```
import pandas as pd
```

```
df = pd.read_csv("/content/car data.csv")
```

```
df.head()
```

	Car_Name	Year	Selling_Price	Present_Price	Kms_Driven	Fuel_Type	Seller_Typ
0	ritz	2014	3.35	5.59	27000	Petrol	Deale
1	sx4	2013	4.75	9.54	43000	Diesel	Deale
2	ciaz	2017	7.25	9.85	6900	Petrol	Deale
3	wagon r	2011	2.85	4.15	5200	Petrol	Deale
4	swift	2014	4.60	6.87	42450	Diesel	Deale

```
df.shape
```

```
(301, 9)
```

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

```
['Dealer' 'Individual']
['Manual' 'Automatic']
[0 1 3]
```

```
# checking the missing or null values

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

```
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
```

```
df.describe()
```

	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

```
df.columns
```

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

```
#df.drop(['Car_Name'],axis=1, inplace=True)
final_df = df[['Year', 'Selling_Price', 'Present_Price', 'Kms_Driven','Fuel_Type', 'Seller_Type', 'Transmission', 'Owner']]
```

```
final_df.head()
```

	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

```
final_df['current_year']=2023
```

```
final_df.head()
```

	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	2023
1	2013	4.75	9.54	43000	Diesel	Dealer	Manual	0	2023
2	2017	7.25	9.85	6900	Petrol	Dealer	Manual	0	2023
3	2011	2.85	4.15	5200	Petrol	Dealer	Manual	0	2023
4	2014	4.60	6.87	42450	Diesel	Dealer	Manual	0	2023

```
final_df['no_years']= final_df['current_year']-final_df['Year']
```

```
final_df.head()
```

	Year	Selling_Price	Present_Price	Kms_Driven	Fuel_Type	Seller_Type	Transmission	Owner	current_year	no_years
0	2014	3.35	5.59	27000	Petrol	Dealer	Manual	0	2023	9
1	2013	4.75	9.54	43000	Diesel	Dealer	Manual	0	2023	10
2	2017	7.25	9.85	6900	Petrol	Dealer	Manual	0	2023	6
3	2011	2.85	4.15	5200	Petrol	Dealer	Manual	0	2023	12
4	2014	4.60	6.87	42450	Diesel	Dealer	Manual	0	2023	9

```
final_df.drop(['Year'],axis=1,inplace=True)
```

```
final_df.drop(['current_year'],axis=1,inplace=True)
```

```
final_df.head()
```

	Selling_Price	Present_Price	Kms_Driven	Fuel_Type	Seller_Type	Transmission	Owner	no_years
0	3.35	5.59	27000	Petrol	Dealer	Manual	0	9
1	4.75	9.54	43000	Diesel	Dealer	Manual	0	10
2	7.25	9.85	6900	Petrol	Dealer	Manual	0	6
3	2.85	4.15	5200	Petrol	Dealer	Manual	0	12
4	4.60	6.87	42450	Diesel	Dealer	Manual	0	9

```
final_df = pd.get_dummies(final_df,drop_first=True)
```

```
final_df.head()
```

	Selling_Price	Present_Price	Kms_Driven	Owner	no_years	Fuel_Type_Diesel	Fuel_Type_Petrol	Seller_Type_Individual	Transmission_M
0	3.35	5.59	27000	0	9	0	1	0	
1	4.75	9.54	43000	0	10	1	0	0	
2	7.25	9.85	6900	0	6	0	1	0	
3	2.85	4.15	5200	0	12	0	1	0	

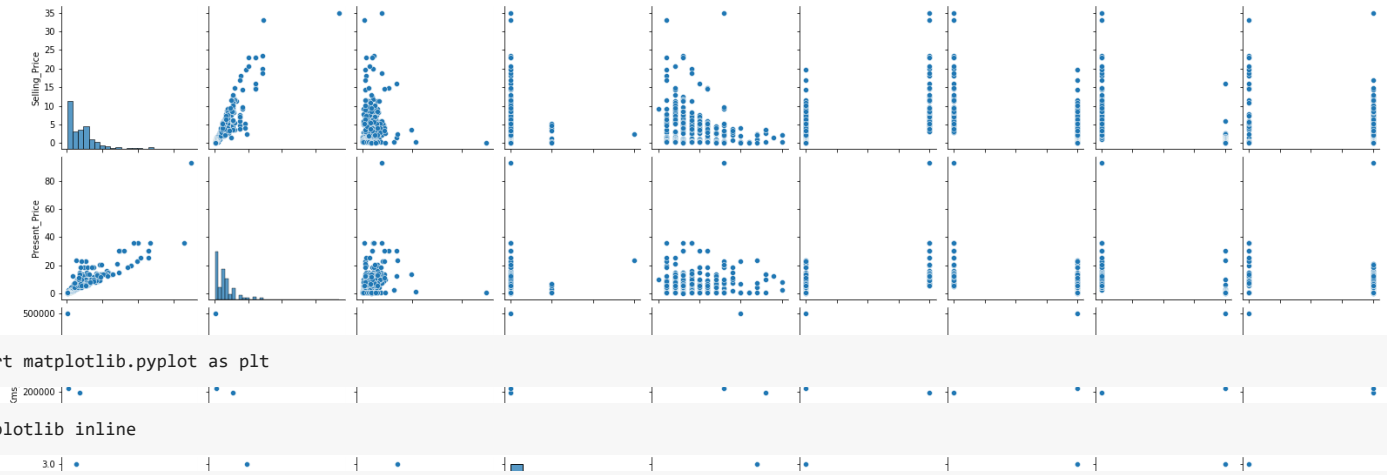
```
final_df.corr()
```

	Selling_Price	Present_Price	Kms_Driven	Owner	no_years	Fuel_Type_Diesel	Fuel_Type_Petrol	Seller_Type_Indiv
Selling_Price	1.000000	0.878983	0.029187	-0.088344	-0.236141	0.552339	-0.540571	-0.5
Present_Price	0.878983	1.000000	0.203647	0.008057	0.047584	0.473306	-0.465244	-0.5
Kms_Driven	0.029187	0.203647	1.000000	0.089216	0.524342	0.172515	-0.172874	-0.1
Owner	-0.088344	0.008057	0.089216	1.000000	0.182104	-0.053469	0.055687	0.1
no_years	-0.236141	0.047584	0.524342	0.182104	1.000000	-0.064315	0.059959	0.0
Fuel_Type_Diesel	0.552339	0.473306	0.172515	-0.053469	-0.064315	1.000000	-0.979648	-0.3
Fuel_Type_Petrol	-0.540571	-0.465244	-0.172874	0.055687	0.059959	-0.979648	1.000000	0.3
Seller_Type_Individual	-0.550724	-0.512030	-0.101419	0.124269	0.039896	-0.350467	0.358321	1.0
Transmission_Manual	-0.367128	-0.348715	-0.162510	-0.050316	-0.000394	-0.098643	0.091013	0.0

```
import seaborn as sns
```

```
sns.pairplot(final_df)
```

```
<seaborn.axisgrid.PairGrid at 0x7fb0a92eee20>
```



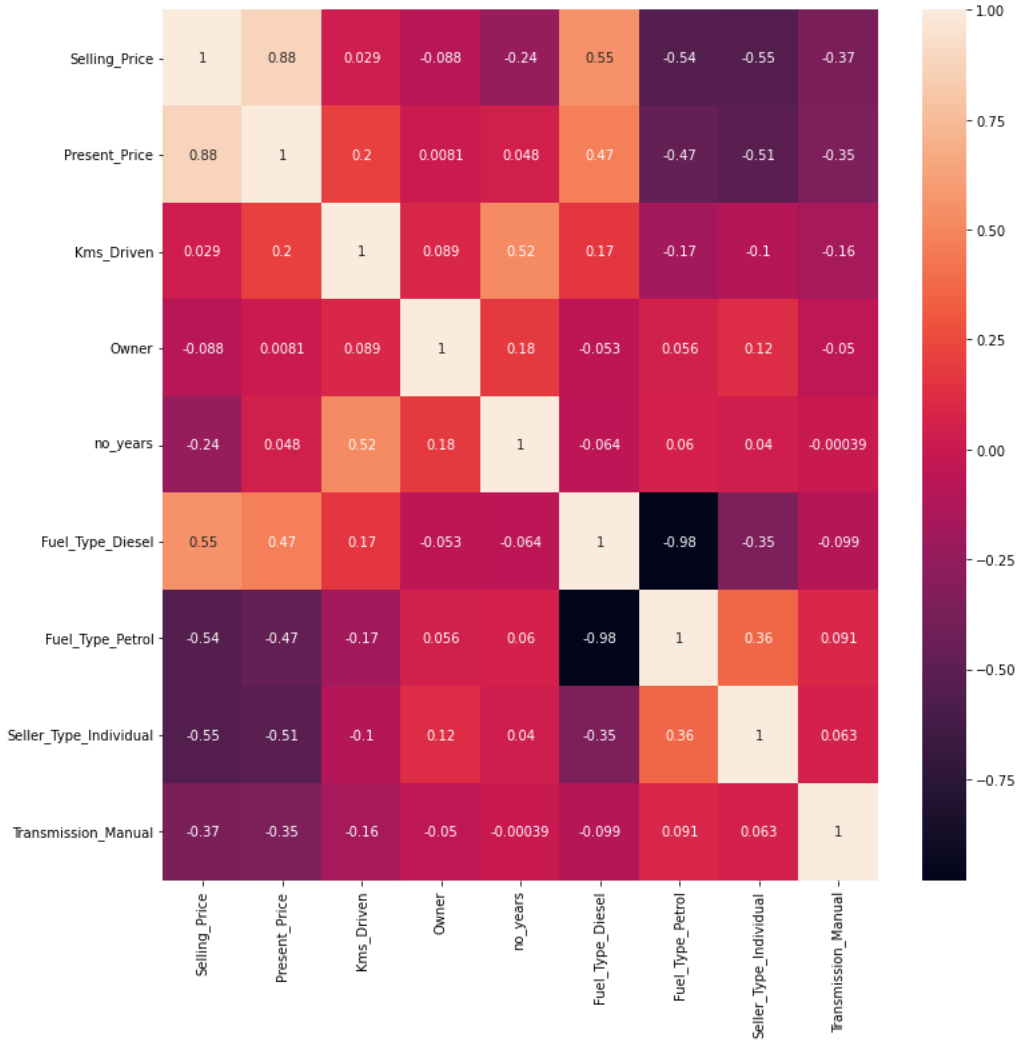

```
import matplotlib.pyplot as plt
```



```
%matplotlib inline
```

```
corrmat = final_df.corr()
top_corr_features = corrmat.index
plt.figure(figsize=(12,12))
#plot heatmap
sns.heatmap(final_df[top_corr_features].corr(),annot=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb0a7f8a370>
```




```
#independent and dependent features

x = final_df.iloc[:,1:]
y = final_df.iloc[:,0]
```

```
x.head()
```

	Present_Price	Kms_Driven	Owner	no_years	Fuel_Type_Diesel	Fuel_Type_Petrol	Seller_Type_Individual	Transmission_Manual
0	5.59	27000	0	9	0	1	0	1
1	9.54	43000	0	10	1	0	0	1
2	9.85	6900	0	6	0	1	0	1
3	4.15	5200	0	12	0	1	0	1
4	6.87	42450	0	9	1	0	0	1

```
y.head()
```

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

```
#feature importance

from sklearn.ensemble import ExtraTreesRegressor
model = ExtraTreesRegressor()
model.fit(x,y)
```

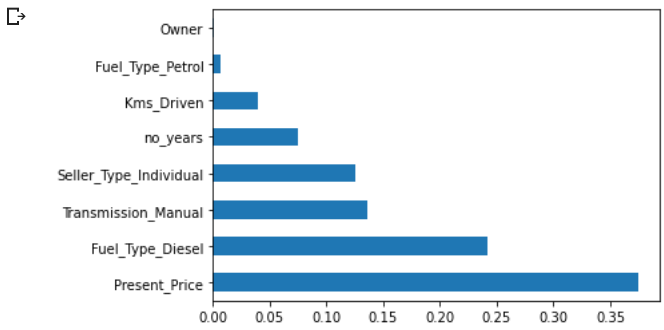
```
ExtraTreesRegressor()
```

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

```
[0.37488899 0.03939027 0.00081991 0.07497967 0.24146927 0.00696529
 0.12515918 0.13632742]
```

```
#plot the graph for feature importance for better visualization

feat_imp = pd.Series(model.feature_importances_, index = x.columns)
feat_imp.nlargest(10).plot(kind='barh')
plt.show()
```




```
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, random_state=42)
```

```
x_train.shape
```

```
(240, 8)
```

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

```
#hyper parameters
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]
```

```
print(f'max_features are :' , max_features)
print(f'max depth is :', max_depth)
print(f'min sample split is :', min_samples_split)
print(f'min_sample leaf is :' , min_samples_leaf)
```

```
max_features are : ['auto', 'sqrt']
max depth is : [5, 10, 15, 20, 25, 30]
min sample split is : [2, 5, 10, 15, 100]
min_sample leaf is : [1, 2, 5, 10]
```

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

```
# randomized searchCV

# 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']

#max number of levels in a tree
max_depth = [int(x) for x in np.linspace(5, 30, num=6)]

#max_depth.append(none)

#min number of samples required to split a node
min_samples_split = [2,5,10,15,100]

#min number of samples required at each node

min_samples_leaf = [1,2,5,10]
```

```
#creating randomgrid

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
```

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

```
rf_random = RandomizedSearchCV(estimator=rf,param_distributions=random_grid,scoring = 'neg_mean_squared_error', cv=5,verbose=2, n_iter=10,rar
```

```
rf_random.fit(x_train,y_train)
```

```
[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=15, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=1100; total time=   1.3s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=1100; total time=   1.3s
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=1100; total time=   1.3s
[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=   1.2s
```

```
[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=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=   0.8s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=10, min_samples_split=5, n_estimators=700; total time=   0.8s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=10, min_samples_split=5, n_estimators=700; total time=   0.8s
[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=   1.3s
[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=   1.5s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=1, min_samples_split=2, n_estimators=1000; total time=   2.1s
[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=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=   1.2s
[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=   1.2s
[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=15, max_features=sqrt, min_samples_leaf=1, min_samples_split=15, n_estimators=300; total time=   0.4s
[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=   0.4s
[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=   1.2s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=700; total time=   1.4s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=700; total time=   1.2s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=700; total time=   0.8s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, 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=   0.9s
[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=   0.9s
[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=   0.9s
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)
```

```
predictions = rf_random.predict(x_test)
```

```
predictions
```

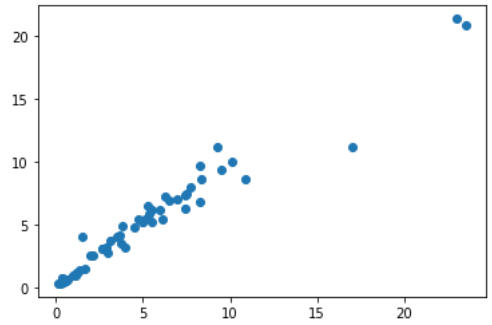
```
array([ 0.75179,  9.9933 ,  5.17728,  0.33078,  6.98499,  6.2624 ,
        1.02993,  0.61726,  0.50817,  6.1902 , 11.16262,  0.90538,
        7.45367,  0.74514,  5.40965,  3.03619,  0.97641, 11.18989,
        0.63514,  1.45111,  0.52301,  8.59059,  5.86227,  2.75645,
        0.54202,  3.52715,  5.2114 ,  3.11072,  1.19504,  1.12149,
        0.64892,  9.64614,  0.44774,  2.58386,  7.96578,  4.119  ,
        6.1063 ,  4.83749,  3.16928,  5.40611,  3.99012,  3.99178,
        4.81095,  0.55995,  6.94965,  0.56941,  7.208  ,  6.48851,
        3.1188 ,  3.69065,  5.40685,  1.38947, 20.86298, 21.39157,
        6.76525,  8.63588,  5.1538 ,  9.38369,  2.57317,  7.3712 ,
        0.29914])
```

```
sns.distplot(y_test-predictions)
```

```
/usr/local/lib/python3.8/dist-packages/seaborn/distributions.py:2619: FutureWarning: `d
  warnings.warn(msg, FutureWarning)
<matplotlib.axes._subplots.AxesSubplot at 0x7fb0a6b25b80>
```

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

```
<matplotlib.collections.PathCollection at 0x7fb0a4760d30>
```




```
import pickle
#open a file where you want to store the data
file = open('random_forest_regression_model.pkl','wb')

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

✓ 0s completed at 3:38 PM