TITLE : PREDICTING HOUSE PRICES USING MACHINE LEARNING

problem statement: Predicting house prices

INTRODUCTION:

- Predicting house prices using Machine Learning is an application of artificial intelligence and data analysis that aims to forecast the market value of residential properties.
- Once trained, these models can make predictions on the value of a house given a set of input features, providing valuable insights to real estate professionals, homebuyers, and sellers.
- Various techniques, including regression models, decision trees, and deep learning, can be applied to this problem, offering different levels of accuracy and interpretability.

Design thinking procss:

Training: Our training data consists of 1,460 examples of houses with 79 features describing every aspect of the house. We are given sale prices (labels) for each house. The training data is what we will use to "teach" our models.

Testing: The test data set consists of 1,459 examples with the same number of features as the training data. Our test data set excludes the sale price because this is what we are trying to predict. Once our models have been built we will run the best one the test data and submit it to the Kaggle leaderboard.

Task: Machine learning tasks are usually split into three categories; supervised, unsupervised and reinforcement. For this competition, our task is supervised learning.

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

Data Preprocessing steps:

- **Handle missing values:** Decide on a strategy to deal with missing data, such as imputation or removal.
- **Data scaling:** Normalize or standardize numerical features to ensure all features have a similar scale.
- **Categorical data**: Encode categorical variables using techniques like one-hot encoding or label encoding.

Model Training:

- Fit the chosen algorithm on the training data using the selected features.
- The algorithm learns the relationships between the input features and the target variable (house prices) during this step.

Model Evaluation:

Assess your model's performance on the validation set using appropriate evaluation metrics such as Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), or R-squared (R2).

Mean Absolute Error (MAE):

MAE measures the average absolute difference between the predicted and actual prices. It provides a simple, easily interpretable measure of the model's error.

Mean Squared Error (MSE):

MSE measures the average squared difference between predicted and actual prices. It amplifies the impact of large errors, making it suitable for identifying major outliers.

Root Mean Squared Error (RMSE):

RMSE is the square root of MSE. It provides the error in the same units as the target variable, making it more interpretable.

R-squared (R^2):

R^2 measures the proportion of the variance in the target variable that is predictable from the independent variables. A higher R^2 indicates a better fit, but it doesn't directly measure the magnitude of prediction errors.

Dividing Dataset in to features and target variable

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

Using Train Test Split

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

Output:

```
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_test.shape
```

Output:

```
(1000,)
```

Standardizing the data

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

Model Building and Evaluation

Model 1 – Linear Regression

```
Model_lr=LinearRegression()
Model_lr.fit(X_train_scal, Y_train)
```

Output:

```
LinearRegression
LinearRegression()
```

Predicting Prices

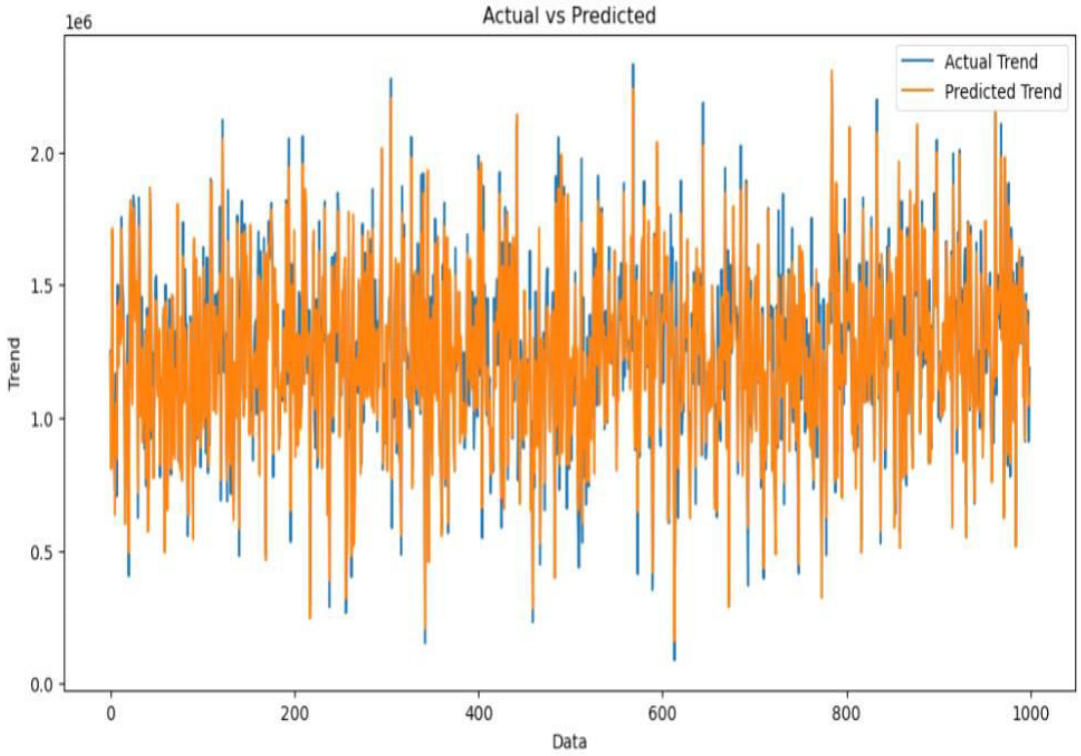
```
Prediction1 = model_lr.predict(X_test_scal)
```

Evaluation of Predicted Data

```
    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')
```

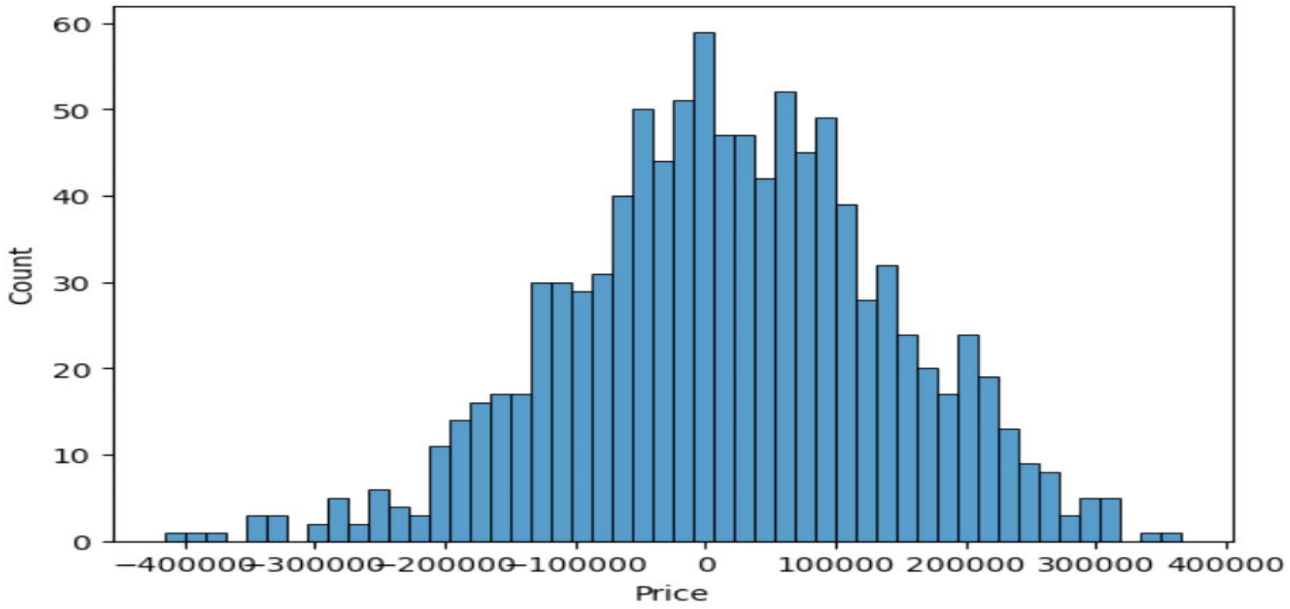
Output:

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



Output:

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



Print(r2_score(Y_test, Prediction1))

Print(mean_absolute_error(Y_test, Prediction1))

Print(mean_squared_error(Y_test, Prediction1))

Output:

0.9182928179392918

82295.49779231755

10469084772.975954

Predicting Prices

Prediction5 = model_xg.predict(X_test_scal)

Evaluation of Predicted Data

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’)

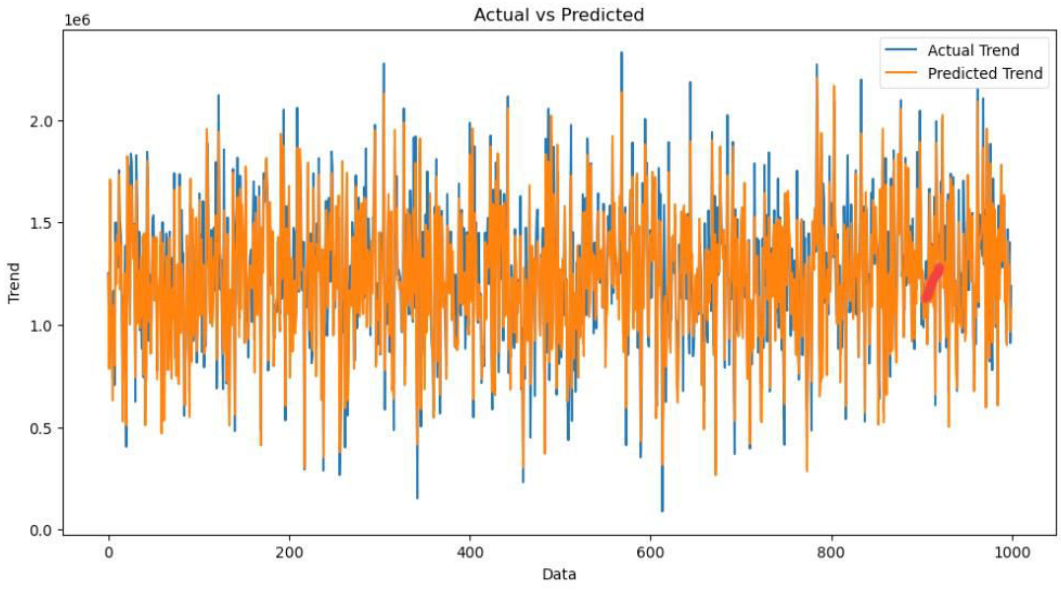
Plt.ylabel(‘Trend’)

Plt.legend()

Plt.title(‘Actual vs Predicted’)

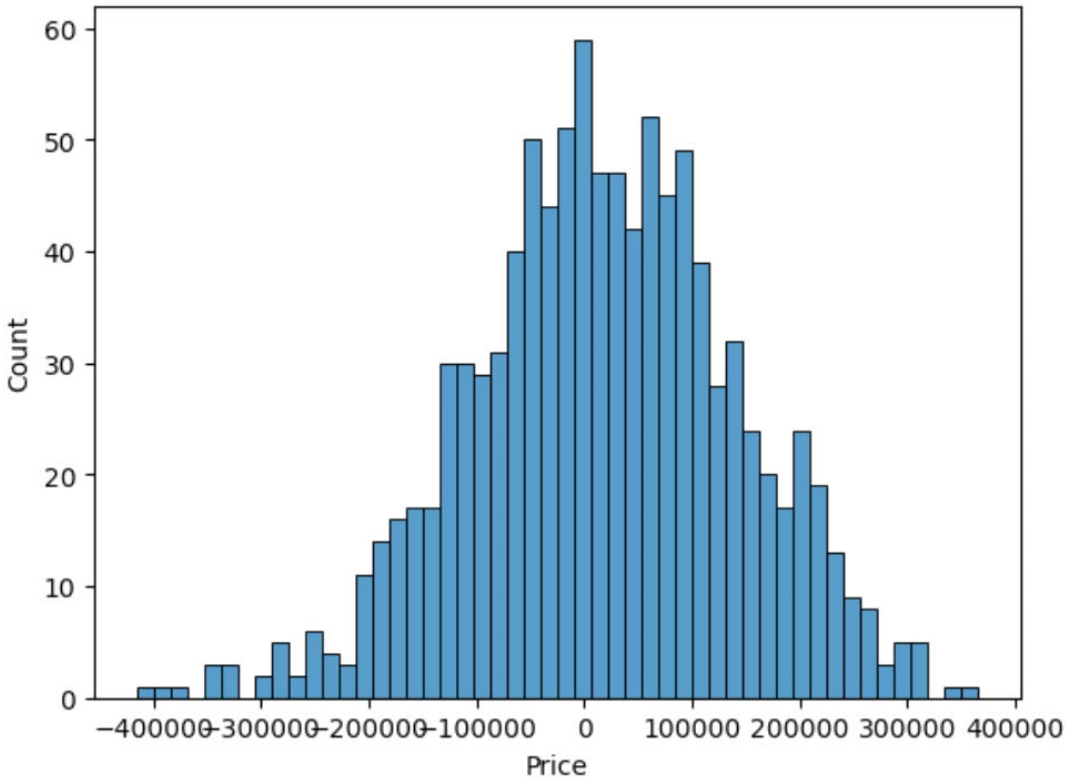
Output:

Text(0.5, 1.0, ‘Actual vs Predicted’)



Sns.histplot((Y_test-Prediction4), bins=50)

Output:



<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))

Output:

-0.0006222175925689744

286137.81086908665

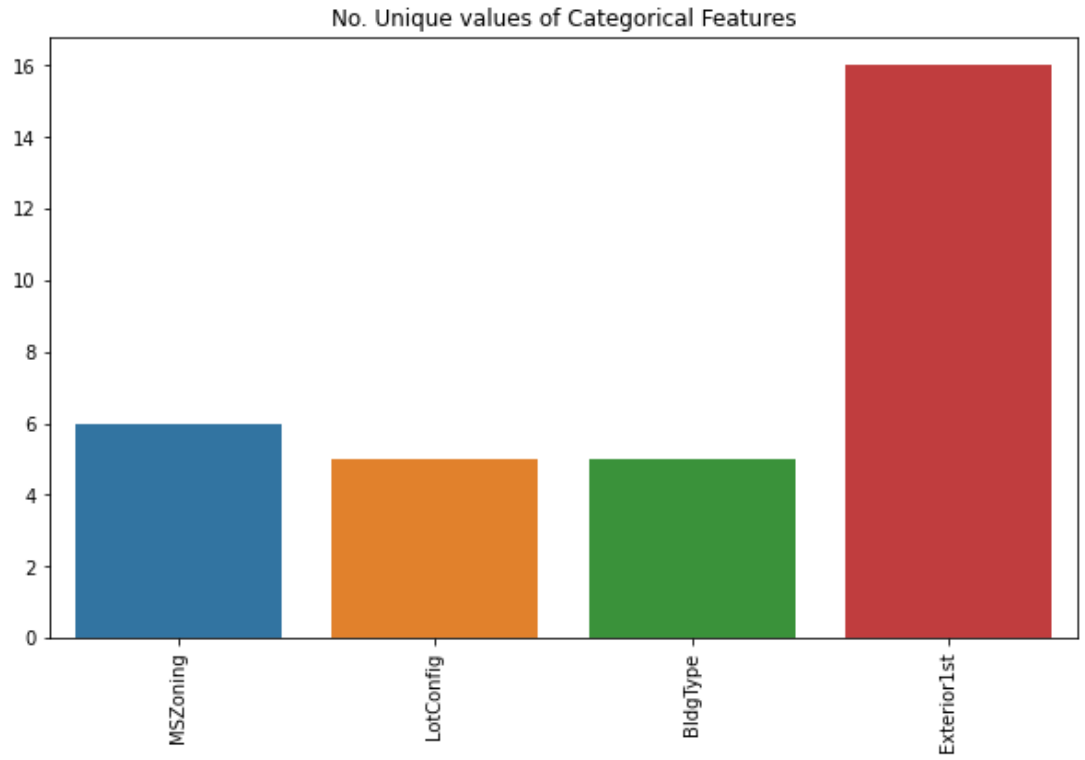
128209033251.4034

Barplot:

unique_values = []for col in object_cols

```
unique_values.append(dataset[col].unique().size)
plt.figure(figsize=(10,6))
plt.title('No. Unique values of Categorical Features')
plt.xticks(rotation=90)
sns.barplot(x=object_cols,y=unique_values)
```

OUTPUT:



Importing Libraries :

Matplotlib

Seaborn

Pandas

Data Preprocessing:

```bash
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))
```

OUTPUT:

Categorical variables : 4

Integer variables : 6

Float variables : 3

Exploratory Data Analysis:

```bash
plt.figure(figsize=(12, 6))
sns.heatmap(dataset.corr(),
cmap = 'BrBG',
```

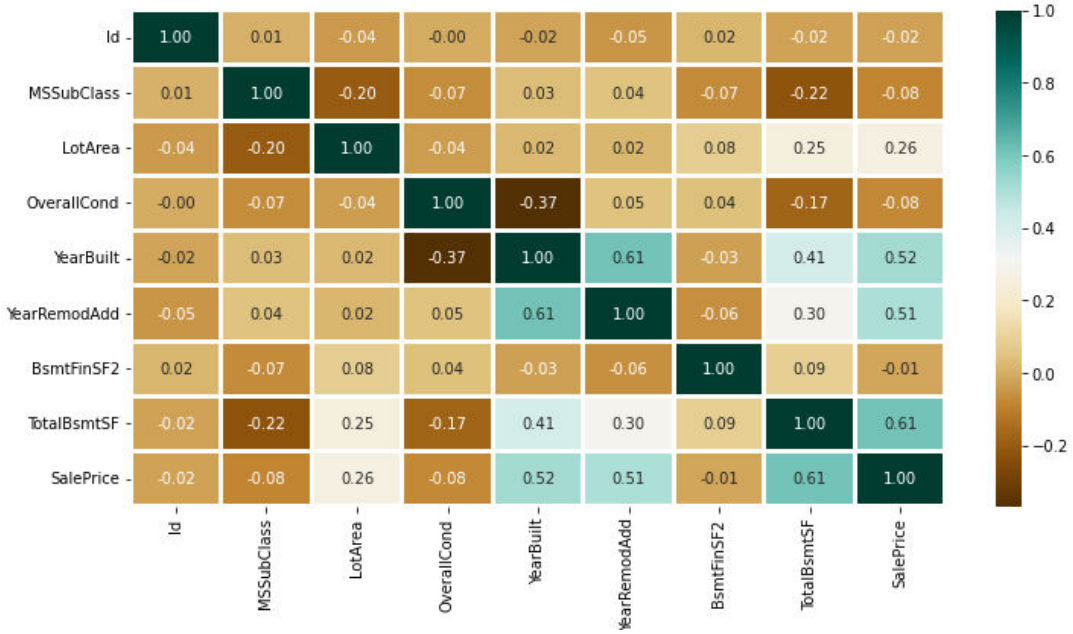
fmt = '.2f',

linewidths = 2,

annot = True)

```

**OUTPUT:**




**Data Cleaning:**

```bash

dataset.drop(['Id'],axis=1,inplace=True)

```

OUTPUT:
```

```
MSSubClass      0
MSZoning        0
LotArea         0
LotConfig       0
BldgType        0
OverallCond     0
YearBuilt       0
YearRemodAdd    0
Exterior1st     0
BsmtFinSF2      0
TotalBsmtSF     0
SalePrice       0
dtype: int64
```

OneHotEncoder – For Label categorical features:

```bash
from sklearn.preprocessing import OneHotEncoder
s = (new_dataset.dtypes == 'object')
object_cols = list(s[s].index)
print("Categorical variables:")
print(object_cols)
print('No. of. categorical features: ',
len(object_cols))
```

OUTPUT:

```
Categorical variables:
['MSZoning', 'LotConfig', 'BldgType', 'Exterior1st']
No. of. categorical features:  4
```

Splitting Dataset into Training and Testing:

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

Standardizing the data:

```bash
Sc = StandardScaler()
X_train_scal = sc.fit_transform(X_train)`J
X_test_scal = sc.fit_transform(X_test)
```

OUTPUT:

LinearRegression

LinearRegression()

Predicting Prices:

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

Conclusion:

Clearly, SVM model is giving better accuracy as the mean absolute error is the least among all the other regressor models i.e. 0.18 approx. To get much better results ensemble learning techniques like Bagging and Boosting can also be used.