House Price Prediction

Dataset - 2

Description:

Accurately predicting house prices can be a daunting task. The buyers are just not concerned about the size(square feet) of the house and there are various other factors that play a key role to decide the price of a house/property. It can be extremely difficult to figure out the right set of attributes that are contributing to understanding the buyer's behavior as such. This dataset has been collected across various property aggregators across.

Date set:

We divided into 3 datasets each daatset conatining 7 cloumns and 1000 rows with csv extension. The data contains the following columns :

'Avg. Area Income' – Avg. The income of the householder of the city house is located. 'Avg. Area House Age' – Avg. Age of Houses in the same city. 'Avg. Area Number of Rooms' – Avg. Number of Rooms for Houses in the same city. 'Avg. Area Number of Bedrooms' – Avg. Number of Bedrooms for Houses in the same city. 'Area Population' – Population of the city. 'Price' – Price that the house sold at. 'Address' – Address of the houses.

visualization:

Pair plot: Pair plot is used to understand the best set of features to explain a relationship between two variables or to form the most separated clusters. Dist plot: Distplot is a combination of a histogram with a line on it. Heat map: A heat map represents these coefficients to visualize the strength of correlation among variables. Scatter plot: Scatter plots shows how much one variable is affected by another or the relationship between them with the help of dots in two dimensions.

Python packages:

Numpy: is a library used to add support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.

Pandas: it is used to work with datasets, for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series.

Seaborn: is a library that uses Matplotlib underneath to plot graphs. It will be used to visualize random distributions.

Matplotlib: is a comprehensive library for creating static, animated, and interactive visualizations in Python

Import Libraries

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

%matplotlib inline
```

Importing Data and Checking out.

In [2]:
```
HouseDF = pd.read_csv('USA_Housing_2.csv')
```

In [3]:
```
HouseDF.head()
```

Out[3]:

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price	Address
0	56423.03428	4.670847	8.109865	3.03	40155.74285	1.000217e+06	460 Morales Islands Apt. 118\nWest Jeffrey, NE...
1	66678.06217	3.907841	7.496089	5.02	23031.17032	5.870078e+05	827 Ferguson Isle\nRosebury, AL 61416-3167
2	71517.01424	7.905595	7.731386	5.02	40239.28257	1.734373e+06	653 Parker Overpass Suite 506\nSusanshire, AZ ...
3	66097.62203	3.979496	6.151771	4.19	37467.59631	7.621673e+05	41962 Castro Groves Suite 053\nJacquelinestad,...
4	76423.31757	7.561879	7.672937	5.25	24402.23731	1.532846e+06	66807 Johnson Prairie Apt. 849\nHernandezhaven...

In [4]:
```
HouseDF.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 7 columns):
 #   Column                       Non-Null Count  Dtype  
---  ------                       --------------  -----  
 0   Avg. Area Income             1000 non-null   float64
 1   Avg. Area House Age          1000 non-null   float64
 2   Avg. Area Number of Rooms    1000 non-null   float64
 3   Avg. Area Number of Bedrooms 1000 non-null   float64
 4   Area Population              1000 non-null   float64
 5   Price                        1000 non-null   float64
 6   Address                      1000 non-null   object 
dtypes: float64(6), object(1)
memory usage: 54.8+ KB
```

In [5]:
```
HouseDF.describe()
```

Out[5]:

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price
count	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1.000000e+03
mean	68494.781137	5.947322	6.997602	4.001170	36079.496617	1.223094e+06
std	10881.128884	1.031223	1.012059	1.232425	9672.583756	3.670742e+05
min	35963.330810	2.644304	3.950973	2.000000	3285.450538	3.114052e+04
25%	61307.895688	5.269245	6.289393	3.170000	29286.267058	9.741244e+05
50%	68566.053405	5.939188	7.002757	4.065000	36241.703370	1.231733e+06
75%	76019.250150	6.658052	7.721871	4.500000	42428.689090	1.474882e+06
max	104702.724300	8.991399	10.759588	6.500000	69592.040240	2.318286e+06

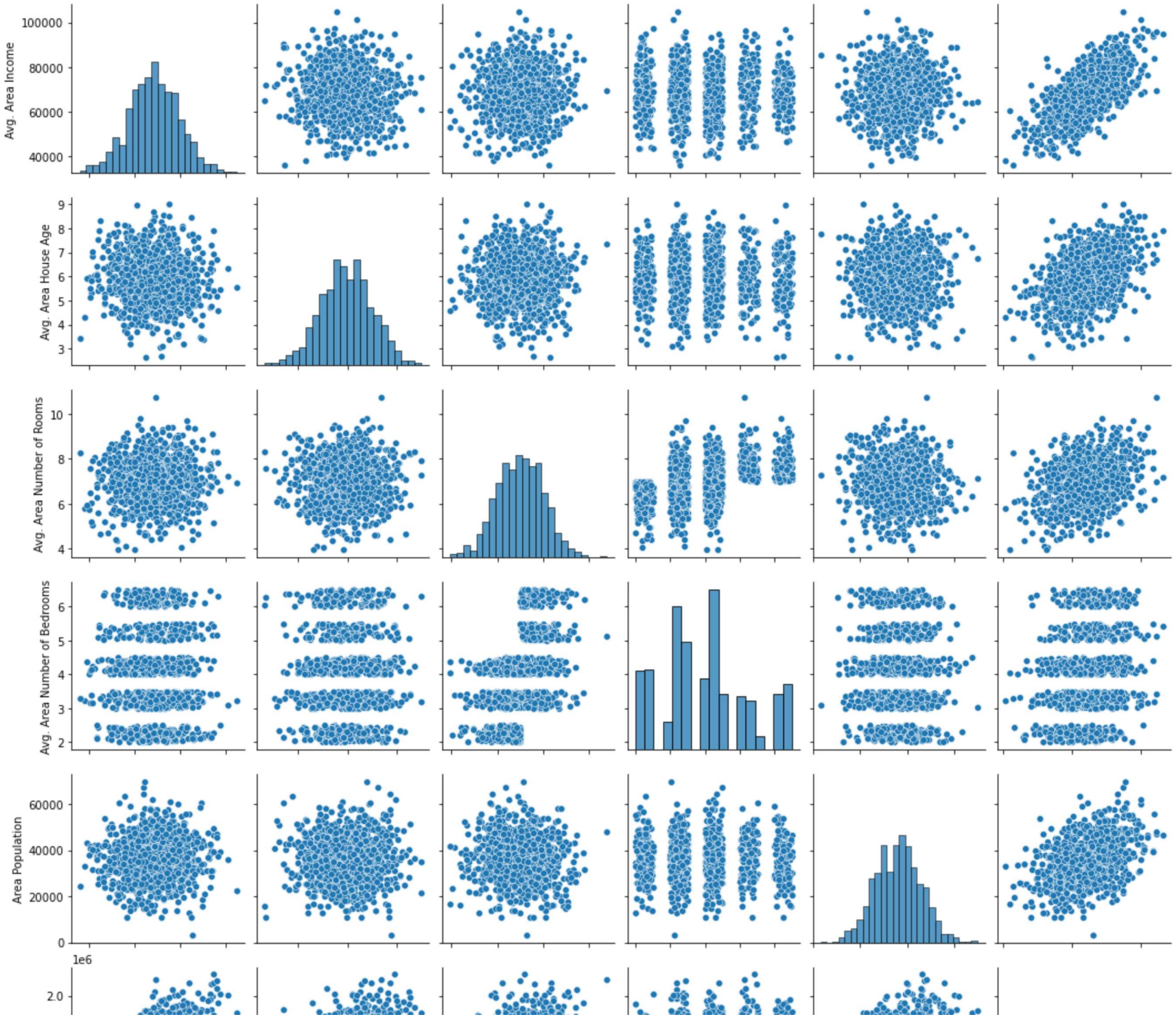
In [6]:
```
HouseDF.columns
```

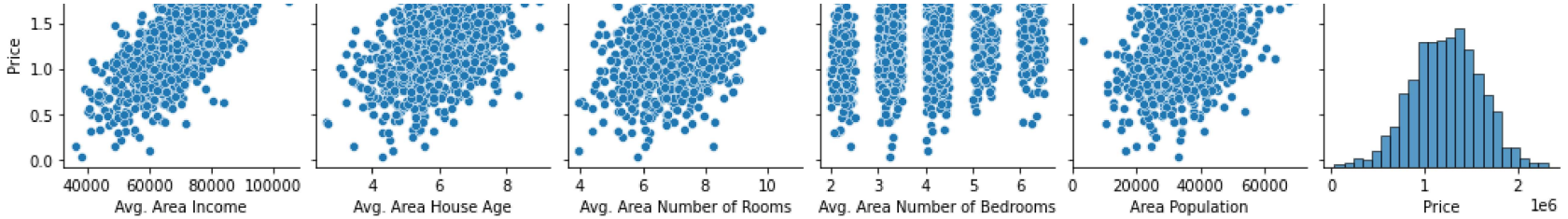
Out[6]:
```
Index(['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number of Rooms',
       'Avg. Area Number of Bedrooms', 'Area Population', 'Price', 'Address'],
      dtype='object')
```

Exploratory Data Analysis for House Price Prediction

In [7]:
```
sns.pairplot(HouseDF)
```

Out[7]:
```
<seaborn.axisgrid.PairGrid at 0x226829f0310>
```

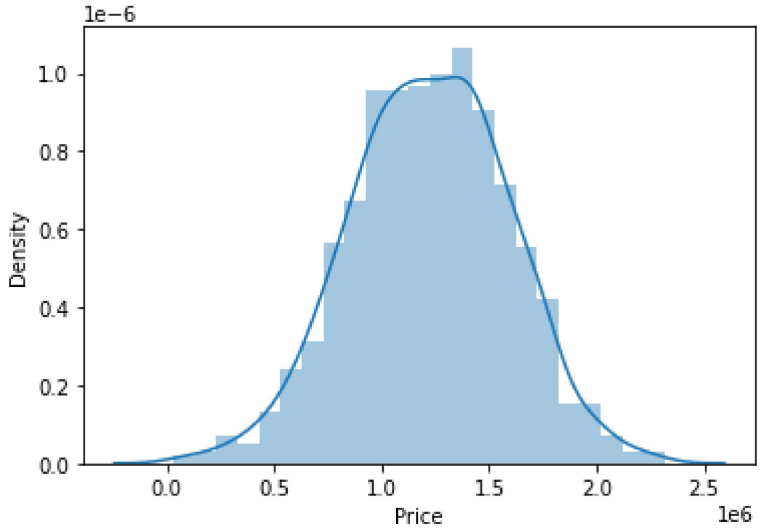

Avg. Area Income
Avg. Area House Age
Avg. Area Number of Rooms
Avg. Area Number of Bedrooms
Area Population
1e6



In [8]:
```
sns.distplot(HouseDF['Price'])
```

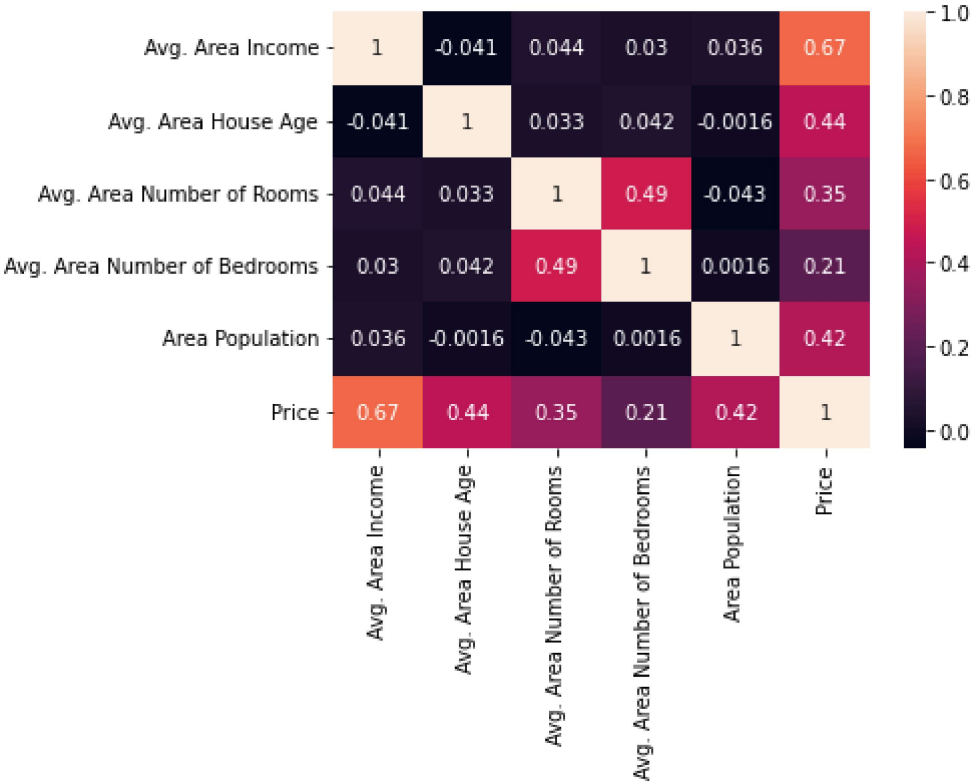
```
C:\Users\DELL\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[8]:
```
<AxesSubplot:xlabel='Price', ylabel='Density'>
```



In [9]:
```
sns.heatmap(HouseDF.corr(), annot=True)
```

Out[9]:
```
<AxesSubplot:>
```



Training a Linear Regression Model

X and y List

In [10]:
```
X = HouseDF[['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number of Rooms',
             'Avg. Area Number of Bedrooms', 'Area Population']]

y = HouseDF['Price']
```

Split Data into Train, Test

In [11]:
```
from sklearn.model_selection import train_test_split
```

In [12]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=101)
```

Creating and Training the LinearRegression Model

In [13]:
```
from sklearn.linear_model import LinearRegression
```

In [14]:
```
lm = LinearRegression()
```

In [15]:
```
lm.fit(X_train,y_train)
```

Out[15]:
```
LinearRegression()
```

LinearRegression Model Evaluation

In [16]:
```
print(lm.intercept_)
```

```
-2644042.5733898
```

In [17]:
```
coeff_df = pd.DataFrame(lm.coef_,X.columns,columns=['Coefficient'])
coeff_df
```

Out[17]:

	Coefficient
Avg. Area Income	22.128306
Avg. Area House Age	161073.104088
Avg. Area Number of Rooms	116647.545539
Avg. Area Number of Bedrooms	2366.880141
Area Population	15.629058

Predictions from our Linear Regression Model

In [18]:
```
predictions = lm.predict(X_test)
```

In [19]:
```
#print(predictions)

from sklearn.metrics import accuracy_score
```

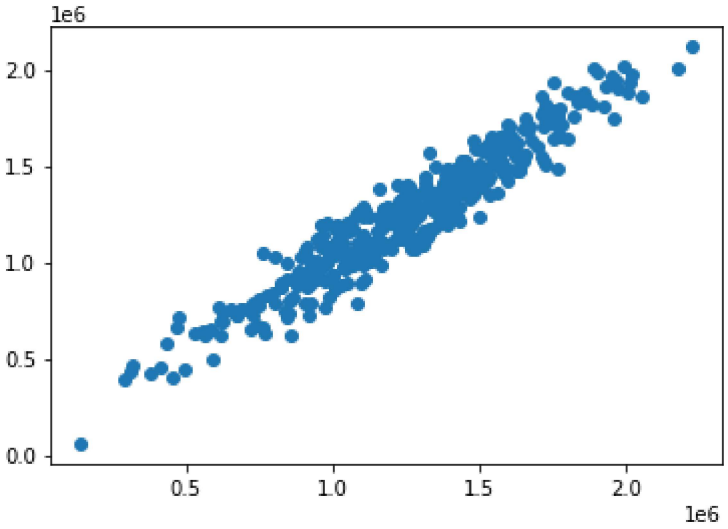
```
#print(accuracy_score(y_test,predictions))
```

In [20]:
```
lm.score(X_test, y_test)
```

Out[20]:
```
0.919702712472234
```

In [21]:
```
plt.scatter(y_test,predictions)
```

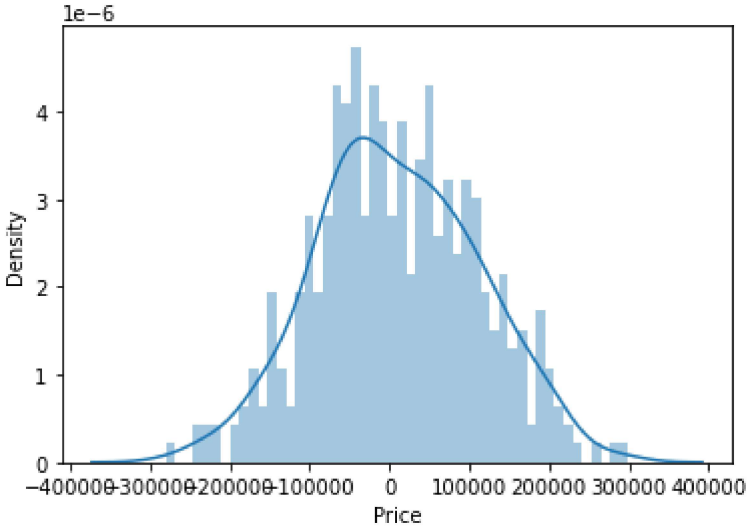
Out[21]:
```
<matplotlib.collections.PathCollection at 0x2268640ba30>
```



In the above scatter plot, we see data is in line shape, which means our model has done good predictions.

In [22]:
```
sns.distplot((y_test-predictions),bins=50);
```

```
C:\Users\DELL\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



In the above histogram plot, we see data is in bell shape (Normally Distributed), which means our model has done good predictions.

Regression Evaluation Metrics

In [23]:
```
from sklearn import metrics
```

In [24]:
```
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: 82858.209065317
MSE: 10476717306.07957
RMSE: 102355.83669766747
```

SVR

In [25]:
```
from sklearn.svm import SVR
svr = SVR(C=100000)
svr.fit(X_train, y_train)
predictions = svr.predict(X_test)
```

In [26]:
```
svr.score(X_test, y_test)
```

Out[26]:
```
0.5631853537104841
```

KNN

In [27]:
```
from sklearn.neighbors import KNeighborsRegressor

knn_model=KNeighborsRegressor()
knn_model.fit(X_train, y_train)
knn_model.score(X_test,y_test)
```

Out[27]:
```
0.466060360408757
```

Desicion tree

In [28]:
```
from sklearn.tree import DecisionTreeRegressor

dtr_model=DecisionTreeRegressor(max_depth=5)
dtr_model.fit(X_train, y_train)
dtr_model.score(X_test,y_test)
```

Out[28]:
```
0.6137625853902076
```

In [ ]:
```

```