## Problem

With the covid 19 impact in the market, we have seen lot of changes in the car market. Now some cars are in demand hence making them costly and some are not in demand hence cheaper. One of our clients works with small traders, who sell used cars. With the change in market due to covid 19 impact, our client is facing problems with their previous car price valuation machine learning models. So, they are looking for new machine learning models from new data. We have to make car price valuation model. This project contains two phase

## Importing required libraries

In [1]:
```
import pandas as pd
import numpy as np
import seaborn  as sb
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LinearRegression
from sklearn import metrics
from sklearn.linear_model import Lasso
from sklearn.preprocessing import LabelEncoder
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

## Data Collection and Processing

In [2]:
```
df=pd.read_csv("car data.csv")
df.head()
```

Out[2]:

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

In [3]:
```
# Checking the number of columns and rows
df.shape
```

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

In [4]:
```
# getting some information about the dataset
df.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 [5]:
```
# Checking the number of missing values
df.isnull().sum()
```

Out[5]:
```
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
```

Here is no missing values

In [6]:
```
# Checking the distribution of categorical data
print(df.Fuel_Type.value_counts())
print(df.Seller_Type.value_counts())
print(df.Transmission.value_counts())
```

```
Petrol    239
Diesel     60
CNG         2
Name: Fuel_Type, dtype: int64
Dealer        195
Individual    106
Name: Seller_Type, dtype: int64
Manual       261
Automatic     40
Name: Transmission, dtype: int64
```

## Encoding the Categorical Data

In [7]:
```
#load the label Ecoder fuction
label_encode=LabelEncoder()
df['Car_Name']=label_encode.fit_transform(df['Car_Name'])
df['Fuel_Type']=label_encode.fit_transform(df['Fuel_Type'])
df['Seller_Type']=label_encode.fit_transform(df['Seller_Type'])
df['Transmission']=label_encode.fit_transform(df['Transmission'])
```

In [8]:
```
df.head(10)
```

Out[8]:

| | Car_Name | Year | Selling_Price | Present_Price | Kms_Driven | Fuel_Type | Seller_Type | Transmission | C |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 90 | 2014 | 3.35 | 5.59 | 27000 | 2 | 0 | 1 | |
| **1** | 93 | 2013 | 4.75 | 9.54 | 43000 | 1 | 0 | 1 | |
| **2** | 68 | 2017 | 7.25 | 9.85 | 6900 | 2 | 0 | 1 | |
| **3** | 96 | 2011 | 2.85 | 4.15 | 5200 | 2 | 0 | 1 | |
| **4** | 92 | 2014 | 4.60 | 6.87 | 42450 | 1 | 0 | 1 | |
| **5** | 95 | 2018 | 9.25 | 9.83 | 2071 | 1 | 0 | 1 | |
| **6** | 68 | 2015 | 6.75 | 8.12 | 18796 | 2 | 0 | 1 | |
| **7** | 91 | 2015 | 6.50 | 8.61 | 33429 | 1 | 0 | 1 | |
| **8** | 68 | 2016 | 8.75 | 8.89 | 20273 | 1 | 0 | 1 | |
| **9** | 68 | 2015 | 7.45 | 8.92 | 42367 | 1 | 0 | 1 | |

In [ ]:
```
```

## Statistical summary of the data -Descriptive Statistics

In [9]:
```
df.describe()
```

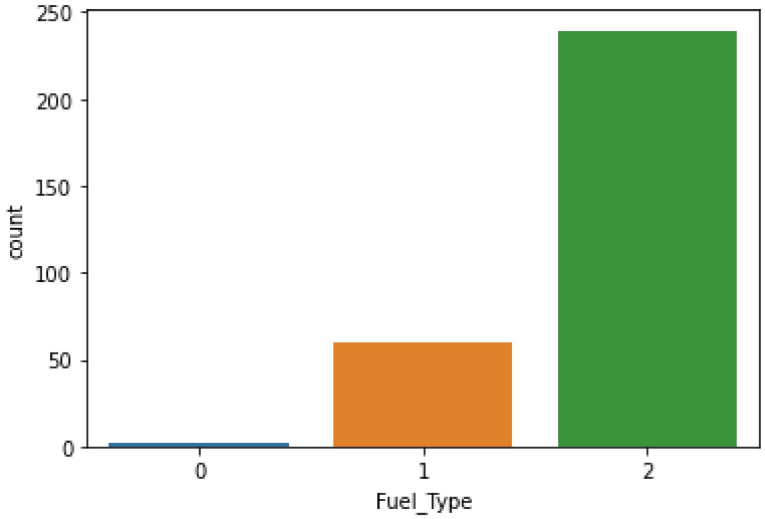
Out[9]:

| | Car_Name | Year | Selling_Price | Present_Price | Kms_Driven | Fuel_Type | Seller_Type |
|---|---|---|---|---|---|---|---|
| **count** | 301.000000 | 301.000000 | 301.000000 | 301.000000 | 301.000000 | 301.000000 | 301.000000 |
| **mean** | 62.571429 | 2013.627907 | 4.661296 | 7.628472 | 36947.205980 | 1.787375 | 0.352159 |
| **std** | 25.573535 | 2.891554 | 5.082812 | 8.644115 | 38886.883882 | 0.425801 | 0.478439 |
| **min** | 0.000000 | 2003.000000 | 0.100000 | 0.320000 | 500.000000 | 0.000000 | 0.000000 |
| **25%** | 47.000000 | 2012.000000 | 0.900000 | 1.200000 | 15000.000000 | 2.000000 | 0.000000 |
| **50%** | 69.000000 | 2014.000000 | 3.600000 | 6.400000 | 32000.000000 | 2.000000 | 0.000000 |
| **75%** | 82.000000 | 2016.000000 | 6.000000 | 9.900000 | 48767.000000 | 2.000000 | 1.000000 |
| **max** | 97.000000 | 2018.000000 | 35.000000 | 92.600000 | 500000.000000 | 2.000000 | 1.000000 |

## Data Visualiztion

In [10]:
```
#countplot for the target column for checking the distribution of target
sns.countplot(x='Fuel_Type',data=df)
print(df["Fuel_Type"].value_counts())
```

```
2    239
1     60
0      2
Name: Fuel_Type, dtype: int64
```

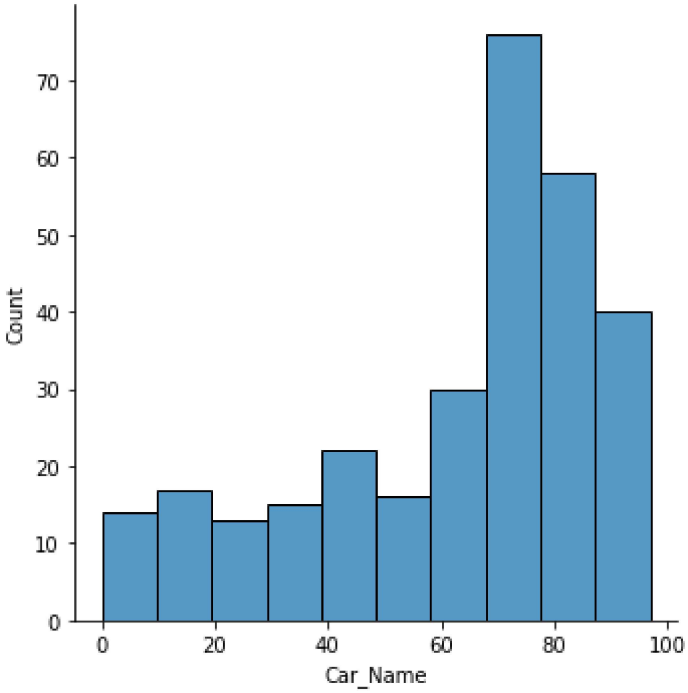


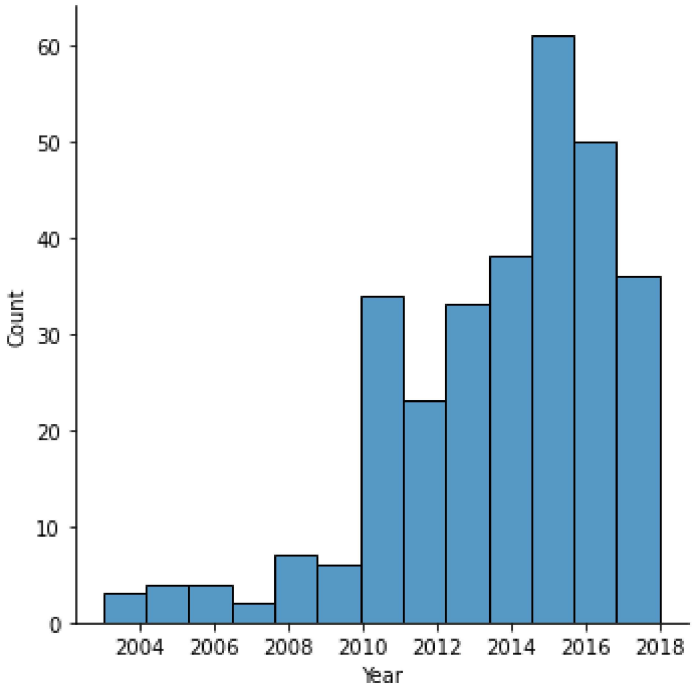

## Distribution plot for all columns

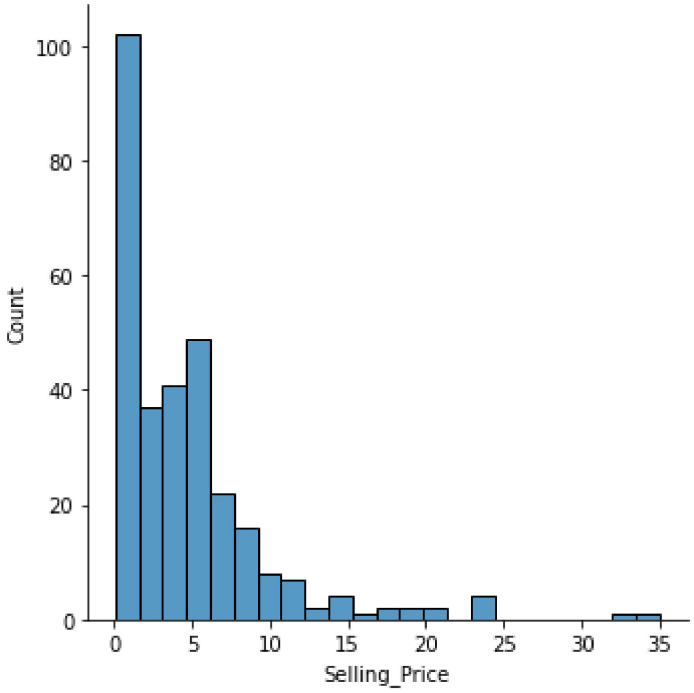
In [11]:
```
#this is how we can get all the column names of the datafram
for column in df:
    print(column)
```

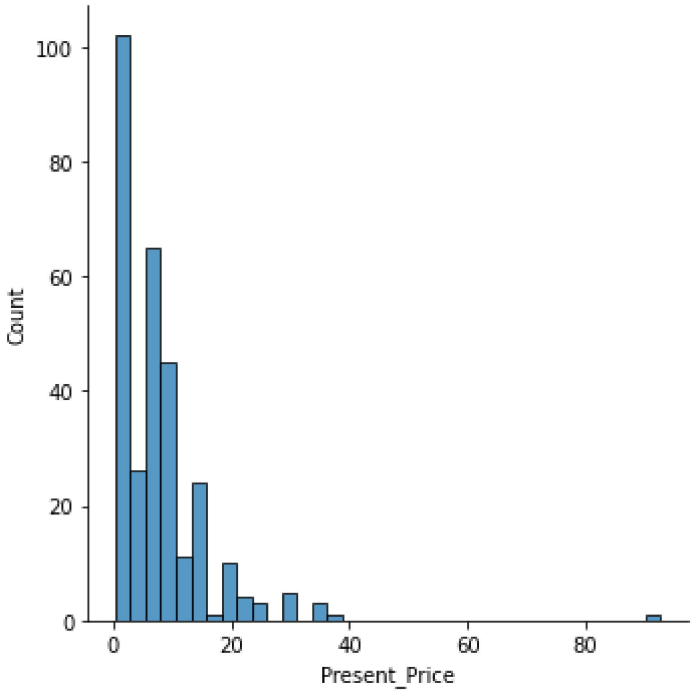
```
Car_Name
Year
Selling_Price
Present_Price
Kms_Driven
Fuel_Type
Seller_Type
Transmission
Owner
```

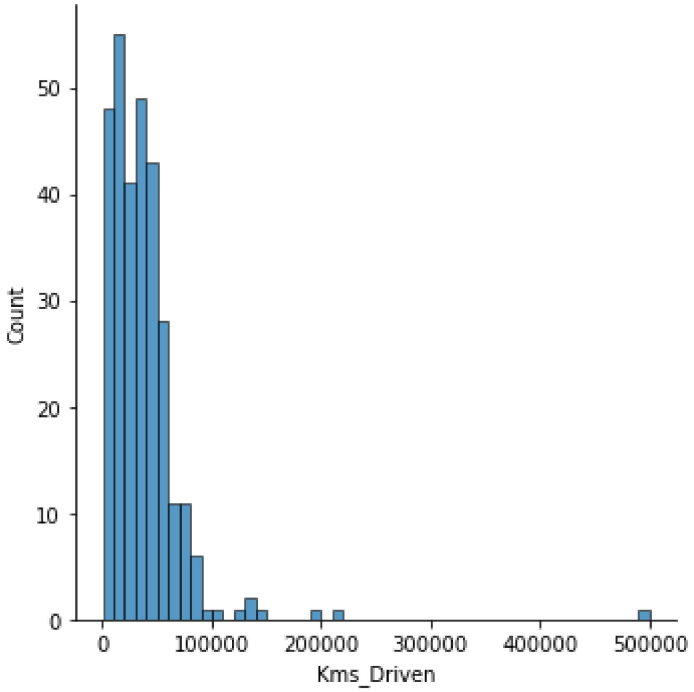
In [12]:
```
#creating a for loop to get the distribution plot for all columns
for column in df:
    sns.displot(x=column,data=df)
```

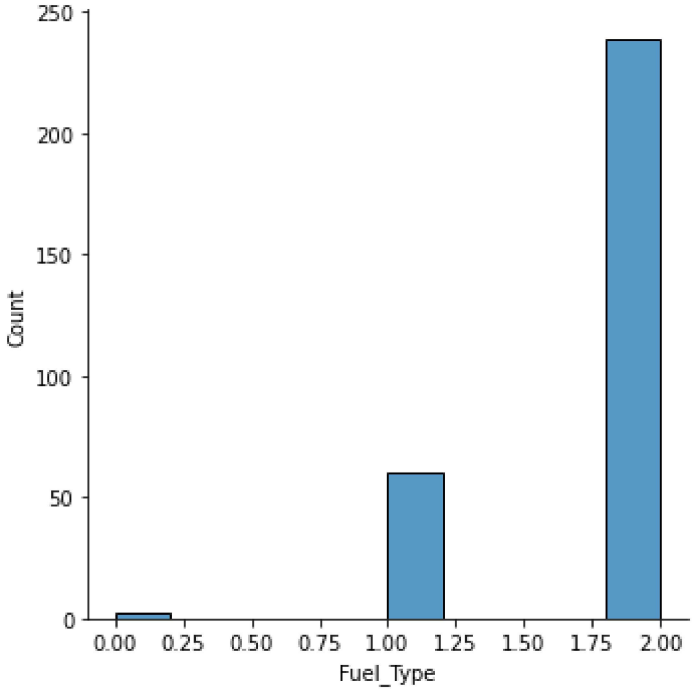


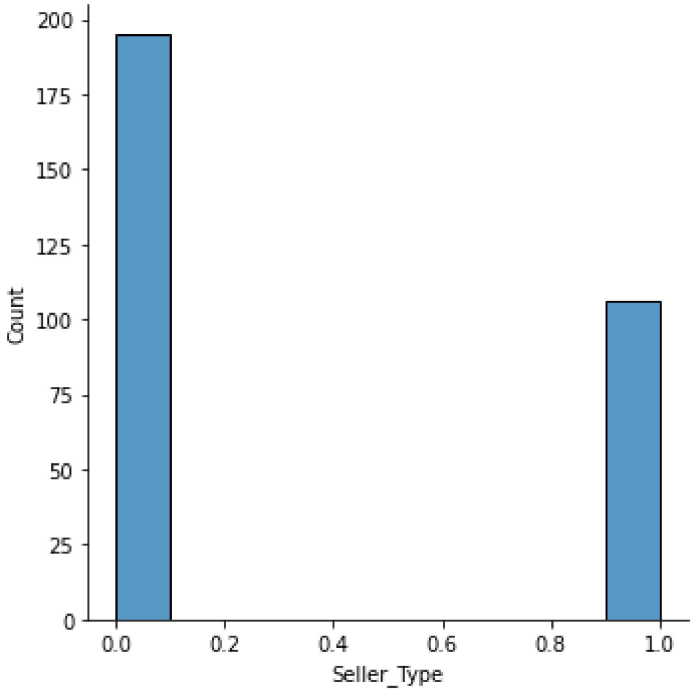

Count
Year
0
10
20
30
40
50
60
2004
2006
2008
2010
2012
2014
2016
2018

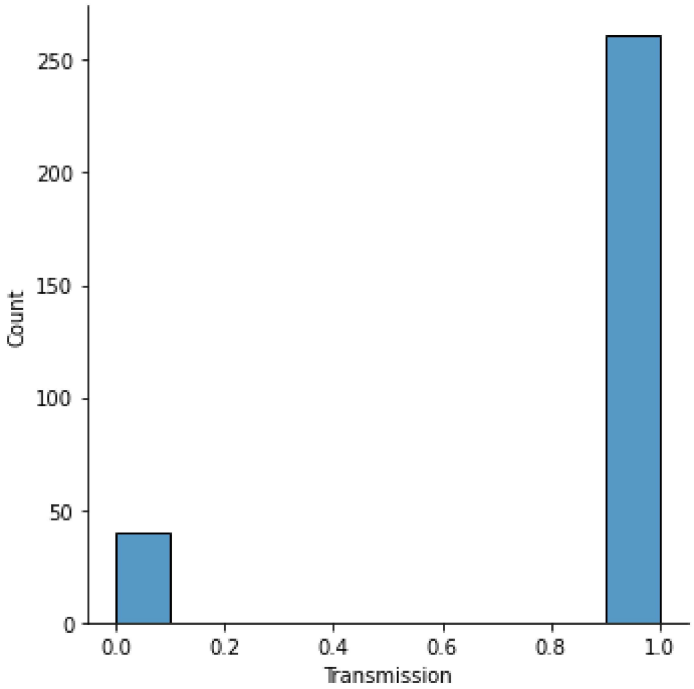

Count
Selling_Price
0
20
40
60
80
100
0
5
10
15
20
25
30
35

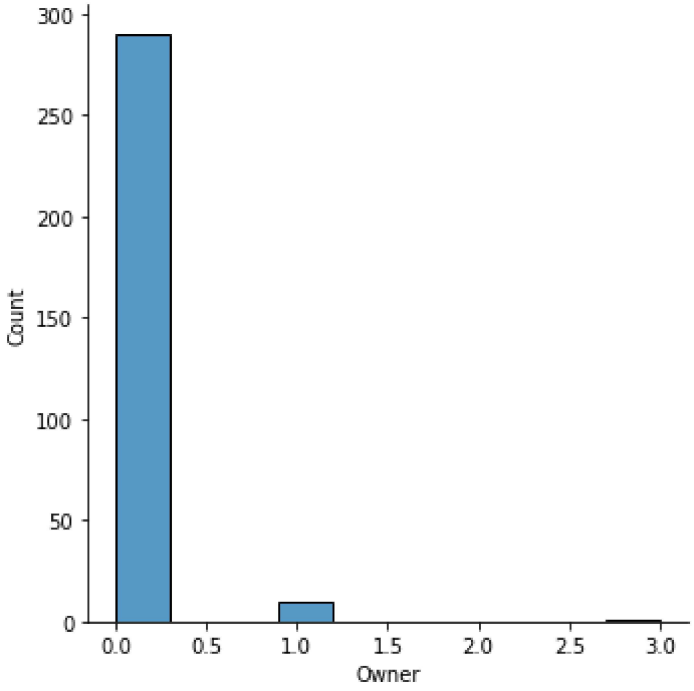

Count
Present_Price
0
20
40
60
80
100


Count
Kms_Driven
0
100000
200000
300000
400000
500000
10
30
50


250
200
150
100
50
0
Count
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
Fuel_Type

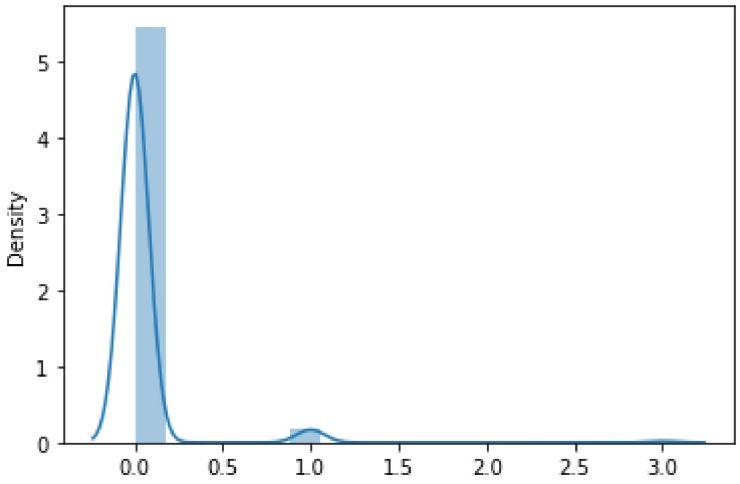

200
175
150
125
100
75
50
25
0
Count
0.0
0.2
0.4
0.6
0.8
1.0
Seller_Type



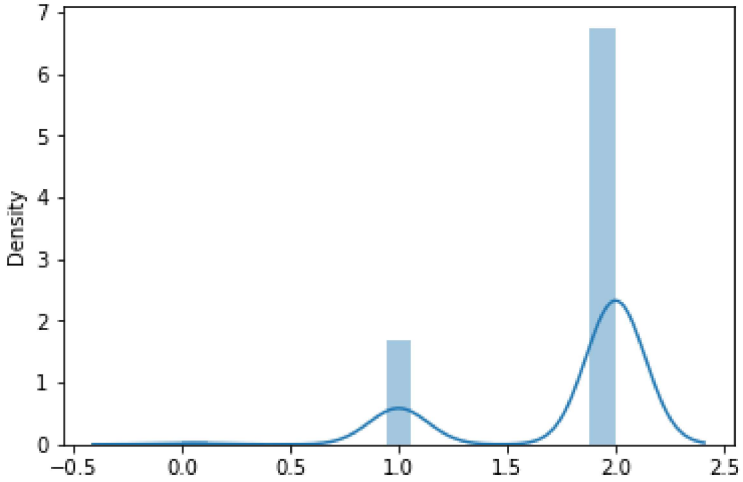

In [13]:
```
sns.distplot(x=df.Owner)
```

Out[13]:
```
<AxesSubplot:ylabel='Density'>
```

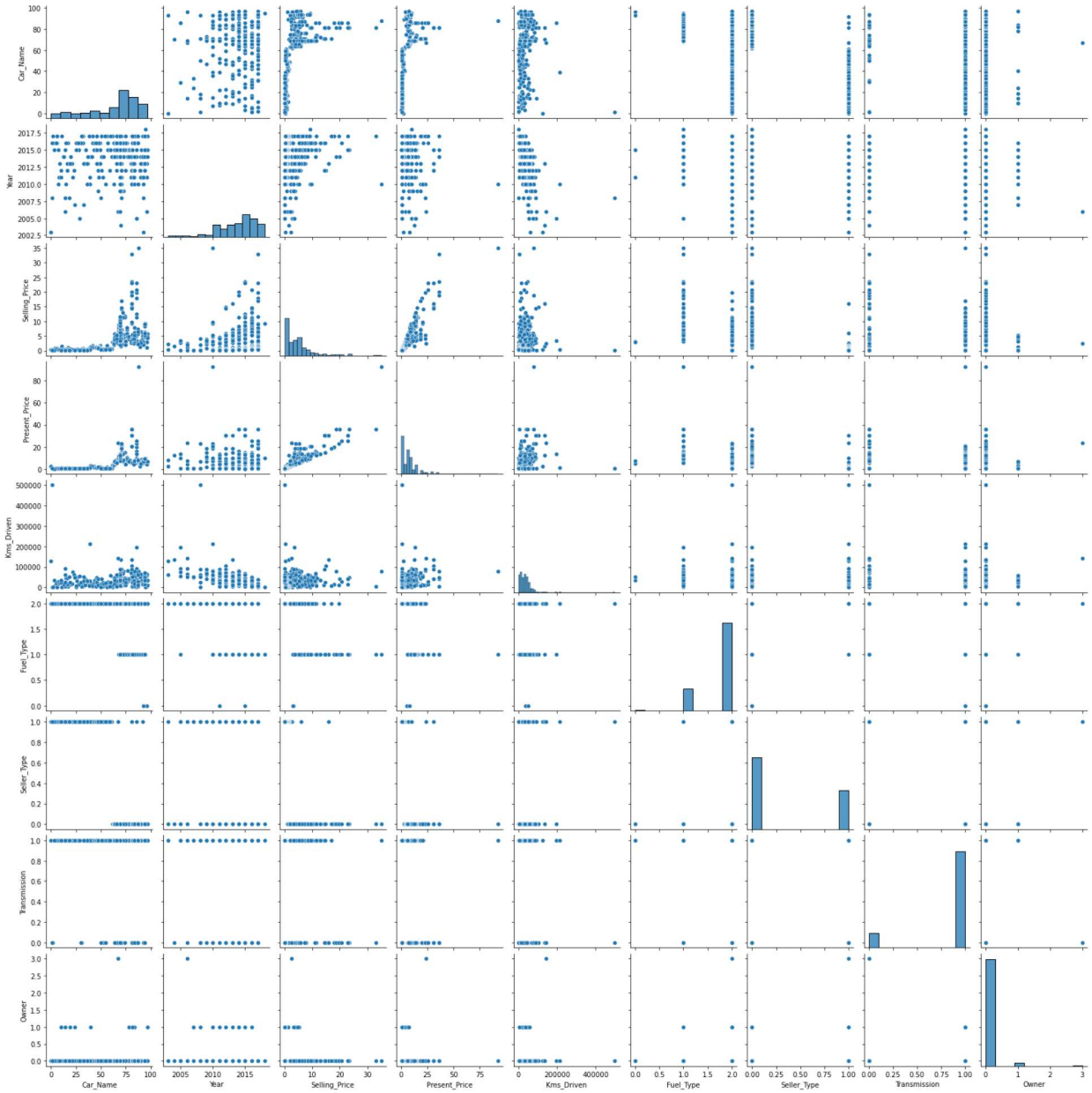



In [14]:
```
sns.distplot(x=df.Fuel_Type)
```

Out[14]:
```
<AxesSubplot:ylabel='Density'>
```



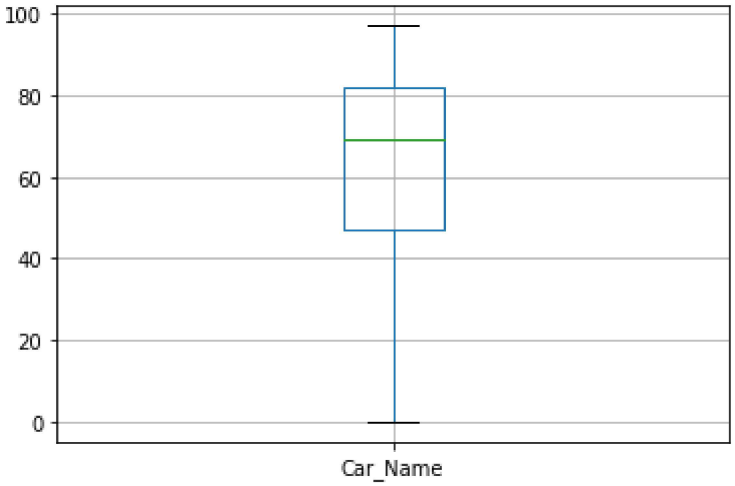

In [15]:
```
#pair plot
sns.pairplot(df)
plt.show()
```

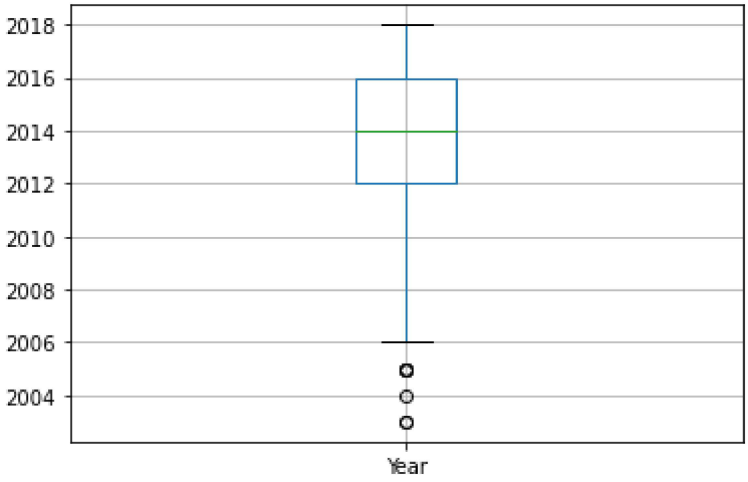


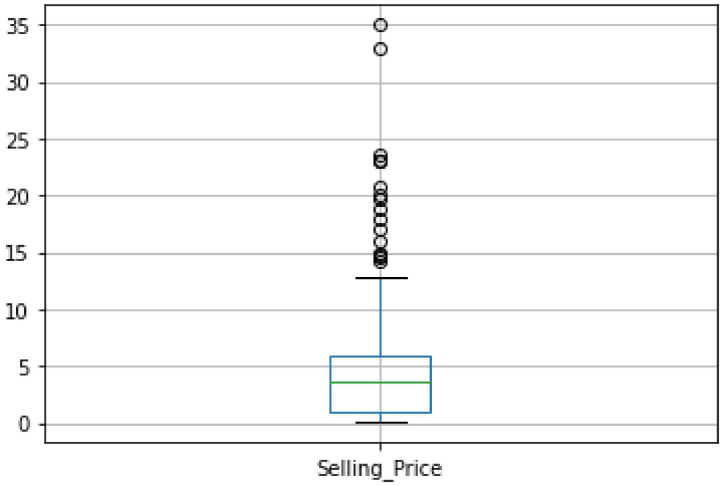

## Outliers Detection

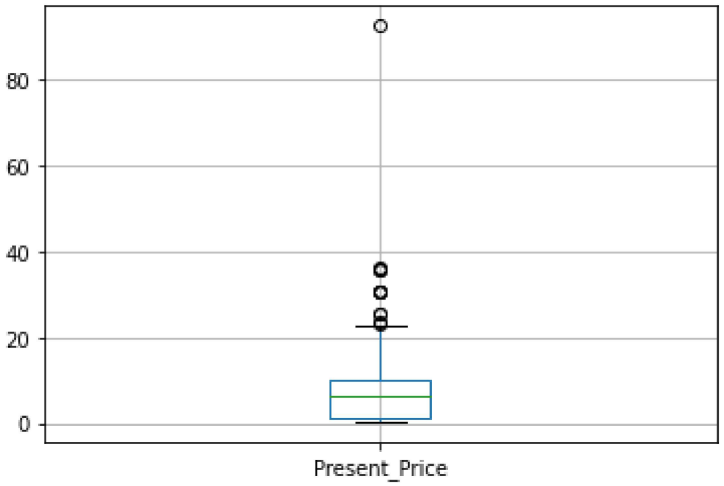
Box Plot for visualization the outliers in the dataset

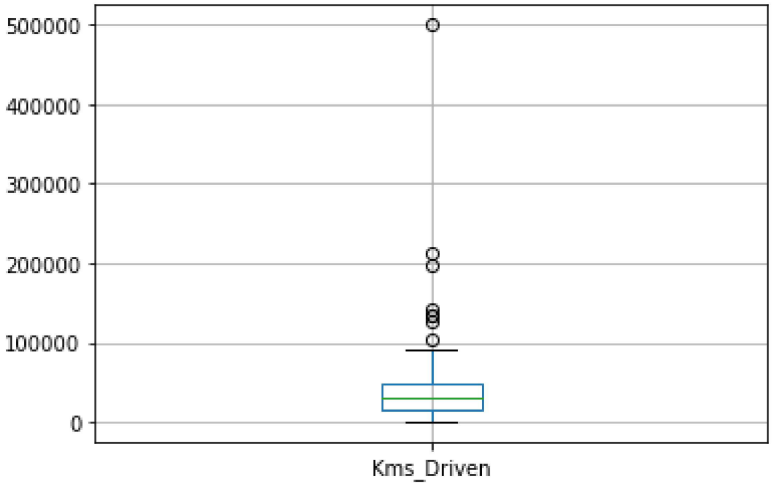
In [16]:
```
for column in df:
    plt.figure()
    df.boxplot([column])
```

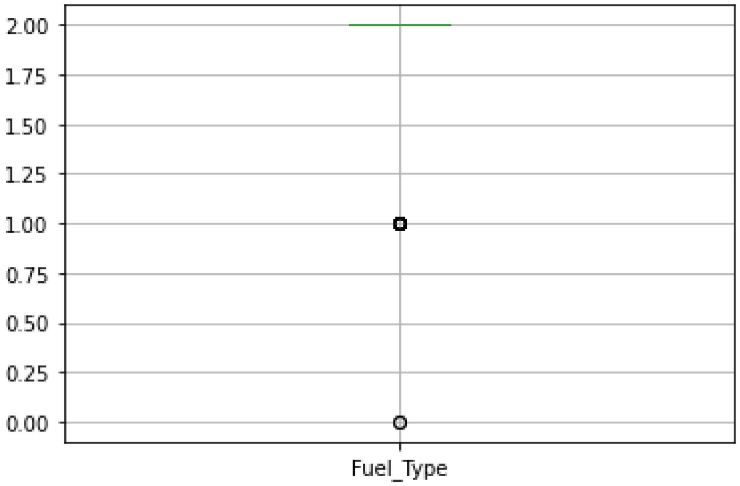

100
80
60
40
20
0
Car_Name

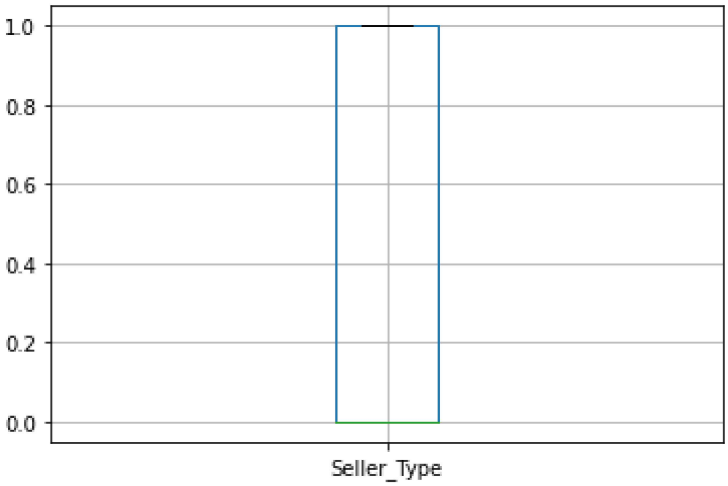

2018
2016
2014
2012
2010
2008
2006
2004
Year

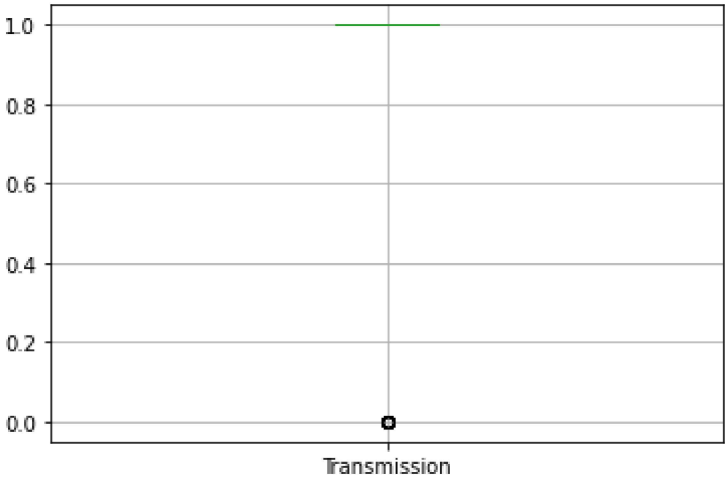

35
30
25
20
15
10
5
0
Selling_Price

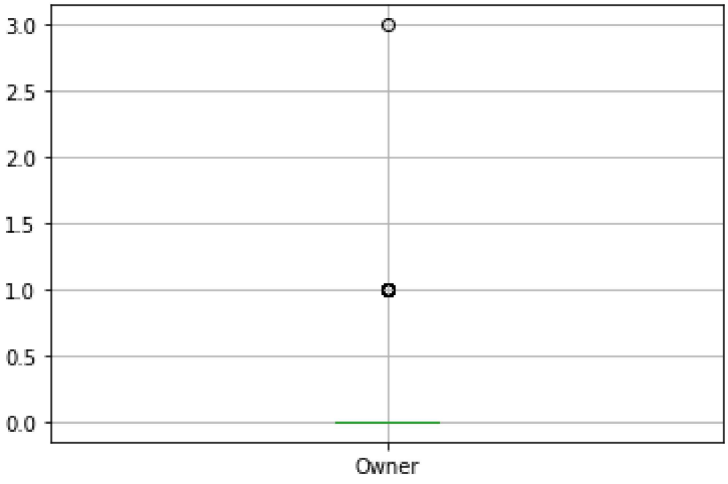

80
60
40
20
0
Present_Price


500000
400000
300000
200000
100000
0
Kms_Driven


2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
Fuel_Type




As we can see here that most of the skewed features have Outliers

## Removal Outliers

In [17]:
```
from scipy.stats import zscore
z=np.abs(zscore(df))
z.head()
```

Out[17]:

| | Car_Name | Year | Selling_Price | Present_Price | Kms_Driven | Fuel_Type | Seller_Type | Transmissio |
|---|---|---|---|---|---|---|---|---|
| **0** | 1.074323 | 0.128897 | 0.258416 | 0.236215 | 0.256224 | 0.500183 | 0.737285 | 0.3914 |
| **1** | 1.191828 | 0.217514 | 0.017481 | 0.221505 | 0.155911 | 1.852241 | 0.737285 | 0.3914 |
| **2** | 0.212627 | 1.168129 | 0.510154 | 0.257427 | 0.773969 | 0.500183 | 0.737285 | 0.3914 |
| **3** | 1.309332 | 0.910335 | 0.356950 | 0.403079 | 0.817758 | 0.500183 | 0.737285 | 0.3914 |
| **4** | 1.152659 | 0.128897 | 0.012079 | 0.087890 | 0.141743 | 1.852241 | 0.737285 | 0.3914 |

In [18]:
```
threshold=3
print(np.where(z>3))
```

```
(array([ 18,  35,  37,  39,  51,  58,  59,  59,  62,  63,  63,  64,  64,
        77,  82,  84,  85,  86,  86,  93,  96, 106, 179, 184, 191, 192,
       193, 196, 198, 201, 205, 241], dtype=int64), array([5, 5, 1, 1, 2, 8, 2, 3, 3,
2, 3, 2, 3, 1, 2, 4, 8, 2, 3, 2, 2, 8,
       4, 8, 8, 8, 8, 4, 8, 8, 8, 8], dtype=int64))
```

In [19]:
```
Q1=df.quantile(0.25)
Q2=df.quantile(0.75)
```

In [20]:
```
Q1
```

Out[20]:
```
Car_Name            47.0
Year              2012.0
Selling_Price        0.9
Present_Price        1.2
Kms_Driven       15000.0
Fuel_Type            2.0
Seller_Type          0.0
Transmission         1.0
Owner                0.0
Name: 0.25, dtype: float64
```

In [21]:
```
Q2
```

Out[21]:
```
Car_Name            82.0
Year              2016.0
Selling_Price        6.0
Present_Price        9.9
Kms_Driven       48767.0
Fuel_Type            2.0
Seller_Type          1.0
Transmission         1.0
Owner                0.0
Name: 0.75, dtype: float64
```

In [22]:
```
IQR=Q2-Q1
IQR
```

Out[22]:
```
Car_Name            35.0
Year                 4.0
Selling_Price        5.1
Present_Price        8.7
Kms_Driven       33767.0
Fuel_Type            0.0
Seller_Type          1.0
Transmission         0.0
Owner                0.0
dtype: float64
```

In [23]:
```
#Here remove the outliers
df_new=df[(z<3).all(axis=1)]
df_new.head()
```

Out[23]:

| | Car_Name | Year | Selling_Price | Present_Price | Kms_Driven | Fuel_Type | Seller_Type | Transmission | C |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 90 | 2014 | 3.35 | 5.59 | 27000 | 2 | 0 | 1 | |
| **1** | 93 | 2013 | 4.75 | 9.54 | 43000 | 1 | 0 | 1 | |
| **2** | 68 | 2017 | 7.25 | 9.85 | 6900 | 2 | 0 | 1 | |
| **3** | 96 | 2011 | 2.85 | 4.15 | 5200 | 2 | 0 | 1 | |
| **4** | 92 | 2014 | 4.60 | 6.87 | 42450 | 1 | 0 | 1 | |

In [24]:
```
#Check the old dataset
df.shape
```

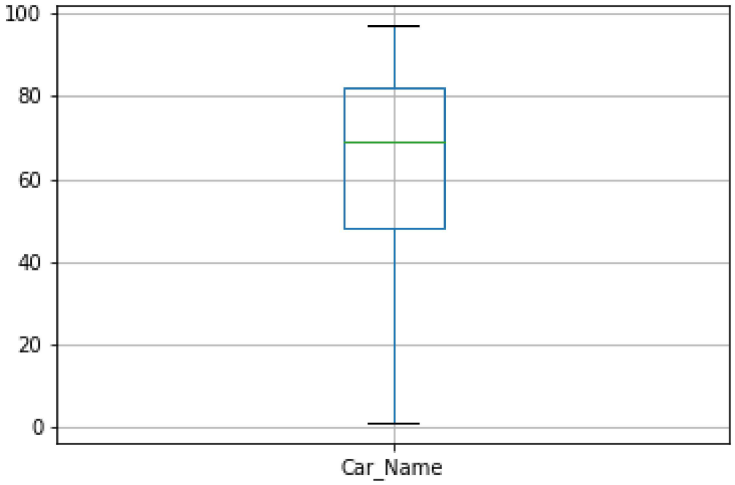
Out[24]:
```
(301, 9)
```

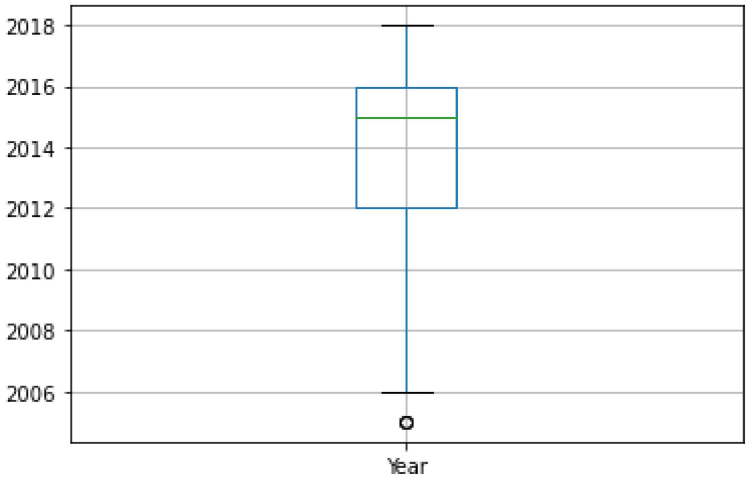
In [25]:
```
#After outliers removal datasets
df_new.shape
```

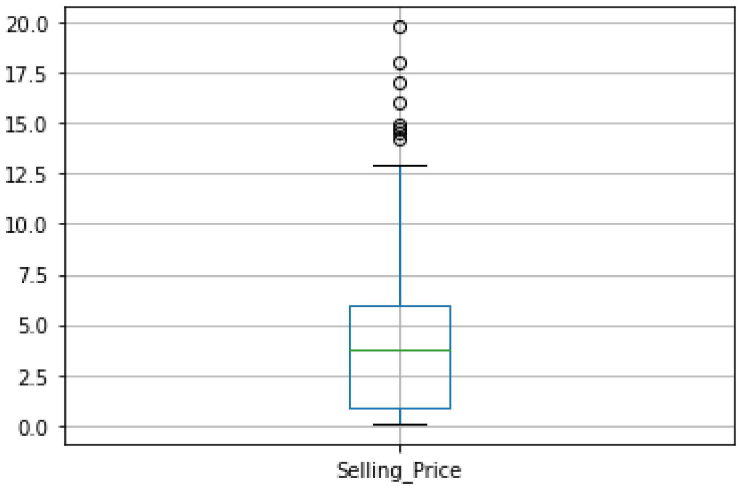
Out[25]:
```
(273, 9)
```

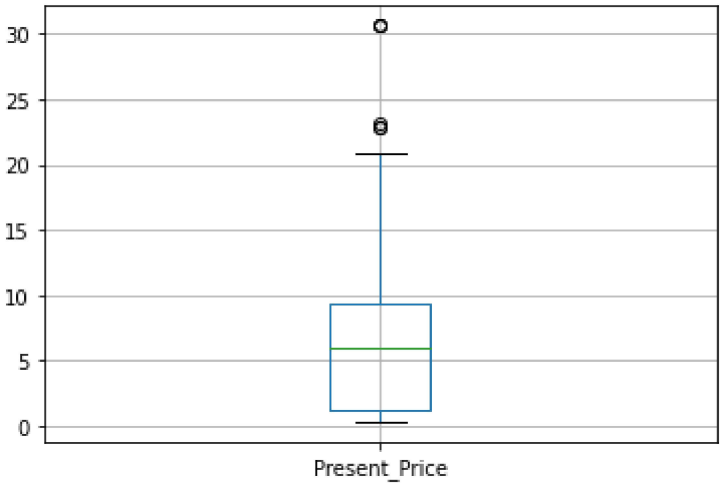
## After removal outliers it will check again with help of boxplot

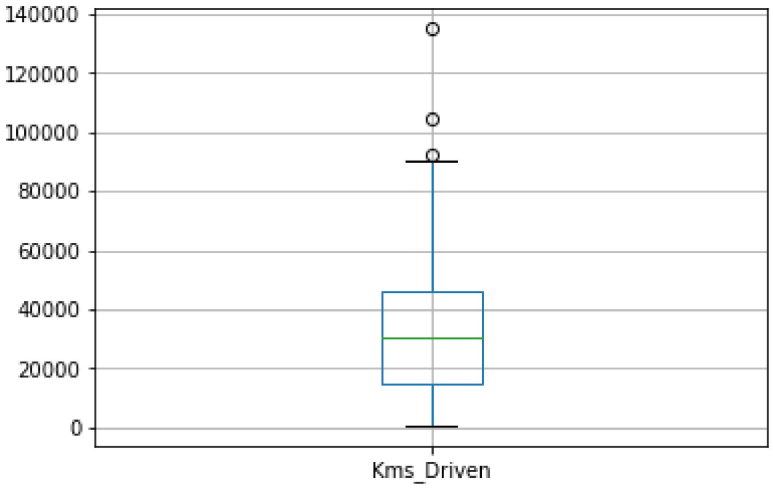
In [26]:
```
for column in df_new:
    plt.figure()
    df_new.boxplot([column])
```

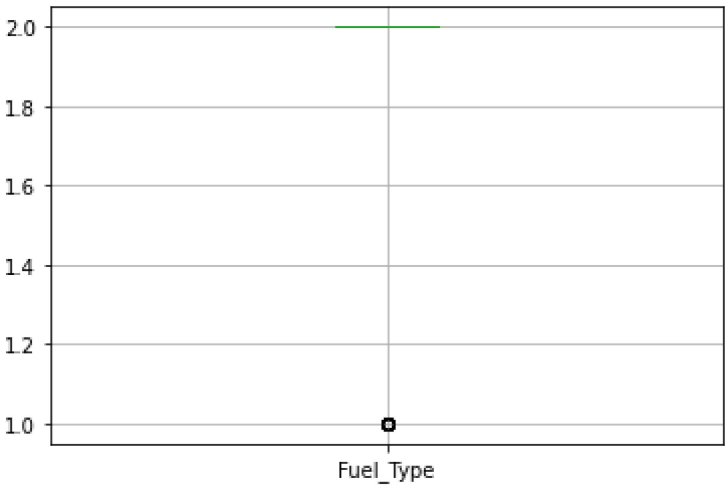

100
80
60
40
20
0
Car_Name

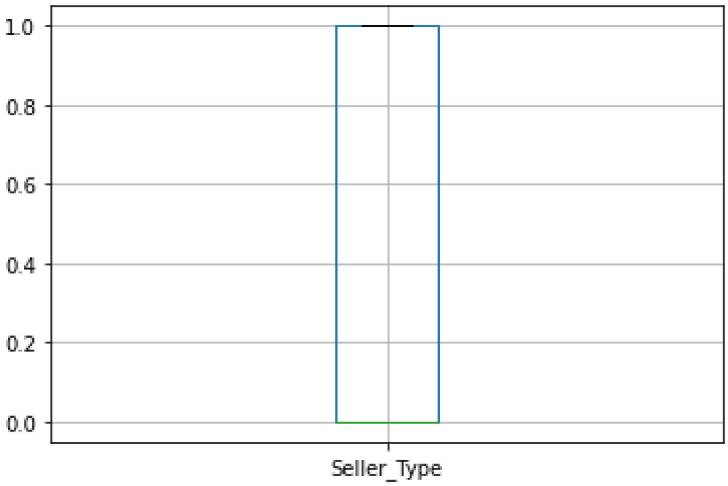

2018
2016
2014
2012
2010
2008
2006
Year

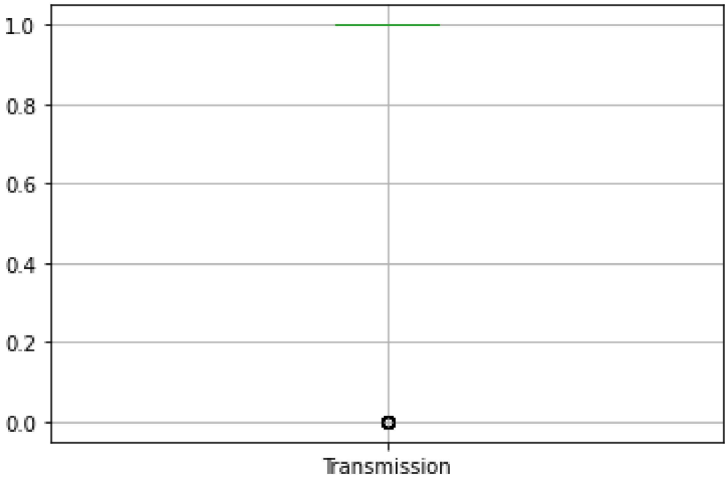

20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
Selling_Price

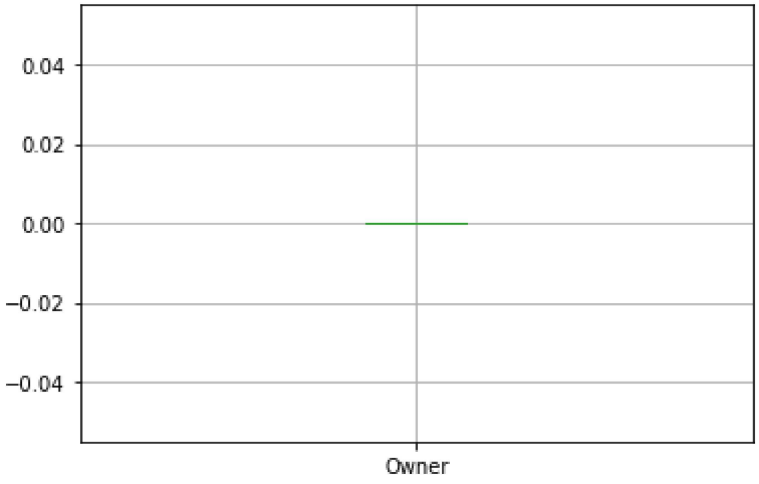

30
25
20
15
10
5
0
Present_Price


140000
120000
100000
80000
60000
40000
20000
0
Kms_Driven


2.0
1.8
1.6
1.4
1.2
1.0
Fuel_Type



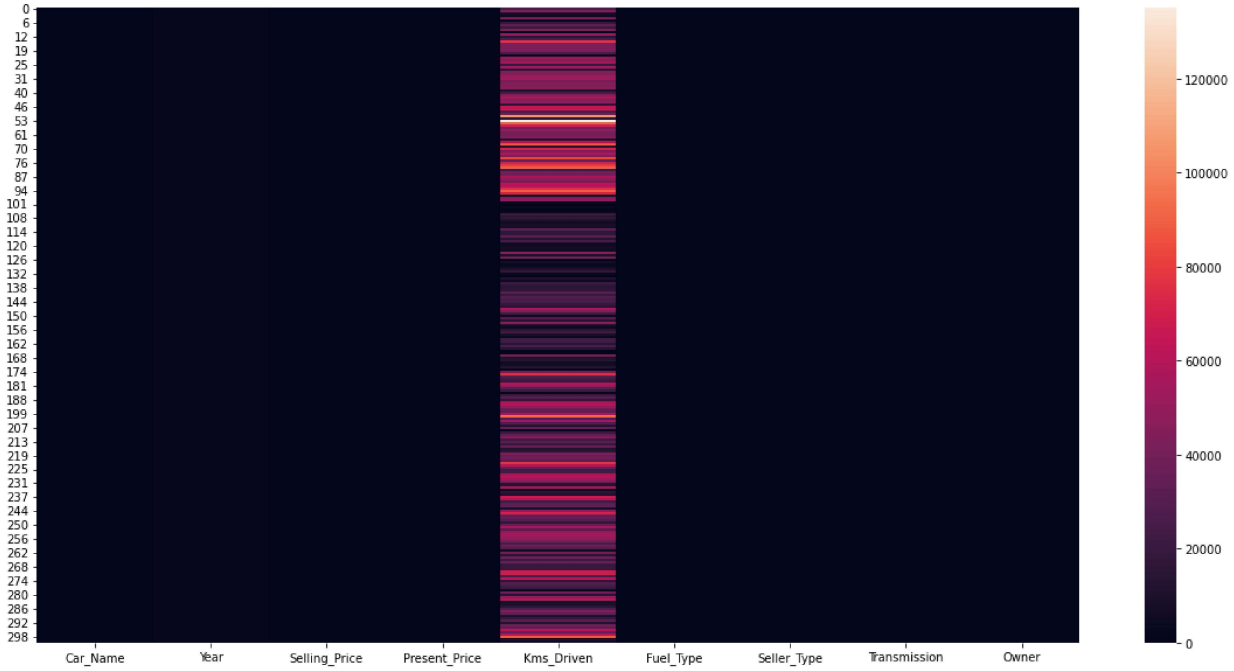




In [ ]:

## Correlation Matrix

In [27]:
```
#Contructing a heat mat to visualize the correlation matrix
plt.figure(figsize=(20,10))
sns.heatmap(df_new)
```

Out[27]: <AxesSubplot:>




In [ ]:

## Splitting the data and Target

In [28]:
```
x=df_new.drop(['Car_Name','Selling_Price'],axis=1)
y=df_new['Selling_Price']
```

In [29]:
```
print(x)
```

```
     Year  Present_Price  Kms_Driven  Fuel_Type  Seller_Type  Transmission  \
0    2014           5.59       27000          2            0             1
1    2013           9.54       43000          1            0             1
2    2017           9.85        6900          2            0             1
3    2011           4.15        5200          2            0             1
4    2014           6.87       42450          1            0             1
..    ...            ...         ...        ...          ...           ...
296  2016          11.60       33988          1            0             1
297  2015           5.90       60000          2            0             1
298  2009          11.00       87934          2            0             1
299  2017          12.50        9000          1            0             1
300  2016           5.90        5464          2            0             1

     Owner
0        0
1        0
2        0
3        0
4        0
..     ...
296      0
297      0
298      0
299      0
300      0

[273 rows x 7 columns]
```

## Splitting Training and Test Data

In [30]:
```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.1,random_state=2)
```

## Model Training

In [31]:
```
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
lr.fit(x_train,y_train)
```

Out[31]:
```
LinearRegression()
```

## Model Evaluation

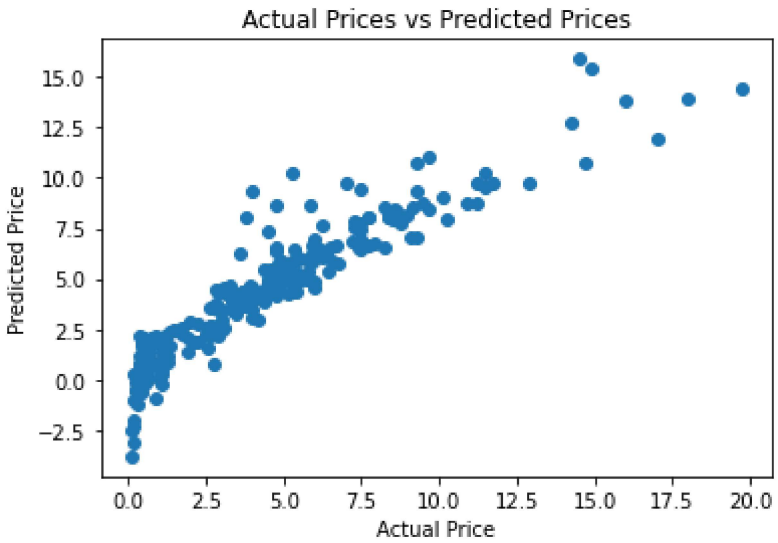
In [32]:
```
#Prediction training data
training_data_prediction=lr.predict(x_train)
```

In [33]:
```
#R squared Error
error_score=metrics.r2_score(y_train,training_data_prediction)
print("R squared Error:",error_score)
```

```
R squared Error: 0.8711381915352807
```

## Visualize the actual price and predicted price

In [34]:
```
plt.scatter(y_train,training_data_prediction)
plt.xlabel('Actual Price')
plt.ylabel('Predicted Price')
plt.title('Actual Prices vs Predicted Prices')
plt.show()
```




## Lasso Regression

In [35]:
```
#Loading the Lasso Regression model
lss_reg_model=Lasso()
lss_reg_model.fit(x_train,y_train)
lss_reg_model.score(x_test,y_test)
```

Out[35]:
```
0.7466375308725579
```

## Model Saving

In [36]:
```
import pickle
filename='Car Price Prediction.pkl'
pickle.dump(lr, open(filename,'wb'))
```

## conclusion

In [37]:
```
a=np.array(y_test)
predicted=np.array(lr.predict(x_test))
df.com=pd.DataFrame({"original":a,"predicted":predicted},index=range(len(a)))
df.com
```

Out[37]:

| | original | predicted |
|---|---|---|
| **0** | 5.50 | 6.969513 |
| **1** | 6.50 | 6.934783 |
| **2** | 3.51 | 4.035335 |
| **3** | 6.10 | 5.320525 |
| **4** | 12.50 | 9.279523 |
| **5** | 4.50 | 5.015481 |
| **6** | 0.52 | -1.174965 |
| **7** | 0.75 | 1.007348 |
| **8** | 6.95 | 9.896061 |
| **9** | 0.35 | 1.220810 |
| **10** | 0.27 | 0.155501 |
| **11** | 3.00 | 4.055080 |
| **12** | 0.75 | 1.606873 |
| **13** | 2.95 | 1.769353 |
| **14** | 1.65 | 2.486920 |
| **15** | 2.85 | 2.927247 |
| **16** | 0.65 | 0.749767 |
| **17** | 0.51 | 1.189541 |
| **18** | 3.10 | 4.193842 |
| **19** | 4.50 | 4.559531 |
| **20** | 0.38 | -1.822236 |
| **21** | 0.45 | 0.314438 |
| **22** | 1.05 | 1.334143 |
| **23** | 0.40 | 0.979408 |
| **24** | 6.85 | 7.333231 |
| **25** | 11.25 | 8.488396 |
| **26** | 5.65 | 5.637110 |
| **27** | 2.35 | 1.896135 |

In [ ]: