import library

```
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

```
#load data
df=pd.read_csv(r"C:\Users\[illegible]\car data.csv")
```

```
#printing first five rows
df.head()
```

	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

```
df.shape #number of rows and column present in a dataset
```

```
(301, 9)
```

```
df.columns #printing name of all the column
```

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

```
#dropping the car_name column
#name of car[illegible] won't affect car's price ,price depends upon the [illegible]
#so i will drop the column car_name from original dataframe
df.drop('Car_Name',axis=1,inplace=True)
```

```
df.isnull().sum() #so there are null value present
```

```
Year             0
Selling_Price    0
Present_Price    0
Kms_Driven       0
Fuel_Type        0
Seller_Type      0
Transmission     0
Owner            0
dtype: int64
```

```
#total 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: [illegible] KB
```

```
#[illegible]
```

```
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]
```

```
df.head()
```

	Year	Selling_Price	Present_Price	Kms_Driven	Fuel_Type	Seller_Type	Transmission	Owner
0	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
1	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
2	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
3	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
4	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

```
#[illegible]
#[illegible]
```

```
df['Current_Year']=[illegible]
```

```
df['No_of_years']=df['Current_Year']-df['Year']
df.drop(["Current_Year","Year"],axis=1[illegible])
df.head()
```

	Selling_Price	Present_Price	Kms_Driven	Fuel_Type	Seller_Type	Transmission	Owner	No_of_years
0	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
1	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
2	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
3	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
4	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

```
df=pd.get_dummies(df,drop_first=True)
df.head()
```

	Selling_Price	Present_Price	Kms_Driven	Owner	No_of_years	Fuel_Type_Diesel	Fuel_Type_Petrol	Seller_Type_Individual	Transmission_Ma[illegible]
0	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

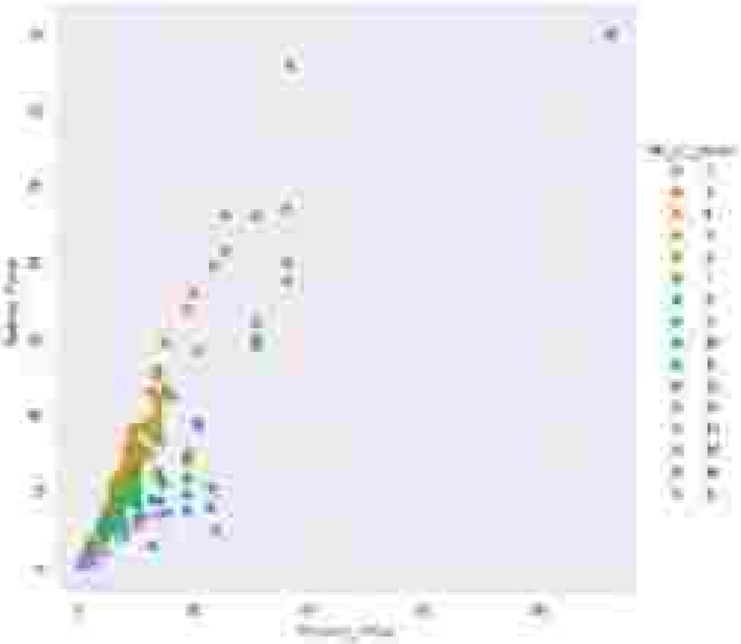
1	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
2	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
3	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
4	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

```
df.corr()
```

	Selling_Price	Present_Price	Kms_Driven	Owner	No_of_years	Fuel_Type_Diesel	Fuel_Type_Petrol	Seller_Type_Indiv
Selling_Price	1.000000	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
Present_Price	[illegible]	1.000000	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
Kms_Driven	[illegible]	[illegible]	1.000000	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
Owner	[illegible]	[illegible]	[illegible]	1.000000	[illegible]	[illegible]	[illegible]	[illegible]
No_of_years	[illegible]	[illegible]	[illegible]	[illegible]	1.000000	[illegible]	[illegible]	[illegible]
Fuel_Type_Diesel	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	1.000000	[illegible]	[illegible]
Fuel_Type_Petrol	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	1.000000	[illegible]
Seller_Type_Individual	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
Transmission_Manual	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

Data Visualization

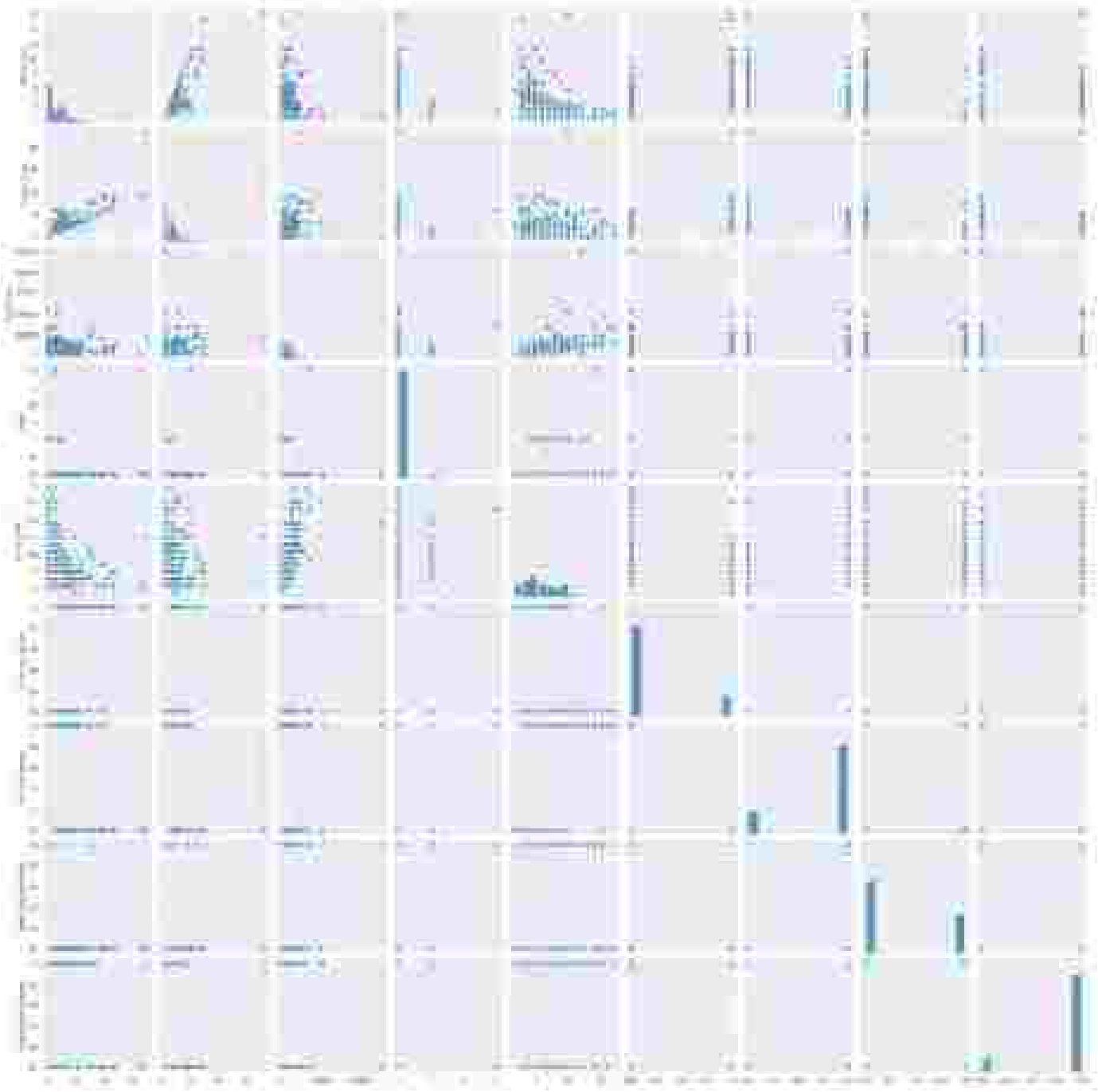
```
sns.set_style('darkgrid')
sns.FacetGrid(df,hue="No_of_years",height=6).map(plt.scatter,"Present_Price","Selling_Price").add_legend()
plt.show()
```



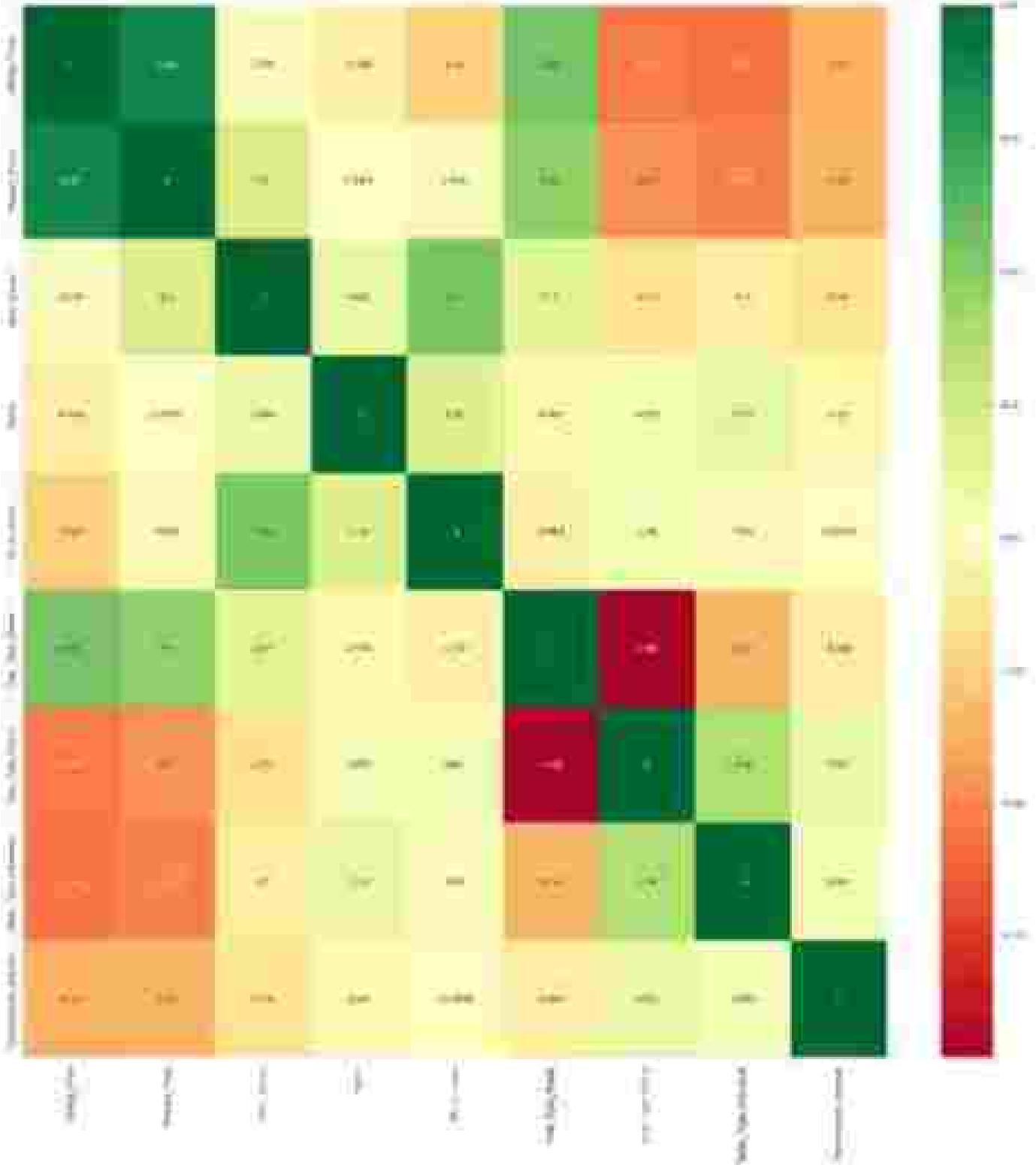
```
#More number of years you will use your car lesser the amount you will get.
```

```
sns.set_style('darkgrid')
sns.FacetGrid(df,hue="Present_Price",height=6).map(plt.scatter,"Kms","Selling_Price")
plt.xlabel("Present Price",fontsize=20)
plt.ylabel("Selling Price",fontsize=20)
plt.show()
```


Selling Price
Present Price



```
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")
```



```
#get correlation of each feature in dataset
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")
```

```
X['Owner'].unique()
```

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

```
X.head()
```

	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

```
y.head()
```

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

```
from sklearn.ensemble import ExtraTreesRegressor
import matplotlib.pyplot as plt
model = ExtraTreesRegressor()
model.fit(X,y)
```

```
ExtraTreesRegressor()
```

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

```
[illegible]
```

```
#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()
```

```
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=0)
```

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

```python
RandomForestRegressor()
```

```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]
```

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

```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]
```

```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]}
```

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

```python
# 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',n_iter
```

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

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
[CV] END max_depth=[illegible], max_features=[illegible], min_samples_leaf=[illegible], min_samples_split=[illegible], n_estimators=[illegible]; total time= [illegible]
[CV] END max_depth=[illegible], max_features=[illegible], min_samples_leaf=[illegible], min_samples_split=[illegible], n_estimators=[illegible]; total time= [illegible]
[CV] END max_depth=[illegible], max_features=[illegible], min_samples_leaf=[illegible], min_samples_split=[illegible], n_estimators=[illegible]; total time= [illegible]
[CV] END max_depth=[illegible], max_features=[illegible], min_samples_leaf=[illegible], min_samples_split=[illegible], n_estimators=[illegible]; total time= [illegible]
[CV] END max_depth=[illegible], max_features=[illegible], min_samples_leaf=[illegible], min_samples_split=[illegible], n_estimators=[illegible]; total time= [illegible]
[CV] END max_depth=[illegible], max_features=[illegible], min_samples_leaf=[illegible], min_samples_split=[illegible], n_estimators=[illegible]; total time= [illegible]
[CV] END max_depth=[illegible], max_features=[illegible], min_samples_leaf=[illegible], min_samples_split=[illegible], n_estimators=[illegible]; total time= [illegible]
[CV] END max_depth=[illegible], max_features=[illegible], min_samples_leaf=[illegible], min_samples_split=[illegible], n_estimators=[illegible]; total time= [illegible]
[CV] END max_depth=[illegible], max_features=[illegible], min_samples_leaf=[illegible], min_samples_split=[illegible], n_estimators=[illegible]; total time= [illegible]
[CV] END max_depth=[illegible], max_features=[illegible], min_samples_leaf=[illegible], min_samples_split=[illegible], n_estimators=[illegible]; total time= [illegible]
[CV] END max_depth=[illegible], max_features=[illegible], min_samples_leaf=[illegible], min_samples_split=[illegible], n_estimators=[illegible]; total time= [illegible]
[CV] END max_depth=[illegible], max_features=[illegible], min_samples_leaf=[illegible], min_samples_split=[illegible], n_estimators=[illegible]; total time= [illegible]
[CV] END max_depth=[illegible], max_features=[illegible], min_samples_leaf=[illegible], min_samples_split=[illegible], n_estimators=[illegible]; total time= [illegible]
[CV] END max_depth=[illegible], max_features=[illegible], min_samples_leaf=[illegible], min_samples_split=[illegible], n_estimators=[illegible]; total time= [illegible]
```

```
rf_random.best_params_
```

```
{'n_estimators': [illegible],
 'min_samples_split': [illegible],
 'min_samples_leaf': [illegible],
 'max_features': 'sqrt',
 'max_depth': [illegible]}
```

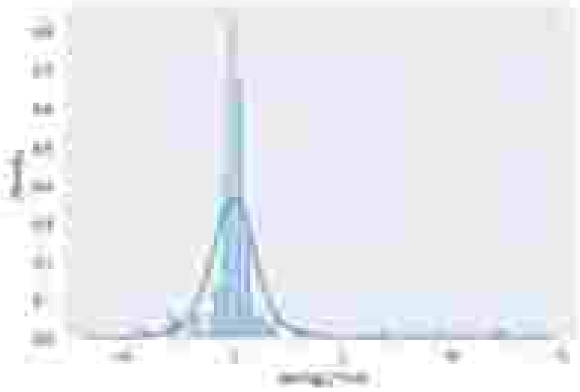
```
rf_random.best_score_
```

```
[illegible]
```

```
prediction=rf_random.predict(X_test)
```

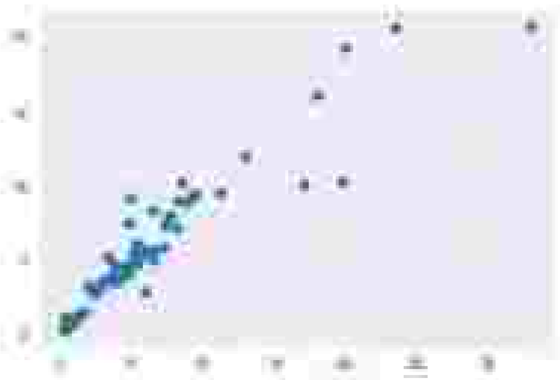
```
sns.distplot(y_test-prediction)
```

```
[illegible] FutureWarning: [illegible]
  warnings.warn(msg, FutureWarning)
<AxesSubplot:xlabel='Selling_Price', ylabel='Density'>
```



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

```
<matplotlib.collections.PathCollection at [illegible]>
```



```
from sklearn import metrics
```

```
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: [illegible]
MSE: [illegible]
RMSE: [illegible]
```

```
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, file)
```