PROJECT 9: HOUSE PRICE PREDICTION ANALYSIS

USING MACHINE LEARNING

PHASE 5

Project Title: Predicting House Prices: A Machine Learning Approach

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

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.

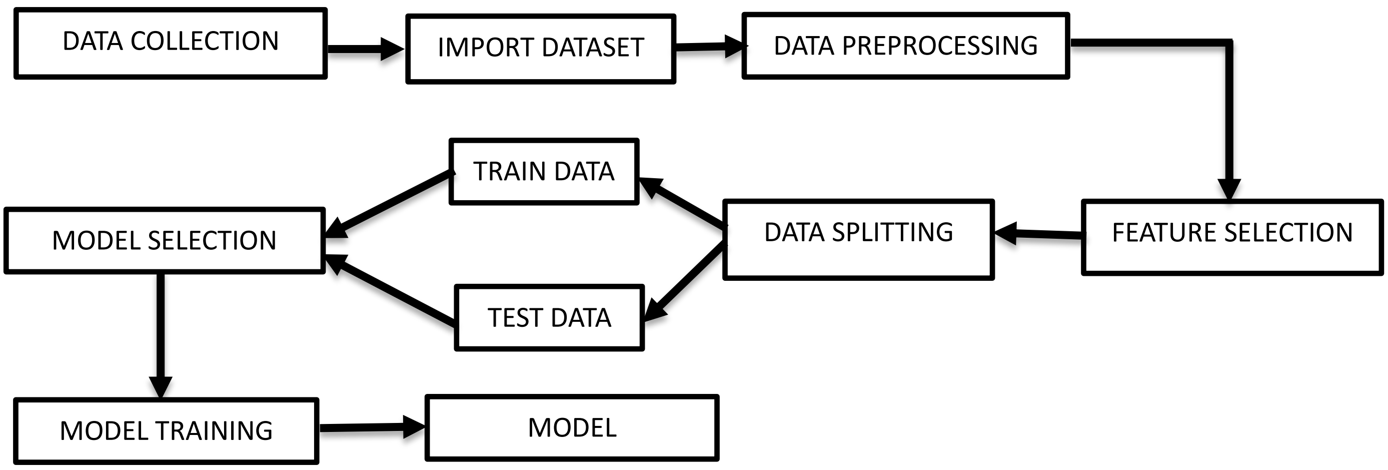
Phase 1: PROBLEM THINKING AND DESIGN THINKING

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:

Flow diagram:



1.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 rep resented 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.

2.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 Min-Max 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.

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.

3.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.

4.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.

Support Vector Machine: Support Vector Machines (SVM) for classification, SVR aims to find a hyperplane that best fits the data while minimizing the error between the predicted and actual values.

Random Forest: Random forest is an ensemble learning method that combines the predictions of multiple decision trees to improve accuracy and robustness in regression tasks.

XG Boost:XG Boost is to create a strong predictive model by combining the outputs of multiple weak models (typically decision trees) in an additive manner.

5. 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.

6. 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.

The Phase 1 gives the specific implementation details and choice of algorithms may vary depending on the dataset and the goals of the project.

Phase 2: INNOVATION

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

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.

<u>Phase 3: DEVELOPMENT PART 1</u>

COLAB LINK:
https://colab.research.google.com/drive/1piK4JcVll0DN8yPCjxJ1Cgc5IRyknG4l?usp=sharing

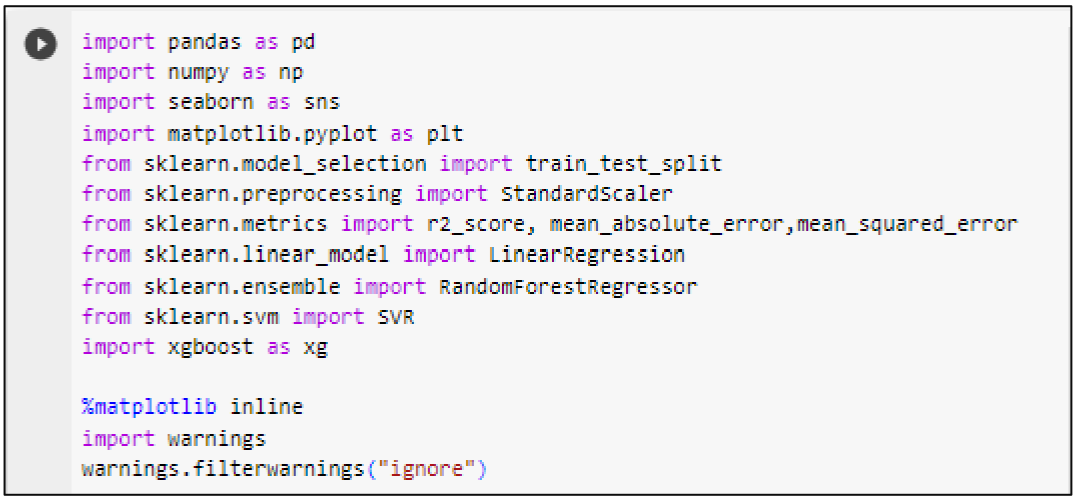
PROGRAM:

Loading the dataset:

Loading data is a crucial step in any data analysis or machine learning task. It involves bringing external datasets into your programming environment so that you can manipulate, analyze, and draw insights from the data.

❖ <u>Importing Libraries:</u>

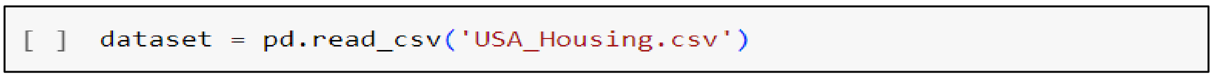
- **Pandas:** Pandas, the abbreviation for **Pan**el-**Da**ta, is a library for representing data on a data-frame.
- **Numpy:** Numpy is a math library that supports many operations on arrays, from simple to complex.
- **Seaborn:** Seaborn is a python library for data visualization. It is built on top of matplotlib.
- **Matplotlib:** Matplotlib supports most of the basic plots that we need when starting with data science.
- **Scikit learn:** It is a simple and very fast tool for predictive data analysis and statistically modeling.
- **XG Boost:** XGBoost, short for "Extreme Gradient Boosting," is a powerful and popular machine learning algorithm, is widely used for both classification and regression tasks.
- **Warnings:** The warnings package in Python is part of the Python Standard Library and is used to issue warning messages in your code.



```python
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.ensemble import RandomForestRegressor
from sklearn.svm import SVR
import xgboost as xg

%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
```

❖ Loading the dataset:

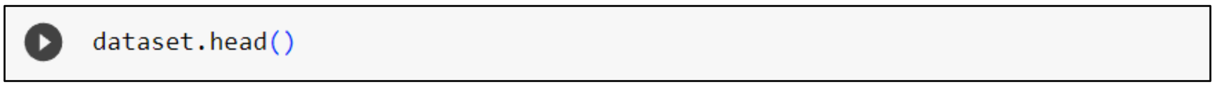
Use Pandas to load the dataset into a DataFrame.



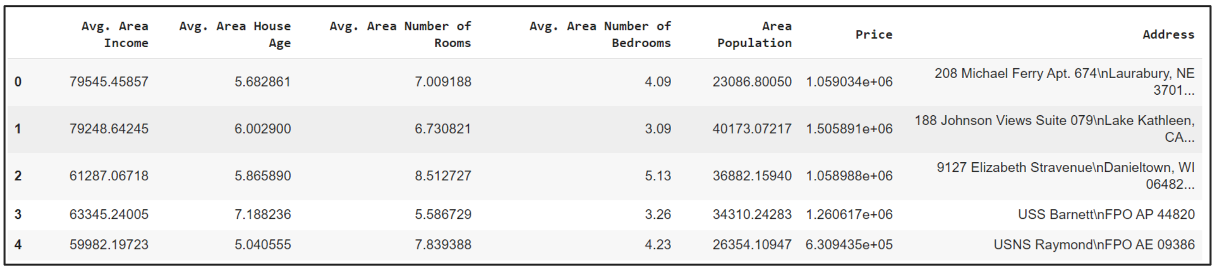
```
[ ]  dataset = pd.read_csv('USA_Housing.csv')
```

Data Exploration:

Check out the first few rows using dataset.head().



```
dataset.head()
```



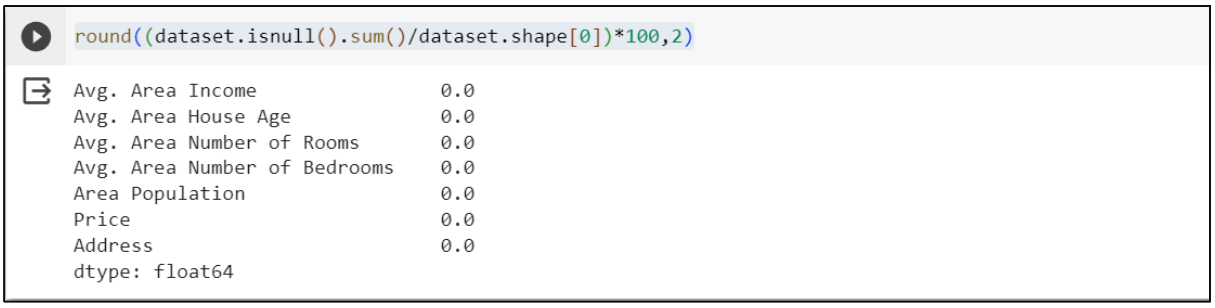
	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price	Address
0	79545.45857	5.682861	7.009188	4.09	23086.80050	1.059034e+06	208 Michael Ferry Apt. 674\nLaurabury, NE 3701...
1	79248.64245	6.002900	6.730821	3.09	40173.07217	1.505891e+06	188 Johnson Views Suite 079\nLake Kathleen, CA...
2	61287.06718	5.865890	8.512727	5.13	36882.15940	1.058988e+06	9127 Elizabeth Stravenue\nDanieltown, WI 06482...
3	63345.24005	7.188236	5.586729	3.26	34310.24283	1.260617e+06	USS Barnett\nFPO AP 44820
4	59982.19723	5.040555	7.839388	4.23	26354.10947	6.309435e+05	USNS Raymond\nFPO AE 09386

Data Preprocessing techniques:

Data preprocessing involves cleaning, transforming, and organizing raw data into a format that can be effectively used for modeling and analysis. Here are some common data preprocessing techniques:

❖ **Data cleaning:**

- **Handling missing data**: You can remove or impute missing values using techniques like mean, median, mode imputation, or more advanced methods like regression imputation.

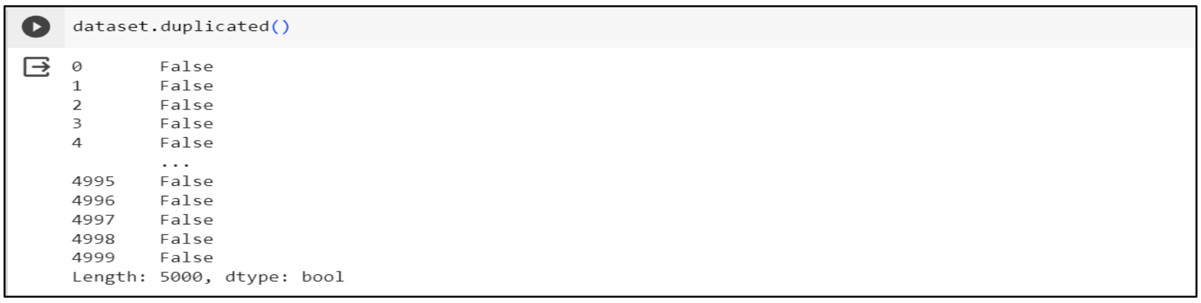


```
round((dataset.isnull().sum()/dataset.shape[0])*100,2)

Avg. Area Income                0.0
Avg. Area House Age             0.0
Avg. Area Number of Rooms       0.0
Avg. Area Number of Bedrooms    0.0
Area Population                 0.0
Price                           0.0
Address                         0.0
dtype: float64
```

For our dataset, since there is no missing values ,no need to use the data handling techniques like mean, median, mode.

- **Outlier detection and treatment:** To identify the outliers, we use techniques like describe(), info(), duplicated() and to handle the outliers, we use techniques like percentiles, z-score using statistical methods or domain knowledge.

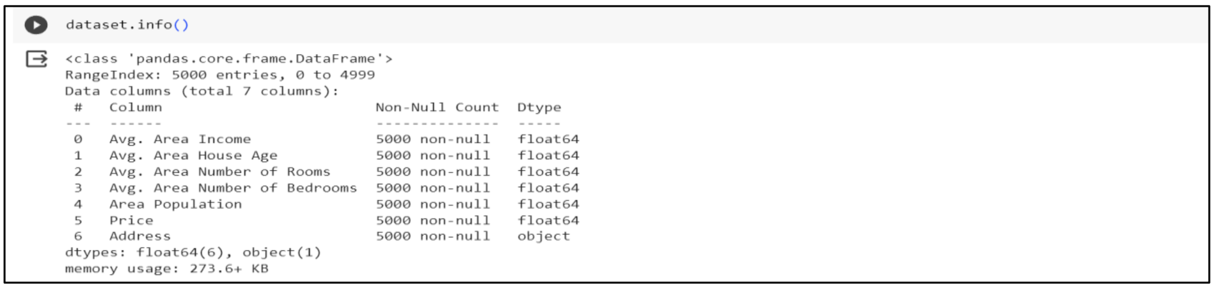
To find if there is any duplicates in the dataset, we use duplicated().



```
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
```

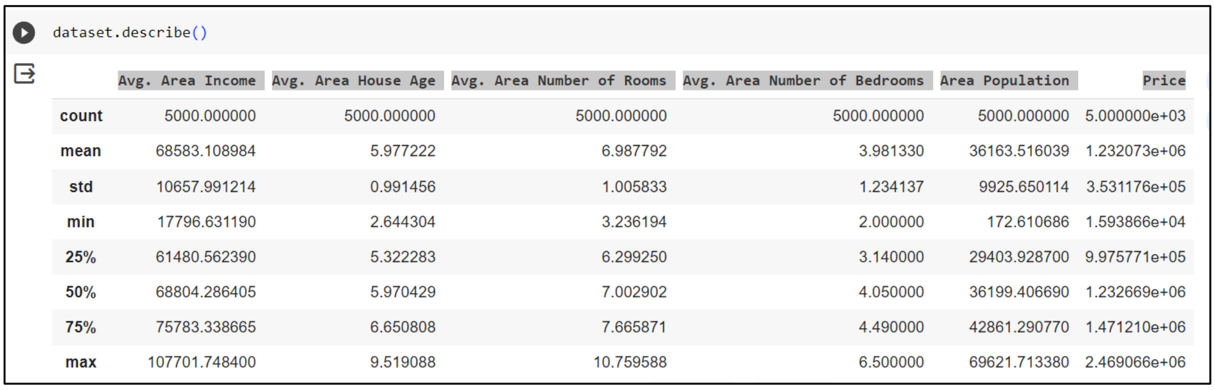
To get an overview of data types and missing values, we use info().



```
dataset.info()
```

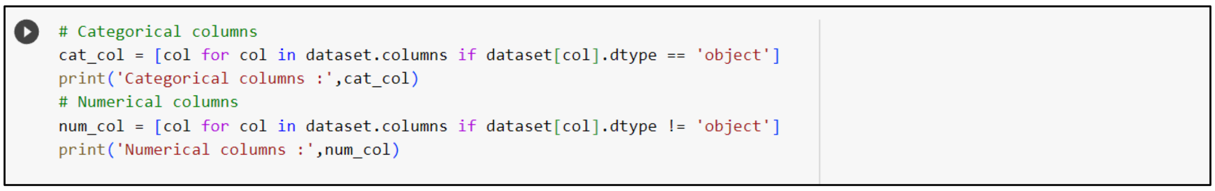
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 7 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   Avg. Area Income              5000 non-null   float64
 1   Avg. Area House 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
```

We use describe(), to get statistical summaries.

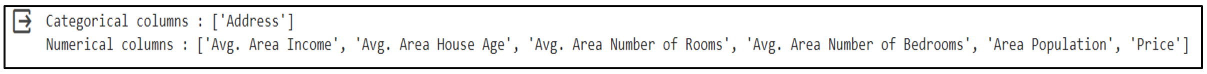


```
dataset.describe()
```

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price
count	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5.000000e+03
mean	68583.108984	5.977222	6.987792	3.981330	36163.516039	1.232073e+06
std	10657.991214	0.991456	1.005833	1.234137	9925.650114	3.531176e+05
min	17796.631190	2.644304	3.236194	2.000000	172.610686	1.593866e+04
25%	61480.562390	5.322283	6.299250	3.140000	29403.928700	9.975771e+05
50%	68804.286405	5.970429	7.002902	4.050000	36199.406690	1.232669e+06
75%	75783.338665	6.650808	7.665871	4.490000	42861.290770	1.471210e+06
max	107701.748400	9.519088	10.759588	6.500000	69621.713380	2.469066e+06

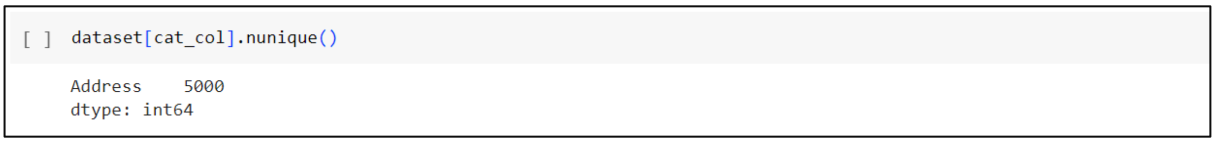


```
# Categorical columns
cat_col = [col for col in dataset.columns if dataset[col].dtype == 'object']
print('Categorical columns :',cat_col)
# Numerical columns
num_col = [col for col in dataset.columns if dataset[col].dtype != 'object']
print('Numerical columns :',num_col)
```



```
Categorical columns : ['Address']
Numerical columns : ['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number of Rooms', 'Avg. Area Number of Bedrooms', 'Area Population', 'Price']
```

This nunique() function in pandas is used to count the number of unique values in a dataframe.



```
[ ]  dataset[cat_col].nunique()
```

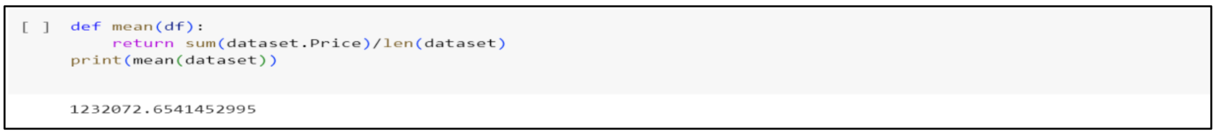
```
Address    5000
dtype: int64
```

Some statistical techniques to handle the missing values and the outliers.

- Mean
- Variance
- Percentile

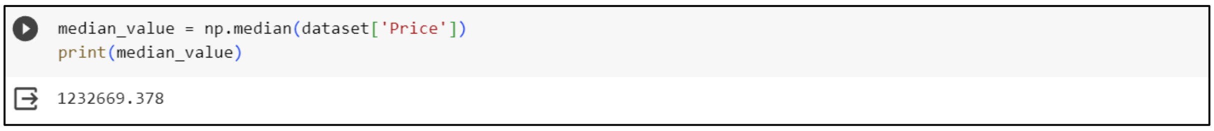
Standard deviation
Median
Mode

The **mean** is a measure of central tendency that represents the average value of a set of numbers.



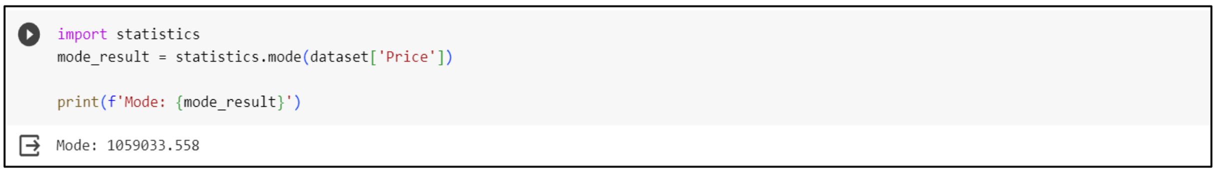
```
[ ] def mean(df):
        return sum(dataset.Price)/len(dataset)
    print(mean(dataset))

    1232072.6541452995
```

The **median** is a measure of central tendency in a dataset, representing the middle value when the data is sorted in ascending or descending order.



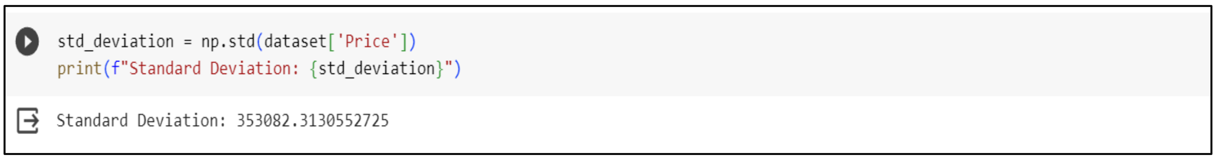
```
median_value = np.median(dataset['Price'])
print(median_value)

1232669.378
```

The **"mode"** is a measure of central tendency, similar to mean (average) and median. (appears most frequently in a particular column or variable).



```
import statistics
mode_result = statistics.mode(dataset['Price'])

print(f'Mode: {mode_result}')

Mode: 1059033.558
```

Standard deviation is a measure of the amount of variation or dispersion in a set of values.



```
std_deviation = np.std(dataset['Price'])
print(f"Standard Deviation: {std_deviation}")

Standard Deviation: 353082.3130552725
```

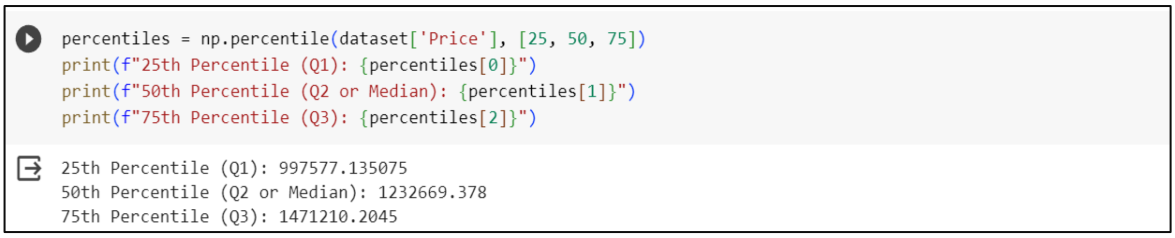
Percentiles are a statistical concept used to describe the relative standing of a particular value within a dataset.

Quartiles:

- Q1 (25th percentile): The value below which 25% of the data falls.
- Q2 (50th percentile, median): The value below which 50% of the data falls.
- Q3 (75th percentile): The value below which 75% of the data falls.

Percentile Ranks:

- The nth percentile is the value below which n% of the data falls.



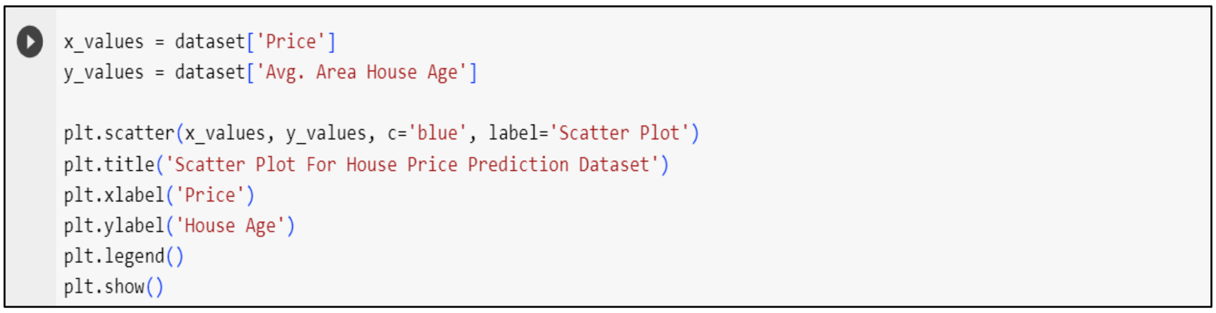
```
percentiles = np.percentile(dataset['Price'], [25, 50, 75])
print(f"25th Percentile (Q1): {percentiles[0]}")
print(f"50th Percentile (Q2 or Median): {percentiles[1]}")
print(f"75th Percentile (Q3): {percentiles[2]}")
```

```
25th Percentile (Q1): 997577.135075
50th Percentile (Q2 or Median): 1232669.378
75th Percentile (Q3): 1471210.2045
```

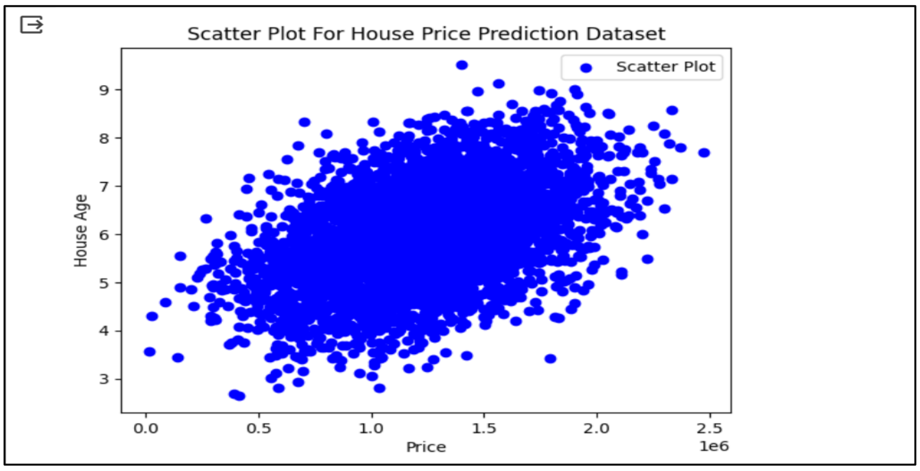
Data Visualization Techniques:

Data visualization helps you to understand the data, identify trends, outliers, and relationships between different variables. Here are some commonly used data visualization techniques :

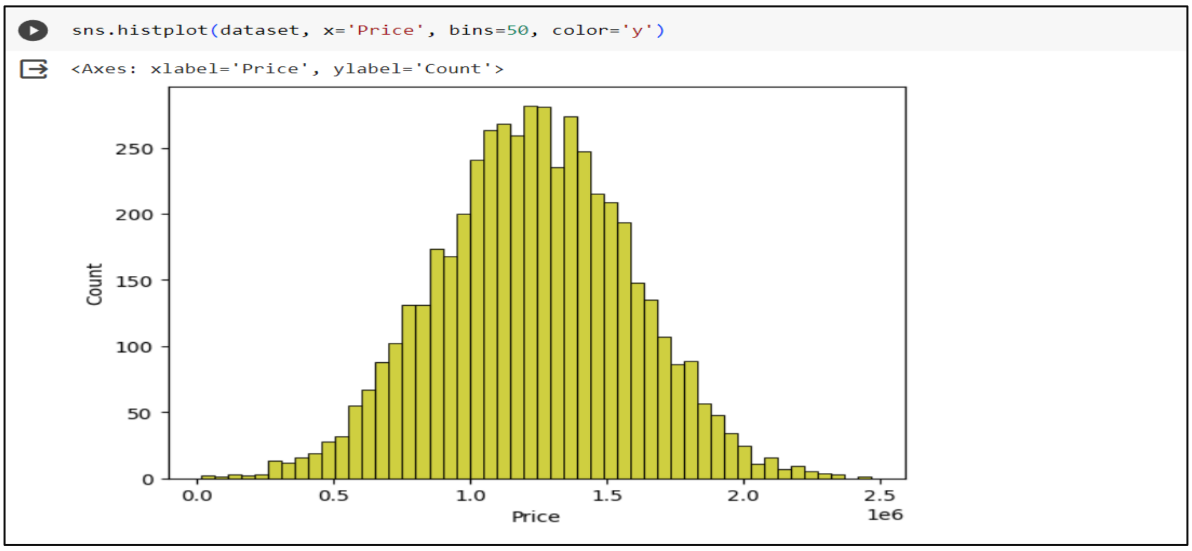
- **Scatter Plot:** Create scatter plots to explore the relationship between variables like square footage, number of bedrooms, or location and house prices. This can help identify patterns and outliers.



```
x_values = dataset['Price']
y_values = dataset['Avg. Area House Age']

plt.scatter(x_values, y_values, c='blue', label='Scatter Plot')
plt.title('Scatter Plot For House Price Prediction Dataset')
plt.xlabel('Price')
plt.ylabel('House Age')
plt.legend()
plt.show()
```



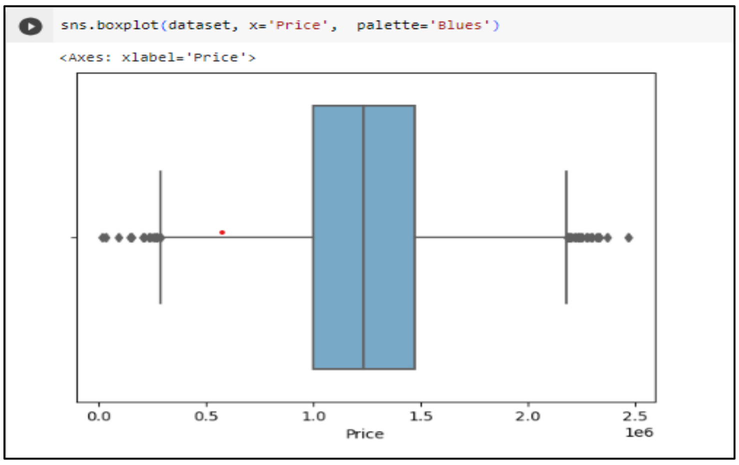
- **Histograms:** Use histograms to visualize the distribution of house prices.It helps you understand the central tendency and spread of the prices.



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

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

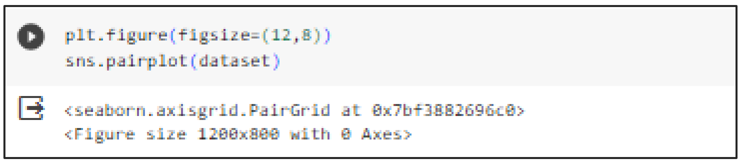
- **Box Plot:** Box plots are useful for visualizing the distribution of house prices by different categories, such as the number of bedrooms or the neighbourhood.



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

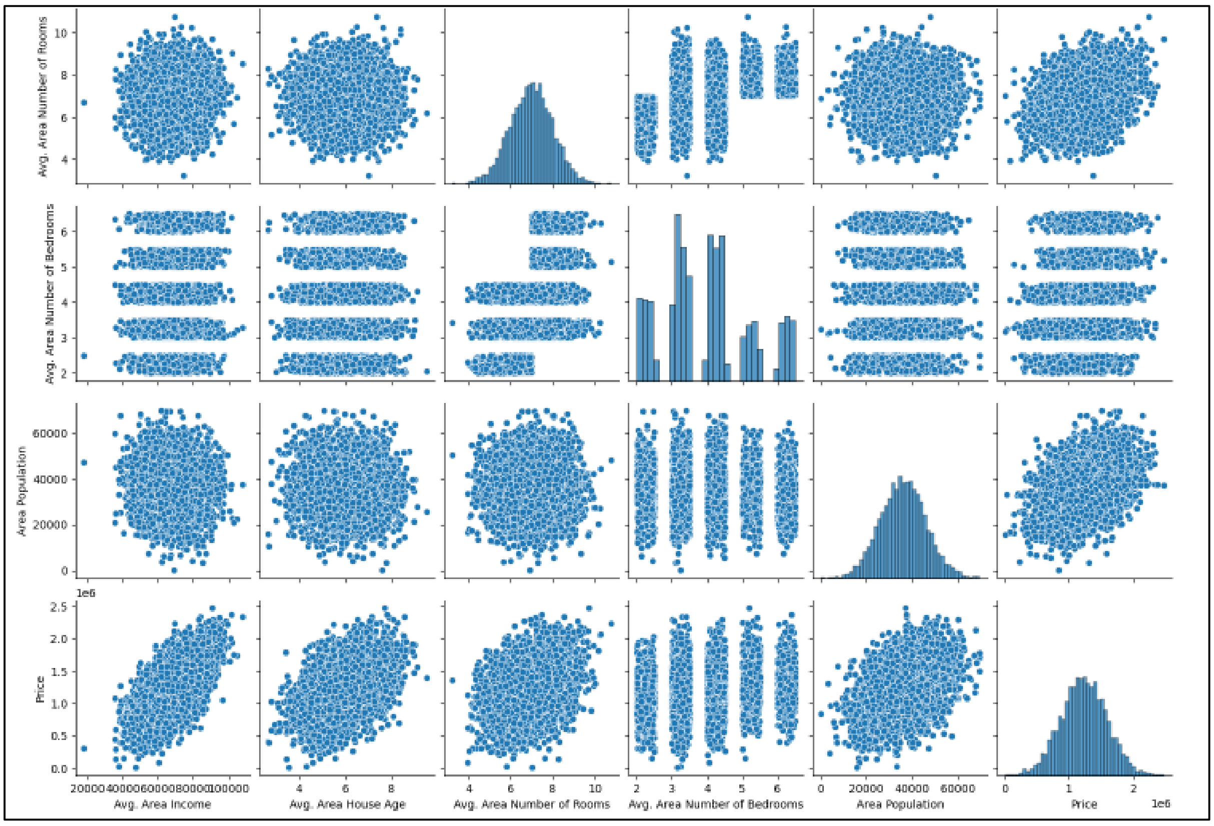
```
<Axes: xlabel='Price'>
```

- **Pairplot**: A pair plot displays scatter plots for multiple variables in your dataset. It's useful for identifying relationships and correlations among multiple features.

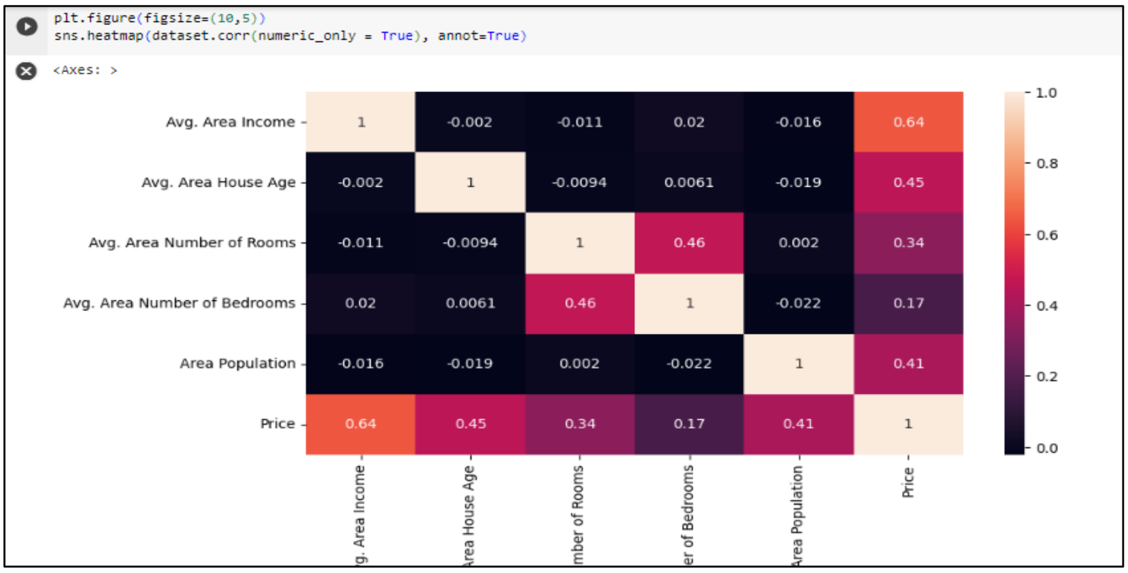


```
plt.figure(figsize=(12,8))
sns.pairplot(dataset)
```

```
<seaborn.axisgrid.PairGrid at 0x7bf3882696c0>
<Figure size 1200x800 with 0 Axes>
```



- **Heatmap:** Heatmaps help to visualize the relationships and patterns in the data, which can be useful for understanding the factors that influence house prices.



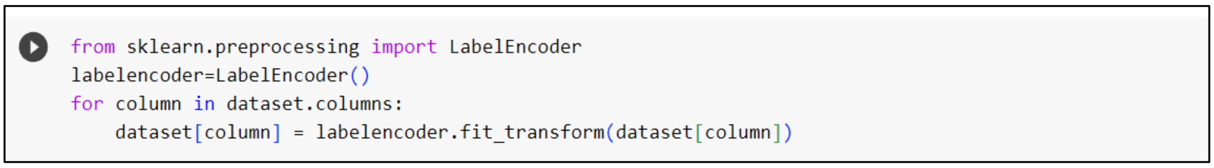
```
plt.figure(figsize=(10,5))
sns.heatmap(dataset.corr(numeric_only = True), annot=True)
```

```
<Axes: >
```

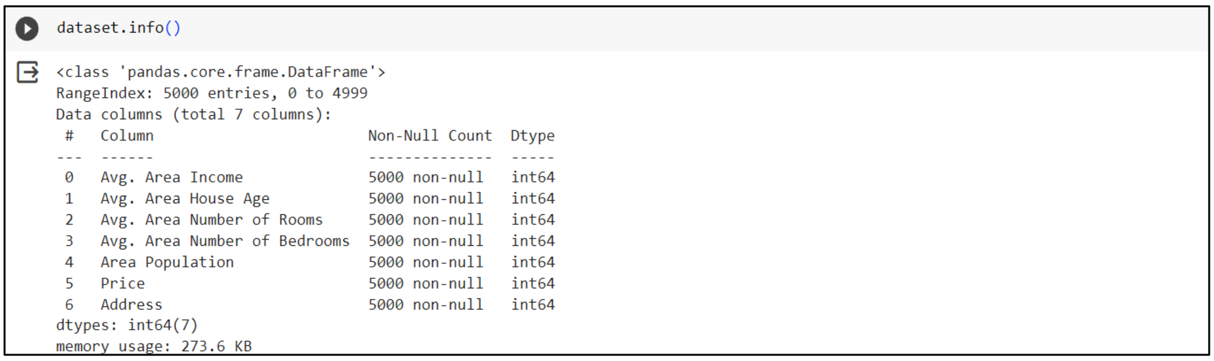
	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price
Avg. Area Income	1	-0.002	-0.011	0.02	-0.016	0.64
Avg. Area House Age	-0.002	1	-0.0094	0.0061	-0.019	0.45
Avg. Area Number of Rooms	-0.011	-0.0094	1	0.46	0.002	0.34
Avg. Area Number of Bedrooms	0.02	0.0061	0.46	1	-0.022	0.17
Area Population	-0.016	-0.019	0.002	-0.022	1	0.41
Price	0.64	0.45	0.34	0.17	0.41	1

Using LabelEncoder:

Label encoder is often used to convert the categorical values like address to the respective numerical values. These label encoder is available on the sklearn.preprocessing package.



```
from sklearn.preprocessing import LabelEncoder
labelencoder=LabelEncoder()
for column in dataset.columns:
    dataset[column] = labelencoder.fit_transform(dataset[column])
```



```
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   int64
 1   Avg. Area House Age           5000 non-null   int64
 2   Avg. Area Number of Rooms     5000 non-null   int64
 3   Avg. Area Number of Bedrooms  5000 non-null   int64
 4   Area Population               5000 non-null   int64
 5   Price                         5000 non-null   int64
 6   Address                       5000 non-null   int64
dtypes: int64(7)
memory usage: 273.6 KB
```

FEATURE SELECTION:

A). Feature Importance-based Selection (using Random Forest Regressor):

1.Random Forest Regressor: A Random Forest Regressor is an ensemble machine learning model that uses multiple decision trees to make predictions.

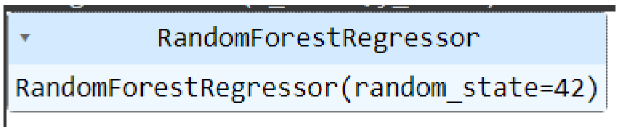
Training the Random Forest Regressor:

CODE:

regressor = RandomForestRegressor(n_estimators=100, random_state=42)

regressor.fit(X_train, y_train)

In this code, a Random Forest Regressor is created with 100 trees (n_estimators=100) for the ensemble, and a random seed (random_state=42) is set for reproducibility.It's trained using the training data X_train (features) and y_train (target).



```
RandomForestRegressor
RandomForestRegressor(random_state=42)
```

2. Getting Feature Importances:

CODE:

*feature_importances = regressor.feature_importances_*

After training, we access the feature importances using the feature_importances_ attribute of the regressor.

3. Creating a DataFrame for Visualization:

CODE:

```
feature_importance_df = pd.DataFrame({'Feature': X_train.columns, 'Importance': feature_importances})

feature_importance_df = feature_importance_df.sort_values(by='Importance', ascending=False)
```

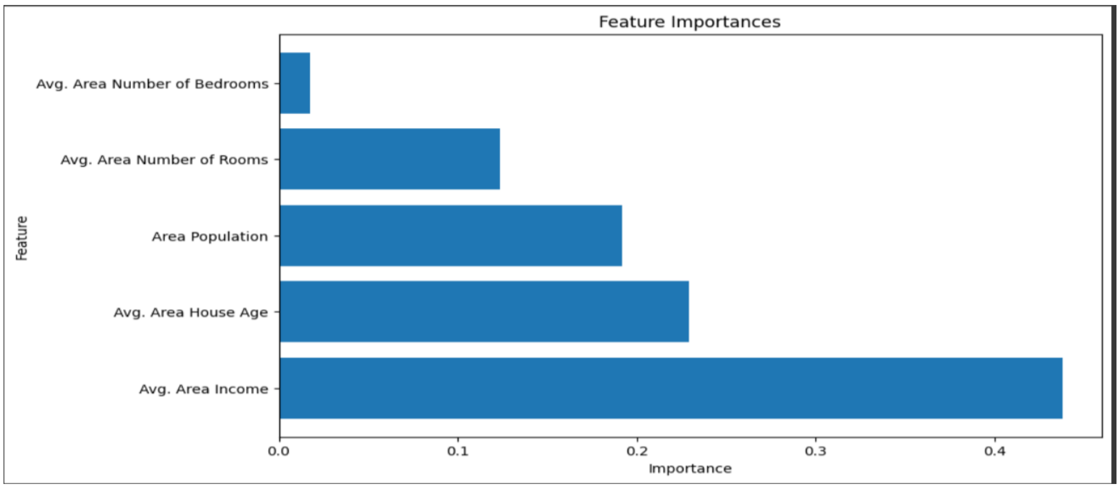
A DataFrame feature_importance_df is created to organize the feature names and their respective importances. It's sorted in descending order based on feature importances.

4. Visualizing Feature Importances:

CODE:

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

plt.barh(feature_importance_df['Feature'], feature_importance_df['Importance'])

plt.title('Feature Importances')

plt.xlabel('Importance')

plt.ylabel('Feature')

plt.show()
```

A horizontal bar plot is generated to visualize the feature importances. This shows which features have the most significant impact on the regression model's predictions.



B). Correlation-based Selection:

1.Calculating the Correlation Matrix:

CODE:

```
corr_matrix = X_train.corr()
```

The code calculates a correlation matrix corr_matrix for the features in your training data (X_train).

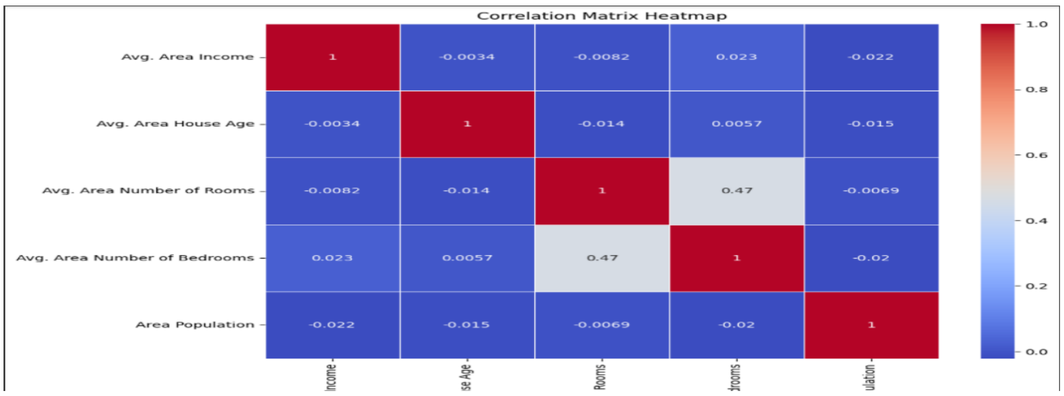
2. Creating a Heatmap for Visualization Correlation Matrix:

CODE:

```
plt.figure(figsize=(10, 8))

sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', linewidths=0.5)

plt.title('Correlation Matrix Heatmap')

plt.show()
```

A heatmap is generated to visualize the correlations between features. This heatmap uses color coding to represent the strength and direction of the correlations. It can help to identify relationships between features.

OUTPUT:



3. Selecting the top k features based on mutual information:

CODE:

```
selector = SelectKBest(score_func=mutual_info_regression, k=3)

X_train_selected = selector.fit_transform(X_train, y_train)
```

The code shows the performance of feature selection using SelectKBest. In this case, it selects the top 5 features based on mutual information with the target variable. The selected features are stored in X_train_selected, and can access them using X_train.columns[selector.get_support()].

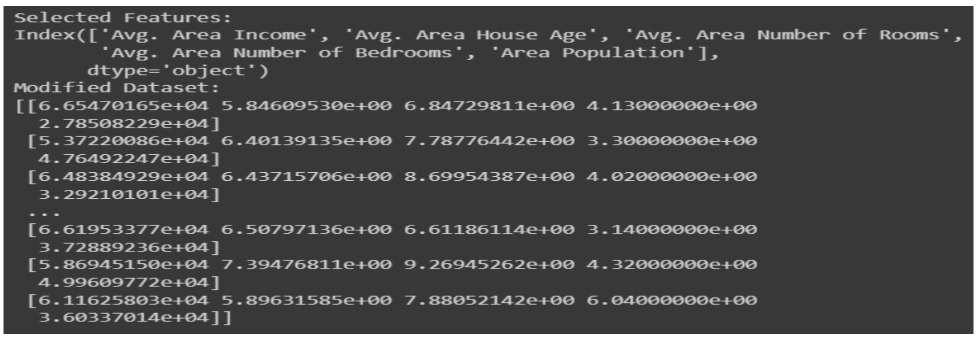
Feature Selection using SelectKBest:

CODE:

```
from sklearn.feature_selection import SelectKBest, f_regression

from sklearn.model_selection import train_test_split

import pandas as pd

data = pd.read_csv('/content/dataset.csv.csv')

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

y = data[['Price']]
```

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

k = 5

selector = SelectKBest(score_func=f_regression, k=k)

X_train_selected = selector.fit_transform(X_train, y_train)

selected_feature_indices = selector.get_support(indices=True)

selected_features = X_train.columns[selected_feature_indices]

X_train.columns[selector.get_support()]

selected_feature_indices = selector.get_support()

selected_features = X_train.columns[selected_feature_indices]

print("Selected Features:")

print(selected_features)

print("Modified Dataset:")

print(X_train_selected)
```

OUTPUT:

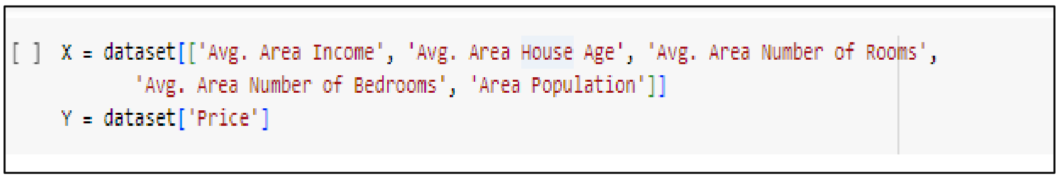


```
Selected Features:
Index(['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number of Rooms',
       'Avg. Area Number of Bedrooms', 'Area Population'],
      dtype='object')
Modified Dataset:
[[6.65470165e+04 5.84609530e+00 6.84729811e+00 4.13000000e+00
  2.78508229e+04]
 [5.37220086e+04 6.40139135e+00 7.78776442e+00 3.30000000e+00
  4.76492247e+04]
 [6.48384929e+04 6.43715706e+00 8.69954387e+00 4.02000000e+00
  3.29210101e+04]
 ...
 [6.61953377e+04 6.50797136e+00 6.61186114e+00 3.14000000e+00
  3.72889236e+04]
 [5.86945150e+04 7.39476811e+00 9.26945262e+00 4.32000000e+00
  4.99609772e+04]
 [6.11625803e+04 5.89631585e+00 7.88052142e+00 6.04000000e+00
  3.60337014e+04]]
```

Phase 4: DEVELOPMENT PART 2

Splitting the data:

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

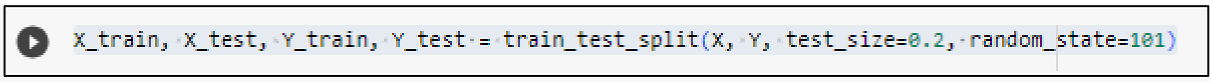


```
[ ] X = dataset[['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number of Rooms',
          'Avg. Area Number of Bedrooms', 'Area Population']]
    Y = dataset['Price']
```

1.Training Set:

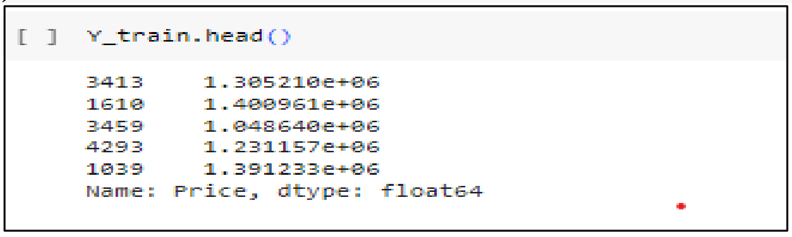
• This is the portion of your data used to train your model. The model learns patterns, relationships, and features from this set.

• The idea is that by exposing your model to a sufficient amount of data, it should be able to learn and understand the underlying patterns in the information.

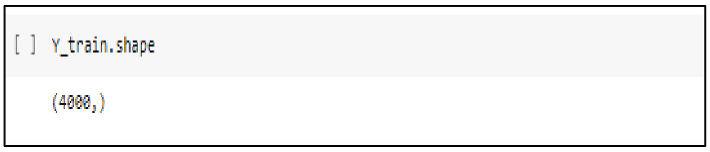


```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=101)
```

)



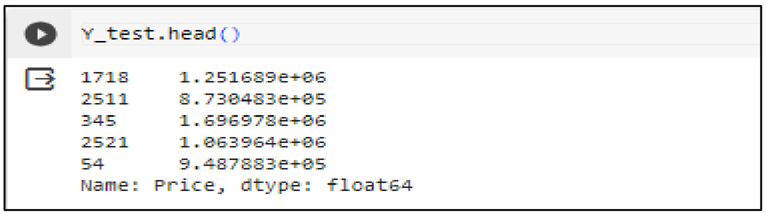
```
[ ] Y_train.head()

    3413    1.305210e+06
    1610    1.400961e+06
    3459    1.048640e+06
    4293    1.231157e+06
    1039    1.391233e+06
    Name: Price, dtype: float64
```



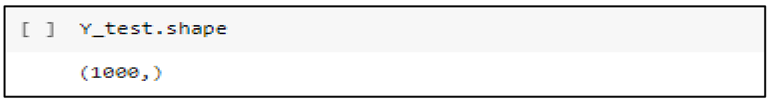
```
[ ] Y_train.shape

    (4000,)
```

2.Testing Set:

- This set is reserved to evaluate how well your model performs on unseen data.
- Once your model is trained, you use the testing set to see how accurately it can make predictions or classifications.
- The testing set serves as a simulation of real-world scenarios where your model encounters new, previously unseen examples.



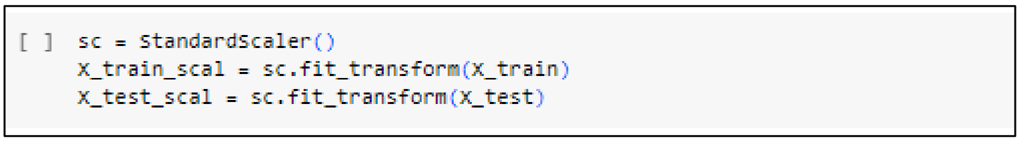
```
Y_test.head()

1718     1.251689e+06
2511     8.730483e+05
345      1.696978e+06
2521     1.063964e+06
54       9.487883e+05
Name: Price, dtype: float64
```



```
[ ] Y_test.shape

    (1000,)
```

Standard Scalar:

Standard Scalar, or standardization, is a technique used in machine learning to scale and center the attributes or features of a dataset. The goal is to ensure that the features have the same scale or distribution.

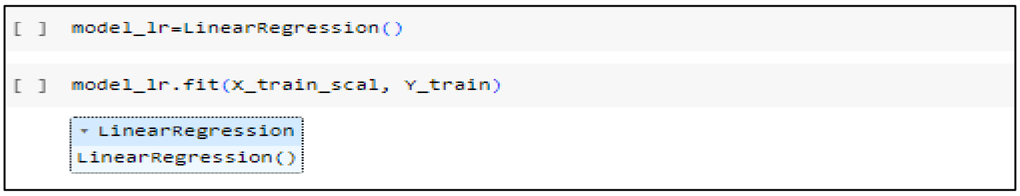


```
[ ] sc = StandardScaler()
    X_train_scal = sc.fit_transform(X_train)
    X_test_scal = sc.fit_transform(X_test)
```

Linear regression:

Linear regression is like fitting a straight line through a cloud of points. It's a simple yet powerful method in statistics and machine learning used for predicting a continuous outcome variable (dependent variable) based on one or more predictor variables (independent variables).

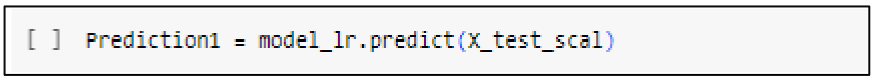
The objective is to find the best-fitting line that minimizes the sum of squared differences between the observed and predicted values.



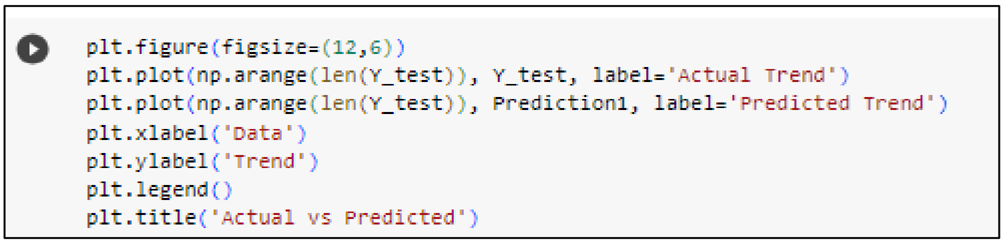
```
[ ] model_lr=LinearRegression()

[ ] model_lr.fit(X_train_scal, Y_train)

    ▾ LinearRegression
    LinearRegression()
```

Predictions:

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

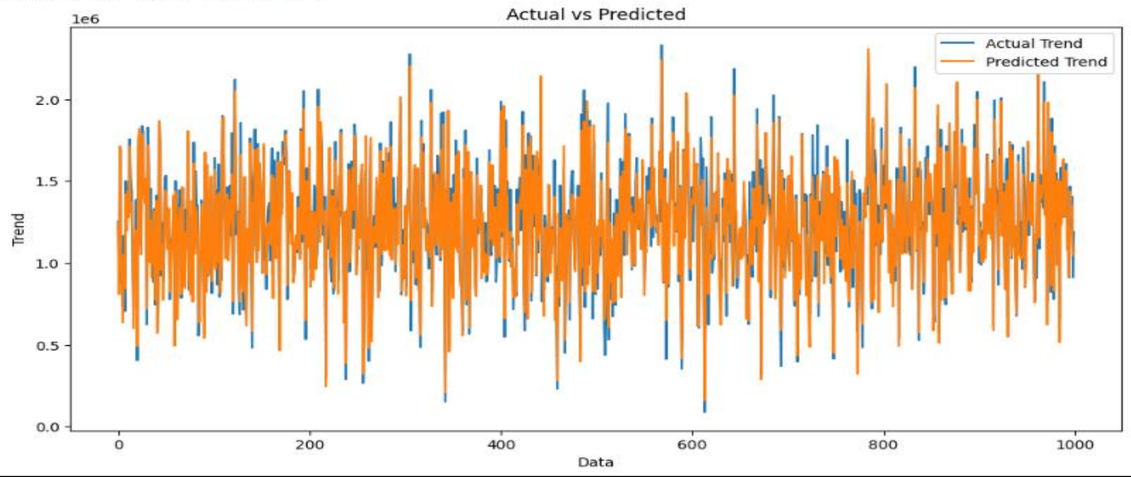


```
[ ] Prediction1 = model_lr.predict(X_test_scal)
```



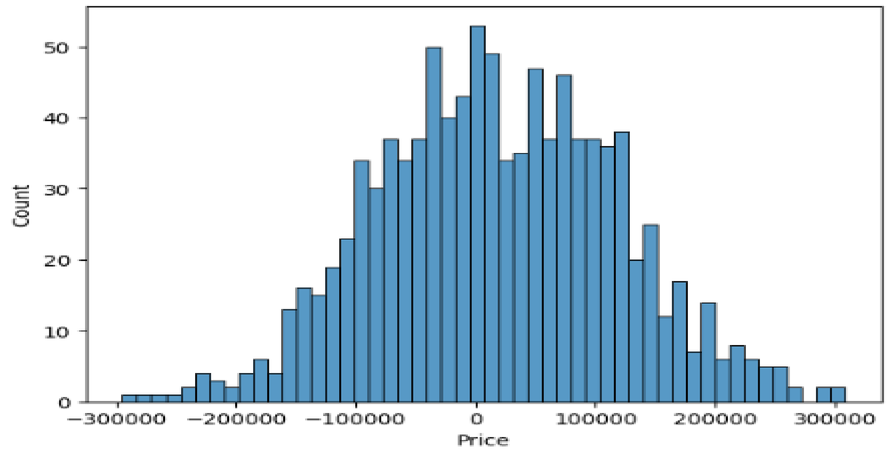
```
plt.figure(figsize=(12,6))
plt.plot(np.arange(len(Y_test)), Y_test, label='Actual Trend')
plt.plot(np.arange(len(Y_test)), Prediction1, label='Predicted Trend')
plt.xlabel('Data')
plt.ylabel('Trend')
plt.legend()
plt.title('Actual vs Predicted')
```

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



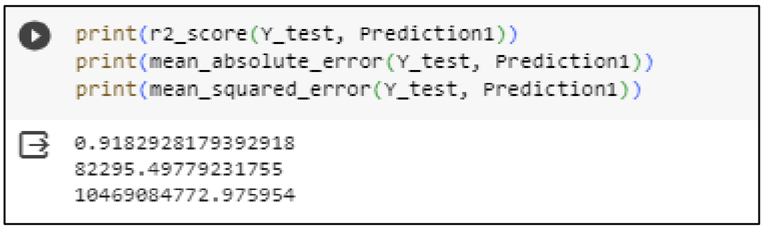
```
sns.histplot((Y_test-Prediction1), bins=50)
```

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



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))
```

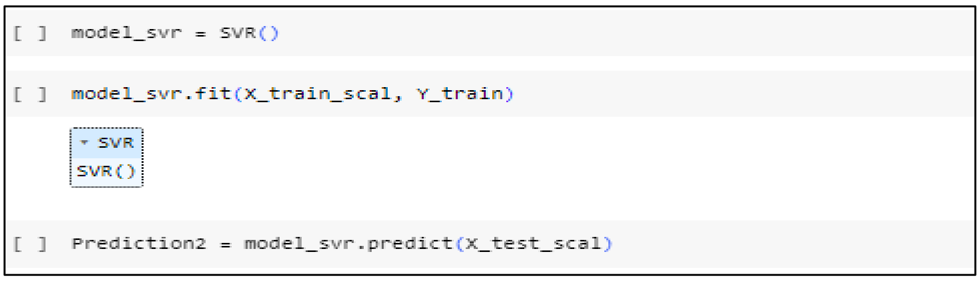
```
0.9182928179392918
82295.49779231755
10469084772.975954
```

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.

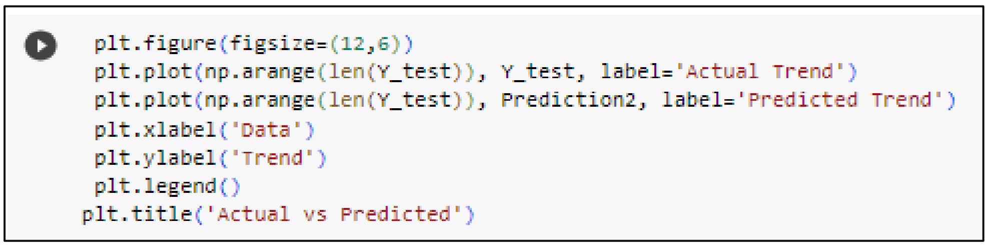
Support vector Regressor:

Support Vector Regression (SVR) is a type of machine learning model that utilizes support vector machines for regression tasks. Similar to Support Vector Machines (SVM) for classification, SVR aims to find a hyperplane that best fits the data while minimizing the error between the predicted and actual values.

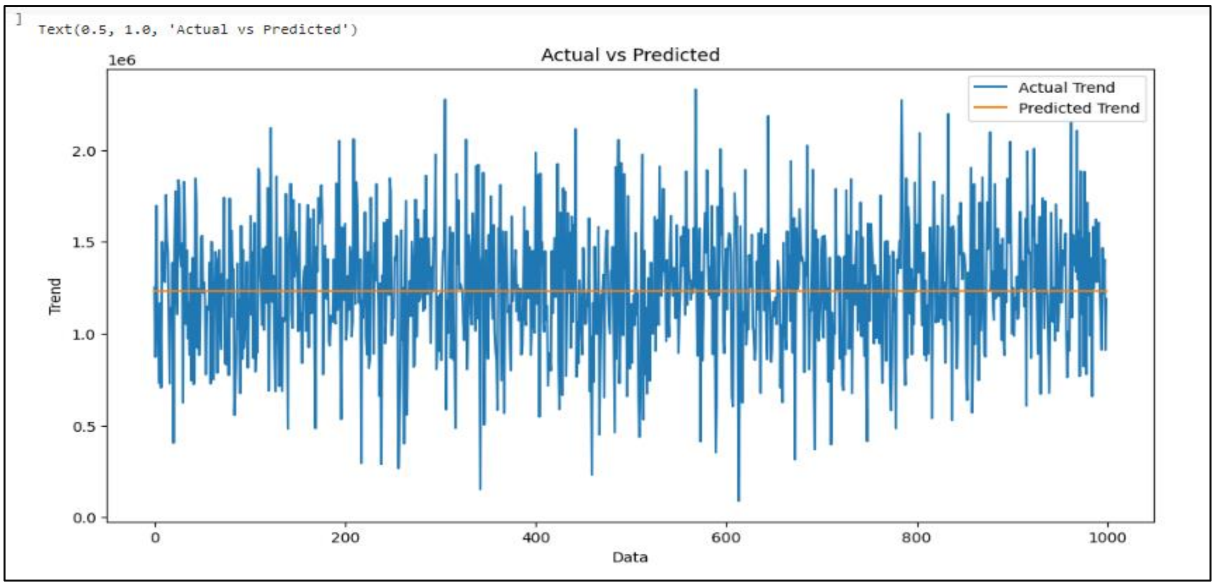
The primary goal of SVR is to find a hyperplane that best represents the trend in the data, while allowing for a margin of error.



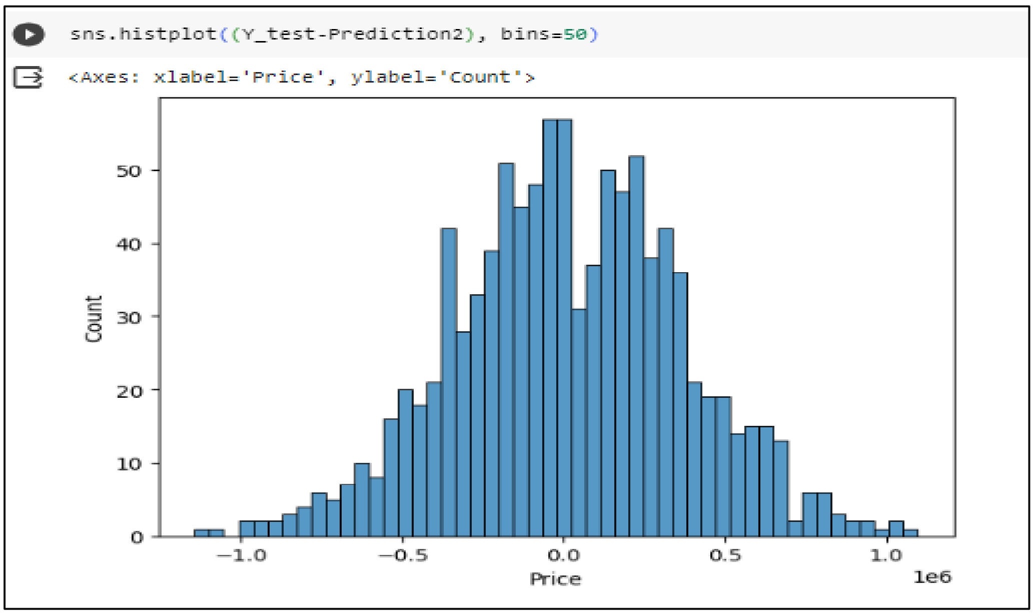
```
[ ]  model_svr = SVR()

[ ]  model_svr.fit(X_train_scal, Y_train)

     ▾ SVR
     SVR()

[ ]  Prediction2 = model_svr.predict(X_test_scal)
```



```
 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')
```

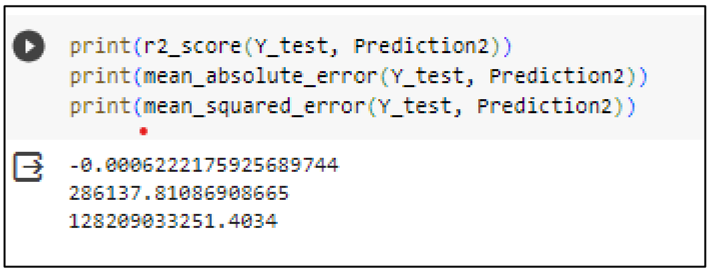


```
Text(0.5, 1.0, 'Actual vs Predicted')
```



```
sns.histplot((Y_test-Prediction2), bins=50)
```

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



```
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
```

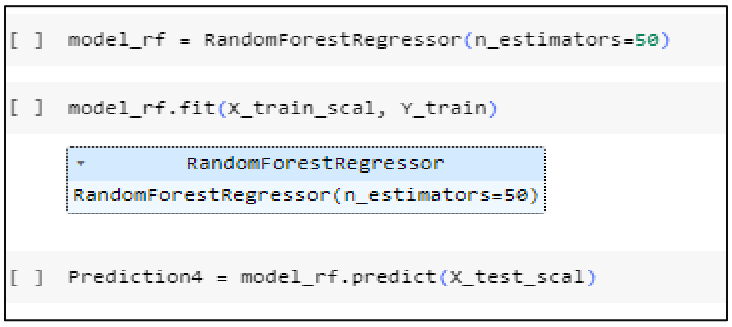
SVR is particularly useful when dealing with datasets where the relationship between the features and the target variable is complex and nonlinear. It's a powerful regression technique, and the choice of the kernel function (linear, polynomial, radial basis function, etc.) can significantly impact its performance on different types of data.

Random Forest Regressor:

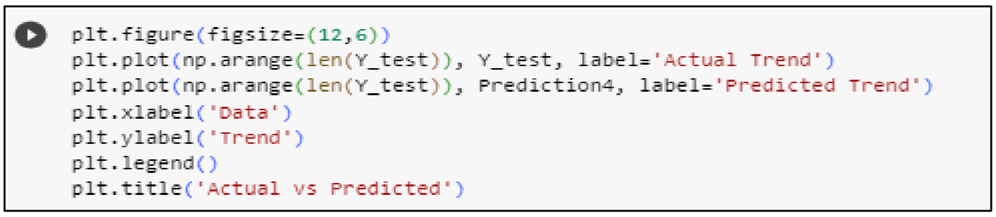
Random Forest Regression is like the wisdom of the crowd applied to predicting numbers. It's an ensemble learning method that combines the predictions of multiple decision trees to improve accuracy and robustness in regression tasks.

Objective:

The primary objective of a Random Forest Regressor is to build an ensemble of decision trees that collectively make accurate predictions for a regression task. Each decision tree is trained on a subset of the data and features, and the final prediction is an average or a weighted average of the predictions of individual trees.

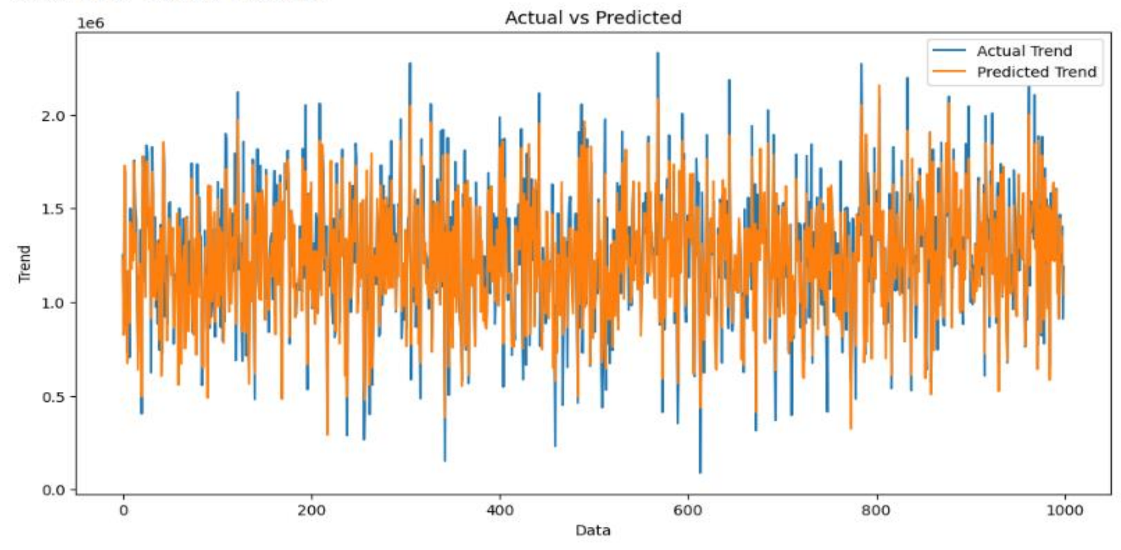


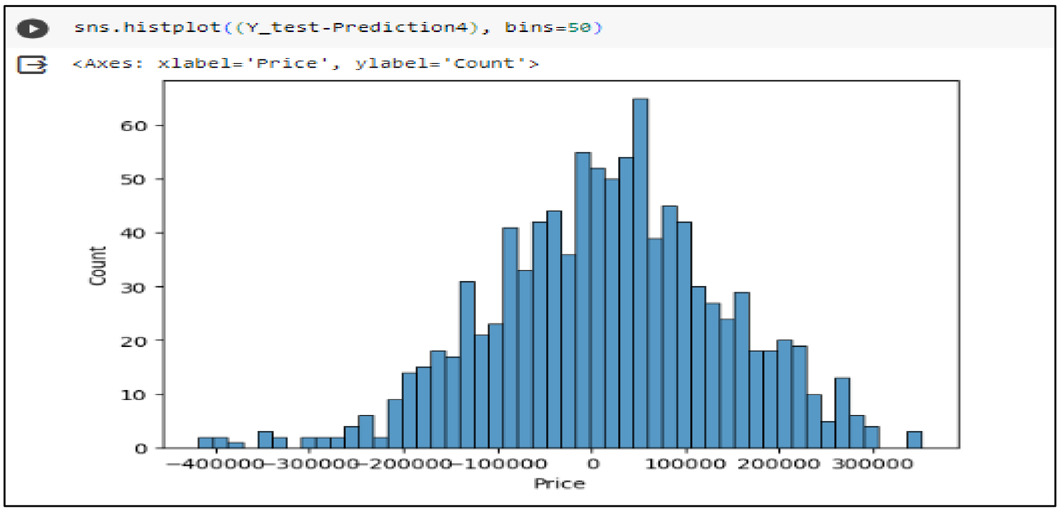
```
[ ]  model_rf = RandomForestRegressor(n_estimators=50)

[ ]  model_rf.fit(X_train_scal, Y_train)

          RandomForestRegressor
     RandomForestRegressor(n_estimators=50)

[ ]  Prediction4 = model_rf.predict(X_test_scal)
```



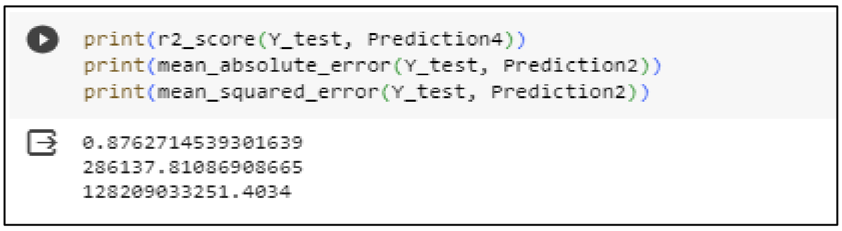
```
plt.figure(figsize=(12,6))
plt.plot(np.arange(len(Y_test)), Y_test, label='Actual Trend')
plt.plot(np.arange(len(Y_test)), Prediction4, label='Predicted Trend')
plt.xlabel('Data')
plt.ylabel('Trend')
plt.legend()
plt.title('Actual vs Predicted')
```

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





```
sns.histplot((Y_test-Prediction4), bins=50)
<Axes: xlabel='Price', ylabel='Count'>
```



```
print(r2_score(Y_test, Prediction4))
print(mean_absolute_error(Y_test, Prediction2))
print(mean_squared_error(Y_test, Prediction2))

0.8762714539301639
286137.81086908665
128209033251.4034
```

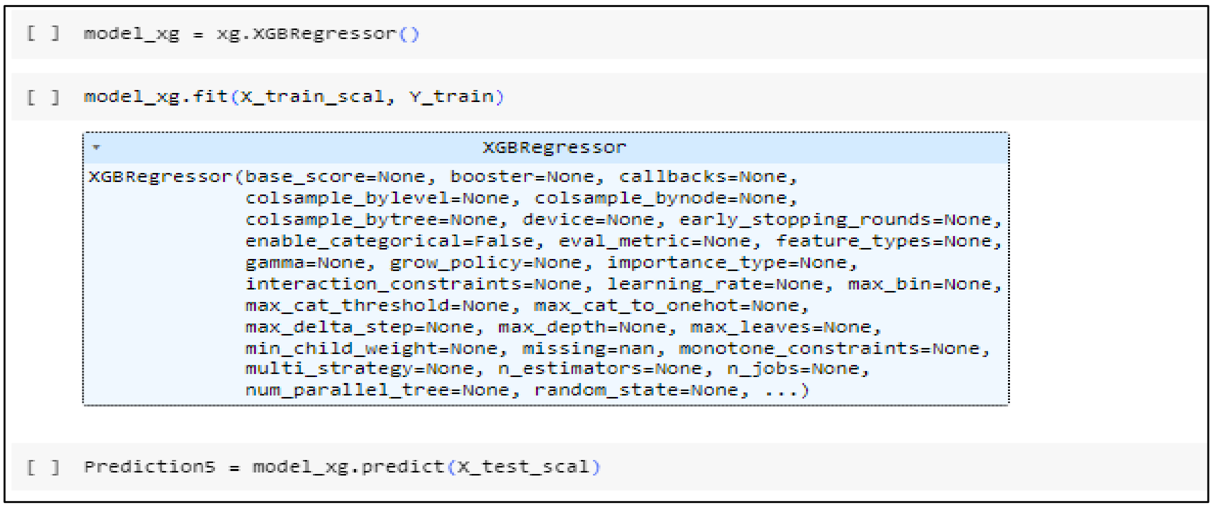
The "random" in Random Forest is key to its success—it adds an element of diversity that prevents overfitting and improves generalization.

XGBoost:

XGBoost is the gradient boosting algorithms—it's powerful, versatile, and can tackle a wide range of machine learning tasks

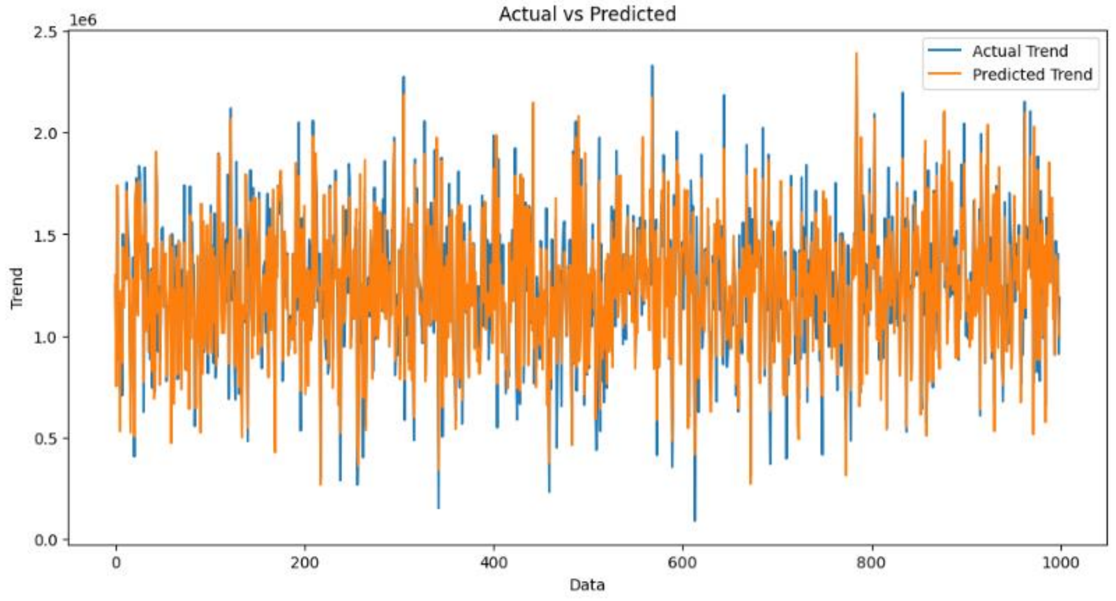
Objective:

The fundamental objective of gradient boosting is to create a strong predictive model by combining the outputs of multiple weak models (typically decision trees) in an additive manner.



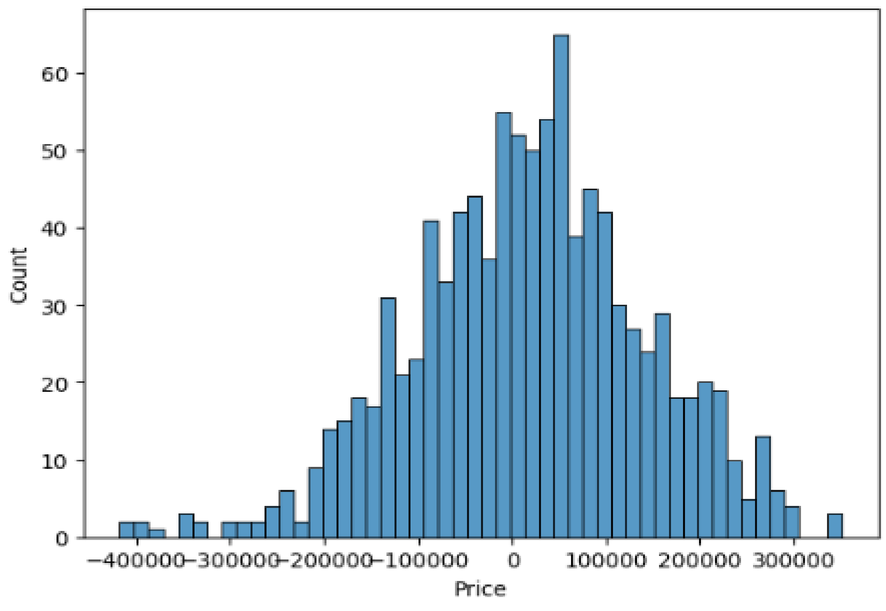
```
[ ] model_xg = xg.XGBRegressor()

[ ] model_xg.fit(X_train_scal, Y_train)

XGBRegressor
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=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,
             multi_strategy=None, n_estimators=None, n_jobs=None,
             num_parallel_tree=None, random_state=None, ...)

[ ] Prediction5 = model_xg.predict(X_test_scal)
[ ] 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')
```

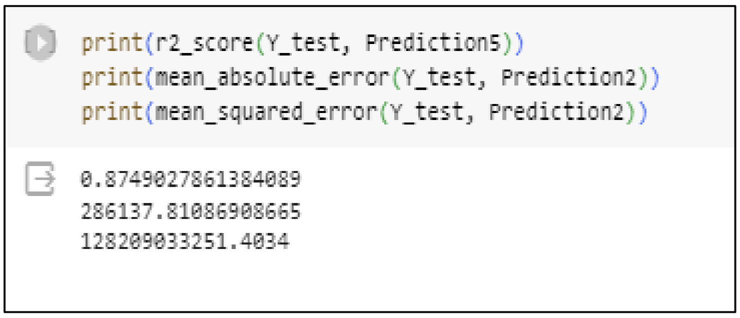
```
Text(0.5, 1.0, 'Actual vs Predicted')
```



```
sns.histplot((Y_test-Prediction4), bins=50)
```

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





```
print(r2_score(Y_test, Prediction5))
print(mean_absolute_error(Y_test, Prediction2))
print(mean_squared_error(Y_test, Prediction2))
```

```
0.8749027861384089
286137.81086908665
128209033251.4034
```

XGBoost optimizes an objective function, which includes a loss function that measures the difference between predicted and actual values, regularization terms to control model complexity, and a component for each tree that corrects errors in the current ensemble. The iterative process of adding trees and optimizing the objective function results in a highly accurate and robust predictive model.

While comparing the machine learning algorithms like linear regression, random forest, support vector machine and XGBoost for the house price prediction analysis, the linear regression algorithm gives the best prediction score.

Linear Regression is giving us best Accuracy.

Thus the linear regression algorithm is a straightforward and interpretable model that works well in many scenarios, particularly when the relationship between the input features and the target variable is approximately linear.

Including an attribute to check accuracy:

Here,we included the Address attribute while training and testing the dataset to check the accuracy.

```
print(r2_score(y_test,y_pred))
```

```
0.8571890180399251
```

While the address attribute is get included we get lesser accuracy when compared when it is not get included.Thus while excluding the address attribute we get more accuracy

CONCLUSION:

These steps provide a simplified overview of concepts like data cleaning, data preprocessing, data analysis, model selection and evaluation for the house price prediction analysis. The specific implementation details and choice of algorithms may vary depending on the dataset and the goals of the project that we work on.