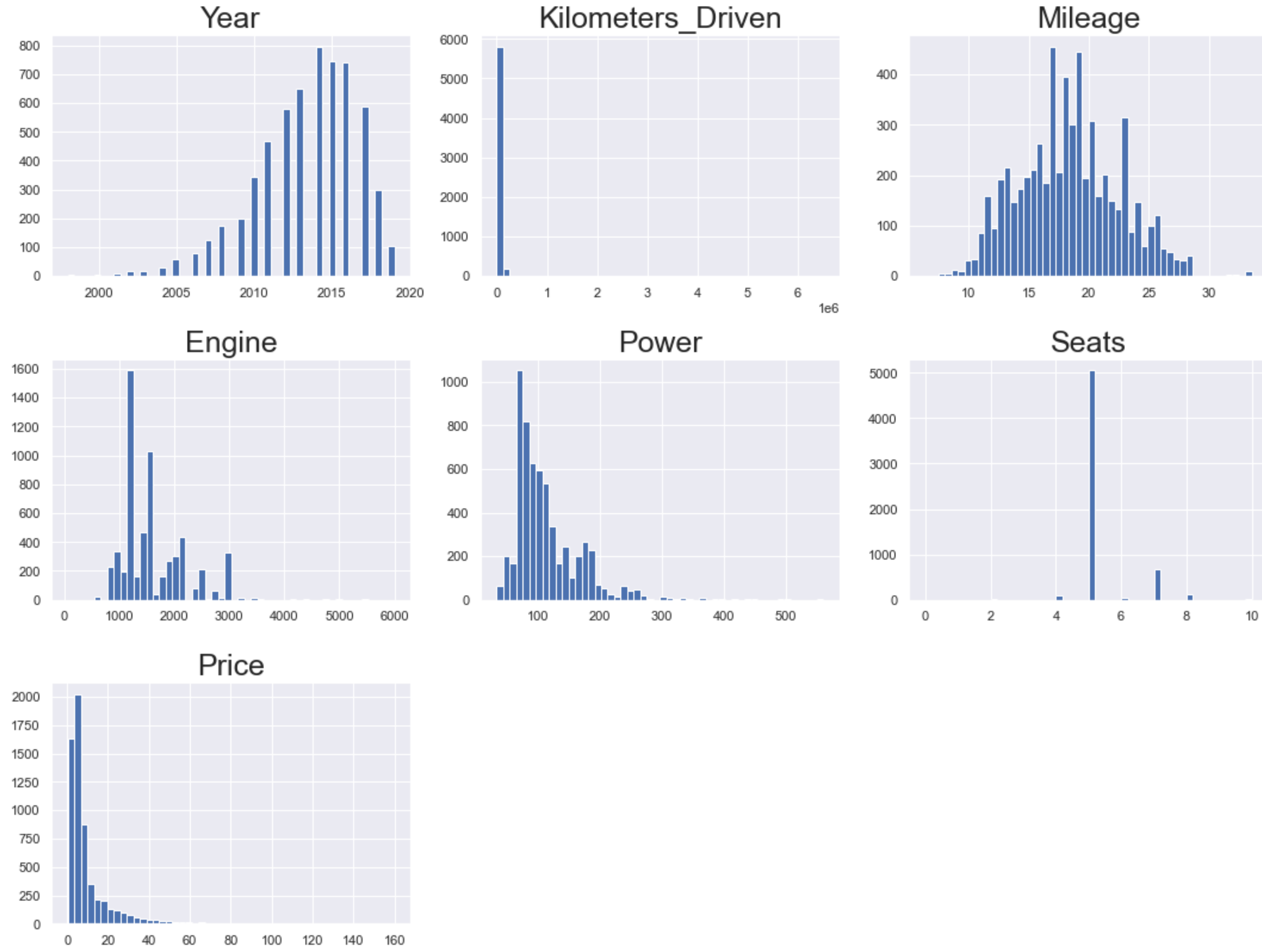


- First owner cars sell for more and as the number of owners go up, the price goes down.

Variable Transformations

Checking skewness of the numeric columns.

In [72]:
```
num_cols = [item for item in df.select_dtypes(include=np.number).columns]

plt.figure(figsize=(15, 45))

for i in range(len(num_cols)):
    plt.subplot(12, 3, i + 1)
    plt.hist(df[num_cols[i]], bins=50)
    plt.tight_layout()
    plt.title(num_cols[i], fontsize=25)

plt.show()
```



Creating copy of dataframe and removing 'Mileage', 'Seats', and 'Price' from the num_cols.

In [73]:
```
df2 = df.copy()

num_cols.remove("Mileage")
num_cols.remove("Seats")
num_cols.remove("Price")
```

Using log transforms on some columns

In [74]:
```
for col in num_cols:
    df2[col + "_log"] = np.log(df2[col] + 1)

df2.drop(num_cols, axis=1, inplace=True)
df2.head()
```

Out[74]:

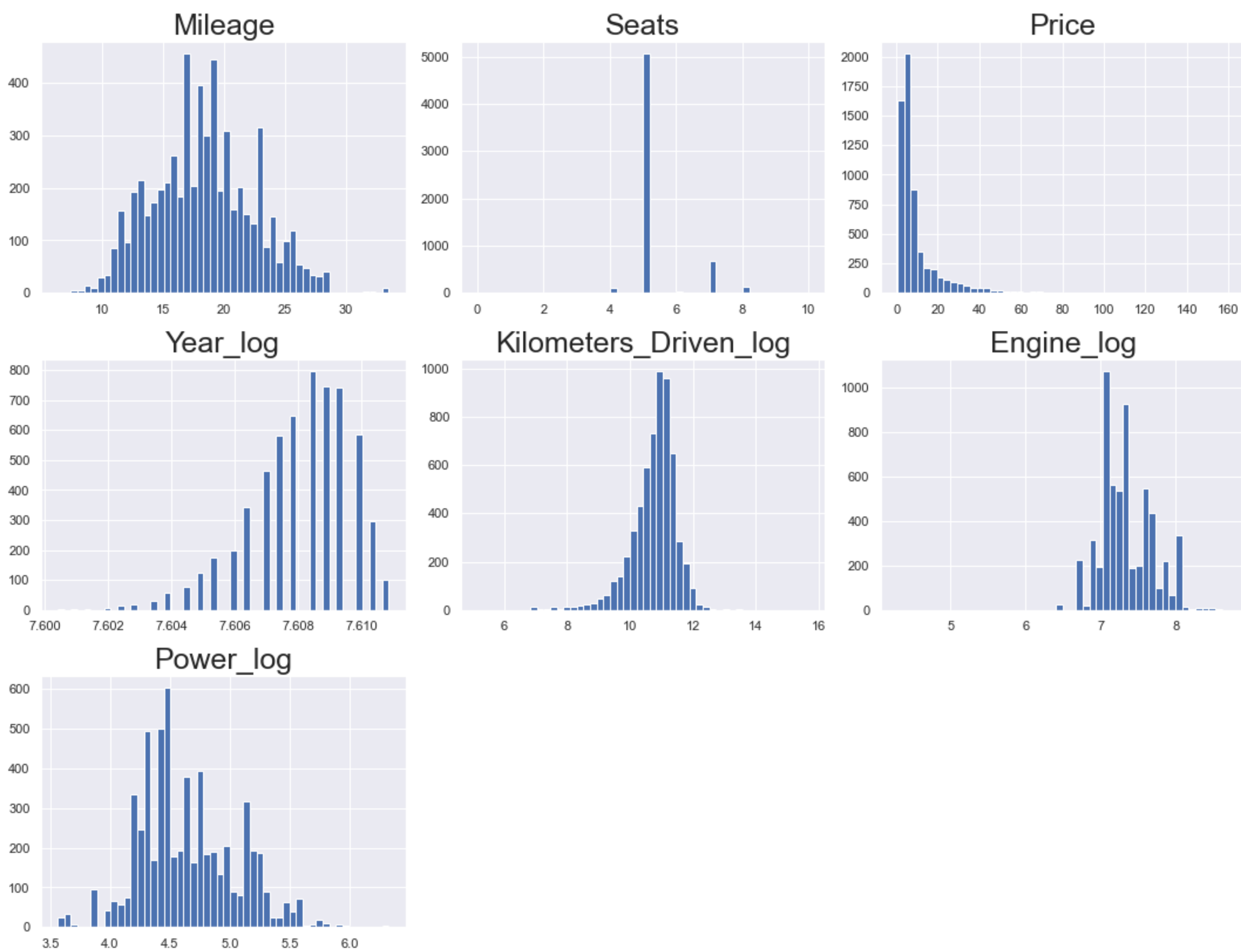
	Name	Location	Fuel_Type	Transmission	Owner_Type	Mileage	Seats	Price	Name2
0	Maruti Wagon R LXI CNG	Mumbai	CNG	Manual	First	26.60	5.0	1.75	Maruti
1	Hyundai Creta 1.6	Pune	Diesel	Manual	First	19.67	5.0	12.50	Hyundai

	Name	Location	Fuel_Type	Transmission	Owner_Type	Mileage	Seats	Price	Name2
	CRDi SX Option								
2	Honda Jazz V	Chennai	Petrol	Manual	First	18.20	5.0	4.50	Honda
3	Maruti Ertiga VDI	Chennai	Diesel	Manual	First	20.77	7.0	6.00	Maruti
4	Audi A4 New 2.0 TDI Multitronic	Coimbatore	Diesel	Automatic	Second	15.20	5.0	17.74	Audi

Checking skewness after applying the log transformation

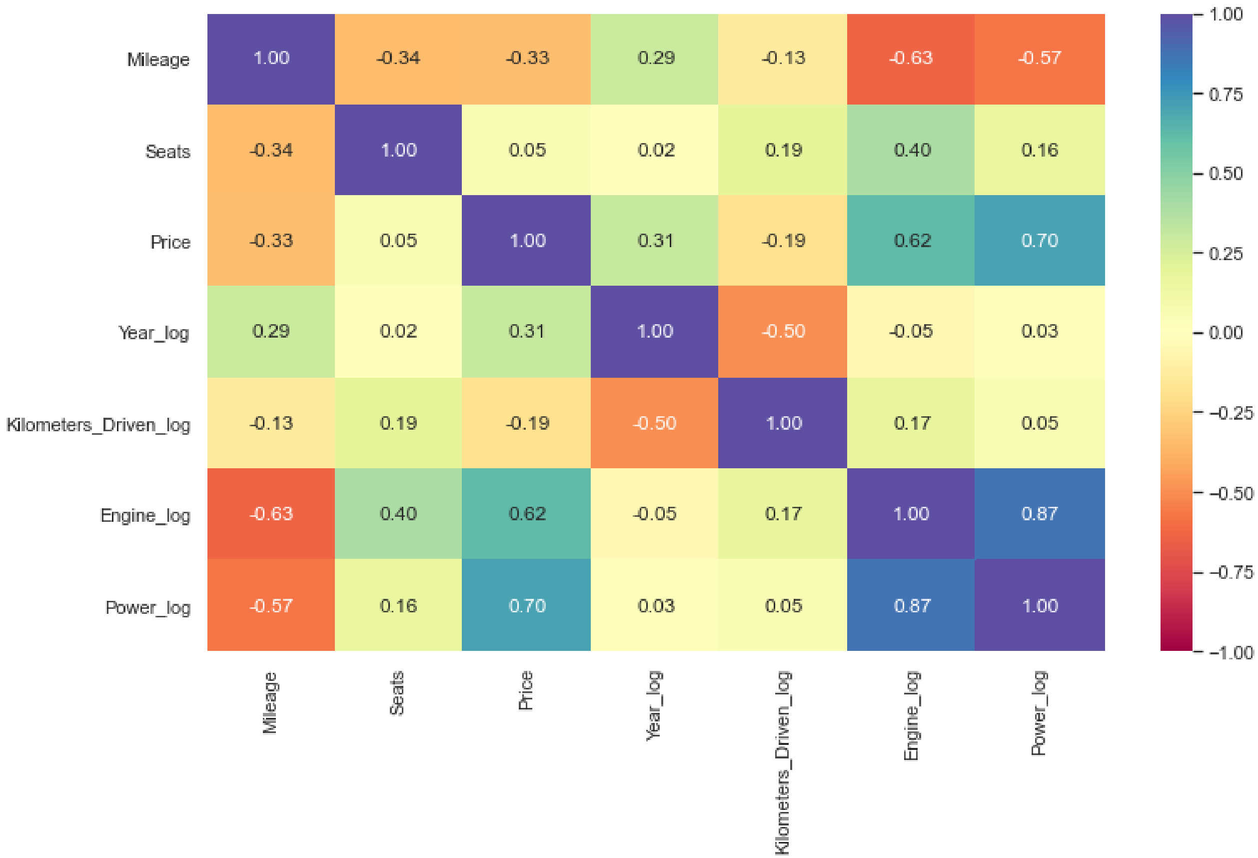
In [75]:

```
num_cols = [item for item in df2.select_dtypes(include=np.number).columns]

plt.figure(figsize=(15, 45))

for i in range(len(num_cols)):
    plt.subplot(12, 3, i + 1)
    plt.hist(df2[num_cols[i]], bins=50)
    plt.tight_layout()
    plt.title(num_cols[i], fontsize=25)

plt.show()
```



Checking for correlations between columns.

In [76]:
```
plt.figure(figsize=(12, 7))
sns.heatmap(
    df2[num_cols].corr(), annot=True, vmin=-1, vmax=1, fmt=".2f", cmap="Spectral
)
plt.show()
```



In [77]:
```
df2.drop(["Name"], axis=1, inplace=True)
df2.shape
```

Out[77]: (6019, 12)

- Dropped the 'Name' column because it will make the model overfit.

In [78]:
```
df2.drop(["Engine_log"], axis=1, inplace=True)
df2.shape
```

Out[78]: (6019, 11)

- Dropped 'Engine_log' as it has a fairly high correlation with 'Power_log'.

Model Building

Defining dependent variable

In [79]:
```
ind_vars = df2.drop(["Price"], axis=1)
dep_var = df2[["Price"]]
```

Creating dummy variables

In [80]:
```
def encode_cat_vars(x):
    x = pd.get_dummies(
        x,
        columns=x.select_dtypes(include=["object", "category"]).columns.tolist()
        drop_first=True,
    )
    return x


ind_vars_num = encode_cat_vars(ind_vars)
ind_vars_num.head()
```

Out[80]:

	Mileage	Seats	Year_log	Kilometers_Driven_log	Power_log	Location_Bangalore	Location_Che
0	26.60	5.0	7.606387	11.184435	4.080246	0	
1	19.67	5.0	7.608871	10.621352	4.845761	0	
2	18.20	5.0	7.606885	10.736418	4.496471	0	
3	20.77	7.0	7.607381	11.373675	4.497139	0	
4	15.20	5.0	7.607878	10.613271	4.954418	0	

In [81]:
```
ind_vars_num.shape
```

Out[81]:
```
(6019, 53)
```

Splitting data into train and test.

In [82]:
```
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(
    ind_vars_num, dep_var, test_size=0.3, random_state=1
)
```

In [83]:
```
print("Number of rows in train data =", x_train.shape[0])
print("Number of rows in test data =", x_test.shape[0])
```

```
Number of rows in train data = 4213
Number of rows in test data = 1806
```

Fitting a linear model.

In [84]:
```
lin_reg_model = LinearRegression()
lin_reg_model.fit(x_train, y_train)
```

Out[84]: LinearRegression()

Checking the coefficients and intercept of the model.

In [85]:
```
coef_df = pd.DataFrame(
    np.append(lin_reg_model.coef_.flatten(), lin_reg_model.intercept_),
    index=x_train.columns.tolist() + ["Intercept"],
    columns=["Coefficients"],
)
coef_df
```

Out[85]:

	Coefficients
Mileage	-2.514926e-01
Seats	1.583721e-01
Year_log	1.679166e+03
Kilometers_Driven_log	-1.851085e+00
Power_log	9.462154e+00
Location_Bangalore	2.475255e+00
Location_Chennai	1.081489e+00
Location_Coimbatore	2.427899e+00
Location_Delhi	-3.868570e-01
Location_Hyderabad	2.053745e+00
Location_Jaipur	1.291546e+00
Location_Kochi	-4.348436e-02
Location_Kolkata	-1.519200e+00
Location_Mumbai	-9.165768e-01
Location_Pune	5.505163e-01
Fuel_Type_Diesel	-1.000122e+00
Fuel_Type_Electric	8.166676e+00
Fuel_Type_LPG	-3.147285e-01
Fuel_Type_Petrol	-2.754225e+00
Transmission_Manual	2.347101e-02
Owner_Type_Fourth & Above	4.398795e-01
Owner_Type_Second	-5.829036e-01
Owner_Type_Third	2.352024e-01
Name2_Audi	2.988422e-01
Name2_BMW	1.698353e-01

	Coefficients
Name2_Bentley	2.861665e+01
Name2_Chevrolet	-1.129657e+01
Name2_Datsun	-1.269417e+01
Name2_Fiat	-1.225300e+01
Name2_Force	-1.204334e+01
Name2_Ford	-1.015721e+01
Name2_Honda	-1.165484e+01
Name2_Hyundai	-1.102101e+01
Name2_ISUZU	-1.964455e+01
Name2_Isuzu	9.094947e-13
Name2_Jaguar	9.140631e+00
Name2_Jeep	-9.951130e+00
Name2_Lamborghini	8.357072e+01
Name2_Land	1.317954e+01
Name2_Mahindra	-1.294632e+01
Name2_Maruti	-9.430385e+00
Name2_Mercedes-Benz	1.398047e+00
Name2_Mini	4.132698e+00
Name2_Mitsubishi	-8.891577e+00
Name2_Nissan	-1.079433e+01
Name2_Porsche	1.527625e+01
Name2_Renault	-1.144091e+01
Name2_Skoda	-1.091606e+01
Name2_Smart	-9.670696e+00
Name2_Tata	-1.125272e+01
Name2_Toyota	-8.652302e+00
Name2_Volkswagen	-1.160832e+01
Name2_Volvo	-5.980995e+00
Intercept	-1.277624e+04

Coefficient interpretations

- Strong positive relation: Power, Electric, Isuzu, Jaguar, Lamborghini.
- Strong negative relation: Mumbai, Jeep, Maruti, Mitsubishi, Smart, Toyota.

Model performance check

In [86]:
```
def adj_r2(ind_vars, targets, predictions):
    r2 = r2_score(targets, predictions)
    n = ind_vars.shape[0]
    k = ind_vars.shape[1]
    return 1 - ((1 - r2) * (n - 1) / (n - k - 1))


def model_perf(model, inp, out):

    y_pred = model.predict(inp)
    y_act = out.values

    return pd.DataFrame(
        {
            "RMSE": np.sqrt(mean_squared_error(y_act, y_pred)),
            "MAE": mean_absolute_error(y_act, y_pred),
            "R^2": r2_score(y_act, y_pred),
            "Adjusted R^2": adj_r2(inp, y_act, y_pred),
        },
        index=[0],
    )
```

Checking model performance on train and test sets.

In [87]:
```
print("Training Performance\n")
print(model_perf(lin_reg_model, x_train, y_train))

print("\n\nTest Performance\n")
print(model_perf(lin_reg_model, x_test, y_test))
```

```
Training Performance

       RMSE       MAE       R^2  Adjusted R^2
0  5.796652  3.153041  0.737218      0.733869


Test Performance

       RMSE       MAE       R^2  Adjusted R^2
0  4.908013  2.967988  0.797156       0.79102
```

- The Test R^2 is above 75 which is nice.
- The Adjusted R^2 has improved from 0.73 to 0.79.
- The Test RMSE has lowered from the Training RMSE, so the model isn't overfitting.

Actionable Insights & Recommendations

If the goal is to sell the most used cars, I would focus on the car brands that are being sold more frequently. There is more risk when it comes to the less frequent brands, as the market for them looks smaller.

- Stick to brands like Maruti, Hyundai, Honda, and Toyota.
- Cars like Audi, BMW, and Porsche have a wider range of price possibilities, so if they are being sold, there are more variables to consider when putting a price tag on.

Features or properties to look out for.

- Cars made between 2012 and 2017.
- Diesel and petrol cars.
- First and Second owned cars.
- Low kilometers put on the car. There is a higher chance of selling it as well as getting more.
- Cars with 5 seats sell more frequently.
- Transmission: automatics generally sell at a higher price, but are less frequent than manual cars.