CAR PRICE PREDICTION WITH MACHINE LEARNING

May 15, 2023

1 CAR PRICE PREDICTION WITH MACHINE LEARNING

[1]:
```
#importing the libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

[2]:
```
#reading the dataset
data=pd.read_csv("C:/Users/MyPc/Downloads/car data.csv")
print(data.shape)
data.head()
```

```
(301, 9)
```

[2]:

	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

[3]:
```
#Checking if there are any missing values
data.isnull().sum()
```

[3]:
```
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
```

There are no missing values in the dataset

2 Checking cardinality of independent categorical variables in the dataset

[4]:
```
print('Unique elements in Seller_Type are',data['Seller_Type'].unique())
print('Unique elements in Fuel_Type are',data['Fuel_Type'].unique())
print('Unique elements in Transmission are',data['Transmission'].unique())
print('Unique elements in Owner are',data['Owner'].unique())
print('Unique elements in Year are',data['Year'].unique())
```

```
Unique elements in Seller_Type are ['Dealer' 'Individual']
Unique elements in Fuel_Type are ['Petrol' 'Diesel' 'CNG']
Unique elements in Transmission are ['Manual' 'Automatic']
Unique elements in Owner are [0 1 3]
Unique elements in Year are [2014 2013 2017 2011 2018 2015 2016 2009 2010 2012
2003 2008 2006 2005
 2004 2007]
```

[5]:
```
print('Unique elements in Car_Name are',data['Car_Name'].nunique())
#98 unique elements
#so, rather than encoding it, we can just drop this columbn as it doesn' make␣
 ↪sense
```

```
Unique elements in Car_Name are 98
```

[6]:
```
data.describe()
```

[6]:

	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

3 Feature Engineering

Dropping the Car_Name Column

[7]:
```
dataset=data[['Year','Selling_Price','Present_Price','Kms_Driven','Fuel_Type','Seller_Type','T
dataset.head()
```

[7]:

	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

Let's make a feature variable 'Present_Year' which has all the element values as 2020. On subtracting 'Present_Year' and 'Year', we can make another feature variable as 'Number_of_Years_Old', which gives us idea about how old the car is.

[8]:
```
dataset['Present_Year']=2020
dataset['Number_of_Years_Old']=dataset['Present_Year']- dataset['Year']
dataset.head()
```

[8]:

	Year	Selling_Price	Present_Price	Kms_Driven	Fuel_Type	Seller_Type	Transmission	Owner	Present_Year	Number_of_Years_Old
0	2014	3.35	5.59	27000	Petrol	Dealer	Manual	0	2020	6
1	2013	4.75	9.54	43000	Diesel	Dealer	Manual	0	2020	7
2	2017	7.25	9.85	6900	Petrol	Dealer	Manual	0	2020	3
3	2011	2.85	4.15	5200	Petrol	Dealer	Manual	0	2020	9
4	2014	4.60	6.87	42450	Diesel	Dealer	Manual	0	2020	6

So, we can now safely drop 'Year' and 'Present_Year' columns

[9]:
```
dataset.drop(labels=['Year', 'Present_Year'],axis=1,inplace=True)
dataset.head()
```

[9]:

	Selling_Price	Present_Price	Kms_Driven	Fuel_Type	Seller_Type	Transmission	Owner	Number_of_Years_Old
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
4           4.60           6.87       42450    Diesel      Dealer

  Transmission  Owner  Number_of_Years_Old
0       Manual      0                    6
1       Manual      0                    7
2       Manual      0                    3
3       Manual      0                    9
4       Manual      0                    6
```

4 Encoding the Categorical Variables

[10]:
```
#select categorical variables from then dataset, and then implement categorical␣
 ↪encoding for nominal variables
Fuel_Type=dataset[['Fuel_Type']]
Fuel_Type=pd.get_dummies(Fuel_Type, drop_first=True)

Seller_Type=dataset[['Seller_Type']]
Seller_Type=pd.get_dummies(Seller_Type, drop_first=True)

Transmission=dataset[['Transmission']]
Transmission=pd.get_dummies(Transmission, drop_first=True)

dataset=pd.concat([dataset,Fuel_Type, Seller_Type, Transmission], axis=1)

dataset.drop(labels=['Fuel_Type', 'Seller_Type', 'Transmission'], axis=1,␣
 ↪inplace=True)

dataset.head()
```

[10]:
```
   Selling_Price  Present_Price  Kms_Driven  Owner  Number_of_Years_Old  \
0           3.35           5.59       27000      0                    6
1           4.75           9.54       43000      0                    7
2           7.25           9.85        6900      0                    3
3           2.85           4.15        5200      0                    9
4           4.60           6.87       42450      0                    6

   Fuel_Type_Diesel  Fuel_Type_Petrol  Seller_Type_Individual  \
0                 0                 1                       0
1                 1                 0                       0
2                 0                 1                       0
3                 0                 1                       0
4                 1                 0                       0

   Transmission_Manual
0                    1
```

```
1                    1
2                    1
3                    1
4                    1
```

[11]:
```
dataset.columns
```

[11]:
```
Index(['Selling_Price', 'Present_Price', 'Kms_Driven', 'Owner',
       'Number_of_Years_Old', 'Fuel_Type_Diesel', 'Fuel_Type_Petrol',
       'Seller_Type_Individual', 'Transmission_Manual'],
      dtype='object')
```

[12]:
```
#Dataset Correlation
dataset.corr()
```

[12]:

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

	Number_of_Years_Old	Fuel_Type_Diesel
Selling_Price	-0.236141	0.552339
Present_Price	0.047584	0.473306
Kms_Driven	0.524342	0.172515
Owner	0.182104	-0.053469
Number_of_Years_Old	1.000000	-0.064315
Fuel_Type_Diesel	-0.064315	1.000000
Fuel_Type_Petrol	0.059959	-0.979648
Seller_Type_Individual	0.039896	-0.350467
Transmission_Manual	-0.000394	-0.098643

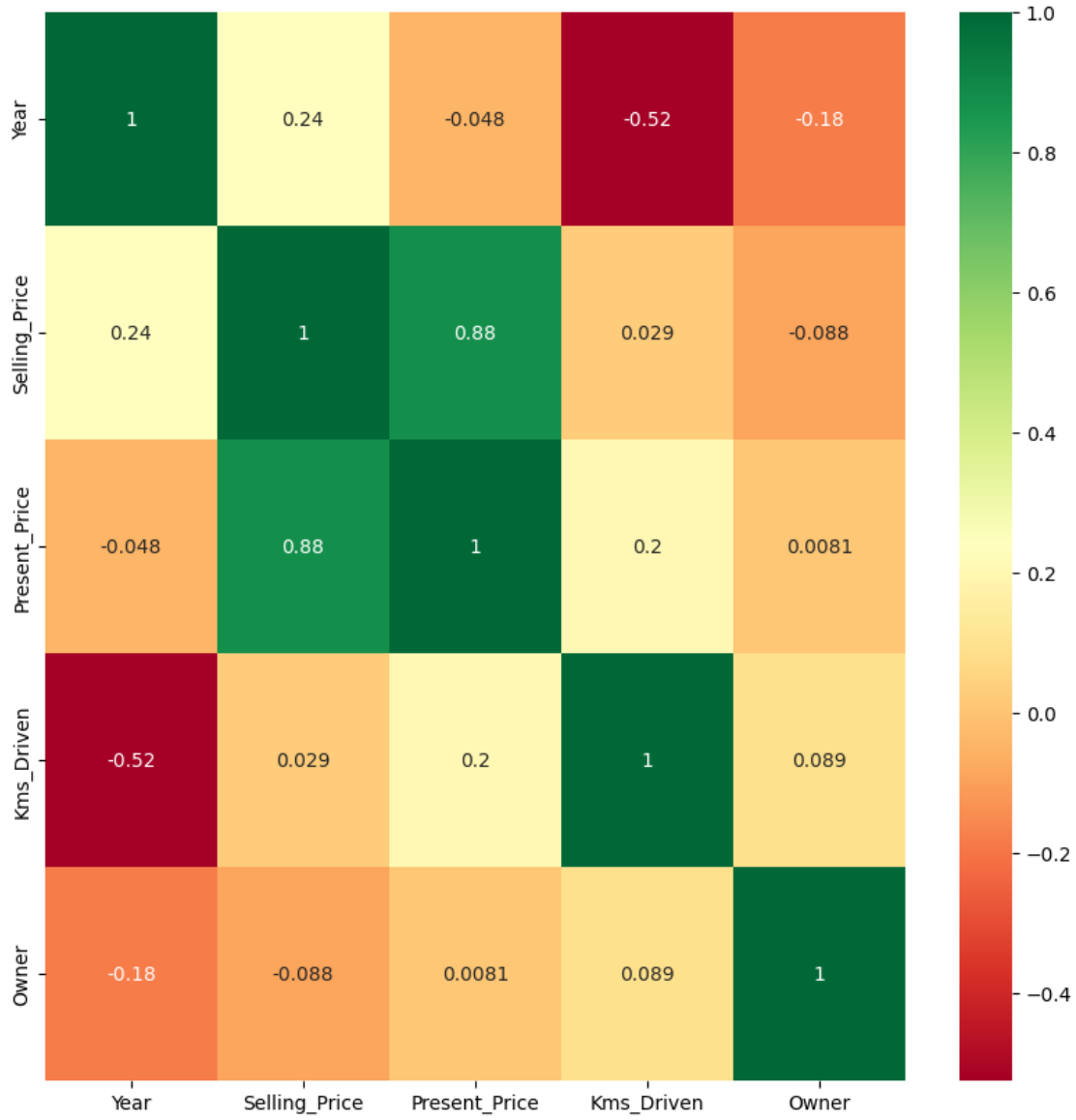
	Fuel_Type_Petrol	Seller_Type_Individual
Selling_Price	-0.540571	-0.550724
Present_Price	-0.465244	-0.512030
Kms_Driven	-0.172874	-0.101419
Owner	0.055687	0.124269
Number_of_Years_Old	0.059959	0.039896
Fuel_Type_Diesel	-0.979648	-0.350467
Fuel_Type_Petrol	1.000000	0.358321
Seller_Type_Individual	0.358321	1.000000
Transmission_Manual	0.091013	0.063240

```
                      Transmission_Manual
Selling_Price                   -0.367128
Present_Price                   -0.348715
Kms_Driven                      -0.162510
Owner                           -0.050316
Number_of_Years_Old             -0.000394
Fuel_Type_Diesel                -0.098643
Fuel_Type_Petrol                 0.091013
Seller_Type_Individual           0.063240
Transmission_Manual              1.000000
```

5 Data Visualization and Correlation

[13]:
```
#Correlations of features in dataset
corrmat = data.corr()
top_corr_features = corrmat.index
plt.figure(figsize=(10,10))
#Plot heat map
sns.heatmap(data[top_corr_features].corr(),annot=True,cmap="RdYlGn")
```

[13]: <AxesSubplot:>



[14]:
```
sell=dataset['Selling_Price']
dataset.drop(['Selling_Price'], axis=1, inplace=True)
dataset=dataset.join(sell)
dataset.head()
```

[14]:

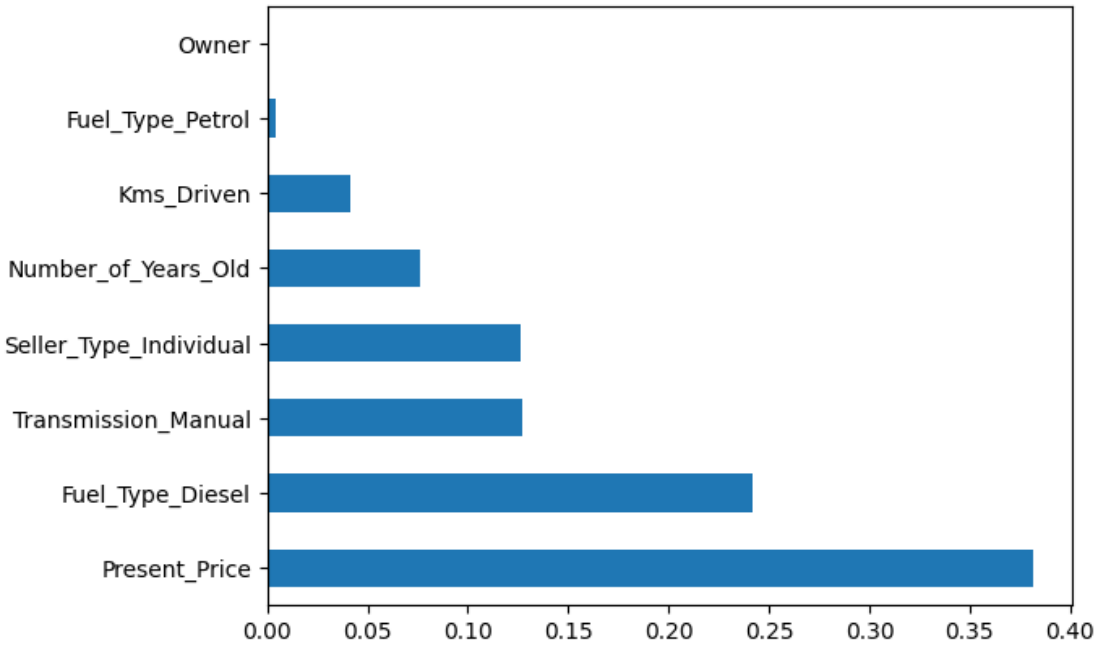
	Present_Price	Kms_Driven	Owner	Number_of_Years_Old	Fuel_Type_Diesel	\
0	5.59	27000	0	6	0	
1	9.54	43000	0	7	1	
2	9.85	6900	0	3	0	
3	4.15	5200	0	9	0	
4	6.87	42450	0	6	1	

```
   Fuel_Type_Petrol  Seller_Type_Individual  Transmission_Manual  \
0                 1                       0                    1
1                 0                       0                    1
2                 1                       0                    1
3                 1                       0                    1
4                 0                       0                    1

   Selling_Price
0           3.35
1           4.75
2           7.25
3           2.85
4           4.60
```

[15]:
```
X=dataset.iloc[:,:-1]
y=dataset.iloc[:,-1]
```

[16]:
```
y=dataset.iloc[:,-1]
### To determine important features, make use of ExtraTreesRegressor
from sklearn.ensemble import ExtraTreesRegressor
model = ExtraTreesRegressor()
model.fit(X,y)

print(model.feature_importances_)

#plot graph of feature importances for better visualization
feat_importances = pd.Series(model.feature_importances_, index=X.columns)
feat_importances.nlargest(10).plot(kind='barh')
plt.show()
```

```
[0.38171483 0.04157732 0.00081952 0.07626293 0.24183132 0.00438483
 0.1264379  0.12697134]
```



Owner' has zero feature importance i.e. nil on the dependent variable, 'Selling_Price'

6 Model Building and Training

[17]:
```
X=dataset.iloc[:,:-1].values
y=dataset.iloc[:,-1].values
```

[18]:
```
from sklearn.model_selection import cross_val_score
from sklearn import metrics
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
#from sklearn.model_selection import RandomizedSearchCV
#from sklearn.model_selection import GridSearchCV
#from sklearn.model_selection import StratifiedKFold
#kfold = StratifiedKFold(n_splits=3)
```

[19]:
```
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)
```

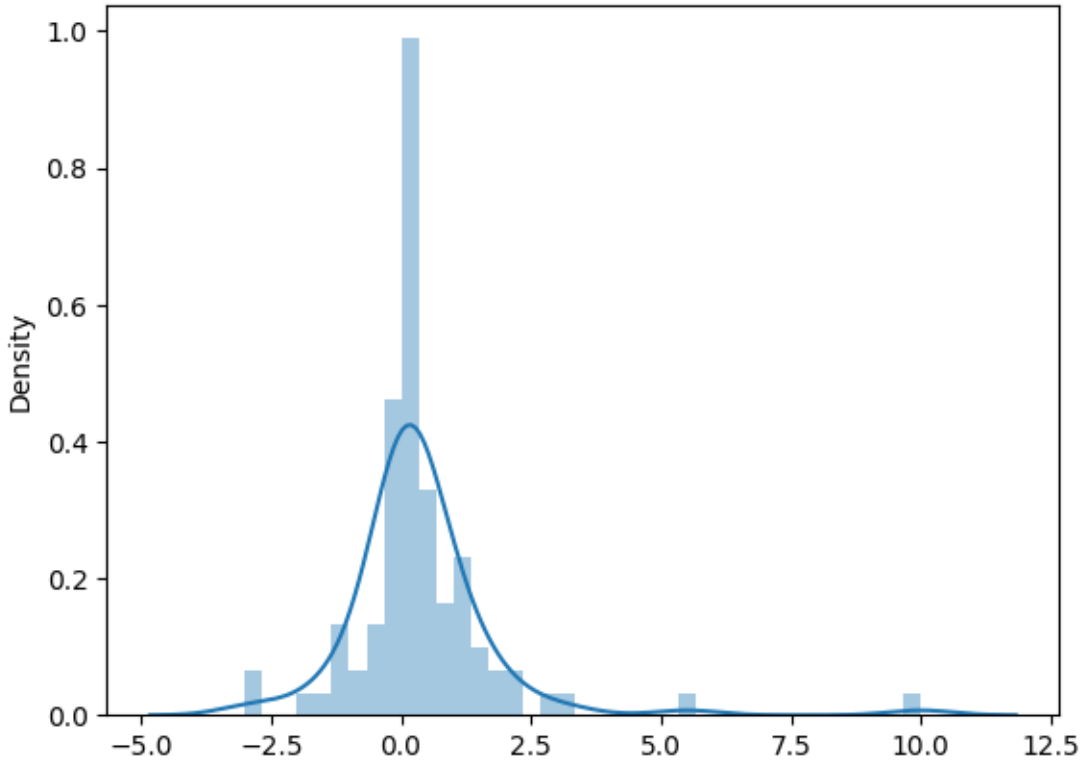
7 Decision Tree Regressor

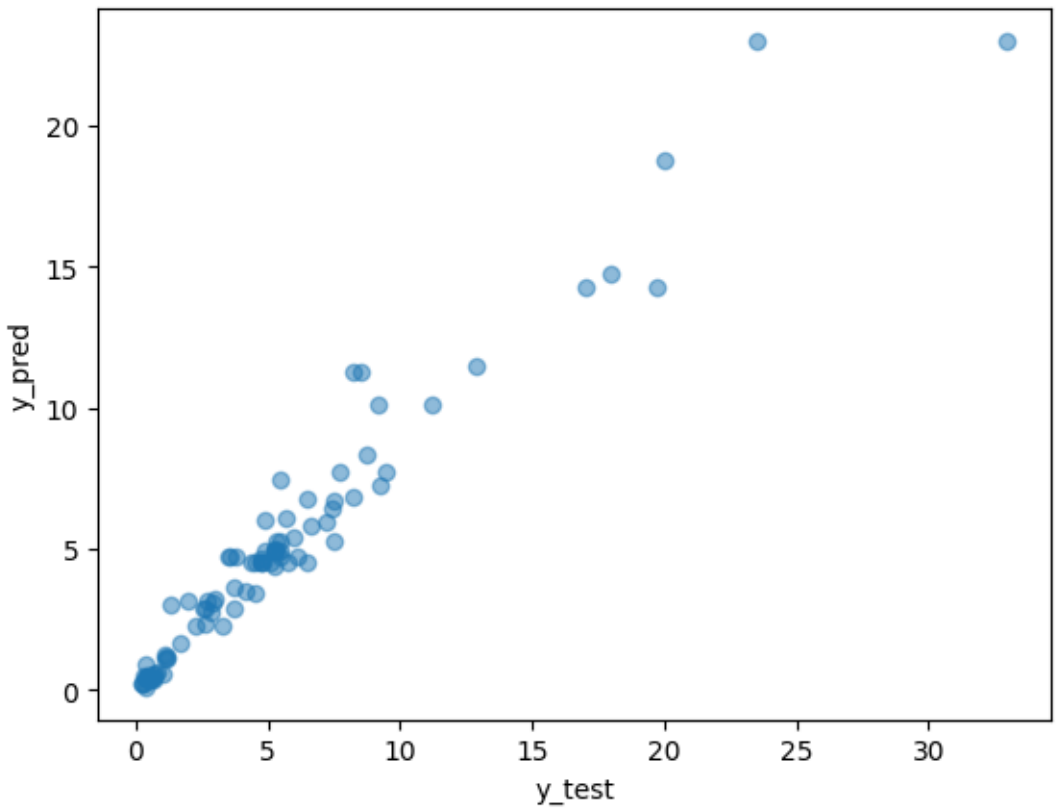
[20]:
```
#Decision Tree Regressor
from sklearn.tree import DecisionTreeRegressor
dt_reg = DecisionTreeRegressor(random_state = 0)
dt_reg.fit(X_train, y_train)
y_pred=dt_reg.predict(X_test)

print("Decision Tree Score on Training set is",dt_reg.score(X_train,␣
 ↪y_train))#Training Accuracy
print("Decision Tree Score on Test Set is",dt_reg.score(X_test,␣
 ↪y_test))#Testing Accuracy

accuracies = cross_val_score(dt_reg, X_train, y_train, cv = 5)
print(accuracies)
print("Accuracy: {:.2f} %".format(accuracies.mean()*100))
print("Standard Deviation: {:.2f} %".format(accuracies.std()*100))

mae=mean_absolute_error(y_pred, y_test)
print("Mean Absolute Error:" , mae)

mse=mean_squared_error(y_test, y_pred)
print("Mean Squared Error:" , mse)

print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))

print('The r2_score is', metrics.r2_score(y_test, y_pred))

sns.distplot(y_test-y_pred)
plt.show()

plt.scatter(y_test, y_pred, alpha = 0.5)
plt.xlabel("y_test")
plt.ylabel("y_pred")
plt.show()
```

```
Decision Tree Score on Training set is 1.0
Decision Tree Score on Test Set is 0.9202815383374512
[0.9542394  0.84409548 0.69916028 0.924205   0.92156403]
Accuracy: 86.87 %
Standard Deviation: 9.22 %
Mean Absolute Error: 0.8102197802197801
Mean Squared Error: 2.3840648351648355
RMSE: 1.5440417206684653
The r2_score is 0.9202815383374512
```

```
C:\Users\MyPc\AppData\Local\Temp\ipykernel_2060\3849505116.py:25: UserWarning:
```

```
`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(y_test-y_pred)
```





8 Random Forest Regressor

[21]:
```
#Random Forest Regression
from sklearn.ensemble import RandomForestRegressor
rf_reg =␣
 ↪RandomForestRegressor(n_estimators=400,min_samples_split=15,min_samples_leaf=2,
max_features='auto', max_depth=30)
rf_reg.fit(X_train, y_train)
y_pred=rf_reg.predict(X_test)

print("Random Forest Score on Training set is",rf_reg.score(X_train,␣
 ↪y_train))#Training Accuracy
print("Random Forest Score on Test Set is",rf_reg.score(X_test,␣
 ↪y_test))#Testing Accuracy

accuracies = cross_val_score(rf_reg, X_train, y_train, cv = 5)
print(accuracies)
print("Accuracy: {:.2f} %".format(accuracies.mean()*100))
print("Standard Deviation: {:.2f} %".format(accuracies.std()*100))
```

```
mae=mean_absolute_error(y_pred, y_test)
print("Mean Absolute Error:" , mae)

mse=mean_squared_error(y_test, y_pred)
print("Mean Squared Error:" , mse)

print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))

print('The r2_score is', metrics.r2_score(y_test, y_pred))

sns.distplot(y_test-y_pred)
plt.show()

plt.scatter(y_test, y_pred, alpha = 0.5)
plt.xlabel("y_test")
plt.ylabel("y_pred")
plt.show()
```

```
C:\python37\lib\site-packages\sklearn\ensemble\_forest.py:416: FutureWarning:
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To
keep the past behaviour, explicitly set `max_features=1.0` or remove this
parameter as it is also the default value for RandomForestRegressors and
ExtraTreesRegressors.
  warn(
```

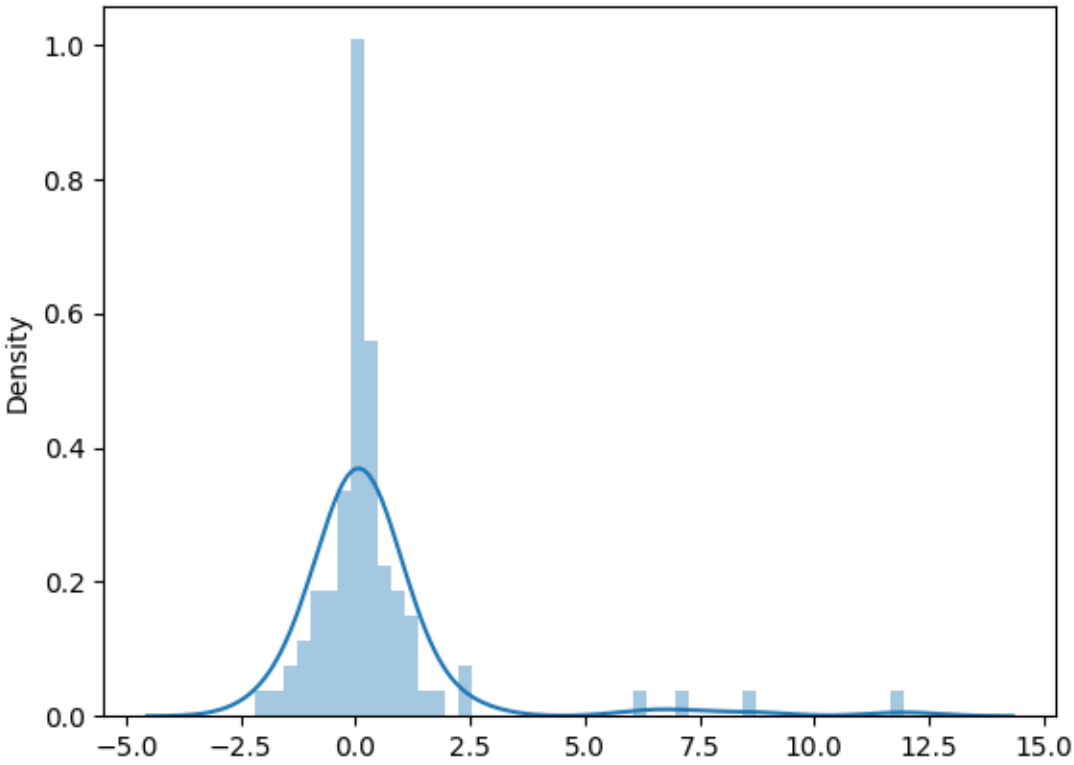
```
Random Forest Score on Training set is 0.910125724627176
Random Forest Score on Test Set is 0.8677638821161013
```

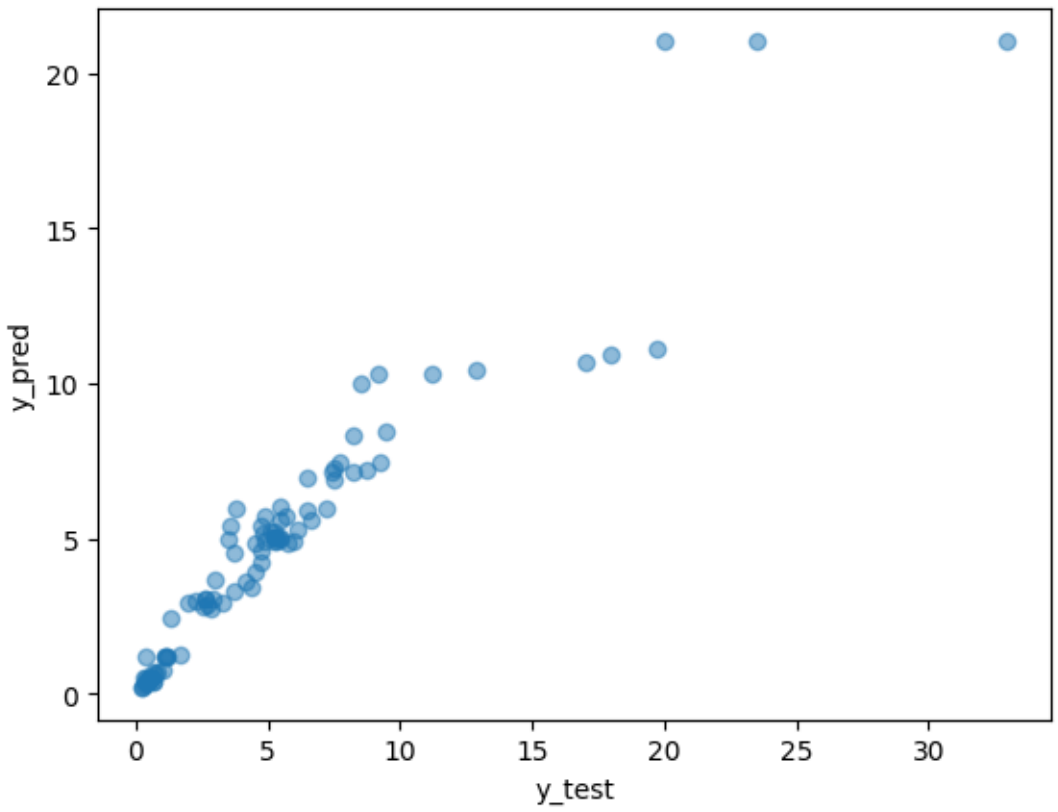
```
C:\python37\lib\site-packages\sklearn\ensemble\_forest.py:416: FutureWarning:
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To
keep the past behaviour, explicitly set `max_features=1.0` or remove this
parameter as it is also the default value for RandomForestRegressors and
ExtraTreesRegressors.
  warn(
C:\python37\lib\site-packages\sklearn\ensemble\_forest.py:416: FutureWarning:
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To
keep the past behaviour, explicitly set `max_features=1.0` or remove this
parameter as it is also the default value for RandomForestRegressors and
ExtraTreesRegressors.
  warn(
C:\python37\lib\site-packages\sklearn\ensemble\_forest.py:416: FutureWarning:
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To
keep the past behaviour, explicitly set `max_features=1.0` or remove this
parameter as it is also the default value for RandomForestRegressors and
ExtraTreesRegressors.
  warn(
C:\python37\lib\site-packages\sklearn\ensemble\_forest.py:416: FutureWarning:
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To
keep the past behaviour, explicitly set `max_features=1.0` or remove this
```

```
parameter as it is also the default value for RandomForestRegressors and
ExtraTreesRegressors.
  warn(
C:\python37\lib\site-packages\sklearn\ensemble\_forest.py:416: FutureWarning:
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To
keep the past behaviour, explicitly set `max_features=1.0` or remove this
parameter as it is also the default value for RandomForestRegressors and
ExtraTreesRegressors.
  warn(
```

```
[0.94954202 0.82350283 0.60814318 0.83645274 0.93317529]
Accuracy: 83.02 %
Standard Deviation: 12.19 %
Mean Absolute Error: 0.8851733316897348
Mean Squared Error: 3.9546608402984447
RMSE: 1.9886329073759301
The r2_score is 0.8677638821161013
```

```
C:\Users\MyPc\AppData\Local\Temp\ipykernel_2060\426049121.py:26: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(y_test-y_pred)
```


Density
1.0
0.8
0.6
0.4
0.2
0.0
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
12.5
15.0



9 Voting Regressor

Voting Regressor is an ensemble meta-estimator that fits several base regressors, each on the whole dataset to average the individual predictions to form a final prediction.

[22]:
```
from sklearn.ensemble import VotingRegressor
vot_reg = VotingRegressor([('DecisionTree', dt_reg), ('RandomForestRegressor',␣
 ↪rf_reg)])
vot_reg.fit(X_train, y_train)
y_pred=vot_reg.predict(X_test)

print("Voting Regresssor Score on Training set is",vot_reg.score(X_train,␣
 ↪y_train))#Training Accuracy
print("Voting Regresssor Score on Test Set is",vot_reg.score(X_test,␣
 ↪y_test))#Testing Accuracy

accuracies = cross_val_score(vot_reg, X_train, y_train, cv = 5)
print(accuracies)
print("Accuracy: {:.2f} %".format(accuracies.mean()*100))
print("Standard Deviation: {:.2f} %".format(accuracies.std()*100))
```

```python
mae=mean_absolute_error(y_pred, y_test)
print("Mean Absolute Error:" , mae)

mse=mean_squared_error(y_test, y_pred)
print("Mean Squared Error:" , mse)

print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))

print('The r2_score is', metrics.r2_score(y_test, y_pred))

sns.distplot(y_test-y_pred)
plt.show()

plt.scatter(y_test, y_pred, alpha = 0.5)
plt.xlabel("y_test")
plt.ylabel("y_pred")
plt.show()
```

```
C:\python37\lib\site-packages\sklearn\ensemble\_forest.py:416: FutureWarning:
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To
keep the past behaviour, explicitly set `max_features=1.0` or remove this
parameter as it is also the default value for RandomForestRegressors and
ExtraTreesRegressors.
  warn(
```

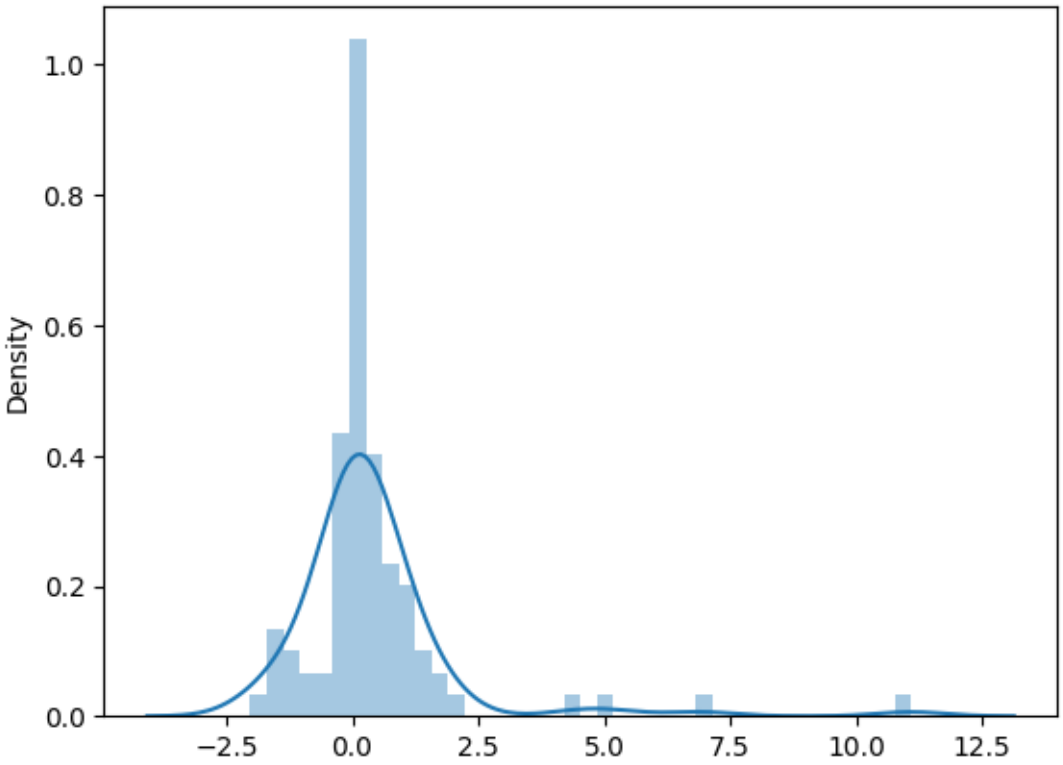
```
Voting Regresssor Score on Training set is 0.97724532755527568
Voting Regresssor Score on Test Set is 0.901260612837499
```

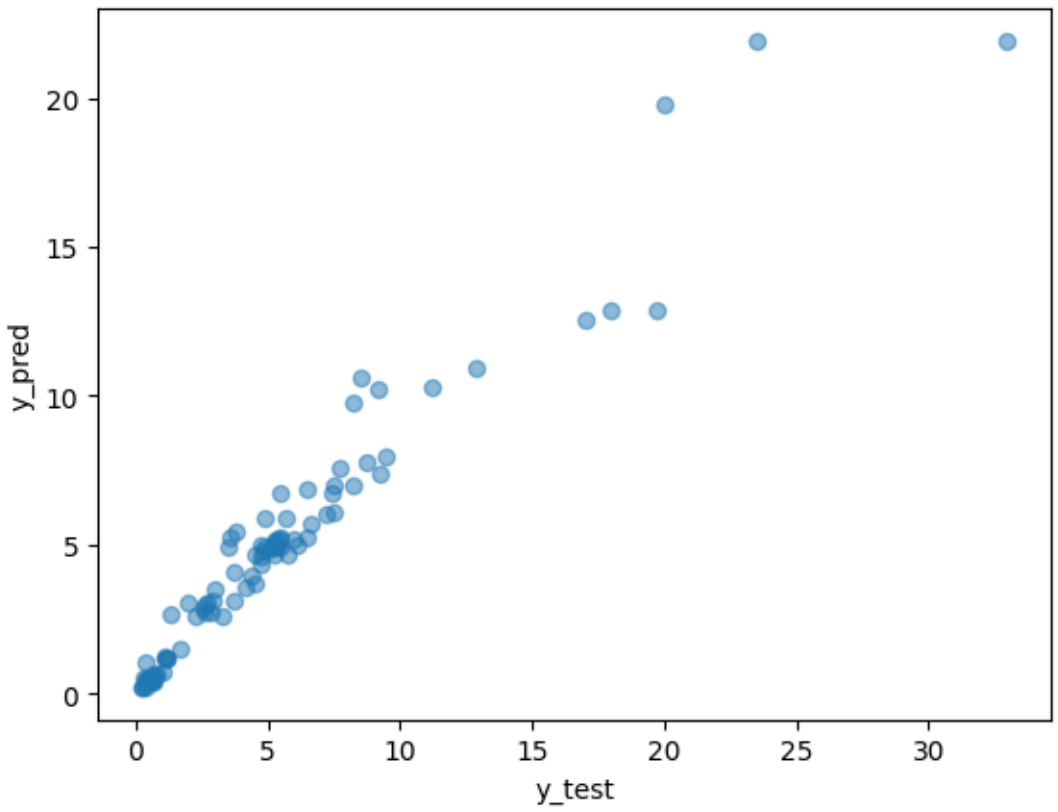
```
C:\python37\lib\site-packages\sklearn\ensemble\_forest.py:416: FutureWarning:
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To
keep the past behaviour, explicitly set `max_features=1.0` or remove this
parameter as it is also the default value for RandomForestRegressors and
ExtraTreesRegressors.
  warn(
C:\python37\lib\site-packages\sklearn\ensemble\_forest.py:416: FutureWarning:
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To
keep the past behaviour, explicitly set `max_features=1.0` or remove this
parameter as it is also the default value for RandomForestRegressors and
ExtraTreesRegressors.
  warn(
C:\python37\lib\site-packages\sklearn\ensemble\_forest.py:416: FutureWarning:
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To
keep the past behaviour, explicitly set `max_features=1.0` or remove this
parameter as it is also the default value for RandomForestRegressors and
ExtraTreesRegressors.
  warn(
C:\python37\lib\site-packages\sklearn\ensemble\_forest.py:416: FutureWarning:
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To
```

```
keep the past behaviour, explicitly set `max_features=1.0` or remove this
parameter as it is also the default value for RandomForestRegressors and
ExtraTreesRegressors.
  warn(
C:\python37\lib\site-packages\sklearn\ensemble\_forest.py:416: FutureWarning:
`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To
keep the past behaviour, explicitly set `max_features=1.0` or remove this
parameter as it is also the default value for RandomForestRegressors and
ExtraTreesRegressors.
  warn(
```

```
[0.9701364  0.84072249 0.68810097 0.89923624 0.93958959]
Accuracy: 86.76 %
Standard Deviation: 9.96 %
Mean Absolute Error: 0.826338509121777
Mean Squared Error: 2.9529057118074653
RMSE: 1.7184020809483052
The r2_score is 0.901260612837499
```

```
C:\Users\MyPc\AppData\Local\Temp\ipykernel_2060\4051451342.py:24: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(y_test-y_pred)
```


Density
1.0
0.8
0.6
0.4
0.2
0.0
−2.5
0.0
2.5
5.0
7.5
10.0
12.5



10 Dump the model selected as a Pickle File

[23]:
```
import pickle
pickle.dump(vot_reg, open("vot_reg.pkl", "wb"))

# load model from file
model = pickle.load(open("vot_reg.pkl", "rb"))

model.predict([[9.85, 6900, 0, 3, 0, 1, 0, 1]])
```

[23]:
```
array([7.36213156])
```