Multiple_linear_Regression

February 2, 2022

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

[2]:
```
df = pd.read_csv("cars24-car-price-clean.csv")
```

[3]:
```
df.head()
```

[3]:

	selling_price	year	km_driven	mileage	engine	max_power	make	model	transmission_type	seats_coupe	seats_family	seats_large	fuel_cng	fuel_diesel	fuel_electric	fuel_lpg	fuel_petrol	seller_dealer	seller_individual	seller_trustmark dealer
0	1.20	2012.0	120000	19.70	796.0	46.30	Maruti	Alto Std	1	0	1	0	0	0	0	0	1	0	1	0
1	5.50	2016.0	20000	18.90	1197.0	82.00	Hyundai	Grand i10 Asta	1	0	1	0	0	0	0	0	1	0	1	0
2	2.15	2010.0	60000	17.00	1197.0	80.00	Hyundai	i20 Asta	1	0	1	0	0	0	0	0	1	0	1	0
3	2.26	2012.0	37000	20.92	998.0	67.10	Maruti	Alto K10 2010-2014 VXI	1	0	1	0	0	0	0	0	1	0	1	0
4	5.70	2015.0	30000	22.77	1498.0	98.59	Ford	Ecosport 2015-2021 1.5 TDCi Titanium BSIV	1	0	1	0	0	1	0	0	0	1	0	0

[4]:
```
df.make.value_counts()
```

[4]:
```
Maruti           5650
Hyundai          3562
Honda            1779
Mahindra         1276
Toyota           1189
Tata              971
Ford              900
Volkswagen        761
Renault           636
Mercedes-Benz     485
BMW               483
Skoda             422
Chevrolet         406
Audi              324
Nissan            289
Datsun            170
Fiat              113
Jaguar             80
Land               51
Volvo              42
Jeep               41
Mitsubishi         39
Kia                33
Porsche            25
Mini               23
MG                 19
Isuzu              10
Lexus              10
Force               5
Ambassador          4
Bentley             4
OpelCorsa           3
ISUZU               2
DC                  2
Premier             2
Maserati            2
Daewoo              2
Lamborghini         1
Opel                1
Rolls-Royce         1
Mercedes-AMG        1
Ferrari             1
Name: make, dtype: int64
```

[5]:
```
df.drop(["make","model"], axis = 1, inplace=True)
```

[6]:
```
X = df[df.columns.drop("selling_price")]
Y = df["selling_price"]
```

[7]:
```
X = X.to_numpy()
Y = Y.to_numpy()
```

[8]:
```
import numpy as np
u = np.mean(X,axis=0)
std = np.std(X,axis=0)
```

[9]:
```
X = (X-u)/std
```

[10]:
```
ones = np.ones((X.shape[0],1))
X = np.hstack((ones,X))
```

[11]:
```
print(X.shape, Y.shape)
```

```
(19820, 18) (19820,)
```

[ ]:
```
def hypothesis(x,theta):
    y_ = 0.0
    n = x.shape[0]
    for i in range(n):
        y_  += (theta[i]*x[i])
    return y_

def error(X,y,theta):
    e = 0.0
    m = X.shape[0]

    for i in range(m):
        y_ = hypothesis(X[i],theta)
        e += (y[i] - y_)**2

    return e/m

def gradient(X,y,theta):
    m,n = X.shape

    grad = np.zeros((n,))

    # for all values of j
    for j in range(n):
        #sum over all examples
        for i in range(m):
            y_ = hypothesis(X[i],theta)
            grad[j] += (y_ - y[i])*X[i][j]
    # Out of the loops
```

```
    return grad/m

def gradient_descent(X,y,learning_rate=0.1,max_epochs=100):
    m,n = X.shape
    theta = np.zeros((n,))
    error_list = []

    for i in range(max_epochs):
        e = error(X,y,theta)
        error_list.append(e)

        # Gradient Descent
        grad = gradient(X,y,theta)
        for j in range(n):
            theta[j] = theta[j] - learning_rate*grad[j]

    return theta,error_list
```

[ ]:
```
import time
start = time.time()
theta, error_list = gradient_descent(X,Y)
end = time.time()
print("Time taken is ", end-start)
```

[14]:
```
import time
def hypothesis(X,theta):
    return np.dot(X,theta)

def error(X,y,theta):
    e = 0.0
    m = X.shape[0]
    y_ = hypothesis(X,theta)
    e = np.sum((y-y_)**2)
    return e/m

def gradient(X,y,theta):

    y_ = hypothesis(X,theta)
    grad = np.dot(X.T,(y_ - y))
    m = X.shape[0]
    return grad/m

def gradient_descent(X,y,learning_rate = 0.1,max_iters=500):

    n = X.shape[1]
    theta = np.zeros((n,))
    error_list = []
```

```
    for i in range(max_iters):
        e = error(X,y,theta)
        error_list.append(e)

        #Gradient descent
        grad = gradient(X,y,theta)
        theta = theta - learning_rate*grad

    return theta, error_list
```

[15]:
```
start = time.time()
theta,error_list = gradient_descent(X,Y)
end = time.time()
print("Time taken by Vectorized Code",end-start)
```

```
Time taken by Vectorized Code 0.19299769401550293
```

[16]:
```
print(theta)
```

```
[ 7.38842289  1.88476748 -0.47374531  0.32749344  1.15203852  5.07149364
 -0.96643086  0.82449862 -0.28208162 -0.22080249  0.06916548  0.07993341
 -0.05119083  0.20479193 -0.11846699  0.03336637 -0.01609964 -0.08716032]
```

[17]:
```
from sklearn.linear_model import LinearRegression
model = LinearRegression()

X = df[df.columns.drop('selling_price')]
Y = df["selling_price"]
```

[18]:
```
from sklearn.preprocessing import StandardScaler, MinMaxScaler
sc = StandardScaler()
cols = X.columns
X[cols] = sc.fit_transform(X[cols])
#x = sc.fit_transform(x) this will result into a numpy array
```

```
<ipython-input-18-96acca03f063>:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  X[cols] = sc.fit_transform(X[cols])
C:\Users\sci\anaconda3\lib\site-packages\pandas\core\indexing.py:1738:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
```

```
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  self._setitem_single_column(loc, value[:, i].tolist(), pi)
```

[19]:
```
X.head()
```

[19]:
```
       year  km_driven   mileage    engine  max_power  transmission_type  \
0 -0.801317   1.195828  0.045745 -1.310754  -1.157780           0.495818
1  0.450030  -0.737872 -0.140402 -0.537456  -0.360203           0.495818
2 -1.426990   0.035608 -0.582501 -0.537456  -0.404885           0.495818
3 -0.801317  -0.409143  0.329620 -0.921213  -0.693085           0.495818
4  0.137194  -0.544502  0.760085  0.042999   0.010435           0.495818

   seats_coupe  seats_family  seats_large  fuel_cng  fuel_diesel  \
0    -0.110946       0.20159    -0.166238 -0.127286    -0.985275
1    -0.110946       0.20159    -0.166238 -0.127286    -0.985275
2    -0.110946       0.20159    -0.166238 -0.127286    -0.985275
3    -0.110946       0.20159    -0.166238 -0.127286    -0.985275
4    -0.110946       0.20159    -0.166238 -0.127286     1.014945

   fuel_electric  fuel_lpg  fuel_petrol  seller_dealer  seller_individual  \
0      -0.020095 -0.056917     1.024622      -1.224101           1.248892
1      -0.020095 -0.056917     1.024622      -1.224101           1.248892
2      -0.020095 -0.056917     1.024622      -1.224101           1.248892
3      -0.020095 -0.056917     1.024622      -1.224101           1.248892
4      -0.020095 -0.056917    -0.975970       0.816926          -0.800710

   seller_trustmark dealer
0                -0.098382
1                -0.098382
2                -0.098382
3                -0.098382
4                -0.098382
```

[20]:
```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
```

[21]:
```
model.fit(X,Y)
```

[21]:
```
LinearRegression()
```

[22]:
```
model.intercept_
```

[22]:
```
7.38842288799193
```

[23]:
```
model.coef_
```

[23]:
```
array([ 1.88489776, -0.47362113,  0.32576595,  1.14813293,  5.0739298 ,
       -0.96600933,  0.82443059, -0.28221431, -0.22059749,  0.06933032,
```

```
 0.08060625, -0.05072673,  0.20467277, -0.11918639,  0.03334649,
-0.01608393, -0.08713903])
```

[24]:
```
model.score(X,Y)
```

[24]:
```
0.6138259085441247
```

[25]:
```
print("Adjusted R-squared:", 1 - (1-model.score(X, Y))*(len(Y)-1)/(len(Y)-X.
↪shape[1]-1))
```

```
Adjusted R-squared: 0.6134943784181399
```

[27]:
```
import statsmodels.api as sm
```

[28]:
```
X_sm = sm.add_constant(X)  #Statmodels default is without intercept, to add␣
↪intercept we need to add constant

sm_model = sm.OLS(Y, X_sm).fit()
```

[29]:
```
print(sm_model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:          selling_price   R-squared:                       0.614
Model:                            OLS   Adj. R-squared:                  0.614
Method:                 Least Squares   F-statistic:                     2249.
Date:                Wed, 02 Feb 2022   Prob (F-statistic):               0.00
Time:                        21:51:33   Log-Likelihood:                -62371.
No. Observations:               19820   AIC:                         1.248e+05
Df Residuals:                   19805   BIC:                         1.249e+05
Df Model:                          14
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025
0.975]
------------------------------------------------------------------------------
const          7.3884      0.040    184.720      0.000       7.310
7.467
year           1.8849      0.046     41.373      0.000       1.796
1.974
km_driven     -0.4736      0.044    -10.682      0.000      -0.561
-0.387
mileage        0.3258      0.068      4.772      0.000       0.192
0.460
engine         1.1481      0.095     12.125      0.000       0.963
1.334
```

```
max_power                 5.0739      0.078     65.084      0.000       4.921
5.227
transmission_type        -0.9660      0.049    -19.736      0.000      -1.062
-0.870
seats_coupe               0.8244      0.036     22.826      0.000       0.754
0.895
seats_family             -0.2822      0.021    -13.349      0.000      -0.324
-0.241
seats_large              -0.2206      0.030     -7.341      0.000      -0.280
-0.162
fuel_cng                  0.0693      0.040      1.745      0.081      -0.009
0.147
fuel_diesel               0.0806      0.029      2.757      0.006       0.023
0.138
fuel_electric            -0.0507      0.043     -1.182      0.237      -0.135
0.033
fuel_lpg                  0.2047      0.040      5.108      0.000       0.126
0.283
fuel_petrol              -0.1192      0.030     -3.996      0.000      -0.178
-0.061
seller_dealer             0.0333      0.021      1.580      0.114      -0.008
0.075
seller_individual        -0.0161      0.021     -0.753      0.451      -0.058
0.026
seller_trustmark dealer  -0.0871      0.039     -2.206      0.027      -0.165
-0.010
==============================================================================
Omnibus:                    36798.081   Durbin-Watson:                   2.015
Prob(Omnibus):                  0.000   Jarque-Bera (JB):        271533084.294
Skew:                          13.327   Prob(JB):                         0.00
Kurtosis:                     575.790   Cond. No.                     3.82e+15
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The smallest eigenvalue is 4.27e-27. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

[ ]:
```
# VIF - Variance inflation factor
```

[30]:
```
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

[31]:
```
vif = pd.DataFrame()
X_t = X
vif['Features'] = X_t.columns
```

```
vif['VIF'] = [variance_inflation_factor(X_t.values, i) for i in range(X_t.
 ↪shape[1])]
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

```
C:\Users\sci\anaconda3\lib\site-
packages\statsmodels\stats\outliers_influence.py:193: RuntimeWarning: divide by
zero encountered in double_scalars
  vif = 1. / (1. - r_squared_i)
```

[31]:

	Features	VIF
8	seats_large	inf
9	fuel_cng	inf
15	seller_individual	inf
14	seller_dealer	inf
13	fuel_petrol	inf
12	fuel_lpg	inf
11	fuel_electric	inf
10	fuel_diesel	inf
16	seller_trustmark dealer	inf
6	seats_coupe	inf
7	seats_family	19806433.12
3	engine	5.60
4	max_power	3.80
2	mileage	2.91
5	transmission_type	1.50
0	year	1.30
1	km_driven	1.23

[32]:
```
cols2 =␣
 ↪["max_power","transmission_type","year","km_driven","fuel_electric","seats_coupe"␣
 ↪]
```

[33]:
```
X2 = X[cols2]

X2_sm = sm.add_constant(X2)  #Statmodels default is without intercept, to add␣
 ↪intercept we need to add constant

sm_model = sm.OLS(Y, X2_sm).fit()
```

[34]:
```
print(sm_model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:          selling_price   R-squared:                       0.607
Model:                            OLS   Adj. R-squared:                  0.606
Method:                 Least Squares   F-statistic:                     5090.
```

```
Date:                Wed, 02 Feb 2022   Prob (F-statistic):               0.00
Time:                        22:15:11   Log-Likelihood:                -62556.
No. Observations:               19820   AIC:                         1.251e+05
Df Residuals:                   19813   BIC:                         1.252e+05
Df Model:                           6
Covariance Type:            nonrobust
================================================================================
=====
                     coef    std err          t      P>|t|      [0.025
0.975]
--------------------------------------------------------------------------------
-----
const                7.3884      0.040    183.034      0.000       7.309
7.468
max_power            5.9311      0.048    122.301      0.000       5.836
6.026
transmission_type   -0.8600      0.049    -17.729      0.000      -0.955
-0.765
year                 1.9042      0.043     44.432      0.000       1.820
1.988
km_driven           -0.2646      0.043     -6.194      0.000      -0.348
-0.181
fuel_electric        0.0303      0.040      0.751      0.453      -0.049
0.110
seats_coupe          0.9502      0.041     23.216      0.000       0.870
1.030
==============================================================================
Omnibus:                    35995.866   Durbin-Watson:                   2.014
Prob(Omnibus):                  0.000   Jarque-Bera (JB):        230783059.888
Skew:                          12.713   Prob(JB):                         0.00
Kurtosis:                     531.023   Cond. No.                         1.92
==============================================================================

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

[35]:
```
# Lets drop one more variable -- fuel_electric
cols2 =␣
 ↪["engine","max_power","transmission_type","year","km_driven","seats_coupe" ]
```

[36]:
```
# build model without all those correlated columns
X2 = X[cols2]

X2_sm = sm.add_constant(X2)  #Statmodels default is without intercept, to add␣
 ↪intercept we need to add constant
```

```
sm_model = sm.OLS(Y, X2_sm).fit()
```

[37]:
```
print(sm_model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:          selling_price   R-squared:                       0.611
Model:                            OLS   Adj. R-squared:                  0.611
Method:                 Least Squares   F-statistic:                     5196.
Date:                Wed, 02 Feb 2022   Prob (F-statistic):               0.00
Time:                        22:18:27   Log-Likelihood:                -62432.
No. Observations:               19820   AIC:                         1.249e+05
Df Residuals:                   19813   BIC:                         1.249e+05
Df Model:                           6
Covariance Type:            nonrobust
==============================================================================
=====
                     coef    std err          t      P>|t|      [0.025
0.975]
------------------------------------------------------------------------------
-----
const                7.3884      0.040    184.191      0.000       7.310
7.467
engine               1.0943      0.069     15.868      0.000       0.959
1.229
max_power            5.0168      0.075     66.851      0.000       4.870
5.164
transmission_type   -0.9303      0.048    -19.246      0.000      -1.025
-0.836
year                 1.9815      0.043     46.227      0.000       1.897
2.066
km_driven           -0.4119      0.043     -9.477      0.000      -0.497
-0.327
seats_coupe          0.9936      0.041     24.382      0.000       0.914
1.074
==============================================================================
Omnibus:                    36607.794   Durbin-Watson:                   2.012
Prob(Omnibus):                  0.000   Jarque-Bera (JB):        262332940.900
Skew:                          13.178   Prob(JB):                         0.00
Kurtosis:                     565.995   Cond. No.                         3.59
==============================================================================

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

[38]:
```
vif = pd.DataFrame()
X_t = X2
vif['Features'] = X_t.columns
vif['VIF'] = [variance_inflation_factor(X_t.values, i) for i in range(X_t.
 ↪shape[1])]
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

[38]:

	Features	VIF
1	max_power	3.50
0	engine	2.96
2	transmission_type	1.45
4	km_driven	1.17
3	year	1.14
5	seats_coupe	1.03

[39]:
```
from sklearn.model_selection import train_test_split
```

[40]:
```
x_train,x_test,y_train,y_test = train_test_split(X2,Y,test_size = 0.1,␣
 ↪random_state=1 )
```

[41]:
```
print("Traning set shape X:", x_train.shape)
print("Traning set shape Y:", y_train.shape)
print("Test set shape X:", x_test.shape)
print("Test set shape Y:", y_test.shape)
```

```
Traning set shape X: (17838, 6)
Traning set shape Y: (17838,)
Test set shape X: (1982, 6)
Test set shape Y: (1982,)
```

[42]:
```
final_model = LinearRegression()
final_model.fit(x_train,y_train)
```

[42]:
```
LinearRegression()
```

[43]:
```
final_model.score(x_train,y_train)
```

[43]:
```
0.6100390860403653
```

[44]:
```
final_model.intercept_
```

[44]:
```
7.395513670805413
```

[45]:
```
final_model.coef_
```

[45]:
```
array([ 1.10446698,  5.05935657, -0.89690768,  1.98218355, -0.39869078,
        1.01856153])
```

[47]:
```
y_pred = final_model.predict(x_test)# this will give preditions on test dataset
```

[48]:
```
# model performance
from sklearn.metrics import mean_absolute_error, mean_squared_error,␣
 ↪mean_absolute_percentage_error
```

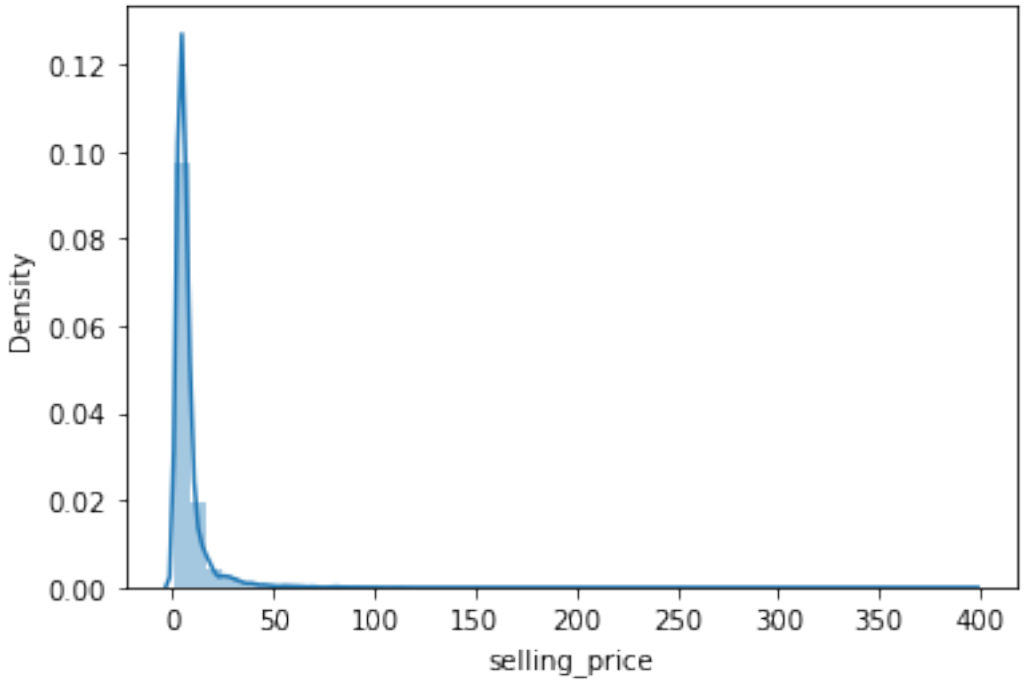
[55]:
```
print("Mean absolute error:", mean_absolute_error(y_pred,y_test))
print("Mean Squared error:", mean_squared_error(y_pred,y_test))
print("Root Mean squared error:", np.sqrt(mean_squared_error(y_pred, y_test)))
print("Mean absolute Percentage error:",␣
 ↪mean_absolute_percentage_error(y_pred,y_test))
```

```
Mean absolute error: 2.8157245895216767
Mean Squared error: 27.036450750588255
Root Mean squared error: 5.1996587148185265
Mean absolute Percentage error: 1.6471312291786118
```

[56]:
```
sns.distplot(y_train)
```

```
C:\Users\sci\anaconda3\lib\site-packages\seaborn\distributions.py:2557:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
```

[56]:
```
<AxesSubplot:xlabel='selling_price', ylabel='Density'>
```



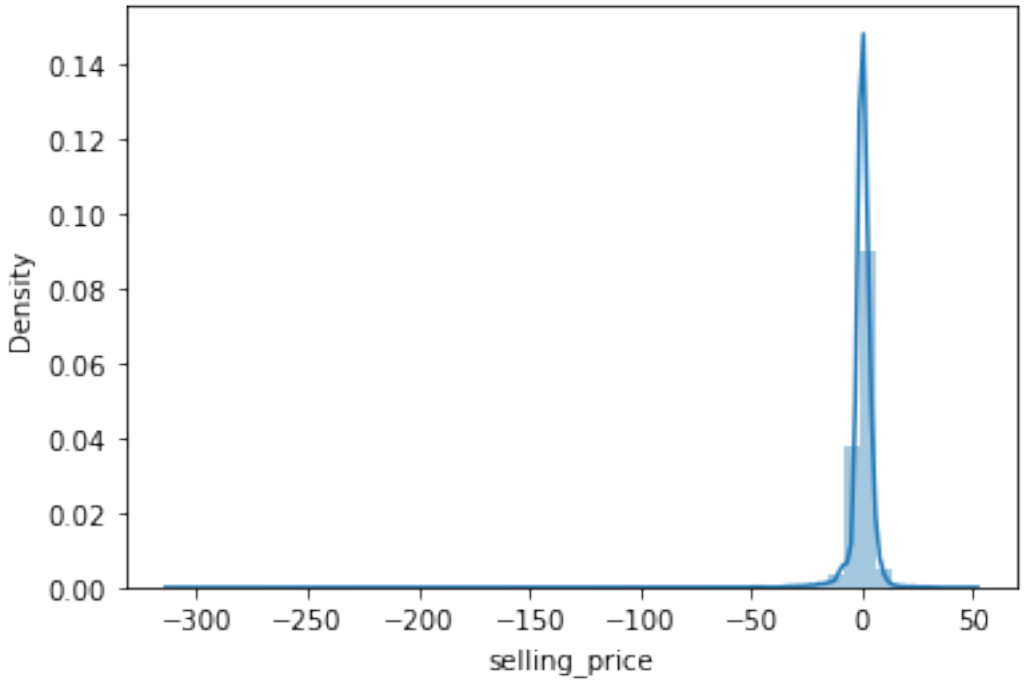
[57]:
```
# Residual Analysis
# errors normally distributed?
pred = final_model.predict(x_train)
```

[58]:
```
errors = pred - y_train # residuals
```

[59]:
```
sns.distplot(errors)
```

```
C:\Users\sci\anaconda3\lib\site-packages\seaborn\distributions.py:2557:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
```

[59]:
```
<AxesSubplot:xlabel='selling_price', ylabel='Density'>
```

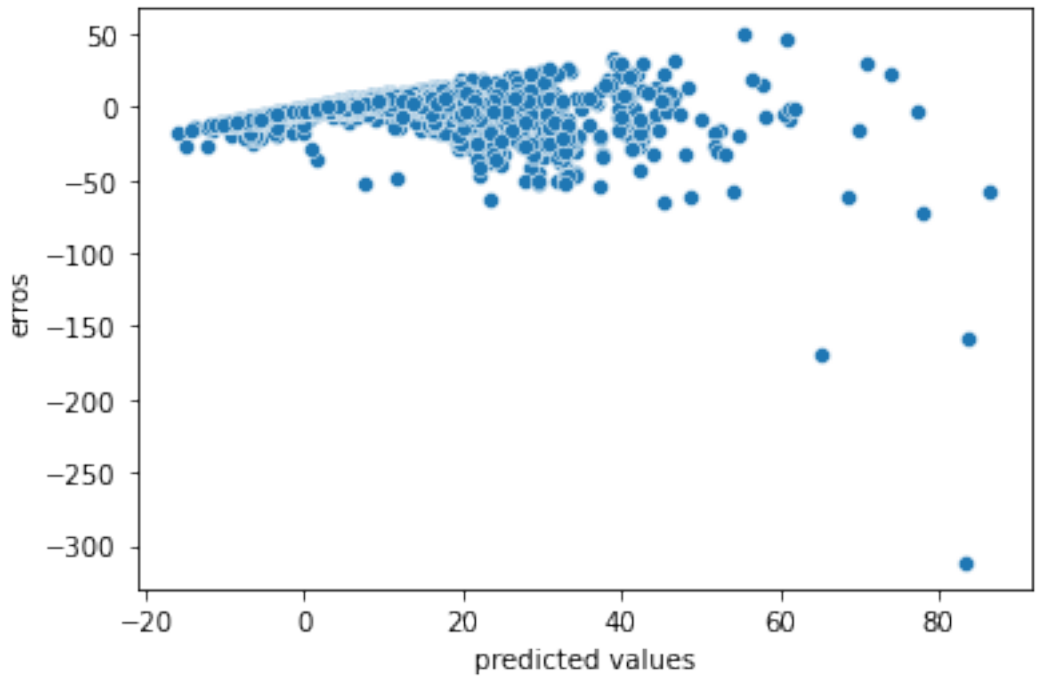


[60]:
```
## Autocorrelation and Heteroscedasticity
## Plot ( predicted values Vs Residuals)

sns.scatterplot(pred,errors )
plt.xlabel('predicted values')
plt.ylabel('erros')
```

```
C:\Users\sci\anaconda3\lib\site-packages\seaborn\_decorators.py:36:
FutureWarning: Pass the following variables as keyword args: x, y. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
```

[60]:
```
Text(0, 0.5, 'erros')
```



[61]:
```
df.columns
```

[61]:
```
Index(['selling_price', 'year', 'km_driven', 'mileage', 'engine', 'max_power',
       'transmission_type', 'seats_coupe', 'seats_family', 'seats_large',
       'fuel_cng', 'fuel_diesel', 'fuel_electric', 'fuel_lpg', 'fuel_petrol',
       'seller_dealer', 'seller_individual', 'seller_trustmark dealer'],
      dtype='object')
```

[62]:
```
df3 = df[df["selling_price"] < 25 ]
```

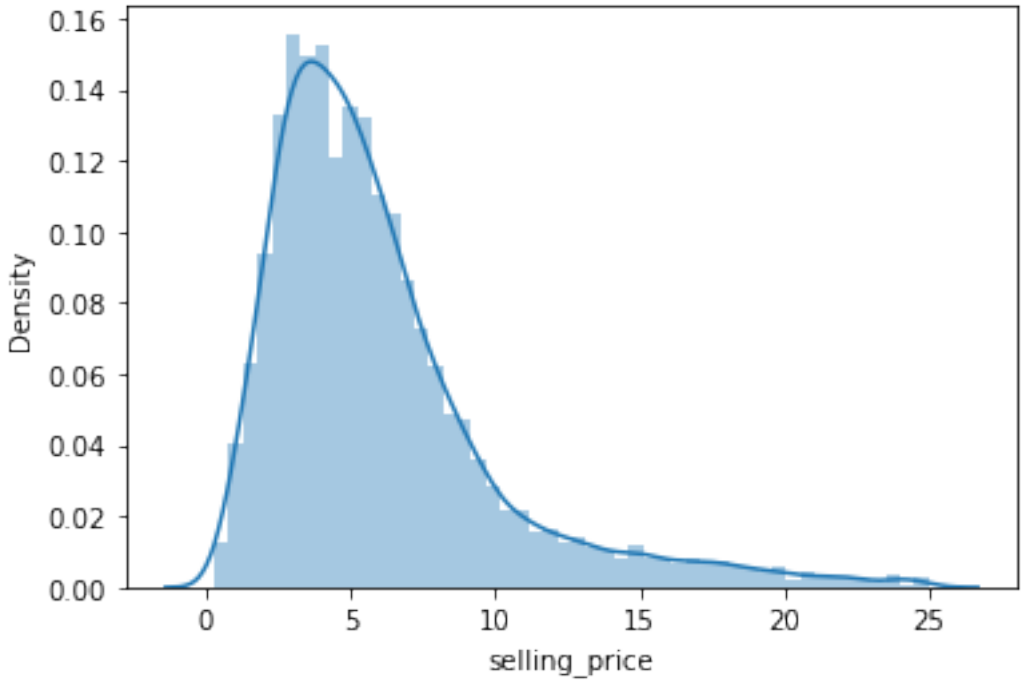
[63]:
```
df3.shape
```

[63]:
```
(19034, 18)
```

[64]:
```
sns.distplot(df3["selling_price"])
```

```
C:\Users\sci\anaconda3\lib\site-packages\seaborn\distributions.py:2557:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
```

[64]:
```
<AxesSubplot:xlabel='selling_price', ylabel='Density'>
```



[66]:
```
X = df3[cols2]
Y = df3["selling_price"]
```

[67]:
```
X.shape
```

[67]: (19034, 6)

[68]:
```
sc = StandardScaler()
cols = X.columns
X[cols] = sc.fit_transform(X[cols])
```

```
<ipython-input-68-53e462b909d4>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  X[cols] = sc.fit_transform(X[cols])
C:\Users\sci\anaconda3\lib\site-packages\pandas\core\indexing.py:1738:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

```
  self._setitem_single_column(loc, value[:, i].tolist(), pi)
```

[70]:
```
X.head()
```

[70]:

	engine	max_power	transmission_type	year	km_driven	seats_coupe
0	-1.371369	-1.295474	0.445493	-0.772055	1.169999	-0.092651
1	-0.508655	-0.311199	0.445493	0.474022	-0.747100	-0.092651
2	-0.508655	-0.366341	0.445493	-1.395093	0.019739	-0.092651
3	-0.936785	-0.722003	0.445493	-0.772055	-0.421194	-0.092651
4	0.138918	0.146199	0.445493	0.162503	-0.555391	-0.092651

[71]:
```
x_train, x_test, y_train, y_test = train_test_split(X,Y, test_size=0.1,␣
↪random_state=1)
```

[72]:
```
final = LinearRegression()
```

[73]:
```
final.fit(x_train,y_train)
```

[73]:
```
LinearRegression()
```

[74]:
```
final.score(x_train,y_train)
```

[74]:
```
0.6963396465522724
```

[77]:
```
print("Adjusted R-squared:", 1 - (1-final.score(x_train,␣
↪y_train))*(len(y_train)-1)/(len(y_train)-x_train.shape[1]-1))
```

```
Adjusted R-squared: 0.6962332421768309
```

[78]:
```
y_pred = final.predict(x_test)
```

[79]:
```
print("Mean absolute error:", mean_absolute_error(y_pred,y_test))
print("Mean Squared error:", mean_squared_error(y_pred,y_test))
print("Root Mean squared error:", np.sqrt(mean_squared_error(y_pred, y_test)))
print("Mean absolute Percentage error:",␣
↪mean_absolute_percentage_error(y_pred,y_test))
```

```
Mean absolute error: 1.5223075684470253
Mean Squared error: 5.18613444393155
Root Mean squared error: 2.2773085965524196
Mean absolute Percentage error: 0.433396790458142
```

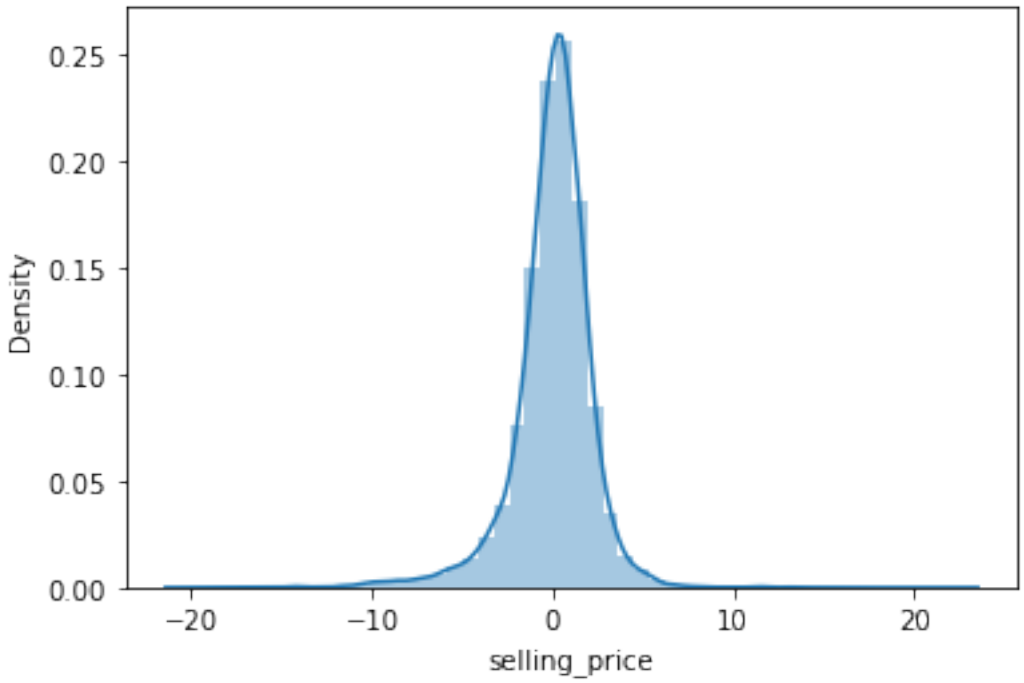
[80]:
```
preds = final.predict(x_train)
```

[81]:
```
errors = preds - y_train
```

[82]:
```
## Residual analysis
sns.distplot(errors)
```

```
C:\Users\sci\anaconda3\lib\site-packages\seaborn\distributions.py:2557:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
```

[82]: `<AxesSubplot:xlabel='selling_price', ylabel='Density'>`

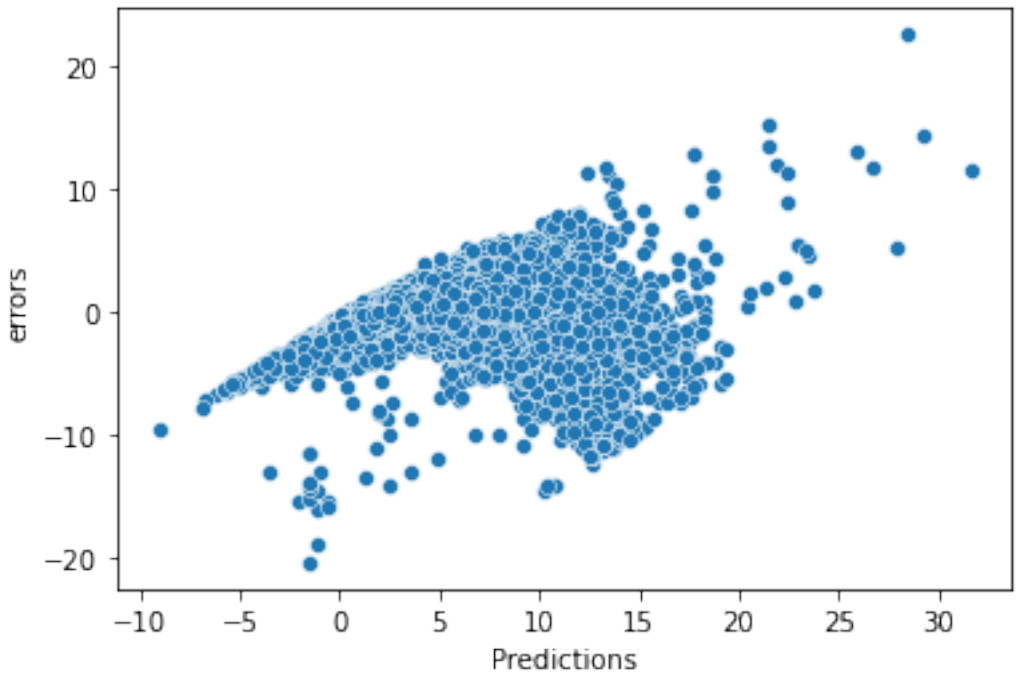


[83]:
```
## Predictions Vs. Residuals plot

sns.scatterplot(preds, errors)
plt.xlabel("Predictions")
plt.ylabel("errors")
```

```
C:\Users\sci\anaconda3\lib\site-packages\seaborn\_decorators.py:36:
FutureWarning: Pass the following variables as keyword args: x, y. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
```

[83]: `Text(0, 0.5, 'errors')`



[85]:
```
Y_train = np.log(y_train)
Y_test = np.log(y_test)
```

[86]:
```
sns.distplot(Y_train)
```

```
C:\Users\sci\anaconda3\lib\site-packages\seaborn\distributions.py:2557:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
```

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

[87]:
```
final2 = LinearRegression()
```

[88]:
```
final2.fit(x_train,Y_train )
```

[88]:
```
LinearRegression()
```

[89]:
```
final2.score(x_train,Y_train )
```

[89]:
```
0.7835758251222755
```

[90]:
```
Y_pred = final2.predict(x_test)
```

[92]:
```
## Performance metrics on log transformed model
print("Mean absolute error:", mean_absolute_error(Y_pred,Y_test))
print("Mean Squared error:", mean_squared_error(Y_pred,Y_test))
print("Root Mean squared error:", np.sqrt(mean_squared_error(Y_pred, Y_test)))
print("Mean absolute Percentage error:",␣
↪mean_absolute_percentage_error(Y_pred,Y_test))
```

```
Mean absolute error: 0.2190299188461953
Mean Squared error: 0.08412007674007366
Root Mean squared error: 0.29003461300347183
Mean absolute Percentage error: 0.21650325035688517
```

[93]:
```
preds = final2.predict(x_train)
```

[94]:
```
errors = preds - Y_train
```

[95]:
```
sns.distplot(errors)
```

```
C:\Users\sci\anaconda3\lib\site-packages\seaborn\distributions.py:2557:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
```

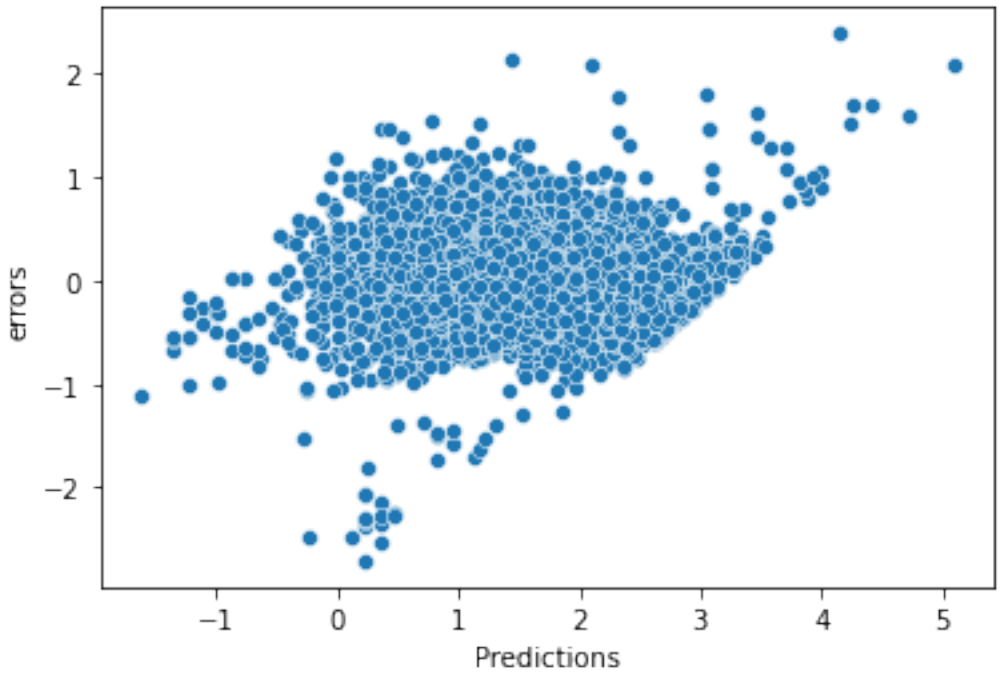
[95]: <AxesSubplot:xlabel='selling_price', ylabel='Density'>

[96]:
```
## Predictions Vs. Residuals plot

sns.scatterplot(preds, errors)
plt.xlabel("Predictions")
plt.ylabel("errors")
```

```
C:\Users\sci\anaconda3\lib\site-packages\seaborn\_decorators.py:36:
FutureWarning: Pass the following variables as keyword args: x, y. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
```

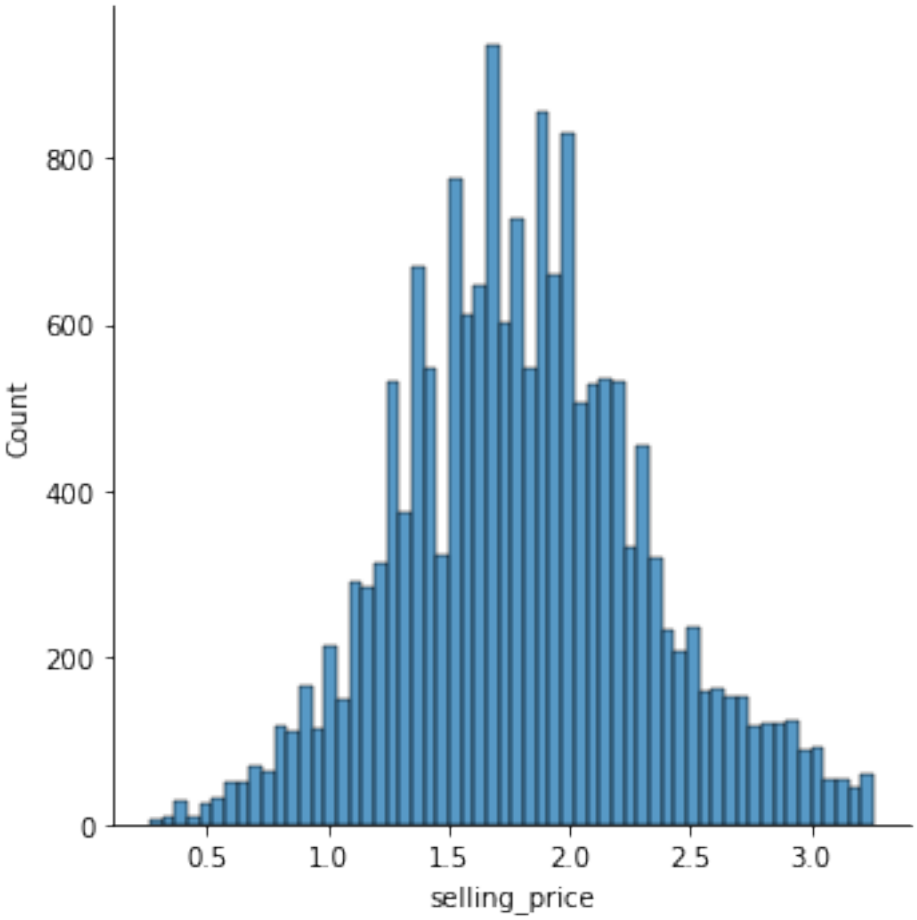
[96]: Text(0, 0.5, 'errors')



[97]:
```
Y_train = np.log(1+y_train)
```

[98]:
```
sns.displot(Y_train)
```

[98]: <seaborn.axisgrid.FacetGrid at 0x282112f33a0>


800
600
400
200
0
Count
0.5
1.0
1.5
2.0
2.5
3.0
selling_price

[ ]: