In [ ]:

In [ ]:

import library

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
```

Data Pre Processing

In [2]:
```
#load data
df=pd.read_csv(r"C:\Users\Mrunali Dupare\Downloads\archive (7)\car data.csv")
```

In [3]:
```
#Printing first five rows
df.head()
```

Out[3]:

	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 [4]:
```
df.shape  #number of rows and columns present in a dataset
```

Out[4]:
```
(301, 9)
```

In [5]:
```
df.columns  #printing name of all the columns
```

Out[5]:
```
Index(['Car_Name', 'Year', 'Selling_Price', 'Present_Price', 'Kms_Driven',
       'Fuel_Type', 'Seller_Type', 'Transmission', 'Owner'],
      dtype='object')
```

In [6]:
```
#dropping the car_name column
#name of companies won't affect car's price ,price depends upon how many year it's been used ,fuel type etc,.
#so i will drop the column car_name from original dataframe.
df.drop("Car_Name",axis=1,inplace=True)
```

In [7]:
```
df.isnull().sum()  #is there any null value present
```

Out[7]:
```
Year             0
Selling_Price    0
Present_Price    0
Kms_Driven       0
Fuel_Type        0
Seller_Type      0
Transmission     0
Owner            0
dtype: int64
```

In [8]:
```
#chcek the number of rows,column and data types
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 301 entries, 0 to 300
Data columns (total 8 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Year           301 non-null    int64
 1   Selling_Price  301 non-null    float64
 2   Present_Price  301 non-null    float64
 3   Kms_Driven     301 non-null    int64
 4   Fuel_Type      301 non-null    object
 5   Seller_Type    301 non-null    object
 6   Transmission   301 non-null    object
 7   Owner          301 non-null    int64
dtypes: float64(2), int64(3), object(3)
memory usage: 18.9+ KB
```

In [9]:
```
#as we see there are some categorial feature are present so we have to store them in new column
```

In [10]:
```
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 [ ]:
```
```

In [11]:
```
df.head()
```

Out[11]:

	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 [12]:
```
#Year represent the year in which car have been purchased
#so how we can estimate the number of year car has been used ?
```

In [13]:
```
df["Current_Year"]=2020
```

In [14]:
```
df["No_of_years"]=df["Current_Year"]-df["Year"]
df=df.drop(["Current_Year","Year"],axis=1)
df.head()
```

Out[14]:

	Selling_Price	Present_Price	Kms_Driven	Fuel_Type	Seller_Type	Transmission	Owner	No_of_years
0	3.35	5.59	27000	Petrol	Dealer	Manual	0	6
1	4.75	9.54	43000	Diesel	Dealer	Manual	0	7
2	7.25	9.85	6900	Petrol	Dealer	Manual	0	3
3	2.85	4.15	5200	Petrol	Dealer	Manual	0	9
4	4.60	6.87	42450	Diesel	Dealer	Manual	0	6

In [15]:
```
df=pd.get_dummies(df,drop_first=True)
df.head()
```

Out[15]:

	Selling_Price	Present_Price	Kms_Driven	Owner	No_of_years	Fuel_Type_Diesel	Fuel_Type_Petrol	Seller_Type_Individual	Transmission_Mar
0	3.35	5.59	27000	0	6	0	1	0	

1	4.75	9.54	43000	0	7	1	0	0
2	7.25	9.85	6900	0	3	0	1	0
3	2.85	4.15	5200	0	9	0	1	0
4	4.60	6.87	42450	0	6	1	0	0

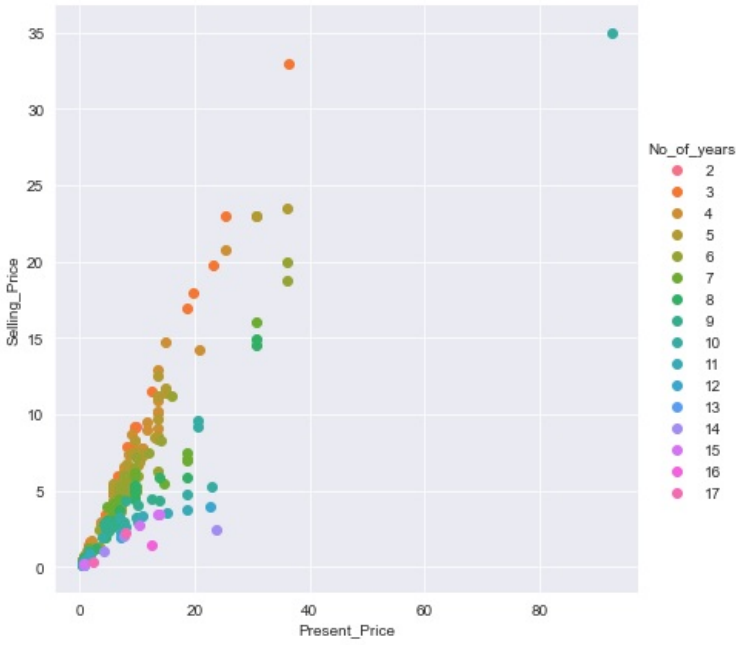
In [16]:
```
df.corr()
```

Out[16]:

	Selling_Price	Present_Price	Kms_Driven	Owner	No_of_years	Fuel_Type_Diesel	Fuel_Type_Petrol	Seller_Type_Indivi
Selling_Price	1.000000	0.878983	0.029187	-0.088344	-0.236141	0.552339	-0.540571	-0.550
Present_Price	0.878983	1.000000	0.203647	0.008057	0.047584	0.473306	-0.465244	-0.512
Kms_Driven	0.029187	0.203647	1.000000	0.089216	0.524342	0.172515	-0.172874	-0.101
Owner	-0.088344	0.008057	0.089216	1.000000	0.182104	-0.053469	0.055687	0.124
No_of_years	-0.236141	0.047584	0.524342	0.182104	1.000000	-0.064315	0.059959	0.039
Fuel_Type_Diesel	0.552339	0.473306	0.172515	-0.053469	-0.064315	1.000000	-0.979648	-0.350
Fuel_Type_Petrol	-0.540571	-0.465244	-0.172874	0.055687	0.059959	-0.979648	1.000000	0.358
Seller_Type_Individual	-0.550724	-0.512030	-0.101419	0.124269	0.039896	-0.350467	0.358321	1.000
Transmission_Manual	-0.367128	-0.348715	-0.162510	-0.050316	-0.000394	-0.098643	0.091013	0.063

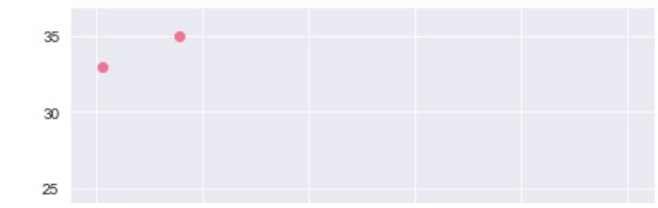
Data Visualization

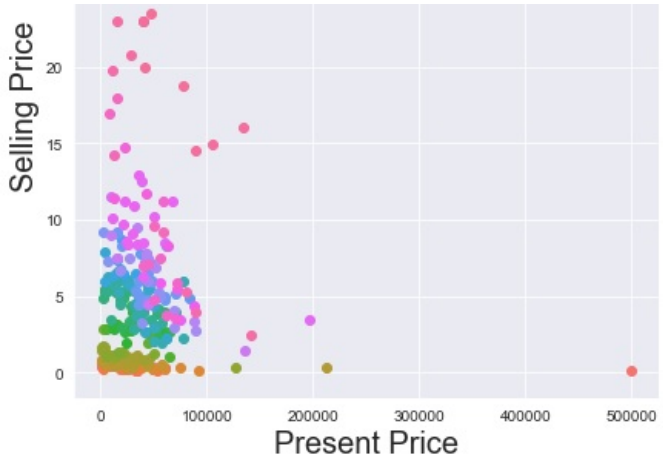
In [17]:
```
sns.set_style("darkgrid")
sns.FacetGrid(df,hue="No_of_years",height=6).map(plt.scatter,"Present_Price","Selling_Price").add_legend()
plt.show()
```



In [18]:
```
#More number of Years you will use your car lesser the amount you will get.
```

In [19]:
```
sns.set_style("darkgrid")
sns.FacetGrid(df,hue="Present_Price",height=6).map(plt.scatter,"Kms_Driven","Selling_Price")
plt.xlabel("Present Price",fontsize=20)
plt.ylabel("Selling Price",fontsize=20)
plt.show()
```





In [ ]:

In [20]:
```
#lesser the car would be driven higher will be
#the cost as we see the graph at max distance i e:- 500000 kilometres the car’s cost is near to Zero or we can sa
```

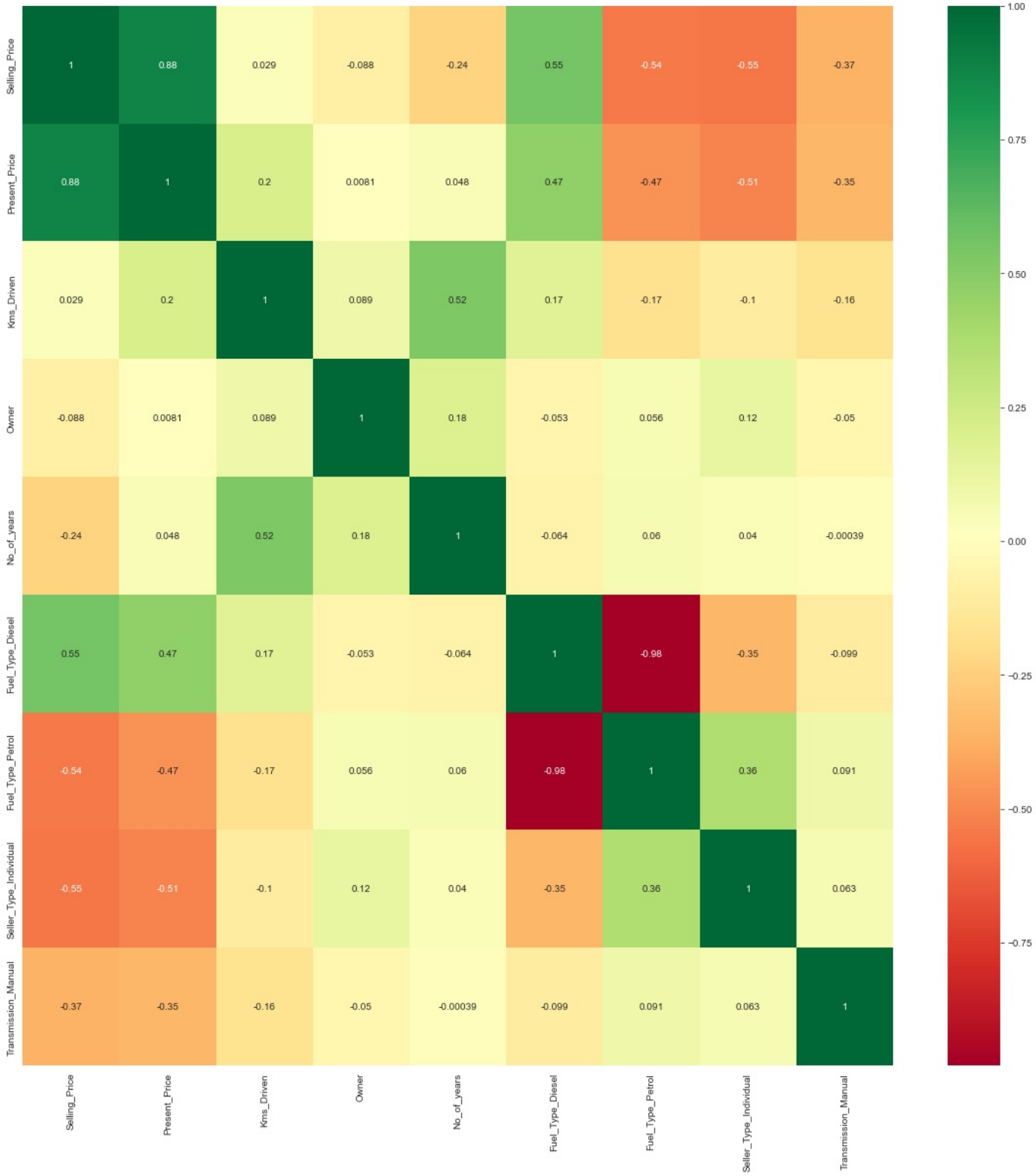
In [21]:
```
sns.pairplot(df)
```

Out[21]:
```
<seaborn.axisgrid.PairGrid at 0x1bb377d5f40>
```

Selling_Price Present_Price Kms_Driven Owner No_of_years Fuel_Type_Diesel Fuel_Type_Petrol Seller_Type_Individual Transmission_Manual

In [22]:
```
#As we see there are very less overlapping in dataset is seen so we cannot use knn ,linear regression,svm and bec
```

In [23]:
```
corrmat = df.corr()
top_corr_features = corrmat.index
plt.figure(figsize=(20,20))
#plot heat map
g=sns.heatmap(df[top_corr_features].corr(),annot=True,cmap="RdYlGn")
```



In [24]:
```
#Uni variate analysis :- when analysis involve single variable most predominantly it is used to find the pattern
```

In [25]:
```
X=df.iloc[:,1:]
y=df.iloc[:,0]
```

In [26]:
```
X['Owner'].unique()
```

Out[26]:
```
array([0, 1, 3], dtype=int64)
```

In [27]:
```
X.head()
```

Out[27]:

	Present_Price	Kms_Driven	Owner	No_of_years	Fuel_Type_Diesel	Fuel_Type_Petrol	Seller_Type_Individual	Transmission_Manual
0	5.59	27000	0	6	0	1	0	1
1	9.54	43000	0	7	1	0	0	1
2	9.85	6900	0	3	0	1	0	1
3	4.15	5200	0	9	0	1	0	1
4	6.87	42450	0	6	1	0	0	1

In [28]:
```
y.head()
```

Out[28]:
```
0    3.35
1    4.75
2    7.25
3    2.85
4    4.60
Name: Selling_Price, dtype: float64
```

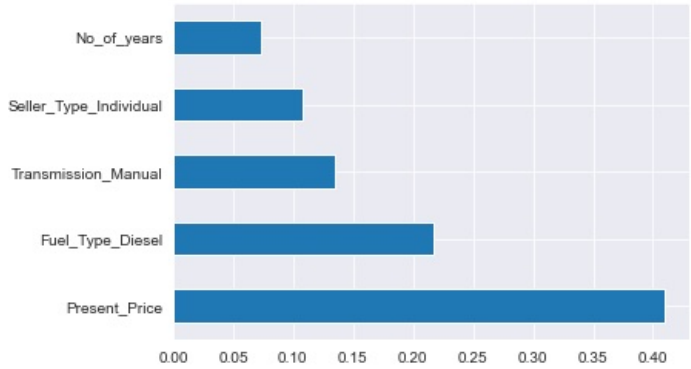
In [29]:
```
from sklearn.ensemble import ExtraTreesRegressor
import matplotlib.pyplot as plt
model = ExtraTreesRegressor()
model.fit(X,y)
```

Out[29]:
```
ExtraTreesRegressor()
```

In [30]:
```
print(model.feature_importances_)
```

```
[0.40963864 0.04268578 0.00042451 0.07336132 0.21664102 0.01420529
 0.10817384 0.1348696 ]
```

In [31]:
```
#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 [32]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)
```

In [33]:
```
from sklearn.ensemble import RandomForestRegressor
```

In [34]:
```
regressor=RandomForestRegressor()
```

In [35]:
```
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 [36]:
```
from sklearn.model_selection import RandomizedSearchCV
```

In [37]:
```
 #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 [38]:
```
# 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 [39]:
```
# Use the random grid to search for best hyperparameters
# First create the base model to tune
rf = RandomForestRegressor()
```

In [40]:
```
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations
rf_random = RandomizedSearchCV(estimator = rf, param_distributions = random_grid,scoring='neg_mean_squared_error'
```

In [41]:
```
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=
1.0s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=5, min_samples_split=5, n_estimators=900; total time=
1.0s
[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=
0.9s
[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
=   1.1s
[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
=   1.1s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=100, n_estimators=300; total time
=   0.2s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=100, n_estimators=300; total time
=   0.2s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=100, n_estimators=300; total time
=   0.2s
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=100, n_estimators=300; total time
=   0.2s
```

```
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=100, n_estimators=300; total time
=   0.2s
[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=
1.1s
[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=
1.1s
[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
=   1.1s
[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
=   1.0s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=1, min_samples_split=15, n_estimators=300; total time=
0.2s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=1, min_samples_split=15, n_estimators=300; total time=
0.2s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=1, min_samples_split=15, n_estimators=300; total time=
0.2s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=1, min_samples_split=15, n_estimators=300; total time=
0.2s
[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=
0.7s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=700; total time=
0.6s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=700; total time=
0.6s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=700; total time=
0.6s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, 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
[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
[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[41]:

```
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 [42]:
```
rf_random.best_params_
```

Out[42]:
```
{'n_estimators': 1000,
 'min_samples_split': 2,
 'min_samples_leaf': 1,
 'max_features': 'sqrt',
 'max_depth': 25}
```

In [43]:
```
rf_random.best_score_
```

Out[43]:
```
-3.965366984331113
```

In [44]:
```
predictions=rf_random.predict(X_test)
```

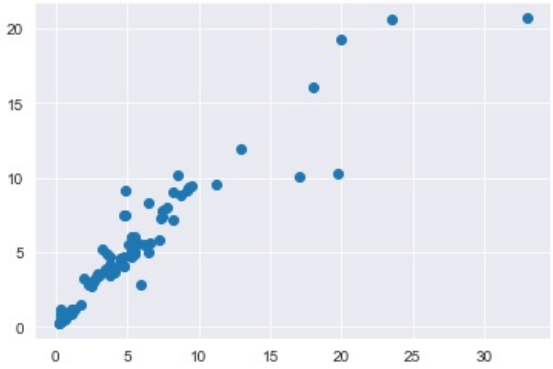
In [45]:
```
sns.distplot(y_test-predictions)
```

```
C:\Users\Mrunali Dupare\Anaconda\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[45]:
```
<AxesSubplot:xlabel='Selling_Price', ylabel='Density'>
```

In [46]:
```
plt.scatter(y_test,predictions)
```

Out[46]:
```
<matplotlib.collections.PathCollection at 0x1bb3be93700>
```



In [47]:
```
from sklearn import metrics
```

In [48]:
```
print('MAE:', metrics.mean_absolute_error(y_test, predictions))
print('MSE:', metrics.mean_squared_error(y_test, predictions))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, predictions)))
```

```
MAE: 0.8952787637362638
MSE: 4.133997895857595
RMSE: 2.0332235233386404
```

In [49]:
```
import pickle
# open a file, where you ant to store the data
file = open('random_forest_regression_model.pkl', 'wb')

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

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: