In [1]:
```
import pandas as pd
```

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

In [3]:
```
df
```

Out[3]:

	Car_Name	Year	Selling_Price	Present_Price	Kms_Driven	Fuel_Type	Seller_Type	Transmissio
0	ritz	2014	3.35	5.59	27000	Petrol	Dealer	Manu
1	sx4	2013	4.75	9.54	43000	Diesel	Dealer	Manu
2	ciaz	2017	7.25	9.85	6900	Petrol	Dealer	Manu
3	wagon r	2011	2.85	4.15	5200	Petrol	Dealer	Manu
4	swift	2014	4.60	6.87	42450	Diesel	Dealer	Manu
...	...	...	...	...	...	...	...	.
296	city	2016	9.50	11.60	33988	Diesel	Dealer	Manu
297	brio	2015	4.00	5.90	60000	Petrol	Dealer	Manu
298	city	2009	3.35	11.00	87934	Petrol	Dealer	Manu
299	city	2017	11.50	12.50	9000	Diesel	Dealer	Manu
300	brio	2016	5.30	5.90	5464	Petrol	Dealer	Manu

301 rows × 9 columns

In [4]:
```
df.head()
```

Out[4]:

	Car_Name	Year	Selling_Price	Present_Price	Kms_Driven	Fuel_Type	Seller_Type	Transmission
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 [5]:
```
df.shape
```

Out[5]:
```
(301, 9)
```

In [6]:
```
df['Seller_Type'].unique()
```

Out[6]:
```
array(['Dealer', 'Individual'], dtype=object)
```

In [7]:
```
print(df['Seller_Type'].unique())
```

```
['Dealer' 'Individual']
```

In [8]:
```
df['Transmission'].unique()
```

Out[8]:
```
array(['Manual', 'Automatic'], dtype=object)
```

In [9]:
```
df['Fuel_Type'].unique()
```

Out[9]:
```
array(['Petrol', 'Diesel', 'CNG'], dtype=object)
```

In [10]:
```
df['Owner'].unique()
```

Out[10]:
```
array([0, 1, 3])
```

In [11]:
```
print([print(feature,"=",df[feature].unique()) for feature in ['Fuel_Type','Sell
```

```
Fuel_Type = ['Petrol' 'Diesel' 'CNG']
Seller_Type = ['Dealer' 'Individual']
Transmission = ['Manual' 'Automatic']
Owner = [0 1 3]
[None, None, None, None]
```

In [12]:
```
##check missing or null values
df.isnull().sum()
```

Out[12]:
```
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 [13]:
```
df.describe()
```

Out[13]:

	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 [14]:
```
df.columns
```

Out[14]:
```
Index(['Car_Name', 'Year', 'Selling_Price', 'Present_Price', 'Kms_Driven',
       'Fuel_Type', 'Seller_Type', 'Transmission', 'Owner'],
      dtype='object')
```

In [15]:
```
final_dataset = df[['Year', 'Selling_Price', 'Present_Price', 'Kms_Driven','Fuel
```

In [16]:
```
final_dataset.head()
```

Out[16]:

	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 [17]:
```
final_dataset['Current_Year'] = 2021
```

In [18]:
```
final_dataset.head()
```

Out[18]:

	Year	Selling_Price	Present_Price	Kms_Driven	Fuel_Type	Seller_Type	Transmission	Owner	Cu
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 [19]:
```
final_dataset['no_year']=final_dataset['Current_Year']-final_dataset['Year']
```

In [20]:
```
final_dataset
```

Out[20]:

	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	

	Year	Selling_Price	Present_Price	Kms_Driven	Fuel_Type	Seller_Type	Transmission	Owner
...	...	...	...	...	...	...	...	...
296	2016	9.50	11.60	33988	Diesel	Dealer	Manual	0
297	2015	4.00	5.90	60000	Petrol	Dealer	Manual	0
298	2009	3.35	11.00	87934	Petrol	Dealer	Manual	0
299	2017	11.50	12.50	9000	Diesel	Dealer	Manual	0
300	2016	5.30	5.90	5464	Petrol	Dealer	Manual	0

301 rows × 10 columns

In [21]:
```
final_dataset.head()
```

Out[21]:

	Year	Selling_Price	Present_Price	Kms_Driven	Fuel_Type	Seller_Type	Transmission	Owner	Cu
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 [22]:
```
final_dataset=final_dataset.drop('Year', axis = 1)
```

In [23]:
```
final_dataset=final_dataset.drop('Current_Year', axis = 1)
```

In [24]:
```
final_dataset
```

Out[24]:

	Selling_Price	Present_Price	Kms_Driven	Fuel_Type	Seller_Type	Transmission	Owner	no_yea
0	3.35	5.59	27000	Petrol	Dealer	Manual	0	
1	4.75	9.54	43000	Diesel	Dealer	Manual	0	
2	7.25	9.85	6900	Petrol	Dealer	Manual	0	
3	2.85	4.15	5200	Petrol	Dealer	Manual	0	1
4	4.60	6.87	42450	Diesel	Dealer	Manual	0	
...	...	...	...	...	...	...	...	
296	9.50	11.60	33988	Diesel	Dealer	Manual	0	
297	4.00	5.90	60000	Petrol	Dealer	Manual	0	
298	3.35	11.00	87934	Petrol	Dealer	Manual	0	1
299	11.50	12.50	9000	Diesel	Dealer	Manual	0	
300	5.30	5.90	5464	Petrol	Dealer	Manual	0	

301 rows × 8 columns

In [25]:
```
final_dataset = pd.get_dummies(final_dataset,drop_first = True)
```

In [26]:
```
final_dataset.head()
```

Out[26]:

	Selling_Price	Present_Price	Kms_Driven	Owner	no_year	Fuel_Type_Diesel	Fuel_Type_Petrol	Se
0	3.35	5.59	27000	0	7	0	1	
1	4.75	9.54	43000	0	8	1	0	
2	7.25	9.85	6900	0	4	0	1	
3	2.85	4.15	5200	0	10	0	1	
4	4.60	6.87	42450	0	7	1	0	

In [27]:
```
final_dataset.corr()
```

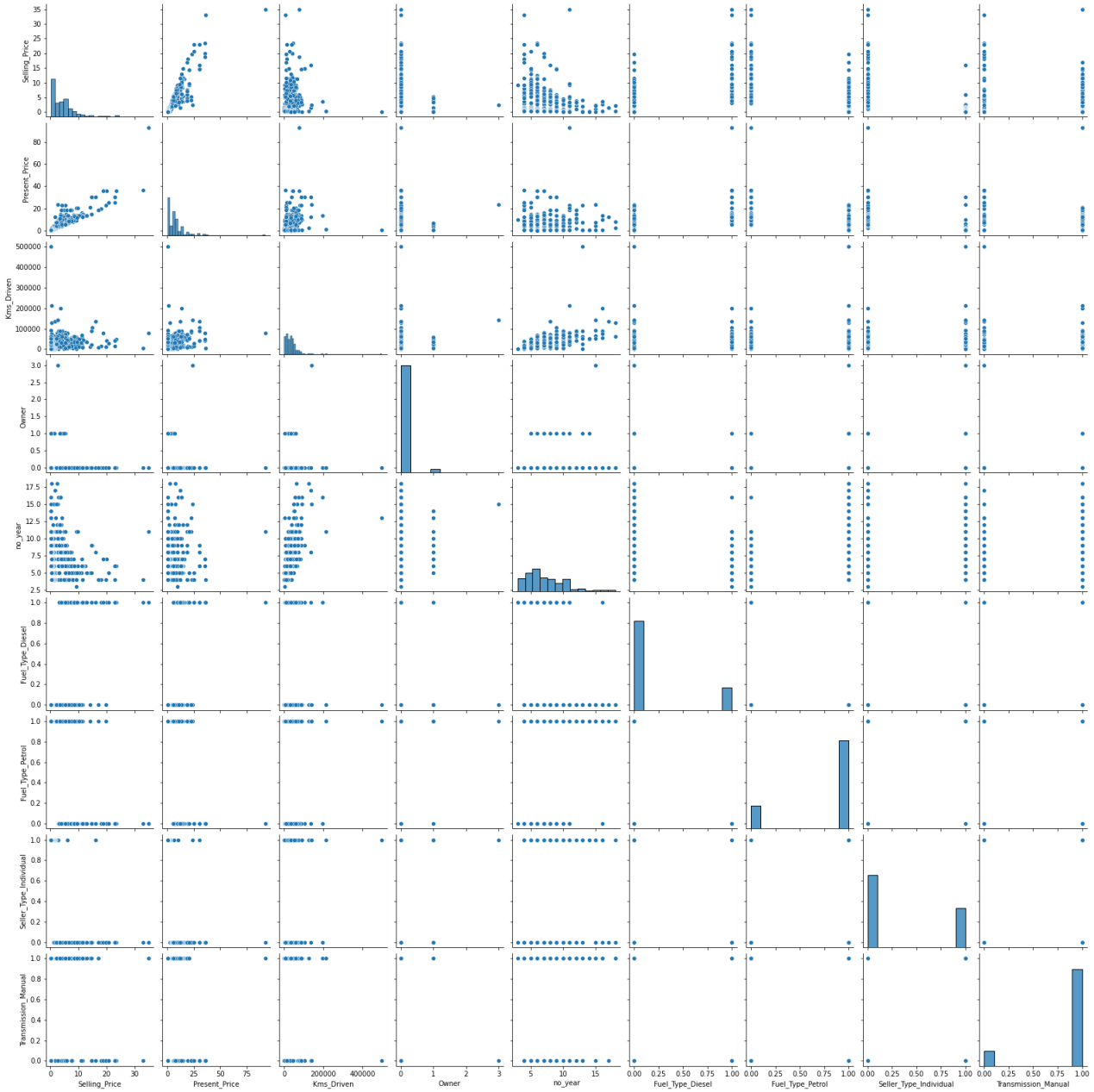
Out[27]:

	Selling_Price	Present_Price	Kms_Driven	Owner	no_year	Fuel_Type_Dies
Selling_Price	1.000000	0.878983	0.029187	-0.088344	-0.236141	0.5523
Present_Price	0.878983	1.000000	0.203647	0.008057	0.047584	0.4733
Kms_Driven	0.029187	0.203647	1.000000	0.089216	0.524342	0.1725
Owner	-0.088344	0.008057	0.089216	1.000000	0.182104	-0.0534
no_year	-0.236141	0.047584	0.524342	0.182104	1.000000	-0.0643
Fuel_Type_Diesel	0.552339	0.473306	0.172515	-0.053469	-0.064315	1.0000
Fuel_Type_Petrol	-0.540571	-0.465244	-0.172874	0.055687	0.059959	-0.9796
Seller_Type_Individual	-0.550724	-0.512030	-0.101419	0.124269	0.039896	-0.3504
Transmission_Manual	-0.367128	-0.348715	-0.162510	-0.050316	-0.000394	-0.0986

In [28]:
```
import seaborn as sns
```

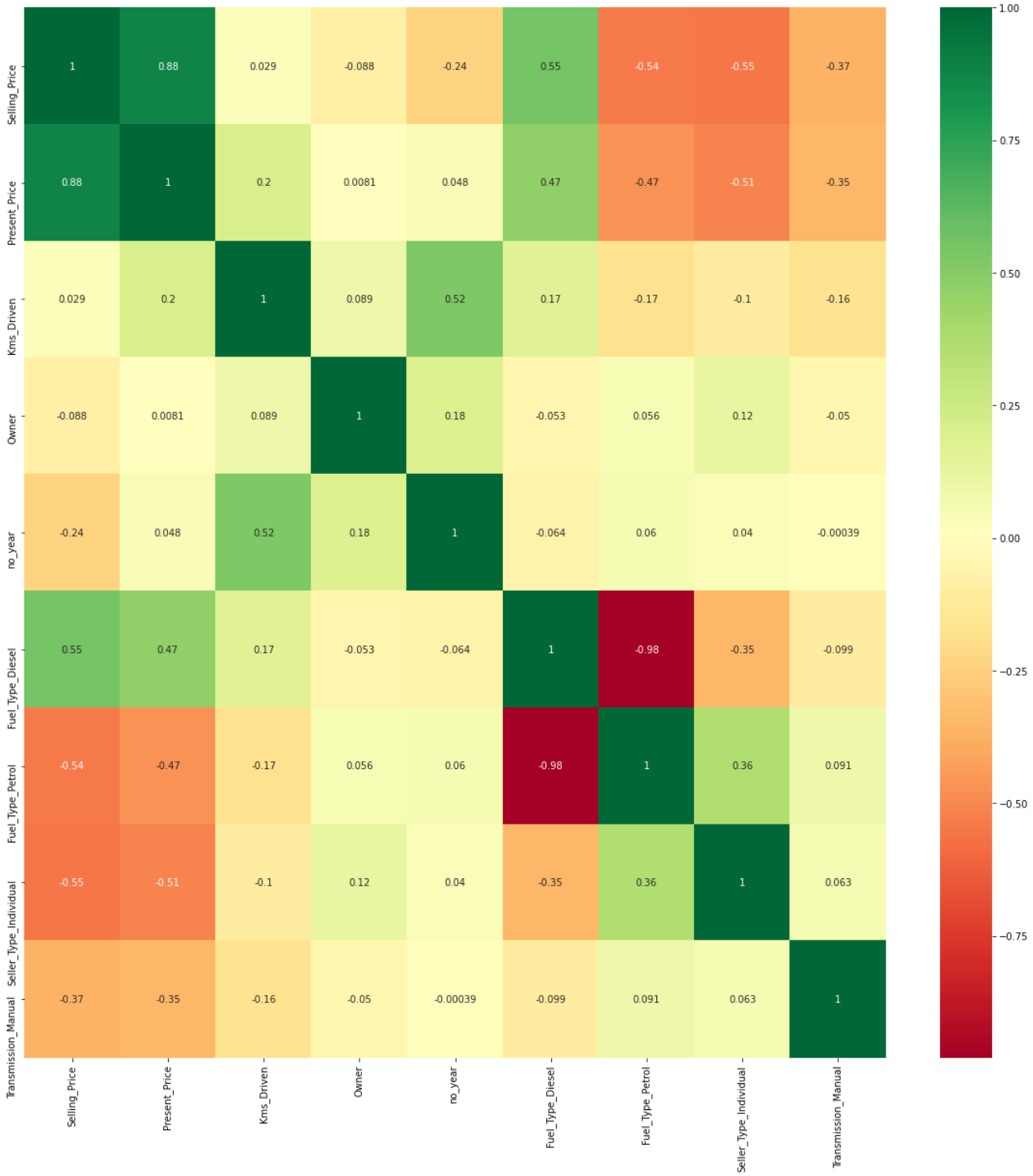
In [29]:
```
sns.pairplot(final_dataset)
```

Out[29]:
```
<seaborn.axisgrid.PairGrid at 0x7f4ccbbbf6a0>
```



In [30]:
```
import matplotlib.pyplot as plt
%matplotlib inline
```

In [31]:
```
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 [32]:
```
##independent and dependent features
X = final_dataset.iloc[:,1:]
y = final_dataset.iloc[:,0]
```

In [33]:
```
X.head()
```

Out[33]:

	Present_Price	Kms_Driven	Owner	no_year	Fuel_Type_Diesel	Fuel_Type_Petrol	Seller_Type_Indiv
0	5.59	27000	0	7	0	1	
1	9.54	43000	0	8	1	0	
2	9.85	6900	0	4	0	1	

	Present_Price	Kms_Driven	Owner	no_year	Fuel_Type_Diesel	Fuel_Type_Petrol	Seller_Type_Indiv
3	4.15	5200	0	10	0	1	
4	6.87	42450	0	7	1	0	

In [34]:
```
y.head()
```

Out[34]:
```
0    3.35
1    4.75
2    7.25
3    2.85
4    4.60
Name: Selling_Price, dtype: float64
```

In [35]:
```
### Feature Importace
from sklearn.ensemble import ExtraTreesRegressor
model=ExtraTreesRegressor()
model.fit(X,y)
```

Out[35]:
```
ExtraTreesRegressor()
```

In [36]:
```
model.fit(X,y)
```

Out[36]:
```
ExtraTreesRegressor()
```

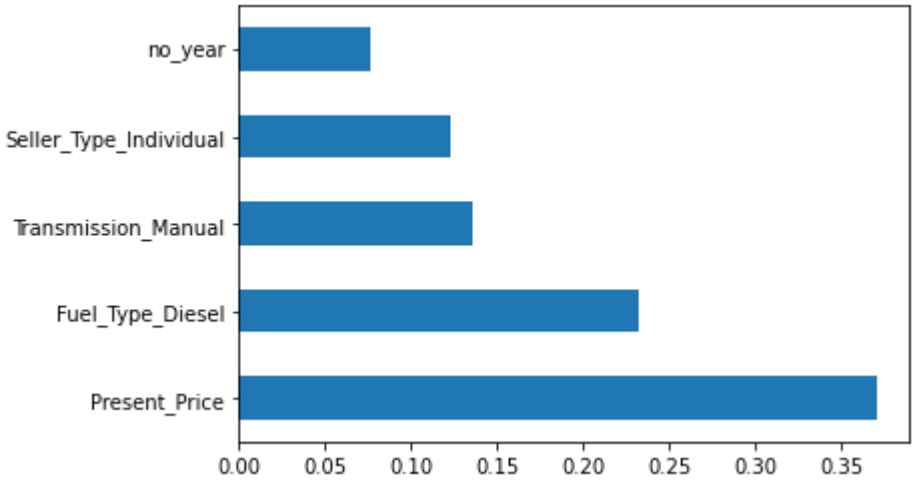
In [37]:
```
print(model.fit(X,y))
```

```
ExtraTreesRegressor()
```

In [38]:
```
print(model.feature_importances_)
```

```
[0.37122892 0.04373307 0.00037995 0.07605244 0.23233597 0.01740404
 0.12283338 0.13603223]
```

In [39]:
```
#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 [40]:
```
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 [41]:
```
X_train.shape
```

Out[41]: (240, 8)

In [42]:
```
from sklearn.ensemble import RandomForestRegressor
rf_random = RandomForestRegressor()
```

In [43]:
```
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 [44]:
```
from sklearn.model_selection import RandomizedSearchCV
```

In [45]:
```
#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 [46]:
```
# 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, 120
0], '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 [47]:

```
# Use the random grid to search for best hyperparameters
# First create the base model to tune
rf = RandomForestRegressor()
```

In [48]:

```
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations
rf_random = RandomizedSearchCV(estimator = rf, param_distributions = random_grid
```

In [49]:

```
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=   0.9s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=5, min_samples_split=
5, n_estimators=900; total time=   0.9s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=5, min_samples_split=
5, n_estimators=900; total time=   0.8s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=5, min_samples_split=
5, n_estimators=900; total time=   0.7s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=5, min_samples_split=
5, n_estimators=900; total time=   0.7s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=2, min_samples_split=
10, n_estimators=1100; 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=   1.1s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=2, min_samples_split=
10, n_estimators=1100; total time=   1.2s
[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=   1.0s
[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=   0.3s
[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=
5, n_estimators=400; 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=   0.4s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=
5, n_estimators=400; 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=   0.4s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=
5, n_estimators=400; total time=   0.4s
```

```
[CV] END max_depth=20, max_features=auto, min_samples_leaf=10, min_samples_split
=5, n_estimators=700; total time=   0.7s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=10, min_samples_split
=5, n_estimators=700; total time=   0.7s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=10, min_samples_split
=5, n_estimators=700; total time=   0.7s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=10, min_samples_split
=5, n_estimators=700; total time=   0.7s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=10, min_samples_split
=5, n_estimators=700; total time=   0.7s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=1, min_samples_split=
2, n_estimators=1000; total time=   1.1s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=1, min_samples_split=
2, n_estimators=1000; total time=   1.1s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=1, min_samples_split=
2, n_estimators=1000; total time=   1.2s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=1, min_samples_split=
2, n_estimators=1000; total time=   1.0s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=1, min_samples_split=
2, n_estimators=1000; total time=   1.1s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=10, min_samples_split=
15, n_estimators=1100; total time=   1.0s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=10, min_samples_split=
15, n_estimators=1100; total time=   1.0s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=10, min_samples_split=
15, n_estimators=1100; total time=   1.2s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=10, min_samples_split=
15, n_estimators=1100; total time=   1.0s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=10, min_samples_split=
15, n_estimators=1100; total time=   1.0s
[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=1
0, n_estimators=700; total time=   0.6s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2, min_samples_split=1
0, n_estimators=700; total time=   0.6s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2, min_samples_split=1
0, n_estimators=700; total time=   0.6s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2, min_samples_split=1
0, n_estimators=700; total time=   1.0s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2, min_samples_split=1
0, n_estimators=700; total time=   0.8s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_split=
15, n_estimators=700; total time=   0.7s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_split=
15, n_estimators=700; total time=   0.8s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_split=
15, n_estimators=700; total time=   0.8s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_split=
15, n_estimators=700; total time=   0.7s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_split=
15, n_estimators=700; total time=   0.7s
```

Out[49]:
```
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 [50]:
```
rf_random.best_params_
```

Out[50]:
```
{'n_estimators': 1000,
 'min_samples_split': 2,
 'min_samples_leaf': 1,
 'max_features': 'sqrt',
 'max_depth': 25}
```

In [51]:
```
rf_random.best_score_
```

Out[51]:
```
-4.181691264304177
```

In [52]:
```
predictions=rf_random.predict(X_test)
```

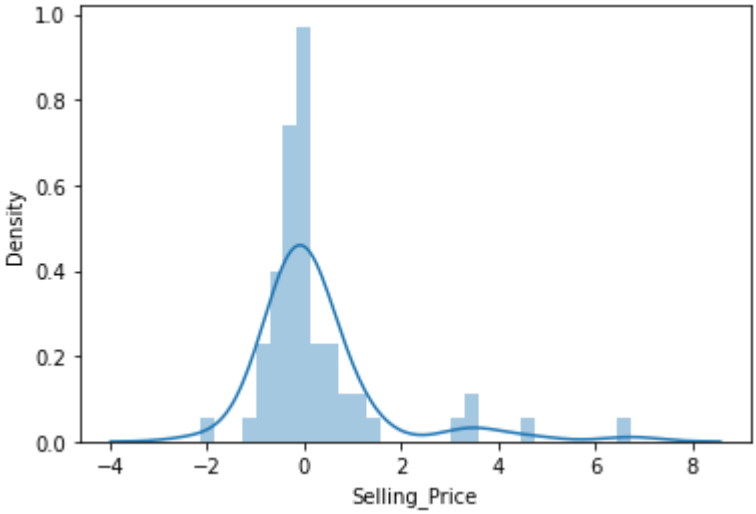
In [53]:
```
predictions
```

Out[53]:
```
array([ 7.7549  ,  1.13089 ,  7.70546 ,  2.59305 ,  4.59825 ,  4.49065 ,
        7.40992 ,  0.76417 ,  0.85875 ,  1.26927 ,  2.93788 ,  3.42128 ,
        5.53768 ,  4.98191 ,  1.13682 ,  0.44663 , 20.76062 , 11.40285 ,
        4.91221 ,  0.54978 ,  0.3927  ,  5.799275,  0.4946  ,  1.02244 ,
        0.71208 ,  1.05705 ,  2.98438 ,  1.16442 ,  2.34622 ,  9.71545 ,
        9.32033 ,  4.94053 ,  0.86095 ,  5.33895 ,  4.86524 ,  0.383   ,
        7.44975 ,  4.0579  ,  8.29463 ,  7.50505 ,  5.5386  ,  0.82441 ,
        2.55158 , 10.26032 ,  4.9734  ,  0.94564 ,  8.51812 ,  0.66368 ,
        7.96495 , 20.64656 , 10.07128 ,  5.70733 ,  6.61997 ,  0.69771 ,
        0.88904 ,  1.03137 ,  5.34451 ,  2.86074 ,  0.57092 ,  2.75715 ,
        8.94981 ])
```

In [54]:
```
sns.distplot(y_test-predictions)
```

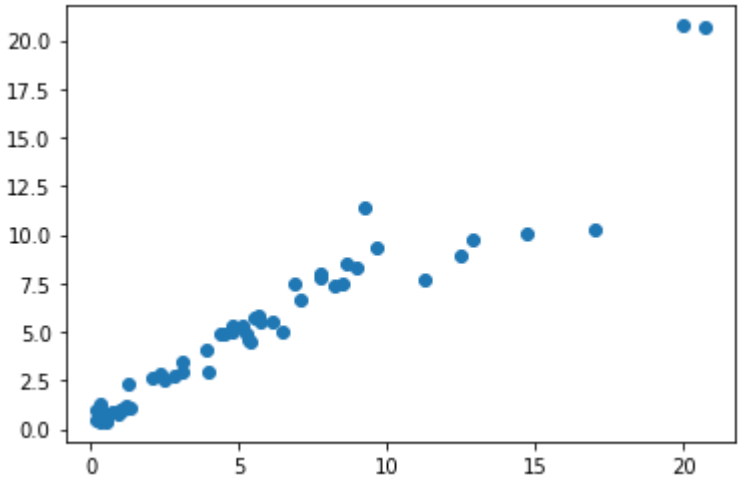
```
/home/swarajsp/.local/lib/python3.9/site-packages/seaborn/distributions.py:2557:
FutureWarning: `distplot` is a deprecated function and will be removed in a futu
re version. Please adapt your code to use either `displot` (a figure-level funct
ion with similar flexibility) or `histplot` (an axes-level function for histogra
ms).
  warnings.warn(msg, FutureWarning)
```

Out[54]:
```
<AxesSubplot:xlabel='Selling_Price', ylabel='Density'>
```



In [55]:
```
plt.scatter(y_test,predictions)
```

Out[55]: <matplotlib.collections.PathCollection at 0x7f4cbc143d90>



In [56]:
```
import pickle
```

In [57]:
```
# open a file, where you are going to store the data
# file = open('random_forest_regression_model.pkl', 'wb')

# dump information to that file
# pickle.dump(rf_random, file)
```

In [ ]: