Car Price Prediction

statement

To be able to predict used cars market value can help both buyers and sellers. There are lots of individuals who are interested in the used car market at some points in their life because they wanted to sell their car or buy a used car. In this process, it’s a big corner to pay too much or sell less then it’s market value.

In this Project, we are going to predict the Price of Used Cars using various features like Present_Price, Selling_Price, Kms_Driven, Fuel_Type, Year etc. The data used in this project was downloaded from Kaggle.

1. Introduction

Hello everyone! In this project we will be working on Vehicle dataset . This dataset contains information about used cars. We are going to use for finding predictions of price with the use of Decision regression models.

The datasets consist of several independent variables include:

- Car_Name
- Year
- Selling_Price = the owmer wants to sell it on second hand (in lakhs).
- Present_Price = ex-showroom price (in lakhs).
- Kms_Driven = the kilometers the car has travelled till now.
- Fuel_Type
- Seller_Type
- Transmission
- Owner

We are going to use some of the variables which we need for Decision tree regression models

In [1]:
```
import numpy as np
import pandas as pd

# matplotlib
import matplotlib.pyplot as plt

#seaborn
import seaborn as sns

# ignore warnings
import warnings
warnings.filterwarnings("ignore")

import os
os.getcwd
# the location of current working directory (CWD).
```

Out[1]:
```
<function nt.getcwd()>
```

In [2]:
```
cpp = pd.read_csv(r'D:\Monty Corporates\Projects\Car Price Prediction\Car Pred
cpp
```

Out[2]:

	Car_Name	Year	Selling_Price	Present_Price	Kms_Driven	Fuel_Type	Seller_Type	Transmi
0	ritz	2014	3.35	5.59	27000	Petrol	Dealer	M
1	sx4	2013	4.75	9.54	43000	Diesel	Dealer	M
2	ciaz	2017	7.25	9.85	6900	Petrol	Dealer	M
3	wagon r	2011	2.85	4.15	5200	Petrol	Dealer	M
4	swift	2014	4.60	6.87	42450	Diesel	Dealer	M
...	...	...	...	...	...	...	...	
296	city	2016	9.50	11.60	33988	Diesel	Dealer	M
297	brio	2015	4.00	5.90	60000	Petrol	Dealer	M
298	city	2009	3.35	11.00	87934	Petrol	Dealer	M
299	city	2017	11.50	12.50	9000	Diesel	Dealer	M
300	brio	2016	5.30	5.90	5464	Petrol	Dealer	M

301 rows × 9 columns

In [3]:
```
cpp.shape
```

Out[3]:
```
(301, 9)
```

In [4]:
```
cpp.head()
```

Out[4]:

	Car_Name	Year	Selling_Price	Present_Price	Kms_Driven	Fuel_Type	Seller_Type	Transmiss
0	ritz	2014	3.35	5.59	27000	Petrol	Dealer	Mar
1	sx4	2013	4.75	9.54	43000	Diesel	Dealer	Mar
2	ciaz	2017	7.25	9.85	6900	Petrol	Dealer	Mar
3	wagon r	2011	2.85	4.15	5200	Petrol	Dealer	Mar
4	swift	2014	4.60	6.87	42450	Diesel	Dealer	Mar

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 301 entries, 0 to 300
Data columns (total 9 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Car_Name       301 non-null    object 
 1   Year           301 non-null    int64  
 2   Selling_Price  301 non-null    float64
 3   Present_Price  301 non-null    float64
 4   Kms_Driven     301 non-null    int64  
 5   Fuel_Type      301 non-null    object 
 6   Seller_Type    301 non-null    object 
 7   Transmission   301 non-null    object 
 8   Owner          301 non-null    int64  
dtypes: float64(2), int64(3), object(4)
memory usage: 21.3+ KB
```

In [6]:
```
cpp.describe()
```

Out[6]:

	Year	Selling_Price	Present_Price	Kms_Driven	Owner
count	301.000000	301.000000	301.000000	301.000000	301.000000
mean	2013.627907	4.661296	7.628472	36947.205980	0.043189
std	2.891554	5.082812	8.644115	38886.883882	0.247915
min	2003.000000	0.100000	0.320000	500.000000	0.000000
25%	2012.000000	0.900000	1.200000	15000.000000	0.000000
50%	2014.000000	3.600000	6.400000	32000.000000	0.000000
75%	2016.000000	6.000000	9.900000	48767.000000	0.000000
max	2018.000000	35.000000	92.600000	500000.000000	3.000000

We don't have any null values

In [7]:
```
#checking Numerical values
numerical_columns = cpp.select_dtypes(include=['int', 'float']).columns
numerical_columns
```

Out[7]:
```
Index(['Year', 'Selling_Price', 'Present_Price', 'Kms_Driven', 'Owner'], dtyp
e='object')
```

In [8]:
```
#checking Categorical values
categorical_columns = cpp.select_dtypes(include=['object']).columns
categorical_columns
```

Out[8]:
```
Index(['Car_Name', 'Fuel_Type', 'Seller_Type', 'Transmission'], dtype='objec
t')
```

In [9]:
```
#checking null values
cpp.isna().sum()
```

Out[9]:
```
Car_Name         0
Year             0
Selling_Price    0
Present_Price    0
Kms_Driven       0
Fuel_Type        0
Seller_Type      0
Transmission     0
Owner            0
dtype: int64
```

In [10]:
```
cpp.isna().any()
```

Out[10]:
```
Car_Name         False
Year             False
Selling_Price    False
Present_Price    False
Kms_Driven       False
Fuel_Type        False
Seller_Type      False
Transmission     False
Owner            False
dtype: bool
```

In [11]:
```
cpp.Owner.unique()
```

Out[11]:
```
array([0, 1, 3], dtype=int64)
```

In [12]:
```
print(cpp.Car_Name.unique())
```

```
['ritz' 'sx4' 'ciaz' 'wagon r' 'swift' 'vitara brezza' 's cross'
 'alto 800' 'ertiga' 'dzire' 'alto k10' 'ignis' '800' 'baleno' 'omni'
 'fortuner' 'innova' 'corolla altis' 'etios cross' 'etios g' 'etios liva'
 'corolla' 'etios gd' 'camry' 'land cruiser' 'Royal Enfield Thunder 500'
 'UM Renegade Mojave' 'KTM RC200' 'Bajaj Dominar 400'
 'Royal Enfield Classic 350' 'KTM RC390' 'Hyosung GT250R'
 'Royal Enfield Thunder 350' 'KTM 390 Duke ' 'Mahindra Mojo XT300'
 'Bajaj Pulsar RS200' 'Royal Enfield Bullet 350'
 'Royal Enfield Classic 500' 'Bajaj Avenger 220' 'Bajaj Avenger 150'
 'Honda CB Hornet 160R' 'Yamaha FZ S V 2.0' 'Yamaha FZ 16'
 'TVS Apache RTR 160' 'Bajaj Pulsar 150' 'Honda CBR 150' 'Hero Extreme'
 'Bajaj Avenger 220 dtsi' 'Bajaj Avenger 150 street' 'Yamaha FZ  v 2.0'
 'Bajaj Pulsar  NS 200' 'Bajaj Pulsar 220 F' 'TVS Apache RTR 180'
 'Hero Passion X pro' 'Bajaj Pulsar NS 200' 'Yamaha Fazer '
 'Honda Activa 4G' 'TVS Sport ' 'Honda Dream Yuga '
 'Bajaj Avenger Street 220' 'Hero Splender iSmart' 'Activa 3g'
 'Hero Passion Pro' 'Honda CB Trigger' 'Yamaha FZ S '
 'Bajaj Pulsar 135 LS' 'Activa 4g' 'Honda CB Unicorn'
 'Hero Honda CBZ extreme' 'Honda Karizma' 'Honda Activa 125' 'TVS Jupyter'
 'Hero Honda Passion Pro' 'Hero Splender Plus' 'Honda CB Shine'
 'Bajaj Discover 100' 'Suzuki Access 125' 'TVS Wego' 'Honda CB twister'
 'Hero Glamour' 'Hero Super Splendor' 'Bajaj Discover 125' 'Hero Hunk'
 'Hero  Ignitor Disc' 'Hero  CBZ Xtreme' 'Bajaj  ct 100' 'i20' 'grand i10'
 'i10' 'eon' 'xcent' 'elantra' 'creta' 'verna' 'city' 'brio' 'amaze'
 'jazz']
```

In [13]:
```
cpp.Car_Name.value_counts()
```

Out[13]:
```
city                        26
corolla altis               16
verna                       14
fortuner                    11
brio                        10
                            ..
Honda CB Trigger             1
Yamaha FZ S                  1
Bajaj Pulsar 135 LS          1
Activa 4g                    1
Bajaj Avenger Street 220     1
Name: Car_Name, Length: 98, dtype: int64
```

In [14]:
```
print(cpp.Fuel_Type.value_counts())
```

```
Petrol    239
Diesel     60
CNG         2
Name: Fuel_Type, dtype: int64
```

In [15]:
```
print(cpp.Seller_Type.value_counts())
```

```
Dealer        195
Individual    106
Name: Seller_Type, dtype: int64
```

In [16]:
```
print(cpp.Transmission.value_counts())
```

```
Manual       261
Automatic     40
Name: Transmission, dtype: int64
```

In [17]:
```
cpp['Present_Year'] = 2023
```

In [18]:
```
cpp
```

Out[18]:

	Car_Name	Year	Selling_Price	Present_Price	Kms_Driven	Fuel_Type	Seller_Type	Transmi
0	ritz	2014	3.35	5.59	27000	Petrol	Dealer	M
1	sx4	2013	4.75	9.54	43000	Diesel	Dealer	M
2	ciaz	2017	7.25	9.85	6900	Petrol	Dealer	M
3	wagon r	2011	2.85	4.15	5200	Petrol	Dealer	M
4	swift	2014	4.60	6.87	42450	Diesel	Dealer	M
...	...	...	...	...	...	...	...	
296	city	2016	9.50	11.60	33988	Diesel	Dealer	M
297	brio	2015	4.00	5.90	60000	Petrol	Dealer	M
298	city	2009	3.35	11.00	87934	Petrol	Dealer	M
299	city	2017	11.50	12.50	9000	Diesel	Dealer	M
300	brio	2016	5.30	5.90	5464	Petrol	Dealer	M

301 rows × 10 columns

In [19]:
```
# Creating a new feature called total no. of years old my car,bcz It's importa
cpp['Car_age'] = cpp['Present_Year']-cpp['Year']
cpp.head()
```

Out[19]:

	Car_Name	Year	Selling_Price	Present_Price	Kms_Driven	Fuel_Type	Seller_Type	Transmiss
0	ritz	2014	3.35	5.59	27000	Petrol	Dealer	Mar
1	sx4	2013	4.75	9.54	43000	Diesel	Dealer	Mar
2	ciaz	2017	7.25	9.85	6900	Petrol	Dealer	Mar
3	wagon r	2011	2.85	4.15	5200	Petrol	Dealer	Mar
4	swift	2014	4.60	6.87	42450	Diesel	Dealer	Mar

In [20]:
```
#dropping the unnessasary attributes
cpp = cpp.drop(columns = ['Year', 'Present_Year'], axis = 0)
cpp.head()
```

Out[20]:

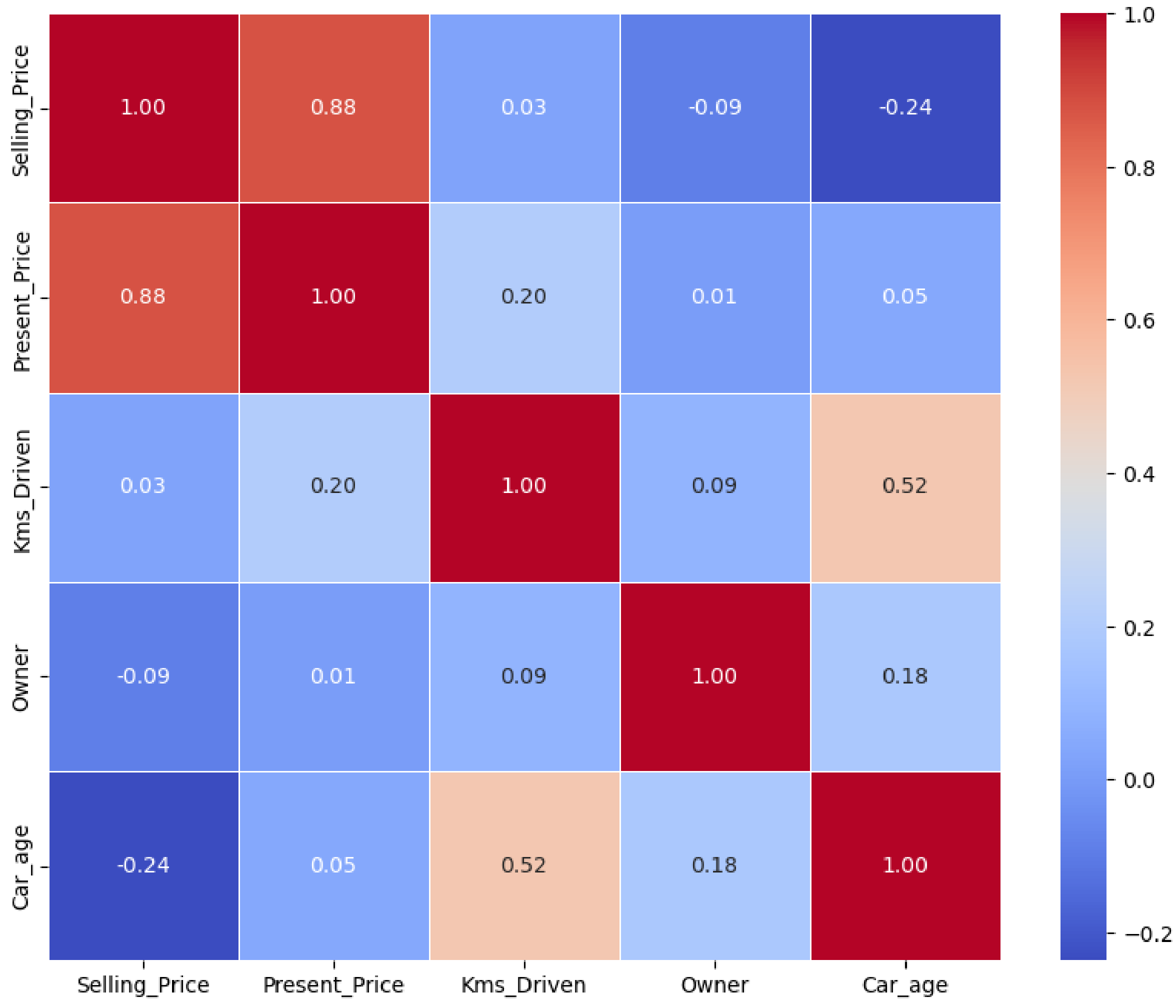
	Car_Name	Selling_Price	Present_Price	Kms_Driven	Fuel_Type	Seller_Type	Transmission	C
0	ritz	3.35	5.59	27000	Petrol	Dealer	Manual	
1	sx4	4.75	9.54	43000	Diesel	Dealer	Manual	
2	ciaz	7.25	9.85	6900	Petrol	Dealer	Manual	
3	wagon r	2.85	4.15	5200	Petrol	Dealer	Manual	
4	swift	4.60	6.87	42450	Diesel	Dealer	Manual	

In [21]:
```
cpp.shape
```

Out[21]:
```
(301, 9)
```

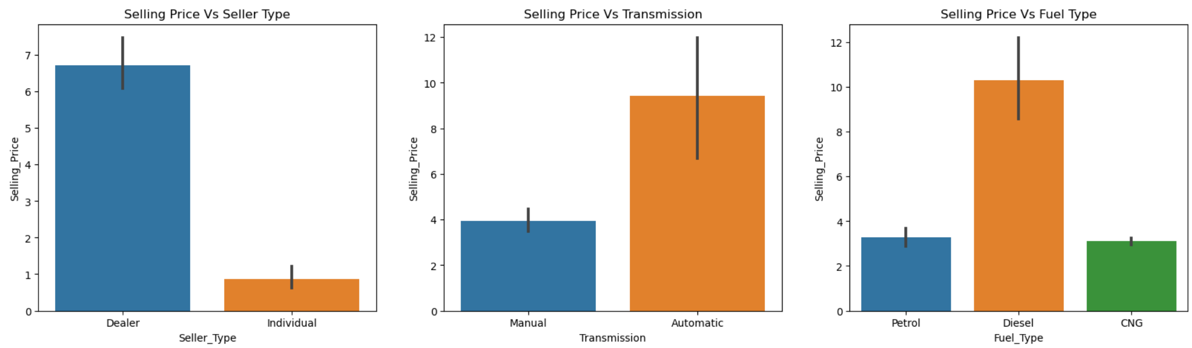
Visualization

In [22]:
```
correlation_data = cpp.corr()
plt.figure(figsize=(10,8))
sns.heatmap(correlation_data, annot = True, cmap = 'coolwarm',fmt ='.2f', linew
plt.show()
```



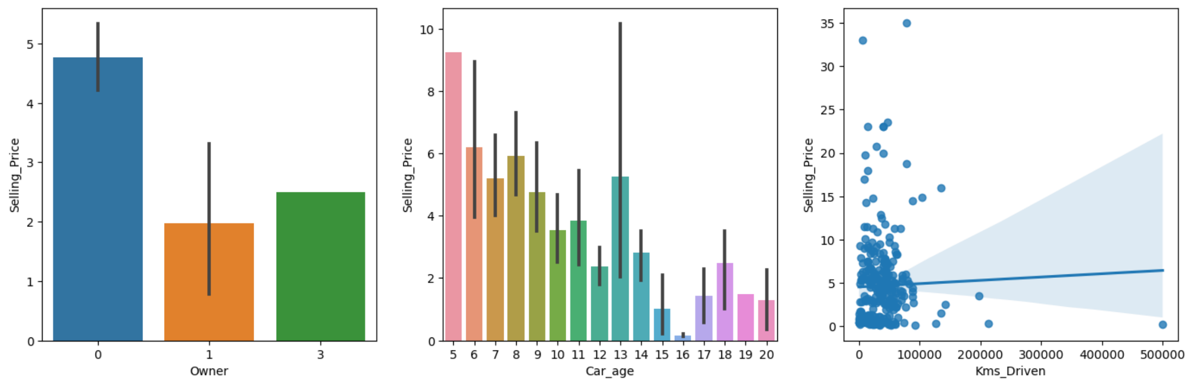
In [23]:
```
#Seller Type, Transmission nd Fuel Type Visualization with target variable
#target variable = selling price
plt.figure(figsize=[20,5])
plt.subplot(1,3,1)
sns.barplot(data = cpp, x= cpp['Seller_Type'], y = cpp['Selling_Price'])
plt.title('Selling Price Vs Seller Type')

plt.subplot(1,3,2)
sns.barplot(data = cpp, x=cpp['Transmission'],y = cpp['Selling_Price'])
plt.title('Selling Price Vs Transmission')

plt.subplot(1,3,3)
sns.barplot(data = cpp,x = cpp['Fuel_Type'],y = cpp['Selling_Price'])
plt.title('Selling Price Vs Fuel Type')
```

Out[23]:
```
Text(0.5, 1.0, 'Selling Price Vs Fuel Type')
```



- Selling Price of cars seems to have higher prices when sold by Dealers when compared to Individuals
- It can be observed that Selling Price would be higher for cars that are Automatic.
- Selling Price of cars with Fuel Type of Diesel is higher than Petrol and CNG

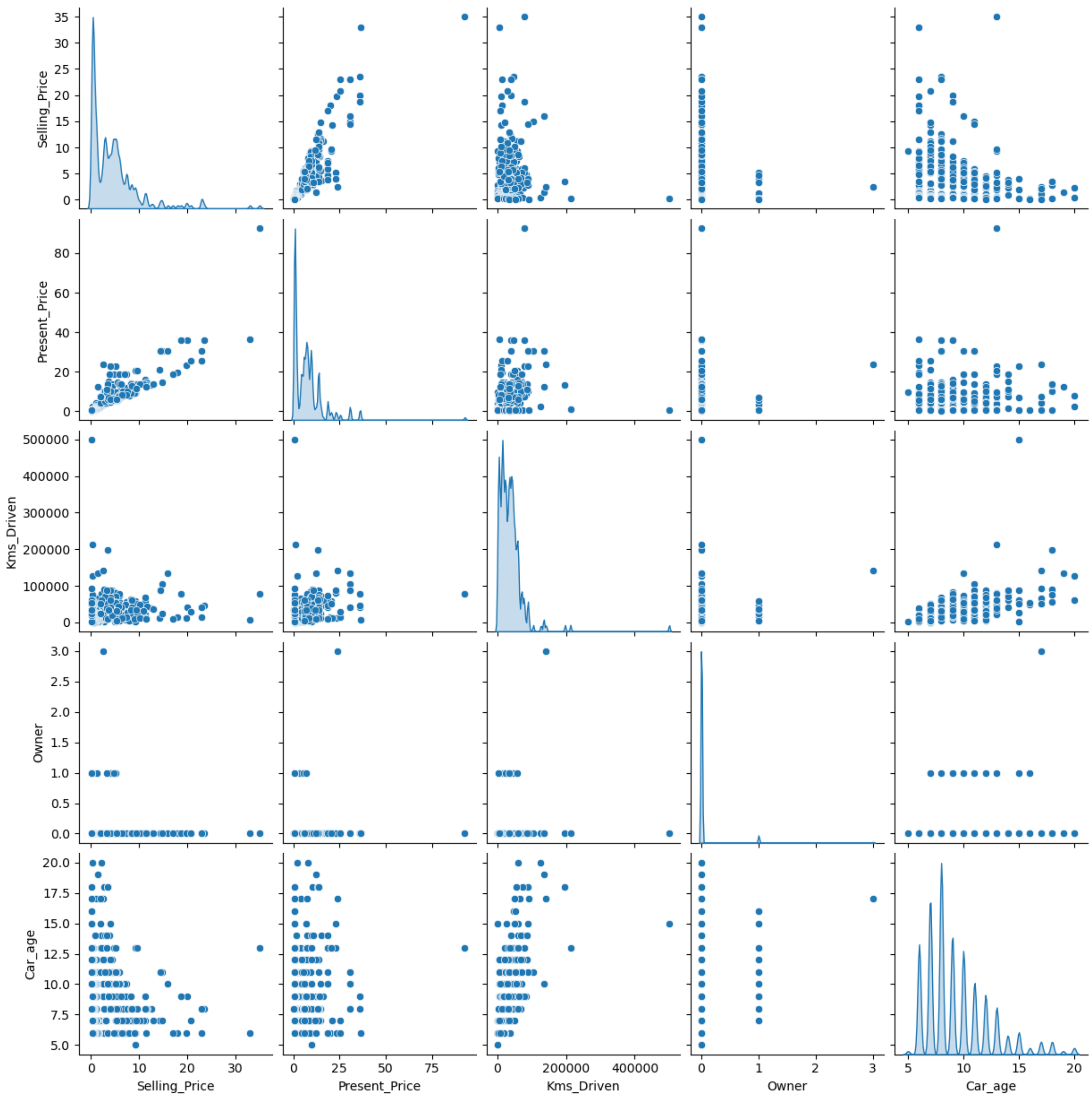
In [24]:
```
plt.figure(figsize=[17,5])
plt.subplot(1,3,1)
sns.barplot(data = cpp,x = cpp['Owner'],y = cpp['Selling_Price'])

plt.subplot(1,3,2)
sns.barplot(data = cpp, x = cpp['Car_age'],y = cpp['Selling_Price'])

plt.subplot(1,3,3)
sns.regplot(data = cpp , x = cpp['Kms_Driven'],y = cpp['Selling_Price'])
plt.show()
```



- Selling Price is high with less Owners used Cars *
- Selling Price of cars 2 years old would be high and gradually decreases with car of 17 years old *
- Lesser the Kms driven higher the Selling Price *

In [25]:
```
plt.figure(figsize=[17,5])
plt.subplot(1,3,1)
sns.regplot(data = cpp, x = cpp['Selling_Price'],y = cpp['Present_Price'])

plt.subplot(1,3,2)
sns.distplot(np.log(cpp['Selling_Price']))
plt.title('Distribution of Selling Price')


plt.subplot(1,3,3)
sns.distplot(np.log(cpp['Kms_Driven']))
plt.title('Distribution of KMS Drived')


plt.title('Kilometers Drived')
plt.show()

#The np.log() is a mathematical function that helps user to calculate Natural
```

In [26]:
```
sns.pairplot(cpp,diag_kind="kde", diag_kws=dict(shade=True, bw=0.05, vertical=
plt.show()
```



In [27]:
```
cpp
```

	Car_Name	Selling_Price	Present_Price	Kms_Driven	Fuel_Type	Seller_Type	Transmission
0	ritz	3.35	5.59	27000	Petrol	Dealer	Manu
1	sx4	4.75	9.54	43000	Diesel	Dealer	Manu
2	ciaz	7.25	9.85	6900	Petrol	Dealer	Manu
3	wagon r	2.85	4.15	5200	Petrol	Dealer	Manu
4	swift	4.60	6.87	42450	Diesel	Dealer	Manu
...	...	...	...	...	...	...	
296	city	9.50	11.60	33988	Diesel	Dealer	Manu
297	brio	4.00	5.90	60000	Petrol	Dealer	Manu
298	city	3.35	11.00	87934	Petrol	Dealer	Manu
299	city	11.50	12.50	9000	Diesel	Dealer	Manu
300	brio	5.30	5.90	5464	Petrol	Dealer	Manu

301 rows × 9 columns

Encoding the categorical data

- Machine learning will doesn't work on categorical data . So we have to convert categorical to numerical data
- ML will not take input as object, it will send error(categorical data)
- ML will take int and float values (numerical data)

In [28]:
```
# Replace categorical values with numerical representations
#Fuel_type : diesel=0,petrol = 1, cng = 2
#seller type : dealer =1,individual = 0
#transmission : manual = 1 , automatic =0
cpp['Fuel_Type'].replace({"Petrol": 1, "Diesel": 0, "CNG": 2}, inplace=True)
cpp['Seller_Type'].replace({"Dealer": 1, "Individual": 0}, inplace=True)
cpp['Transmission'].replace({"Manual": 1, "Automatic": 0}, inplace=True)

# Convert columns to integer data type
cpp[['Fuel_Type', 'Seller_Type', 'Transmission']] = cpp[['Fuel_Type', 'Seller_
```

In [29]:
```
cpp
```

Out[29]:

	Car_Name	Selling_Price	Present_Price	Kms_Driven	Fuel_Type	Seller_Type	Transmission
0	ritz	3.35	5.59	27000	1	1	1
1	sx4	4.75	9.54	43000	0	1	1
2	ciaz	7.25	9.85	6900	1	1	1
3	wagon r	2.85	4.15	5200	1	1	1
4	swift	4.60	6.87	42450	0	1	1
...	...	...	...	...	...	...	...
296	city	9.50	11.60	33988	0	1	1
297	brio	4.00	5.90	60000	1	1	1
298	city	3.35	11.00	87934	1	1	1
299	city	11.50	12.50	9000	0	1	1
300	brio	5.30	5.90	5464	1	1	1

301 rows × 9 columns

In [30]:
```
#dropping the unnessasary attributes
cpp = cpp.drop(columns = ['Car_Name'], axis = 0)
cpp.head()
```

Out[30]:

	Selling_Price	Present_Price	Kms_Driven	Fuel_Type	Seller_Type	Transmission	Owner	Car_a
0	3.35	5.59	27000	1	1	1	0	
1	4.75	9.54	43000	0	1	1	0	
2	7.25	9.85	6900	1	1	1	0	
3	2.85	4.15	5200	1	1	1	0	
4	4.60	6.87	42450	0	1	1	0	

Building the model

In [31]:
```
x = cpp.drop('Selling_Price',axis = 1)
y = cpp['Selling_Price']
```

In [32]:
```
x.head()
```

Out[32]:

	Present_Price	Kms_Driven	Fuel_Type	Seller_Type	Transmission	Owner	Car_age
0	5.59	27000	1	1	1	0	9
1	9.54	43000	0	1	1	0	10
2	9.85	6900	1	1	1	0	6
3	4.15	5200	1	1	1	0	12
4	6.87	42450	0	1	1	0	9

In [33]:
```
y.head()
```

Out[33]:
```
0    3.35
1    4.75
2    7.25
3    2.85
4    4.60
Name: Selling_Price, dtype: float64
```

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

Out[34]: (301, 7)

In [35]:
```
y.shape
```

Out[35]: (301,)

In [36]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x , y , test_size=0.25, rando
```

In [37]:
```
x_train.shape
```

Out[37]: (225, 7)

In [38]:
```
x_test.shape
```

Out[38]: (76, 7)

In [39]:
```
y_train.shape
```

Out[39]: (225,)

In [40]:
```
y_test.shape
```

Out[40]: (76,)

In [41]:
```
from sklearn.tree import DecisionTreeRegressor

# Define the hyperparameters as a dictionary
hyperparameters = {
    'splitter': 'best',
    'random_state': 66,
    'min_samples_split': 21,
    'min_samples_leaf': 9,
    'max_features': 'auto',
    'max_depth': 60,
    'criterion': 'friedman_mse'
}

# Initialize the DecisionTreeRegressor with the hyperparameters
regressor = DecisionTreeRegressor(**hyperparameters)

# Fit the model
regressor.fit(x_train, y_train)
```

Out[41]:
```
▾                        DecisionTreeRegressor

DecisionTreeRegressor(criterion='friedman_mse', max_depth=60,
                      max_features='auto', min_samples_leaf=9,
                      min_samples_split=21, random_state=66)
```

In [42]:
```
y_pred = regressor.predict(x_test)
y_pred
```

Out[42]:
```
array([ 7.42842105,  0.48545455,  5.37857143,  7.42842105, 10.84157895,
        5.37857143,  3.94      ,  0.423125  ,  3.94      ,  5.07666667,
        2.8575    ,  0.65933333,  5.37857143,  7.42842105,  7.42842105,
       10.84157895,  7.42842105,  3.94      ,  0.48545455,  1.28153846,
        5.07666667,  5.37857143,  5.07666667, 10.84157895,  0.21705882,
        0.65933333,  0.21705882,  0.48545455,  0.48545455,  5.85285714,
        2.43      ,  5.07666667,  0.48545455,  7.42842105,  2.43      ,
        1.28153846,  5.37857143,  4.5       ,  0.21705882, 10.84157895,
        7.42842105, 20.98888889,  5.37857143,  4.5       ,  5.07666667,
       10.84157895,  0.21705882,  0.65933333,  5.07666667,  7.42842105,
        5.07666667,  2.8575    ,  5.07666667, 20.98888889,  1.28153846,
        1.28153846,  0.21705882,  2.8575    ,  3.94      ,  2.8575    ,
        5.85285714,  5.07666667,  2.8575    , 20.98888889,  3.94      ,
        5.37857143, 10.84157895,  5.85285714,  0.423125  ,  3.94      ,
        2.8575    ,  2.8575    ,  0.423125  ,  5.07666667,  0.48545455,
        5.37857143])
```

In [44]:
```
#in regression type models we will use Mean Squared Error (MSE),Root Mean Squa
```

In [45]:
```
# we calculated the R-squared score using the r2_score() function from scikit-
# to evaluate the performance of the Decision Tree Regressor model.
#r2 value in between 0 and 1
#It compares the predicted values (y_pred) with the actual target values (y_te
from sklearn.metrics import r2_score #find the accuracy of model
r2_score(y_test , y_pred)
```

Out[45]: 0.8572640488749044

In [46]:
```
bias = regressor.score(x_train, y_train)
bias
```

Out[46]: 0.8970544467935754

In [47]:
```
variance = regressor.score(x_test, y_test)
variance
```

Out[47]: 0.8572640488749044

In [ ]: