car-price-prediction-randomforestregression-vs-xgbregression

February 27, 2024

1 CAR PRICE PREDICTION & EDA WITH XGBoost Regression

2 Import library

[1]:
```
#Linear algebra & data processing
import numpy as np
import pandas as pd

#Data visualization libraries
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px

#Import transofmers
from sklearn.preprocessing import StandardScaler, MinMaxScaler, LabelEncoder

#Import Regression method
from sklearn.svm import SVR
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from xgboost import XGBRegressor
from sklearn.linear_model import Ridge, ElasticNet, Lasso, LogisticRegression,␣
↪LinearRegression

#Import model selection
from sklearn.model_selection import train_test_split, cross_val_score,␣
↪GridSearchCV, KFold

#Import Accuracy Metrics
from sklearn.metrics import  r2_score, max_error,mean_squared_error,␣
↪mean_absolute_error
from time import time
```

```python
import warnings
warnings.filterwarnings("ignore")
```

3 Some functions

[2]:
```python
def pourcentage(data):
    n = data.shape[0]
    ret = pd.DataFrame(data.isnull().sum(), columns=['missing_number'])
    ret['pourcentage_missing_number'] = (ret['missing_number']/n)*100
    ret['types'] = data.dtypes
    ret['duplicate'] = data.duplicated(keep=False).sum()
    ret['NAN'] = data.isna().sum()
    return ret
```

4 Import data

[3]:
```python
df_car = pd.read_csv("/kaggle/input/carr-details/Car details v3.csv")
df_car.head()
```

[3]:
```
                           name  year  selling_price  km_driven    fuel  \
0        Maruti Swift Dzire VDI  2014         450000     145500  Diesel
1  Skoda Rapid 1.5 TDI Ambition  2014         370000     120000  Diesel
2      Honda City 2017-2020 EXi  2006         158000     140000  Petrol
3     Hyundai i20 Sportz Diesel  2010         225000     127000  Diesel
4        Maruti Swift VXI BSIII  2007         130000     120000  Petrol

  seller_type transmission         owner     mileage   engine   max_power  \
0  Individual       Manual   First Owner   23.4 kmpl  1248 CC      74 bhp
1  Individual       Manual  Second Owner  21.14 kmpl  1498 CC  103.52 bhp
2  Individual       Manual   Third Owner   17.7 kmpl  1497 CC      78 bhp
3  Individual       Manual   First Owner   23.0 kmpl  1396 CC      90 bhp
4  Individual       Manual   First Owner   16.1 kmpl  1298 CC    88.2 bhp

                     torque  seats
0            190Nm@ 2000rpm    5.0
1       250Nm@ 1500-2500rpm    5.0
2     12.7@ 2,700(kgm@ rpm)    5.0
3  22.4 kgm at 1750-2750rpm    5.0
4     11.5@ 4,500(kgm@ rpm)    5.0
```

5 Pre-processiong Data

[4]:
```python
pourcentage(df_car)
```

[4]:
```
              missing_number  pourcentage_missing_number    types  duplicate  \
name                       0                    0.000000   object       1827
year                       0                    0.000000    int64       1827
selling_price              0                    0.000000    int64       1827
km_driven                  0                    0.000000    int64       1827
fuel                       0                    0.000000   object       1827
seller_type                0                    0.000000   object       1827
transmission               0                    0.000000   object       1827
owner                      0                    0.000000   object       1827
mileage                  221                    2.718996   object       1827
engine                   221                    2.718996   object       1827
max_power                215                    2.645177   object       1827
torque                   222                    2.731299   object       1827
seats                    221                    2.718996  float64       1827

               NAN
name             0
year             0
selling_price    0
km_driven        0
fuel             0
seller_type      0
transmission     0
owner            0
mileage        221
engine         221
max_power      215
torque         222
seats          221
```

[5]:
```python
df_car = df_car.dropna(axis=0)

def convertToNumber(s:str):
    d=""
    for i in list(s):
        if i.isdigit():
            d += i
    return eval(d)

df_car["mileage"] = df_car["mileage"].apply(convertToNumber)
df_car["engine"] = df_car["engine"].apply(convertToNumber)
df_car["max_power"] = df_car["max_power"].apply(convertToNumber)
```

[6]:
```python
pourcentage(df_car)
```

[6]:
```
              missing_number  pourcentage_missing_number    types  duplicate  \
name                       0                         0.0   object       1801
```

```
year                            0                         0.0    int64       1801
selling_price                   0                         0.0    int64       1801
km_driven                       0                         0.0    int64       1801
fuel                            0                         0.0   object       1801
seller_type                     0                         0.0   object       1801
transmission                    0                         0.0   object       1801
owner                           0                         0.0   object       1801
mileage                         0                         0.0    int64       1801
engine                          0                         0.0    int64       1801
max_power                       0                         0.0    int64       1801
torque                          0                         0.0   object       1801
seats                           0                         0.0  float64       1801

               NAN
name             0
year             0
selling_price    0
km_driven        0
fuel             0
seller_type      0
transmission     0
owner            0
mileage          0
engine           0
max_power        0
torque           0
seats            0
```

[7]:
```
df_car.head()
```

[7]:
```
                           name  year  selling_price  km_driven    fuel  \
0        Maruti Swift Dzire VDI  2014         450000     145500  Diesel
1  Skoda Rapid 1.5 TDI Ambition  2014         370000     120000  Diesel
2      Honda City 2017-2020 EXi  2006         158000     140000  Petrol
3     Hyundai i20 Sportz Diesel  2010         225000     127000  Diesel
4        Maruti Swift VXI BSIII  2007         130000     120000  Petrol

  seller_type transmission         owner  mileage  engine  max_power  \
0  Individual       Manual   First Owner      234    1248         74
1  Individual       Manual  Second Owner     2114    1498      10352
2  Individual       Manual   Third Owner      177    1497         78
3  Individual       Manual   First Owner      230    1396         90
4  Individual       Manual   First Owner      161    1298        882

                  torque  seats
0           190Nm@ 2000rpm    5.0
1      250Nm@ 1500-2500rpm    5.0
```

```
2     12.7@ 2,700(kgm@ rpm)    5.0
3  22.4 kgm at 1750-2750rpm    5.0
4     11.5@ 4,500(kgm@ rpm)    5.0
```

[8]:
```
data = df_car.drop(['name', 'torque', 'seller_type', 'owner'], axis=1)
data.head()
```

[8]:
```
   year  selling_price  km_driven    fuel transmission  mileage  engine  \
0  2014         450000     145500  Diesel       Manual      234    1248
1  2014         370000     120000  Diesel       Manual     2114    1498
2  2006         158000     140000  Petrol       Manual      177    1497
3  2010         225000     127000  Diesel       Manual      230    1396
4  2007         130000     120000  Petrol       Manual      161    1298

   max_power  seats
0         74    5.0
1      10352    5.0
2         78    5.0
3         90    5.0
4        882    5.0
```

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

[9]:
```
              year  selling_price     km_driven      mileage       engine  \
count  7906.000000   7.906000e+03  7.906000e+03  7906.000000  7906.000000
mean   2013.983936   6.498137e+05  6.918866e+04   947.702378  1458.708829
std       3.863695   8.135827e+05  5.679230e+04   925.336832   503.893057
min    1994.000000   2.999900e+04  1.000000e+00     0.000000   624.000000
25%    2012.000000   2.700000e+05  3.500000e+04   185.000000  1197.000000
50%    2015.000000   4.500000e+05  6.000000e+04   240.000000  1248.000000
75%    2017.000000   6.900000e+05  9.542500e+04  1944.000000  1582.000000
max    2020.000000   1.000000e+07  2.360457e+06  3344.000000  3604.000000

           max_power        seats
count    7906.000000  7906.000000
mean     2766.125348     5.416393
std      5162.123778     0.959208
min        35.000000     2.000000
25%       100.000000     5.000000
50%       739.000000     5.000000
75%      3748.000000     5.000000
max    108495.000000    14.000000
```

[10]:
```
data.fuel.unique()
```

[10]:
```
array(['Diesel', 'Petrol', 'LPG', 'CNG'], dtype=object)
```

[11]:
```
data_new = pd.get_dummies(data=data, columns=['fuel'], drop_first=True,␣
↪dtype=int)
```

[12]:
```
data_new.head()
```

[12]:
```
   year  selling_price  km_driven transmission  mileage  engine  max_power  \
0  2014         450000     145500       Manual      234    1248         74
1  2014         370000     120000       Manual     2114    1498      10352
2  2006         158000     140000       Manual      177    1497         78
3  2010         225000     127000       Manual      230    1396         90
4  2007         130000     120000       Manual      161    1298        882

   seats  fuel_Diesel  fuel_LPG  fuel_Petrol
0    5.0            1         0            0
1    5.0            1         0            0
2    5.0            0         0            1
3    5.0            1         0            0
4    5.0            0         0            1
```

[13]:
```
data_new["transmission"] = data_new["transmission"].replace({'Automatic': 1,␣
↪'Manual': 0})
```

[14]:
```
data_new.head()
```

[14]:
```
   year  selling_price  km_driven  transmission  mileage  engine  max_power  \
0  2014         450000     145500             0      234    1248         74
1  2014         370000     120000             0     2114    1498      10352
2  2006         158000     140000             0      177    1497         78
3  2010         225000     127000             0      230    1396         90
4  2007         130000     120000             0      161    1298        882

   seats  fuel_Diesel  fuel_LPG  fuel_Petrol
0    5.0            1         0            0
1    5.0            1         0            0
2    5.0            0         0            1
3    5.0            1         0            0
4    5.0            0         0            1
```

6 Scaling data

[15]:
```
mmScaler = MinMaxScaler()
mmScaler_y = MinMaxScaler()

label_enc = LabelEncoder()

x = data_new[['year', 'km_driven','transmission', 'mileage', 'engine',␣
↪'max_power', 'seats', 'fuel_Diesel', 'fuel_LPG', 'fuel_Petrol']].values
```

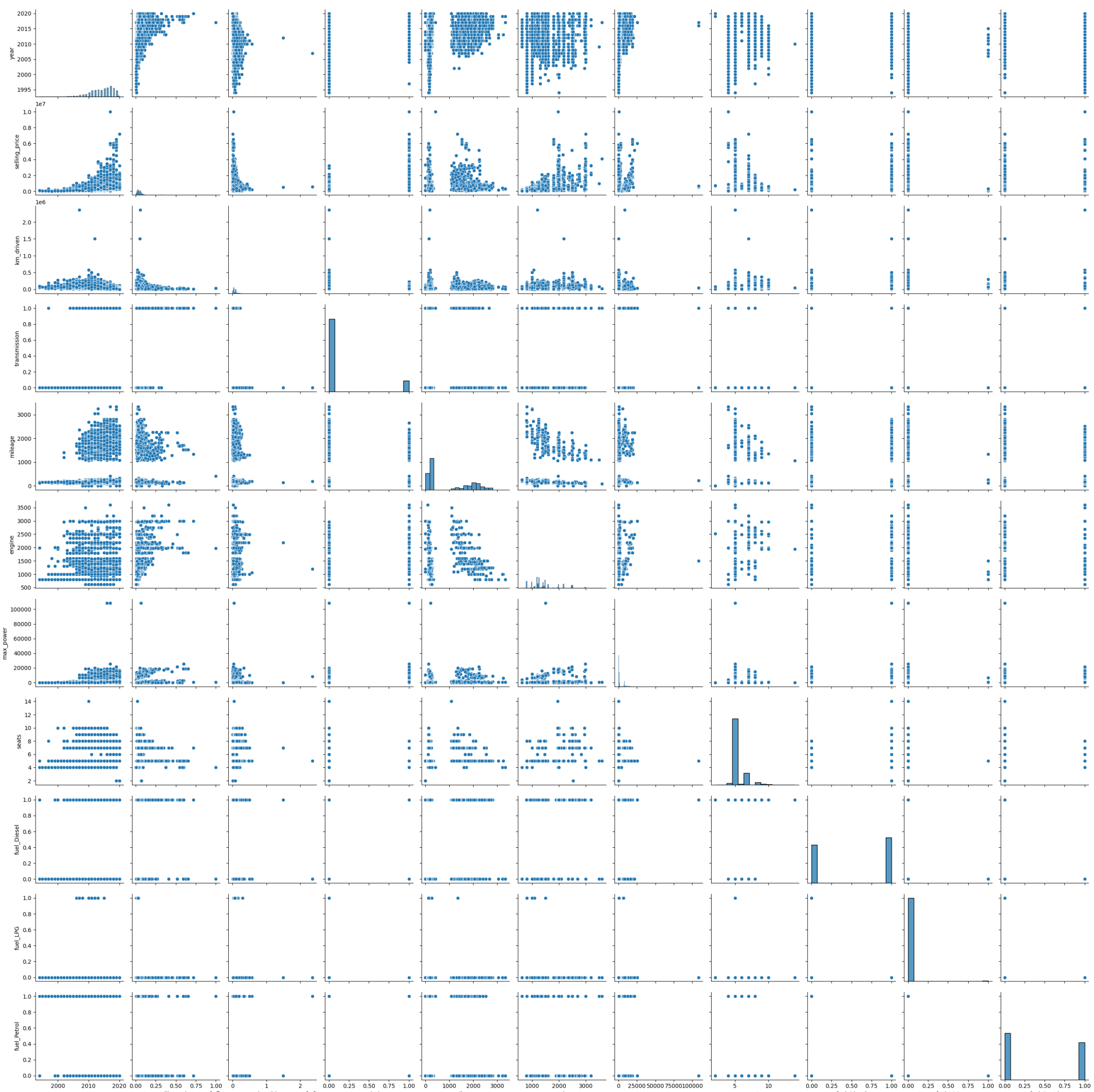
```
y = data_new[['selling_price']].values
```

[16]:
```
x[:, 0] = label_enc.fit_transform(x[:, 0])
x = mmScaler.fit_transform(x)
y = mmScaler_y.fit_transform(y)
```

7 Exploration Data Analysis And Correlation

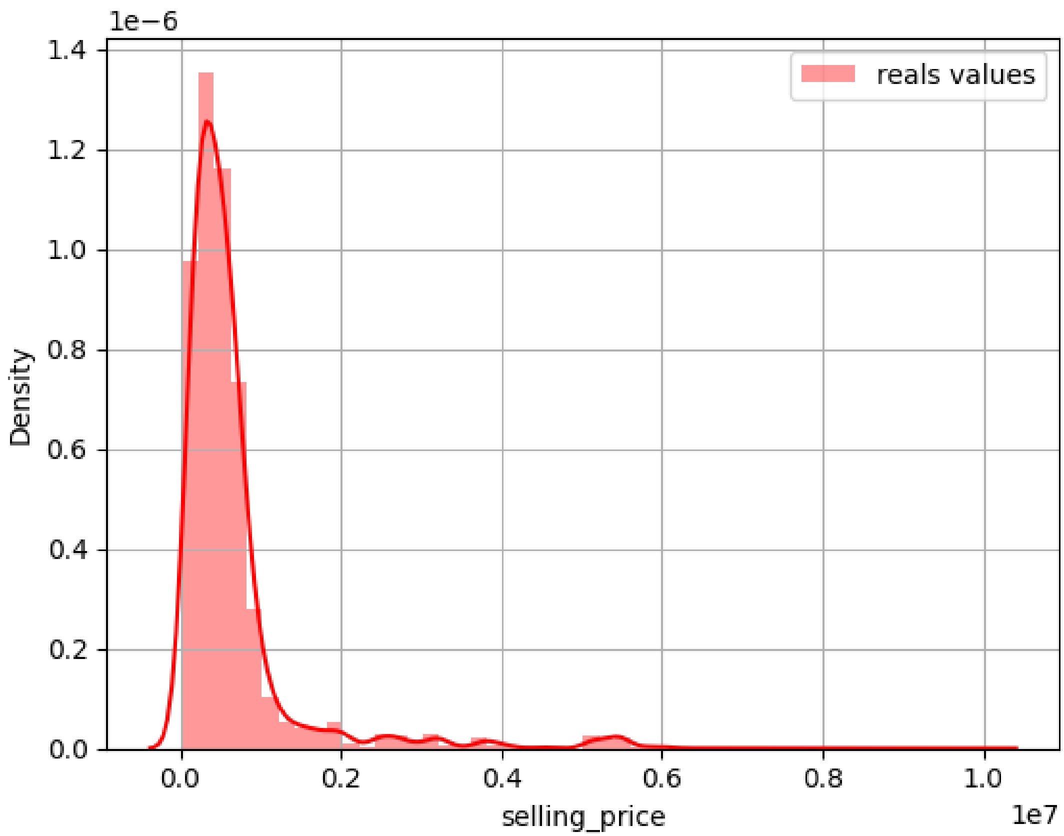
[17]:
```
#Pair plot labels
sns.pairplot(data_new)
```

[17]: <seaborn.axisgrid.PairGrid at 0x7f4458c5ba30>



[18]:

```
sns.distplot(data_new['selling_price'], label="reals values", color='red')
plt.legend()
plt.grid()
plt.show()
```



[19]:

```
#correlation
corr = pd.DataFrame(data_new.corrwith(data_new['selling_price']))
corr
```

[19]:

	0
year	0.412302
selling_price	1.000000
km_driven	-0.222158
transmission	0.590269
mileage	0.098988
engine	0.455682
max_power	0.137042
seats	0.041617
fuel_Diesel	0.204831
fuel_LPG	-0.035978

```
fuel_Petrol   -0.195074
```

8 Split Data

```
[20]: X_train, X_test, Y_train, Y_test = train_test_split(x, y, test_size=0.3,␣
        ↪random_state = 42)
      print("size train : ",X_train.shape)
      print("size test : ",X_test.shape)
```

```
size train :  (5534, 10)
size test :  (2372, 10)
```

9 Setting up models

```
[21]: regression = [
          Ridge(),
          KNeighborsRegressor(),
          LinearRegression(),
          RandomForestRegressor(),
          SVR(),
          DecisionTreeRegressor(),
          ElasticNet(),
          Lasso(),
          XGBRegressor()
      ]
```

```
[22]: head = 10
      for model in regression[:head]:
          start = time()
          model.fit(X_train, Y_train)
          train_time = time() - start
          start = time()
          Y_pred = model.predict(X_test)
          predict_time = time()-start
          print(model)
          print("\t Temps d'entrainement : %0.3fs" % train_time)
          print("\t Temps de prédiction : %0.3fs" % predict_time)
          print("\t MAE score :", mean_absolute_error(Y_test, Y_pred))
          print("\t R2 score :", r2_score(Y_test, Y_pred))
          print("\t Max_error : ", max_error(Y_test, Y_pred))
          print("\t MSE score : ", mean_squared_error(Y_test, Y_pred))
          print()
```

```
Ridge()
         Temps d'entrainement : 0.014s
         Temps de prédiction : 0.001s
         MAE score : 0.02996937997545131
```

```
        R2 score : 0.563275406429672
        Max_error :  0.4539752867762097
        MSE score :  0.0029490586316302625

KNeighborsRegressor()
        Temps d'entrainement : 0.015s
        Temps de prédiction : 0.114s
        MAE score : 0.009781175000509661
        R2 score : 0.9246806113199936
        Max_error :  0.27733196817131717
        MSE score :  0.0005086072471897975

LinearRegression()
        Temps d'entrainement : 0.018s
        Temps de prédiction : 0.003s
        MAE score : 0.03028970476108429
        R2 score : 0.5578155727903901
        Max_error :  0.5171904628972888
        MSE score :  0.002985927106083595

RandomForestRegressor()
        Temps d'entrainement : 1.698s
        Temps de prédiction : 0.063s
        MAE score : 0.007393877374907403
        R2 score : 0.9686006258702937
        Max_error :  0.1779911285187745
        MSE score :  0.00021202972460969646

SVR()
        Temps d'entrainement : 0.062s
        Temps de prédiction : 0.015s
        MAE score : 0.028460252361910517
        R2 score : 0.7667076511856659
        Max_error :  0.3561446033383627
        MSE score :  0.0015753470839361351

DecisionTreeRegressor()
        Temps d'entrainement : 0.021s
        Temps de prédiction : 0.001s
        MAE score : 0.008135735594842439
        R2 score : 0.9587544824359456
        Max_error :  0.2156469191928867
        MSE score :  0.00027851751739908505

ElasticNet()
        Temps d'entrainement : 0.009s
        Temps de prédiction : 0.005s
        MAE score : 0.043042230420574736
```

```
        R2 score : -0.00010336126465948503
        Max_error :  0.5888351724741655
        MSE score :  0.006753371560663051

Lasso()
        Temps d'entrainement : 0.008s
        Temps de prédiction : 0.001s
        MAE score : 0.043042230420574736
        R2 score : -0.00010336126465948503
        Max_error :  0.5888351724741655
        MSE score :  0.006753371560663051

XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=None, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=None, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             multi_strategy=None, n_estimators=None, n_jobs=None,
             num_parallel_tree=None, random_state=None, …)
        Temps d'entrainement : 0.242s
        Temps de prédiction : 0.005s
        MAE score : 0.0069020062504309134
        R2 score : 0.9733403469987905
        Max_error :  0.17102317009828172
        MSE score :  0.00018002393489393432
```

10 THE WINNER IS XGBRegressor AND RandomForestRegressor

10.1 looking for the best parameters

10.2 RandomForestRegressor

[23]:
```
modelRFR= RandomForestRegressor()
modelRFR
```

[23]:
```
RandomForestRegressor()
```

[24]:
```
modelRFR.fit(X_train, Y_train)
```

[24]:
```
RandomForestRegressor()
```

[25]:
```
parameters = {'n_estimators': np.arange(1,30), 'criterion': ["squared_error",␣
↪"friedman_mse", "absolute_error", "poisson"],
              'max_features': ["sqrt", "log2", "None"], 'random_state': np.
↪arange(1,5)}
```

[26]:
```
kf = KFold(n_splits = 5, shuffle=True, random_state=5)
grid = GridSearchCV(modelRFR, parameters, cv=kf, verbose=1)
```

[27]:
```
grid.fit(X_train, Y_train)
```

```
Fitting 5 folds for each of 1392 candidates, totalling 6960 fits
```

[27]:
```
GridSearchCV(cv=KFold(n_splits=5, random_state=5, shuffle=True),
             estimator=RandomForestRegressor(),
             param_grid={'criterion': ['squared_error', 'friedman_mse',
                                       'absolute_error', 'poisson'],
                         'max_features': ['sqrt', 'log2', 'None'],
                         'n_estimators': array([ 1,  2,  3,  4,  5,  6,  7,  8,
 9, 10, 11, 12, 13, 14, 15, 16, 17,
       18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29]),
                         'random_state': array([1, 2, 3, 4])},
             verbose=1)
```

[28]:
```
grid.best_estimator_
```

[28]:
```
RandomForestRegressor(criterion='friedman_mse', max_features='sqrt',
                      n_estimators=26, random_state=4)
```

[29]:
```
grid.best_score_
```

[29]:
```
0.9438128032607228
```

[30]:
```
modelRFR = grid.best_estimator_
modelRFR
```

[30]:
```
RandomForestRegressor(criterion='friedman_mse', max_features='sqrt',
                      n_estimators=26, random_state=4)
```

[31]:
```
modelRFR.fit(X_train, Y_train)
```

[31]:
```
RandomForestRegressor(criterion='friedman_mse', max_features='sqrt',
                      n_estimators=26, random_state=4)
```

[32]:
```
modelRFR.score(X_train, Y_train)
```

[32]:
```
0.9904613453056764
```

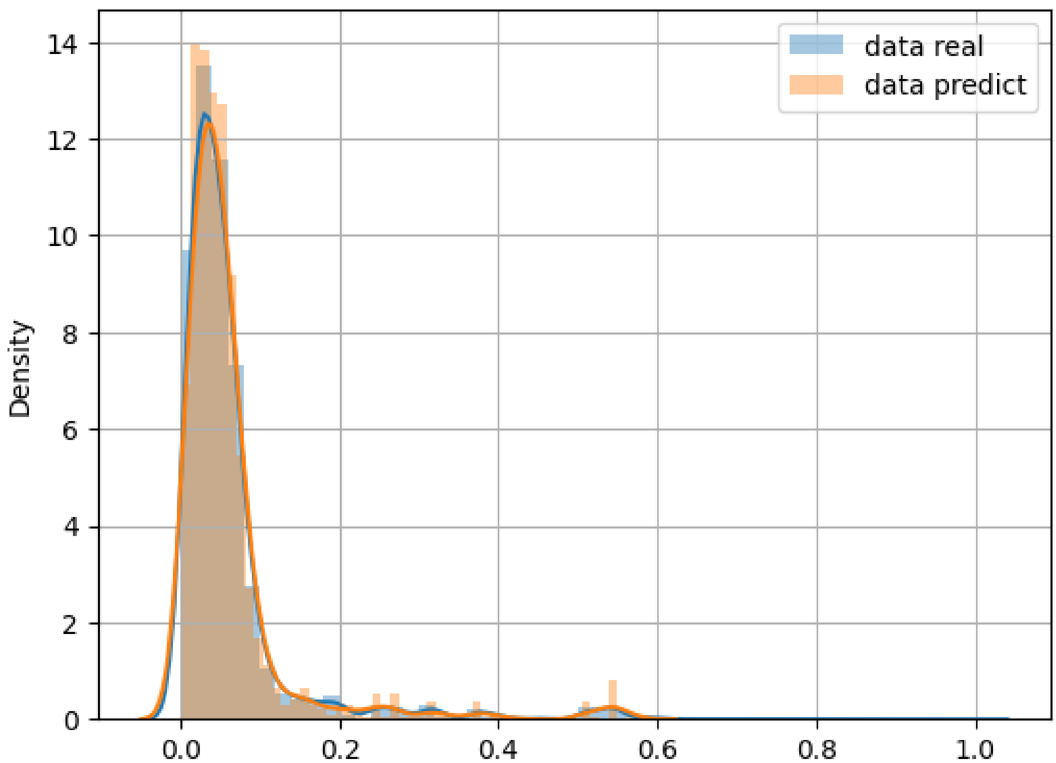
[33]:
```
modelRFR.score(X_test, Y_test)
```

[33]: 0.9707851741448748

[34]:
```python
Y_pred_RFR = modelRFR.predict(X_test)
print(Y_pred_RFR)
```

```
[0.05589856 0.0483477  0.01444343 … 0.05063276 0.08351529 0.07728772]
```

[35]:
```python
plt.grid(True)
sns.distplot(y, label='data real')
sns.distplot(Y_pred_RFR, label='data predict')
plt.legend()
plt.show()
```



10.3 XGBRegression

[36]:
```python
modelXG = XGBRegressor()
modelXG
```

[36]:
```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
```

```
                 gamma=None, grow_policy=None, importance_type=None,
                 interaction_constraints=None, learning_rate=None, max_bin=None,
                 max_cat_threshold=None, max_cat_to_onehot=None,
                 max_delta_step=None, max_depth=None, max_leaves=None,
                 min_child_weight=None, missing=nan, monotone_constraints=None,
                 multi_strategy=None, n_estimators=None, n_jobs=None,
                 num_parallel_tree=None, random_state=None, …)
```

[37]:
```
modelXG.fit(X_train, Y_train)
```

[37]:
```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=None, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=None, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             multi_strategy=None, n_estimators=None, n_jobs=None,
             num_parallel_tree=None, random_state=None, …)
```

[38]:
```
parameters = {'n_estimators': np.arange(1,50), 'max_depth': np.arange(1,5),
              'max_features': ["sqrt", "log2", "None"], 'random_state': np.
 ↪arange(1,5)}
```

[39]:
```
kf1 = KFold(n_splits = 5, shuffle=True, random_state=5)
grid1 = GridSearchCV(modelXG, parameters, cv=kf1, verbose=1)
```

[40]:
```
grid1.fit(X_train, Y_train)
```

```
Fitting 5 folds for each of 2352 candidates, totalling 11760 fits
```

[40]:
```
GridSearchCV(cv=KFold(n_splits=5, random_state=5, shuffle=True),
             estimator=XGBRegressor(base_score=None, booster=None,
                                    callbacks=None, colsample_bylevel=None,
                                    colsample_bynode=None,
                                    colsample_bytree=None, device=None,
                                    early_stopping_rounds=None,
                                    enable_categorical=False, eval_metric=None,
                                    feature_types=None, gamma=None,
                                    grow_policy=None, importance_type=None,
                                    inter…
                                    multi_strategy=None, n_estimators=None,
                                    n_jobs=None, num_parallel_tree=None,
                                    random_state=None, …),
             param_grid={'max_depth': array([1, 2, 3, 4]),
                         'max_features': ['sqrt', 'log2', 'None'],
```

```
                         'n_estimators': array([ 1,  2,  3,  4,  5,  6,  7,  8,
9, 10, 11, 12, 13, 14, 15, 16, 17,
       18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34,
       35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49]),
                         'random_state': array([1, 2, 3, 4])},
             verbose=1)
```

[41]:
```
modelXG = grid1.best_estimator_
modelXG
```

[41]:
```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=None, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=4, max_features='sqrt',
             max_leaves=None, min_child_weight=None, missing=nan,
             monotone_constraints=None, multi_strategy=None, n_estimators=49,
             n_jobs=None, num_parallel_tree=None, …)
```

[42]:
```
grid1.best_score_
```

[42]: 0.9325353333323065

[43]:
```
grid1.best_estimator_
```

[43]:
```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=None, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=4, max_features='sqrt',
             max_leaves=None, min_child_weight=None, missing=nan,
             monotone_constraints=None, multi_strategy=None, n_estimators=49,
             n_jobs=None, num_parallel_tree=None, …)
```

[44]:
```
modelXG.score(X_train, Y_train)
```

[44]: 0.9772679815268545

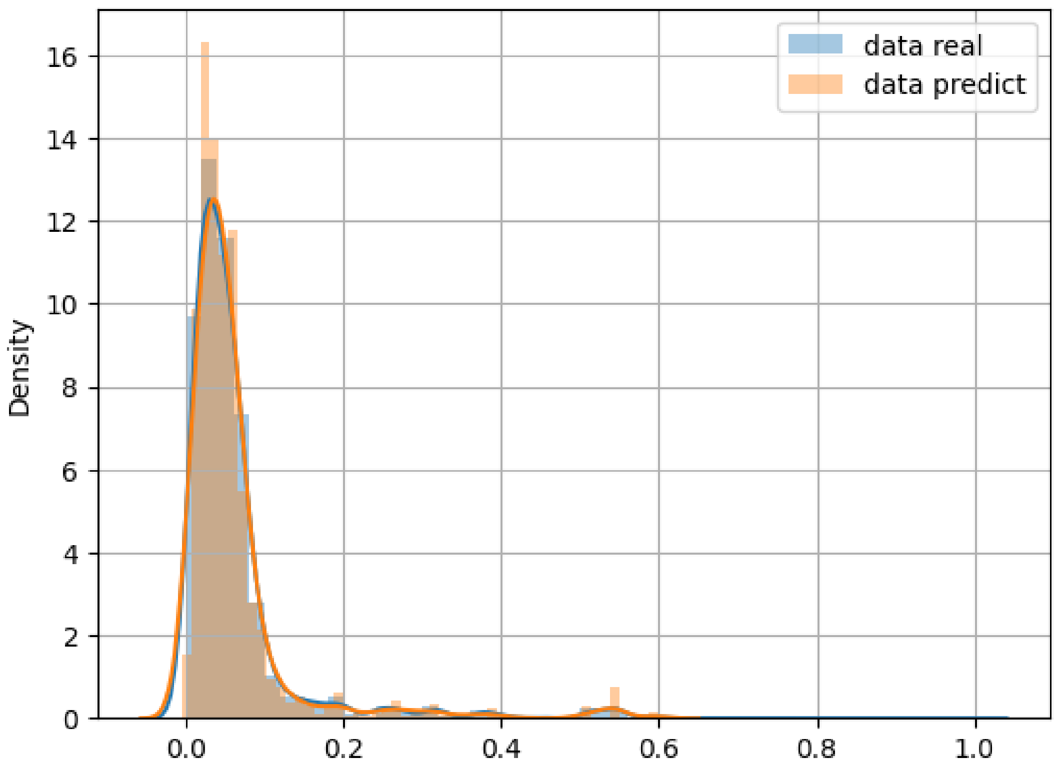
[45]:
```
modelXG.score(X_test, Y_test)
```

[45]: 0.9613768098316412

[46]:
```
Y_pred_XG = modelXG.predict(X_test)
print(Y_pred_XG)
```

```
[0.04059134 0.05420575 0.02151247 … 0.05664548 0.08507872 0.10265834]
```

[47]:
```
plt.grid(True)
sns.distplot(y, label='data real')
sns.distplot(Y_pred_XG, label='data predict')
plt.legend()
plt.show()
```



11 Conclusion

[48]:
```
finale_report = pd.DataFrame({
    'Model': ['RandomForestRegresion()', 'XGBRegression()'],
    'Score Train': [modelRFR.score(X_train, Y_train), modelXG.score(X_train,␣
 ↪Y_train)],
    'Score Test': [modelRFR.score(X_test, Y_test), modelXG.score(X_test,␣
 ↪Y_test)]
})
```

[49]:
```
print(finale_report)
```

	Model	Score Train	Score Test
0	RandomForestRegresion()	0.990461	0.970785
1	XGBRegression()	0.977268	0.961377

12 Conclusion

12.1 THE WINNER IS RANDOMFORESTREGRESSION