House-Price- Prediction Using Machine Learning



**Abstract :**

Predicting house prices is a crucial task in the real estate industry, offering valuable insights for buyers, sellers, and investors. Machine learning techniques have become a powerful tool for this purpose. This abstract provides an overview of a typical approach to house price prediction using machine learning.

The process begins with data collection, where historical housing prices and relevant property features are gathered. Data preprocessing follows, involving cleaning, encoding , and normalization. Feature selection and engineering help in improving the model's accuracy. The dataset is then divided into training and testing sets, and an appropriate regression algorithm is selected for model training.

Evaluation metrics such as Mean Absolute Error (MAE), Mean Squared Error (MSE), and Root Mean Squared Error (RMSE) are employed to assess the model's performance. Hyperparameter tuning is conducted to optimize the model, and once satisfactory results are achieved, the model is deployed in a production environment.

Continual monitoring and maintenance are crucial, as house prices can change over time. Ethical considerations and compliance with regulations, if applicable, are essential throughout the process.This abstract highlights the key steps and considerations involved in house price prediction using machine learning, emphasizing the significance of data quality, model selection, and ongoing model management for accurate and reliable predictions in the dynamic real estate market.

**Introduction :**

The prediction of house prices is a fundamental and practical challenge within the real estate industry. Accurate price estimations are vital for buyers, sellers, and investors to make informed decisions. Traditionally, real estate professionals rely on market knowledge and expertise to gauge property values. However, in the era of big data and advanced technology, machine learning has emerged as a powerful tool for predicting house prices with a high degree of precision.

This introduction sets the stage for the discussion of house price prediction using machine learning. It outlines the significance of the task, the role of data, and the promise of machine learning in providing robust, data-driven insights.

**Data Processing :**

Data processing is a critical step in the house price prediction using machine learning. It involves several tasks aimed at preparing the raw data for model training and analysis. Here are the key data processing stepss.

**Data Collection :**

Gather a comprehensive dataset that includes historical house prices and relevant features. Sources may include real estate websites, government records, or data providers.

**Data Cleaning :**

Address missing values: Identify and handle missing data points in the dataset. Options include imputation, deletion, or using domain-specific knowledge to fill in missing values.

Handle outliers: Detect and manage outliers in the data. Outliers can significantly affect model performance, so you may choose to remove them or transform them.

**Feature Selection and Engineering :**

Feature selection: Identify and select the most relevant features that are likely to impact house prices. This step reduces dimensionality and enhances model efficiency.

Feature engineering: Create new features that may improve the model's performance. For example, you could calculate the total area of a property by combining the dimensions of its rooms.

**Data Transformation :**

Categorical data encoding: Convert categorical variables (e.g., property type, location) into a numerical format. Common techniques include one-hot encoding or label encoding. Scaling and normalization: Rescale numerical features to bring them to a common scale (e.g., between 0 and 1) to ensure that no feature dominates others during modeling.

Logarithmic transformation: Apply logarithmic transformations to features with skewed distributions to make them more normally distributed.

**Data Splitting :**

Divide the preprocessed dataset into training and testing sets. A common split ratio is 80% for training and 20% for testing. This separation ensures that the model is evaluated on unseen data to assess its generalization capability.

**Data Visualization** (Optional but helpful) :

Create visualizations such as histograms, scatter plots, or correlation matrices to gain insights into the data and relationships between variables. This step can be particularly valuable for understanding feature importance.

**Data Quality Check :**

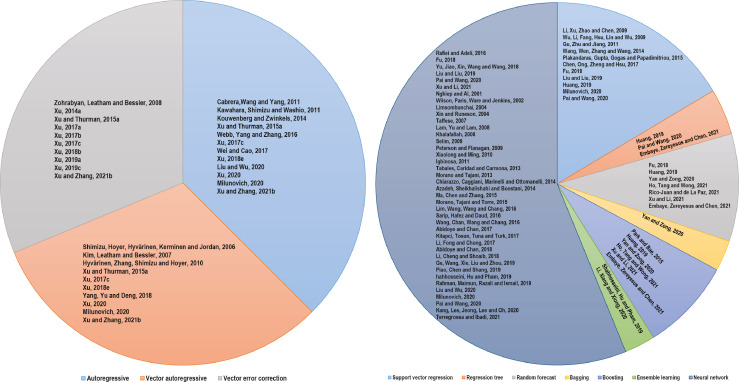
Verify that the preprocessing steps have effectively handled missing values, outliers, and transformed features. Ensure that the data is now in a suitable format for model training.

**Save Processed Data :**

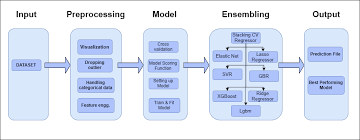
Save the preprocessed data to a separate file for easy access and reproducibility during model training and testing.

Effective data processing is essential for building an accurate house price prediction model. The quality of the data and the success of these preprocessing steps can significantly impact the model's performance. It is crucial to strike a balance between cleaning and transforming data while preserving valuable information to make informed predictions.

**Flow chart :-**

****





**PROGRAM:-**

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

In [2]:

df=pd.read\_csv("/kaggle/input/usa-housing/USA\_Housing.csv")

veri = df.copy()

veri.head()

Out[2]:

|  | Avg. Area Income | Avg. Area House Age | Avg. Area Number of Rooms | Avg. Area Number of Bedrooms | Area Population | Price | Address |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 79545.458574 | 5.682861 | 7.009188 | 4.09 | 23086.800503 | 1.059034e+06 | 208 Michael Ferry Apt. 674\nLaurabury, NE 3701... |
| 1 | 79248.642455 | 6.002900 | 6.730821 | 3.09 | 40173.072174 | 1.505891e+06 | 188 Johnson Views Suite 079\nLake Kathleen, CA... |
| 2 | 61287.067179 | 5.865890 | 8.512727 | 5.13 | 36882.159400 | 1.058988e+06 | 9127 Elizabeth Stravenue\nDanieltown, WI 06482... |
| 3 | 63345.240046 | 7.188236 | 5.586729 | 3.26 | 34310.242831 | 1.260617e+06 | USS Barnett\nFPO AP 44820 |
| 4 | 59982.197226 | 5.040555 | 7.839388 | 4.23 | 26354.109472 | 6.309435e+05 | USNS Raymond\nFPO AE 09386 |

In [3]:

veri.info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 5000 entries, 0 to 4999

Data columns (total 7 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 Avg. Area Income 5000 non-null float64

1 Avg. Area House Age 5000 non-null float64

2 Avg. Area Number of Rooms 5000 non-null float64

3 Avg. Area Number of Bedrooms 5000 non-null float64

4 Area Population 5000 non-null float64

5 Price 5000 non-null float64

6 Address 5000 non-null object

dtypes: float64(6), object(1)

memory usage: 273.6+ KB

In [4]:

veri= veri.drop(columns="Address",axis=1)

In [5]:

df.describe().T

Out[5]:

|  | count | mean | std | min | 25% | 50% | 75% | max |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Avg. Area Income | 5000.0 | 6.858311e+04 | 10657.991214 | 17796.631190 | 61480.562388 | 6.880429e+04 | 7.578334e+04 | 1.077017e+05 |
| Avg. Area House Age | 5000.0 | 5.977222e+00 | 0.991456 | 2.644304 | 5.322283 | 5.970429e+00 | 6.650808e+00 | 9.519088e+00 |
| Avg. Area Number of Rooms | 5000.0 | 6.987792e+00 | 1.005833 | 3.236194 | 6.299250 | 7.002902e+00 | 7.665871e+00 | 1.075959e+01 |
| Avg. Area Number of Bedrooms | 5000.0 | 3.981330e+00 | 1.234137 | 2.000000 | 3.140000 | 4.050000e+00 | 4.490000e+00 | 6.500000e+00 |
| Area Population | 5000.0 | 3.616352e+04 | 9925.650114 | 172.610686 | 29403.928702 | 3.619941e+04 | 4.286129e+04 | 6.962171e+04 |
| Price | 5000.0 | 1.232073e+06 | 353117.626581 | 15938.657923 | 997577.135049 | 1.232669e+06 | 1.471210e+06 | 2.469066e+06 |

In [6]:

sns.pairplot(veri)

/opt/conda/lib/python3.10/site-packages/seaborn/axisgrid.py:118: UserWarning: The figure layout has changed to tight

self.\_figure.tight\_layout(\*args, \*\*kwargs)

Out[6]:

<seaborn.axisgrid.PairGrid at 0x7819c8fe0d90>

In [7]:

sns.heatmap(veri.corr(),annot=True)

Out[7]:

<Axes: >

In [8]:

import statsmodels.api as sm

from statsmodels.stats.outliers\_influence import variance\_inflation\_factor

y = veri["Price"]

X= veri.drop(columns="Price",axis=1)

cons = sm.add\_constant(X)

vif= pd.DataFrame()

vif["variables"]=X.columns

vif["vif"]=[variance\_inflation\_factor(cons,i+1) for i **in** range(X.shape[1])]

vif

Out[8]:

|  | variables | vif |
| --- | --- | --- |
| 0 | Avg. Area Income | 1.001159 |
| 1 | Avg. Area House Age | 1.000577 |
| 2 | Avg. Area Number of Rooms | 1.273535 |
| 3 | Avg. Area Number of Bedrooms | 1.274413 |
| 4 | Area Population | 1.001266 |

In [9]:

from sklearn.model\_selection import train\_test\_split,cross\_val\_score

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y,

test\_size=0.2,

random\_state=42)

In [10]:

from sklearn.preprocessing import StandardScaler

ss = StandardScaler()

X\_train = ss.fit\_transform(X\_train)

X\_test = ss.transform(X\_test)

In [11]:

import sklearn.metrics as mt

def cross\_val(model):

vali=cross\_val\_score(model,X,y,cv=10)

return vali.mean()

def success(true\_,pred):

rmse=mt.mean\_absolute\_error(true\_,pred)

r2=mt.r2\_score(true\_,pred)

return[rmse,r2]

In [12]:

from sklearn.linear\_model import LinearRegression,Ridge,Lasso,ElasticNet

li\_model=LinearRegression()

li\_model.fit(X\_train,y\_train)

li\_pred = li\_model.predict(X\_test)

ridge\_model=Ridge(alpha=0.1)

ridge\_model.fit(X\_train,y\_train)

ridge\_pred = ridge\_model.predict(X\_test)

lasso\_model=Lasso(alpha=0.1)

lasso\_model.fit(X\_train,y\_train)

lasso\_pred = lasso\_model.predict(X\_test)

elas\_model=ElasticNet(alpha=0.1)

elas\_model.fit(X\_train,y\_train)

elas\_pred = elas\_model.predict(X\_test)

In [13]:

result=[["Linear model",success(y\_test,li\_pred)[0],success(y\_test,li\_pred)[1],cross\_val(li\_model)],

["Ridge model",success(y\_test,ridge\_pred)[0],success(y\_test,ridge\_pred)[1],cross\_val(ridge\_model)],

["Lasso model",success(y\_test,lasso\_pred)[0],success(y\_test,lasso\_pred)[1],cross\_val(lasso\_model)],

["ElasticNet model",success(y\_test,elas\_pred)[0],success(y\_test,elas\_pred)[1],cross\_val(elas\_model)]

]

pd.options.display.float\_format="**{:.4f}**".format

result=pd.DataFrame(result,columns=["Model","RMSE","R2","Verification"])

result

Out[13]:

|  | Model | RMSE | R2 | Verification |
| --- | --- | --- | --- | --- |
| 0 | Linear model | 80879.0972 | 0.9180 | 0.9174 |
| 1 | Ridge model | 80878.9638 | 0.9180 | 0.9174 |
| 2 | Lasso model | 80879.0910 | 0.9180 | 0.9174 |
| 3 | ElasticNet model | 81617.9048 | 0.9157 | 0.9165 |