## PREDICTION OF HOUSE PRICE USING MACHINE LEARNING

PHASE 1: PROBLEM DEFINITION AND DESIGN THINKING

**INTRODUCTION:**

Predicting house prices using machine learning involves creating a model that can estimate **t**he selling price of a house based on various features or attributes on various features or attributes. Here’s a breakdown of the problem definition and the machine learning for this task.

**PROBLEM DEFINITION**

The goal is to predict the deal price of the houses using various machine learning algorithms. Housing sales prices are determined by numerous factors such as area of the property, location of the house, material used for construction, age of the property, number of bedrooms and garages and so on. This uses machine learning algorithm algorithms to build prediction models for houses. Here machine learning algorithms such as logistic regression and support vector regression, Lasso Regression technique and Decision Tree are employed to build a predictive model.

**DESIGN THINKING**:

**MACHINE LEARNING PROCESS:**

**1.Data Collection:** Gather a dataset that includes both the target variable (house prices) and the features. This dataset should be comprehensive and clean. The given dataset link is <https://www.kaggle.com/datasets/vedavyasv/usa-housing>. The dataset includes the following features-Area income, Area house age, Area number of rooms, Area number of bedrooms, Area population, Price, Address.

**2.Data Preprocessing:** Clean and preprocess the data by handling missing values, encoding categorical variables and scaling or normalizing numerical features.

**3.Feature Engineering;** Create new features or transform existing ones to improve the model’s performance. For example, you can calculate the price per square foot, create dummy variables for categorical features, or extract meaning information from text descriptions.

**4. Splitting the data:** Divide the data set into two parts: a training set and the testing test. The training set is used to train the machine learning model, while the testing set is used to evaluate its performance.

**5. Model Selection:** Choose an appropriate machine algorithm for regression task. Common choices include linear regression tasks. Common choices include linear regression, decision trees, random forest, gradient boosting and neural network. experiment with different algorithms Experiment with different algorithms to find one that performs best for your specific data set.

**6.Model Training:** Train the selected model on the training data. The Model learns the relationship between the input features and the target variable during this step.

**7. Model Evaluation:** Evaluate the model’s performance using metrics such as Mean Absolute Error, Mean Squared Error, and Root Squared Error on the testing data. This step helps you assess how well your model generalizes to unseen data.

**8.Hyperparameter Tuning:** Fine- tune the model by adjusting hyperparameters (e.g., learning rate, tree depth) to improve its performance.

**9.Model Deployment:** Once you are satisfied with the model’s performance. Deploy it in a real-world environment where it can make predictions on new, unseen data.

**10.Monitoring and Maintenance:** Continuously monitor the model’s performance and update it as needed. House prices can change over time, so your model may require periodic retraining.

**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.

**Data Source**

A good data source for house price prediction using machine learning should be Accurate, Complete, Covering the geographic area of interest, Accessible.

**Dataset Link: (https://www.kaggle.com/datasets/vedavyasv/usa-housing)**

**Data Collection and Preprocessing:**

Importing the dataset: Obtain a comprehensive dataset containing relevant features such as square footage, number of bedrooms, location, amenities, etc.  Data preprocessing: Clean the data by handling missing values, outliers, and categorical variables. Standardize or normalize numerical features.

**Exploratory Data Analysis (EDA):**

Visualize and analyze the dataset to gain insights into the relationships between variables.  Identify correlations and patterns that can inform feature selection and engineering.  Present various data visualizations to gain insights into the dataset.  Explore correlations between features and the target variable (house prices).  Discuss any significant findings from the EDA phase that inform feature selection.

**Feature Engineering:**

Create new features or transform existing ones to capture valuable information.  Utilize domain knowledge to engineer features that may impact house prices, such as proximity to schools, transportation, or crime rates.  Explain the process of creating new features or transforming existing ones.  Showcase domain-specific feature engineering, such as proximity scores or composite indicators.  Emphasize the impact of engineered features on model performance.

**Advanced Regression Techniques:**

Ridge Regression: Introduce L2 regularization to mitigate multicollinearity and overfitting.

Lasso Regression: Employ L1 regularization to perform feature selection and simplify the model.

ElasticNet Regression: Combine both L1 and L2 regularization to benefit from their respective advantages.

Random Forest Regression: Implement an ensemble technique to handle nonlinearity and capture complex relationships in the data.

Gradient Boosting Regressors (e.g., XGBoost, LightGBM): Utilize gradient boosting algorithms for improved accuracy.

**Model Evaluation and Selection:**

Split the dataset into training and testing sets.  Evaluate models using appropriate metrics (e.g., Mean Absolute Error, Mean Squared Error, R-squared) to assess their performance.  Use cross-validation techniques to tune hyperparameters and ensure model stability.  Compare the results with traditional linear regression models to highlight improvements.  Select the best-performing model for further analysis**.**

**Model Interpretability:**

 Explain how to interpret feature importance from Gradient Boosting and XGBoost models.  Discuss the insights gained from feature importance analysis and their relevance to house price prediction.  Interpret feature importance from ensemble models like Random Forest and Gradient Boosting to understand the factors influencing house prices**.**

**Deployment and Prediction:**

Deploy the chosen regression model to predict house prices.  Develop a user-friendly interface for users to input property features and receive price

**Program:**

House Price Prediction

Importing Dependencies

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")

/opt/conda/lib/python3.10/site-packages/scipy/\_init\_.py:146: UserWarning: A NumPy version >=1.16.5 and <1.23.0 is required for this version of SciPy (detected version 1.23.5

warnings.warn(f"A NumPy version >={np\_minversion} and <{np\_maxversion}"

Loading Dataset

dataset = pd.read\_csv('E:/USA\_Housing.csv')

**Model 1 - Linear Regression**

In [1]:

model\_lr=LinearRegression()

In [2]:

model\_lr.fit(X\_train\_scal, Y\_train)

Out[2]:



**Predicting Prices**

In [3]:

Prediction1 = model\_lr.predict(X\_test\_scal)

**Evaluation of Predicted Data**

In [4]:

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

plt.plot(np.arange(len(Y\_test)), Y\_test, label='Actual Trend')

plt.plot(np.arange(len(Y\_test)), Prediction1, label='Predicted Trend')

plt.xlabel('Data')

plt.ylabel('Trend')

plt.legend()

plt.title('Actual vs Predicted')

Out[4]:

Text(0.5, 1.0, 'Actual vs Predicted')

In [5]:

sns.histplot((Y\_test-Prediction1), bins=50)

Out[5]:

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

**In [6]:**

print(r2\_score(Y\_test, Prediction1))

print(mean\_absolute\_error(Y\_test, Prediction1))

print(mean\_squared\_error(Y\_test, Prediction1))

**Out[6]:**

0.9182928179392918

82295.49779231755

10469084772.975954

**Model 2 - Support Vector Regressor**

In [7]:

model\_svr = SVR()

In [8]:

model\_svr.fit(X\_train\_scal, Y\_train)

Out[8]:



**Predicting Prices**

In [9]:

Prediction2 = model\_svr.predict(X\_test\_scal)

**Evaluation of Predicted Data**

**In [10]:**

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

plt.plot(np.arange(len(Y\_test)), Y\_test, label='Actual Trend')

plt.plot(np.arange(len(Y\_test)), Prediction2, label='Predicted Trend')

plt.xlabel('Data')

plt.ylabel('Trend')

plt.legend()

plt.title('Actual vs Predicted')

**Out[10]:**

Text(0.5, 1.0, 'Actual vs Predicted')

**In [11]:**

sns.histplot((Y\_test-Prediction2), bins=50)

**Out[12]:**

<Axes: xlabel='Price', ylabel='Count' price

**In [12]:**

print(r2\_score(Y\_test, Prediction2))

print(mean\_absolute\_error(Y\_test, Prediction2))

print(mean\_squared\_error(Y\_test, Prediction2))

-0.0006222175925689744

286137.81086908665

128209033251.4034

**Model 3 - Lasso Regression**

**In [13]:**

model\_lar = Lasso(alpha=1)

**In [14]:**

model\_lar.fit(X\_train\_scal,Y\_train)

**Out[14]:**

|  |
| --- |
| Lasso |
| Lasso(alpha=1) |
|  |

**Predicting Prices**

In [15]:

Prediction3 = model\_lar.predict(X\_test\_scal)

**Evaluation of Predicted Data**

**In [16]:**

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

plt.plot(np.arange(len(Y\_test)), Y\_test, label='Actual Trend')

plt.plot(np.arange(len(Y\_test)), Prediction3, label='Predicted Trend')

plt.xlabel('Data')

plt.ylabel('Trend')

plt.legend()

plt.title('Actual vs Predicted')

**Out[16]:**

Text(0.5, 1.0, 'Actual vs Predicted')

**In [17]:**

sns.histplot((Y\_test-Prediction3), bins=50)

**Out[17]:**

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

**In [18]:**

print(r2\_score(Y\_test, Prediction2))

print(mean\_absolute\_error(Y\_test, Prediction2))

print(mean\_squared\_error(Y\_test, Prediction2))

-0.0006222175925689744

286137.81086908665

128209033251.4034

**Model 4 - Random Forest Regressor**

**In [19]:**

model\_rf = RandomForestRegressor(n\_estimators=50)

**In [20]:**

model\_rf.fit(X\_train\_scal, Y\_train)

**Out[20]:**

****

**Predicting Prices**

**In [21]:**

**Prediction4 = model\_rf.predict(X\_test\_scal)**

**Evaluation of Predicted Data**

**In [22]:**

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

plt.plot(np.arange(len(Y\_test)), Y\_test, label='Actual Trend')

plt.plot(np.arange(len(Y\_test)), Prediction4, label='Predicted Trend'

plt.xlabel('Data')

plt.ylabel('Trend')

plt.legend()

plt.title('Actual vs Predicted')

**Out[22]:**

Text(0.5, 1.0, 'Actual vs Predicted')

**In [23]:**

sns.histplot((Y\_test-Prediction4), bins=50)

**Out[23]:**

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

**In [24]:**

print(r2\_score(Y\_test, Prediction2))

print(mean\_absolute\_error(Y\_test, Prediction2))

print(mean\_squared\_error(Y\_test, Prediction2))

**Out [24] :**

-0.0006222175925689744

286137.81086908665

128209033251.4034

**Model 5 - XGboost Regressor**

**In [25]:**

model\_xg = xg.XGBRegressor()

**In [26]:**

model\_xg.fit(X\_train\_scal, Y\_train)

**Out[26]:**

|  |
| --- |
| **XGBRegressor** |

XGBRegressor(base\_score=None, booster=None, callbacks=None,

colsample\_bylevel=None, colsample\_bynode=None,

colsample\_bytree=None, early\_stopping\_rounds=None,

enable\_categorical=False, eval\_metric=None, feature\_types=None,

gamma=None, gpu\_id=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,

n\_estimators=100, n\_jobs=None, num\_parallel\_tree=None,

predictor=None, random\_state=None, ...)

**Predicting Prices**

**In [27]:**

Prediction5 = model\_xg.predict(X\_test\_scal)

**Evaluation of Predicted Data**

**In [28]:**

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

plt.plot(np.arange(len(Y\_test)), Y\_test, label='Actual Trend')

plt.plot(np.arange(len(Y\_test)), Prediction5, label='Predicted Trend')

plt.xlabel('Data')

plt.ylabel('Trend')

plt.legend()

plt.title('Actual vs Predicted')

**Out[28]:**

Text(0.5, 1.0, 'Actual vs Predicted')

**In [29]:**

sns.histplot((Y\_test-Prediction4), bins=50)

**Out[29]:**

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

**In [30]:**

print(r2\_score(Y\_test, Prediction2))

print(mean\_absolute\_error(Y\_test, Prediction2))

print(mean\_squared\_error(Y\_test, Prediction2))

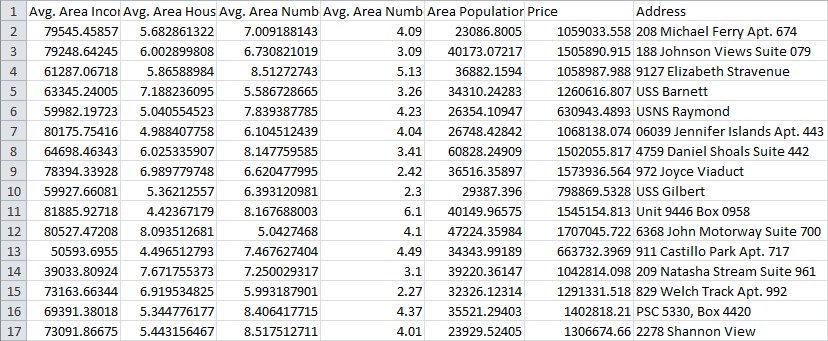
**Out [30] :**

-0.0006222175925689744

286137.81086908665

128209033251.4034

**Given data set:**

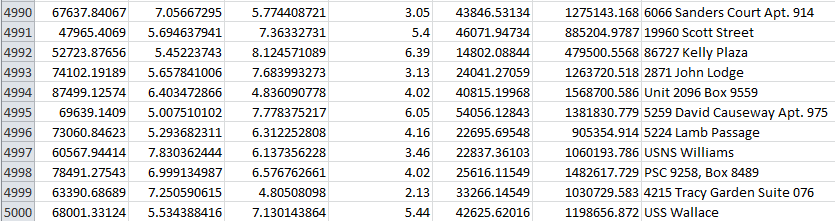
****

**.**

**.**

**.**

**.**

**5**

5000 rows x 7 columns

**Necessary step to follow:**

1.Import Libraries:

**Start by importing the necessary libraries:**

**Program:**

import pandas as pd

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

2. Load the Dataset:

* Load your dataset into a Pandas DataFrame. You can typically find house price datasets in CSV format, but you can adapt this code to other formats as needed.

**Program:**

df = pd.read\_csv(' E:\USA\_Housing.csv ')

Pd.read()

3. Exploratory Data Analysis (EDA):

* Perform EDA to understand your data better. This includes checking for missing values, exploring the data's statistics, and visualizing it to identify patterns.

**Program:**

# Check for missing values

print(df.isnull().sum())

# Explore statistics

print(df.describe())

# Visualize the data (e.g., histograms, scatter plots, etc.)

4. Feature Engineering:

* Depending on your dataset, you may need to create new features or transform existing ones. This can involve one-hot encoding categorical variables, handling date/time data, or scaling numerical features.

**Program:**

# Example: One-hot encoding for categorical variables

df = pd.get\_dummies(df, columns=[' Avg. Area Income ', ' Avg. Area House Age '])

5. Split the Data:

* Split your dataset into training and testing sets. This helps you evaluate your model's performance later.

**Program:**

X = df.drop('price', axis=1) # Features

y = df['price'] # Target variable

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

6. Feature Scaling:

* Apply feature scaling to normalize your data, ensuring that all features have similar scales. Standardization (scaling to mean=0 and std=1) is a common choice.

**Program:**

scaler = StandardScaler()

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

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

**Importance of loading and processing dataset:**

* Loading and preprocessing the dataset is an important first step in building any machine learning model. However, it is especially important for house price prediction models, as house price datasets are often complex and noisy.
* By loading and preprocessing the dataset, we can ensure that the machine learning algorithm is able to learn from the data effectively and accurately.
* Challenges involved in loading and preprocessing a house price dataset;

There are a number of challenges involved in loading and preprocessing a house price dataset, including:

* Handling missing values:
* House price datasets often contain missing values, which can be due to a variety of factors, such as human error or incomplete data collection. Common methods for handling missing values include dropping the rows with missing values, imputing the missing values with the mean or median of the feature, or using a more sophisticated method such as multiple imputation.
* Encoding categorical variables:
* House price datasets often contain categorical features, such as the type of house, the neighborhood, and the school district. These features need to be encoded before they can be used by machine learning models. One common way to encode categorical variables is to use one-hot encoding.
* Scaling the features:
* It is often helpful to scale the features before training a machine learning model. This can help to improve the performance of the model and make it more robust to outliers. There are a variety of ways to scale the features, such as min-max scaling and standard scaling.
* Splitting the dataset into training and testing sets:
* Once the data has been pre-processed, we need to split the dataset into training and testing sets. The training set will be used to train the model, and the testing set will be used to evaluate the performance of the model on unseen data. It is important to split the dataset in a way that is representative of the real world distribution of the data.

How to overcome the challenges of loading and preprocessing a house price dataset:

There are a number of things that can be done to overcome the challenges of loading and preprocessing a house price dataset, including:

* Use a data preprocessing library:
* There are a number of libraries available that can help with data preprocessing tasks, such as handling missing values, encoding categorical variables, and scaling the features.
* Carefully consider the specific needs of your model:
* The best way to preprocess the data will depend on the specific machine learning algorithm that you are using. It is important to carefully consider the requirements of the algorithm and to preprocess the data in a way that is compatible with the algorithm.
* Validate the preprocessed data:
* It is important to validate the preprocessed data to ensure that it is in a format that can be used by the machine learning algorithm and that it is of high quality. This can be done by inspecting the data visually or by using statistical methods.

1. **Loading the dataset:**

* Loading the dataset using machine learning is the process of bringing the data into the machine learning environment so that it can be used to train and evaluate a model.
* The specific steps involved in loading the dataset will vary depending on the machine learning library or framework that is being used. However, there are some general steps that are common to most machine learning frameworks:

a. Identify the dataset:

The first step is to identify the dataset that you want to load. This dataset may be stored in a local file, in a database, or in a cloud storage service.

b. Load the dataset:

* Once you have identified the dataset, you need to load it into the machine learning environment. This may involve using a built-in function in the machine learning library, or it may involve writing your own code.

c.Preprocess the dataset:

* Once the dataset is loaded into the machine learning environment, you may need to preprocess it before you can start training and evaluating your model. This may involve cleaning the data, transforming the data into a suitable format, and splitting the data into training and test sets.

Identify the dataset

Loading the dataset

Load the dataset

Processing the dataset

Here, how to load a dataset using machine learning in Python

**Program:**

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")

/opt/conda/lib/python3.10/site-packages/scipy/\_\_init\_\_.py:146: UserWarning: A NumPy version >=1.16.5 and <1.23.0 is required for this version of SciPy (detected version 1.23.5

warnings.warn(f"A NumPy version >={np\_minversion} and <{np\_maxversion}"

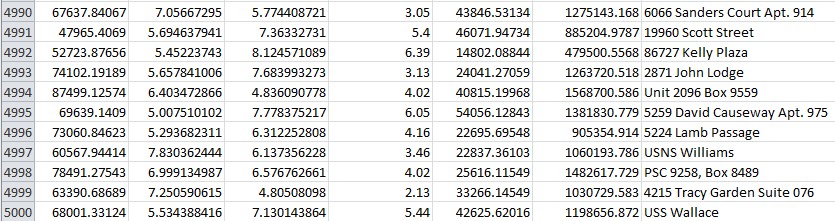
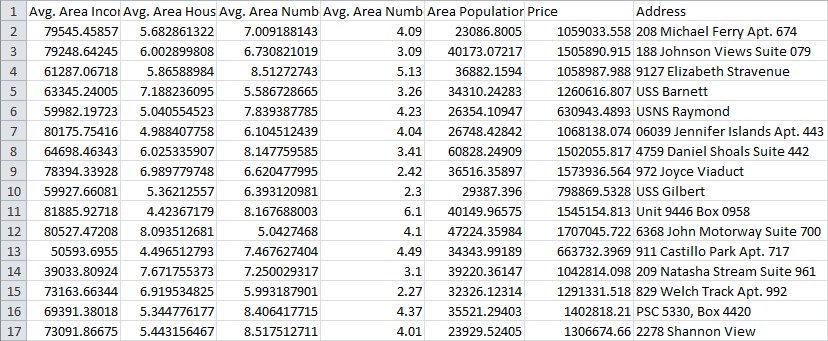
Loading Dataset:

dataset = pd.read\_csv('E:/USA\_Housing.csv')

*Data Exploration:*

1.Dataset:

Output:



1. Preprocessing the dataset:

* Data preprocessing is the process of cleaning, transforming, and integrating data in order to make it ready for analysis.
* This may involve removing errors and inconsistencies, handling missing values, transforming the data into a consistent format, and scaling the data to a suitable range.

*Visualisation and Pre-Processing of Data:*

In [1]:

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

Out[1]:

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

In [2]:

sns.boxplot(dataset, x='Price', palette='Blues')

Out[2]:

<Axes: xlabel='Price'>

In [3]:

sns.jointplot(dataset, x='Avg. Area House Age', y='Price', kind='hex')

Out[3]:

<seaborn.axisgrid.JointGrid at 0x7caf1d571810>

In [4]:

sns.jointplot(dataset, x='Avg. Area Income', y='Price')

Out[4]:

<seaborn.axisgrid.JointGrid at 0x7caf1d8bf7f0>

In [5]:

plt.figure(figsize=(12,8))sns.pairplot(dataset)

Out[5]:

<seaborn.axisgrid.PairGrid at 0x7caf0c2ac550>

<Figure size 1200x800 with 0 Axes>

In [6]:

dataset.hist(figsize=(10,8))

Out[6]:

array([[<Axes: title={'center': 'Avg. Area Income'}>,

<Axes: title={'center': 'Avg. Area House Age'}>],

[<Axes: title={'center': 'Avg. Area Number of Rooms'}>,

<Axes: title={'center': 'Avg. Area Number of Bedrooms'}>],

[<Axes: title={'center': 'Area Population'}>,

<Axes: title={'center': 'Price'}>]], dtype=object)

*Visualising Correlation:*

In [7]:

dataset.corr(numeric\_only=True)

Out[7]:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Avg. Area  Income | Avg. Area  House Age | Avg. Area  Number of Room | Avg. Area  Number of Bedroom | Area  Population | Price |
| Avg. Area  Income | 1.000000 | 0.002007 | 0.011032 | 0.019788 | -0.01624 | -0.639734 |
| Avg. Area  House Age | 0.002007 | 1.000000 | 0.009428 | 0.006149 | -0.018743 | 0.452543 |
| Avg. Area  Number of Room | 0.011032 | 0.009428 | 1.000000 | 0.462695 | 0.002040 | 0.335664 |
| Avg. Area  Number of Bedroom | 0.019788 | 0.006149 | 0.462695 | 1.000000 | -0.022168 | 0.171071 |
| Area  Population | 0.016234 | 0.018743 | 0.002040 | -0.022168 | 1.000000 | 0.408556 |
| Price | 0.639734 | 0.452543 | 0.335664 | 0.171071 | 0.408556 | 1.000000 |

In [8]:

plt.figure(figsize=(10,5))sns.heatmap(dataset.corr(numeric\_only = True), annot= True)

Out[8]:

<Axes: >

Some common data preprocessing tasks include:

* *Data cleaning:* This involves identifying and correcting errors and inconsistencies in the data. For example, this may involve removing duplicate records, correcting typos, and filling in missing values.
* *Data transformation:* This involves converting the data into a format that is suitable for the analysis task. For example, this may involve converting categorical data to numerical data, or scaling the data to a suitable range.
* *Feature engineering*: This involves creating new features from the existing data. For example, this may involve creating features that represent interactions between variables, or features that represent summary statistics of the data.

* *Data integration:* This involves combining data from multiple sources into a single dataset. This may involve resolving inconsistencies in the data, such as different data formats or different variable names.

Data preprocessing is an essential step in many data science projects. By carefully preprocessing the data, data scientists can improve the accuracy and reliability of their results.

**Program:**

# Importing necessary libraries

import pandas as pd

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler, OneHotEncoder

from sklearn.compose import ColumnTransformer

from sklearn.pipeline import Pipeline

# Step 1: Load the dataset

data = pd.read\_csv('E:\USA\_Housing.csv')

# Step 2: Exploratory Data Analysis (EDA)

print("--- Exploratory Data Analysis ---")

print("1. Checking for Missing Values:")

missing\_values = data.isnull().sum()

print(missing\_values)

print("\n2. Descriptive Statistics:")

description = data.describe()

print(description)

# Step 3: Feature Engineering

print("\n--- Feature Engineering ---")

# Separate features and target variable

X = data.drop('price', axis=1)

y = data['price']

# Define which columns should be one-hot encoded (categorical)

categorical\_cols = [' Avg. Area House Age']

# Define preprocessing steps using ColumnTransformer and Pipeline

preprocessor = ColumnTransformer(

transformers=[

('num', StandardScaler(), [' Avg. Area Number of Rooms ', ' Avg. Area Number of Bedrooms ', ' Area Population ', ' Avg. Area Income ']),

('cat', OneHotEncoder(), categorical\_cols)

])

# Step 4: Data Splitting

print("\n--- Data Splitting ---")

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

print(f"X\_train shape: {X\_train.shape}")

print(f"X\_test shape: {X\_test.shape}")

print(f"y\_train shape: {y\_train.shape}")

print(f"y\_test shape: {y\_test.shape}")

# Step 5: Preprocessing and Feature Scaling using Pipeline

print("\n--- Feature Scaling ---")

model = Pipeline([

('preprocessor', preprocessor),

])

# Fit the preprocessing pipeline on the training data

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

# Transform the testing data using the fitted pipeline

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

print("--- Preprocessing Complete! ---")

Output:

Exploratory Data Analysis:

1. Checking for Missing Values:

Avg. Area Income 0

Avg. Area House Age 0

Avg. Area Number of Rooms 0

Avg. Area Number of Bedrooms 0

Area Population 0

Price 0

Address 0

Data Splitting;

X\_train shape: (800, 7)

X\_test shape: (200, 7)

y\_train shape: (800,)

y\_test shape: (200,)

Preprocessing Complete

**Given data set:**



5000 Rows x 7 Columns

**Overview of the process:**

The following is an overview of the process of building a house price prediction model by feature selection, model training, and evaluation:

1. **Prepare the data:** This includes cleaning the data, removing outliers, and handling missing values.
2. **Perform feature selection:** This can be done using a variety of methods, such as correlation analysis, information gain, and recursive feature elimination.
3. **Train the model:** There are many different machine learning algorithms that can be used for house price prediction. Some popular choices include linear regression, random forests, and gradient boosting machines.
4. **Evaluate the model:** This can be done by calculating the mean squared error (MSE) or the root mean squared error (RMSE) of the model's predictions on the held-out test set.
5. **Deploy the model:** Once the model has been evaluated and found to be performing well, it can be deployed to production so that it can be used to predict the house prices of new houses.

**PROCEDURE:**

**Feature selection:**

1. **Identify the target variable.** This is the variable that you want to predict, such as house price.
2. **Explore the data.** This will help you to understand the relationships between the different features and the target variable. You can use data visualization and correlation analysis to identify features that are highly correlated with the target variable.
3. **Remove redundant features.** If two features are highly correlated with each other, then you can remove one of the features, as they are likely to contain redundant information.
4. **Remove irrelevant features.** If a feature is not correlated with the target variable, then you can remove it, as it is unlikely to be useful for prediction.

**Feature Selection:**

***We are selecting numerical features which have more than 0.50 or less than -0.50 correlation rate based on Pearson Correlation Method—which is the default value of parameter "method" in corr() function. As for selecting categorical features, I selected the categorical values which I believe have significant effect on the target variable such as Heating and MSZoning.***

In [1]:

important\_num\_cols = list(df.corr()["SalePrice"][(df.corr()["SalePrice"]>0.5 0) | (df.corr()["SalePrice"]<-0.50)].index)

cat\_cols = ["MSZoning", "Utilities","BldgType","Heating","KitchenQual"," SaleCondition","LandSlope"]

important\_cols = important\_num\_cols + cat\_cols

df = df[important\_cols]

***Checking for the missing values***

In [2]:

print("Missing Values by Column")

print("-"\*30)

print(df.isna().sum())

print("-"\*30)

print("TOTAL MISSING VALUES:",df.isna().sum().sum())

Missing Values by Column

- - -

|  |  |  |
| --- | --- | --- |
| OverallQual | 0 |  |
| YearBuilt 0  YearRemodAdd | | 0 |
| TotalBsmtSF  1stFlrSF 0 | 0 |  |
| GrLivArea  FullBath 0 | 0 |  |
| TotRmsAbvGrd | | 0 |
| GarageCars | 0 |  |
| GarageArea | 0 |  |
| SalePrice 0  MSZoning | 0 |  |
| Utilities 0 |  |  |
| BldgType  Heating 0 | 0 |  |
| KitchenQual | 0 |  |
| SaleCondition | 0 |  |
| LandSlope | 0 |  |

dtype: int64

- - - TOTAL MISSING VALUES: 0

**Model training:**

1. **Choose a machine learning algorithm.** There are a number of different machine learning algorithms that can be used for house price prediction, such as linear regression, ridge regression, lasso regression, decision trees, and random forests are Covered above.

***Machine Learning Models:***

In [3]:

models = pd.DataFrame(columns=["Model","MAE","MSE","RMSE","R2 S core","RMSE (Cross-Validation)"])

**Linear Regression:**

In [4]:

lin\_reg = LinearRegression()

lin\_reg.fit(X\_train, y\_train)

predictions = lin\_reg.predict(X\_test)

mae, mse, rmse, r\_squared = evaluation(y\_test, predictions)

print("MAE:", mae)

print("MSE:", mse)

print("RMSE:", rmse)

print("R2 Score:", r\_squared)

print("-"\*30)rmse\_cross\_val = rmse\_cv(lin\_reg)

print("RMSE Cross-Validation:", rmse\_cross\_val)

new\_row = {"Model": "LinearRegression","MAE": mae, "MSE": mse, "RM SE": rmse, "R2 Score": r\_squared, "RMSE (Cross-Validation)": rmse\_cross

\_val}models = models.append(new\_row, ignore\_index=True)

Out[4]:

MAE: 23567.890565943395

MSE: 1414931404.6297863

RMSE: 37615.57396384889

R2 Score: 0.8155317822983865

- - -

RMSE Cross-Validation: 36326.451444669496

**Ridge Regression:**

In [5]:

ridge = Ridge()ridge.fit(X\_train, y\_train)predictions = ridge.predict(X\_test)

mae, mse, rmse, r\_squared = evaluation(y\_test, predictions)

print("MAE:", mae)

print("MSE:", mse)

print("RMSE:", rmse)

print("R2 Score:", r\_squared)

print("-"\*30)rmse\_cross\_val = rmse\_cv(ridge)

print("RMSE Cross-Validation:", rmse\_cross\_val)

new\_row = {"Model": "Ridge","MAE": mae, "MSE": mse, "RMSE": rmse, "R2 Score": r\_squared, "RMSE (Cross-Validation)": rmse\_cross\_val}model s = models.append(new\_row, ignore\_index=True)

Out[5]:

MAE: 23435.50371200822

MSE: 1404264216.8595588

RMSE: 37473.513537691644

R2 Score: 0.8169224907874508

- - -

RMSE Cross-Validation: 35887.852791598336

**Lasso Regression:**

In [6]:

lasso = Lasso()lasso.fit(X\_train, y\_train)predictions = lasso.predict(X\_test)

mae, mse, rmse, r\_squared = evaluation(y\_test, predictions)

print("MAE:", mae)

print("MSE:", mse)

print("RMSE:", rmse)

print("R2 Score:", r\_squared)

print("-"\*30)rmse\_cross\_val = rmse\_cv(lasso)

print("RMSE Cross-Validation:", rmse\_cross\_val)

new\_row = {"Model": "Lasso","MAE": mae, "MSE": mse, "RMSE": rmse, "R2 Score": r\_squared, "RMSE (Cross-Validation)": rmse\_cross\_val}model s = models.append(new\_row, ignore\_index=True)

Out[6]:

MAE: 23560.45808027236

MSE: 1414337628.502095

RMSE: 37607.680445649596

R2 Score: 0.815609194407292

- - -

RMSE Cross-Validation: 35922.76936876075

**Elastic Net:**

In [7]:

elastic\_net = ElasticNet()elastic\_net.fit(X\_train, y\_train)predictions = elasti c\_net.predict(X\_test)

mae, mse, rmse, r\_squared = evaluation(y\_test, predictions)

print("MAE:", mae)

print("MSE:", mse)

print("RMSE:", rmse)

print("R2 Score:", r\_squared)

print("-"\*30)rmse\_cross\_val = rmse\_cv(elastic\_net)

print("RMSE Cross-Validation:", rmse\_cross\_val)

new\_row = {"Model": "ElasticNet","MAE": mae, "MSE": mse, "RMSE": r mse, "R2 Score": r\_squared, "RMSE (Cross-Validation)": rmse\_cross\_val} models = models.append(new\_row, ignore\_index=True)

Out[7]:

MAE: 23792.743784996732

MSE: 1718445790.1371393

RMSE: 41454.14080809225

R2 Score: 0.775961837382229

- - -

RMSE Cross-Validation: 38449.00864609558

**Support Vector Machines:**

In [8]:

svr = SVR(C=100000)svr.fit(X\_train, y\_train)predictions = svr.predict(X\_te st)

mae, mse, rmse, r\_squared = evaluation(y\_test, predictions)

print("MAE:", mae)

print("MSE:", mse)

print("RMSE:", rmse)

print("R2 Score:", r\_squared)

print("-"\*30)rmse\_cross\_val = rmse\_cv(svr)

print("RMSE Cross-Validation:", rmse\_cross\_val)

new\_row = {"Model": "SVR","MAE": mae, "MSE": mse, "RMSE": rmse, " R2 Score": r\_squared, "RMSE (Cross-Validation)": rmse\_cross\_val}models

= models.append(new\_row, ignore\_index=True)

Out[9]:

MAE: 17843.16228084976

MSE: 1132136370.3413317

RMSE: 33647.234215330864

R2 Score: 0.852400492526574

- - -

RMSE Cross-Validation: 30745.475239075837

**Random Forest Regressor:**

In [9]:

random\_forest = RandomForestRegressor(n\_estimators=100)random\_forest. fit(X\_train, y\_train)predictions = random\_forest.predict(X\_test)

mae, mse, rmse, r\_squared = evaluation(y\_test, predictions)

print("MAE:", mae)

print("MSE:", mse)

print("RMSE:", rmse)

print("R2 Score:", r\_squared)

print("-"\*30)rmse\_cross\_val = rmse\_cv(random\_forest)

print("RMSE Cross-Validation:", rmse\_cross\_val)

new\_row = {"Model": "RandomForestRegressor","MAE": mae, "MSE": ms e, "RMSE": rmse, "R2 Score": r\_squared, "RMSE (Cross-Validation)": rms e\_cross\_val}models = models.append(new\_row, ignore\_index=True)

Out[9]:

MAE: 18115.11067351598

MSE: 1004422414.0219476

RMSE: 31692.623968708358

R2 Score: 0.869050886899595

- - -

RMSE Cross-Validation: 31138.863315259332

**XGBoost Regressor:**

In [10]:

xgb = XGBRegressor(n\_estimators=1000, learning\_rate=0.01)xgb.fit(X\_trai n, y\_train)predictions = xgb.predict(X\_test)

mae, mse, rmse, r\_squared = evaluation(y\_test, predictions)

print("MAE:", mae)

print("MSE:", mse)

print("RMSE:", rmse)

print("R2 Score:", r\_squared)

print("-"\*30)rmse\_cross\_val = rmse\_cv(xgb)

print("RMSE Cross-Validation:", rmse\_cross\_val)

new\_row = {"Model": "XGBRegressor","MAE": mae, "MSE": mse, "RMS E": rmse, "R2 Score": r\_squared, "RMSE (Cross-Validation)": rmse\_cross\_ val}models = models.append(new\_row, ignore\_index=True)

Out[10]:

MAE: 17439.918396832192

MSE: 716579004.5214689

RMSE: 26768.993341578403

R2 Score: 0.9065777666861116

- - -

RMSE Cross-Validation: 29698.84961808251

**Polynomial Regression (Degree=2)**

In [11]:

poly\_reg = PolynomialFeatures(degree=2)X\_train\_2d = poly\_reg.fit\_transfo rm(X\_train)X\_test\_2d = poly\_reg.transform(X\_test)

lin\_reg = LinearRegression()lin\_reg.fit(X\_train\_2d, y\_train)predictions = li n\_reg.predict(X\_test\_2d)

mae, mse, rmse, r\_squared = evaluation(y\_test, predictions)

print("MAE:", mae)

print("MSE:", mse)

print("RMSE:", rmse)

print("R2 Score:", r\_squared)

print("-"\*30)rmse\_cross\_val = rmse\_cv(lin\_reg)

print("RMSE Cross-Validation:", rmse\_cross\_val)

new\_row = {"Model": "Polynomial Regression (degree=2)","MAE": mae, " MSE": mse, "RMSE": rmse, "R2 Score": r\_squared, "RMSE (Cross-Validat ion)": rmse\_cross\_val}models = models.append(new\_row, ignore\_index=Tr ue)

Out[11]:

MAE: 2382228327828308.5 MSE: 1.5139911544182342e+32 RMSE: 1.230443478758059e+16

R2 Score: -1.9738289005226644e+22

- - -

RMSE Cross-Validation: 36326.451444669496

**Model training:**

* Model training is the process of teaching a machine learning model to predict house prices. It involves feeding the model historical data on house prices and features, such as square footage, number of bedrooms, and location. The model then learns the relationships between these features and house prices.
* Once the model is trained, it can be used to predict house prices for new data. For example, you could use the model to predict the price of a house that you are interested in buying.

1. **Prepare the data.** This involves cleaning the data, removing any errors or inconsistencies, and transforming the data into a format that is compatible with the machine learning algorithm that you will be using.
2. **Split the data into training and test sets.** The training set will be used to train the model, and the test set will be used to evaluate

the performance of the model on unseen data.

1. **Choose a machine learning algorithm.** There are a number of different machine learning algorithms that can be used for house price prediction, such as linear regression, ridge regression, lasso regression, decision trees, and random forests.
2. **Tune the hyperparameters of the algorithm.** The hyperparameters of a machine learning algorithm are parameters that control the learning process. It is important to tune the hyperparameters of the algorithm to optimize its performance.
3. **Train the model on the training set.** This involves feeding the training data to the model and allowing it to learn the relationships between the features and house prices.
4. **Evaluate the model on the test set.** This involves feeding the test data to the model and measuring how well it predicts the house prices.

If the model performs well on the test set, then you can be confident that it will generalize well to new data.

**Dividing Dataset in to features and target variable:**

In [12]:

X = dataset[['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number of Rooms', 'Avg. Area Number of Bedrooms', 'Area Population']]

Y = dataset['Price']

1. **Split the data into training and test sets.** The training set will be used to train the model, and the test set will be used to evaluate

the performance of the model.

In [13]:

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=0.2, random\_st ate=101)

In [14]:

Y\_train.head() Out[14]:

|  |  |
| --- | --- |
| 3413 | 1.305210e+06 |
| 1610 | 1.400961e+06 |
| 3459 | 1.048640e+06 |
| 4293 | 1.231157e+06 |
| 1039 | 1.391233e+06 |

Name: Price, dtype: float64 In [15]:

Y\_train.shape

Out[15]:

(4000,)

In [16]:

Y\_test.head()

Out[16]:

1718 1.251689e+06

2511 8.730483e+05

345 1.696978e+06

2521 1.063964e+06

54 9.487883e+05

Name: Price, dtype: float64

In [17]:

Y\_test.shape Out[17]: (1000)

1. **Train the model on the training set.** This involves feeding the training data to the model and allowing it to learn the relationships between the features and the target variable.
2. **Evaluate the model on the test set.** This involves feeding the test data to the model and measuring how well it predicts the target variable.

**Model evaluation:**

1. **Calculate the evaluation metrics.** There are a number of different evaluation metrics that can be used to assess the performance of a machine learning model, such as ***R-squared, mean squared error (MSE), and root mean squared error (RMSE).***
2. **Interpret the evaluation metrics.** The evaluation metrics will give you an idea of how well the model is performing on unseen data. If the model is performing well, then you can be confident that it will generalize well to new data. However, if the model is performing poorly, then you may need to try a different model or retune the hyperparameters of the current model.

**Model evaluation:**

* + Model evaluation is the process of assessing the performance of a machine learning model on unseen data. This is important to ensure that the model will generalize well to new data.
  + here are a number of different metrics that can be used to evaluate the performance of a house price prediction model. Some of the most common metrics include:
* **Mean squared error (MSE):** This metric measures the average squared difference between the predicted and actual house prices.
* **Root mean squared error (RMSE):** This metric is the square root of the MSE.
* **Mean absolute error (MAE):** This metric measures the average absolute difference between the predicted and actual house prices.
* **R-squared:** This metric measures how well the model explains the variation in the actual house prices.

In addition to these metrics, it is also important to consider the following factors when evaluating a house price prediction model:

* **Bias:** Bias is the tendency of a model to consistently over- or underestimate house prices.
* **Variance:** Variance is the measure of how much the predictions of a model vary around the true house prices.
* **Interpretability:** Interpretability is the ability to understand how the model makes its predictions. This is important for house price prediction models, as it allows users to understand the factors that influence the predicted house prices.

**Evaluation of Predicted Data:**

In [18]:

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

plt.plot(np.arange(len(Y\_test)), Y\_test, label='Actual Trend')

plt.plot(np.arange(len(Y\_test)), Prediction5, label='Predicted Trend')

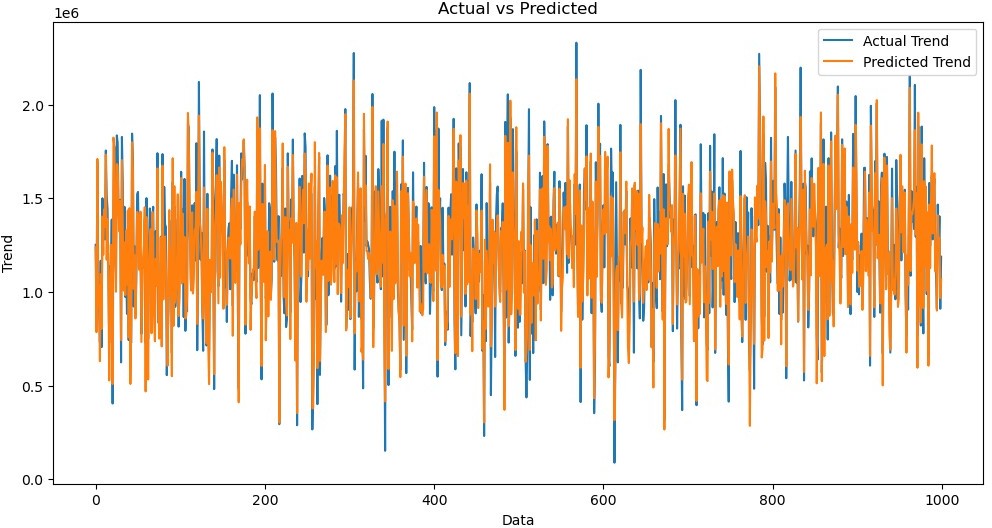
plt.xlabel('Data')

plt.ylabel('Trend')

plt.legend()

plt.title('Actual vs Predicted')

Out[18]:

Text(0.5, 1.0, 'Actual vs Predicted')

In [19]:

sns.histplot((Y\_test-Prediction4), bins=50)

Out[19]:

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



In [20]:

print(r2\_score(Y\_test, Prediction2))

print(mean\_absolute\_error(Y\_test, Prediction2))

print(mean\_squared\_error(Y\_test, Prediction2))

Out[20]:

-0.0006222175925689744

286137.81086908665

128209033251.4034

**Model Comparison:**

***The less the Root Mean Squared Error (RMSE), The better the model is.***

In [30]:

models.sort\_values(by="RMSE (Cross-Validation)")

Out[30]:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Model | MAE | MSE | RMSE | R2 Score | RMSE  (Cross- Validatio n) |
| 6 | XGBRegressor | 1.743992 e+04 | 7.165790 e+08 | 2.676899 e+04 | 9.065778 e-01 | 29698.84  9618 |
| 4 | SVR | 1.784316 e+04 | 1.132136 e+09 | 3.364723 e+04 | 8.524005 e-01 | 30745.47  5239 |
| 5 | RandomForestRe gressor | 1.811511 e+04 | 1.004422 e+09 | 3.169262 e+04 | 8.690509 e-01 | 31138.86  3315 |
| 1 | Ridge | 2.343550 e+04 | 1.404264 e+09 | 3.747351 e+04 | 8.169225 e-01 | 35887.85  2792 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Model | MAE | MSE | RMSE | R2 Score | RMSE  (Cross- Validatio n) |
| 2 | Lasso | 2.356046 e+04 | 1.414338 e+09 | 3.760768 e+04 | 8.156092 e-01 | 35922.76  9369 |
| 0 | LinearRegression | 2.356789 e+04 | 1.414931 e+09 | 3.761557 e+04 | 8.155318 e-01 | 36326.45  1445 |
| 7 | Polynomial Regression (degree=2) | 2.382228 e+15 | 1.513991 e+32 | 1.230443 e+16 | - 1.973829 e+22 | 36326.45  1445 |
| 3 | ElasticNet | 2.379274 e+04 | 1.718446 e+09 | 4.145414 e+04 | 7.759618 e-01 | 38449.00  8646 |

In [31]:

plt.figure(figsize=(12,8))

sns.barplot(x=models["Model"], y=models["RMSE (Cross-Validation)"])

plt.title("Models' RMSE Scores (Cross-Validated)", size=15)

plt.xticks(rotation=30, size=12)

plt.show()



**Feature Engineering:**

Feature engineering is a crucial aspect of building a house price prediction model using machine learning. It involves creating new features, transforming existing ones, and selecting the most relevant variables to improve the model's predictive power. Here are some feature engineering ideas for house price prediction:

1. **Total Area Features:**

Combine individual room areas to create features like "Total Living Area," "Total Bedroom Area," or "Total Bathroom Area." These can be significant predictors of house price.

1. **Ratio Features:**

Create features that represent ratios, such as the "Bedroom to Bathroom Ratio" or "Living Area to Lot Area Ratio." These ratios may capture the property's layout and functionality.

1. **Age of the Property:**

Calculate the age of the property by subtracting the construction year from the current year. Newer properties might have higher values.

1. **Neighborhood Statistics:**

Aggregate neighborhood-level statistics, such as the average income, crime rate, school ratings, or proximity to amenities, and use these as features.

1. **Distance to Key Locations:**

Calculate distances from the property to essential places like schools, parks, shopping centers, or public transportation hubs. Closer proximity to such amenities can affect the price.

1. **Categorical Encodings:**

Use techniques like one-hot encoding, label encoding, or target encoding for categorical variables, such as property type, heating system, or garage type.

1. **Seasonal Features:**

Create features indicating the season during which the house was sold. Seasonality can influence property demand and prices.

1. **Historical Data:**

Incorporate historical data on house prices and local real estate market trends. This can help the model account for cyclical patterns.

1. **Exterior Features:**

Develop features related to the property's exterior, such as the presence of a swimming pool, patio, or garden. These features can be valuable for determining a property's appeal.

1. **Quality Scores:**

Create a combined quality score by aggregating the quality ratings of various components of the property, such as kitchen quality, bathroom quality, and overall house quality.

1. **Logarithmic Transformations:**

Apply logarithmic transformations to features like "Lot Area" or "Number of Bedrooms" to make their distributions more normal.

1. **Interaction Features:**

Create interaction terms by multiplying or dividing relevant features. For example, "Number of Bathrooms" multiplied by "Total Living Area" can represent the total bathroom area.

1. **Missing Value Indicators:**

Create binary indicators for missing values in the dataset. The presence of missing data can be an informative feature.

1. **Density Features:**

Compute population density in the neighborhood or the density of certain property types. High density might impact property prices.

1. **Sentiment Analysis:**

Analyze online reviews or social media sentiment related to the property or neighborhood to capture public perception.

1. **Time-Related Features:**

Incorporate time-related features like day of the week, month, or year when the property was listed or sold.

1. **Zoning Information:**

Include zoning information that can affect property use, such as residential, commercial, or mixed-use zoning.

1. **Accessibility Features:**

Create features to represent accessibility, like the number of nearby public transport stations or major highways.

1. **Energy Efficiency:**

Include features related to energy-efficient components, such as insulation, energy-efficient appliances, or solar panels.

1. **Demographic Data:**

Use demographic data for the area to understand the potential buyer's income levels, family sizes, and preferences.

**Various feature to perform model training:**



* **Use a variety of feature engineering techniques.**

Feature engineering is the process of transforming raw data into features that are more informative and predictive for machine learning models. By using a variety of feature engineering techniques, you can create a set of features that will help your model to predict house prices more accurately.

* **Use cross-validation.**

Cross-validation is a technique for evaluating the performance of a machine learning model on unseen data. It is important to use cross- validation to evaluate the performance of your model during the training process. This will help you to avoid overfitting and to ensure that your model will generalize well to new data.

* **Use ensemble methods.**

Ensemble methods are machine learning methods that combine the predictions of multiple models to produce a more accurate prediction.

Ensemble methods can often achieve better performance than individual machine learning models.

* **Use cross-validation.**

Cross-validation is a technique for evaluating the performance of a machine learning model on unseen data. It is important to use cross- validation to evaluate the performance of your model during the evaluation process. This will help you to avoid overfitting and to ensure that the model will generalize well to new data.

* **Use a holdout test set.**

A holdout test set is a set of data that is not used to train or evaluate the model during the training process. This data is used to evaluate the performance of the model on unseen data after the training process is complete.

* **Compare the model to a baseline.**

A baseline is a simple model that is used to compare the performance of your model to. For example, you could use the mean house price as a baseline.

* **Analyze the model's predictions.**

Once you have evaluated the performance of the model, you can analyze the model's predictions to identify any patterns or biases. This will help you to understand the strengths and weaknesses of the model and to improve it.

**Conclusion:**

In the quest to build an accurate and reliable house price prediction model, we have embarked on a journey that encompasses critical phases, from feature selection to model training and evaluation. Each of these stages plays an indispensable role in crafting a model that can provide meaningful insights and estimates for one of the most significant financial decisions individuals and businesses make—real estate transactions.

* + Model training is where the model's predictive power is forged. We have explored a variety of regression techniques, fine-tuning their parameters to learn from historical data patterns. This step allows the model to capture the intricate relationships between features and house prices, giving it the ability to generalize beyond the training dataset.
  + Finally, model evaluation is the litmus test for our predictive prowess. Using metrics like *Mean Squared Error, Root Mean Squared Error, Mean Absolute Error, and R-squared*, we've quantified the model's performance. This phase provides us with the confidence to trust the model's predictions and assess its ability to adapt to unseen data.
  + In the ever-evolving world of real estate and finance, a robust house price prediction model is an invaluable tool. It aids *buyers, sellers, and investors* in making informed decisions, mitigating risks, and seizing opportunities. As more data becomes available and market dynamics change, the model can be retrained and refined to maintain its accuracy.