Untitled

June 18, 2022

[1]:
```
### IMPORT: ----------------------------------
import scipy.stats as stats
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import warnings
warnings.filterwarnings('ignore') # To supress warnings
 # set the background for the graphs
from scipy.stats import skew
plt.style.use('ggplot')
from sklearn.model_selection import train_test_split # Sklearn package's␣
 ↪randomized data splitting function
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OneHotEncoder
import math
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
pd.set_option('display.float_format', lambda x: '%.3f' % x)
pd.set_option('display.max_rows', 300)
pd.set_option('display.max_colwidth',400)
pd.set_option('display.float_format', lambda x: '%.5f' % x) # To supress␣
 ↪numerical display in scientific notations
import statsmodels.api as sm
print("Load Libraries- Done")
```

```
Load Libraries- Done
```

[2]:
```
# importing dataset
data=pd.read_csv('CAR DETAILS FROM CAR DEKHO.csv')
```

[3]:
```
data.head(5)
```

[3]:
```
                       name  year  selling_price  km_driven    fuel  \
0             Maruti 800 AC  2007          60000      70000  Petrol
1  Maruti Wagon R LXI Minor  2007         135000      50000  Petrol
```

```
2       Hyundai Verna 1.6 SX  2012         600000     100000  Diesel
3     Datsun RediGO T Option  2017         250000      46000  Petrol
4      Honda Amaze VX i-DTEC  2014         450000     141000  Diesel

  seller_type transmission         owner
0  Individual       Manual   First Owner
1  Individual       Manual   First Owner
2  Individual       Manual   First Owner
3  Individual       Manual   First Owner
4  Individual       Manual  Second Owner
```

1 assumptions

- normality: our data is looking to be normal, but to make sure, we will create visuals of every independent variable.
- linearity: the data is supposed to be linear,though we will remove some outliers if present
- Multicollinearity: as the data set is about the price of houses against the features of house, the Multicollinearity is not supposed to be present as the dependent variable follows different independent variables
- Autocorrelation: we are assuming the this is not present in the data

[4]:
```
data.isna().sum()
```

[4]:
```
name             0
year             0
selling_price    0
km_driven        0
fuel             0
seller_type      0
transmission     0
owner            0
dtype: int64
```

[5]:
```
data.corr()
```

[5]:
```
                  year  selling_price  km_driven
year           1.00000        0.41392   -0.41969
selling_price  0.41392        1.00000   -0.19229
km_driven     -0.41969       -0.19229    1.00000
```

[6]:
```
# Making a list of all categorical variables
cat_col = [
    "fuel",
    "seller_type",
    "transmission",
    "year",
    "owner",
```

```python
]
# Printing number of count of each unique value in each column
for column in cat_col:
    print(data[column].value_counts())
    print("#" * 40)
```

```
Diesel      2153
Petrol      2123
CNG           40
LPG           23
Electric       1
Name: fuel, dtype: int64
########################################
Individual          3244
Dealer               994
Trustmark Dealer     102
Name: seller_type, dtype: int64
########################################
Manual       3892
Automatic     448
Name: transmission, dtype: int64
########################################
2017    466
2015    421
2012    415
2013    386
2014    367
2018    366
2016    357
2011    271
2010    234
2019    195
2009    193
2008    145
2007    134
2006    110
2005     85
2020     48
2004     42
2003     23
2002     21
2001     20
1998     12
2000     12
1999     10
1997      3
```

```
1996      2
1995      1
1992      1
Name: year, dtype: int64
########################################
First Owner             2832
Second Owner            1106
Third Owner              304
Fourth & Above Owner      81
Test Drive Car            17
Name: owner, dtype: int64
########################################
```

2 Maximum car being sold have fuel type as Diesel.

- Mumbai has highest numbers of car availabe for purchase.
- Most of the cars are 5 seaters and First owned.
- Years of car ranges form 1996- 2015

[7]:
```
# data processsing
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4340 entries, 0 to 4339
Data columns (total 8 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   name           4340 non-null   object
 1   year           4340 non-null   int64
 2   selling_price  4340 non-null   int64
 3   km_driven      4340 non-null   int64
 4   fuel           4340 non-null   object
 5   seller_type    4340 non-null   object
 6   transmission   4340 non-null   object
 7   owner          4340 non-null   object
dtypes: int64(3), object(5)
memory usage: 271.4+ KB
```

[8]:
```
data[['fuel','seller_type','transmission','owner']].sample(10)
```

[8]:

	fuel	seller_type	transmission	owner
709	Petrol	Dealer	Manual	First Owner
2930	Petrol	Individual	Automatic	Second Owner
4316	Diesel	Individual	Manual	First Owner
1740	Petrol	Individual	Manual	Second Owner
539	Diesel	Individual	Automatic	Second Owner
3485	Petrol	Individual	Manual	First Owner

```
977   Diesel  Individual       Manual   First Owner
4296  Petrol      Dealer       Manual   First Owner
662   Petrol  Individual       Manual   First Owner
3448  Diesel  Individual       Manual   First Owner
```

[9]:
```python
data["fuel"] = data["fuel"].astype("category")
data["seller_type"] = data["seller_type"].astype("category")
data["owner"] = data["owner"].astype("category")
data["transmission"] = data["transmission"].astype("category")
```

[10]:
```python
data.describe().T
```

[10]:
```
                   count         mean          std         min          25%  \
year           4340.00000   2013.09078      4.21534  1992.00000   2011.00000
selling_price  4340.00000 504127.31175 578548.73614 20000.00000 208749.75000
km_driven      4340.00000  66215.77742  46644.10219     1.00000  35000.00000

                       50%          75%           max
year            2014.00000   2016.00000    2020.00000
selling_price 350000.00000 600000.00000 8900000.00000
km_driven      60000.00000  90000.00000  806599.00000
```

[11]:
```python
data['Current_year']=2021
data['Ageofcar']=data['Current_year']-data['year']
data.drop('Current_year',axis=1,inplace=True)
data.head()
```

[11]:
```
                       name  year  selling_price  km_driven    fuel  \
0             Maruti 800 AC  2007          60000      70000  Petrol
1  Maruti Wagon R LXI Minor  2007         135000      50000  Petrol
2      Hyundai Verna 1.6 SX  2012         600000     100000  Diesel
3    Datsun RediGO T Option  2017         250000      46000  Petrol
4     Honda Amaze VX i-DTEC  2014         450000     141000  Diesel

  seller_type transmission         owner  Ageofcar
0  Individual       Manual   First Owner        14
1  Individual       Manual   First Owner        14
2  Individual       Manual   First Owner         9
3  Individual       Manual   First Owner         4
4  Individual       Manual  Second Owner         7
```

[12]:
```python
#As mentioned in dataset car name has Brand and model so extracting it ,This␣
↪can help to fill missing values of price column as brand
data['Brand'] = data['name'].str.split(' ').str[0] #Separating Brand name from␣
↪the Name
data['Model'] = data['name'].str.split(' ').str[1] + data['name'].str.split('␣
↪').str[2]
```

[13]:
```
x=data.Brand.unique()
```

[14]:
```
data.Brand.nunique()
```

[14]: 29

[15]:
```
y=data.groupby(data.Brand).size().sort_values(ascending =False)
```

[16]:
```
data.Model.isnull().sum()
```

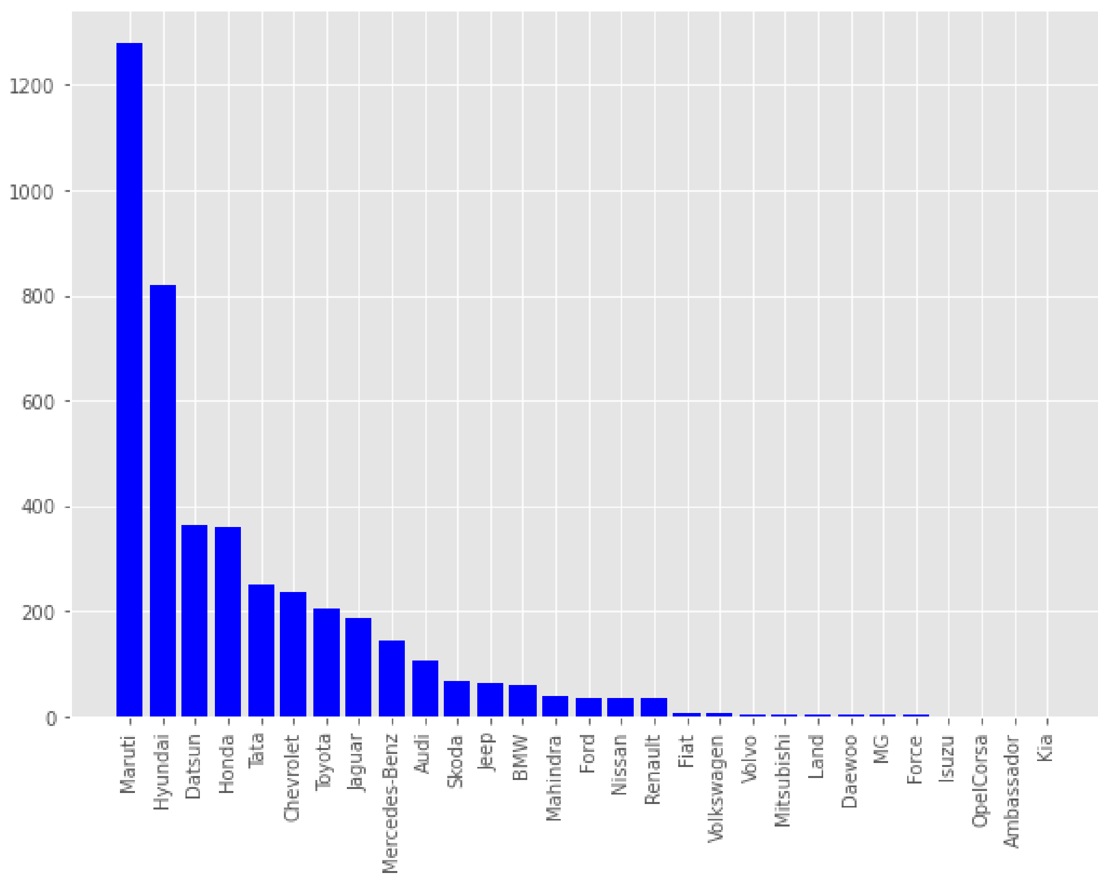
[16]: 1

[17]:
```
data.dropna(subset=['Model'],axis=0,inplace=True)
```

[18]:
```
data.Model.nunique()
```

[18]: 612

[19]:
```
fig, ax = plt.subplots(figsize=(10,7))
ax.bar(x,y, color = 'b', width = 0.8)
plt.xticks(rotation=90)
plt.show()
```



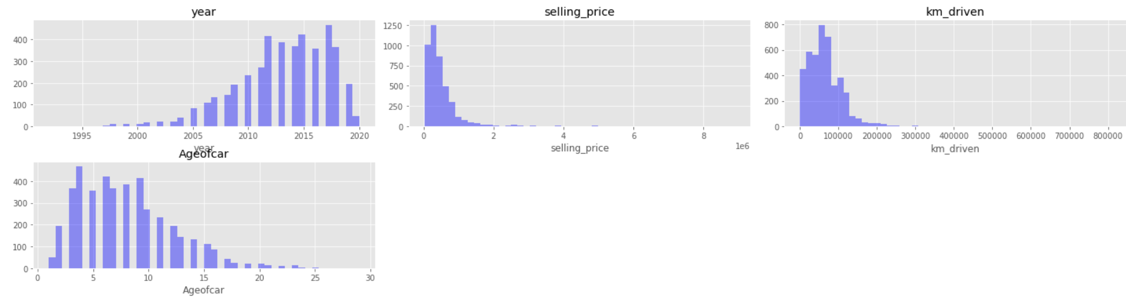
[20]:
```
data.groupby('Model')['Model'].size().nlargest(30)
```

[20]:
```
Model
WagonR          164
SwiftDzire      139
Grandi10        112
Alto800          87
Innova2.5        83
Verna1.6         71
SantroXing       68
IndicaVista      67
AltoLXi          62
SwiftVDI         60
AltoK10          51
AltoLX           47
Creta1.6         46
EONEra           43
BeatDiesel       40
i10Magna         40
800AC            37
FigoDiesel       36
Duster85PS       35
EcoSport1.5      35
i20Asta          32
Cityi            31
XUV500W8         31
VernaCRDi        30
EONMagna         29
Spark1.0         29
BoleroPower      28
KWIDRXT          28
NewSafari        28
ZenEstilo        28
Name: Model, dtype: int64
```

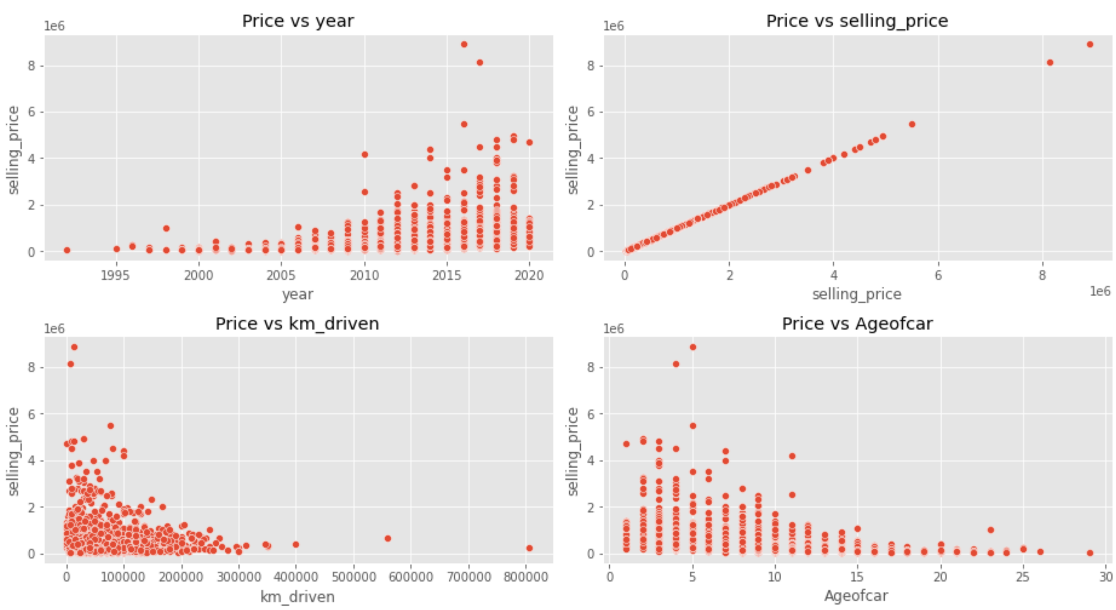
3 wagonR is the most popular model

[21]:
```
plt.style.use('ggplot')
#select all quantitative columns for checking the spread
numeric_columns = data.select_dtypes(include=np.number).columns.tolist()
plt.figure(figsize=(20,25))

for i, variable in enumerate(numeric_columns):
                     plt.subplot(10,3,i+1)
```

```
                    sns.distplot(data[variable],kde=False,color='blue')
                    plt.tight_layout()
                    plt.title(variable)
```



[22]:
```
numeric_columns= numeric_columns = data.select_dtypes(include=np.number).
 ↪columns.tolist()
plt.figure(figsize=(13,17))

for i, variable in enumerate(numeric_columns):
                    plt.subplot(5,2,i+1)
                    sns.scatterplot(x=data[variable],y=data['selling_price']).
 ↪set(title='Price vs '+ variable)
                    #plt.xticks(rotation=90)
                    plt.tight_layout()
```



[23]:
```
# grouping the cars by high ot low profile cars
Low=['Maruti',
     'Hyundai',
     'Ambassdor',
     'Hindustan',
     'Force',
     'Chevrolet',
     'Fiat',
     'Tata',
     'Smart',
     'Renault',
     'Datsun',
     'Mahindra',
     'Skoda',
     'Ford',
     'Toyota',
     'Isuzu',
     'Mitsubishi','Honda','Land', 'Daewoo', 'MG', 'Ambassador', 'Kia',
       'OpelCorsa']
High=['Audi',
      'Mini Cooper',
      'Bentley',
      'Mercedes-Benz',
      'Lamborghini',
      'Volkswagen',
      'Porsche',
      'Land Rover',
      'Nissan',
      'Volvo',
      'Jeep',
      'Jaguar',
      'BMW']# more than 30lakh
```

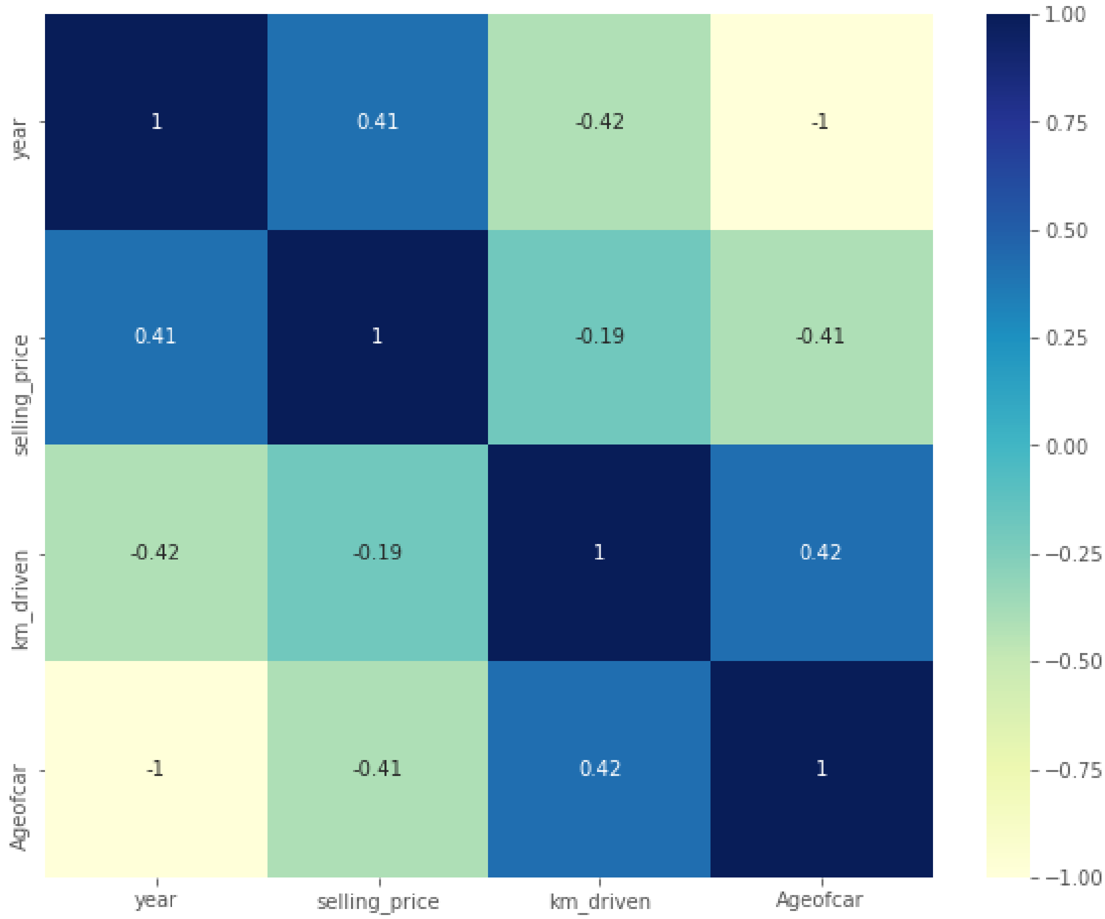
[24]:
```
def classrange(x):
    if x in Low:
        return "Low"
    elif x in High:
        return "High"
    else:
        return x
```

[25]:
```
data['Brand_Class'] = data['Brand'].apply(lambda x: classrange(x))
```

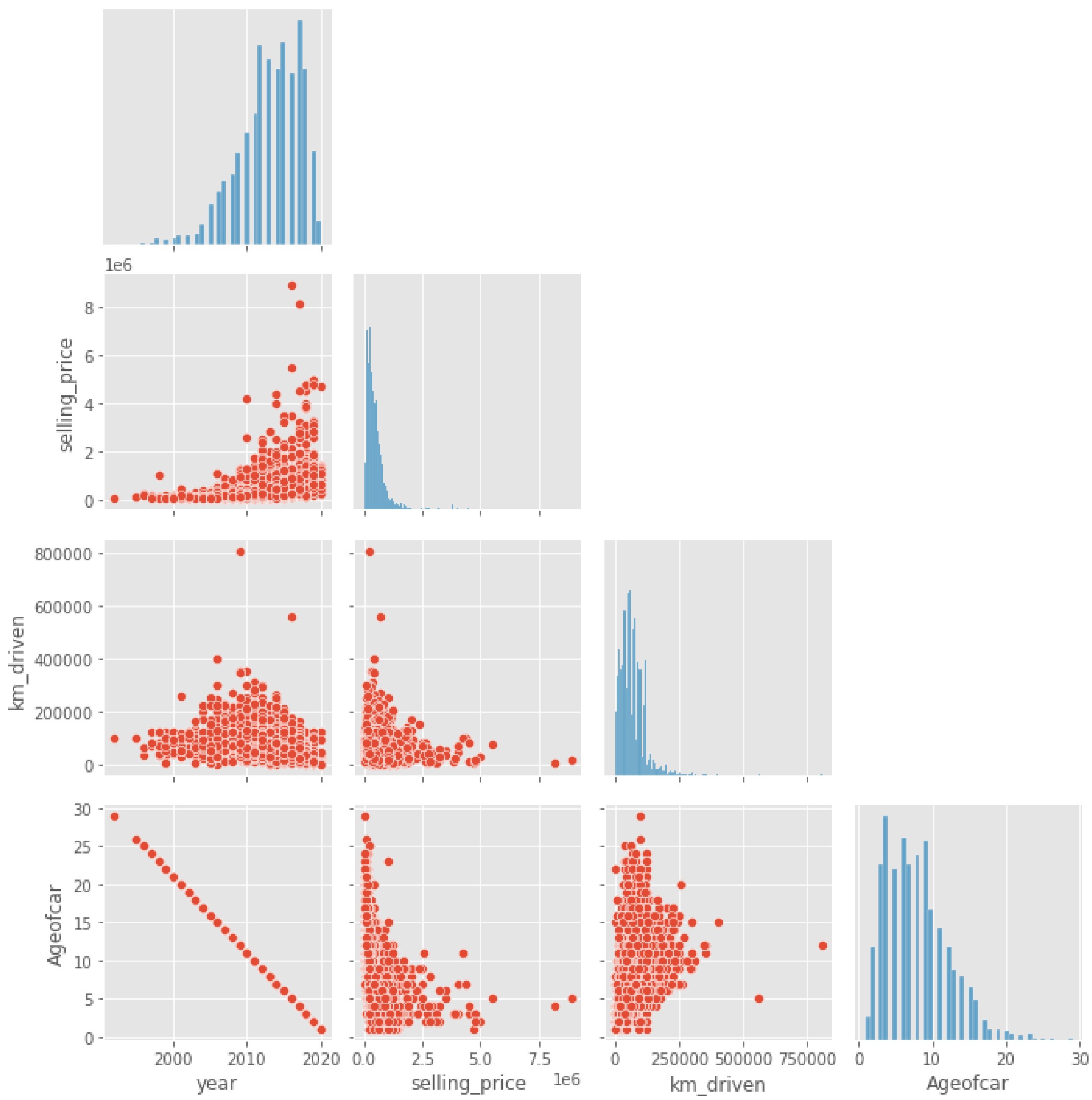
[26]:
```
data['Brand_Class'].unique()
```

[26]:
```
array(['Low', 'High'], dtype=object)
```

[27]:
```
plt.figure(figsize=(10,8))
sns.heatmap(data.corr(),annot=True ,cmap="YlGnBu" )
plt.show()
```

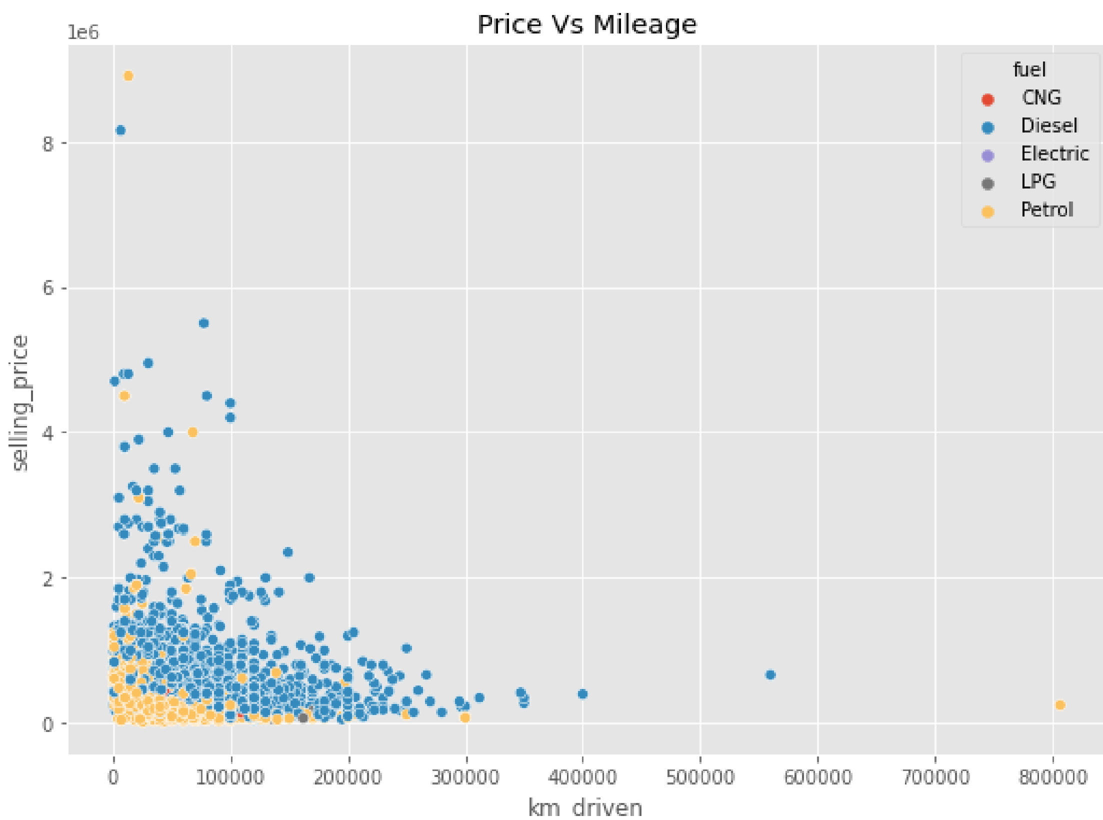


[28]:
```
sns.pairplot(data=data, corner=True)
plt.show()
```

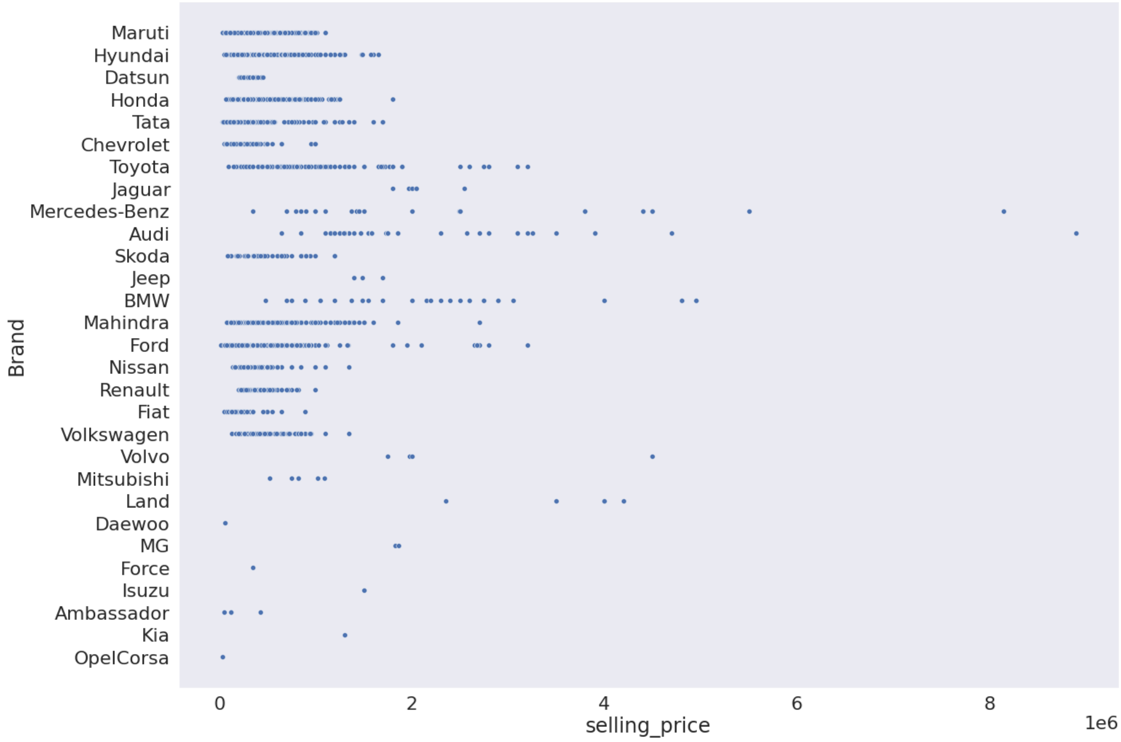


[29]:
```
plt.figure(figsize=(10,7))
plt.title("Price Vs Mileage")
sns.scatterplot(y='selling_price', x='km_driven', hue='fuel', data=data)
```

[29]:
```
<AxesSubplot:title={'center':'Price Vs Mileage'}, xlabel='km_driven',
ylabel='selling_price'>
```

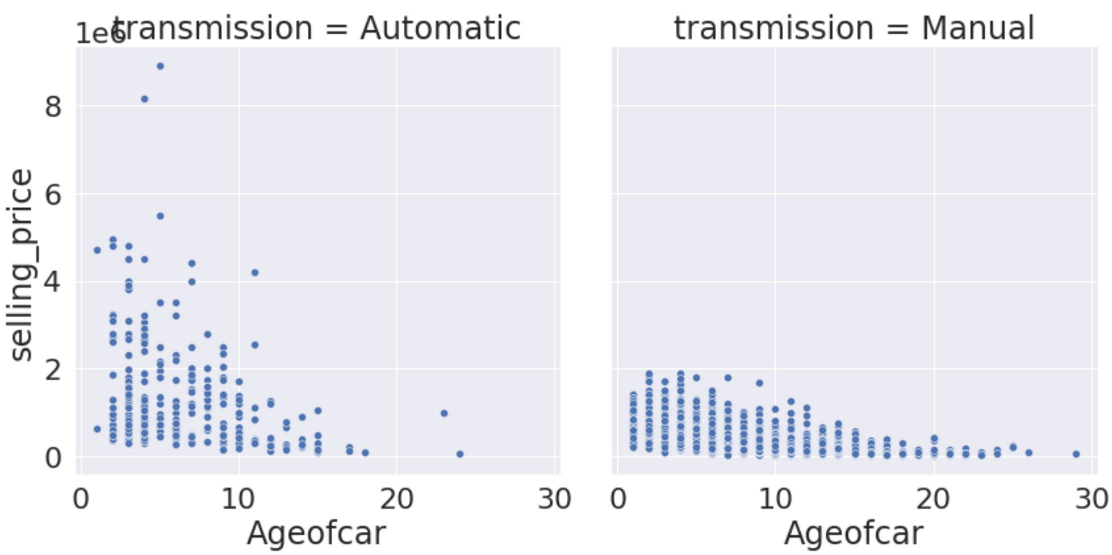


[30]:
```
plt.figure(figsize=(20,15))
sns.set(font_scale=2)
sns.scatterplot(x='selling_price', y='Brand', data=data)
plt.grid()
```

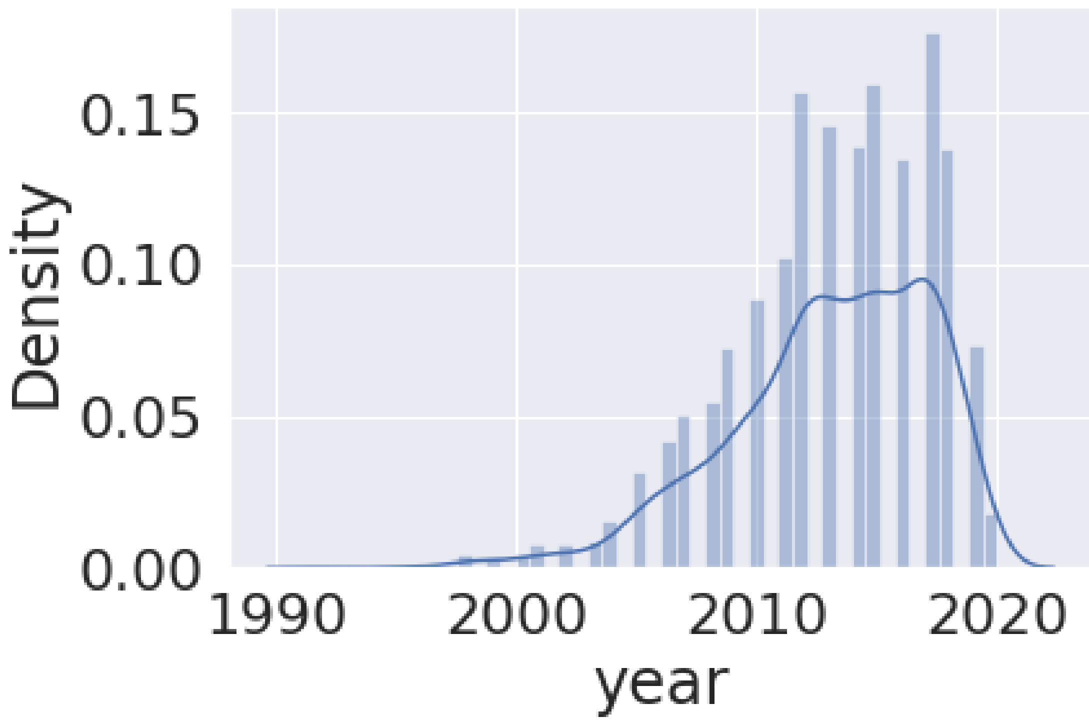


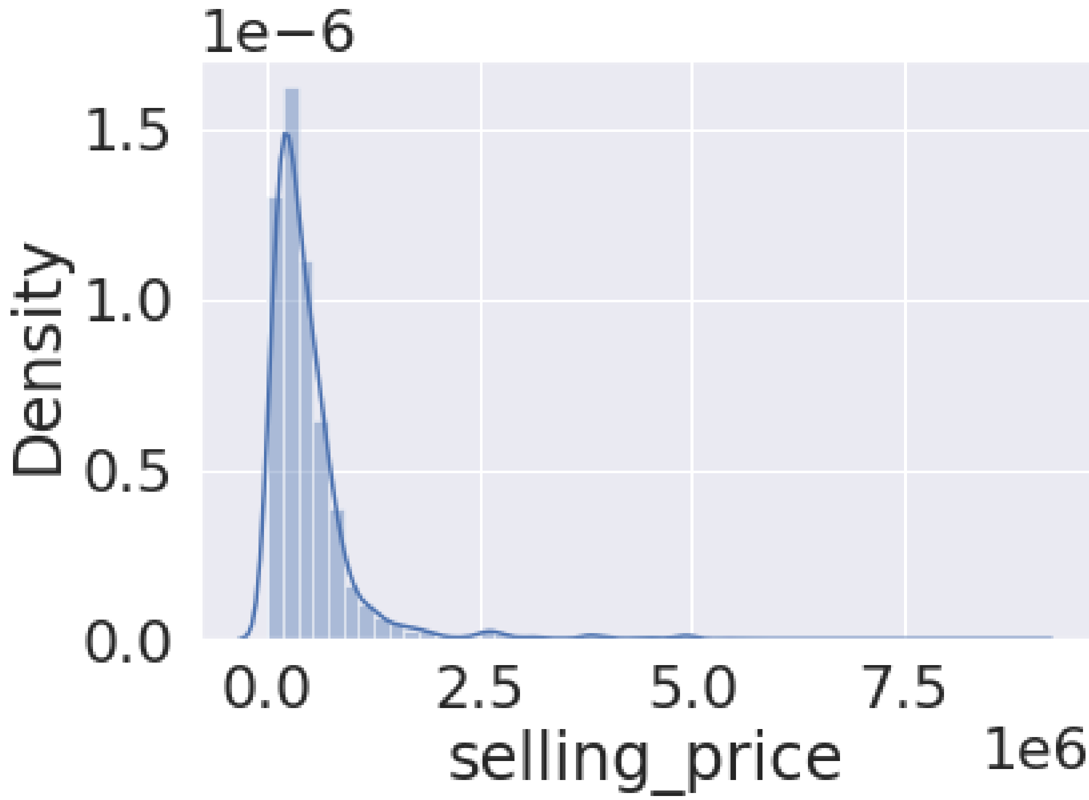
[31]:
```
sns.relplot(data=data,␣
↪y='selling_price',x='Ageofcar',col='transmission',aspect=1,height=6)
```

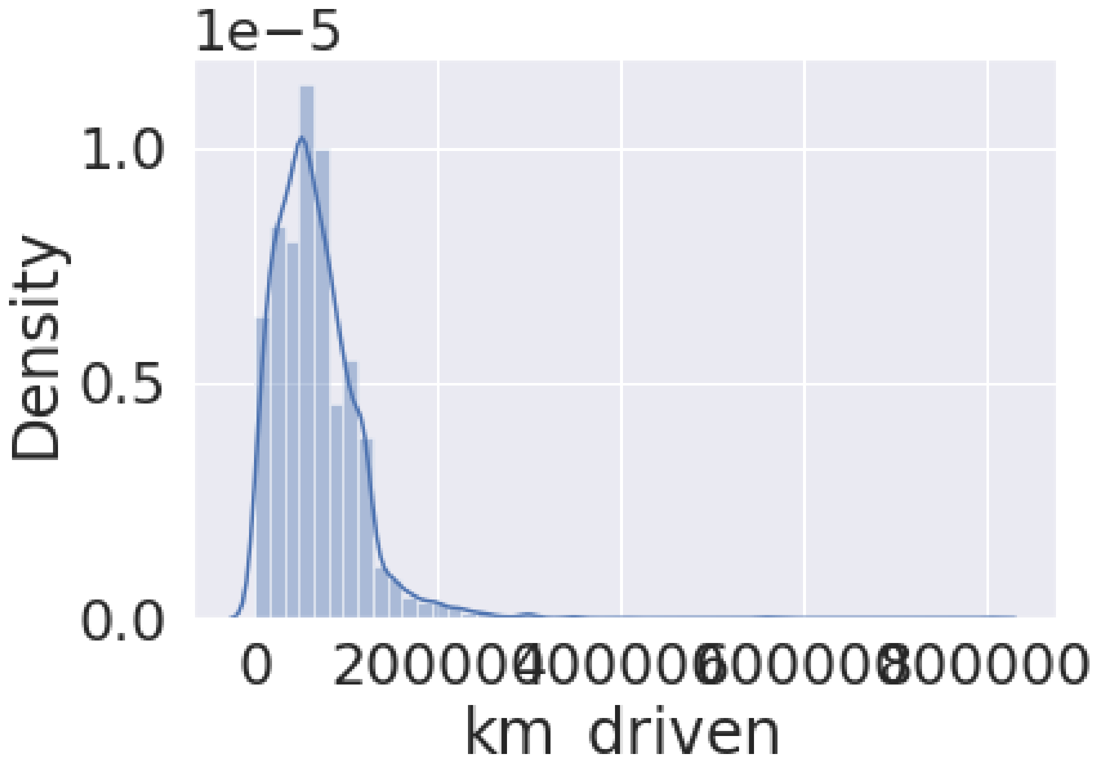
[31]: <seaborn.axisgrid.FacetGrid at 0x7f8d555aa6a0>

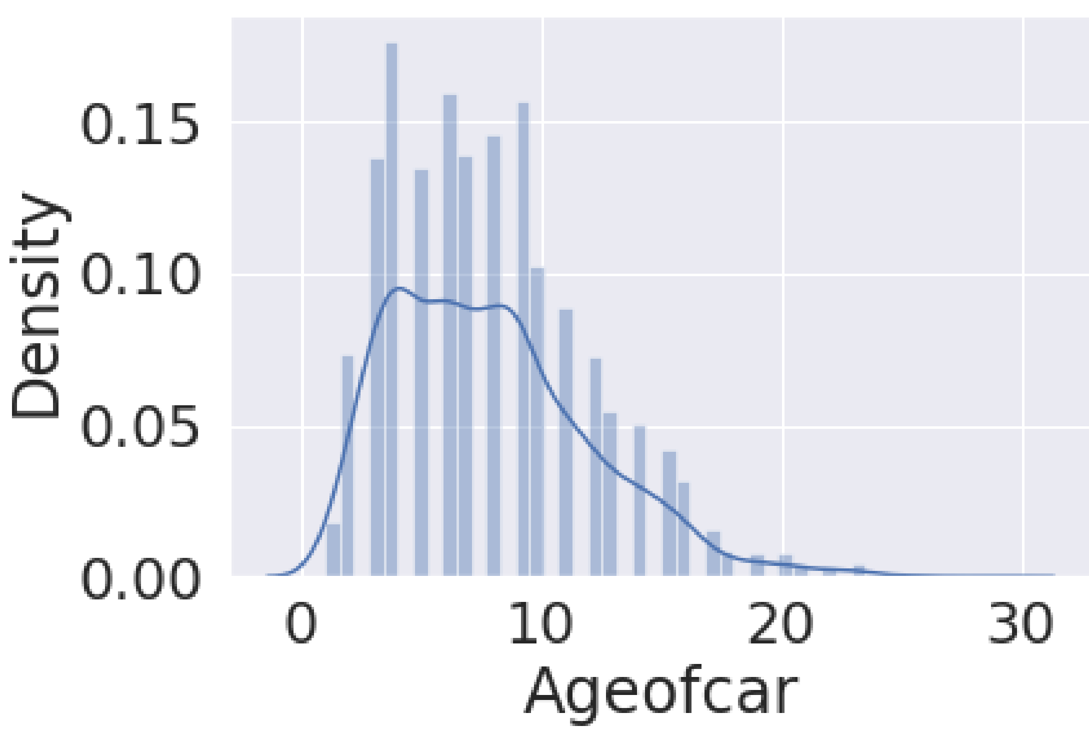


[32]:
```
# check distrubution if skewed. If distrubution is skewed , it is advice to use␣
↪log transform
cols_to_log = data.select_dtypes(include=np.number).columns.tolist()
for colname in cols_to_log:
    sns.distplot(data[colname], kde=True)
    plt.show()
```




1e−6
1.5
1.0
0.5
0.0
Density
0.0
2.5
5.0
7.5
selling_price
1e6


1e−5
1.0
0.5
0.0
Density
0
200000
400000
600000
800000
km_driven


0.15
0.10
0.05
0.00
Density
0
10
20
30
Ageofcar

[33]:
```python
def Perform_log_transform(df,col_log):
    """#Perform Log Transformation of dataframe , and list of columns """
    for colname in col_log:
        df[colname + '_log'] = np.log(df[colname])
    #df.drop(col_log, axis=1, inplace=True)
    df.info()
```

[34]:
```python
Perform_log_transform(data,['km_driven','selling_price'])
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 4339 entries, 0 to 4339
Data columns (total 14 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   name               4339 non-null   object
 1   year               4339 non-null   int64
 2   selling_price      4339 non-null   int64
 3   km_driven          4339 non-null   int64
 4   fuel               4339 non-null   category
 5   seller_type        4339 non-null   category
 6   transmission       4339 non-null   category
 7   owner              4339 non-null   category
 8   Ageofcar           4339 non-null   int64
 9   Brand              4339 non-null   object
 10  Model              4339 non-null   object
 11  Brand_Class        4339 non-null   object
 12  km_driven_log      4339 non-null   float64
 13  selling_price_log  4339 non-null   float64
dtypes: category(4), float64(2), int64(4), object(4)
memory usage: 390.5+ KB
```

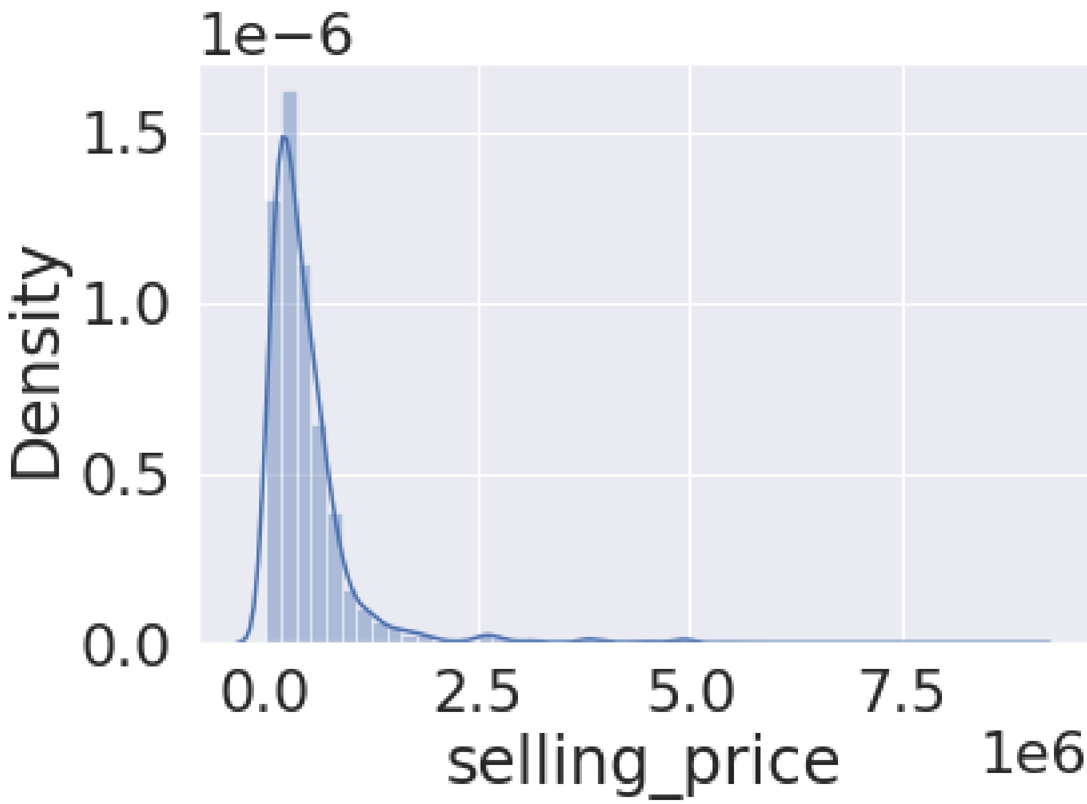
[35]:
```python
data.drop(['name','Model','year','Brand'],axis=1,inplace=True)
```

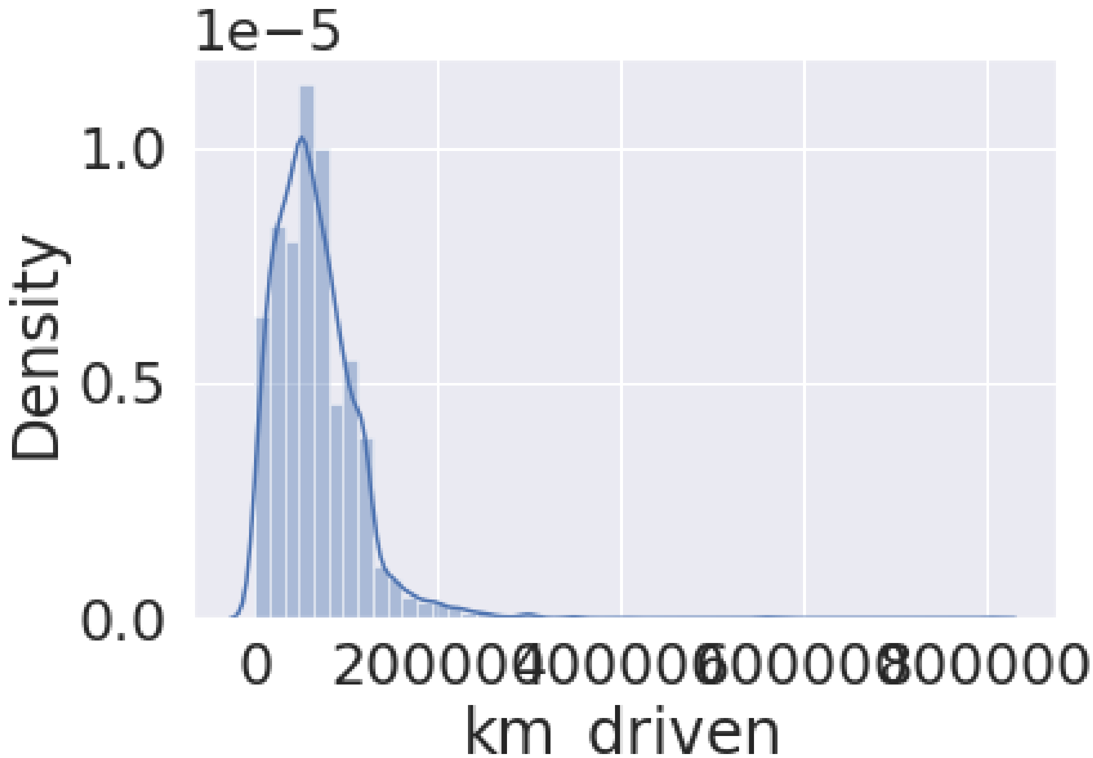
[36]:
```python
data.info()
```

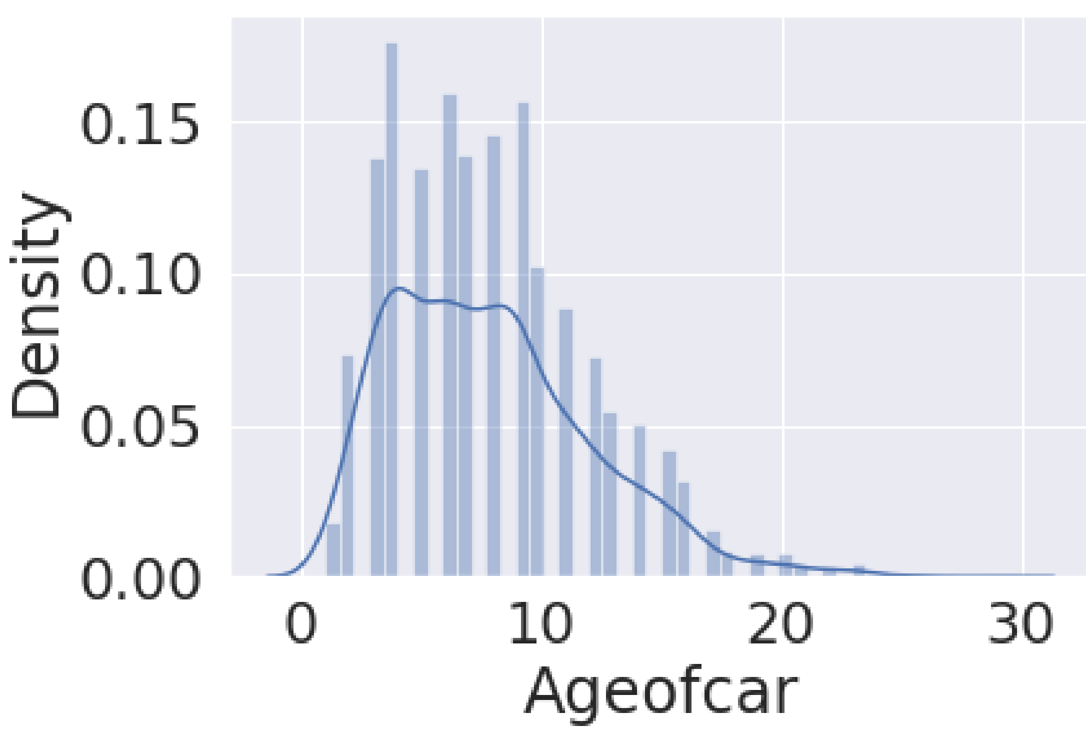
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 4339 entries, 0 to 4339
Data columns (total 10 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   selling_price      4339 non-null   int64
 1   km_driven          4339 non-null   int64
 2   fuel               4339 non-null   category
 3   seller_type        4339 non-null   category
 4   transmission       4339 non-null   category
 5   owner              4339 non-null   category
```

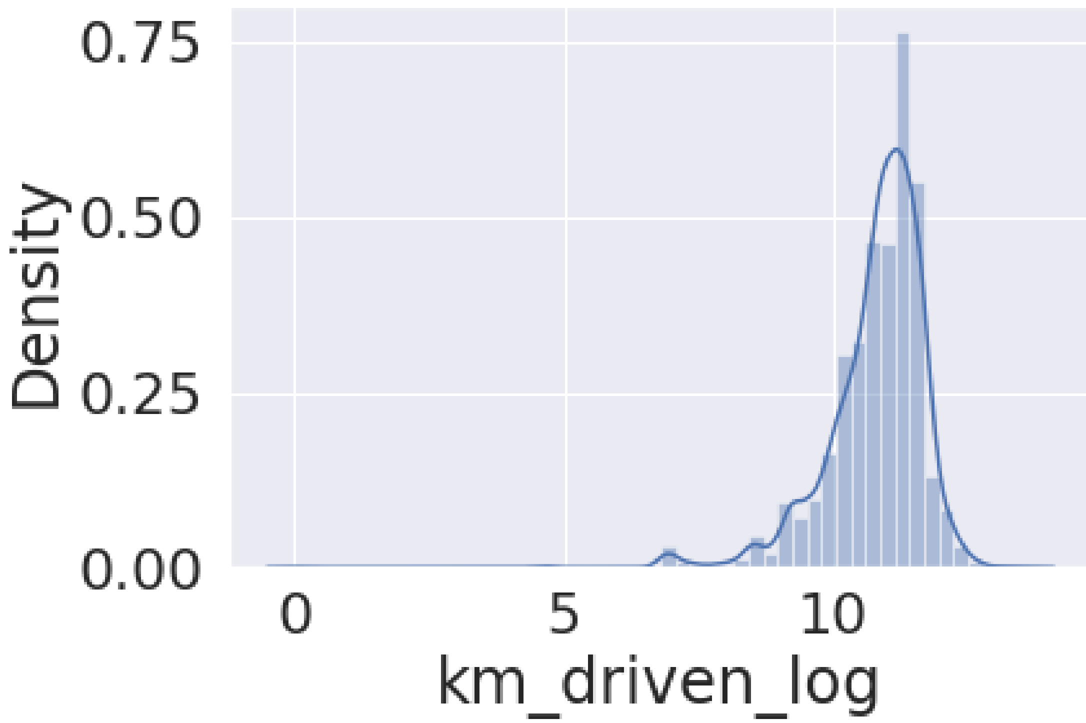
```
 6   Ageofcar           4339 non-null   int64
 7   Brand_Class        4339 non-null   object
 8   km_driven_log      4339 non-null   float64
 9   selling_price_log  4339 non-null   float64
dtypes: category(4), float64(2), int64(3), object(1)
memory usage: 254.9+ KB
```

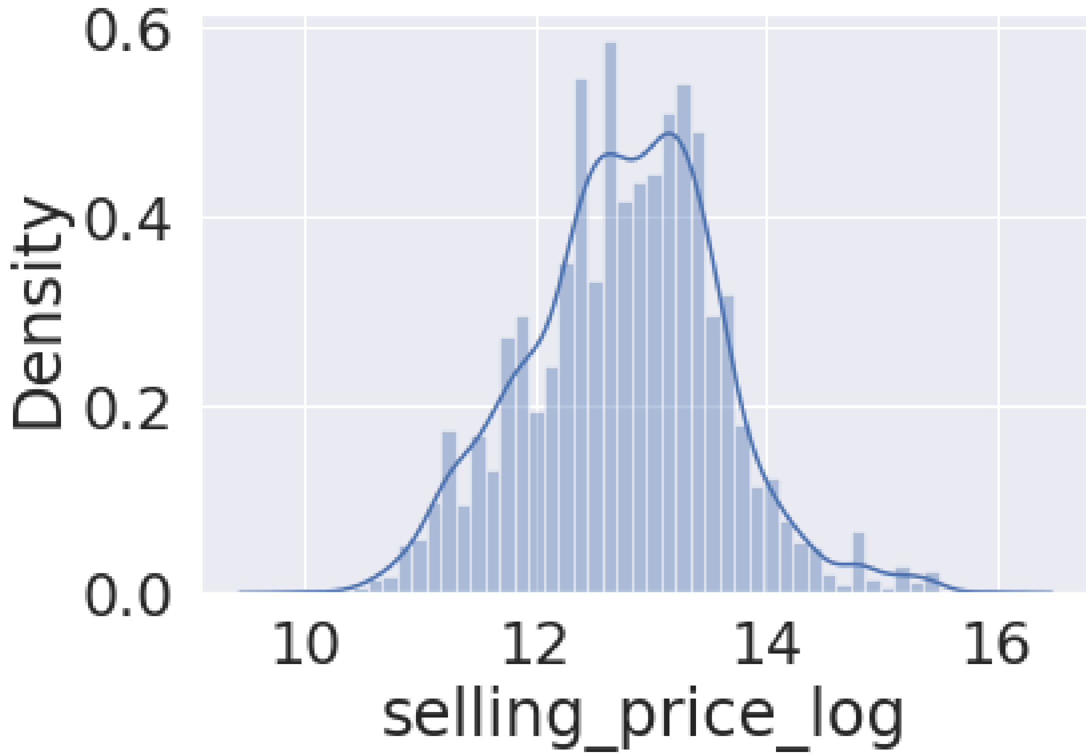
[37]:
```python
cols_to_log = data.select_dtypes(include=np.number).columns.tolist()
for colname in cols_to_log:
    sns.distplot(data[colname], kde=True)
    plt.show()
```




1e−5
1.0
0.5
0.0
Density
0
200000
400000
600000
800000
km_driven


0.15
0.10
0.05
0.00
Density
0
10
20
30
Ageofcar


0.75
0.50
0.25
0.00
Density
0
5
10
km_driven_log


0.6
0.4
0.2
0.0
Density
10
12
14
16
selling_price_log

4 model building

4.0.1 creating first modelwith olny posetive corelated columns

[38]:
```
data.head(2)
```

[38]:
```
   selling_price  km_driven    fuel seller_type transmission        owner  \
0          60000      70000  Petrol  Individual       Manual  First Owner
1         135000      50000  Petrol  Individual       Manual  First Owner

   Ageofcar Brand_Class  km_driven_log  selling_price_log
0        14         Low       11.15625           11.00210
1        14         Low       10.81978           11.81303
```

[39]:
```
data=data.drop(['selling_price'],axis=1)
```

[40]:
```
data.head()
```

[40]:
```
   km_driven    fuel seller_type transmission         owner  Ageofcar  \
0      70000  Petrol  Individual       Manual   First Owner        14
1      50000  Petrol  Individual       Manual   First Owner        14
2     100000  Diesel  Individual       Manual   First Owner         9
3      46000  Petrol  Individual       Manual   First Owner         4
4     141000  Diesel  Individual       Manual  Second Owner         7

  Brand_Class  km_driven_log  selling_price_log
0         Low       11.15625           11.00210
1         Low       10.81978           11.81303
2         Low       11.51293           13.30468
3         Low       10.73640           12.42922
4         Low       11.85652           13.01700
```

[41]:
```
X = data.drop(["selling_price_log",'km_driven','Ageofcar'], axis=1)
y = data[["selling_price_log"]]
```

[42]:
```
def encode_cat_vars(x):
    x = pd.get_dummies(
        x,
        columns=x.select_dtypes(include=["object", "category"]).columns.
 ↪tolist(),
        drop_first=True,
    )
    return x
```

[43]:
```
X = encode_cat_vars(X)
X.head()
```

[43]:
```
   km_driven_log  fuel_Diesel  fuel_Electric  fuel_LPG  fuel_Petrol  \
0       11.15625            0              0         0            1
1       10.81978            0              0         0            1
2       11.51293            1              0         0            0
3       10.73640            0              0         0            1
4       11.85652            1              0         0            0

   seller_type_Individual  seller_type_Trustmark Dealer  transmission_Manual  \
0                       1                             0                    1
1                       1                             0                    1
2                       1                             0                    1
3                       1                             0                    1
4                       1                             0                    1

   owner_Fourth & Above Owner  owner_Second Owner  owner_Test Drive Car  \
0                           0                   0                     0
1                           0                   0                     0
2                           0                   0                     0
3                           0                   0                     0
4                           0                   1                     0

   owner_Third Owner  Brand_Class_Low
0                  0                1
1                  0                1
2                  0                1
3                  0                1
4                  0                1
```

[44]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
↪random_state=42)
X_train.reset_index()
print("X_train:",X_train.shape)
print("X_test:",X_test.shape)
print("y_train:",y_train.shape)
print("y_test:",y_test.shape)
```

```
X_train: (3037, 13)
X_test: (1302, 13)
y_train: (3037, 1)
y_test: (1302, 1)
```

[45]:
```
# Statsmodel api does not add a constant by default. We need to add it␣
↪explicitly.
X_train = sm.add_constant(X_train)
```

```python
# Add constant to test data
X_test = sm.add_constant(X_test)


def build_ols_model(train):
    # Create the model
    olsmodel = sm.OLS(y_train["selling_price_log"], train)
    return olsmodel.fit()
#fit statmodel
olsmodel1 = build_ols_model(X_train)
print(olsmodel1.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:      selling_price_log   R-squared:                       0.513
Model:                            OLS   Adj. R-squared:                  0.511
Method:                 Least Squares   F-statistic:                     245.4
Date:                Mon, 13 Jun 2022   Prob (F-statistic):               0.00
Time:                        04:53:18   Log-Likelihood:                -2693.4
No. Observations:                3037   AIC:                             5415.
Df Residuals:                    3023   BIC:                             5499.
Df Model:                          13
Covariance Type:            nonrobust
================================================================================
================
                                  coef    std err          t      P>|t|
[0.025      0.975]
--------------------------------------------------------------------------------
----------------
const                          16.4916      0.180     91.767      0.000
16.139      16.844
km_driven_log                  -0.2442      0.013    -18.258      0.000
-0.270      -0.218
fuel_Diesel                     0.5360      0.106      5.077      0.000
0.329       0.743
fuel_Electric                  -0.4174      0.600     -0.696      0.486
-1.593       0.758
fuel_LPG                       -0.2872      0.185     -1.557      0.120
-0.649       0.075
fuel_Petrol                    -0.1569      0.105     -1.487      0.137
-0.364       0.050
seller_type_Individual         -0.0816      0.028     -2.959      0.003
-0.136      -0.028
seller_type_Trustmark Dealer    0.5276      0.074      7.140      0.000
0.383       0.672
transmission_Manual            -0.7914      0.039    -20.072      0.000
```

```
-0.869      -0.714
owner_Fourth & Above Owner      -0.7552      0.076     -9.884      0.000
-0.905      -0.605
owner_Second Owner              -0.3278      0.026    -12.530      0.000
-0.379      -0.276
owner_Test Drive Car             0.0824      0.170      0.485      0.628
-0.251       0.415
owner_Third Owner               -0.5366      0.044    -12.275      0.000
-0.622      -0.451
Brand_Class_Low                 -0.4077      0.045     -8.968      0.000
-0.497      -0.319
==============================================================================
Omnibus:                       52.817   Durbin-Watson:                   1.969
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               56.969
Skew:                          -0.297   Prob(JB):                     4.26e-13
Kurtosis:                       3.311   Cond. No.                         620.
==============================================================================

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

5 second model

5.0.1 using all the columns

[46]:
```
data.head()
```

[46]:
```
   km_driven    fuel seller_type transmission         owner  Ageofcar  \
0      70000  Petrol  Individual       Manual   First Owner        14
1      50000  Petrol  Individual       Manual   First Owner        14
2     100000  Diesel  Individual       Manual   First Owner         9
3      46000  Petrol  Individual       Manual   First Owner         4
4     141000  Diesel  Individual       Manual  Second Owner         7

  Brand_Class  km_driven_log  selling_price_log
0         Low       11.15625           11.00210
1         Low       10.81978           11.81303
2         Low       11.51293           13.30468
3         Low       10.73640           12.42922
4         Low       11.85652           13.01700
```

[47]:
```
X=data.drop(['selling_price_log'],axis=1)
y=data['selling_price_log']
```

[48]:
```
def encode_cat_vars(x):
    x = pd.get_dummies(
```

```
        x,
        columns=x.select_dtypes(include=["object", "category"]).columns.
↪tolist(),
        drop_first=True,
    )
    return x
```

[49]:
```
X = encode_cat_vars(X)
X.head()
```

[49]:
```
   km_driven  Ageofcar  km_driven_log  fuel_Diesel  fuel_Electric  fuel_LPG  \
0      70000        14       11.15625            0              0         0
1      50000        14       10.81978            0              0         0
2     100000         9       11.51293            1              0         0
3      46000         4       10.73640            0              0         0
4     141000         7       11.85652            1              0         0

   fuel_Petrol  seller_type_Individual  seller_type_Trustmark Dealer  \
0            1                       1                             0
1            1                       1                             0
2            0                       1                             0
3            1                       1                             0
4            0                       1                             0

   transmission_Manual  owner_Fourth & Above Owner  owner_Second Owner  \
0                    1                           0                   0
1                    1                           0                   0
2                    1                           0                   0
3                    1                           0                   0
4                    1                           0                   1

   owner_Test Drive Car  owner_Third Owner  Brand_Class_Low
0                     0                  0                1
1                     0                  0                1
2                     0                  0                1
3                     0                  0                1
4                     0                  0                1
```

[50]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
↪random_state=42)
X_train.reset_index()
print("X_train:",X_train.shape)
print("X_test:",X_test.shape)
print("y_train:",y_train.shape)
print("y_test:",y_test.shape)
```

```
X_train: (3037, 15)
```

```
X_test: (1302, 15)
y_train: (3037,)
y_test: (1302,)
```

[51]:
```
y_train
```

[51]:
```
929     12.34583
2799    12.97154
2117    12.25486
2880    13.15192
246     13.04979
          …
3445    11.00210
466     11.79810
3093    12.95984
3773    12.12811
860     12.07254
Name: selling_price_log, Length: 3037, dtype: float64
```

[52]:
```
# Statsmodel api does not add a constant by default. We need to add it␣
↪explicitly.
X_train = sm.add_constant(X_train)
# Add constant to test data
X_test = sm.add_constant(X_test)


model2 = sm.OLS(y,X)
model2 = model2.fit()
model2.summary()
```

[52]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                                 OLS Regression Results
================================================================================
=======
Dep. Variable:      selling_price_log   R-squared (uncentered):
0.996
Model:                            OLS   Adj. R-squared (uncentered):
0.996
Method:                 Least Squares   F-statistic:
7.326e+04
Date:                Mon, 13 Jun 2022   Prob (F-statistic):
0.00
Time:                        04:53:19   Log-Likelihood:
-5193.2
No. Observations:                4339   AIC:
1.042e+04
```

```
Df Residuals:                    4324   BIC:
1.051e+04
Df Model:                          15
Covariance Type:            nonrobust
================================================================================
================
                                   coef    std err          t      P>|t|
[0.025      0.975]
--------------------------------------------------------------------------------
----------------
km_driven                    -1.46e-05    3.6e-07    -40.492      0.000
-1.53e-05   -1.39e-05
Ageofcar                       -0.1467      0.004    -39.889      0.000
-0.154      -0.139
km_driven_log                   1.1018      0.012     89.104      0.000
1.078       1.126
fuel_Diesel                     3.7638      0.114     33.092      0.000
3.541       3.987
fuel_Electric                   3.6455      0.812      4.489      0.000
2.053       5.238
fuel_LPG                        3.0706      0.202     15.189      0.000
2.674       3.467
fuel_Petrol                     3.4075      0.113     30.274      0.000
3.187       3.628
seller_type_Individual         -0.1007      0.031     -3.234      0.001
-0.162      -0.040
seller_type_Trustmark Dealer    0.3937      0.084      4.687      0.000
0.229       0.558
transmission_Manual            -0.5973      0.044    -13.431      0.000
-0.685      -0.510
owner_Fourth & Above Owner     -0.0226      0.094     -0.240      0.811
-0.208       0.162
owner_Second Owner             -0.0682      0.031     -2.172      0.030
-0.130      -0.007
owner_Test Drive Car            3.0741      0.199     15.424      0.000
2.683       3.465
owner_Third Owner              -0.1490      0.052     -2.849      0.004
-0.252      -0.046
Brand_Class_Low                 0.0142      0.051      0.279      0.780
-0.086       0.114
==============================================================================
Omnibus:                     2408.720   Durbin-Watson:                   1.860
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            37072.363
Skew:                           2.314   Prob(JB):                         0.00
Kurtosis:                      16.551   Cond. No.                     5.40e+06
==============================================================================
```

```
Notes:
[1] R² is computed without centering (uncentered) since the model does not
contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[3] The condition number is large, 5.4e+06. This might indicate that there are
strong multicollinearity or other numerical problems.
"""
```

[54]:

```
y_pred=model2.predict(X)
y_pred
```

[54]:

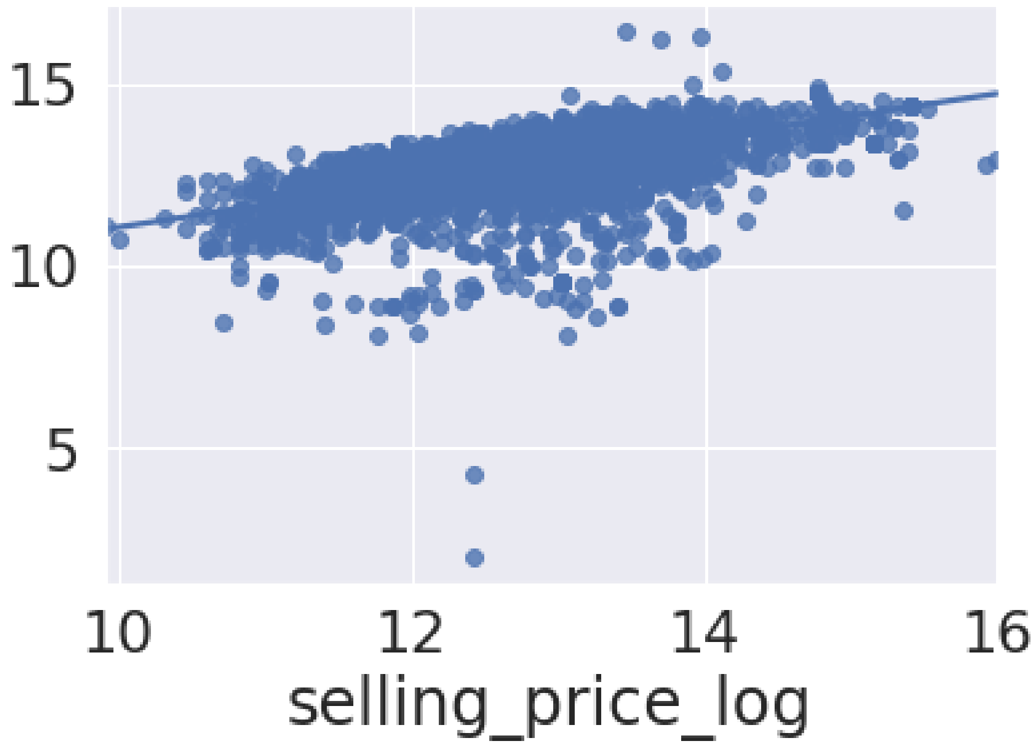
```
0       11.94016
1       11.86135
2       12.98502
3       13.29476
4       12.99034
          …
4335    13.25623
4336    13.25623
4337    12.16327
4338    13.60165
4339    13.08165
Length: 4339, dtype: float64
```

[55]:

```
sns.regplot(x=y, y=y_pred, ci=None, color="b")
```

[55]:

```
<AxesSubplot:xlabel='selling_price_log'>
```



6 Recommendations and insights

- need to aquire more automatic cars
- should focus on new cars to gain more profit
- diseal cars are more popular
- first degree of owner has more value as price
- budget cars are popular
- high profile cars have low significant values

[ ]: