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

In [2]:
```
df=pd.read_csv('C:/Users/Ritik Sharma/OneDrive/Desktop/Extra/miniproject1/car dat
```

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

Out[3]:

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

301 rows × 9 columns

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

Out[4]:

	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

In [5]:
```
df.shape
```

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

In [6]:

```
print(df['Seller_Type'].unique())
print(df['Owner'].unique())
print(df['Transmission'].unique())
print(df['Fuel_Type'].unique())
#print(df['Car_Name'].unique())
print(df['Year'].unique())
```

```
['Dealer' 'Individual']
[0 1 3]
['Manual' 'Automatic']
['Petrol' 'Diesel' 'CNG']
[2014 2013 2017 2011 2018 2015 2016 2009 2010 2012 2003 2008 2006 2005
 2004 2007]
```

In [7]:

```
df.head(10)
```

Out[7]:

	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
5	vitara brezza	2018	9.25	9.83	2071	Diesel	Dealer	Manu
6	ciaz	2015	6.75	8.12	18796	Petrol	Dealer	Manu
7	s cross	2015	6.50	8.61	33429	Diesel	Dealer	Manu
8	ciaz	2016	8.75	8.89	20273	Diesel	Dealer	Manu
9	ciaz	2015	7.45	8.92	42367	Diesel	Dealer	Manu

Checking null or missing values

In [8]:

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

Out[8]:

```
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 [9]:
```
df.describe()
```

Out[9]:

	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 [9]:
```
df.columns
```

Out[9]:
```
Index(['Car_Name', 'Year', 'Selling_Price', 'Present_Price', 'Kms_Driven',
       'Fuel_Type', 'Seller_Type', 'Transmission', 'Owner'],
      dtype='object')
```

In [10]:
```
final_dataset=df[['Year', 'Selling_Price', 'Present_Price', 'Kms_Driven',
       'Fuel_Type', 'Seller_Type', 'Transmission', 'Owner']]
```

In [11]:
```
final_dataset.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]:
```
final_dataset['Current_Year']=2020
```

In [13]:
```
final_dataset.head()
```

Out[13]:

	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 [14]:
```
final_dataset['No_of_years']= final_dataset['Current_Year']-final_dataset['Year']
```

In [15]:
```
final_dataset.head()
```

Out[15]:

	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 [16]:
```
final_dataset.drop(['Year'],axis=1)
```

Out[16]:

	Selling_Price	Present_Price	Kms_Driven	Fuel_Type	Seller_Type	Transmission	Owner	Curr
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	
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	
299	11.50	12.50	9000	Diesel	Dealer	Manual	0	
300	5.30	5.90	5464	Petrol	Dealer	Manual	0	

301 rows × 9 columns

In [17]:
```
final_dataset.drop(['Year'],axis=1,inplace=True)
```

In [18]:
```
final_dataset.drop(['Current_Year'],axis=1,inplace=True)
```

In [19]:
```
final_dataset.head()
```

Out[19]:

	Selling_Price	Present_Price	Kms_Driven	Fuel_Type	Seller_Type	Transmission	Owner	No_of_
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	
4	4.60	6.87	42450	Diesel	Dealer	Manual	0	

In [20]:
```
final_dataset=pd.get_dummies(final_dataset,drop_first=True)
```

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

Out[21]:

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

In [22]:
```
print(final_dataset['Owner'].unique())
```

```
[0 1 3]
```

In [23]:
```
print(final_dataset['Fuel_Type_Diesel'].unique())
```

```
[0 1]
```

In [24]:
```
final_dataset.corr()
```

Out[24]:

	Selling_Price	Present_Price	Kms_Driven	Owner	No_of_years	Fuel_Type
Selling_Price	1.000000	0.878983	0.029187	-0.088344	-0.236141	0
Present_Price	0.878983	1.000000	0.203647	0.008057	0.047584	0
Kms_Driven	0.029187	0.203647	1.000000	0.089216	0.524342	0
Owner	-0.088344	0.008057	0.089216	1.000000	0.182104	-0
No_of_years	-0.236141	0.047584	0.524342	0.182104	1.000000	-0
Fuel_Type_Diesel	0.552339	0.473306	0.172515	-0.053469	-0.064315	1
Fuel_Type_Petrol	-0.540571	-0.465244	-0.172874	0.055687	0.059959	-0
Seller_Type_Individual	-0.550724	-0.512030	-0.101419	0.124269	0.039896	-0
Transmission_Manual	-0.367128	-0.348715	-0.162510	-0.050316	-0.000394	-0

In [25]:
```
import seaborn as sns
```

In [26]:
```
sns.pairplot(final_dataset)
```

Out[26]:
```
<seaborn.axisgrid.PairGrid at 0x22724cbf8c8>
```

In [27]:
```
import matplotlib.pyplot as plt
%matplotlib inline
```

In [29]:
```
corrmat.index
```

Out[29]:
```
Index(['Selling_Price', 'Present_Price', 'Kms_Driven', 'Owner', 'No_of_years',
       'Fuel_Type_Diesel', 'Fuel_Type_Petrol', 'Seller_Type_Individual',
       'Transmission_Manual'],
      dtype='object')
```

In [28]:
```
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 [30]:
```
final_dataset.head()
```

Out[30]:

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

In [31]:
```
# Dependent and Independent features
x=final_dataset.iloc[:,1:]
y=final_dataset.iloc[:,0]
```

In [32]:
```
x.head()
```

Out[32]:

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

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

Out[33]:
```
0    3.35
1    4.75
2    7.25
3    2.85
4    4.60
Name: Selling_Price, dtype: float64
```

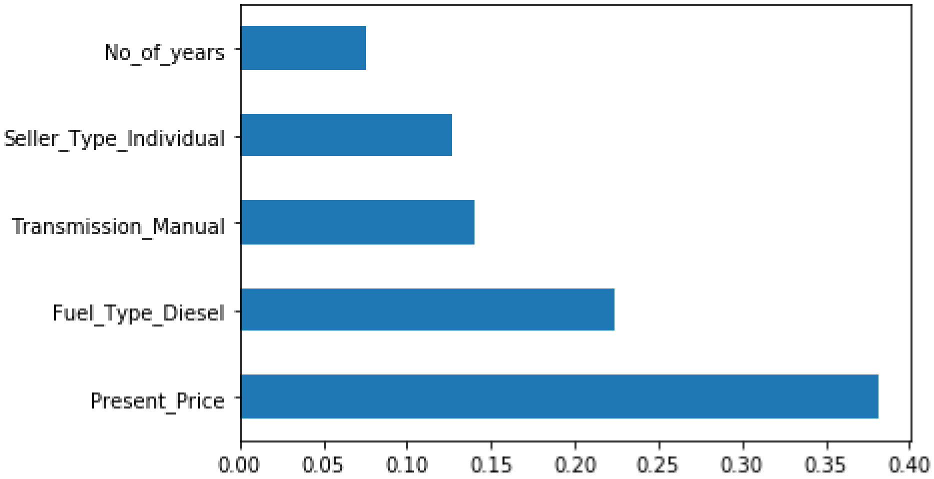
In [34]:
```
# feature importance
from sklearn.ensemble import ExtraTreesRegressor
model=ExtraTreesRegressor()
model.fit(x,y)
```

Out[34]:
```
ExtraTreesRegressor(bootstrap=False, ccp_alpha=0.0, criterion='mse',
                    max_depth=None, max_features='auto', max_leaf_nodes=None,
                    max_samples=None, min_impurity_decrease=0.0,
                    min_impurity_split=None, min_samples_leaf=1,
                    min_samples_split=2, min_weight_fraction_leaf=0.0,
                    n_estimators=100, n_jobs=None, oob_score=False,
                    random_state=None, verbose=0, warm_start=False)
```

In [35]:
```
print(model.feature_importances_)
```

```
[0.38116433 0.03918062 0.00080062 0.07605712 0.22345561 0.0122779
 0.12737728 0.13968652]
```

In [36]:
```
# plot graph of feature importances for better visualisation
feat_importances= pd.Series(model.feature_importances_ ,index=x.columns)
feat_importances.nlargest(5).plot(kind='barh') #or can be 'bar'
plt.show()
```



In [37]:
```
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 [38]:
```
x_train.head()
```

Out[38]:

	Present_Price	Kms_Driven	Owner	No_of_years	Fuel_Type_Diesel	Fuel_Type_Petrol	Seller_T
86	92.60	78000	0	10	1	0	
158	0.54	8600	0	3	0	1	
288	13.60	34000	0	5	0	1	
269	10.00	18828	0	5	0	1	
246	6.79	35000	0	8	0	1	

In [39]:
```
x_train.shape
```

Out[39]:
```
(240, 8)
```

In [40]:
```
from sklearn.ensemble import RandomForestRegressor
rf_random=RandomForestRegressor()
```

In [41]:
```
## Hyperparameter Tuning
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 [42]:
```
from sklearn.model_selection import RandomizedSearchCV
```

In [43]:
```
#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 [44]:
```
# 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], 'mi
n_samples_split': [2, 5, 10, 15, 100], 'min_samples_leaf': [1, 2, 5, 10]}
```

In [45]:
```
# Use the random grid to search for best hyperparameters
# First create the base model to tune
rf = RandomForestRegressor()
```

In [46]:
```
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations
rf_random = RandomizedSearchCV(estimator = rf, param_distributions = random_grid,
```

In [47]:
```
rf_random.fit(x_train,y_train)
```

```
0.0,
                                                    n_estimators=100,
                                                    n_jobs=None, oob_score=Fal
s...
                   iid='deprecated', n_iter=10, 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]},
                   pre_dispatch='2*n_jobs', random_state=42, refit=True,
                   return_train_score=False, scoring='neg_mean_squared_erro
r',
                   verbose=2)
```

In [48]:
```
predictions=rf_random.predict(x_test)
```

```
In [49]: predictions
```

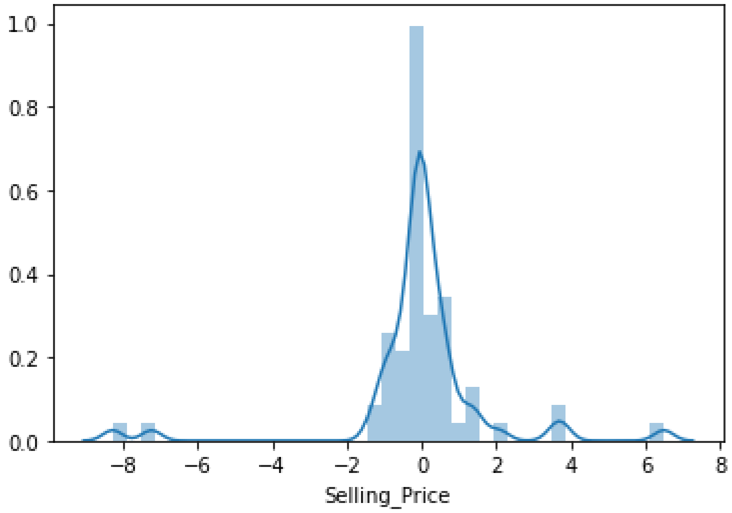
```
Out[49]: array([ 0.25334823,  4.29629821,  5.30576821,  1.34476422,  0.69247289,
                19.22688526,  2.89488607,  5.65776773,  4.51678034,  0.21199698,
                 4.5292329 ,  6.23465731, 12.49095497,  0.46867931,  6.06574334,
                 3.01889338, 10.50722163,  0.25234559,  4.40223703,  1.32005599,
                 2.7844756 ,  4.38257045,  0.53021438,  5.32334255,  0.71675867,
                 7.28661508, 20.93280564,  0.85100108,  4.03971146,  0.42158992,
                10.01074427, 13.81184551,  2.98971858, 12.28712786,  6.4782795 ,
                 4.19941011,  6.22137853,  7.46573664,  7.43064684,  0.28942827,
                 1.14834537,  6.64447942,  7.30520049,  7.66045385, 11.12648063,
                 3.04431124,  2.9382025 ,  9.63460567,  5.05565645,  5.82070724,
                 8.00248927,  4.76673435,  2.92792113,  4.55834366,  3.37968707,
                 9.82336909,  0.61775414,  0.50449838,  4.48025007,  0.29748661,
                 2.98052931])
```

```
In [50]: y=rf_random.predict(y_test)
         y
```

...

```
In [51]: sns.distplot(y_test-predictions)
```

```
Out[51]: <matplotlib.axes._subplots.AxesSubplot at 0x2272b46b788>
```

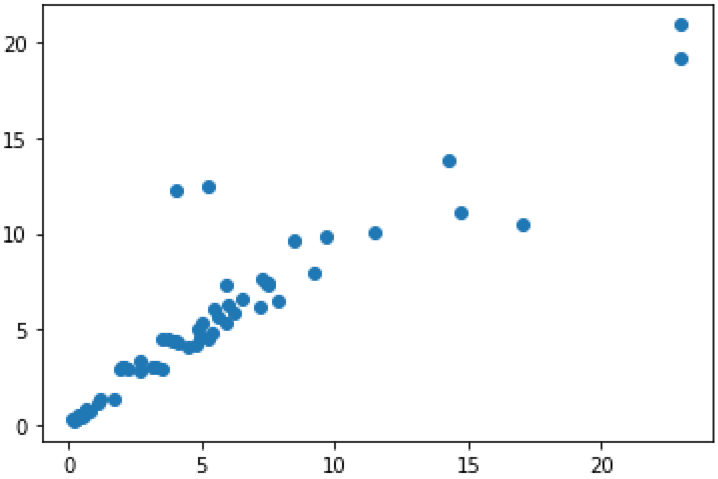


In [52]:

```
plt.scatter(y_test,predictions)
```

Out[52]:

```
<matplotlib.collections.PathCollection at 0x2272b779d88>
```



In [53]:

```
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 [ ]: