In [13]:
```
import pandas as pd

data = pd.read_csv('car data.csv')
data.head()
```

Out[13]:

	Car_Name	Year	Selling_Price	Present_Price	Kms_Driven	Fuel_Type	Seller_Type	Transmission	O
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 [14]:
```
data.shape #will tell the number of rows and coloumn in the dataset
```

Out[14]:
```
(301, 9)
```

In [15]:
```
#will see the unique value in categorical variable

print(data['Fuel_Type'].unique())
print(data['Seller_Type'].unique())
print(data['Transmission'].unique())
print(data['Owner'].unique())
```

```
['Petrol' 'Diesel' 'CNG']
['Dealer' 'Individual']
['Manual' 'Automatic']
[0 1 3]
```

In [16]:
```
#chech for missing and null value
data.isnull().sum()
#since all the vakues are zero so no null values
```

Out[16]:
```
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 [17]:
```
data.describe() #basic statistical details for Quantitative variable
```

Out[17]:

	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

In [25]:
```
data.columns #will give names of all column in the data set
```

Out[25]:
```
Index(['Car_Name', 'Year', 'Selling_Price', 'Present_Price', 'Kms_Driven',
       'Fuel_Type', 'Seller_Type', 'Transmission', 'Owner'],
      dtype='object')
```

In [26]:
```
# creating a new data frame from existing data which wil contain only useful data
df = data[['Year', 'Selling_Price', 'Present_Price', 'Kms_Driven',
       'Fuel_Type', 'Seller_Type', 'Transmission', 'Owner']]
df.head()
```

Out[26]:

	Year	Selling_Price	Present_Price	Kms_Driven	Fuel_Type	Seller_Type	Transmission	Owner
0	2014	3.35	5.59	27000	Petrol	Dealer	Manual	0
1	2013	4.75	9.54	43000	Diesel	Dealer	Manual	0
2	2017	7.25	9.85	6900	Petrol	Dealer	Manual	0
3	2011	2.85	4.15	5200	Petrol	Dealer	Manual	0
4	2014	4.60	6.87	42450	Diesel	Dealer	Manual	0

In [28]:
```
#adding new column to my finaldata set
df['Current_Year'] = 2020
df['Number_Of_Years'] = df['Current_Year'] - df['Year']
df.head()
```

Out[28]:

	Year	Selling_Price	Present_Price	Kms_Driven	Fuel_Type	Seller_Type	Transmission	Owner	Curre
0	2014	3.35	5.59	27000	Petrol	Dealer	Manual	0	
1	2013	4.75	9.54	43000	Diesel	Dealer	Manual	0	
2	2017	7.25	9.85	6900	Petrol	Dealer	Manual	0	
3	2011	2.85	4.15	5200	Petrol	Dealer	Manual	0	
4	2014	4.60	6.87	42450	Diesel	Dealer	Manual	0	

In [29]:
```
#dropping some of the column in dataset which are of no use
df.drop(['Year'],axis=1,inplace=True)
```

In [30]:
```
df.drop(['Current_Year'],axis=1,inplace=True)
```

In [30]:
```
df.head()
```

Out[30]:

	Selling_Price	Present_Price	Kms_Driven	Fuel_Type	Seller_Type	Transmission	Owner	Current_Yea
0	3.35	5.59	27000	Petrol	Dealer	Manual	0	202(
1	4.75	9.54	43000	Diesel	Dealer	Manual	0	202(
2	7.25	9.85	6900	Petrol	Dealer	Manual	0	202(
3	2.85	4.15	5200	Petrol	Dealer	Manual	0	202(
4	4.60	6.87	42450	Diesel	Dealer	Manual	0	202(

In [31]:
```
#in this stage we are providing dummy variable to each categorical variable it will ga

df = pd.get_dummies(df,drop_first=True)
#frop first is true it will drop one of the category fro  the column to prevent from c
df.head()
```

Out[31]:

	Selling_Price	Present_Price	Kms_Driven	Owner	Current_Year	Number_Of_Years	Fuel_Type_Diesel
0	3.35	5.59	27000	0	2020	6	0
1	4.75	9.54	43000	0	2020	7	1
2	7.25	9.85	6900	0	2020	3	0
3	2.85	4.15	5200	0	2020	9	0
4	4.60	6.87	42450	0	2020	6	1

In [32]:
```
#calculating correlation
df.corr()
```

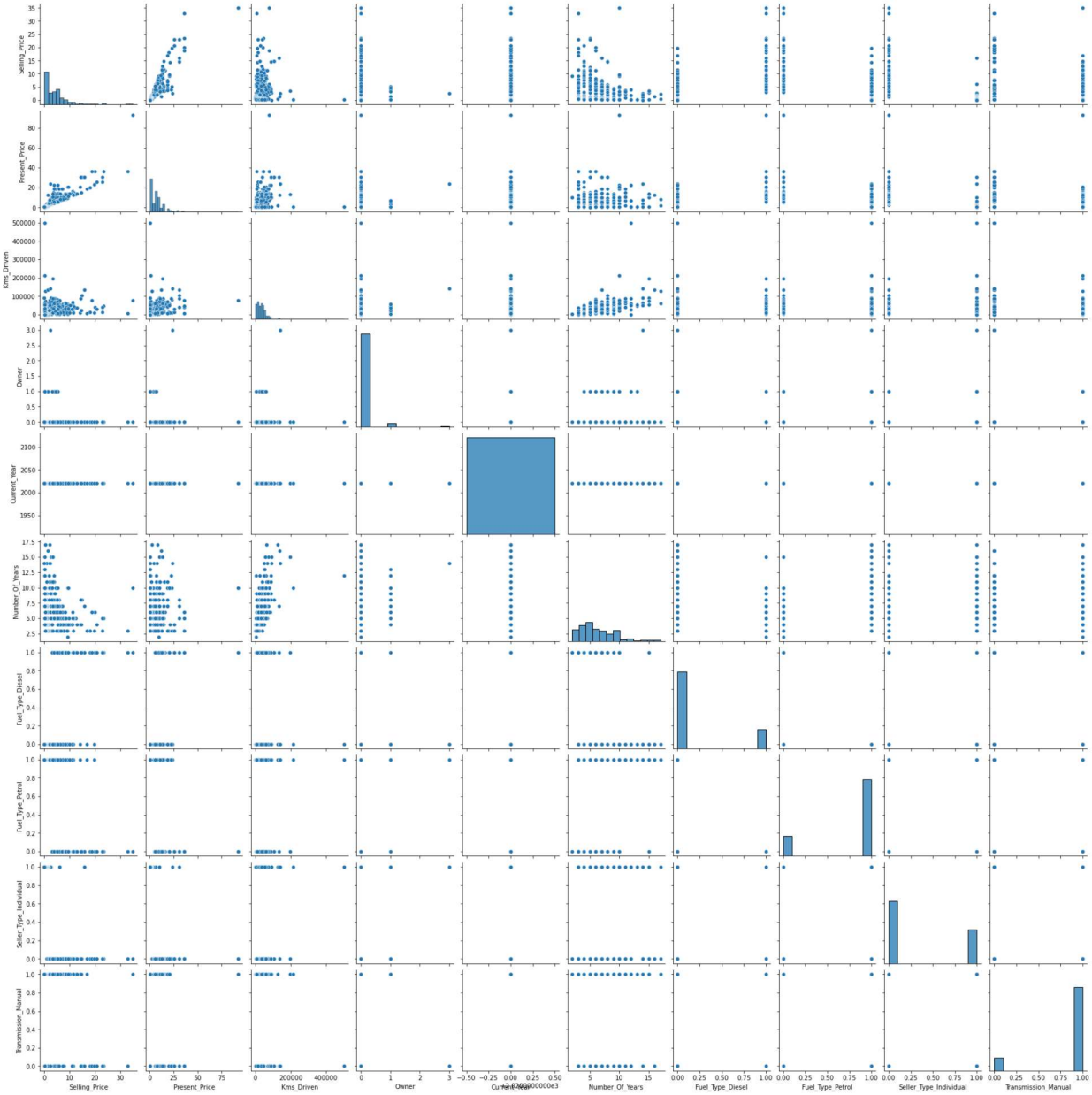
Out[32]:

	Selling_Price	Present_Price	Kms_Driven	Owner	Current_Year	Number_Of_Y
Selling_Price	1.000000	0.878983	0.029187	-0.088344	NaN	-0.236
Present_Price	0.878983	1.000000	0.203647	0.008057	NaN	0.047
Kms_Driven	0.029187	0.203647	1.000000	0.089216	NaN	0.524
Owner	-0.088344	0.008057	0.089216	1.000000	NaN	0.182
Current_Year	NaN	NaN	NaN	NaN	NaN	N
Number_Of_Years	-0.236141	0.047584	0.524342	0.182104	NaN	1.000
Fuel_Type_Diesel	0.552339	0.473306	0.172515	-0.053469	NaN	-0.064
Fuel_Type_Petrol	-0.540571	-0.465244	-0.172874	0.055687	NaN	0.059
Seller_Type_Individual	-0.550724	-0.512030	-0.101419	0.124269	NaN	0.039
Transmission_Manual	-0.367128	-0.348715	-0.162510	-0.050316	NaN	-0.000

In [33]:
```
import seaborn as sns
```

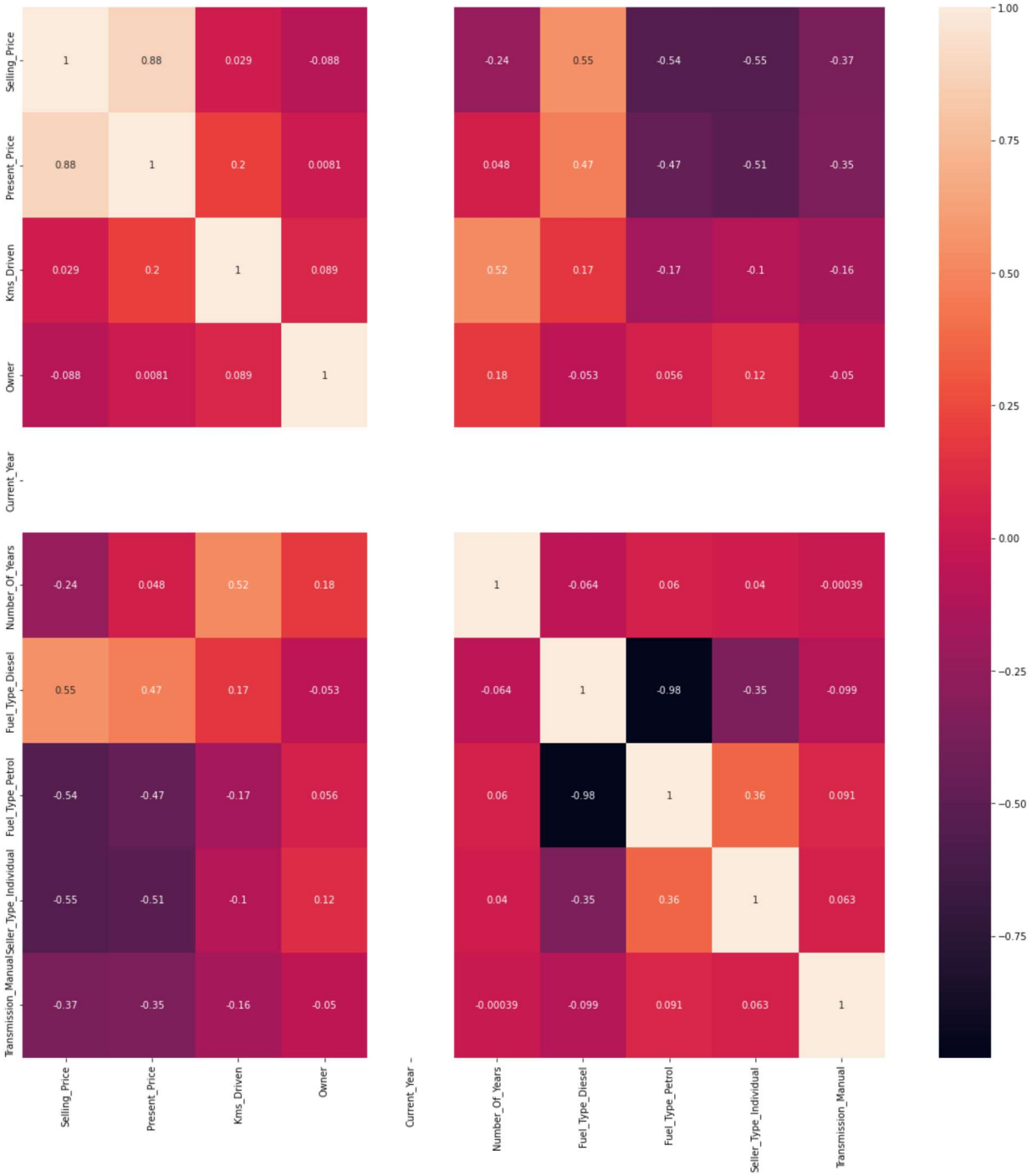
In [34]:
```
sns.pairplot(df)
```

Out[34]:
```
<seaborn.axisgrid.PairGrid at 0x170fc9b0700>
```



In [35]:
```
import matplotlib.pyplot as plt
%matplotlib inline
```

In [36]:
```
x = df.corr()
top_corr_feature = x.index
plt.figure(figsize=(20,20))
#plotting heat map
g=sns.heatmap(df[top_corr_feature].corr(),annot = True)
```



In [ ]:

In [37]:
```
df.head()
```

Out[37]:

	Selling_Price	Present_Price	Kms_Driven	Owner	Current_Year	Number_Of_Years	Fuel_Type_Diesel
0	3.35	5.59	27000	0	2020	6	0
1	4.75	9.54	43000	0	2020	7	1
2	7.25	9.85	6900	0	2020	3	0
3	2.85	4.15	5200	0	2020	9	0
4	4.60	6.87	42450	0	2020	6	1

In [38]:
```
#defining independent and dependent feature
X=df.iloc[:,1:] #[row, col]
Y=df.iloc[:,0]
```

In [39]:
```
X.head()
```

Out[39]:

	Present_Price	Kms_Driven	Owner	Current_Year	Number_Of_Years	Fuel_Type_Diesel	Fuel_Type_Pet
0	5.59	27000	0	2020	6	0	
1	9.54	43000	0	2020	7	1	
2	9.85	6900	0	2020	3	0	
3	4.15	5200	0	2020	9	0	
4	6.87	42450	0	2020	6	1	

In [40]:
```
Y.head()
```

Out[40]:
```
0    3.35
1    4.75
2    7.25
3    2.85
4    4.60
Name: Selling_Price, dtype: float64
```

In [41]:
```
#feature importance to undestand which feature is important

from sklearn.ensemble import ExtraTreesRegressor as etr
model = etr()
model.fit(X,Y)
```

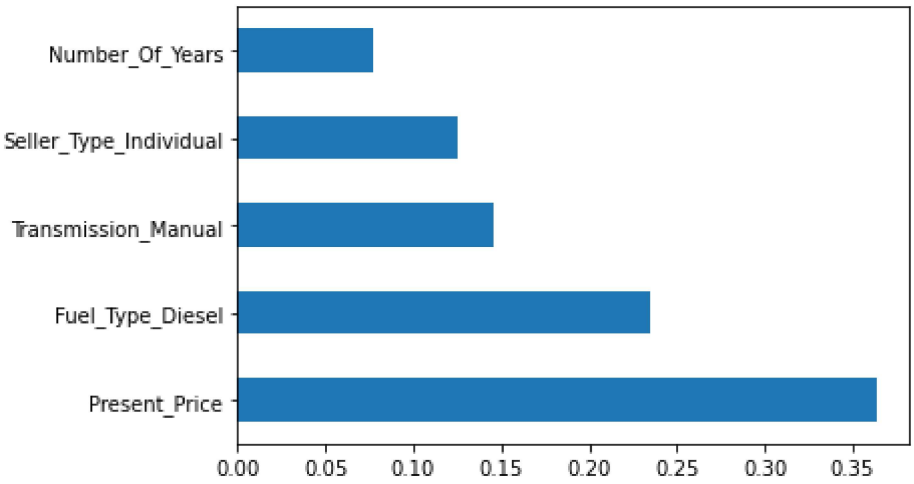
Out[41]:
```
ExtraTreesRegressor()
```

In [42]:
```
print(model.feature_importances_) #it will give feature importance of X in same sequer
```

```
[0.36385218 0.04084952 0.00045779 0.         0.0774205  0.2342227
 0.0120227  0.12557436 0.14560025]
```

In [43]:
```
 #we can print these feature importance to get better undestanding
fet_imp = pd.Series(model.feature_importances_,index= X.columns)
fet_imp.nlargest(5).plot(kind='barh') #nlargest will plot only 5 largest values
plt.show
```

Out[43]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



In [44]:
```
from sklearn.model_selection import train_test_split
X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size =0.2)
```

In [45]:
```
X_train
X_train.shape
```

Out[45]:
```
(240, 9)
```

In [46]:
```
#using random forest regressor
from sklearn.ensemble import RandomForestRegressor
rf_random=RandomForestRegressor()
```

In [47]:
```
#hyper parameter
import numpy as np
n_estimators = [int(x) for x in np.linspace(start=100,stop=1200,num=12)]
print(n_estimators)
```

```
[100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1100, 1200]
```

In [60]:
```
#randomized serchedCV
#number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start=100,stop=1200,num=12)]
#number of feature to consider at every split
max_features = ['auto','sqrt']
#maximus number of levels in tree
max_depth = [int(x) for x in np.linspace(5,30,num=6)]
#minimum number of samples require to split a node
min_samples_split = [2,5,10,15,100]
#min number of sample require at leaf node
min_samples_leaf = [1,2,5,10]
```

In [51]:
```
from sklearn.model_selection import RandomizedSearchCV
```

In [61]:
```
#create random grid
random_grid = {'n_estimators':n_estimators,
               'max_features':max_features,
                'max_depth':max_depth,
                 'min_samples_split':min_samples_split,
```

```
                'min_samples_leaf':min_samples_leaf}
print(random_grid)
```

```
{'n_estimators': [100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1100, 1200], 'ma
x_features': ['auto', 'sqrt'], 'max_depth': [5, 10, 15, 20, 25, 30], 'min_samples_spl
it': [2, 5, 10, 15, 100], 'min_samples_leaf': [1, 2, 5, 10]}
```

In [62]:
```
#use random grid to searchfor best hyper parameter
#create base model
rf = RandomForestRegressor()
```

In [63]:
```
rf_random = RandomizedSearchCV(estimator = rf, param_distributions=random_grid,scoring
```

In [64]:
```
rf_random.fit(X_train,Y_train)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=5, min_samples_split=5, n_
estimators=900; total time=   1.5s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=5, min_samples_split=5, n_
estimators=900; total time=   1.7s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=5, min_samples_split=5, n_
estimators=900; total time=   1.2s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=5, min_samples_split=5, n_
estimators=900; total time=   1.1s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=5, min_samples_split=5, n_
estimators=900; total time=   1.1s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n
_estimators=1100; total time=   1.4s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n
_estimators=1100; total time=   1.4s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n
_estimators=1100; total time=   1.4s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n
_estimators=1100; total time=   1.4s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n
_estimators=1100; total time=   1.5s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=100,
n_estimators=300; total time=   0.3s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=100,
n_estimators=300; total time=   0.3s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=100,
n_estimators=300; total time=   0.3s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=100,
n_estimators=300; total time=   0.4s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=100,
n_estimators=300; total time=   0.4s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=5, n_
estimators=400; total time=   0.7s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=5, n_
estimators=400; total time=   0.7s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=5, n_
estimators=400; total time=   0.6s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=5, n_
estimators=400; total time=   0.5s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=5, n_
estimators=400; total time=   0.5s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=10, min_samples_split=5, n
_estimators=700; total time=   0.9s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=10, min_samples_split=5, n
_estimators=700; total time=   0.9s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=10, min_samples_split=5, n
_estimators=700; total time=   0.9s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=10, min_samples_split=5, n
_estimators=700; total time=   0.9s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=10, min_samples_split=5, n
_estimators=700; total time=   0.9s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=1, min_samples_split=2, n_
estimators=1000; total time=   1.4s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=1, min_samples_split=2, n_
estimators=1000; total time=   1.4s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=1, min_samples_split=2, n_
estimators=1000; total time=   1.4s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=1, min_samples_split=2, n_
estimators=1000; total time=   1.4s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=1, min_samples_split=2, n_
```

```
estimators=1000; total time=   1.6s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=10, min_samples_split=15, n
_estimators=1100; total time=   1.9s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=10, min_samples_split=15, n
_estimators=1100; total time=   1.9s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=10, min_samples_split=15, n
_estimators=1100; total time=   1.3s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=10, min_samples_split=15, n
_estimators=1100; total time=   1.3s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=10, min_samples_split=15, n
_estimators=1100; total time=   1.3s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=1, min_samples_split=15, n
_estimators=300; total time=   0.3s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=1, min_samples_split=15, n
_estimators=300; total time=   0.3s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=1, min_samples_split=15, n
_estimators=300; total time=   0.3s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=1, min_samples_split=15, n
_estimators=300; total time=   0.3s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=1, min_samples_split=15, n
_estimators=300; total time=   0.3s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_
estimators=700; total time=   1.0s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_
estimators=700; total time=   0.9s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_
estimators=700; total time=   1.0s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_
estimators=700; total time=   0.9s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_
estimators=700; total time=   1.0s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_split=15, n
_estimators=700; total time=   1.2s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_split=15, n
_estimators=700; total time=   1.2s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_split=15, n
_estimators=700; total time=   1.1s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_split=15, n
_estimators=700; total time=   0.9s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_split=15, n
_estimators=700; total time=   1.0s
```

Out[64]:

```
RandomizedSearchCV(cv=5, estimator=RandomForestRegressor(), n_jobs=1,
                   param_distributions={'max_depth': [5, 10, 15, 20, 25, 30],
                                        'max_features': ['auto', 'sqrt'],
                                        'min_samples_leaf': [1, 2, 5, 10],
                                        'min_samples_split': [2, 5, 10, 15,
                                                              100],
                                        'n_estimators': [100, 200, 300, 400,
                                                         500, 600, 700, 800,
                                                         900, 1000, 1100,
                                                         1200]},
                   random_state=42, scoring='neg_mean_squared_error',
                   verbose=2)
```

In [65]:

```
prediction = rf_random.predict(X_test)
```

In [66]:

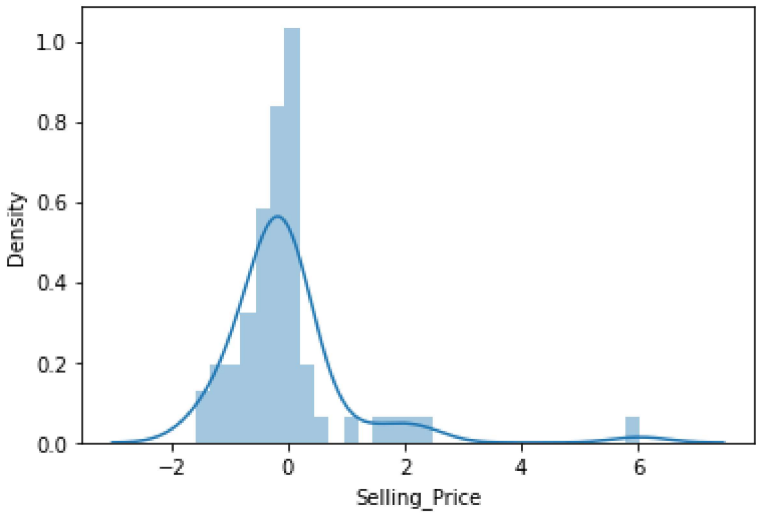
```
prediction
```

Out[66]:
```
array([ 1.00386,  0.43282,  4.5179 ,  5.81194,  2.90844, 10.96677,
        6.90219,  2.30097, 10.83033,  4.69215,  0.51685,  2.8197 ,
        5.50395, 20.81549,  5.30535,  7.82126,  3.7104 , 11.3751 ,
        1.15888,  4.1446 ,  0.52154,  0.52163,  1.40623,  0.54137,
        5.1113 , 10.11654,  8.83546,  4.5184 ,  0.75603,  0.62863,
       15.79207,  8.93569,  6.50605,  3.5527 ,  0.22398,  2.43092,
        5.2502 ,  0.69895,  3.74855,  5.952  ,  0.4849 ,  3.13717,
        0.60088,  5.51483,  0.66077,  4.3787 ,  5.7938 , 21.28182,
        1.14332,  4.45099,  3.04225,  4.18355,  0.51642,  5.08035,
        9.32222,  0.54229,  0.27577,  0.45903,  2.91212,  5.08035,
        0.56717])
```

In [67]:
```
sns.distplot(Y_test-prediction)
```

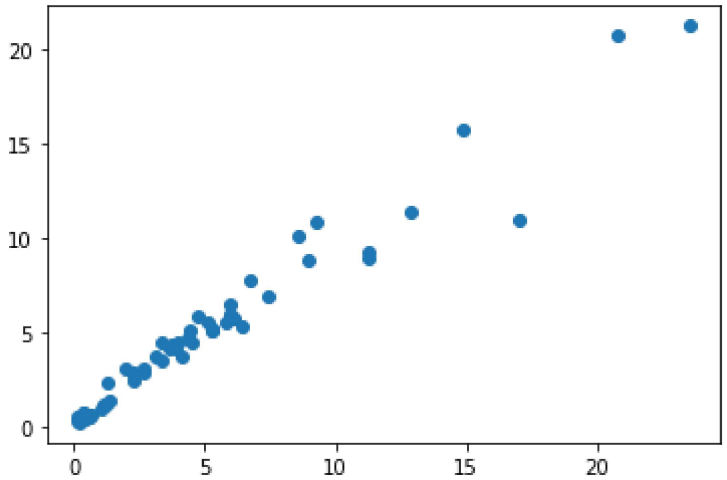
```
C:\Users\Harsh\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[67]:
```
<AxesSubplot:xlabel='Selling_Price', ylabel='Density'>
```



In [68]:
```
plt.scatter(Y_test,prediction)
```

Out[68]:
```
<matplotlib.collections.PathCollection at 0x1708418a700>
```



In [69]:
```
from sklearn.metrics import mean_squared_error
```

In [70]:
```
mean_squared_error(Y_test,prediction)
```

Out[70]:
```
1.1614358062409884
```

In [71]:
```
from sklearn.metrics import r2_score
```

In [72]:
```
r2_score(Y_test,prediction)
```

Out[72]:
```
0.9532086238479118
```

In [86]:
```
Y_test.head()
```

Out[86]:
```
120    1.05
163    0.45
0      3.35
1      4.75
14     2.25
Name: Selling_Price, dtype: float64
```

In [84]:
```
y_test = Y_test.to_numpy()
y_test
```

Out[84]:
```
array([ 1.05,  0.45,  3.35,  4.75,  2.25, 17.  ,  7.45,  1.25,  9.25,
        4.35,  0.3 ,  2.25,  5.11, 20.75,  6.45,  6.7 ,  4.15, 12.9 ,
        1.15,  3.9 ,  0.5 ,  0.45,  1.35,  0.6 ,  5.25,  8.55,  8.99,
        4.5 ,  0.35,  0.65, 14.9 , 11.25,  5.95,  3.35,  0.2 ,  2.25,
        5.25,  0.35,  3.1 ,  5.95,  0.6 ,  2.  ,  0.48,  5.85,  0.45,
        3.75,  6.15, 23.5 ,  1.2 ,  4.  ,  2.7 ,  3.65,  0.25,  5.25,
       11.25,  0.42,  0.15,  0.38,  2.65,  4.4 ,  0.17])
```

In [73]:

In [87]:

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Input In [87], in <cell line: 1>()
----> 1 accuracy_score(y_test,prediction,normalize= False)

File ~\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:211, in accuracy_score(y_true, y_pred, normalize, sample_weight)
    145 """Accuracy classification score.
    146
    147 In multilabel classification, this function computes subset accuracy:
   (...)
    207 0.5
    208 """
    210 # Compute accuracy for each possible representation
--> 211 y_type, y_true, y_pred = _check_targets(y_true, y_pred)
    212 check_consistent_length(y_true, y_pred, sample_weight)
    213 if y_type.startswith("multilabel"):

File ~\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:104, in _check_targets(y_true, y_pred)
    102 # No metrics support "multiclass-multioutput" format
    103 if y_type not in ["binary", "multiclass", "multilabel-indicator"]:
--> 104     raise ValueError("{0} is not supported".format(y_type))
    106 if y_type in ["binary", "multiclass"]:
    107     y_true = column_or_1d(y_true)

ValueError: continuous is not supported
```

In [ ]: