In [1]:
```
# Import Libraries
import pandas as pd
import numpy as np
%matplotlib inline
import matplotlib.pyplot as plt
import os
```

In [2]:
```
# Create a dir. to store project's figures
PROJECT_ROOT_DIR = "."
PROJECT_NAME = "CarDekho"
IMAGES_PATH = os.path.join(PROJECT_ROOT_DIR, "figures", PROJECT_NAME)
os.makedirs(IMAGES_PATH, exist_ok=True)

# Creating a function to save generated figures in the above dir.
def save_fig(fig_id, tight_layout = True, fig_extension = "png", resolution = 300
    path = os.path.join(IMAGES_PATH, fig_id + "." + fig_extension)
    if tight_layout:
        plt.tight_layout()
    plt.savefig(path, format = fig_extension, dpi = resolution)
    print("Figure saved")
```

In [3]:
```
# Read the CSV file
df = pd.read_csv("D:\\DataSc\\Datasets\\carDekho_car.csv")
df.head()
```

Out[3]:

	name	year	selling_price	km_driven	fuel	seller_type	transmission	owner	mileage	eng
0	Maruti Swift Dzire VDI	2014	450000	145500	Diesel	Individual	Manual	First Owner	23.4 kmpl	1
1	Skoda Rapid 1.5 TDI Ambition	2014	370000	120000	Diesel	Individual	Manual	Second Owner	21.14 kmpl	1
2	Honda City 2017-2020 EXi	2006	158000	140000	Petrol	Individual	Manual	Third Owner	17.7 kmpl	1
3	Hyundai i20 Sportz Diesel	2010	225000	127000	Diesel	Individual	Manual	First Owner	23.0 kmpl	1
4	Maruti Swift VXI BSIII	2007	130000	120000	Petrol	Individual	Manual	First Owner	16.1 kmpl	1

In [4]:
```
# Used Car prediction

# Dataset: Kaggle data set on used car prices by CarDekho.com (Dataset is avaiabl

# Before exploring further let's try to understand the data first and what we war
# Features present:
#          name: Name of the model along with manufacturing company's name
#          year: The year the model was released!
#          selling price: The current selling price of the car
#          km_driven: Total distance driven in the car
#          fuel: Type of car(petrol/diesel/etc.)
#          seller_type: The car is sold by a person or by a dealer
#          transmission: automatic or manual transmission
#          owner: no. of person that has owned the car
#          mileage: a number of miles travelled or covered in kmpl
#          engine: the capacity of the engine or piston chambers in cubic centime
#          max_power: Horsepower of the car!
#          torque:
#          seats: no. of seats in the car model
```

In [5]:
```
# Now that we have defined our dataset, let's make a plan what to do with it!
# Our goal would be observe and explore the price of car models based on the give
# Understand which factors affect most on the price of a car.
# And finally try to build a linear model to predict car prices!
```

In [6]:
```
# Let's get some info on the dataset
print(df.info())
print("\n Dataset size:   ", df.shape)

df.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8128 entries, 0 to 8127
Data columns (total 13 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   name           8128 non-null   object 
 1   year           8128 non-null   int64  
 2   selling_price  8128 non-null   int64  
 3   km_driven      8128 non-null   int64  
 4   fuel           8128 non-null   object 
 5   seller_type    8128 non-null   object 
 6   transmission   8128 non-null   object 
 7   owner          8128 non-null   object 
 8   mileage        7907 non-null   object 
 9   engine         7907 non-null   object 
 10  max_power      7913 non-null   object 
 11  torque         7906 non-null   object 
 12  seats          7907 non-null   float64
dtypes: float64(1), int64(3), object(9)
memory usage: 825.6+ KB
None

 Dataset size:    (8128, 13)
```

Out[6]:

	name	year	selling_price	km_driven	fuel	seller_type	transmission	owner	mileage	e
0	Maruti Swift Dzire VDI	2014	450000	145500	Diesel	Individual	Manual	First Owner	23.4 kmpl	
1	Skoda Rapid 1.5 TDI Ambition	2014	370000	120000	Diesel	Individual	Manual	Second Owner	21.14 kmpl	
2	Honda City 2017-2020 EXi	2006	158000	140000	Petrol	Individual	Manual	Third Owner	17.7 kmpl	
3	Hyundai i20 Sportz Diesel	2010	225000	127000	Diesel	Individual	Manual	First Owner	23.0 kmpl	
4	Maruti Swift VXI BSIII	2007	130000	120000	Petrol	Individual	Manual	First Owner	16.1 kmpl	

In [7]:
```
# Our dataset is not perfect! We have to manipulate the dataset to make it fit!
# Notice that the fuel, seller_type, transmission, and owner columns are categori
# While mileage, engine, max_power, and max_power are defined as object type!
# We will have to convert the above objects into float or int type by removing th
```

In [8]:
```
# Checking Categorical data
for col in df[['fuel', 'seller_type', 'transmission', 'owner']]:
    print(df[[col]].value_counts(), "\n")

# We will use Scikit learn lib. to convert these categorical values into numerica
```

```
fuel
Diesel    4402
Petrol    3631
CNG         57
LPG         38
dtype: int64

seller_type
Individual          6766
Dealer              1126
Trustmark Dealer     236
dtype: int64

transmission
Manual          7078
Automatic       1050
dtype: int64

owner
First Owner             5289
Second Owner            2105
Third Owner              555
Fourth & Above Owner     174
Test Drive Car             5
dtype: int64
```

In [9]:
```
# Let's clean our dataset first! A good Dataset can help us make better models ar
# Check of duplicates in the dataset!
duplicateRows = df[df.duplicated()]
duplicateRows
```

Out[9]:

	name	year	selling_price	km_driven	fuel	seller_type	transmission	owner	mileage
291	Hyundai Grand i10 Sportz	2017	450000	35000	Petrol	Individual	Manual	First Owner	18.9 kmpl
296	Maruti Swift VXI	2012	330000	50000	Petrol	Individual	Manual	Second Owner	18.6 kmpl
370	Jaguar XE 2016-2019 2.0L Diesel Prestige	2017	2625000	9000	Diesel	Dealer	Automatic	First Owner	13.6 kmpl
371	Lexus ES 300h	2019	5150000	20000	Petrol	Dealer	Automatic	First Owner	22.37 kmpl
372	Jaguar XF 2.0 Diesel Portfolio	2017	3200000	45000	Diesel	Dealer	Automatic	First Owner	19.33 kmpl
...	...	...	...	...	...	...	...	...	...
7987	Renault Captur 1.5 Diesel RXT	2018	1265000	12000	Diesel	Individual	Manual	First Owner	20.37 kmpl
7988	Maruti Ciaz Alpha Diesel	2019	1025000	32000	Diesel	Individual	Manual	First Owner	28.09 kmpl
8117	Maruti Swift Dzire VDI	2015	625000	50000	Diesel	Individual	Manual	First Owner	26.59 kmpl
8126	Tata Indigo CR4	2013	290000	25000	Diesel	Individual	Manual	First Owner	23.57 kmpl
8127	Tata Indigo CR4	2013	290000	25000	Diesel	Individual	Manual	First Owner	23.57 kmpl

1202 rows × 13 columns

In [10]:
```
# Dropping the duplicates, but keeping the last instance of the duplicated data!
df_new = df.drop_duplicates(keep='last')
print(df_new.shape[0])
```

```
6926
```

In [11]:
```
df_new.tail()
```

Out[11]:

	name	year	selling_price	km_driven	fuel	seller_type	transmission	owner	mileage
8122	Hyundai i20 Magna 1.4 CRDi	2014	475000	80000	Diesel	Individual	Manual	Second Owner	22.54 kmpl
8123	Hyundai i20 Magna	2013	320000	110000	Petrol	Individual	Manual	First Owner	18.5 kmpl
8124	Hyundai Verna CRDi SX	2007	135000	119000	Diesel	Individual	Manual	Fourth & Above Owner	16.8 kmpl
8125	Maruti Swift Dzire ZDi	2009	382000	120000	Diesel	Individual	Manual	First Owner	19.3 kmpl
8127	Tata Indigo CR4	2013	290000	25000	Diesel	Individual	Manual	First Owner	23.57 kmpl

In [12]:
```
# Now, let's look for null values in our data sets!
print(df_new.isna().sum())
print("\n")
print(df_new.isna().sum()*100/df_new.shape[0])
```

```
name              0
year              0
selling_price     0
km_driven         0
fuel              0
seller_type       0
transmission      0
owner             0
mileage         208
engine          208
max_power       205
torque          209
seats           208
dtype: int64


name            0.000000
year            0.000000
selling_price   0.000000
km_driven       0.000000
fuel            0.000000
seller_type     0.000000
transmission    0.000000
owner           0.000000
mileage         3.003176
engine          3.003176
max_power       2.959861
torque          3.017615
seats           3.003176
dtype: float64
```

In [13]:
```
# About ~2-3% of the data in mileage, engine, max_power, torque, and seats are mi
# Let's drop these NaNs and check our dataset(I'm preffering to drop the value as
df_new_drop = df_new.dropna()

# disable chained assignments so we dont get attribute warnings!
pd.options.mode.chained_assignment = None

print(df_new_drop.isna().sum()*100/df_new_drop.shape[0])
```

```
name             0.0
year             0.0
selling_price    0.0
km_driven        0.0
fuel             0.0
seller_type      0.0
transmission     0.0
owner            0.0
mileage          0.0
engine           0.0
max_power        0.0
torque           0.0
seats            0.0
dtype: float64
```

In [14]:
```
# Nulls have been taken care of! But before we go further, let's compare the data
print('Original dataset size:                    ', df.shape)
print('Dataset size after dropping duplicates:   ', df_new.shape)
print('Dataset size after dropping NaNs:         ', df_new_drop.shape)
print('Loss is data(in %) from original:         ', ((df.shape[0]-df_new_drop.sha
```

```
Original dataset size:                    (8128, 13)
Dataset size after dropping duplicates:   (6926, 13)
Dataset size after dropping NaNs:         (6717, 13)
Loss is data(in %) from original:         17.35974409448819
```

In [15]:
```
# 17.4% of the data has been removed or dropped from the original dataset!
# We still have a lot of data to work with! But as general rule of thumb: MORE DA
# let's look at our new dataset first!
df_new_drop.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 6717 entries, 0 to 8127
Data columns (total 13 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   name           6717 non-null   object 
 1   year           6717 non-null   int64  
 2   selling_price  6717 non-null   int64  
 3   km_driven      6717 non-null   int64  
 4   fuel           6717 non-null   object 
 5   seller_type    6717 non-null   object 
 6   transmission   6717 non-null   object 
 7   owner          6717 non-null   object 
 8   mileage        6717 non-null   object 
 9   engine         6717 non-null   object 
 10  max_power      6717 non-null   object 
 11  torque         6717 non-null   object 
 12  seats          6717 non-null   float64
dtypes: float64(1), int64(3), object(9)
memory usage: 734.7+ KB
```

In [16]:
```
df_new_drop.head()
```

Out[16]:

	name	year	selling_price	km_driven	fuel	seller_type	transmission	owner	mileage	eng
0	Maruti Swift Dzire VDI	2014	450000	145500	Diesel	Individual	Manual	First Owner	23.4 kmpl	1
1	Skoda Rapid 1.5 TDI Ambition	2014	370000	120000	Diesel	Individual	Manual	Second Owner	21.14 kmpl	1
2	Honda City 2017-2020 EXi	2006	158000	140000	Petrol	Individual	Manual	Third Owner	17.7 kmpl	1
3	Hyundai i20 Sportz Diesel	2010	225000	127000	Diesel	Individual	Manual	First Owner	23.0 kmpl	1
4	Maruti Swift VXI BSIII	2007	130000	120000	Petrol	Individual	Manual	First Owner	16.1 kmpl	1

In [17]:
```
df_new_drop.tail()
```

Out[17]:

	name	year	selling_price	km_driven	fuel	seller_type	transmission	owner	mileage
8122	Hyundai i20 Magna 1.4 CRDi	2014	475000	80000	Diesel	Individual	Manual	Second Owner	22.54 kmpl
8123	Hyundai i20 Magna	2013	320000	110000	Petrol	Individual	Manual	First Owner	18.5 kmpl
8124	Hyundai Verna CRDi SX	2007	135000	119000	Diesel	Individual	Manual	Fourth & Above Owner	16.8 kmpl
8125	Maruti Swift Dzire ZDi	2009	382000	120000	Diesel	Individual	Manual	First Owner	19.3 kmpl
8127	Tata Indigo CR4	2013	290000	25000	Diesel	Individual	Manual	First Owner	23.57 kmpl

In [18]:
```
# Ok, so we need to reset of index! We also need to modify the mileage, engine, c
# Let's change our column data first!

# Dropping the SI-units in the 'mileage', 'engine', 'max_power'!
df_new_drop['mileage'] = df_new_drop['mileage'].replace({'kmpl': '*1', 'km/kg':'*
df_new_drop['engine'] = df_new_drop['engine'].replace({'CC': '*1'}, regex = True)
df_new_drop['max_power'] = df_new_drop['max_power'].replace({'bhp': '*1'}, regex

df_new_drop.head(3)
```

Out[18]:

	name	year	selling_price	km_driven	fuel	seller_type	transmission	owner	mileage	eng
0	Maruti Swift Dzire VDI	2014	450000	145500	Diesel	Individual	Manual	First Owner	23.40	1
1	Skoda Rapid 1.5 TDI Ambition	2014	370000	120000	Diesel	Individual	Manual	Second Owner	21.14	1
2	Honda City 2017-2020 EXi	2006	158000	140000	Petrol	Individual	Manual	Third Owner	17.70	1

In [19]:
```
df_new_drop.tail(3)
```

Out[19]:

	name	year	selling_price	km_driven	fuel	seller_type	transmission	owner	mileage	e
8124	Hyundai Verna CRDi SX	2007	135000	119000	Diesel	Individual	Manual	Fourth & Above Owner	16.80	
8125	Maruti Swift Dzire ZDi	2009	382000	120000	Diesel	Individual	Manual	First Owner	19.30	
8127	Tata Indigo CR4	2013	290000	25000	Diesel	Individual	Manual	First Owner	23.57	

In [20]:
```
print(df_new_drop.shape)
# We have converted or selected features, now let's reset our index!
# Notice how the index is 8127, when we only have 6717 entries!
df_new_drop = df_new_drop.reset_index()
df_new_drop
```

```
(6717, 13)
```

Out[20]:

	index	name	year	selling_price	km_driven	fuel	seller_type	transmission	owner	mi
0	0	Maruti Swift Dzire VDI	2014	450000	145500	Diesel	Individual	Manual	First Owner	
1	1	Skoda Rapid 1.5 TDI Ambition	2014	370000	120000	Diesel	Individual	Manual	Second Owner	
2	2	Honda City 2017-2020 EXi	2006	158000	140000	Petrol	Individual	Manual	Third Owner	
3	3	Hyundai i20 Sportz Diesel	2010	225000	127000	Diesel	Individual	Manual	First Owner	
4	4	Maruti Swift VXI BSIII	2007	130000	120000	Petrol	Individual	Manual	First Owner	
...	...	...	...	...	...	...	...	...	...	
6712	8122	Hyundai i20 Magna 1.4 CRDi	2014	475000	80000	Diesel	Individual	Manual	Second Owner	
6713	8123	Hyundai i20 Magna	2013	320000	110000	Petrol	Individual	Manual	First Owner	
6714	8124	Hyundai Verna CRDi SX	2007	135000	119000	Diesel	Individual	Manual	Fourth & Above Owner	
6715	8125	Maruti Swift Dzire ZDi	2009	382000	120000	Diesel	Individual	Manual	First Owner	
6716	8127	Tata Indigo CR4	2013	290000	25000	Diesel	Individual	Manual	First Owner	

6717 rows × 14 columns

In [21]:
```
# Since we already have the max_power of the car. I will drop the torque feature!
# Also, drop the 'index' column, and make a new column 'company' to store the car

# Dropping the index and torque columns
df_clean = df_new_drop.drop(['index', 'torque'], axis=1)

# Creating a new column 'company'
df_clean['company'] = df_clean['name'].str.split().str[0]

df_clean.head()
```

Out[21]:

	name	year	selling_price	km_driven	fuel	seller_type	transmission	owner	mileage	eng
0	Maruti Swift Dzire VDI	2014	450000	145500	Diesel	Individual	Manual	First Owner	23.40	1
1	Skoda Rapid 1.5 TDI Ambition	2014	370000	120000	Diesel	Individual	Manual	Second Owner	21.14	1
2	Honda City 2017-2020 EXi	2006	158000	140000	Petrol	Individual	Manual	Third Owner	17.70	1
3	Hyundai i20 Sportz Diesel	2010	225000	127000	Diesel	Individual	Manual	First Owner	23.00	1
4	Maruti Swift VXI BSIII	2007	130000	120000	Petrol	Individual	Manual	First Owner	16.10	1

In [22]:
```
df_clean.tail()
```

Out[22]:

	name	year	selling_price	km_driven	fuel	seller_type	transmission	owner	mileage
6712	Hyundai i20 Magna 1.4 CRDi	2014	475000	80000	Diesel	Individual	Manual	Second Owner	22.54
6713	Hyundai i20 Magna	2013	320000	110000	Petrol	Individual	Manual	First Owner	18.50
6714	Hyundai Verna CRDi SX	2007	135000	119000	Diesel	Individual	Manual	Fourth & Above Owner	16.80
6715	Maruti Swift Dzire ZDi	2009	382000	120000	Diesel	Individual	Manual	First Owner	19.30
6716	Tata Indigo CR4	2013	290000	25000	Diesel	Individual	Manual	First Owner	23.57

In [23]:
```
df_clean.shape
```

Out[23]:
```
(6717, 13)
```

In [24]:
```
df_clean.isna().sum()
```

Out[24]:
```
name             0
year             0
selling_price    0
km_driven        0
fuel             0
seller_type      0
transmission     0
owner            0
mileage          0
engine           0
max_power        0
seats            0
company          0
dtype: int64
```

In [25]:
```
df_clean.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6717 entries, 0 to 6716
Data columns (total 13 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   name           6717 non-null   object 
 1   year           6717 non-null   int64  
 2   selling_price  6717 non-null   int64  
 3   km_driven      6717 non-null   int64  
 4   fuel           6717 non-null   object 
 5   seller_type    6717 non-null   object 
 6   transmission   6717 non-null   object 
 7   owner          6717 non-null   object 
 8   mileage        6717 non-null   float64
 9   engine         6717 non-null   int32  
 10  max_power      6717 non-null   float64
 11  seats          6717 non-null   float64
 12  company        6717 non-null   object 
dtypes: float64(3), int32(1), int64(3), object(6)
memory usage: 656.1+ KB
```

In [26]:
```
# We have converted our SI-units data into ints and floats!
# We have cleaned the dataset by removing duplicate values and NaNs!
# We have also reset our index to match with the dataset length!
# Let's move on to EDA
```

In [27]:
```
# Strat EDA
df_clean.describe()
```

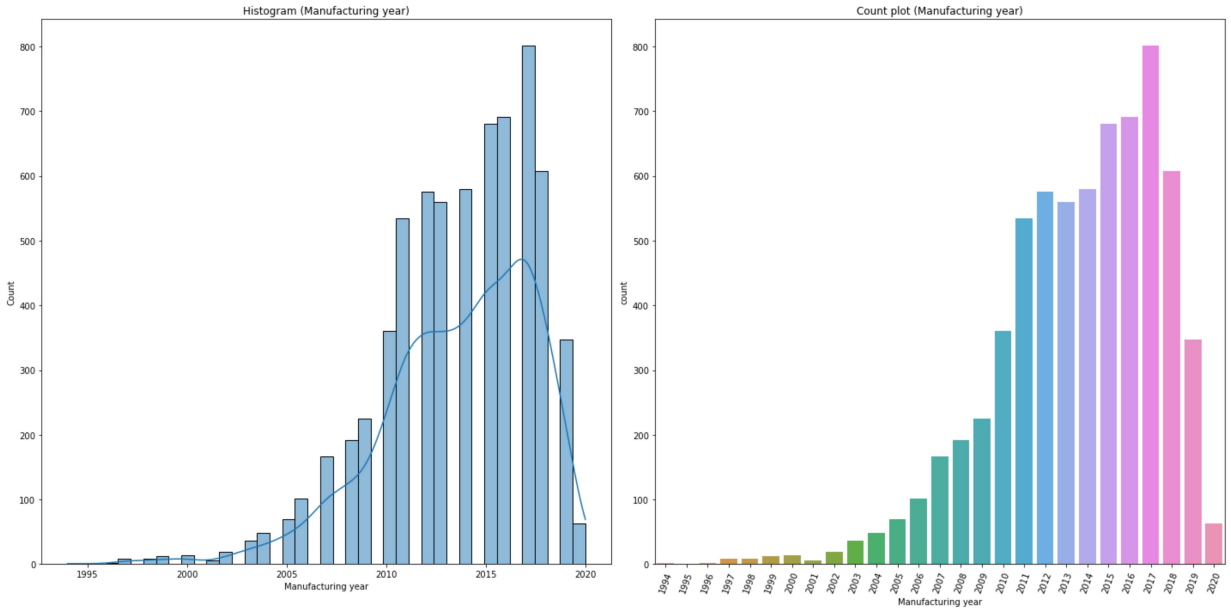
Out[27]:

	year	selling_price	km_driven	mileage	engine	max_power	sea
count	6717.000000	6.717000e+03	6.717000e+03	6717.000000	6717.000000	6717.000000	6717.0000
mean	2013.611136	5.263860e+05	7.339834e+04	19.466585	1430.985857	87.766100	5.4342
std	3.897402	5.235504e+05	5.870328e+04	4.048102	493.469198	31.724555	0.9838
min	1994.000000	2.999900e+04	1.000000e+00	0.000000	624.000000	32.800000	2.0000
25%	2011.000000	2.500000e+05	3.800000e+04	16.800000	1197.000000	67.100000	5.0000
50%	2014.000000	4.200000e+05	6.820300e+04	19.440000	1248.000000	81.830000	5.0000
75%	2017.000000	6.500000e+05	1.000000e+05	22.500000	1498.000000	100.000000	5.0000
max	2020.000000	1.000000e+07	2.360457e+06	42.000000	3604.000000	400.000000	14.0000

In [28]:
```
# import seaborn for data viz.
import seaborn as sns
```

In [29]:
```
# So, before we process our dataset for our ML algo., we should try to explore ou
#-or see how to much our dataset reflects the real-world scenario!

# Let's start with year!
%matplotlib inline
figure, axes = plt.subplots(1, 2, figsize=(20, 10))

sns.histplot(data=df_clean["year"], kde=True, ax=axes[0]).set(xlabel = "Manufactu
sns.countplot(x ='year', data = df_clean, ax=axes[1]).set(xlabel = "Manufacturing
plt.xticks(rotation=70)

save_fig("Hist and Count plot by years")
plt.show()
plt.tight_layout()
```

```
Figure saved
```

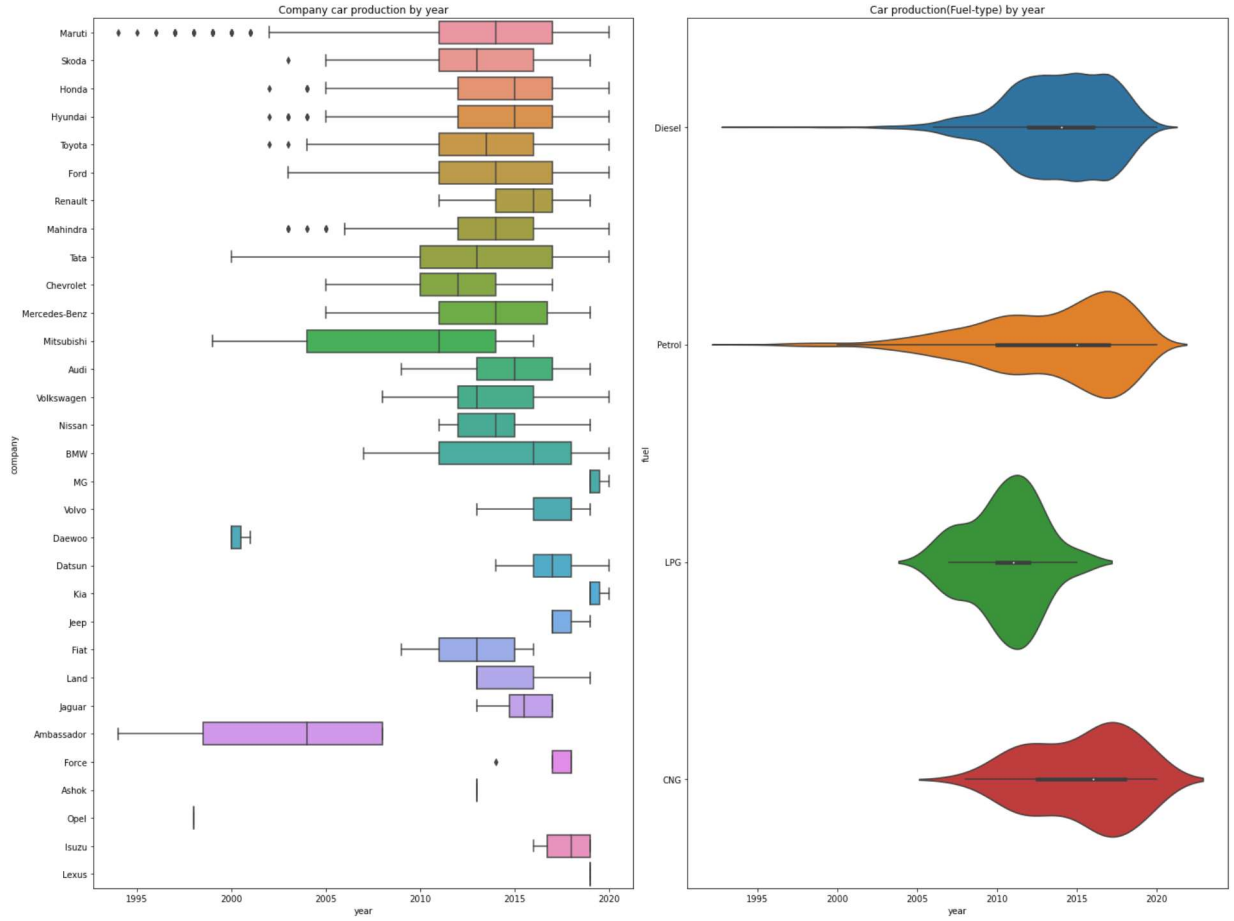



```
<Figure size 432x288 with 0 Axes>
```

In [30]:
```
# From the above plots, we can see that most of the car models present in the dat
# With ~800 car models manufactured in 2017!
```

In [31]:
```
%matplotlib inline
figure, axes = plt.subplots(1, 2, figsize=(20, 15))

sns.boxplot(x='year', y="company", data=df_clean, ax=axes[0]).set(title='Company
sns.violinplot(x='year', y="fuel", data=df_clean, ax=axes[1]).set(title='Car proc

save_fig("Car companies and fuel type by the years")
plt.show()
plt.tight_layout()
```

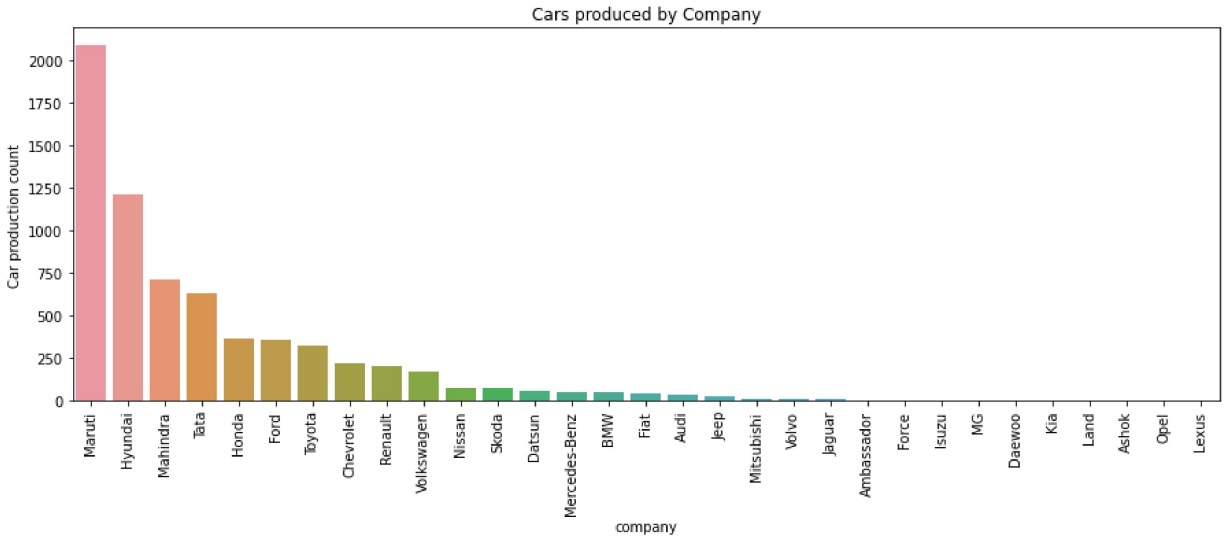
```
Figure saved
```




```
<Figure size 432x288 with 0 Axes>
```

In [32]:
```
# Looking at the above graphs, we can get interesting insights such as:

#        LPG cars were a thing between 2005-2015, a lot of LPG car models were ma
#        Production of CNG cars took an increase after the year of 2015!
#        Diesel cars have been in constant production and seems to be growing!
#        Petrol car production has also increased after the year 2010!

#        Foreign Motor companies like BMW, LEXUS, Kia, Audi, Renault, MG, Isuzu,
#-to the previous decades!

#        Indian companies like Maruti and Tata have been consistent with yheir co
#        Our beloved Ambassador company have stopped its car production after 201

# The dataset reflects the real-world! With Maruti, Tata, Honda, Skoda, etc. domi
```

In [33]:
```
%matplotlib inline
figure, axes = plt.subplots( figsize=(15, 5))

sns.countplot(x="company", data=df_clean, order = df_clean['company'].value_count
plt.xticks(rotation=90)
plt.ylabel("Car production count")

plt.show()
plt.tight_layout()
```

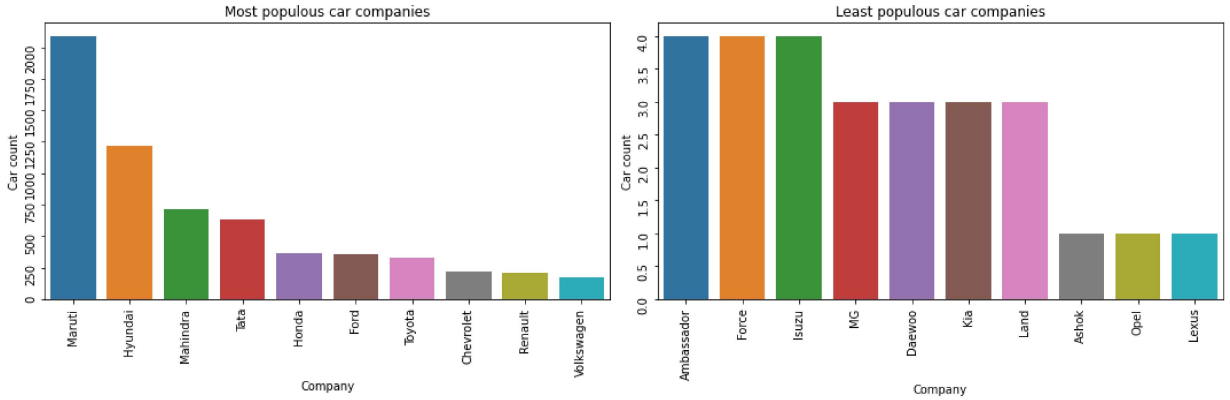



```
<Figure size 432x288 with 0 Axes>
```

In [34]:
```
# Our dataset has car models manufactured by Maruti followed by Hyundai, Mahindra
# Let's see car models manufactured based on most and least populous companies ir
```

In [35]:
```
figure, axes = plt.subplots(1, 2, figsize=(15, 5))

sns.countplot(x="company", data=df_clean, ax=axes[0], order = df_clean['company'
sns.countplot(x="company", data=df_clean, ax=axes[1], order = df_clean['company'

for subplot in axes:
    subplot.set_ylabel("Car count")
    subplot.set_xlabel("Company")
    subplot.tick_params(rotation=90)

# Save the figure
save_fig('Most and Least populous car companies')

plt.show()
plt.tight_layout()
```

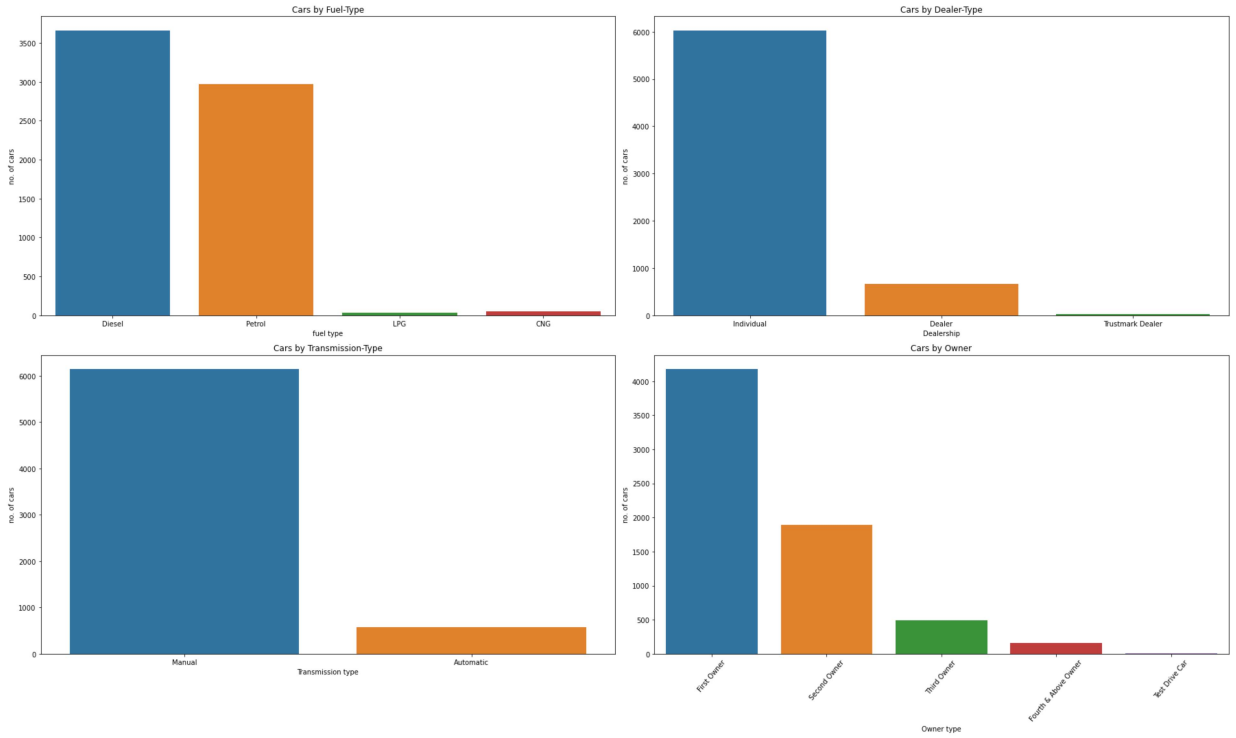
```
Figure saved
```




```
<Figure size 432x288 with 0 Axes>
```

In [36]:
```
%matplotlib inline
figure, ax = plt.subplots(2, 2, figsize=(25, 15))

sns.countplot(x ='fuel', data = df_clean, ax=ax[0, 0]).set(title="Cars by Fuel-Ty
sns.countplot(x ='seller_type', data = df_clean, ax=ax[0, 1]).set(title="Cars by
sns.countplot(x ='transmission', data = df_clean, ax=ax[1, 0]).set(title="Cars by
sns.countplot(x ='owner', data = df_clean, ax=ax[1, 1]).set(title="Cars by Owner'
plt.xticks(rotation=50)

# Save the figure
save_fig('Car count based on categorical features')

plt.show()
plt.tight_layout()
```

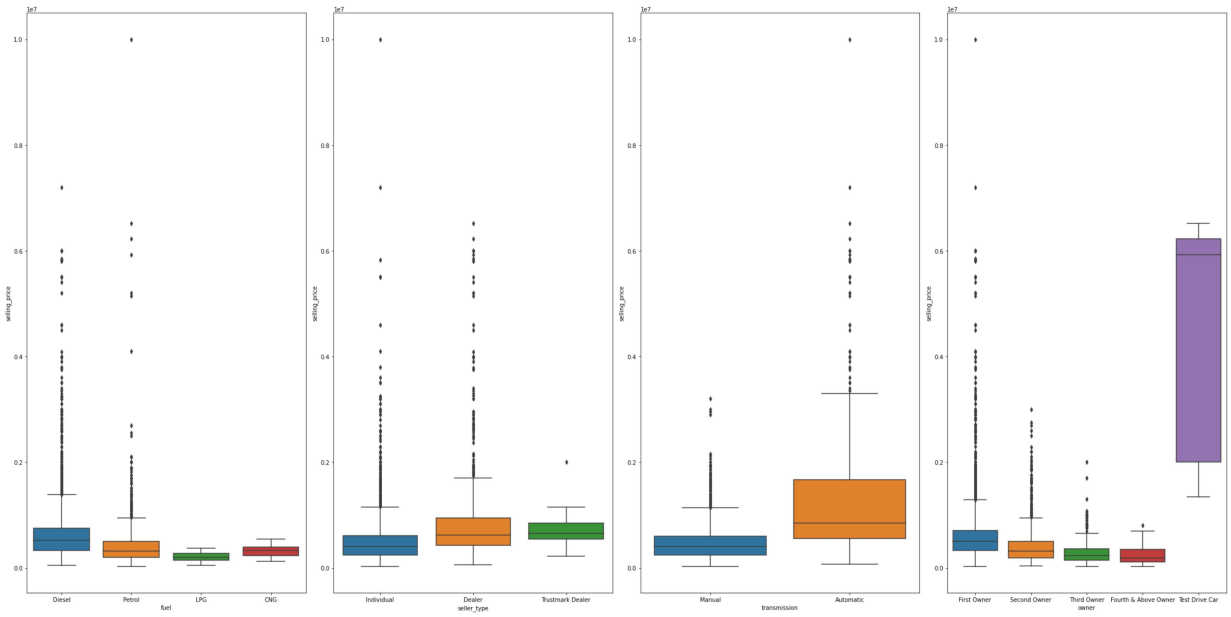
Figure saved



```
<Figure size 432x288 with 0 Axes>
```

In [37]:
```
# Let's Boxplot our categorical features with the numerical features!
figure, ax = plt.subplots(1,4, figsize=(30, 15))

sns.boxplot(x='fuel', y="selling_price", ax=ax[0], data=df_clean)
sns.boxplot(x='seller_type', y="selling_price", ax=ax[1], data=df_clean)
sns.boxplot(x='transmission', y="selling_price", ax=ax[2], data=df_clean)
sns.boxplot(x='owner', y="selling_price", ax=ax[3], data=df_clean)

save_fig('Selling price vs categorical features')

plt.show()
plt.tight_layout()
```

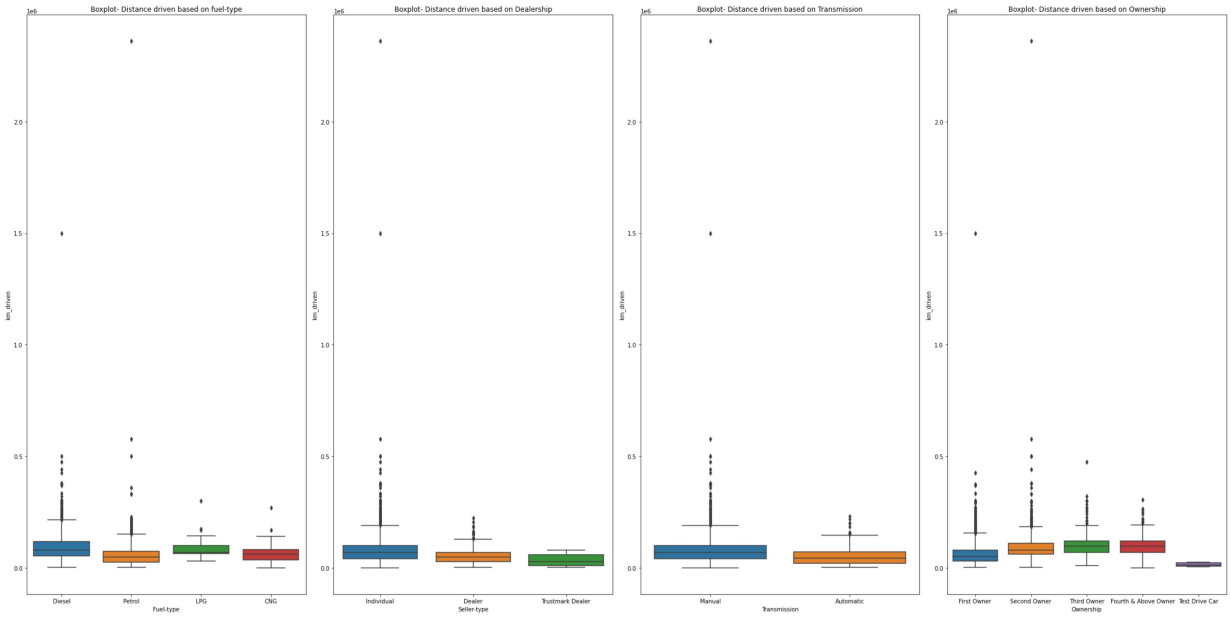
```
Figure saved
```




```
<Figure size 432x288 with 0 Axes>
```

In [38]:
```
# Let's Boxplot our categorical features with the numerical features!
figure, ax = plt.subplots(1,4, figsize=(30, 15))

sns.boxplot(x='fuel', y="km_driven", ax=ax[0], data=df_clean).set(title='Boxplot-
sns.boxplot(x='seller_type', y="km_driven", ax=ax[1], data=df_clean).set(title='E
sns.boxplot(x='transmission', y="km_driven", ax=ax[2], data=df_clean).set(title='
sns.boxplot(x='owner', y="km_driven", ax=ax[3], data=df_clean).set(title='Boxplot

save_fig('Distance driven vs categorical features')

plt.show()
plt.tight_layout()
```

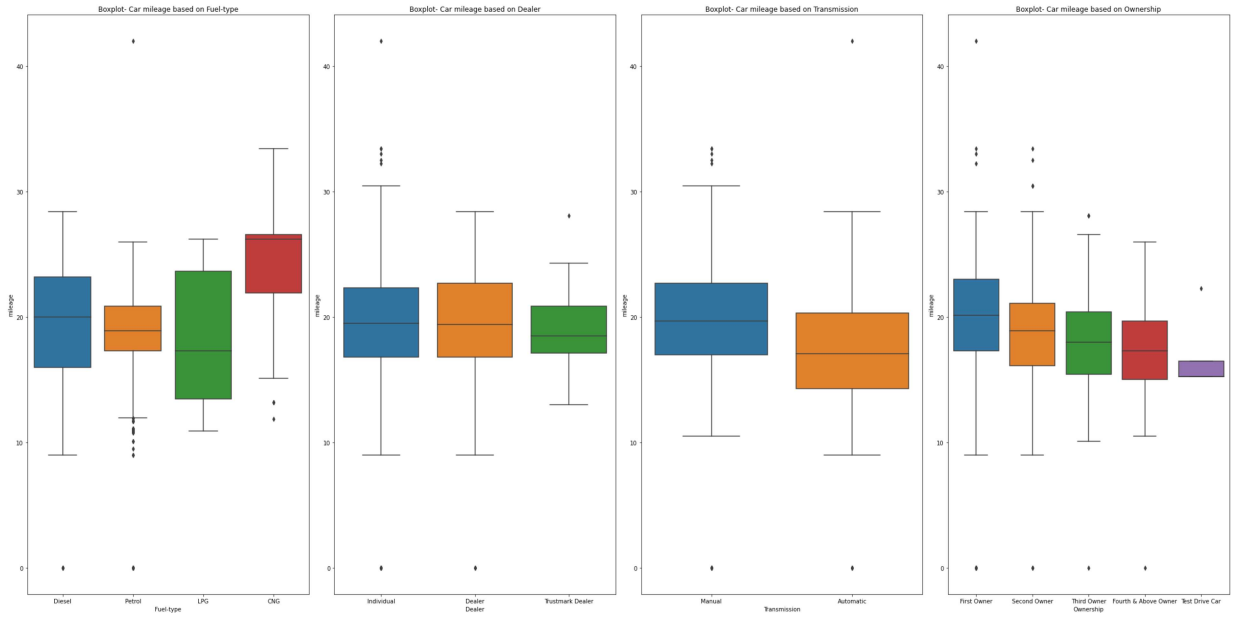
Figure saved



<Figure size 432x288 with 0 Axes>

In [39]:
```
# Let's Boxplot our categorical features with the numerical features!
figure, ax = plt.subplots(1,4, figsize=(30, 15))

sns.boxplot(x='fuel', y="mileage", ax=ax[0], data=df_clean).set(title='Boxplot- C
sns.boxplot(x='seller_type', y="mileage", ax=ax[1], data=df_clean).set(title='Box
sns.boxplot(x='transmission', y="mileage", ax=ax[2], data=df_clean).set(title='Bo
sns.boxplot(x='owner', y="mileage", ax=ax[3], data=df_clean).set(title='Boxplot-

save_fig('Car mileage vs categorical features')

plt.show()
plt.tight_layout()
```

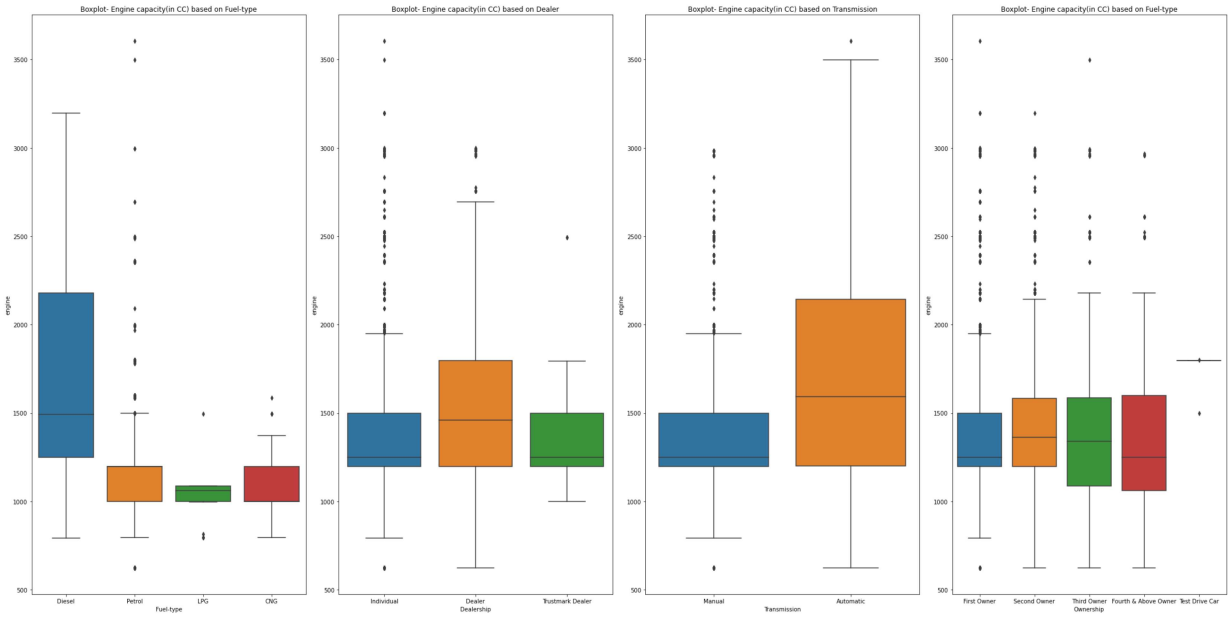
```
Figure saved
```




```
<Figure size 432x288 with 0 Axes>
```

In [40]:
```
figure, ax = plt.subplots(1,4, figsize=(30, 15))

sns.boxplot(x='fuel', y="engine", ax=ax[0], data=df_clean).set(title='Boxplot- Er
sns.boxplot(x='seller_type', y="engine", ax=ax[1], data=df_clean).set(title='Boxp
sns.boxplot(x='transmission', y="engine", ax=ax[2], data=df_clean).set(title='Box
sns.boxplot(x='owner', y="engine", ax=ax[3], data=df_clean).set(title='Boxplot- E

save_fig('Engine Capacity(cc) vs categorical features')

plt.show()
plt.tight_layout()
```

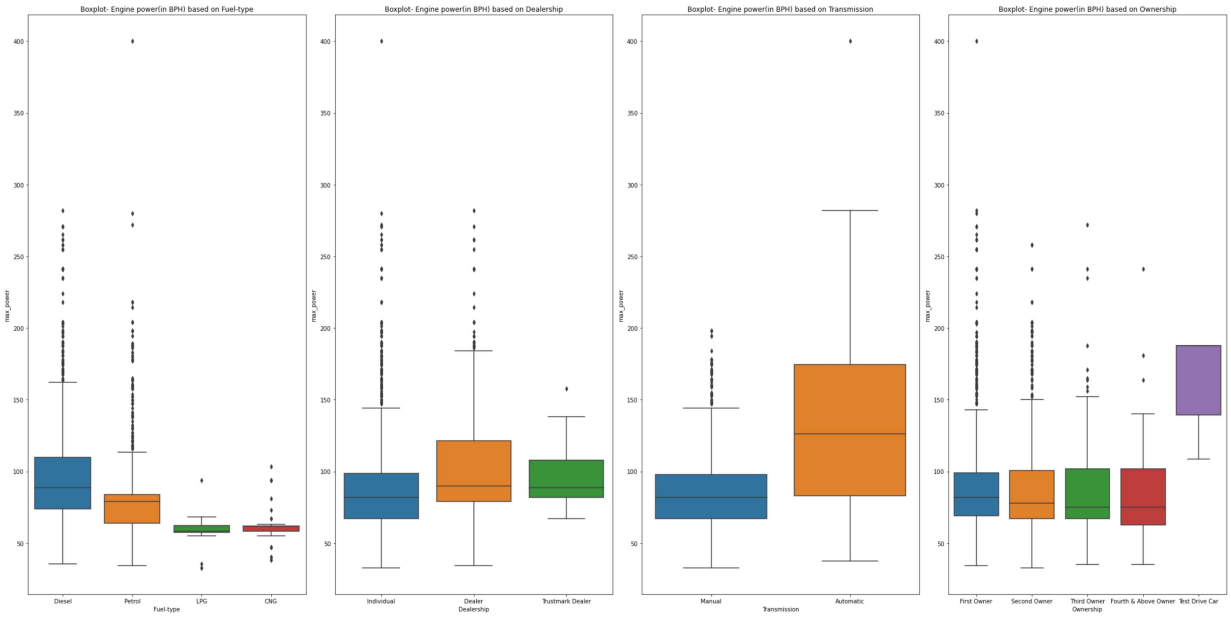
```
Figure saved
```




```
<Figure size 432x288 with 0 Axes>
```

In [41]:
```
figure, ax = plt.subplots(1,4, figsize=(30, 15))

sns.boxplot(x='fuel', y="max_power", ax=ax[0], data=df_clean).set(title='Boxplot-
sns.boxplot(x='seller_type', y="max_power", ax=ax[1], data=df_clean).set(title='E
sns.boxplot(x='transmission', y="max_power", ax=ax[2], data=df_clean).set(title='
sns.boxplot(x='owner', y="max_power", ax=ax[3], data=df_clean).set(title='Boxplot

save_fig('Engine power(BHP) vs categorical features')

plt.show()
plt.tight_layout()
```

```
Figure saved
```




```
<Figure size 432x288 with 0 Axes>
```

In [42]:
```
# We have explored the categorical features in the dataset based on numerical cat
# Let's go a bit deeper and explore these numerical features based on car models!

# Create a new DataFrame with car name and its features!
att = ['name','selling_price', 'km_driven', 'mileage', 'engine', 'max_power', 'se
df_car_details = df_clean[att]

# Dividing the selling price by 1,00,000 (1 lakh) to get a simplify the selling p
df_car_details['selling_price'] = df_car_details['selling_price']/100000

# Dividing the km_driven by 1,000
df_car_details['km_driven'] = df_car_details['km_driven']/1000
df_car_details
```

Out[42]:

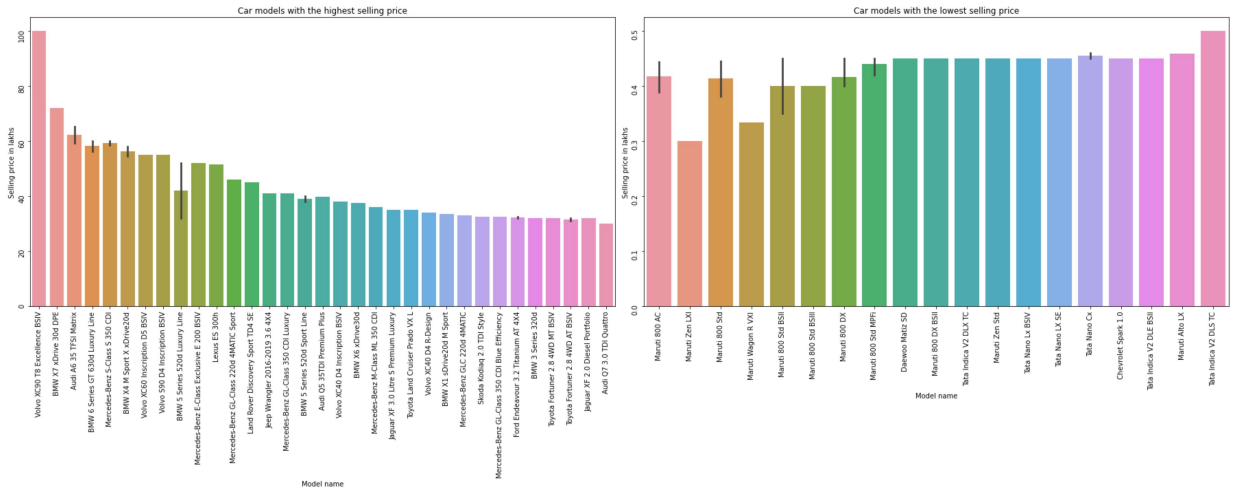
	name	selling_price	km_driven	mileage	engine	max_power	seats
0	Maruti Swift Dzire VDI	4.50	145.5	23.40	1248	74.00	5.0
1	Skoda Rapid 1.5 TDI Ambition	3.70	120.0	21.14	1498	103.52	5.0
2	Honda City 2017-2020 EXi	1.58	140.0	17.70	1497	78.00	5.0
3	Hyundai i20 Sportz Diesel	2.25	127.0	23.00	1396	90.00	5.0
4	Maruti Swift VXI BSIII	1.30	120.0	16.10	1298	88.20	5.0
...	...	...	...	...	...	...	...
6712	Hyundai i20 Magna 1.4 CRDi	4.75	80.0	22.54	1396	88.73	5.0
6713	Hyundai i20 Magna	3.20	110.0	18.50	1197	82.85	5.0
6714	Hyundai Verna CRDi SX	1.35	119.0	16.80	1493	110.00	5.0
6715	Maruti Swift Dzire ZDi	3.82	120.0	19.30	1248	73.90	5.0
6716	Tata Indigo CR4	2.90	25.0	23.57	1396	70.00	5.0

6717 rows × 7 columns

In [43]:
```
# Let's explore our car models based on price, km_driven, mileage, engine, and ma
df_car_details_sorted_price = df_car_details.sort_values(by=['selling_price'], as
df_car_details_sorted_price_asc = df_car_details.sort_values(by=['selling_price']

df_car_details_sorted_driven = df_car_details.sort_values(by=['km_driven'], ascen
df_car_details_sorted_driven_asc = df_car_details.sort_values(by=['km_driven'], a

df_car_details_sorted_mileage = df_car_details.sort_values(by=['mileage'], ascend
df_car_details_sorted_mileage_asc = df_car_details.sort_values(by=['mileage'], as

df_car_details_sorted_engine = df_car_details.sort_values(by=['engine'], ascendin
df_car_details_sorted_engine_asc = df_car_details.sort_values(by=['engine'], asce

df_car_details_sorted_power = df_car_details.sort_values(by=['max_power'], ascend
df_car_details_sorted_power_asc = df_car_details.sort_values(by=['max_power'], as
```

In [44]:
```
%matplotlib inline
figure, axes = plt.subplots(1, 2, figsize=(25, 10))

sns.barplot(x='name', y='selling_price', ax=axes[0], data=df_car_details_sorted_p
sns.barplot(x='name', y='selling_price', ax=axes[1], data=df_car_details_sorted_p

for subplot in axes:
    subplot.tick_params(rotation=90)


save_fig('Cars with highest and lowest prices')

plt.show()
plt.tight_layout()
```

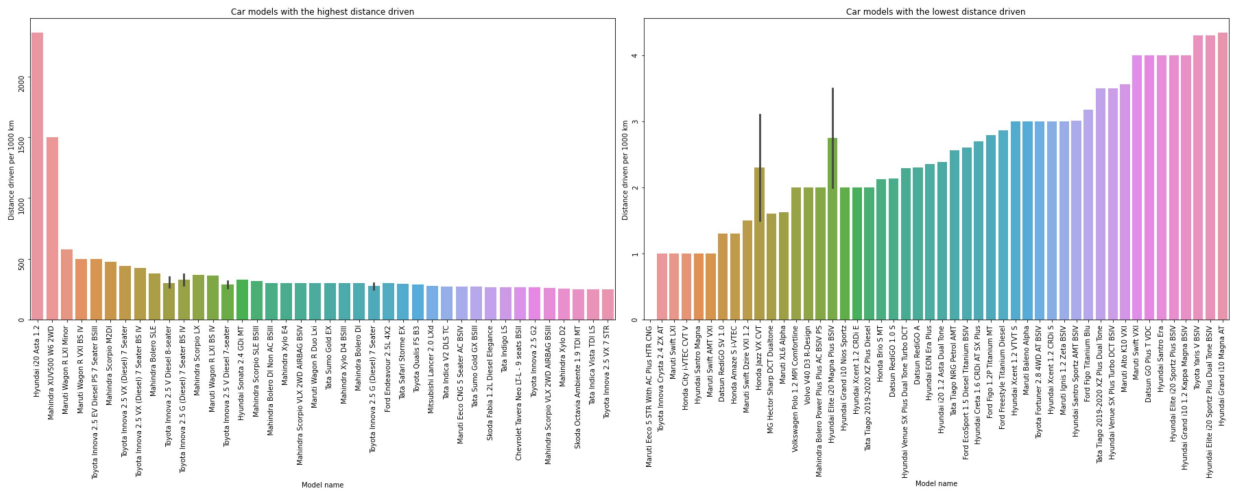
```
Figure saved
```




```
<Figure size 432x288 with 0 Axes>
```

In [45]:
```
%matplotlib inline
figure, axes = plt.subplots(1, 2, figsize=(25, 10))

sns.barplot(x='name', y='km_driven', ax=axes[0], data=df_car_details_sorted_drive
sns.barplot(x='name', y='km_driven', ax=axes[1], data=df_car_details_sorted_drive


for subplot in axes:
    subplot.tick_params(rotation=90)

save_fig('Cars with highest and lowest distance travelled')

plt.show()
plt.tight_layout()
```

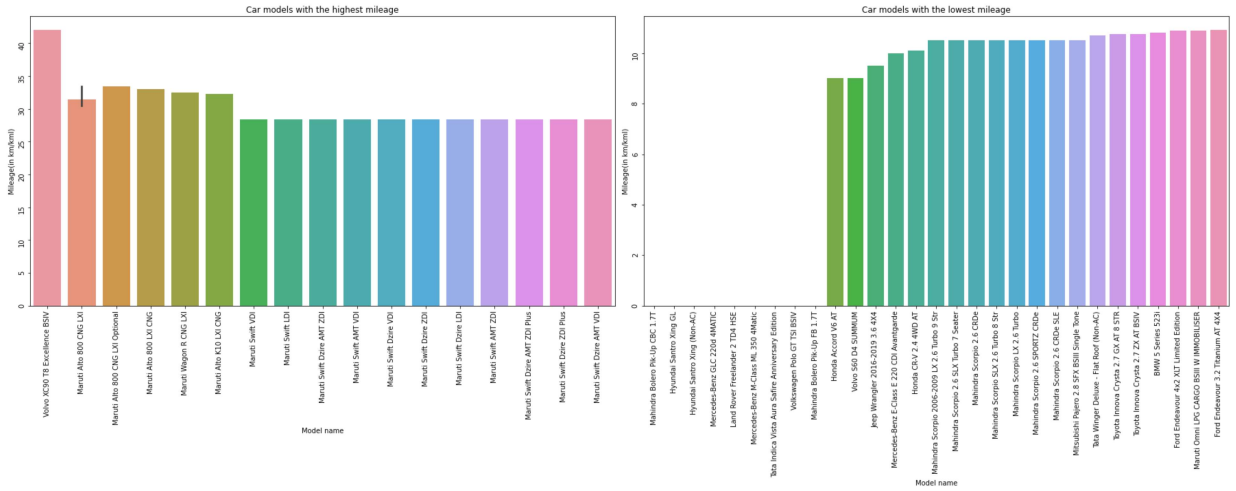
```
Figure saved
```




```
<Figure size 432x288 with 0 Axes>
```

In [46]:
```
%matplotlib inline
figure, axes = plt.subplots(1, 2, figsize=(25, 10))

sns.barplot(x='name', y='mileage', ax=axes[0], data=df_car_details_sorted_mileage
sns.barplot(x='name', y='mileage', ax=axes[1], data=df_car_details_sorted_mileage


for subplot in axes:
    subplot.tick_params(rotation=90)

save_fig('Cars with highest and lowest mileage')

plt.show()
plt.tight_layout()
```

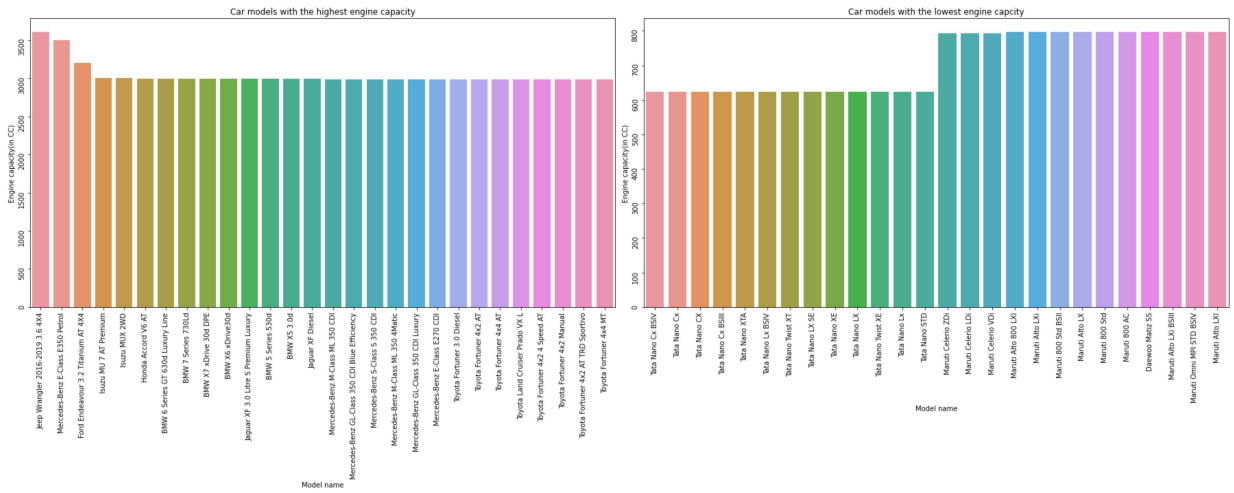
```
Figure saved
```




```
<Figure size 432x288 with 0 Axes>
```

In [47]:
```
%matplotlib inline
figure, axes = plt.subplots(1, 2, figsize=(25, 10))

sns.barplot(x='name', y='engine', ax=axes[0], data=df_car_details_sorted_engine).
sns.barplot(x='name', y='engine', ax=axes[1], data=df_car_details_sorted_engine_a


for subplot in axes:
    subplot.tick_params(rotation=90)

save_fig('Cars with highest and lowest engine capacity')

plt.show()
plt.tight_layout()
```

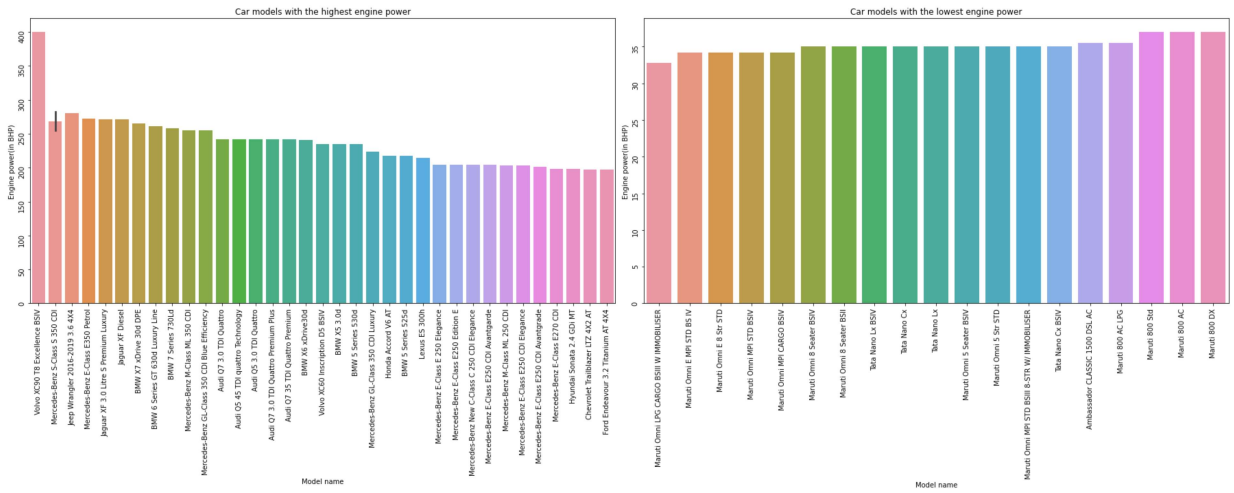
```
Figure saved
```




```
<Figure size 432x288 with 0 Axes>
```

In [48]:
```
%matplotlib inline
figure, axes = plt.subplots(1, 2, figsize=(25, 10))

sns.barplot(x='name', y='max_power', ax=axes[0], data=df_car_details_sorted_power
sns.barplot(x='name', y='max_power', ax=axes[1], data=df_car_details_sorted_power


for subplot in axes:
    subplot.tick_params(rotation=90)

save_fig('Cars with highest and lowest engine power')

plt.show()
plt.tight_layout()
```

```
Figure saved
```




```
<Figure size 432x288 with 0 Axes>
```

In [49]:
```
# Multi-variable analysis and correlation!
# Let's hist- and scatter-plot our dataset and see its distribution!
df_clean.head()
```

Out[49]:

	name	year	selling_price	km_driven	fuel	seller_type	transmission	owner	mileage	eng
0	Maruti Swift Dzire VDI	2014	450000	145500	Diesel	Individual	Manual	First Owner	23.40	1
1	Skoda Rapid 1.5 TDI Ambition	2014	370000	120000	Diesel	Individual	Manual	Second Owner	21.14	1
2	Honda City 2017-2020 EXi	2006	158000	140000	Petrol	Individual	Manual	Third Owner	17.70	1
3	Hyundai i20 Sportz Diesel	2010	225000	127000	Diesel	Individual	Manual	First Owner	23.00	1
4	Maruti Swift VXI BSIII	2007	130000	120000	Petrol	Individual	Manual	First Owner	16.10	1

In [50]:
```
# From the above plots we can see that some of our data may be missing!
# Some of our car models showed zero('0') mileage! So, either these cars are beir
# Let's look at these zero mileage models and see what could we do!
df_clean.loc[df_clean['mileage'] == 0]

# These models seem to have been driven for a while. So, it is very unlikely that
# So, assuming that these cars are missing their mileage data, we can drop these
```

Out[50]:

	name	year	selling_price	km_driven	fuel	seller_type	transmission	owner	mileag
457	Tata Indica Vista Aura Safire Anniversary Edition	2009	135000	28900	Petrol	Individual	Manual	Second Owner	0.
568	Hyundai Santro Xing GL	2009	120000	90000	Petrol	Individual	Manual	Second Owner	0.
1238	Hyundai Santro Xing GL	2008	105000	128000	Petrol	Individual	Manual	First Owner	0.
1579	Land Rover Freelander 2 TD4 HSE	2013	1650000	64788	Diesel	Dealer	Automatic	First Owner	0.
2079	Hyundai Santro Xing (Non-AC)	2013	184000	15000	Petrol	Individual	Manual	First Owner	0.
3513	Mercedes-Benz M-Class ML 350 4Matic	2011	1700000	110000	Diesel	Individual	Automatic	Third Owner	0.
4162	Hyundai Santro Xing GL	2008	175000	40000	Petrol	Individual	Manual	First Owner	0.
4661	Volkswagen Polo GT TSI BSIV	2014	574000	28080	Petrol	Dealer	Automatic	First Owner	0.
4664	Volkswagen Polo GT TSI BSIV	2014	575000	28100	Petrol	Dealer	Automatic	First Owner	0.
4718	Mahindra Bolero Pik-Up FB 1.7T	2020	679000	5000	Diesel	Individual	Manual	First Owner	0.
5304	Hyundai Santro Xing GL	2010	150000	110000	Petrol	Individual	Manual	First Owner	0.
5359	Mahindra Bolero Pik-Up CBC 1.7T	2019	722000	80000	Diesel	Individual	Manual	First Owner	0.
5529	Hyundai Santro Xing GL	2011	150000	40000	Petrol	Individual	Manual	Fourth & Above Owner	0.

	name	year	selling_price	km_driven	fuel	seller_type	transmission	owner	mileag
5683	Hyundai Santro Xing (Non-AC)	2010	110000	80000	Petrol	Individual	Manual	Second Owner	0.
6005	Mercedes-Benz GLC 220d 4MATIC	2017	3300000	60000	Diesel	Dealer	Automatic	First Owner	0.

In [51]:
```
df_clean.loc[df_clean['mileage'] == 0].shape
```

Out[51]:
```
(15, 13)
```

In [52]:
```
df_clean.shape
```

Out[52]:
```
(6717, 13)
```

In [53]:
```
index_names = df_clean.loc[df_clean['mileage'] == 0].index
index_names
```

Out[53]:
```
Int64Index([ 457,  568, 1238, 1579, 2079, 3513, 4162, 4661, 4664, 4718, 5304,
            5359, 5529, 5683, 6005],
           dtype='int64')
```

In [54]:
```
df_clean = df_clean.drop(index_names)
df_clean
```

Out[54]:

	name	year	selling_price	km_driven	fuel	seller_type	transmission	owner	mileage
0	Maruti Swift Dzire VDI	2014	450000	145500	Diesel	Individual	Manual	First Owner	23.40
1	Skoda Rapid 1.5 TDI Ambition	2014	370000	120000	Diesel	Individual	Manual	Second Owner	21.14
2	Honda City 2017-2020 EXi	2006	158000	140000	Petrol	Individual	Manual	Third Owner	17.70
3	Hyundai i20 Sportz Diesel	2010	225000	127000	Diesel	Individual	Manual	First Owner	23.00
4	Maruti Swift VXI BSIII	2007	130000	120000	Petrol	Individual	Manual	First Owner	16.10
...	...	...	...	...	...	...	...	...	...
6712	Hyundai i20 Magna 1.4 CRDi	2014	475000	80000	Diesel	Individual	Manual	Second Owner	22.54
6713	Hyundai i20 Magna	2013	320000	110000	Petrol	Individual	Manual	First Owner	18.50
6714	Hyundai Verna CRDi SX	2007	135000	119000	Diesel	Individual	Manual	Fourth & Above Owner	16.80
6715	Maruti Swift Dzire ZDi	2009	382000	120000	Diesel	Individual	Manual	First Owner	19.30
6716	Tata Indigo CR4	2013	290000	25000	Diesel	Individual	Manual	First Owner	23.57

6702 rows × 13 columns

In [55]:
```
df_clean.shape
```

Out[55]:
```
(6702, 13)
```

In [56]:
```
df_clean = df_clean.reset_index()
df_clean = df_clean.drop(['index'], axis=1)
df_clean
```

Out[56]:

	name	year	selling_price	km_driven	fuel	seller_type	transmission	owner	mileage
0	Maruti Swift Dzire VDI	2014	450000	145500	Diesel	Individual	Manual	First Owner	23.40
1	Skoda Rapid 1.5 TDI Ambition	2014	370000	120000	Diesel	Individual	Manual	Second Owner	21.14
2	Honda City 2017-2020 EXi	2006	158000	140000	Petrol	Individual	Manual	Third Owner	17.70
3	Hyundai i20 Sportz Diesel	2010	225000	127000	Diesel	Individual	Manual	First Owner	23.00
4	Maruti Swift VXI BSIII	2007	130000	120000	Petrol	Individual	Manual	First Owner	16.10
...	...	...	...	...	...	...	...	...	...
6697	Hyundai i20 Magna 1.4 CRDi	2014	475000	80000	Diesel	Individual	Manual	Second Owner	22.54
6698	Hyundai i20 Magna	2013	320000	110000	Petrol	Individual	Manual	First Owner	18.50
6699	Hyundai Verna CRDi SX	2007	135000	119000	Diesel	Individual	Manual	Fourth & Above Owner	16.80
6700	Maruti Swift Dzire ZDi	2009	382000	120000	Diesel	Individual	Manual	First Owner	19.30
6701	Tata Indigo CR4	2013	290000	25000	Diesel	Individual	Manual	First Owner	23.57

6702 rows × 13 columns

In [57]:
```
# Let's build our model to predict car prices!
# Our target variable will be 'selling_price'
# Let's build two models: Linear Regression model and Random Forest and compare t
# Before building the model, let's note some important aspects:
#                             1. Our dataset is clean i.e it has no null or duplic
#                             2. The dataset has 4 categorical features(['fuel',
#                             3. The dataset has 7-1 numerical features(['selling_
#                             4. The dataset has a lot of Outliers which we have t
#                             5. The dataset has multicollinearity which will affe
```

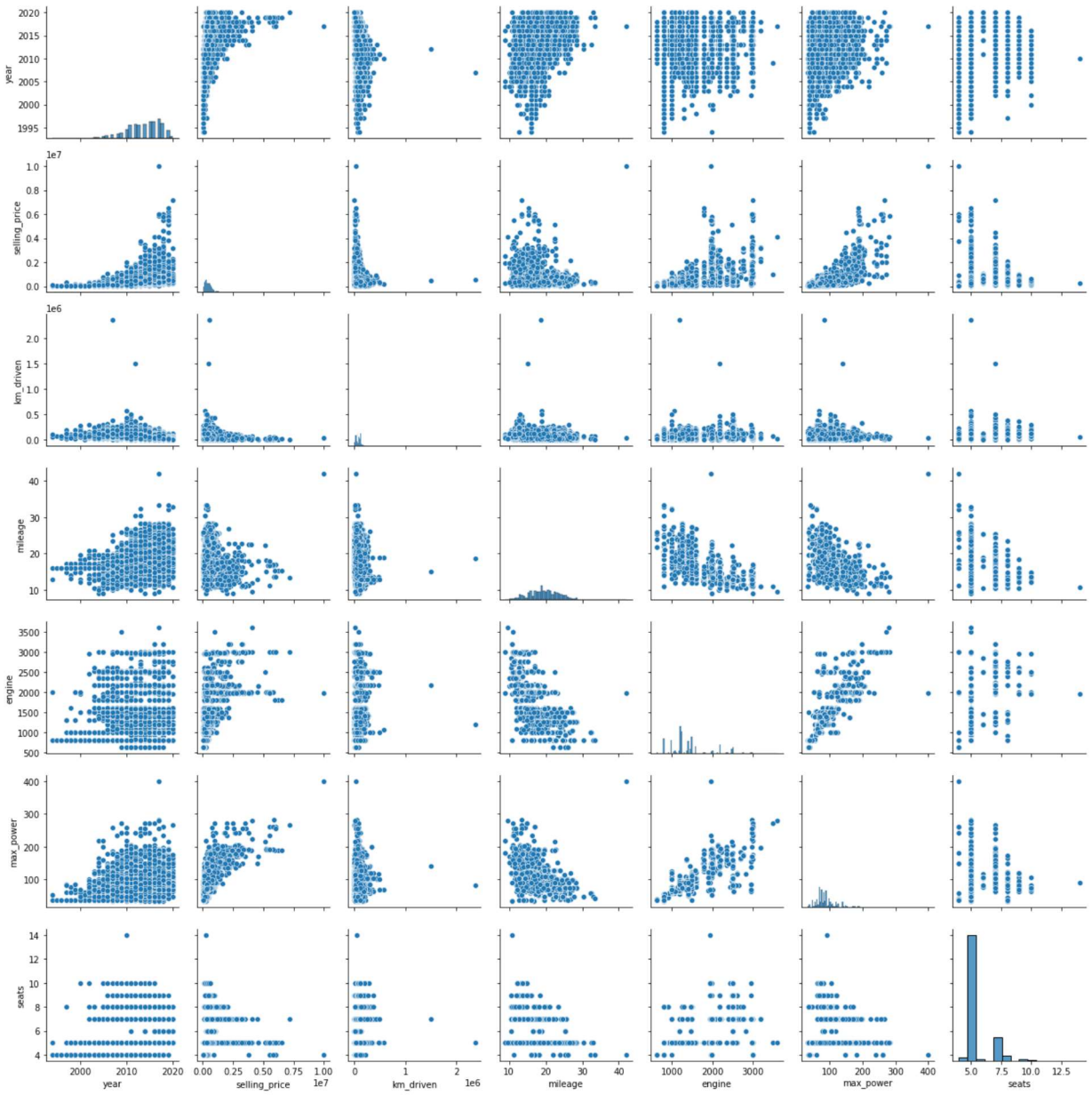
In [58]:
```
# Model making: Based on above info, we should convert our categorical data into
# Let's look at our categorical and numerical data!
```

In [59]:
```
attr_cat = ['fuel', 'seller_type', 'transmission', 'owner']
attr_num = ['selling_price', 'km_driven', 'mileage', 'engine', 'max_power', 'seat

print(df_clean[attr_cat]) # Categorical variables
print(df_clean[attr_num]) # Numerical variables
```

```
        fuel seller_type transmission                owner
0     Diesel  Individual       Manual            First Owner
1     Diesel  Individual       Manual           Second Owner
2     Petrol  Individual       Manual            Third Owner
3     Diesel  Individual       Manual            First Owner
4     Petrol  Individual       Manual            First Owner
...      ...         ...          ...                  ...
6697  Diesel  Individual       Manual           Second Owner
6698  Petrol  Individual       Manual            First Owner
6699  Diesel  Individual       Manual  Fourth & Above Owner
6700  Diesel  Individual       Manual            First Owner
6701  Diesel  Individual       Manual            First Owner

[6702 rows x 4 columns]
      selling_price  km_driven  mileage  engine  max_power  seats
0            450000     145500    23.40    1248      74.00    5.0
1            370000     120000    21.14    1498     103.52    5.0
2            158000     140000    17.70    1497      78.00    5.0
3            225000     127000    23.00    1396      90.00    5.0
4            130000     120000    16.10    1298      88.20    5.0
...             ...        ...      ...     ...        ...    ...
6697         475000      80000    22.54    1396      88.73    5.0
6698         320000     110000    18.50    1197      82.85    5.0
6699         135000     119000    16.80    1493     110.00    5.0
6700         382000     120000    19.30    1248      73.90    5.0
6701         290000      25000    23.57    1396      70.00    5.0

[6702 rows x 6 columns]
```

In [60]:
```
# Let's use the hist feature of matplotlib to map-out all the numerical variables
%matplotlib inline
sns.pairplot(df_clean)
save_fig("Attribute_histogram_plots_car_prices")
plt.show()
plt.tight_layout()
```

```
Figure saved
```




```
<Figure size 432x288 with 0 Axes>
```

In [61]:
```
# Make and use a copy our dataset!
df_preprocessed = df_clean.copy()
df_preprocessed
```

Out[61]:

	name	year	selling_price	km_driven	fuel	seller_type	transmission	owner	mileage
0	Maruti Swift Dzire VDI	2014	450000	145500	Diesel	Individual	Manual	First Owner	23.40
1	Skoda Rapid 1.5 TDI Ambition	2014	370000	120000	Diesel	Individual	Manual	Second Owner	21.14
2	Honda City 2017-2020 EXi	2006	158000	140000	Petrol	Individual	Manual	Third Owner	17.70
3	Hyundai i20 Sportz Diesel	2010	225000	127000	Diesel	Individual	Manual	First Owner	23.00
4	Maruti Swift VXI BSIII	2007	130000	120000	Petrol	Individual	Manual	First Owner	16.10
...	...	...	...	...	...	...	...	...	...
6697	Hyundai i20 Magna 1.4 CRDi	2014	475000	80000	Diesel	Individual	Manual	Second Owner	22.54
6698	Hyundai i20 Magna	2013	320000	110000	Petrol	Individual	Manual	First Owner	18.50
6699	Hyundai Verna CRDi SX	2007	135000	119000	Diesel	Individual	Manual	Fourth & Above Owner	16.80
6700	Maruti Swift Dzire ZDi	2009	382000	120000	Diesel	Individual	Manual	First Owner	19.30
6701	Tata Indigo CR4	2013	290000	25000	Diesel	Individual	Manual	First Owner	23.57

6702 rows × 13 columns

In [62]:
```
# Import libraries!
from sklearn.preprocessing import OneHotEncoder
from sklearn import preprocessing

# NOTE: I would be using OneHotEncoder/ Dummy-values to convert all of the catego
#-it will be a better idea to use the above encoder!
# I would also use min-max scaler to transform our numerical features!
```

In [63]:
```
# One-Hot
one_hot_encoded_data = pd.get_dummies(df_preprocessed, columns = ['fuel', 'seller
print(one_hot_encoded_data.columns, '\n')
one_hot_encoded_data
```

```
Index(['name', 'year', 'selling_price', 'km_driven', 'mileage', 'engine',
       'max_power', 'seats', 'company', 'fuel_CNG', 'fuel_Diesel', 'fuel_LPG',
       'fuel_Petrol', 'seller_type_Dealer', 'seller_type_Individual',
       'seller_type_Trustmark Dealer', 'transmission_Automatic',
       'transmission_Manual', 'owner_First Owner',
       'owner_Fourth & Above Owner', 'owner_Second Owner',
       'owner_Test Drive Car', 'owner_Third Owner'],
      dtype='object')
```

Out[63]:

	name	year	selling_price	km_driven	mileage	engine	max_power	seats	company	fuel
0	Maruti Swift Dzire VDI	2014	450000	145500	23.40	1248	74.00	5.0	Maruti	
1	Skoda Rapid 1.5 TDI Ambition	2014	370000	120000	21.14	1498	103.52	5.0	Skoda	
2	Honda City 2017-2020 EXi	2006	158000	140000	17.70	1497	78.00	5.0	Honda	
3	Hyundai i20 Sportz Diesel	2010	225000	127000	23.00	1396	90.00	5.0	Hyundai	
4	Maruti Swift VXI BSIII	2007	130000	120000	16.10	1298	88.20	5.0	Maruti	
...	...	...	...	...	...	...	...	...	...	...
6697	Hyundai i20 Magna 1.4 CRDi	2014	475000	80000	22.54	1396	88.73	5.0	Hyundai	
6698	Hyundai i20 Magna	2013	320000	110000	18.50	1197	82.85	5.0	Hyundai	
6699	Hyundai Verna CRDi SX	2007	135000	119000	16.80	1493	110.00	5.0	Hyundai	
6700	Maruti Swift Dzire ZDi	2009	382000	120000	19.30	1248	73.90	5.0	Maruti	

	name	year	selling_price	km_driven	mileage	engine	max_power	seats	company	fuel
6701	Tata Indigo CR4	2013	290000	25000	23.57	1396	70.00	5.0	Tata	

6702 rows × 23 columns

In [64]:
```
one_hot_encoded_data.head()
```

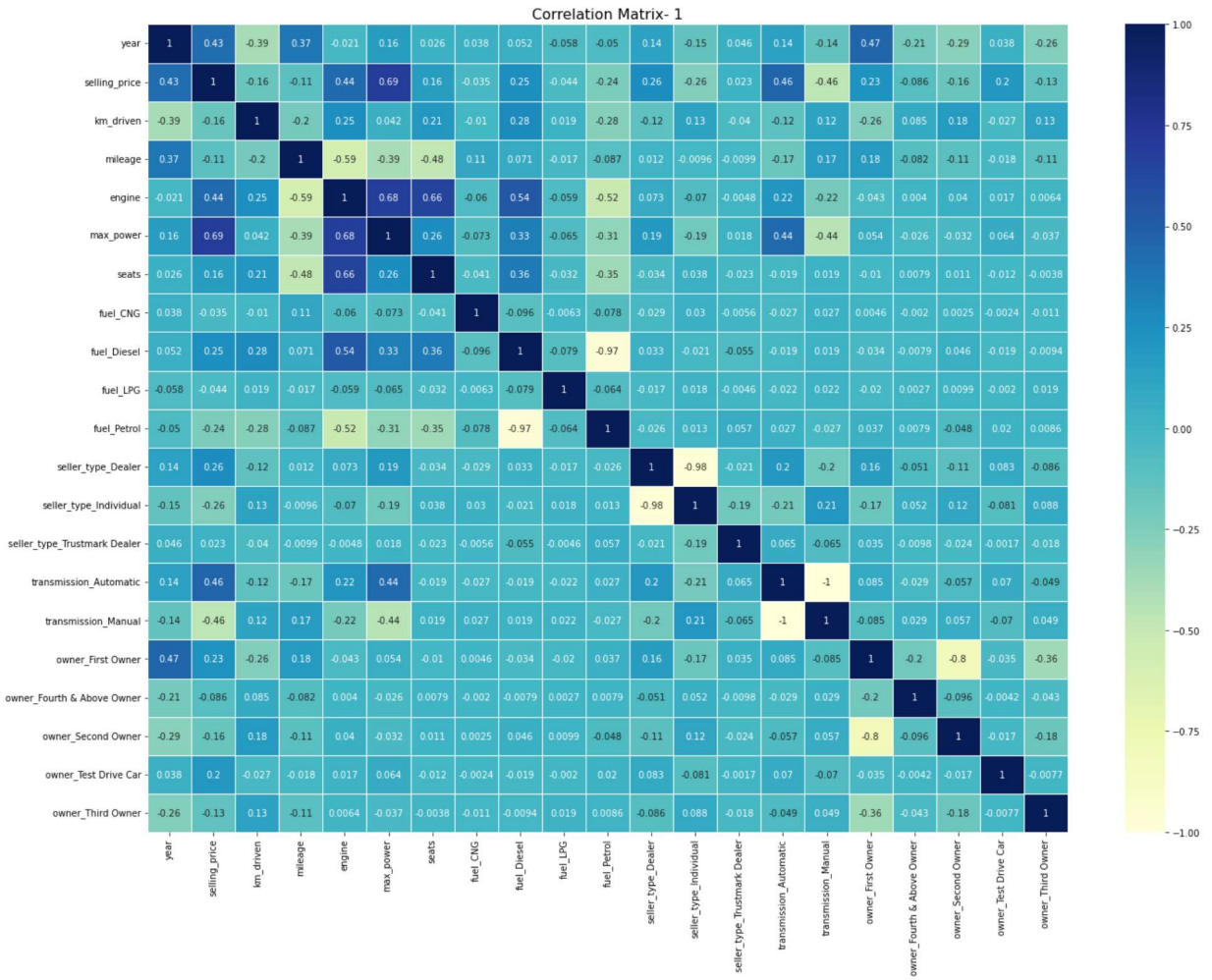
Out[64]:

	name	year	selling_price	km_driven	mileage	engine	max_power	seats	company	fuel_CN
0	Maruti Swift Dzire VDI	2014	450000	145500	23.40	1248	74.00	5.0	Maruti	
1	Skoda Rapid 1.5 TDI Ambition	2014	370000	120000	21.14	1498	103.52	5.0	Skoda	
2	Honda City 2017-2020 EXi	2006	158000	140000	17.70	1497	78.00	5.0	Honda	
3	Hyundai i20 Sportz Diesel	2010	225000	127000	23.00	1396	90.00	5.0	Hyundai	
4	Maruti Swift VXI BSIII	2007	130000	120000	16.10	1298	88.20	5.0	Maruti	

5 rows × 23 columns

In [65]:
```
corr = one_hot_encoded_data.corr()

%matplotlib inline
fig, ax = plt.subplots(figsize=(20, 15))
ax = sns.heatmap(corr, cmap="YlGnBu", linewidth = 0.5, annot=True)
plt.title("Correlation Matrix- 1", fontsize=16)

save_fig('Car prices Correlation matrix-1')

plt.show()
plt.tight_layout()
```

```
Figure saved
```



```
<Figure size 432x288 with 0 Axes>
```

In [66]:
```
# Let's check the correlation between our features!
corr["selling_price"].sort_values(ascending=False)
```

Out[66]:
```
selling_price                   1.000000
max_power                       0.691354
transmission_Automatic          0.463834
engine                          0.441895
year                            0.427717
seller_type_Dealer              0.257992
fuel_Diesel                     0.252788
owner_First Owner               0.230681
owner_Test Drive Car            0.202809
seats                           0.159964
seller_type_Trustmark Dealer    0.023384
fuel_CNG                       -0.034923
fuel_LPG                       -0.043705
owner_Fourth & Above Owner     -0.086160
mileage                        -0.108498
owner_Third Owner              -0.128838
owner_Second Owner             -0.157697
km_driven                      -0.161845
fuel_Petrol                    -0.241010
seller_type_Individual         -0.258331
transmission_Manual            -0.463834
Name: selling_price, dtype: float64
```

In [67]:
```
# Let's scale our numerical data first!
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()

one_hot_encoded_data[attr_num] = scaler.fit_transform(one_hot_encoded_data[attr_r
one_hot_encoded_data
```

Out[67]:

	name	year	selling_price	km_driven	mileage	engine	max_power	seats	company	f
0	Maruti Swift Dzire VDI	2014	0.042126	0.061640	0.436364	0.209396	0.112200	0.1	Maruti	
1	Skoda Rapid 1.5 TDI Ambition	2014	0.034102	0.050837	0.367879	0.293289	0.192593	0.1	Skoda	
2	Honda City 2017-2020 EXi	2006	0.012839	0.059310	0.263636	0.292953	0.123094	0.1	Honda	
3	Hyundai i20 Sportz Diesel	2010	0.019559	0.053803	0.424242	0.259060	0.155773	0.1	Hyundai	
4	Maruti Swift VXI BSIII	2007	0.010030	0.050837	0.215152	0.226174	0.150871	0.1	Maruti	
...	...	...	...	...	...	...	...	...	...	
6697	Hyundai i20 Magna 1.4 CRDi	2014	0.044634	0.033891	0.410303	0.259060	0.152315	0.1	Hyundai	
6698	Hyundai i20 Magna	2013	0.029087	0.046601	0.287879	0.192282	0.136302	0.1	Hyundai	
6699	Hyundai Verna CRDi SX	2007	0.010532	0.050414	0.236364	0.291611	0.210240	0.1	Hyundai	
6700	Maruti Swift Dzire ZDi	2009	0.035306	0.050837	0.312121	0.209396	0.111928	0.1	Maruti	
6701	Tata Indigo CR4	2013	0.026078	0.010591	0.441515	0.259060	0.101307	0.1	Tata	

6702 rows × 23 columns

In [68]:
```
# Drop the columns['fuel_Petrol', 'transmission_Manual', 'seller_type_Individual
#-by the other-same-variable columns! (also because of their negative correlatior
# for example:

one_hot_encoded_data = one_hot_encoded_data.drop(['fuel_Petrol', 'transmission_Ma
one_hot_encoded_data
```

Out[68]:

	name	year	selling_price	km_driven	mileage	engine	max_power	seats	company	f
0	Maruti Swift Dzire VDI	2014	0.042126	0.061640	0.436364	0.209396	0.112200	0.1	Maruti	
1	Skoda Rapid 1.5 TDI Ambition	2014	0.034102	0.050837	0.367879	0.293289	0.192593	0.1	Skoda	
2	Honda City 2017-2020 EXi	2006	0.012839	0.059310	0.263636	0.292953	0.123094	0.1	Honda	
3	Hyundai i20 Sportz Diesel	2010	0.019559	0.053803	0.424242	0.259060	0.155773	0.1	Hyundai	
4	Maruti Swift VXI BSIII	2007	0.010030	0.050837	0.215152	0.226174	0.150871	0.1	Maruti	
...	...	...	...	...	...	...	...	...	...	...
6697	Hyundai i20 Magna 1.4 CRDi	2014	0.044634	0.033891	0.410303	0.259060	0.152315	0.1	Hyundai	
6698	Hyundai i20 Magna	2013	0.029087	0.046601	0.287879	0.192282	0.136302	0.1	Hyundai	
6699	Hyundai Verna CRDi SX	2007	0.010532	0.050414	0.236364	0.291611	0.210240	0.1	Hyundai	
6700	Maruti Swift Dzire ZDi	2009	0.035306	0.050837	0.312121	0.209396	0.111928	0.1	Maruti	
6701	Tata Indigo CR4	2013	0.026078	0.010591	0.441515	0.259060	0.101307	0.1	Tata	

6702 rows × 19 columns

In [69]:
```
one_hot_encoded_data.columns
```

Out[69]:
```
Index(['name', 'year', 'selling_price', 'km_driven', 'mileage', 'engine',
       'max_power', 'seats', 'company', 'fuel_CNG', 'fuel_Diesel', 'fuel_LPG',
       'seller_type_Dealer', 'seller_type_Trustmark Dealer',
       'transmission_Automatic', 'owner_First Owner',
       'owner_Fourth & Above Owner', 'owner_Second Owner',
       'owner_Third Owner'],
      dtype='object')
```

In [70]:
```
# import libraries!
from sklearn.model_selection import train_test_split
import statsmodels.api as sm
import statsmodels.stats.outliers_influence as smo
from sklearn.linear_model import LinearRegression
```

In [71]:
```
# Let's start building our Linear Regression model!

indepent_var = ['year', 'km_driven', 'mileage', 'engine',
       'max_power', 'seats', 'fuel_CNG', 'fuel_Diesel', 'fuel_LPG',
       'seller_type_Dealer', 'seller_type_Trustmark Dealer',
       'transmission_Automatic', 'owner_First Owner',
       'owner_Fourth & Above Owner', 'owner_Second Owner',
       'owner_Third Owner']

X = one_hot_encoded_data[indepent_var]
y = one_hot_encoded_data["selling_price"]
```

In [72]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size = 0.8, randc
```

In [73]:
```
linear_reg = LinearRegression()
linear_reg.fit(X_train, y_train)
y_pred= linear_reg.predict(X_test)

print("Accuracy on Traing set: ",linear_reg.score(X_train,y_train))
print("Accuracy on Testing set: ",linear_reg.score(X_test,y_test))
```

```
Accuracy on Traing set:  0.6378248027748032
Accuracy on Testing set:  0.6404950240674857
```

In [74]:
```
X_train = sm.add_constant(X_train)

# Using Ordinary Least Square method!
lm = sm.OLS(y_train, X_train).fit()
print(lm.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:          selling_price   R-squared:                       0.638
Model:                            OLS   Adj. R-squared:                  0.637
Method:                 Least Squares   F-statistic:                     588.2
Date:                Mon, 19 Dec 2022   Prob (F-statistic):               0.00
Time:                        11:19:45   Log-Likelihood:                 11021.
No. Observations:                5361   AIC:                        -2.201e+04
Df Residuals:                    5344   BIC:                        -2.190e+04
Df Model:                          16
Covariance Type:            nonrobust
================================================================================
=================
                                   coef    std err          t      P>|t|
[0.025      0.975]
--------------------------------------------------------------------------------
-----------------
const                           -7.1311      0.313    -22.753      0.000      -
7.746      -6.517
year                             0.0037      0.000     23.446      0.000
0.003       0.004
km_driven                       -0.1380      0.019     -7.261      0.000      -
0.175      -0.101
mileage                         -0.0040      0.007     -0.601      0.548      -
0.017       0.009
engine                           0.0219      0.006      3.586      0.000
0.010       0.034
max_power                        0.2833      0.008     34.884      0.000
0.267       0.299
seats                           -0.0120      0.007     -1.800      0.072      -
0.025       0.001
fuel_CNG                         0.0041      0.005      0.825      0.409      -
0.006       0.014
fuel_Diesel                      0.0094      0.001      7.054      0.000
0.007       0.012
fuel_LPG                         0.0143      0.005      2.664      0.008
0.004       0.025
seller_type_Dealer               0.0111      0.002      7.403      0.000
0.008       0.014
seller_type_Trustmark Dealer    -0.0074      0.007     -1.130      0.258      -
0.020       0.005
transmission_Automatic           0.0308      0.002     17.495      0.000
0.027       0.034
owner_First Owner               -0.2274      0.016    -14.574      0.000      -
0.258      -0.197
owner_Fourth & Above Owner      -0.2338      0.016    -14.711      0.000      -
0.265      -0.203
owner_Second Owner              -0.2333      0.016    -14.915      0.000      -
0.264      -0.203
owner_Third Owner               -0.2337      0.016    -14.865      0.000      -
0.265      -0.203
```

```
==============================================================================
Omnibus:                     5358.296   Durbin-Watson:                   1.964
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           584780.336
Skew:                           4.639   Prob(JB):                         0.00
Kurtosis:                      53.318   Cond. No.                     1.49e+06
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctl
y specified.
[2] The condition number is large, 1.49e+06. This might indicate that there are
strong multicollinearity or other numerical problems.
```

In [75]:
```
# Ok, so our data may have multicollinearity, which we need to remove!
# Also, notice a lot of our features have huge p values! We need to drop them too
```

In [76]:
```
# Let's check out the VIF for these features!
vif_info = pd.DataFrame()
vif_info["VIF"] = [smo.variance_inflation_factor(X[indepent_var].values, ix) for
vif_info["Columns"] = X.columns
vif_info
```

Out[76]:

	VIF	Columns
0	1409.589896	year
1	3.238715	km_driven
2	22.182420	mileage
3	20.889899	engine
4	10.302283	max_power
5	6.808721	seats
6	1.051357	fuel_CNG
7	5.285502	fuel_Diesel
8	1.016232	fuel_LPG
9	1.219978	seller_type_Dealer
10	1.014698	seller_type_Trustmark Dealer
11	1.462638	transmission_Automatic
12	840.155401	owner_First Owner
13	32.017423	owner_Fourth & Above Owner
14	380.912193	owner_Second Owner
15	100.161477	owner_Third Owner

In [77]:
```
# Let's drop the features with VIF > 10
# And create a new model with the new gathered data!

indepent_var = ['km_driven', 'max_power', 'fuel_CNG', 'fuel_Diesel', 'fuel_LPG',
       'seller_type_Dealer', 'seller_type_Trustmark Dealer',
       'transmission_Automatic']

X = one_hot_encoded_data[indepent_var]
y = one_hot_encoded_data["selling_price"]
```

In [78]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size = 0.8, randc
```

In [79]:
```
linear_reg = LinearRegression()
linear_reg.fit(X_train, y_train)
y_pred= linear_reg.predict(X_test)

print("Accuracy on Traing set: ",linear_reg.score(X_train, y_train))
print("Accuracy on Testing set: ",linear_reg.score(X_test, y_test))
```

```
Accuracy on Traing set:  0.5510337559643956
Accuracy on Testing set:  0.573190862852567
```

In [80]:
```
X_train = sm.add_constant(X_train)

lm = sm.OLS(y_train, X_train).fit()
print(lm.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:          selling_price   R-squared:                       0.551
Model:                            OLS   Adj. R-squared:                  0.550
Method:                 Least Squares   F-statistic:                     821.1
Date:                Mon, 19 Dec 2022   Prob (F-statistic):               0.00
Time:                        11:19:46   Log-Likelihood:                 10445.
No. Observations:                5361   AIC:                        -2.087e+04
Df Residuals:                    5352   BIC:                        -2.081e+04
Df Model:                           8
Covariance Type:            nonrobust
================================================================================
=================
                                   coef    std err          t      P>|t|
[0.025      0.975]
--------------------------------------------------------------------------------
-----------------
const                          1.577e-05      0.001      0.014      0.989      -
0.002       0.002
km_driven                        -0.3719      0.019    -19.348      0.000      -
0.410      -0.334
max_power                         0.3272      0.007     49.483      0.000
0.314       0.340
fuel_CNG                          0.0121      0.005      2.218      0.027
0.001       0.023
fuel_Diesel                       0.0139      0.001     13.040      0.000
0.012       0.016
fuel_LPG                          0.0080      0.006      1.332      0.183      -
0.004       0.020
seller_type_Dealer                0.0168      0.002     10.197      0.000
0.014       0.020
seller_type_Trustmark Dealer      0.0007      0.007      0.101      0.920      -
0.013       0.015
transmission_Automatic            0.0342      0.002     17.574      0.000
0.030       0.038
==============================================================================
Omnibus:                     4867.571   Durbin-Watson:                   1.981
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           420078.059
Skew:                           4.006   Prob(JB):                         0.00
Kurtosis:                      45.619   Cond. No.                         48.3
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctl
y specified.
```

In [81]:
```
# We have remove multicollinearity!
# Although, our R^2 value has decreased, the F-stats value has increased signific
# Let's continue and drop more features with high p-values!
```

In [82]:
```
vif_info = pd.DataFrame()
vif_info["VIF"] = [smo.variance_inflation_factor(X[indepent_var].values, ix) for
vif_info["Columns"] = X.columns
vif_info
```

Out[82]:

	VIF	Columns
0	2.324454	km_driven
1	3.684230	max_power
2	1.008956	fuel_CNG
3	2.760812	fuel_Diesel
4	1.009464	fuel_LPG
5	1.181234	seller_type_Dealer
6	1.010696	seller_type_Trustmark Dealer
7	1.419754	transmission_Automatic

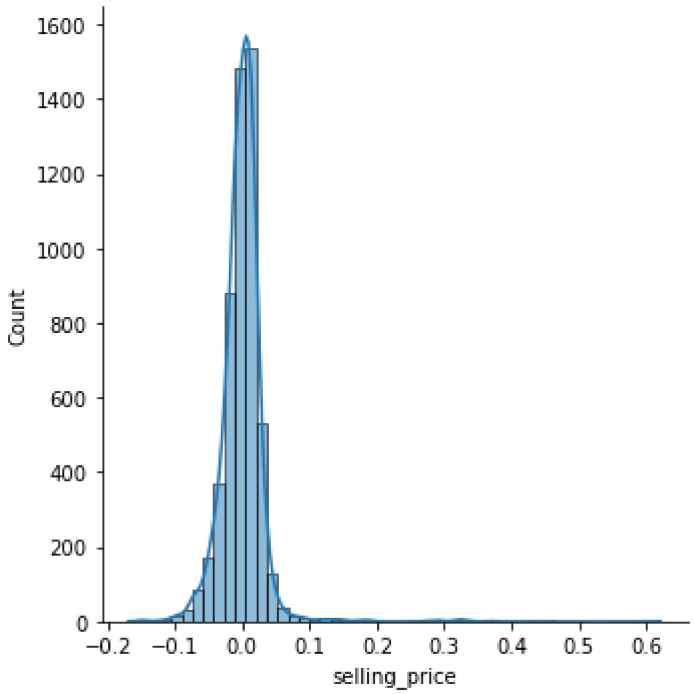
In [83]:
```
indepent_var = ['km_driven', 'max_power', 'fuel_Diesel',
       'seller_type_Dealer', 'transmission_Automatic']

X = one_hot_encoded_data[indepent_var]
y = one_hot_encoded_data["selling_price"]
```

In [84]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size = 0.8, rando
```

In [85]:
```
linear_reg = LinearRegression()
linear_reg.fit(X_train, y_train)
y_pred = linear_reg.predict(X_test)

print("Accuracy on Traing set: ",linear_reg.score(X_train, y_train))
print("Accuracy on Testing set: ",linear_reg.score(X_test, y_test))
```

```
Accuracy on Traing set:  0.561694856397433
Accuracy on Testing set:  0.5273936147758962
```

In [86]:
```
y_pred = linear_reg.predict(X_train)

%matplotlib inline
sns.displot((y_train - y_pred), bins=50, kde=True)

plt.show()
plt.tight_layout()
```




```
<Figure size 432x288 with 0 Axes>
```

In [87]:
```
# Our error plot is almost normaly distributed i.e model can predict car prices w
# Our model can be deployed! But, let's see what we can do further!
```

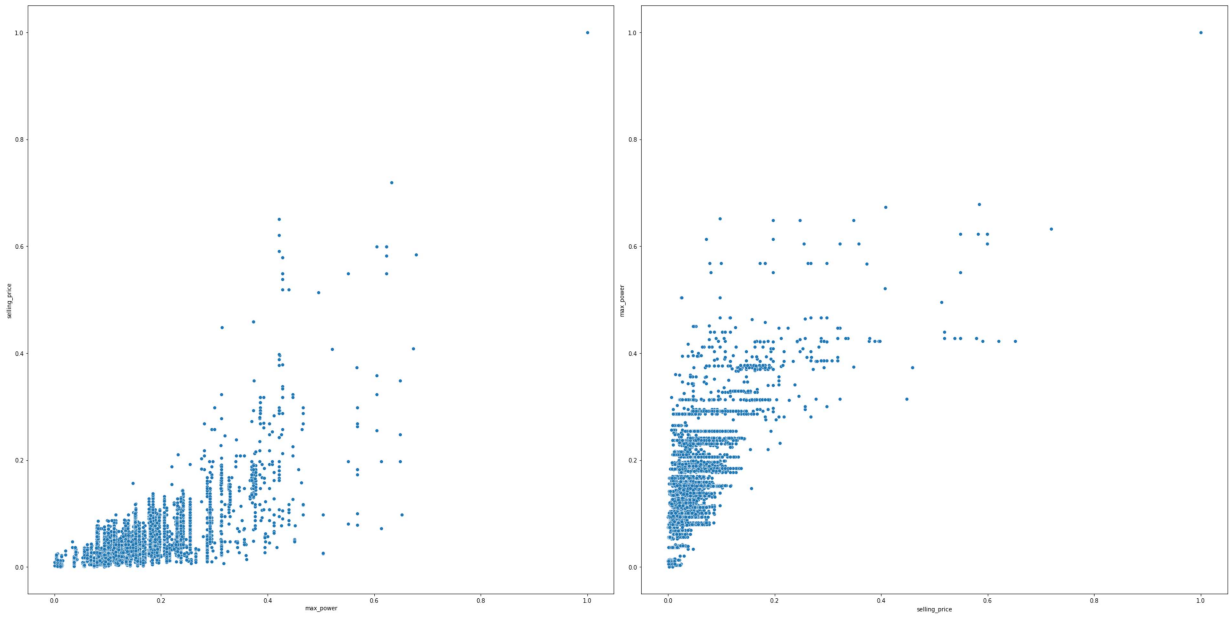
In [88]:
```
X_train = sm.add_constant(X_train)

lm = sm.OLS(y_train, X_train).fit()
print(lm.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:          selling_price   R-squared:                       0.562
Model:                            OLS   Adj. R-squared:                  0.561
Method:                 Least Squares   F-statistic:                     1373.
Date:                Mon, 19 Dec 2022   Prob (F-statistic):               0.00
Time:                        11:19:46   Log-Likelihood:                 10348.
No. Observations:                5361   AIC:                        -2.068e+04
Df Residuals:                    5355   BIC:                        -2.064e+04
Df Model:                           5
Covariance Type:            nonrobust
===========================================================================
==========
                              coef    std err          t      P>|t|      [0.025
0.975]
---------------------------------------------------------------------------
----------
const                      -0.0023      0.001     -2.026      0.043      -0.004
-7.37e-05
km_driven                  -0.3592      0.020    -18.255      0.000      -0.398
-0.321
max_power                   0.3500      0.007     52.107      0.000       0.337
0.363
fuel_Diesel                 0.0112      0.001     10.503      0.000       0.009
0.013
seller_type_Dealer          0.0167      0.002     10.032      0.000       0.013
0.020
transmission_Automatic      0.0351      0.002     17.799      0.000       0.031
0.039
==============================================================================
Omnibus:                     5245.605   Durbin-Watson:                   1.984
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           638040.622
Skew:                           4.423   Prob(JB):                         0.00
Kurtosis:                      55.708   Cond. No.                         48.6
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctl
y specified.
```

In [89]:
```
# R^2 of 56% in Train and Test data set meaning our model has fitted data well! A
# This may be beacuse of the noise and outliers present in our dataset!
```

In [90]:
```
figure, ax = plt.subplots(1,2, figsize=(30, 15))

sns.scatterplot(x='max_power', y="selling_price", ax=ax[0], data = one_hot_encode
sns.scatterplot(x='selling_price', y="max_power", ax=ax[1], data = one_hot_encode

save_fig('Scatter plot- Price vs power')

plt.show()
plt.tight_layout()
```

```
Figure saved
```




```
<Figure size 432x288 with 0 Axes>
```

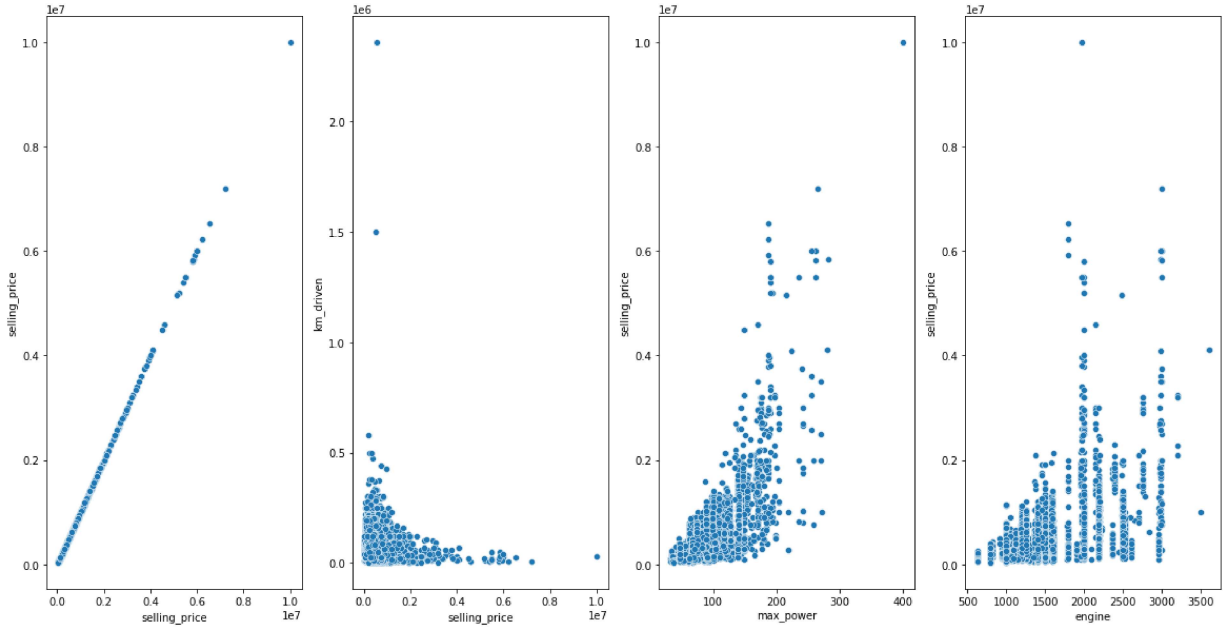
In [91]:
```
# Let's build the above linear model again after removing outliers
df_preprocessed = df_clean.copy()
df_preprocessed
```

Out[91]:

	name	year	selling_price	km_driven	fuel	seller_type	transmission	owner	mileage
0	Maruti Swift Dzire VDI	2014	450000	145500	Diesel	Individual	Manual	First Owner	23.40
1	Skoda Rapid 1.5 TDI Ambition	2014	370000	120000	Diesel	Individual	Manual	Second Owner	21.14
2	Honda City 2017-2020 EXi	2006	158000	140000	Petrol	Individual	Manual	Third Owner	17.70
3	Hyundai i20 Sportz Diesel	2010	225000	127000	Diesel	Individual	Manual	First Owner	23.00
4	Maruti Swift VXI BSIII	2007	130000	120000	Petrol	Individual	Manual	First Owner	16.10
...	...	...	...	...	...	...	...	...	...
6697	Hyundai i20 Magna 1.4 CRDi	2014	475000	80000	Diesel	Individual	Manual	Second Owner	22.54
6698	Hyundai i20 Magna	2013	320000	110000	Petrol	Individual	Manual	First Owner	18.50
6699	Hyundai Verna CRDi SX	2007	135000	119000	Diesel	Individual	Manual	Fourth & Above Owner	16.80
6700	Maruti Swift Dzire ZDi	2009	382000	120000	Diesel	Individual	Manual	First Owner	19.30
6701	Tata Indigo CR4	2013	290000	25000	Diesel	Individual	Manual	First Owner	23.57

6702 rows × 13 columns

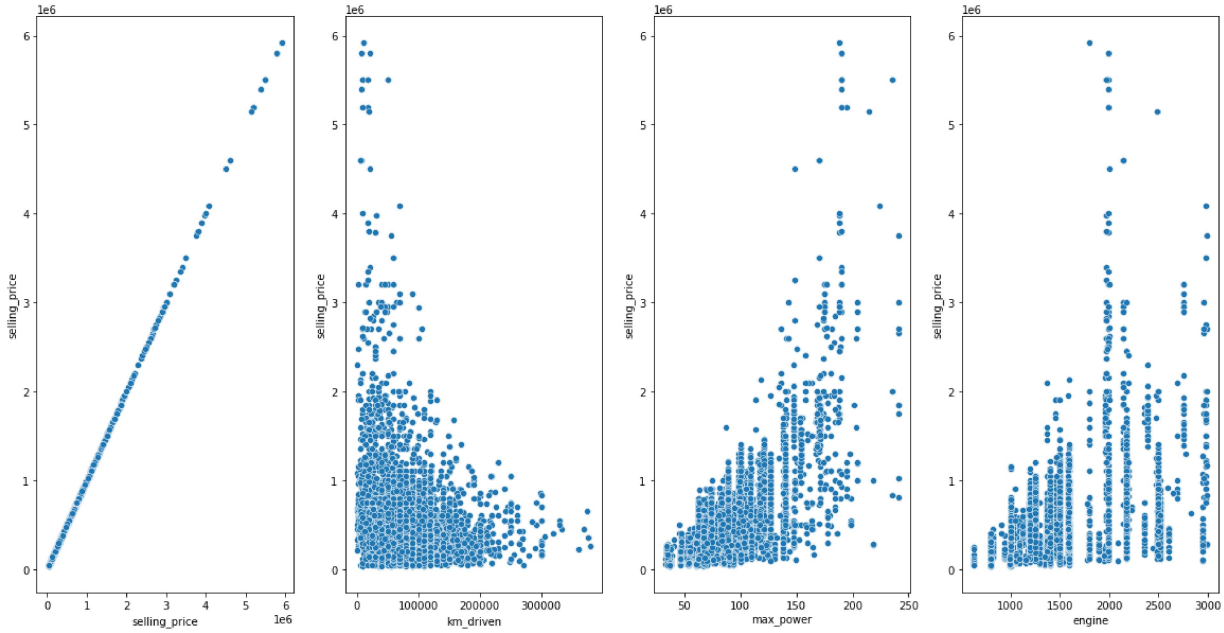
In [92]:
```
figure, ax = plt.subplots(1,4, figsize=(20, 10))

sns.scatterplot(x='selling_price', y='selling_price', ax=ax[0], data = df_preproc
sns.scatterplot(x='selling_price', y="km_driven", ax=ax[1], data = df_preprocesse
sns.scatterplot(x='max_power', y='selling_price', ax=ax[2], data = df_preprocesse
sns.scatterplot(x='engine', y='selling_price', ax=ax[3], data = df_preprocessed)

plt.show()
plt.tight_layout()
```



<Figure size 432x288 with 0 Axes>

In [93]:
```
df_preprocessed = df_preprocessed[df_preprocessed['selling_price'] < 6000000] # M
df_preprocessed = df_preprocessed[df_preprocessed['km_driven'] < 400000] # Models
df_preprocessed = df_preprocessed[df_preprocessed['max_power'] < 250] # Models wi
df_preprocessed = df_preprocessed[df_preprocessed['engine'] < 3000] # Models with
```

In [94]:
```
figure, ax = plt.subplots(1,4, figsize=(20, 10))

sns.scatterplot(x='selling_price', y='selling_price', ax=ax[0], data = df_preproc
sns.scatterplot(x='km_driven', y="selling_price", ax=ax[1], data = df_preprocesse
sns.scatterplot(x='max_power', y='selling_price', ax=ax[2], data = df_preprocesse
sns.scatterplot(x='engine', y='selling_price', ax=ax[3], data = df_preprocessed)

plt.show()
plt.tight_layout()
```




```
<Figure size 432x288 with 0 Axes>
```

In [95]:
```
df_preprocessed = df_preprocessed.reset_index(drop=True)
df_preprocessed.shape
```

Out[95]:
```
(6670, 13)
```

In [96]:
```
df_clean.shape
```

Out[96]:
```
(6702, 13)
```

In [97]:
```
one_hot_encoded_data = pd.get_dummies(df_preprocessed, columns = ['fuel', 'seller
print(one_hot_encoded_data.columns, '\n')
one_hot_encoded_data
```

```
Index(['name', 'year', 'selling_price', 'km_driven', 'mileage', 'engine',
       'max_power', 'seats', 'company', 'fuel_CNG', 'fuel_Diesel', 'fuel_LPG',
       'fuel_Petrol', 'seller_type_Dealer', 'seller_type_Individual',
       'seller_type_Trustmark Dealer', 'transmission_Automatic',
       'transmission_Manual', 'owner_First Owner',
       'owner_Fourth & Above Owner', 'owner_Second Owner',
       'owner_Test Drive Car', 'owner_Third Owner'],
      dtype='object')
```

Out[97]:

	name	year	selling_price	km_driven	mileage	engine	max_power	seats	company	f
0	Maruti Swift Dzire VDI	2014	450000	145500	23.40	1248	74.00	5.0	Maruti	
1	Skoda Rapid 1.5 TDI Ambition	2014	370000	120000	21.14	1498	103.52	5.0	Skoda	
2	Honda City 2017-2020 EXi	2006	158000	140000	17.70	1497	78.00	5.0	Honda	
3	Hyundai i20 Sportz Diesel	2010	225000	127000	23.00	1396	90.00	5.0	Hyundai	
4	Maruti Swift VXI BSIII	2007	130000	120000	16.10	1298	88.20	5.0	Maruti	
...	...	...	...	...	...	...	...	...	...	
6665	Hyundai i20 Magna 1.4 CRDi	2014	475000	80000	22.54	1396	88.73	5.0	Hyundai	
6666	Hyundai i20 Magna	2013	320000	110000	18.50	1197	82.85	5.0	Hyundai	
6667	Hyundai Verna CRDi SX	2007	135000	119000	16.80	1493	110.00	5.0	Hyundai	
6668	Maruti Swift Dzire ZDi	2009	382000	120000	19.30	1248	73.90	5.0	Maruti	

	name	year	selling_price	km_driven	mileage	engine	max_power	seats	company	f
6669	Tata Indigo CR4	2013	290000	25000	23.57	1396	70.00	5.0	Tata	

6670 rows × 23 columns

In [98]:
```
one_hot_encoded_data.head()
```

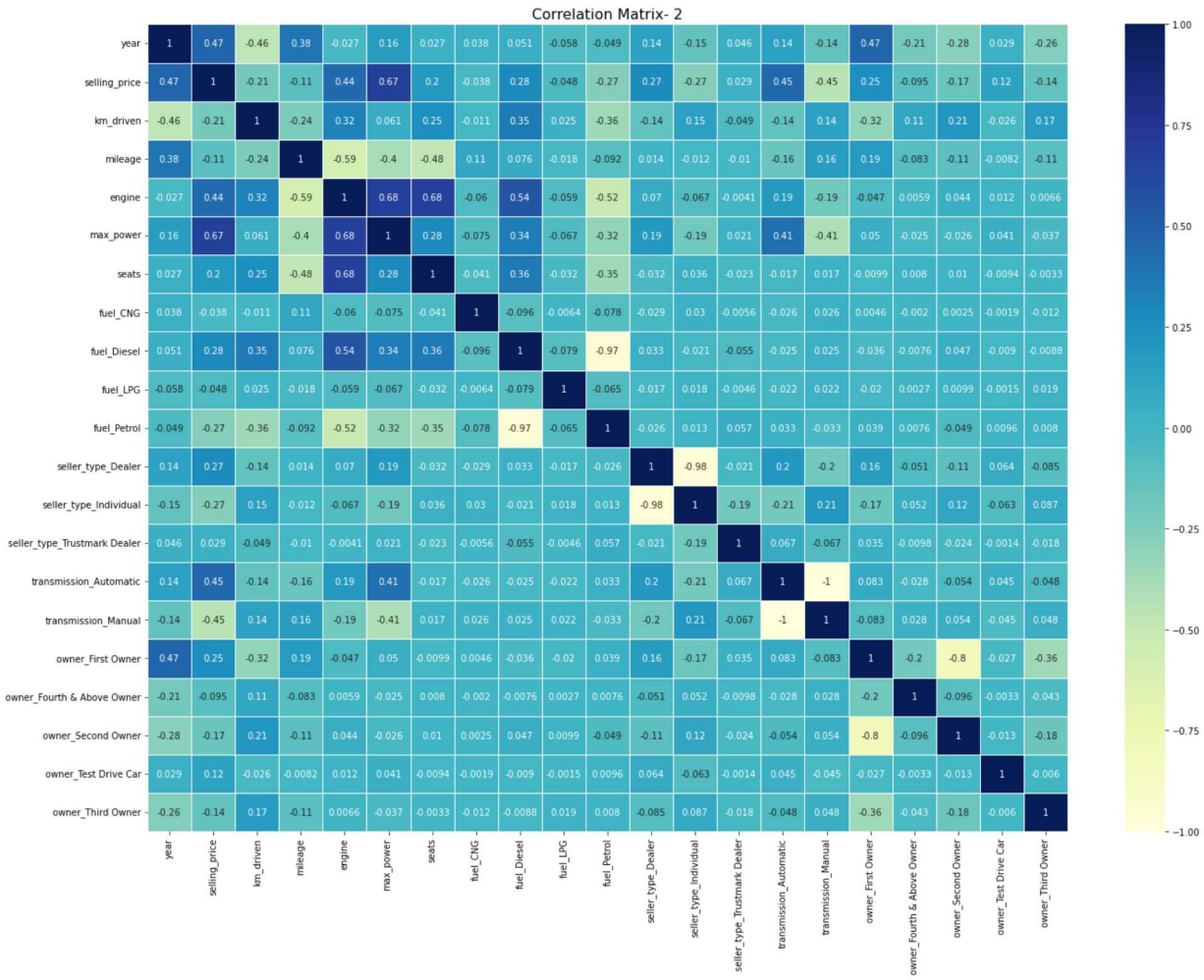
Out[98]:

	name	year	selling_price	km_driven	mileage	engine	max_power	seats	company	fuel_CN
0	Maruti Swift Dzire VDI	2014	450000	145500	23.40	1248	74.00	5.0	Maruti	
1	Skoda Rapid 1.5 TDI Ambition	2014	370000	120000	21.14	1498	103.52	5.0	Skoda	
2	Honda City 2017-2020 EXi	2006	158000	140000	17.70	1497	78.00	5.0	Honda	
3	Hyundai i20 Sportz Diesel	2010	225000	127000	23.00	1396	90.00	5.0	Hyundai	
4	Maruti Swift VXI BSIII	2007	130000	120000	16.10	1298	88.20	5.0	Maruti	

5 rows × 23 columns

In [99]:
```
corr = one_hot_encoded_data.corr()

fig, ax = plt.subplots(figsize=(20, 15))
ax = sns.heatmap(corr, cmap="YlGnBu", linewidth = 0.5, annot=True)
plt.title("Correlation Matrix- 2", fontsize=16)

save_fig('Car price prediction correlation Matrix- 2')

plt.show()
plt.tight_layout()
```

```
Figure saved
```



```
<Figure size 432x288 with 0 Axes>
```

In [100]:
```
corr["selling_price"].sort_values(ascending=False)
```

Out[100]:
```
selling_price                  1.000000
max_power                      0.670445
year                           0.466586
transmission_Automatic         0.446450
engine                         0.436963
fuel_Diesel                    0.281074
seller_type_Dealer             0.268663
owner_First Owner              0.253435
seats                          0.195299
owner_Test Drive Car           0.120128
seller_type_Trustmark Dealer   0.028753
fuel_CNG                      -0.037619
fuel_LPG                      -0.048162
owner_Fourth & Above Owner    -0.094643
mileage                       -0.107849
owner_Third Owner             -0.140758
owner_Second Owner            -0.165723
km_driven                     -0.206495
fuel_Petrol                   -0.268214
seller_type_Individual        -0.269910
transmission_Manual           -0.446450
Name: selling_price, dtype: float64
```

In [101]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler() #Using standar scaler this time!

one_hot_encoded_data[attr_num] = scaler.fit_transform(one_hot_encoded_data[attr_r
one_hot_encoded_data
```

Out[101]:

	name	year	selling_price	km_driven	mileage	engine	max_power	seats	compa
0	Maruti Swift Dzire VDI	2014	-0.137899	1.548896	0.985605	-0.364913	-0.437711	-0.443831	Mar
1	Skoda Rapid 1.5 TDI Ambition	2014	-0.313619	1.007008	0.409700	0.151186	0.543945	-0.443831	Sko
2	Honda City 2017-2020 EXi	2006	-0.779276	1.432018	-0.466897	0.149122	-0.304695	-0.443831	Hon
3	Hyundai i20 Sportz Diesel	2010	-0.632111	1.155762	0.883675	-0.059382	0.094352	-0.443831	Hyun
4	Maruti Swift VXI BSIII	2007	-0.840778	1.007008	-0.874617	-0.261693	0.034495	-0.443831	Mar
...	...	...	...	...	...	...	...	...	
6665	Hyundai i20 Magna 1.4 CRDi	2014	-0.082986	0.156987	0.766455	-0.059382	0.052119	-0.443831	Hyun
6666	Hyundai i20 Magna	2013	-0.423444	0.794503	-0.263037	-0.470198	-0.143414	-0.443831	Hyun
6667	Hyundai Verna CRDi SX	2007	-0.829796	0.985757	-0.696240	0.140864	0.759430	-0.443831	Hyun
6668	Maruti Swift Dzire ZDi	2009	-0.287261	1.007008	-0.059177	-0.364913	-0.441036	-0.443831	Mar
6669	Tata Indigo CR4	2013	-0.489339	-1.011792	1.028925	-0.059382	-0.570726	-0.443831	Ta

6670 rows × 23 columns

In [102]:
```
corr = one_hot_encoded_data.corr()
corr["selling_price"].sort_values(ascending=False)
```

Out[102]:
```
selling_price                  1.000000
max_power                      0.670445
year                           0.466586
transmission_Automatic         0.446450
engine                         0.436963
fuel_Diesel                    0.281074
seller_type_Dealer             0.268663
owner_First Owner              0.253435
seats                          0.195299
owner_Test Drive Car           0.120128
seller_type_Trustmark Dealer   0.028753
fuel_CNG                      -0.037619
fuel_LPG                      -0.048162
owner_Fourth & Above Owner    -0.094643
mileage                       -0.107849
owner_Third Owner             -0.140758
owner_Second Owner            -0.165723
km_driven                     -0.206495
fuel_Petrol                   -0.268214
seller_type_Individual        -0.269910
transmission_Manual           -0.446450
Name: selling_price, dtype: float64
```

In [103]:
```
indepent_var = ['year', 'km_driven', 'mileage', 'engine',
       'max_power', 'seats', 'fuel_CNG', 'fuel_Diesel', 'fuel_LPG',
       'seller_type_Dealer', 'seller_type_Trustmark Dealer',
       'transmission_Automatic', 'owner_First Owner',
       'owner_Fourth & Above Owner', 'owner_Second Owner',
       'owner_Third Owner']

X = one_hot_encoded_data[indepent_var]
y = one_hot_encoded_data["selling_price"]
```

In [104]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size = 0.8, rando
```

In [105]:
```
linear_reg = LinearRegression()
linear_reg.fit(X_train, y_train)
y_pred= linear_reg.predict(X_test)

print("Accuracy on Traing set: ", linear_reg.score(X_train,y_train))
print("Accuracy on Testing set: ", linear_reg.score(X_test,y_test))
```

```
Accuracy on Traing set:  0.6356803930062693
Accuracy on Testing set:  0.653328041161366
```

In [106]:
```
X_train = sm.add_constant(X_train)

lm = sm.OLS(y_train, X_train).fit()
print(lm.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:          selling_price   R-squared:                       0.636
Model:                            OLS   Adj. R-squared:                  0.635
Method:                 Least Squares   F-statistic:                     580.1
Date:                Mon, 19 Dec 2022   Prob (F-statistic):               0.00
Time:                        11:19:53   Log-Likelihood:                -4801.6
No. Observations:                5336   AIC:                             9637.
Df Residuals:                    5319   BIC:                             9749.
Df Model:                          16
Covariance Type:            nonrobust
================================================================================
=================
                                   coef    std err          t      P>|t|
[0.025      0.975]
--------------------------------------------------------------------------------
-----------------
const                          -154.7014      6.255    -24.731      0.000    -16
6.964    -142.439
year                              0.0774      0.003     25.062      0.000
0.071       0.083
km_driven                        -0.1254      0.011    -11.719      0.000      -
0.146      -0.104
mileage                          -0.0355      0.015     -2.313      0.021      -
0.066      -0.005
engine                            0.0559      0.019      2.936      0.003
0.019       0.093
max_power                         0.4384      0.013     33.562      0.000
0.413       0.464
seats                            -0.0048      0.012     -0.387      0.699      -
0.029       0.020
fuel_CNG                          0.1068      0.098      1.091      0.275      -
0.085       0.299
fuel_Diesel                       0.2929      0.026     11.187      0.000
0.242       0.344
fuel_LPG                          0.3139      0.112      2.807      0.005
0.095       0.533
seller_type_Dealer                0.2203      0.029      7.604      0.000
0.163       0.277
seller_type_Trustmark Dealer     -0.0950      0.128     -0.742      0.458      -
0.346       0.156
transmission_Automatic            0.6121      0.034     18.117      0.000
0.546       0.678
owner_First Owner                -1.2644      0.597     -2.117      0.034      -
2.435      -0.094
owner_Fourth & Above Owner       -1.2796      0.600     -2.132      0.033      -
2.456      -0.103
owner_Second Owner               -1.3598      0.598     -2.275      0.023      -
2.531      -0.188
owner_Third Owner                -1.3352      0.598     -2.232      0.026      -
2.508      -0.162
==============================================================================
```

```
Omnibus:                     5465.321   Durbin-Watson:                   1.894
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           661351.457
Skew:                           4.803   Prob(JB):                         0.00
Kurtosis:                      56.687   Cond. No.                     1.54e+06
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctl
y specified.
[2] The condition number is large, 1.54e+06. This might indicate that there are
strong multicollinearity or other numerical problems.
```

In [107]:
```
vif_info = pd.DataFrame()
vif_info["VIF"] = [smo.variance_inflation_factor(X[indepent_var].values, ix) for
vif_info["Columns"] = X.columns
vif_info
```

Out[107]:

	VIF	Columns
0	2226.060692	year
1	1.463617	km_driven
2	2.791363	mileage
3	5.551435	engine
4	2.454375	max_power
5	2.191094	seats
6	1.053246	fuel_CNG
7	5.580628	fuel_Diesel
8	1.017534	fuel_LPG
9	1.221422	seller_type_Dealer
10	1.015358	seller_type_Trustmark Dealer
11	1.411154	transmission_Automatic
12	1383.211704	owner_First Owner
13	52.292864	owner_Fourth & Above Owner
14	626.695141	owner_Second Owner
15	164.328435	owner_Third Owner

In [108]:
```
indepent_var = ['km_driven', 'mileage', 'engine',
       'max_power', 'seats', 'fuel_Diesel',
       'seller_type_Dealer',
       'transmission_Automatic']

X = one_hot_encoded_data[indepent_var]
y = one_hot_encoded_data["selling_price"]
```

In [109]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size = 0.8, rando

linear_reg = LinearRegression()
linear_reg.fit(X_train, y_train)
y_pred= linear_reg.predict(X_test)

print("Accuracy on Traing set:  ",linear_reg.score(X_train, y_train))
print("Accuracy on Testing set: ",linear_reg.score(X_test, y_test))
```

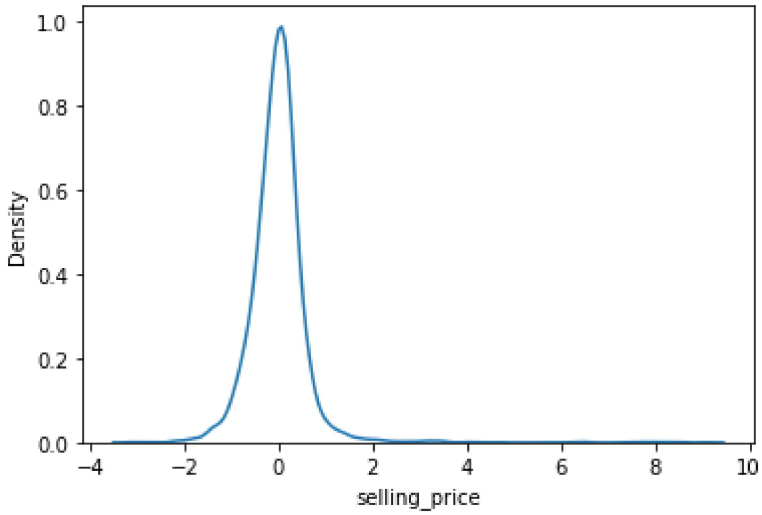
```
Accuracy on Traing set:   0.5836766006085661
Accuracy on Testing set:  0.5966531678811873
```

In [110]:
```
lm = sm.OLS(y_train, X_train).fit()
print(lm.summary())
```

```
                                 OLS Regression Results
================================================================================
========
Dep. Variable:          selling_price   R-squared (uncentered):
0.576
Model:                            OLS   Adj. R-squared (uncentered):
0.576
Method:                 Least Squares   F-statistic:
906.2
Date:                Mon, 19 Dec 2022   Prob (F-statistic):
0.00
Time:                        11:19:53   Log-Likelihood:
-5351.9
No. Observations:                5336   AIC:
1.072e+04
Df Residuals:                    5328   BIC:
1.077e+04
Df Model:                           8
Covariance Type:            nonrobust
================================================================================
===========
                              coef    std err          t      P>|t|      [0.025
0.975]
--------------------------------------------------------------------------------
-----------
km_driven                  -0.2260      0.010    -21.718      0.000      -0.246
-0.206
mileage                     0.2424      0.012     19.471      0.000       0.218
0.267
engine                      0.1817      0.019      9.523      0.000       0.144
0.219
max_power                   0.5612      0.014     40.421      0.000       0.534
0.588
seats                       0.1007      0.013      7.619      0.000       0.075
0.127
fuel_Diesel                -0.0576      0.015     -3.809      0.000      -0.087
-0.028
seller_type_Dealer          0.2417      0.031      7.735      0.000       0.180
0.303
transmission_Automatic      0.6040      0.036     16.629      0.000       0.533
0.675
==============================================================================
Omnibus:                     4953.736   Durbin-Watson:                   2.003
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           491577.103
Skew:                           4.108   Prob(JB):                         0.00
Kurtosis:                      49.298   Cond. No.                         6.90
==============================================================================

Notes:
[1] R² is computed without centering (uncentered) since the model does not cont
ain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctl
y specified.
```

In [111]:
```
y_pred = linear_reg.predict(X_train)
sns.kdeplot((y_train - y_pred))
```

Out[111]: <AxesSubplot:xlabel='selling_price', ylabel='Density'>



In [112]:
```
indepent_var = ['km_driven', 'max_power', 'fuel_Diesel',
       'seller_type_Dealer', 'transmission_Automatic']

X = one_hot_encoded_data[indepent_var]
y = one_hot_encoded_data["selling_price"]
```

In [113]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size = 0.8, randc

linear_reg = LinearRegression()
linear_reg.fit(X_train, y_train)
y_pred = linear_reg.predict(X_test)

print("Accuracy on Traing set:  ",linear_reg.score(X_train, y_train))
print("Accuracy on Testing set: ",linear_reg.score(X_test, y_test))
```

```
Accuracy on Traing set:   0.5753778476125024
Accuracy on Testing set:  0.554120545663904
```

In [114]:
```
X_train = sm.add_constant(X_train)

lm = sm.OLS(y_train, X_train).fit()
print(lm.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:          selling_price   R-squared:                       0.575
Model:                            OLS   Adj. R-squared:                  0.575
Method:                 Least Squares   F-statistic:                     1444.
Date:                Mon, 19 Dec 2022   Prob (F-statistic):               0.00
Time:                        11:19:53   Log-Likelihood:                -5291.9
No. Observations:                5336   AIC:                         1.060e+04
Df Residuals:                    5330   BIC:                         1.064e+04
Df Model:                           5
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         -0.3023      0.015    -20.076      0.000      -0.332      -0.273
km_driven     -0.2759      0.010    -27.885      0.000      -0.295      -0.256
max_power      0.5239      0.011     48.913      0.000       0.503       0.545
fuel_Diesel    0.4069      0.021     19.728      0.000       0.366       0.447
seller_type_Dealer      0.2940      0.031      9.479      0.000       0.233       0.355
transmission_Automatic     0.6773      0.037     18.192      0.000       0.604       0.750
==============================================================================
Omnibus:                     4544.270   Durbin-Watson:                   2.047
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           364838.951
Skew:                           3.618   Prob(JB):                         0.00
Kurtosis:                      42.857   Cond. No.                         5.14
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

In [115]:
```
from sklearn.metrics import mean_squared_error, mean_absolute_error

indepent_var = ['km_driven', 'max_power', 'fuel_Diesel',
       'seller_type_Dealer', 'transmission_Automatic']

X = one_hot_encoded_data[indepent_var]
y = one_hot_encoded_data["selling_price"]

X_train, X_test, y_train, y_test = train_test_split(X, y, train_size = 0.8, randc

lm = sm.OLS(y_train, X_train).fit()

y_train_pred = lm.predict(X_train)
y_test_pred = lm.predict(X_test)

print("Train MAE:  ", mean_absolute_error(y_train, y_train_pred))
print("Train RMSE: ", np.sqrt(mean_absolute_error(y_train, y_train_pred)), '\n')

print("Test MAE:   ", mean_absolute_error(y_test, y_test_pred))
print("Test RMSE:  ", np.sqrt(mean_absolute_error(y_test, y_test_pred)))
```

```
Train MAE:   0.4204538338667191
Train RMSE:  0.6484241157350018

Test MAE:    0.4366808487281159
Test RMSE:   0.6608183174883365
```

In [116]:
```
from sklearn.linear_model import Ridge

# Using Ridge regression to improve our accuracy!

lr = Ridge(alpha=0.001)

indepent_var = ['year', 'km_driven', 'mileage', 'engine',
       'max_power', 'seats', 'fuel_CNG', 'fuel_Diesel', 'fuel_LPG',
       'seller_type_Dealer', 'seller_type_Trustmark Dealer',
       'transmission_Automatic', 'owner_First Owner',
       'owner_Fourth & Above Owner', 'owner_Second Owner',
       'owner_Third Owner']

X = one_hot_encoded_data[indepent_var]
y = one_hot_encoded_data["selling_price"]

X_train, X_test, y_train, y_test = train_test_split(X, y, train_size = 0.8, randc

lr.fit(X_train, y_train)

y_train_pred = lr.predict(X_train)
y_test_pred = lr.predict(X_test)
```

In [117]:
```
print("-----------Ridge-----------")
print("Train MAE:    ", mean_absolute_error(y_train, y_train_pred))
print("Train RMSE:   ", np.sqrt(mean_absolute_error(y_train, y_train_pred)))

print("Test MAE:     ", mean_absolute_error(y_test, y_test_pred))
print("Test RMSE:    ", np.sqrt(mean_absolute_error(y_test, y_test_pred)))

print("\nRidge co-eff: ", lr.coef_)
```

```
-----------Ridge-----------
Train MAE:     0.3414554188734116
Train RMSE:    0.5843418681503249
Test MAE:      0.3543623449522238
Test RMSE:     0.595283415653606

Ridge co-eff:  [ 0.07196522 -0.12507216 -0.0234349   0.06608283  0.44191638 -0.
0062302
  0.10430374  0.276384    0.27391453  0.2402174  -0.01023961  0.65392998
 -3.63463468 -3.729999   -3.7336606  -3.72224959]
```

In [118]:
```
# No significant accuracy increase!
```

In [119]:
```
from sklearn.model_selection import RandomizedSearchCV
```

In [120]:
```
from sklearn.linear_model import Lasso
# Using Lasso regression to improve our accuracy!

lr = Lasso(alpha=0.001)

lr.fit(X_train, y_train)

y_train_pred = lr.predict(X_train)
y_test_pred = lr.predict(X_test)
```

In [121]:
```
print("-----------Lasso-----------")
print("Train MAE:  ", mean_absolute_error(y_train, y_train_pred))
print("Train RMSE: ", np.sqrt(mean_absolute_error(y_train, y_train_pred)))

print("Test MAE:   ", mean_absolute_error(y_test, y_test_pred))
print("Test RMSE:  ", np.sqrt(mean_absolute_error(y_test, y_test_pred)))

print("\nLasso co-eff: ", lr.coef_)
```

```
-----------Lasso-----------
Train MAE:   0.3421765621597634
Train RMSE:  0.5849585986715328
Test MAE:    0.3541745177139357
Test RMSE:   0.5951256318744268

Lasso co-eff:  [ 0.07315582 -0.12354116 -0.01530178  0.07000975  0.44661629 -0.
00348994
  0.          0.25341892  0.08932491  0.25038661 -0.          0.64798963
  0.         -0.03070646 -0.08653632 -0.0610363 ]
```

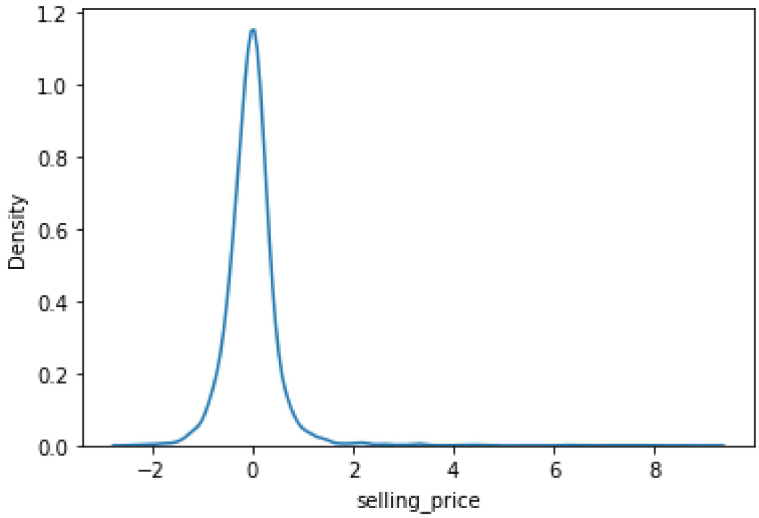
In [122]:
```
# No significant accuracy increase either!
```

In [123]:
```
# Lasso regression with Randomized Saerch!

ls = Lasso()
alpha = np.logspace(-3,3,num=14) # range for alpha

ls_rs = RandomizedSearchCV(estimator = ls, param_distributions = dict(alpha=alpha

X_train, X_test, y_train, y_test = train_test_split(X, y, train_size = 0.8, rando

ls_rs.fit(X_train, y_train)

y_train_pred = ls_rs.predict(X_train)
y_test_pred = ls_rs.predict(X_test)

print("-----------Lasso-----------")
print("Train MAE:  ", mean_absolute_error(y_train, y_train_pred))
print("Train RMSE: ", np.sqrt(mean_absolute_error(y_train, y_train_pred)))

print("Test MAE:   ", mean_absolute_error(y_test, y_test_pred))
print("Test RMSE:  ", np.sqrt(mean_absolute_error(y_test, y_test_pred)))

print("\nLasso co-eff: ", lr.coef_)
```

```
-----------Lasso-----------
Train MAE:   0.3421765621597634
Train RMSE:  0.5849585986715328
Test MAE:    0.3541745177139357
Test RMSE:   0.5951256318744268

Lasso co-eff:  [ 0.07315582 -0.12354116 -0.01530178  0.07000975  0.44661629 -0.
00348994
  0.          0.25341892  0.08932491  0.25038661 -0.          0.64798963
  0.         -0.03070646 -0.08653632 -0.0610363 ]
```

In [124]:
```
sns.kdeplot((y_train - y_train_pred))
```

Out[124]: <AxesSubplot:xlabel='selling_price', ylabel='Density'>



In [125]:
```
from sklearn.ensemble import RandomForestRegressor
rf_reg = RandomForestRegressor()

indepent_var = ['year', 'km_driven', 'mileage', 'engine',
       'max_power', 'seats', 'fuel_CNG', 'fuel_Diesel', 'fuel_LPG',
       'seller_type_Dealer', 'seller_type_Trustmark Dealer',
       'transmission_Automatic', 'owner_First Owner',
       'owner_Fourth & Above Owner', 'owner_Second Owner',
       'owner_Third Owner']

X = one_hot_encoded_data[indepent_var]
y = one_hot_encoded_data["selling_price"]

X_train, X_test, y_train, y_test = train_test_split(X, y, train_size = 0.8, randc

rf_reg.fit(X_train, y_train)
y_pred= rf_reg.predict(X_test)

print("Accuracy on Traing set:  ",rf_reg.score(X_train, y_train))
print("Accuracy on Testing set: ",rf_reg.score(X_test, y_test))
```

```
Accuracy on Traing set:   0.9857310605047523
Accuracy on Testing set:  0.8810933733358634
```

In [126]:
```
# Seems like Linear Regression models perform not-so-well with our datset!
# Random Forest has given us an accuracy >95% on the dataset, while >86% on the t
# This may point to a overfitting of the data by the RF model!
# We can move further with this model, rather than working with OLS method!

# SO, it seems like regression models can predict the prices of the car models wi
```

In [127]:
```
# One last model to plot a regression line!
indepent_var = ['km_driven', 'max_power', 'fuel_Diesel',
       'seller_type_Dealer', 'transmission_Automatic']

X = one_hot_encoded_data[indepent_var]
y = one_hot_encoded_data["selling_price"]
```

In [128]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size = 0.8, randc
```

In [129]:
```
linear_reg = LinearRegression()
linear_reg.fit(X_train, y_train)
y_pred = linear_reg.predict(X_test)

print("Accuracy on Traing set: ", linear_reg.score(X_train,y_train))
print("Accuracy on Testing set: ", linear_reg.score(X_test,y_test))
```

```
Accuracy on Traing set:  0.5690630877448358
Accuracy on Testing set:  0.5782933844467597
```

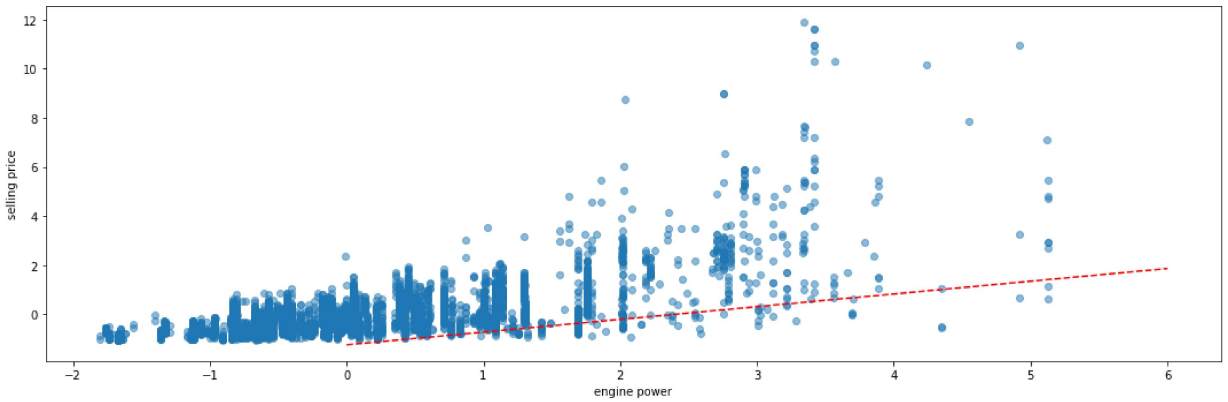
In [130]:
```
X_train = sm.add_constant(X_train)

lm = sm.OLS(y_train, X_train).fit()
print(lm.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:          selling_price   R-squared:                       0.569
Model:                            OLS   Adj. R-squared:                  0.569
Method:                 Least Squares   F-statistic:                     1408.
Date:                Mon, 19 Dec 2022   Prob (F-statistic):               0.00
Time:                        11:19:56   Log-Likelihood:                -5270.7
No. Observations:                5336   AIC:                         1.055e+04
Df Residuals:                    5330   BIC:                         1.059e+04
Df Model:                           5
Covariance Type:            nonrobust
================================================================================
===========
                               coef    std err          t      P>|t|      [0.025
0.975]
--------------------------------------------------------------------------------
-----------
const                       -0.3040      0.015    -20.199      0.000      -0.334
-0.274
km_driven                   -0.2648      0.010    -27.340      0.000      -0.284
-0.246
max_power                    0.5177      0.011     48.552      0.000       0.497
0.539
fuel_Diesel                  0.4035      0.020     19.706      0.000       0.363
0.444
seller_type_Dealer           0.2832      0.031      9.048      0.000       0.222
0.345
transmission_Automatic       0.6801      0.037     18.435      0.000       0.608
0.752
==============================================================================
Omnibus:                     4641.010   Durbin-Watson:                   1.984
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           400931.943
Skew:                           3.722   Prob(JB):                         0.00
Kurtosis:                      44.808   Cond. No.                         5.13
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctl
y specified.
```

In [131]:
```
# Not much of change in R^2 values! But notice how the F-statistic is around 1406
# This means that our model has fitted the data very well! and also notice how nc
```

In [132]:
```
x = one_hot_encoded_data['max_power']
y = one_hot_encoded_data['selling_price']

fig, ax = plt.subplots(figsize=(15, 5))

plt.scatter(x, y, alpha=0.5)
ax.set_xlabel('engine power')
ax.set_ylabel('selling price')

max_x = one_hot_encoded_data['max_power'].max()
min_x = one_hot_encoded_data['max_power'].min()

x = np.arange(min_x, max_x, 1)
y = (0.5177 * x) - 0.3040      #Using intercept and constant from the OLS summary

save_fig('Selling price vs max power(scaled) regression line')

plt.plot(y, 'r--')
plt.show()
```

```
Figure saved
```



In [133]:
```
x = one_hot_encoded_data['km_driven']
y = one_hot_encoded_data['selling_price']

fig, ax = plt.subplots(figsize=(15, 5))

plt.scatter(x, y, alpha=0.5)
ax.set_xlabel('distance driven')
ax.set_ylabel('selling price')

max_x = one_hot_encoded_data['km_driven'].max()
min_x = one_hot_encoded_data['km_driven'].min()

x = np.arange(min_x, max_x, 1)
y = (-0.2648 * x) - 0.3040   #Using intercept and constant from the OLS summary

save_fig('Selling price vs distance driven(scaled) regression line')

plt.plot(y, 'r--')
plt.show()
```

```
Figure saved
```