

PROJECT 9: PREDICTING HOUSE PRICE USING MACHINE LEARNING

PHASE 5:PROJECT DOCUMENTATION & SUBMISSION

Project Title: House Price Predictor

Problem Statement:

The housing market is an important and complex sector that impacts people's lives in many ways. For many individuals and families, buying a house is one of the biggest investments they will make in their lifetime. Therefore, it is essential to accurately predict the prices of houses so that buyers and sellers can make informed decisions. This project aims to use machine learning techniques to predict house prices based on various features such as location, square footage, number of bedrooms and bathrooms, and other relevant factors.

Project Steps

Phase 5:

Problem Definition:

The problem is to predict house prices using machine learning techniques. The objective is to develop a model that accurately predicts the prices of houses based on a set of features such as location, square footage, number of bedrooms and bathrooms, and other relevant factors. This project involves data preprocessing, feature engineering, model selection, training, and evaluation.

Design Thinking:

DATA SOURCE:

Choose a dataset containing information about houses, including features like Income, House Age, Number of Rooms, Number of Bedrooms, Population, Price and Address.

Features/Attributes:

These are variables or characteristics of a property that are believed to have an impact on its price. Common features includes

- **Number of Rooms and Bedrooms:** The count of rooms and bedrooms in the house, which can affect its price due to the increased area and size.
- **Address:** The geographical location of the property, which can include factors like neighbourhood, city, and proximity to amenities or landmarks.
- **House**

Age: The age of the house or building can also influence its price.

- **Income:** The income earned by the people living in those areas can also influence its price.
- **Population:** The number of people who are currently living in those areas can also influence the price of the houses.
- **Price:** The total cost of the house can be displayed here based on certain factors like area, number of rooms, age etc..
- **Target Variable:** This is the variable you're trying to predict, which is the price of the house or property. It is typically represented as the dependent variable in a regression analysis.
- **Dataset Size:** The number of records or data points in the dataset, which can range from a few hundred to thousands or more.
- **Data Sources:** Information about where the data was collected from, such as real estate listings, government records, or surveys.

DATA PREPROCESSING:

It is a crucial step in preparing a house price prediction dataset for analysis or machine learning. Here's a short overview of the key steps involved:

Data Cleaning:

Handle missing values by filling them with appropriate values (e.g., mean, median, or mode). Remove duplicates if they exist in the dataset. Correct any inconsistent or erroneous data entries.

Feature Selection:

Choose relevant features that are likely to impact house prices. Remove irrelevant or redundant features to simplify the dataset. Consider using domain knowledge and feature importance techniques.

Feature Encoding:

Convert categorical variables into numerical representations through techniques like one-hot encoding or label encoding. Standardize or normalize numerical features to have a consistent scale (e.g., using MinMax scaling or z-score normalization).

Outlier Detection and Handling:

Identify and handle outliers in the data, either by removing them or transforming them. Use visualization and statistical methods (e.g., Z-scores or IQR) to detect outliers.

Data Splitting:

Split the dataset into training, validation, and test sets to assess model performance effectively. A common split ratio is 70-80% training, 10-15% validation, and 10-15% testing.

Handling Skewed Data:

If the target variable (house prices) or some features are highly skewed, consider applying transformations like log transformations to make the data more symmetric.

Scaling and Normalization:

Ensure that numerical features are scaled or normalized to have similar

scales, preventing some features from dominating others during modeling.

Data Transformation (if needed):

For some modeling algorithms, you might need to transform the data to meet their assumptions (e.g., transforming the target variable for linear regression).

Data preprocessing ensures that the dataset is clean, well-structured, and ready for analysis or modeling. The specific steps may vary depending on the dataset and the machine learning algorithm you plan to use, but these general steps provide a solid foundation for preparing data for house price prediction tasks.

FEATURE SELECTION:

Feature selection for a house price prediction dataset involves choosing the most relevant and informative features while excluding irrelevant ones to improve the accuracy of your prediction model. Here's a concise guide to feature selection:

Correlation Analysis:

Identify features that have a strong correlation with the target variable (house price) using techniques like Pearson correlation. Keep features with high correlations and eliminate those with low or negative correlations.

Feature Importance:

If you're using tree-based models (e.g., Random Forest, XGBoost), use their feature importance scores to rank and select the most important features.

Cross-Validation:

Use cross-validation to evaluate different feature subsets' performance and select the one that yields the best model performance metrics.

Regularization Hyperparameters:

When using models like Ridge or Elastic Net, tune their regularization hyperparameters to encourage feature selection during training.

MODEL SELECTION:

Choose a suitable regression algorithm (e.g., Linear Regression, Random Forest Regressor) for predicting house prices.

Linear Regression:

Start with a basic linear regression model, which is simple and interpretable. It's a good baseline.

Decision Trees:

Try decision tree-based models like Random Forest and Gradient Boosting. They often perform well and handle non-linear relationships.

Neural Networks:

Experiment with deep learning models, such as neural networks, if you have a large dataset and complex features.

MODEL TRAINING:

Train the selected model using the preprocessed data.

Train the selected model on the training data using the features to predict house prices.

The model will learn the relationships between the features and target variable.

EVALUATION:

Evaluate the model's performance using metrics like Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and R-squared.

MAE: It represents the average absolute error in house price predictions. Lower MAE values indicate better model accuracy. It's easy to understand and suitable for comparing models.

RMSE: RMSE penalizes larger errors more than MAE because of the squaring. It's also sensitive to outliers. Lower RMSE values indicate better model accuracy.

R-squared: R-squared measures the goodness of fit. It tells you the proportion of variance in the target variable that is explained by the model. Higher R-squared values (closer to 1) indicate a better fit.

INNOVATION:

Consider exploring advanced regression techniques like Gradient Boosting or XGBoost for improved prediction accuracy.

INTRODUCTION:

Gradient Boosting and XGBoost are fantastic choices for improving prediction accuracy, especially in regression tasks like house price prediction. These techniques belong to the ensemble learning family, where multiple models are combined to create a stronger, more accurate predictor. Gradient Boosting builds trees sequentially, with each tree correcting the errors of the previous ones. XGBoost, or Extreme Gradient Boosting, is an optimized implementation of gradient boosting with additional features like regularization and parallel processing, making it even more powerful and efficient.

These algorithms have proven to be highly effective in various machine learning competitions and real-world applications. They handle complex relationships between features and the target variable, capture non-linear patterns, and often outperform traditional linear models. When implementing these techniques, it's crucial to tune hyperparameters carefully to achieve the best performance. Crossvalidation and grid search can help identify the optimal combination of parameters for your specific dataset.

DATASET:

For the above dataset, we have the following features:

- Avg. Area Income
- Avg. Area House Age
- Avg. Area Number of Rooms

- Avg. Area Number of Bedrooms
- Area Population
- Price
- Address

GRADIENT BOOSTING:

Gradient Boosting is an ensemble learning technique used for both classification and regression tasks. It builds a predictive model in a stage-wise fashion, where each stage corrects the errors of the previous one. The general idea is to combine the predictions of multiple weak learners (often decision trees) to create a strong, accurate model.

Here's a high-level overview of how Gradient Boosting works:

Initialization:

A simple model is created, often the mean or median of the target variable for regression tasks.

Iteration:

Sequential trees (weak learners) are built, and each subsequent tree corrects the errors of the combined predictions of the existing trees.

Learning Rate:

A hyperparameter called the learning rate controls the contribution of each tree to the final prediction. A lower learning rate requires more trees but may result in better generalization.

Handles Non-Linearity:

Gradient Boosting is well-suited for capturing non-linear relationships between input features and the target variable. It can automatically adapt to complex patterns in the data.

Stopping Criteria:

The process continues until a specified number of trees are built or until a certain level of performance is reached.

SAMPLE CODE:

```
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
# Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
/ random_state=42)
# Gradient Boosting Model
gb_model = GradientBoostingRegressor(learning_rate=0.1, n_estimators=100,
max_depth=3)
# Model Evaluation
y_pred_gb = gb_model.predict(X_test)
mse_gb = mean_squared_error(y_test, y_pred_gb)
```

```
Print(f'Mean Squared Error (Gradient Boosting): {mse_gb}')
```

XGBOOST:

XGBoost (Extreme Gradient Boosting) is a specific implementation of gradient boosting that is highly efficient and scalable. It was developed by Tianqi Chen and is widely used in machine learning competitions and real-world applications. Key features of

XGBoost include:

Regularization:

XGBoost incorporates L1 (Lasso) and L2 (Ridge) regularization terms in its objective function to control overfitting.

Parallelization:

It is designed for parallel and distributed computing, making it faster than traditional gradient boosting implementations.

Missing Value Handling:

XGBoost can handle missing values in the dataset, eliminating the need for imputation.

Tree Pruning:

Trees are pruned during the building process to prevent overfitting and improve computational efficiency.

Cross-Validation:

XGBoost has built-in cross-validation capabilities to help with hyperparameter tuning.

Optimized Implementation:

The algorithm is optimized for performance and memory usage.

SAMPLE CODE:

```
from xgboost import XGBRegressor
# XGBoost Model
xgb_model = XGBRegressor(learning_rate=0.1, n_estimators=100, max_depth=3)
xgb_model.fit(X_train, y_train)
# Model Evaluation
y_pred_xgb = xgb_model.predict(X_test)
mse_xgb = mean_squared_error(y_test, y_pred_xgb)
print(f'Mean Squared Error (XGBoost): {mse_xgb}')
```

PURPOSE OF USING THESE TECHNIQUES:

High Predictive Accuracy:

Both Gradient Boosting and XGBoost often outperform traditional linear models in terms of predictive accuracy, especially when dealing with complex relationships and non-linear patterns.

Versatility:

These techniques are versatile and can be applied to a wide range of regression problems, from predicting house prices to financial forecasting.

Robustness:

The regularization techniques employed by these methods contribute to building more robust models, reducing the risk of overfitting.

When applying these techniques, it's essential to fine-tune hyperparameters, handle feature engineering thoughtfully, and ensure proper validation to achieve the best performance on your specific dataset.

PHASE OF DEVELOPMENT :

INTRODUCTION:

A house price prediction project in machine learning is a common and practical application of regression analysis. The goal is to develop a model that can predict the selling price of houses based on various features or attributes such as square footage, number of bedrooms, location, and more. Here are the general steps involved in such a project.

GOOGLE COLAB LINK:

 [house price prediction](#)

IMPORTING DATASET:

A house price dataset that suits your project's requirements, Once you've identified a suitable dataset, download it to your local machine. House price datasets are typically available in formats like CSV, Excel, or database files. Make sure you know where the dataset is saved on your computer.

IMPORTING LIBRARIES:

- Pandas is a powerful data manipulation and analysis library.
- NumPy is used for numerical and mathematical operations Python, especially for working with arrays and matrices.

- Seaborn is a data visualization library built on top of Matplotlib. It provides a high-level interface for creating informative and attractive statistical graphics.
- Matplotlib is a widely used library for creating static, animated, and interactive visualizations in Python.
- The `train_test_split` function is used to split a dataset into training and testing subsets, which is crucial for model training and evaluation in machine learning.
- This preprocessing step is often necessary to make sure features are on the same scale, which can improve the performance of many machine learning algorithms.

```
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")
```

LOAD DATASET:

- `pd` is an alias for the Pandas library, which you imported earlier.
- `read_csv()` is a Pandas function that is used to read CSV files and load their data into a DataFrame.

```
dataset = pd.read_csv('USA_Housing.csv')
```

DATA CLEANING:

Data cleaning is a crucial step in the data preparation process. It involves identifying and correcting errors or inconsistencies in your data to ensure that it is accurate, reliable, and suitable for analysis.

- Data Collection
- Handling Missing Values
- Handling Duplicate Data
- Data Transformation
- Data Encoding
- Data Validation

```

dataset.drop(['Avg. Area Income'],
             axis=1,
             inplace=True)

```

This line of code drops the 'Avg. Area Income' column from the DataFrame named 'dataset'.

```

[4] dataset['Price'] = dataset['Price'].fillna(
    dataset['Price'].mean())
    print(dataset['Price'])

```

0	1.059034e+06
1	1.505891e+06
2	1.058988e+06
3	1.260617e+06
4	6.309435e+05
...	
4995	1.060194e+06
4996	1.482618e+06
4997	1.030730e+06
4998	1.198657e+06
4999	1.298950e+06

Name: Price, Length: 5000, dtype: float64

This code fills missing values in the 'Price' column of the 'dataset' DataFrame with the mean value of the 'Price' column and then prints the updated 'Price' column.

```

[ ] new_dataset = dataset.dropna()
    print(new_dataset)

```

	Avg. Area House Age	Avg. Area Number of Rooms	\
0	5.682861	7.009188	
1	6.002900	6.730821	
2	5.865890	8.512727	
3	7.188236	5.586729	
4	5.040555	7.839388	
...	
4995	7.830362	6.137356	
4996	6.999135	6.576763	
4997	7.250591	4.805081	
4998	5.534388	7.130144	
4999	5.992305	6.792336	

```
✓ 0s dataset.duplicated()

0      False
1      False
2      False
3      False
4      False
...
4995   False
4996   False
4997   False
4998   False
4999   False
Length: 5000, dtype: bool
```

```
✓ 0s dataset.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
```

```
[ ] dataset.isnull()

   Avg. Area Income  Avg. Area House Age  Avg. Area Number of Rooms  Avg. Area Number of Bedrooms  Area Population  Price  Address
0              False              False              False              False              False  False  False
1              False              False              False              False              False  False  False
2              False              False              False              False              False  False  False
3              False              False              False              False              False  False  False
4              False              False              False              False              False  False  False
...              ...                  ...                  ...                  ...                  ...      ...
4995             False              False              False              False              False  False  False
4996             False              False              False              False              False  False  False
4997             False              False              False              False              False  False  False
4998             False              False              False              False              False  False  False
4999             False              False              False              False              False  False  False

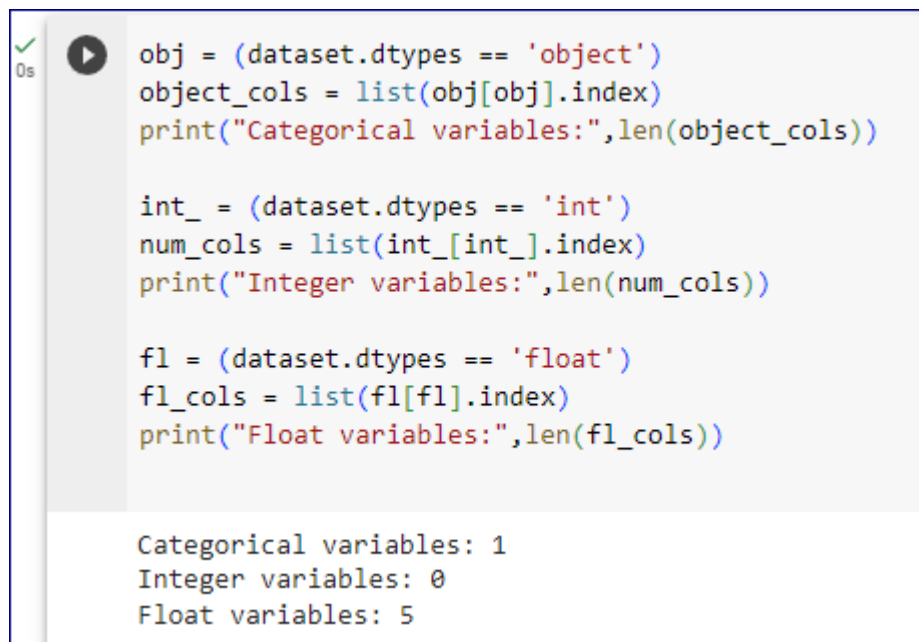
5000 rows x 7 columns
```

It creates a new DataFrame called 'new_dataset' by removing rows with missing values (NaN) from the original 'dataset' and then prints the 'new_dataset'.

DATA PREPROCESSING:

It is defined as Collection, manipulation, and processing of collected data for the required use. It is a task of converting data from a given form to a much more usable and desired form i.e. making it more meaningful and informative. Using Machine Learning algorithms, mathematical modelling and statistical knowledge, this entire process can be automated.

- Data Transformation
- Feature Selection
- Handling Outliers
- Normalization and Scaling
- Data Splitting
- Data Imputation



```
obj = (dataset.dtypes == 'object')
object_cols = list(obj[obj].index)
print("Categorical variables:", len(object_cols))

int_ = (dataset.dtypes == 'int')
num_cols = list(int_[int_].index)
print("Integer variables:", len(num_cols))

fl = (dataset.dtypes == 'float')
fl_cols = list(fl[fl].index)
print("Float variables:", len(fl_cols))
```

Categorical variables: 1
Integer variables: 0
Float variables: 5

This code is used to identify and categorize the columns in your dataset into three types: categorical, integer, and float variables.

Data preprocessing plays a significant role in the quality and reliability of data-driven projects. It sets the stage for accurate analysis, meaningful insights, and the development of predictive models in various domains, from finance and healthcare to marketing and more. Proper data preprocessing can make the difference between a successful and unsuccessful data-driven project.

The code you've provided is using scikit-learn's LabelEncoder to transform the categorical (object) columns in the dataset into numerical labels. This is a common preprocessing step when working with machine learning models that require numerical input data

```
✓ 0s ▶ from sklearn.preprocessing import LabelEncoder
labelencoder=LabelEncoder()
for columns in dataset.columns:
    dataset[columns]=labelencoder.fit_transform(dataset[columns])
print(dataset.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 6 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   Avg. Area House Age                   5000 non-null   int64
1   Avg. Area Number of Rooms             5000 non-null   int64
2   Avg. Area Number of Bedrooms          5000 non-null   int64
3   Area Population                       5000 non-null   int64
4   Price                                 5000 non-null   int64
5   Address                               5000 non-null   int64
dtypes: int64(6)
memory usage: 234.5 KB
None
```

Splitting the data:

To do this, you split your dataset into two main parts: a training set and a testing Set.

Training Dataset:

- The training dataset is a subset of your overall dataset that is used to train your machine learning model. It contains both the input features (often denoted as "X") and the corresponding target or output values (often denoted as "y").
- When you train a machine learning model, the algorithm learns patterns, relationships, and rules from the data in the training dataset. It uses this information to make predictions or classifications.
- The model adjusts its internal parameters during training to minimize the difference between its predictions and the actual target values in the training data. This process is known as model training or fitting.
- The training dataset is the data your model has seen and learned from. It is essential for building a predictive model.

Testing Dataset (or Validation Dataset):

- The testing dataset is another subset of your overall dataset that is kept separate from the training dataset. It is used to evaluate the performance of your trained machine learning model.
- The testing dataset also contains input features ("X") and their corresponding target values ("y"), but the model has not seen this data during training.
- Once the model is trained, it is tested on the testing dataset to assess how well it generalizes to new, unseen data. This evaluation provides an indication of the model's performance and whether it can make accurate predictions on real-world

data.

- The testing dataset helps you determine if your model has learned to make predictions or classifications that are not specific to the training data but instead can be applied to new, unseen examples.

The training dataset is used to teach the machine learning model, while the testing dataset is used to assess how well the model has learned and to estimate its performance on data it hasn't encountered before. Properly splitting the data into training and testing sets is crucial for avoiding overfitting, where the model memorizes the training data but fails to generalize to new data, and for ensuring that the model can make useful predictions in practice.

```
0s X = dataset[['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number of Rooms',  
             'Avg. Area Number of Bedrooms', 'Area Population']]  
Y = dataset['Price']  
  
0s [7] X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=101)
```

```
0s [8] Y_train.head()  
  
      3413      1.305210e+06  
      1610      1.400961e+06  
      3459      1.048640e+06  
      4293      1.231157e+06  
      1039      1.391233e+06  
      Name: Price, dtype: float64
```

```
0s [9] Y_train.shape  
  
      (4000,)  
  
0s [10] Y_test.head()  
  
      1718      1.251689e+06  
      2511      8.730483e+05  
      345      1.696978e+06  
      2521      1.063964e+06  
      54      9.487883e+05  
      Name: Price, dtype: float64
```

Standard scalar:

- Standard scaling is a preprocessing technique used in machine learning to standardize or normalize the feature values of a dataset.

- When you standard scale a feature, you transform it so that it has a mean (average) of 0 and a standard deviation of 1.

- This means that the values are rescaled so that they have a common scale and

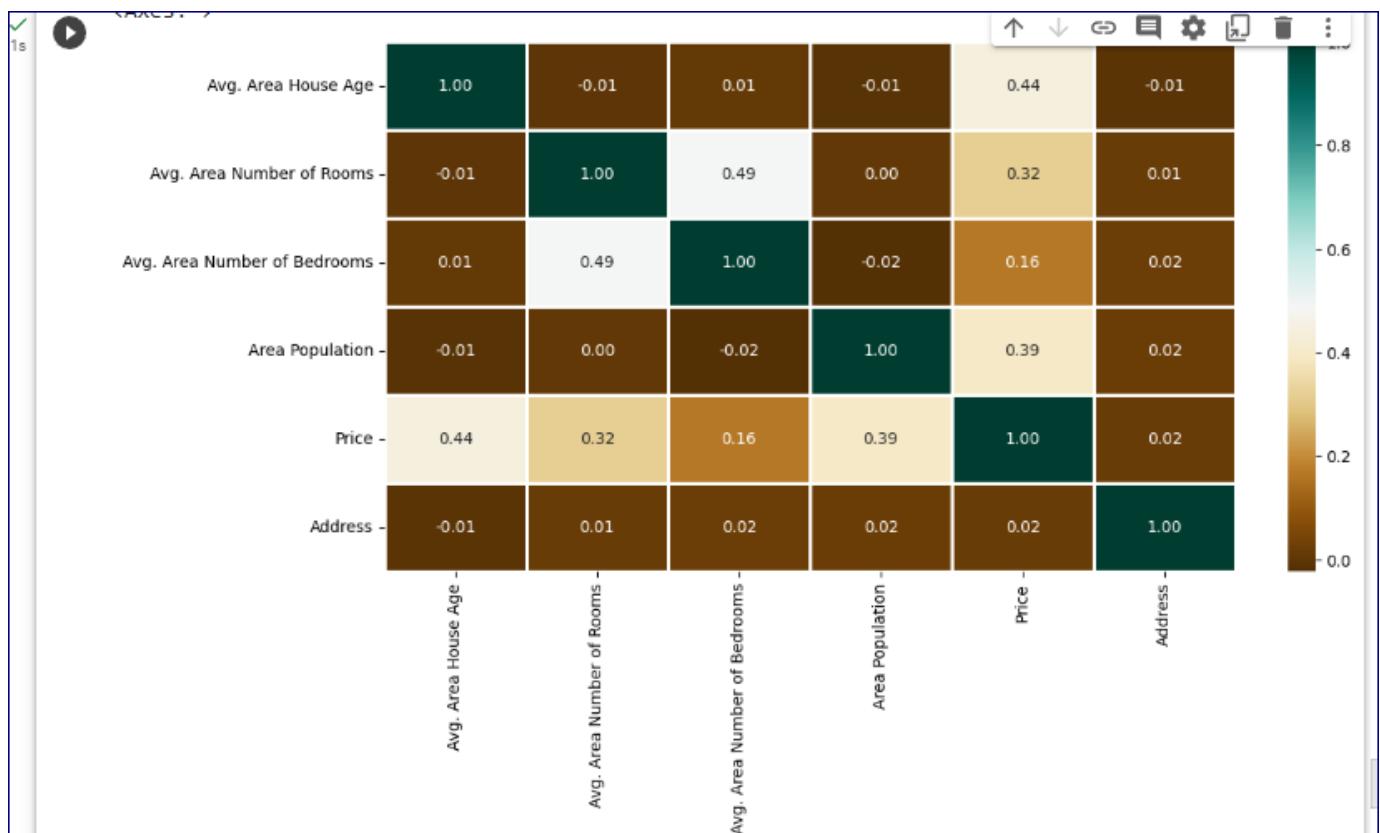
are centered around zero

```
✓ [12] sc = StandardScaler()  
0s X_train_scal = sc.fit_transform(X_train)  
X_test_scal = sc.fit_transform(X_test)
```

DATA ANALYSIS:

Data analysis is the process of inspecting, cleaning, transforming, and modeling data to discover useful information, draw meaningful conclusions, and support decision-making. It is a critical step in various fields, from business and science to healthcare and social sciences, where data-driven insights are essential for making informed choices.

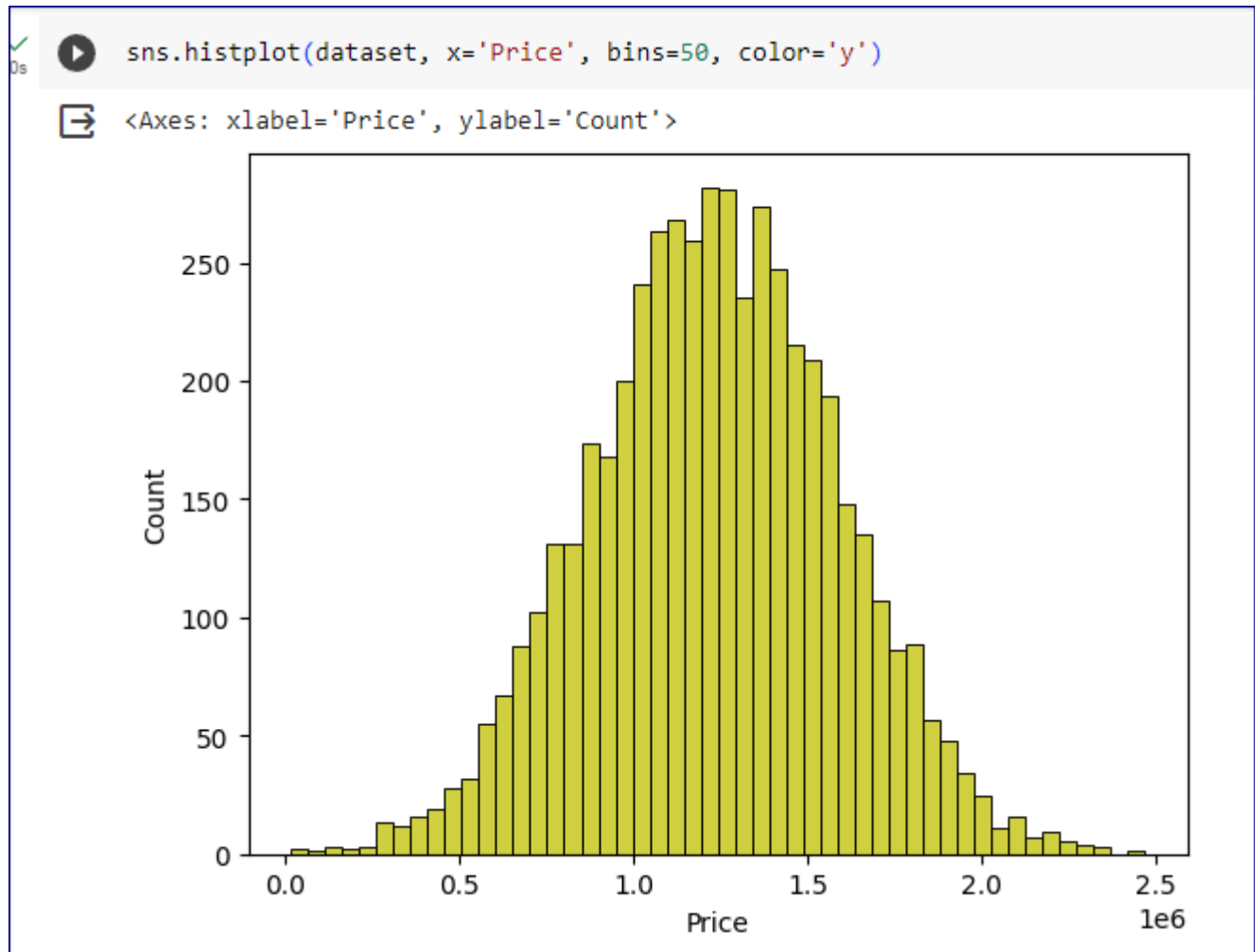
```
✓ 1s ▶ plt.figure(figsize=(12, 6))  
sns.heatmap(dataset.corr(),  
             cmap = 'BrBG',  
             fmt = '.2f',  
             linewidths = 2,  
             annot = True)
```



This code uses Matplotlib and Seaborn to create a heatmap of the correlation matrix of your dataset.

The code generates a heatmap that visually represents the correlation between pairs of variables in your dataset. Positive correlations are shown in one color, while negative correlations are shown in another color.

The degree of correlation is indicated by the color intensity. Heatmaps are a useful tool for exploring relationships between variables and identifying potential multicollinearity (high correlations) in the data, which can be important when building predictive models.

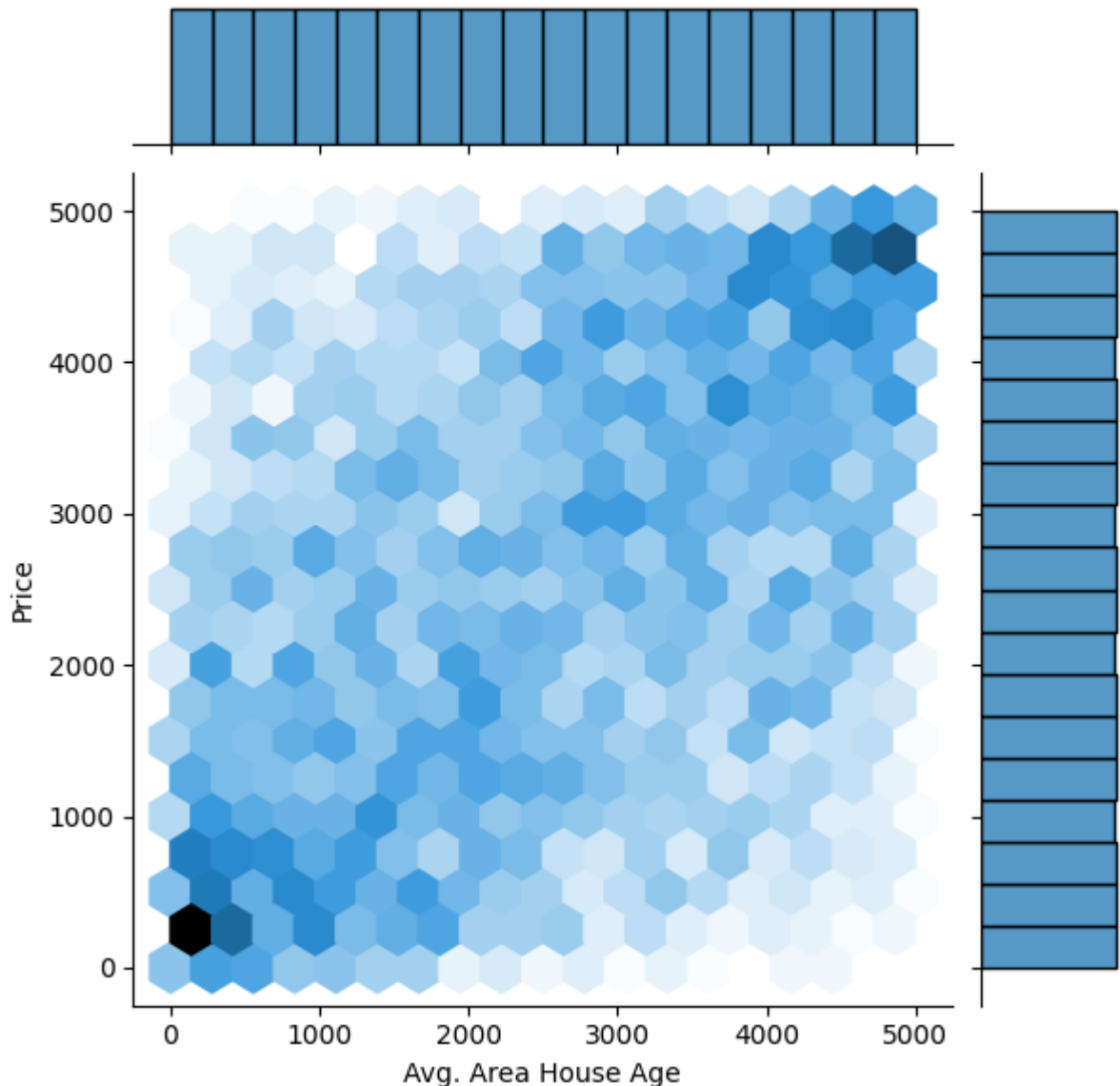


The code you've provided uses Seaborn to create a histogram plot of the 'Price' column in your dataset.

The code will generate a histogram of the 'Price' column, showing the distribution of house prices. The histogram will have 50 bins, and each bin represents a range of house prices. The height of the bars in each bin represents the frequency of houses falling within that price range. Histograms are useful for visualizing the distribution of a numerical variable and understanding its central tendency and spread. They can provide insights into the typical price range for houses in your dataset.

```
✓ 2s sns.jointplot(dataset, x='Avg. Area House Age', y='Price', kind='hex')
```

```
<seaborn.axisgrid.JointGrid at 0x7ab322e87580>
```

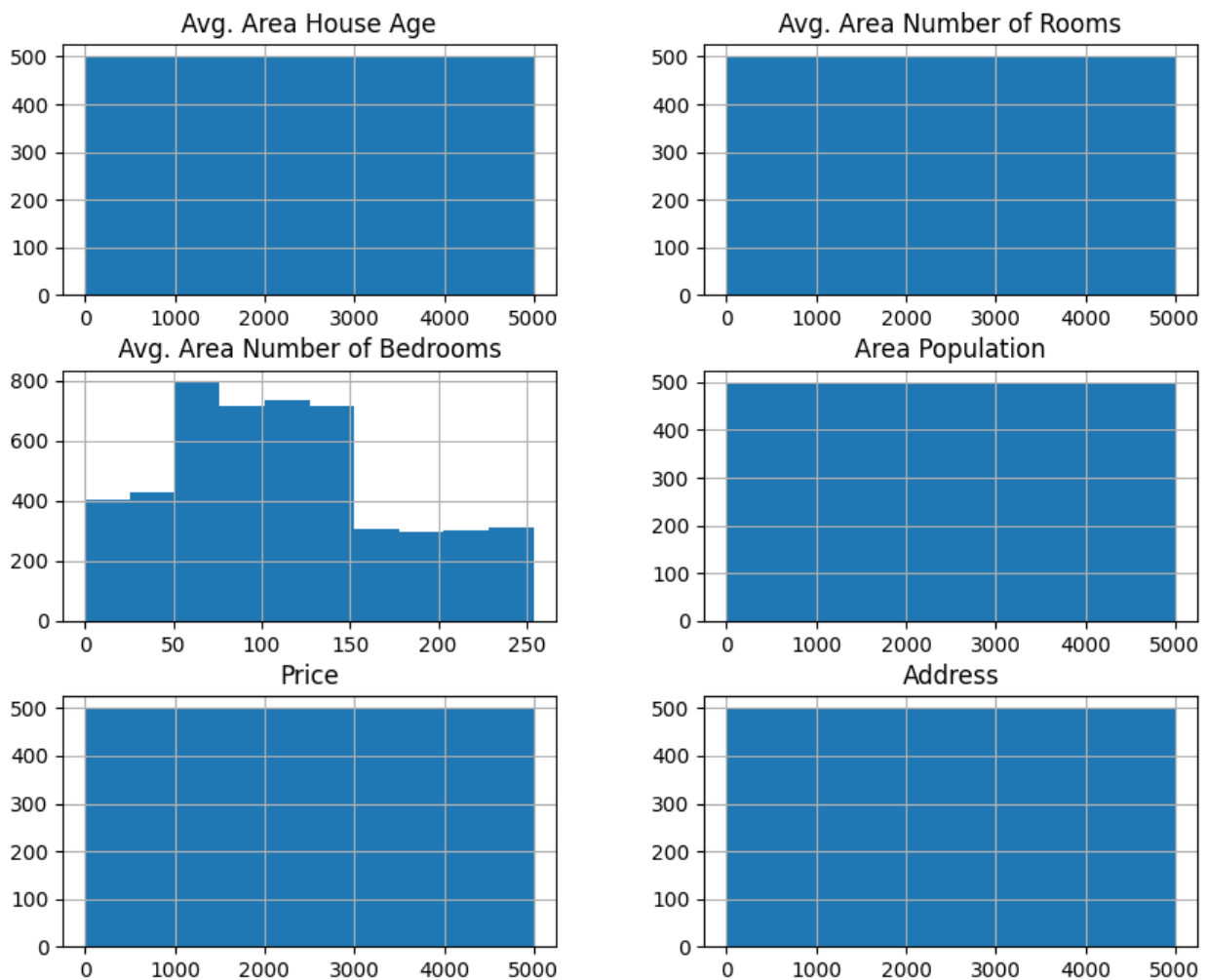


The code you've provided uses Seaborn to create a joint plot between the 'Avg. Area House Age' and 'Price' columns in your dataset. Specifically, it uses a hexbin plot for visualization.

The code will generate a joint plot that shows the relationship between the average area house age and house prices. The hexbin plot will represent the density of data points, with darker hexagons indicating areas where more data points are concentrated. This type of plot helps visualize the distribution and relationship between two numerical variables.


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

```
array([[<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'}>],  
       [<Axes: title={'center': 'Address'}>]], dtype=object)
```



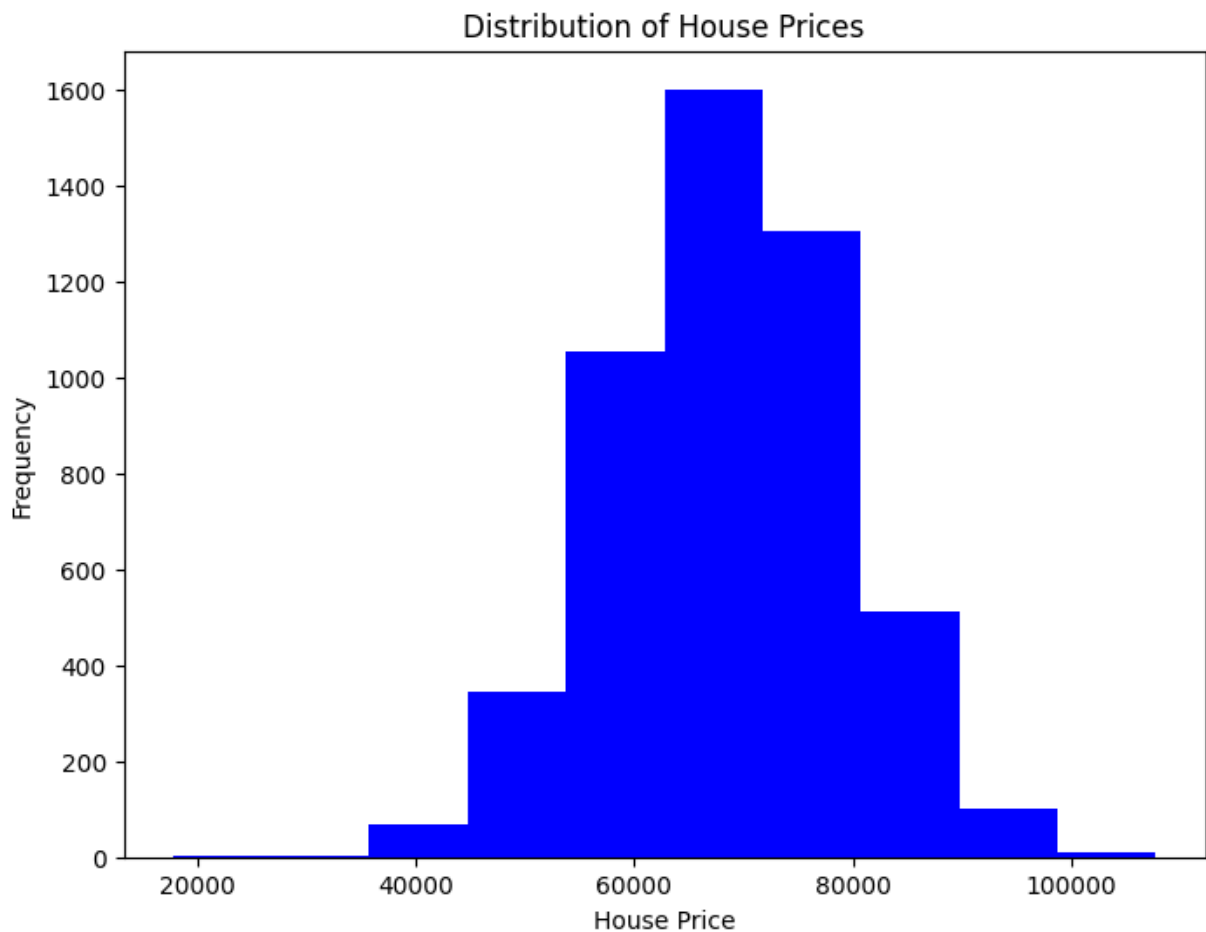
The code you've provided uses Seaborn to create a joint plot between the 'Avg. Area House Age' and 'Price' columns in your dataset. Specifically, it uses a hexbin plot for visualization.

The code will generate a grid of histograms, with each histogram representing the distribution of a numerical column in your dataset. This provides a quick visual overview of the data's distribution, which can be useful for understanding the range, spread, and central tendency of each numerical variable.

Each histogram in the grid will show the frequency (or count) of data points within different intervals (bins) of the variable. This visualization is a helpful initial step in exploring your data and identifying potential patterns or outliers in the numerical features.

```
import matplotlib.pyplot as plt

# Assuming 'data' is your dataset and 'house_price' is the target variable
plt.figure(figsize=(8, 6))
plt.hist(dataset['Avg. Area Income'], bins=10, color='blue')
plt.title('Distribution of House Prices')
plt.xlabel('House Price')
plt.ylabel('Frequency')
plt.show()
```



The code you provided uses the matplotlib library in Python to create a histogram of a dataset, specifically focusing on the 'Avg. Area Income' feature.

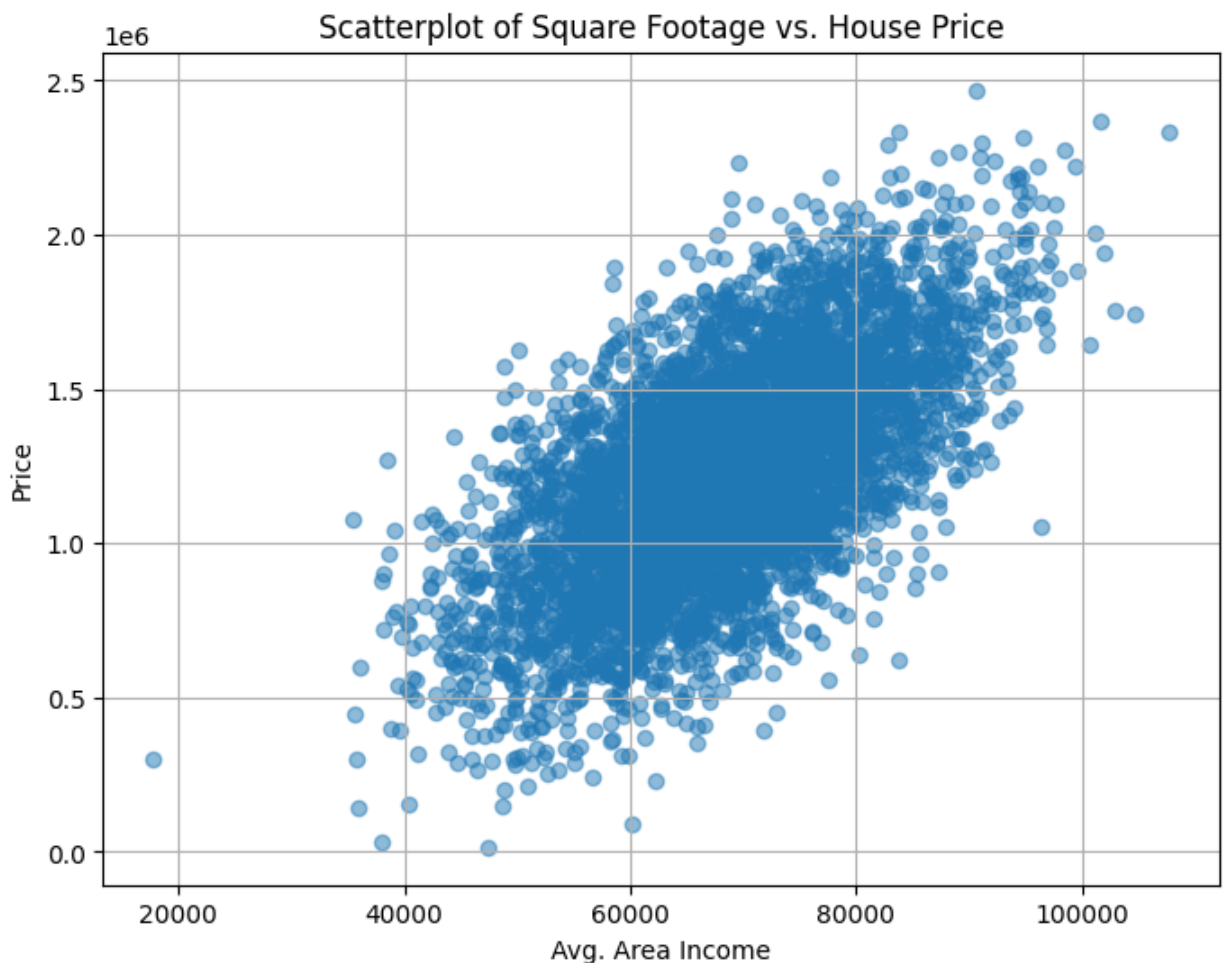
The code creates a histogram of the 'Avg. Area Income' feature from your dataset using the matplotlib library. It sets up the figure size, labels, and the number of bins for the histogram and then displays the plot. However, it might be a good idea to update the title and x-axis label to accurately reflect the data you're visualizing

✓
2s



```
import matplotlib.pyplot as plt

# Assuming 'data' is your dataset
plt.figure(figsize=(8, 6))
plt.scatter(dataset['Avg. Area Income'], dataset['Price'], alpha=0.5)
plt.title('Scatterplot of Square Footage vs. House Price')
plt.xlabel('Avg. Area Income')
plt.ylabel('Price')
plt.grid(True)
plt.show()
```



The code you provided uses the matplotlib library in Python to create a scatter plot of two variables from your dataset, 'Avg. Area Income' on the x-axis and 'Price' on the y-axis.

In summary, the code creates a scatter plot of 'Avg. Area Income' against 'Price' from your dataset using matplotlib. It sets up the figure size, labels for both axes, a title, and a grid for better visualization of the data points. Just ensure that the title accurately describes the data being visualized, as it currently mentions "Square Footage" instead of "Avg. Area Income."

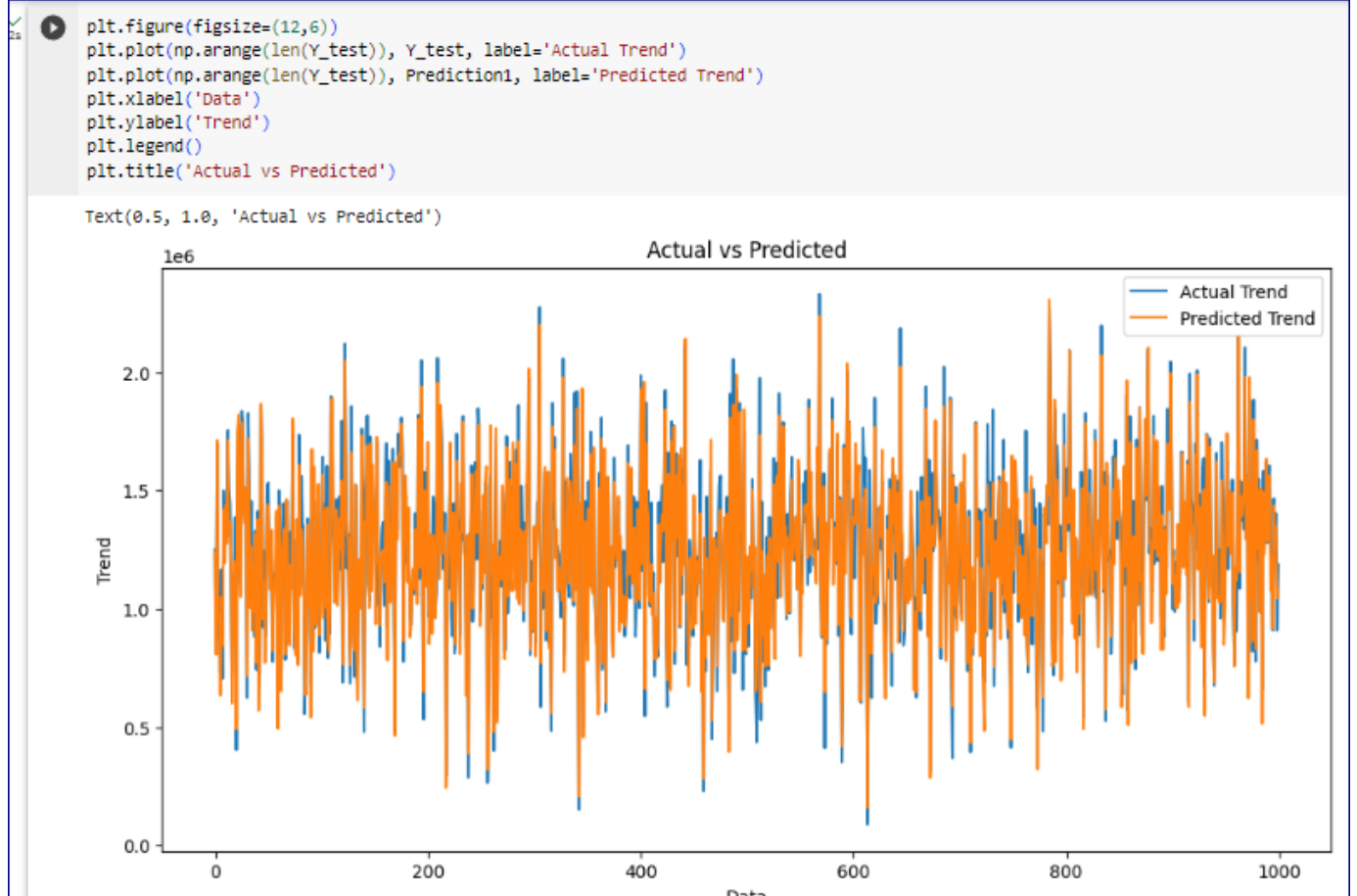
LINEAR REGRESSION MODEL FOR HOUSE PRICE PREDICTION:

Linear regression is a commonly used machine learning technique for predicting house prices. In this context, it's often referred to as "linear regression for house price prediction" or simply "house price prediction using linear regression"

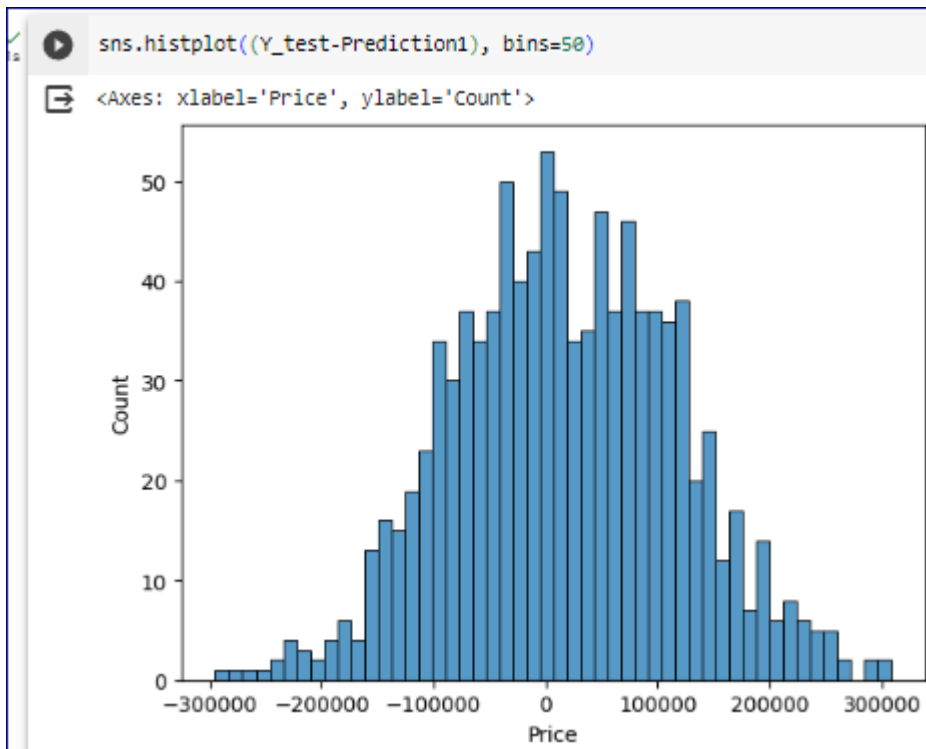
```
[13] model_lr=LinearRegression()  
  
[14] model_lr.fit(X_train_scal, Y_train)
```

▼ LinearRegression
LinearRegression()

```
Prediction1 = model_lr.predict(X_test_scal)
```



Once trained, the model can make predictions for new, unseen data. You input a value for x, and the model predicts the corresponding y



IF WE REMOVING ADDRESS COLUMN AND THE ACCURACY:

If you want to remove one column (feature) from the dataset and then predict the accuracy of the linear regression model, you can do the following:

- Remove the column you want to exclude.

- Train the model with the modified dataset.

- Evaluate the model's accuracy using a metric like R-squared.

```
[23] column_to_remove = 'Address'
```

```
[25] dataset = dataset.drop(column_to_remove, axis=1)
```

```
[27] X = dataset.drop('Price', axis=1) # Assuming 'Price' is the target variable
      y = dataset['Price']
```

```
[28] X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
[30] y_pred = model.predict(X_test)
```

```
[31] r2 = r2_score(y_test, y_pred)

      print(f"R-squared: {r2}")

      R-squared: 0.9179971706834289
```

Import Necessary Libraries:

The code starts by importing the required Python libraries, including NumPy for numerical operations, pandas for handling the dataset, scikit-learn for machine learning tools, and specifically LinearRegression and r2_score for linear regression modeling and evaluation.

Load the Dataset:

It reads the dataset from a CSV file named 'usa_housing.csv' using the pd.read_csv function. Make sure to replace 'usa_housing.csv' with the actual filename of your dataset.

Specify the Column to Be Removed:

You need to specify the name of the column you want to remove from the dataset. In the code, it's denoted as column_to_remove. You should replace 'ColumnNameToRemove' with the actual name of the column you want to exclude from the analysis.

Remove the Specified Column:

The specified column is removed from the dataset using the data.drop() method, with axis=1 indicating that you are dropping a column, not a row.

Split the Data:

The dataset is divided into features (X) and the target variable (y). The target variable, in this case, is 'Price'. The dataset is further split into training and testing sets using train_test_split from scikit-learn. The training set will be used to train the linear regression model, while the testing set is used to evaluate its performance.

Create and Train the Linear Regression Model:

An instance of the LinearRegression model is created, and it's then trained using the training data (X_train and y_train) with the model.fit() method.

Make Predictions on the Test Data:

The model makes predictions on the testing data using the model.predict() method, resulting in a set of predicted values (y_pred).

Evaluate the Model Using R-squared:

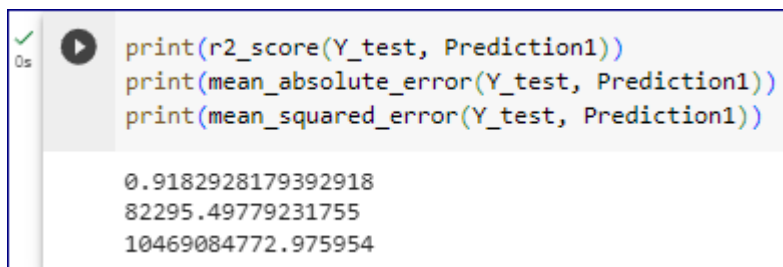
The code calculates the R-squared score for the model's predictions on the test data using the r2_score() function. R-squared (R^2) is a metric that quantifies how well the model explains the variance in the target variable. A higher R^2 score indicates a better fit of the model to the data.

Print R-squared Value:

The code prints the calculated R-squared score to the console, indicating how well the linear regression model performs after removing the specified column.

EVALUATION:

Common metrics for evaluating linear regression models include Mean Squared Error (MSE) and R-squared. MSE measures the average squared difference between predicted and actual values, while R-squared represents the proportion of the variance in the dependent variable that is predictable from the independent variables.



```
print(r2_score(Y_test, Prediction1))
print(mean_absolute_error(Y_test, Prediction1))
print(mean_squared_error(Y_test, Prediction1))
```

0.9182928179392918
82295.49779231755
10469084772.975954

Linear regression is a great starting point for many predictive modeling tasks, and it forms the foundation for more complex models. It's widely used in various fields due to its simplicity and interpretability.

CONCLUSION:

Creating a house price prediction project is a complex task that involves several phases, from dataset preprocessing and analysis to model development. Here's a high-level conclusion summarizing the key aspects of the project using a design thinking approach:

Empathize: Understanding the Problem

- The project begins with empathizing with the problem at hand, which is to predict house prices accurately.
- During this phase, you gather insights, conduct market research, and identify stakeholders' needs and requirements.

Define: Problem Statement and Scope

- Based on the insights gained, you define the problem statement, objectives, and scope of the project.
- In this phase, you decide what factors will be considered in the prediction model, such as location, square footage, number of bedrooms, and more.

Ideate: Data Collection and Preprocessing

- Gathering and preparing the dataset is a critical step. This involves data collection, cleaning, and preprocessing.

- Data preprocessing includes handling missing values, encoding categorical features, scaling, and feature selection.

Prototype: Model Development and Evaluation

- You create a prototype of the house price prediction model.
- Experiment with different algorithms and model parameters to find the best model.
- Employ evaluation metrics such as MAE, RMSE, R-squared, and cross-validation to assess model performance.
- Fine-tune the model to improve accuracy and robustness.

Test: Model Validation

- Test the model on a holdout dataset to ensure it generalizes well to new, unseen data.
- Perform a thorough analysis of the model's predictions, including residual analysis, feature importance, and validation against domain knowledge.

Implement: Deployment and User Interface

- If the model is satisfactory, prepare it for deployment.
- Create a user interface (web app, mobile app, etc.) to allow users to input property information and receive price predictions.

Feedback and Iterate: Continuous Improvement

- Gather user feedback and iterate on the model and the user interface.
- Continue to refine the model as new data becomes available and as user needs evolve.

Deliver: Final Product

- Deliver the final house price prediction application to users.
- Provide documentation and support for users and stakeholders.

Impact: Measure and Reflect

- After deployment, measure the impact of the project by assessing user satisfaction and monitoring the accuracy and performance of the model.
- Reflect on the lessons learned and areas for improvement.

In conclusion, a successful house price prediction project requires a holistic approach that combines data preprocessing, model development, and design thinking principles to meet the needs of users and stakeholders. It's an iterative process that aims to provide accurate and useful predictions to assist in real estate decision-making. Continuous improvement and adaptation to changing market conditions are crucial for the long-term success of such a project.

