In [1]:

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

In [2]:

```python
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	Owner
0	ritz	2014	3.35	5.59	27000	Petrol	Dealer	Manual	0
1	sx4	2013	4.75	9.54	43000	Diesel	Dealer	Manual	0
2	ciaz	2017	7.25	9.85	6900	Petrol	Dealer	Manual	0
3	wagon r	2011	2.85	4.15	5200	Petrol	Dealer	Manual	0
4	swift	2014	4.60	6.87	42450	Diesel	Dealer	Manual	0

In [3]:

```python
df.shape
```

Out[3]:

```
(301, 9)
```

In [4]:

```python
df.isnull().sum()
```

Out[4]:

```
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 [5]:

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

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

In [6]:

```python
df.describe()
```

Out[6]:

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

In [7]:

```python
final_dataset=df[['Year','Selling_Price','Present_Price','Kms_Driven','Fuel_Type','Seller_Type','Transmission','Owner']]
```

In [8]:

```python
final_dataset.head()
```

Out[8]:

	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 [9]:

```python
final_dataset['Current Year']=2022
```

In [10]:

```python
final_dataset.head()
```

Out[10]:

	Year	Selling_Price	Present_Price	Kms_Driven	Fuel_Type	Seller_Type	Transmission	Owner	Current Year
0	2014	3.35	5.59	27000	Petrol	Dealer	Manual	0	2022
1	2013	4.75	9.54	43000	Diesel	Dealer	Manual	0	2022
2	2017	7.25	9.85	6900	Petrol	Dealer	Manual	0	2022
3	2011	2.85	4.15	5200	Petrol	Dealer	Manual	0	2022
4	2014	4.60	6.87	42450	Diesel	Dealer	Manual	0	2022

In [11]:

```python
final_dataset['no_year']=final_dataset['Current Year']- final_dataset['Year']
```

In [12]:

```python
final_dataset.head()
```

Out[12]:

	Year	Selling_Price	Present_Price	Kms_Driven	Fuel_Type	Seller_Type	Transmission	Owner	Current Year	no_year
0	2014	3.35	5.59	27000	Petrol	Dealer	Manual	0	2022	8
1	2013	4.75	9.54	43000	Diesel	Dealer	Manual	0	2022	9
2	2017	7.25	9.85	6900	Petrol	Dealer	Manual	0	2022	5
3	2011	2.85	4.15	5200	Petrol	Dealer	Manual	0	2022	11
4	2014	4.60	6.87	42450	Diesel	Dealer	Manual	0	2022	8

In [13]:

```python
final_dataset.drop(['Year'],axis=1,inplace=True)
final_dataset.drop(['Current Year'],axis=1,inplace=True)
```

In [14]:

```python
final_dataset.head()
```

Out[14]:

	Selling_Price	Present_Price	Kms_Driven	Fuel_Type	Seller_Type	Transmission	Owner	no_year
0	3.35	5.59	27000	Petrol	Dealer	Manual	0	8
1	4.75	9.54	43000	Diesel	Dealer	Manual	0	9
2	7.25	9.85	6900	Petrol	Dealer	Manual	0	5
3	2.85	4.15	5200	Petrol	Dealer	Manual	0	11
4	4.60	6.87	42450	Diesel	Dealer	Manual	0	8

In [15]:

```python
final_dataset=pd.get_dummies(final_dataset,drop_first=True)
```

In [16]:

```python
final_dataset.head()
```

Out[16]:

	Selling_Price	Present_Price	Kms_Driven	Owner	no_year	Fuel_Type_Diesel	Fuel_Type_Petrol	Seller_Type_Individual	Transmission_Manual
0	3.35	5.59	27000	0	8	0	1	0	1
1	4.75	9.54	43000	0	9	1	0	0	1
2	7.25	9.85	6900	0	5	0	1	0	1
3	2.85	4.15	5200	0	11	0	1	0	1
4	4.60	6.87	42450	0	8	1	0	0	1

In [17]:

```python
final_dataset.corr()
```

Out[17]:

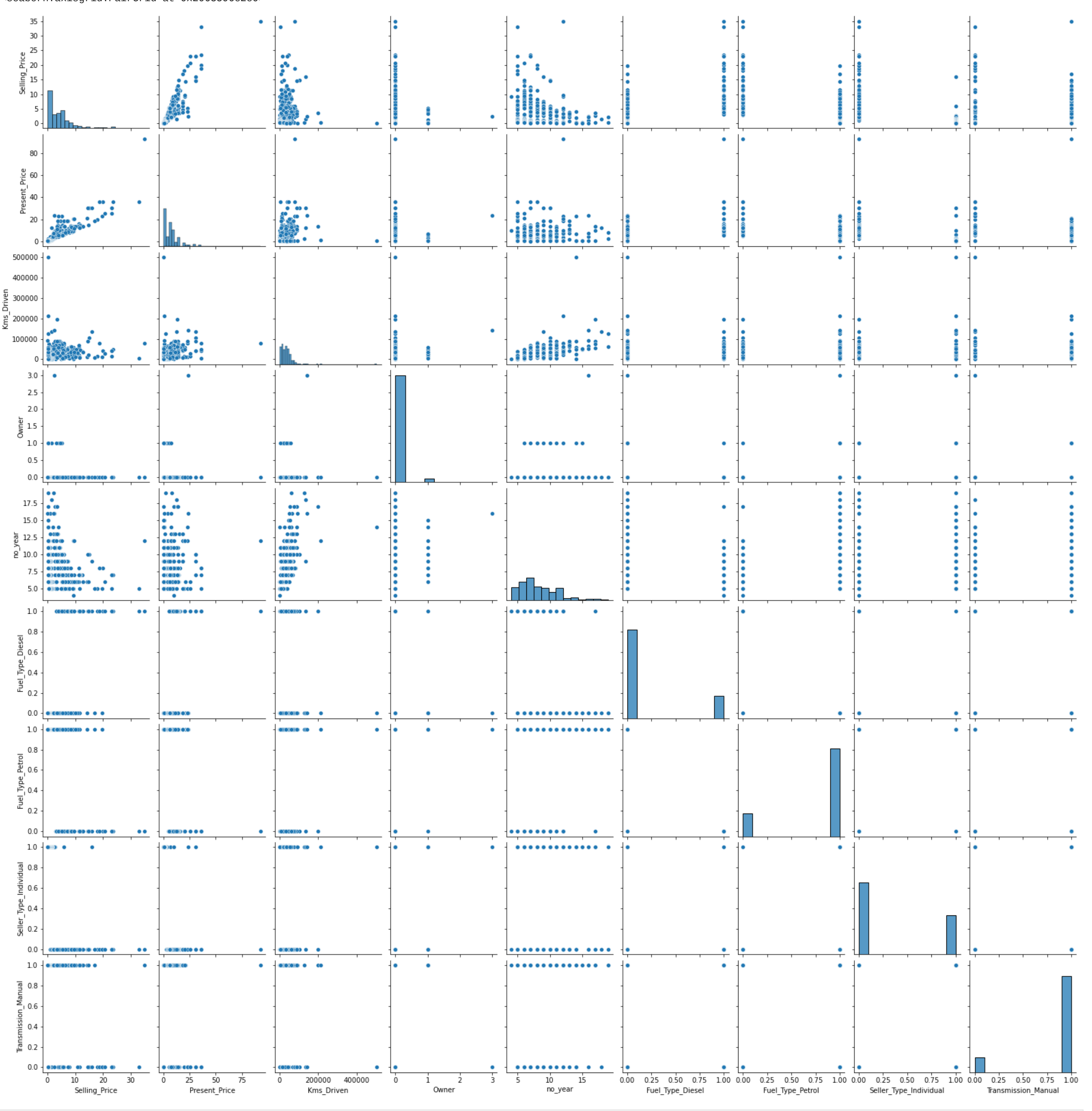
	Selling_Price	Present_Price	Kms_Driven	Owner	no_year	Fuel_Type_Diesel	Fuel_Type_Petrol	Seller_Type_Individual	Transmission_Manual
Selling_Price	1.000000	0.878983	0.029187	-0.088344	-0.236141	0.552339	-0.540571	-0.550724	-0.367128
Present_Price	0.878983	1.000000	0.203647	0.008057	0.047584	0.473306	-0.465244	-0.512030	-0.348715
Kms_Driven	0.029187	0.203647	1.000000	0.089216	0.524342	0.172515	-0.172874	-0.101419	-0.162510
Owner	-0.088344	0.008057	0.089216	1.000000	0.182104	-0.053469	0.055687	0.124269	-0.050316
no_year	-0.236141	0.047584	0.524342	0.182104	1.000000	-0.064315	0.059959	0.039896	-0.000394
Fuel_Type_Diesel	0.552339	0.473306	0.172515	-0.053469	-0.064315	1.000000	-0.979648	-0.350467	-0.098643
Fuel_Type_Petrol	-0.540571	-0.465244	-0.172874	0.055687	0.059959	-0.979648	1.000000	0.358321	0.091013
Seller_Type_Individual	-0.550724	-0.512030	-0.101419	0.124269	0.039896	-0.350467	0.358321	1.000000	0.063240
Transmission_Manual	-0.367128	-0.348715	-0.162510	-0.050316	-0.000394	-0.098643	0.091013	0.063240	1.000000

In [18]:

```python
sns.pairplot(final_dataset)
```

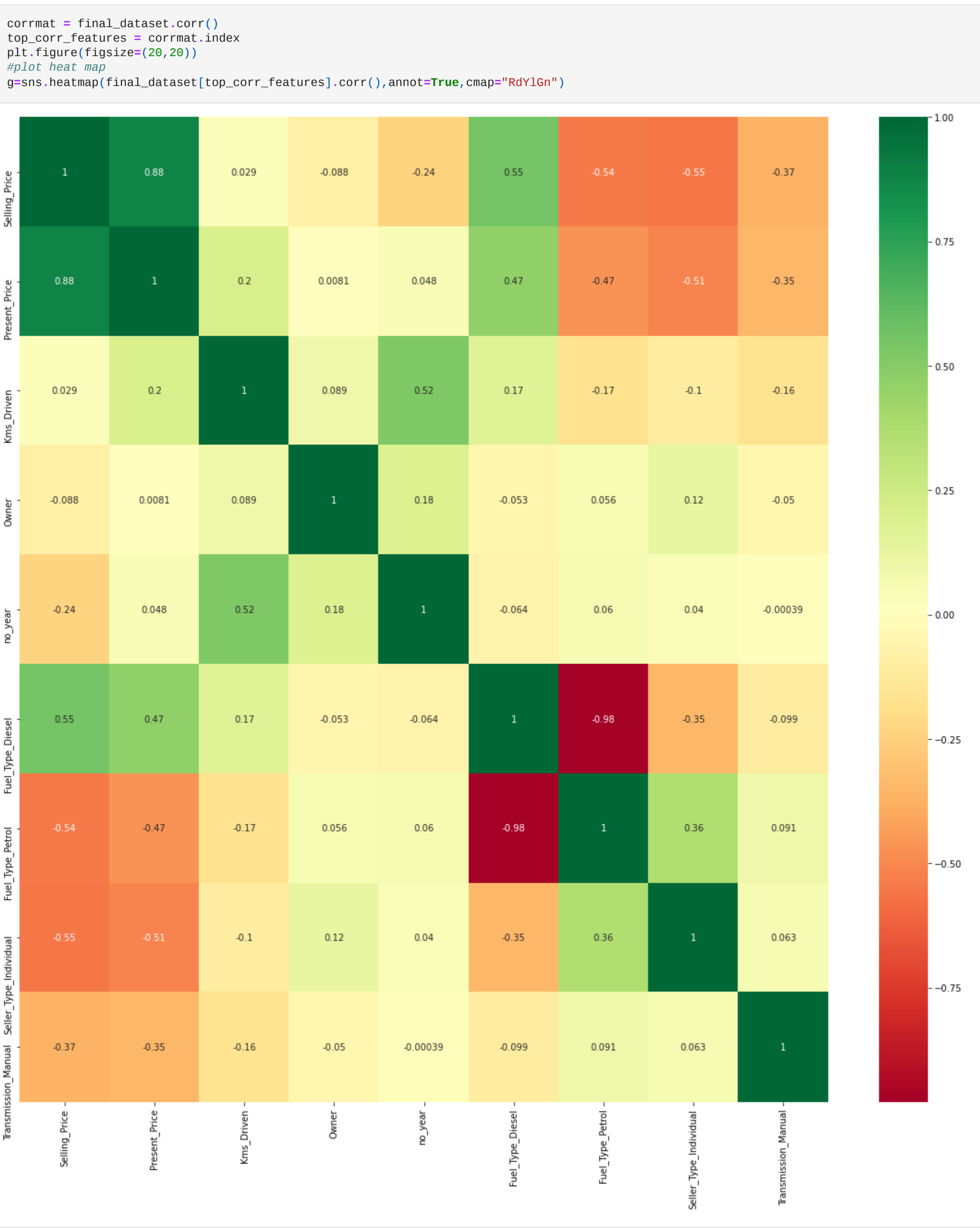
Out[18]:

```
<seaborn.axisgrid.PairGrid at 0x2068596e2e0>
```



In [19]:

```python
corrmat = final_dataset.corr()
top_corr_features = corrmat.index
plt.figure(figsize=(20,20))
#plot heat map
g=sns.heatmap(final_dataset[top_corr_features].corr(),annot=True,cmap="RdYlGn")
```



In [20]:

```python
x=final_dataset.drop(['Selling_Price'],axis=1)
y=final_dataset['Selling_Price']
```

In [21]:

```python
x.head()
```

Out[21]:

	Present_Price	Kms_Driven	Owner	no_year	Fuel_Type_Diesel	Fuel_Type_Petrol	Seller_Type_Individual	Transmission_Manual
0	5.59	27000	0	8	0	1	0	1
1	9.54	43000	0	9	1	0	0	1
2	9.85	6900	0	5	0	1	0	1
3	4.15	5200	0	11	0	1	0	1
4	6.87	42450	0	8	1	0	0	1

In [22]:

```python
y.head()
```

Out[22]:

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

Feature importance checking

Extra Tree Regressor is use to check which are the importance feature in this datasets & which are not.

In [23]:

```python
from sklearn.ensemble import ExtraTreesRegressor
```

In [24]:

```python
model=ExtraTreesRegressor()
```

In [25]:

```python
model.fit(x,y)
```

Out[25]:

```
ExtraTreesRegressor()
```

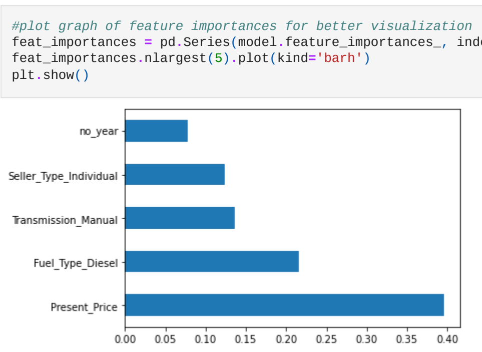
In [26]:

```python
print(model.feature_importances_)
```

```
[3.96234216e-01 4.06737581e-02 2.94386327e-04 7.79593949e-02
 2.16200228e-01 8.57559231e-03 1.23728867e-01 1.36333558e-01]
```

In [27]:

```python
#plot graph of feature importances for better visualization
feat_importances = pd.Series(model.feature_importances_, index=x.columns)
feat_importances.nlargest(5).plot(kind='barh')
plt.show()
```



In [28]:

```python
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 [29]:

```python
from sklearn.ensemble import RandomForestRegressor
```

HYPERPARAMETER TUNING

In [30]:

```python
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 [31]:

```python
from sklearn.model_selection import RandomizedSearchCV
```

In [32]:

```python
#Randomized Search CV

# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 100, stop = 1200, num = 12)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(5, 30, num = 6)]
# max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10, 15, 100]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 5, 10]
```

In [33]:

```python
# Create the 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], 'max_features': ['auto', 'sqrt'], 'max_depth': [5, 10, 15, 20, 25, 30], 'min_samples_split': [2, 5, 10, 15, 100], 'min_samples_leaf': [1, 2, 5, 10]}
```

In [34]:

```python
rf = RandomForestRegressor()
```

In [35]:

```python
rf_random = RandomizedSearchCV(estimator = rf, param_distributions = random_grid,scoring='neg_mean_squared_error', n_iter = 10, cv = 5, verbose=2, random_state=42, n_jobs = -1)
```

In [36]:

```python
rf_random.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
```

Out[36]:

```
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 [37]:

```python
rf_random.best_params_
```

Out[37]:

```
{'n_estimators': 1000,
 'min_samples_split': 2,
 'min_samples_leaf': 1,
 'max_features': 'sqrt',
 'max_depth': 25}
```

In [38]:

```python
predition = rf_random.predict(X_test)
```

In [39]:

```python
predition
```

Out[39]:

```
array([ 4.4422 ,  4.1966 ,  2.8974 ,  3.75851,  0.94295,  4.83198,
        4.63905,  0.51226,  4.62238,  7.69968,  1.4181 ,  0.47679,
       19.7589 ,  9.41492,  0.96747,  8.4545 ,  0.66826,  0.61162,
       13.10156,  0.9198 ,  0.88532,  0.74269,  1.0072 ,  4.88912,
        6.86339, 10.2402 ,  5.63783,  6.22873,  2.83336, 11.02798,
        0.54862,  0.4558 ,  6.11584,  4.42008,  3.7779 ,  0.68074,
        2.31865,  3.43232,  0.95022,  1.33602,  0.74304,  3.4631 ,
        0.54592,  2.53845,  2.35074,  9.54179,  1.2443 ,  0.67101,
        8.19261,  4.22226,  5.40478,  8.57167,  2.8195 ,  4.59   ,
        0.60865,  3.73006,  6.25788,  0.47274,  7.4058 ,  6.48212,
        1.21582])
```

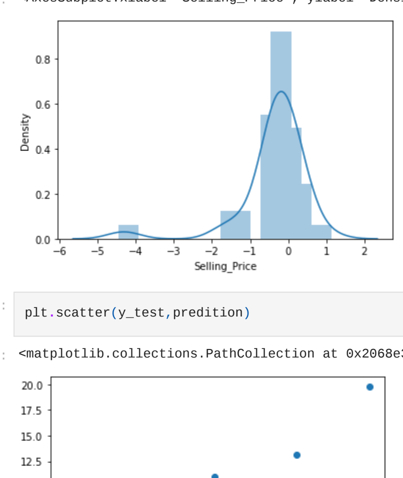
In [40]:

```python
sns.distplot(y_test-predition)
```

```
C:\Users\lenovo\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)
```

Out[40]:

```
<AxesSubplot:xlabel='Selling_Price', ylabel='Density'>
```

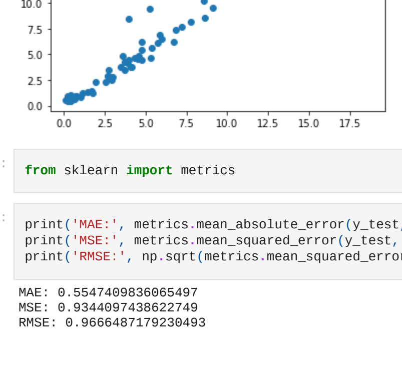


In [41]:

```python
plt.scatter(y_test,predition)
```

Out[41]:

```
<matplotlib.collections.PathCollection at 0x2068e309400>
```



In [42]:

```python
from sklearn import metrics
```

In [43]:

```python
print('MAE:', metrics.mean_absolute_error(y_test, predition))
print('MSE:', metrics.mean_squared_error(y_test, predition))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, predition)))
```

```
MAE: 0.5547409836065497
MSE: 0.9344097438622749
RMSE: 0.9666487179230493
```