

HOUSE PRICE PREDICTION USING ML

Phase 3- Development phase

INTRODUCTION



House price prediction can help the developer determine the selling price of a house and can help the customer to arrange the right time to purchase a house. There are three factors that influence the price of a house which include physical conditions, concept and location.

The trend of the sudden drop or constant rising of housing prices has attracted interest from the researcher as well as many other interested people. There have been various research works that use different methods and techniques to address the question of the changing of house prices. This work considers the issue of changing house price as a classification problem and discuss machine learning techniques to predict whether house prices will rise or fall using available data. This work applies various feature selection techniques such as variance influence factor, Information value, principle component analysis, and data transformation techniques such as outlier and missing value treatment as well as different transformation techniques. The performance of the machine learning techniques is measured by the four parameters of accuracy, precision, specificity, and sensitivity. The work considers two discrete values 0 and 1 as respective classes. If the value of the class is 0 then we consider that the price of the house has decreased and if the value of the class is 1 then we consider that the price of the house has increased.

Given data set

<https://www.kaggle.com/datasets/vedavyasv/usa-housing>

Load the dataset by importing the libraries

Step 1 import necessary standard libraries

We begin by importing the libraries we are going to use:

- ❖ numpy
- ❖ pandas
- ❖ matplotlib
- ❖ Seaborn

1.Import the libraries

```
import pandas as pd  
import numpy as np  
Import matplotlib as mpl  
Import matplotlib. Pyplot as plt  
%matplotlib inline  
from sklearn.model_selection import train_test_split  
from sklearn.preprocessing import StandardScaler
```

2.load data set

Usually in python we use read CSV to load the given data sets in jupyter notebook

```
Data=pd.read_csv('data.csv')
```

Data preprocessing :

Data preprocessing is the concept of changing the raw data into a clean data set. The dataset is preprocessed in order to check missing values, noisy data, and other inconsistencies before executing it to the algorithm.

Data preprocessing includes;

- ✓ Data cleaning
- ✓ Data integration
- ✓ Data transformation
- ✓ Data reduction



Data cleaning:

This involves identifying and correcting errors or inconsistencies in the data, such as missing values, outliers, and duplicates. Various techniques can be used for data cleaning, such as imputation, removal, and transformation.

```
data=pd.read_csv('usa housing.csv')
```

```
data.dropna()
```

- Remove rows

```
data.fillna(value)
```

- Replace missing value with our specific value

`Data.drop_duplicate()`

- Remove duplicate data

`Data['column_name'].replace(old_value, new_value, inplace=True)`

- Replace the value of old value which is no more needed, to a value with newly updated

Data integration :

This involves combining data from multiple sources to create a unified dataset. Data integration can be challenging as it requires handling data with different formats, structures, and semantics. Techniques such as record linkage and data fusion can be used for data integration.

- ❖ We can merge multiple pandas DataFrames using the merge function.

`Merge()`

```
df1 = pd.read_csv(dataset1, header = 0)
```

```
df2 = pd.read_csv(dataset2, header = 0)
```

```
df1.head() #to print first any no of rows in 1st dataframe
```

```
df2.head() #to print first any no of rows in 2nd dataframe
```

If any of the attributes have common value then use merge function

`pd.merge()`

```
df = pd.merge(df1, df2, on = 'common attribute ')
```

```
df.head(10)
```

Data Transformation :

This involves converting the data into a suitable format for analysis. Common techniques used in data transformation include normalization, standardization, and discretization. Normalization is used to scale the data to a common range, while standardization is used to transform the data to have zero mean and unit variance. Discretization is used to convert continuous data into discrete categories.

The Python Pandas library contains a large variety of functions for manipulating data, including tools to accomplish all three types of transformations.

- a) Manipulate the form of data
- b) Engineer the features of data
- c) Transform data values

Manipulate the form of data:

Changing the form of data is one of the most common use cases when transforming data using Python. Often raw data needs to be formed, reshaped, or cleaned up to be used by a data analyst or data scientist.

Sort and filter:

`sort_values`, `query`, and `filter`.

```
#Sort by name
```

```
sorted = df.sort_values(by=['name'])  
display(sorted)
```

```
#Filter rows  
just_students = df.query('is_student==True')  
display(just_students)
```

#necessary row attributes

```
#Filter columns  
no_birthday = df.filter(['name', 'is_student', 'target'])  
display(no_birthday)
```

#necessary column attributes

Renaming columns:

Sometimes a raw dataset will have column names that make sense to the original owner of the data but may not make sense in a broader analytic sense.

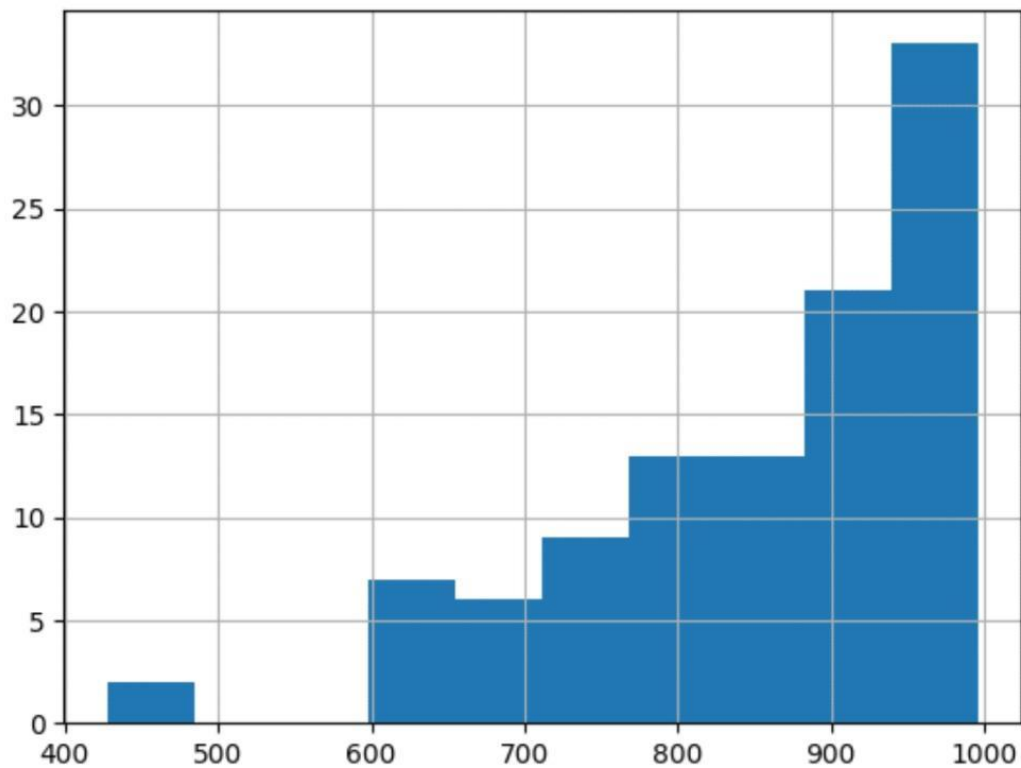
```
#Rename column  
renamed = df.rename(columns={'target': 'target_score'})  
display(renamed)
```

Splitting and combining:

Often when working with text columns, data needs to be split and/or combined in various ways.

```
#Splitting
splitnames = df.copy()
split = splitnames['name'].str.split(' ', expand = True)
splitnames['first'] = split[0]
splitnames['last'] = split[1]
display(splitnames)
```

```
#Data Value transforms
df['target'].hist()
```

Why data preprocessing is important

It improves accuracy and reliability.

Preprocessing data removes missing or inconsistent data values resulting from human or computer error, which can improve the accuracy and quality of a dataset, making it more reliable.

```
import pandas as pd
```

```
import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import r2_score,
mean_absolute_error, mean_squared_error

from sklearn.linear_model import LinearRegression

from sklearn.linear_model import Lasso

from sklearn.ensemble import RandomForestRegressor

from sklearn.svm import SVR

import xgboost as xg

%matplotlib inline

import warnings

Warnings.filterwarnings("ignore")
```

```
Sns.histplot(dataset, x='Price', bins=50, color='y')
```

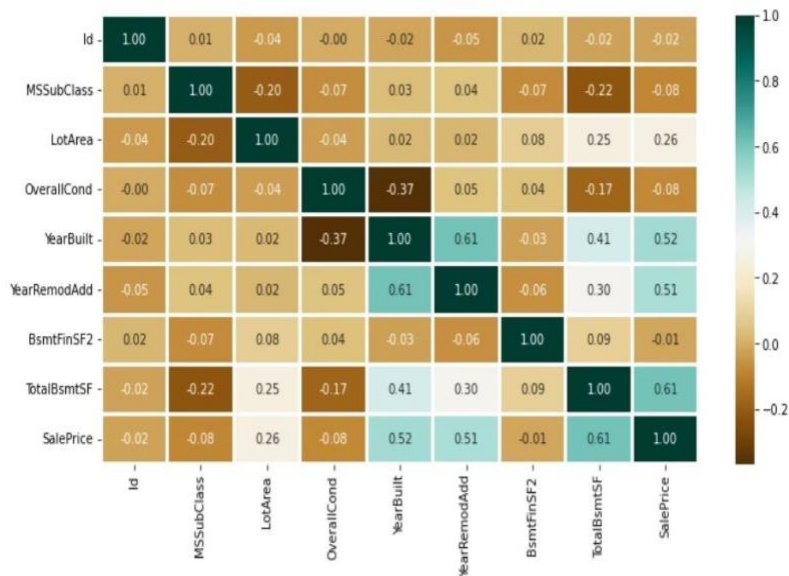
Out:

```
<Axes: xlabel='Price', ylabel='Count'>
```

```
plt.figure(figsize=(12, 6))
```

```
Sns.heatmap(dataset.corr(), Cmap = 'BrBG' Fmt = '.2f'
```

```
Linewidths = 2, Annot = True)
```



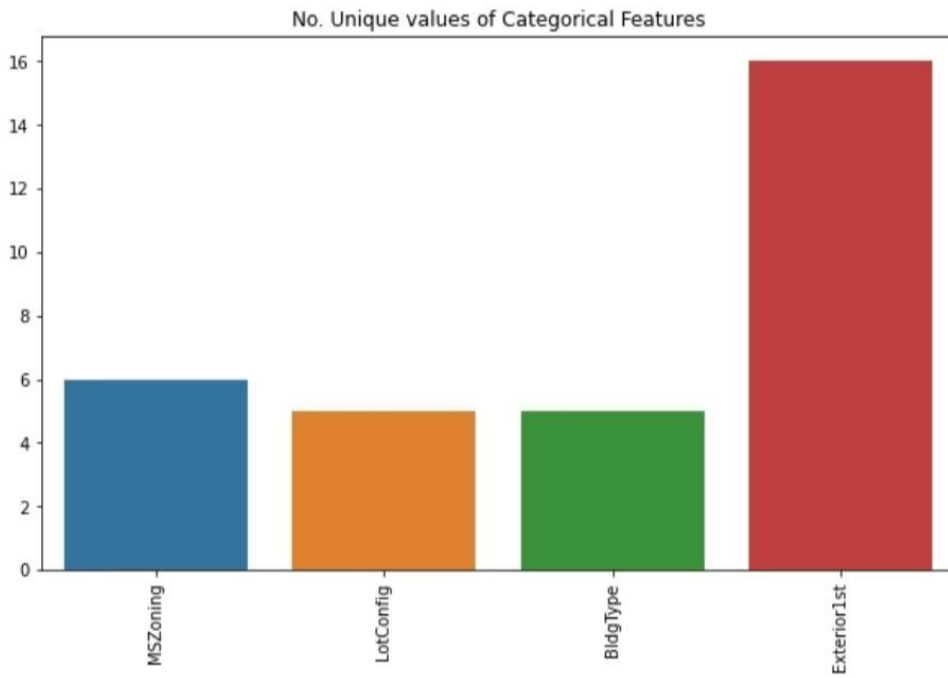
```
unique_values = []
```

```
for col in object_cols:
```

```
    unique_values.append(dataset[col].unique().size)
```

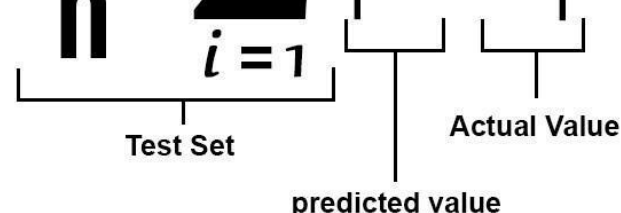
```
plt.figure(figsize=(10,6))
```

```
plt.title('No. Unique values of Categorical Features')
```



Mean absolute percentage Error:

$$\text{MAE} = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i|$$



```
from sklearn.linear_model import LinearRegression
model_LR = LinearRegression()
model_LR.fit(X_train, Y_train)
y_pred = model_LR.predict(X_valid)
print(mean_absolute_percentage_error(Y_valid, Y_pred))
```

Conclusion:

- Thus, the machine learning model using linear regression algorithm is very helpful in predicting the house prices for real estate customers. Here we have used a supervised learning approach in machine learning field which will yield us a best possible result.

- The machine learning model is given the test data but without the price of the properties in order to predict the price for them given the various features for the properties. The predicted price is then compared to the actual price in the test data.

