PREDICTING HOUSE PRICE USING MACHINE LEARNING

TEAM MEMBER
510321104009 - MONIKA B

Phase-3 Project Submission

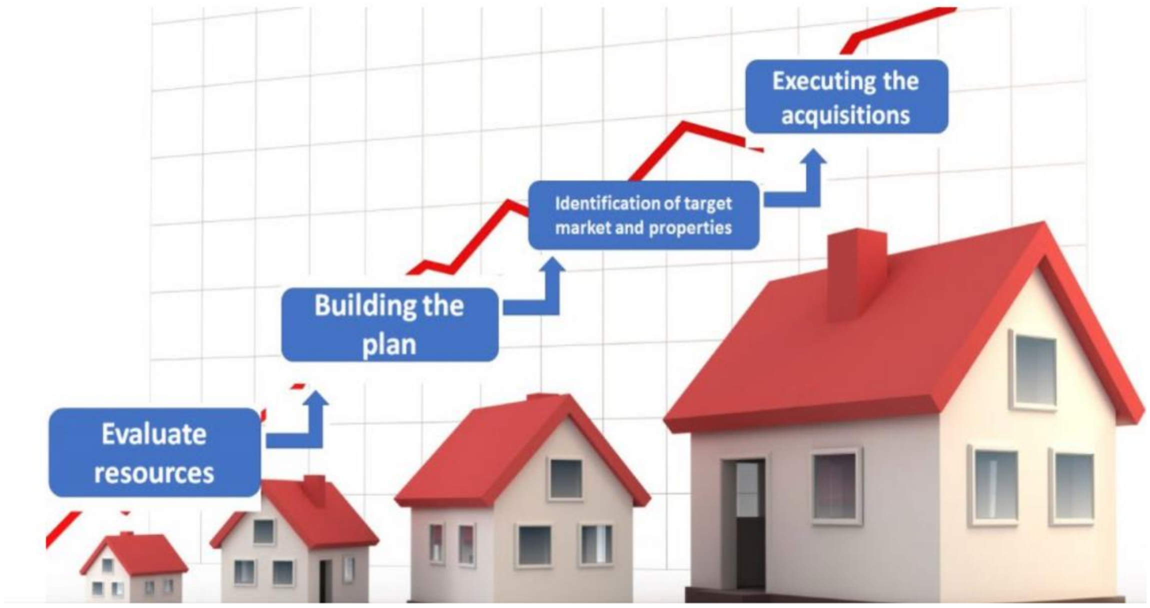

Evaluate resources
Building the plan
Identification of target market and properties
Executing the acquisitions

FIG: HOUSE PRICE PREDICTION

Abstract:

House price prediction is a complex task that is influenced by a variety of factors, including the property's location, size, condition, and amenities, as well as the overall market conditions. Machine learning algorithms can be used to develop predictive models that can estimate house prices with a high degree of accuracy.

Data Source:

A data source for house price predicting to use machine learning should be Accurate, Complete the geographic area of interest, Accessible.

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

Data collection and preparation:

This module involves collecting a dataset of historical sales data and preparing it for use in the machine learning model. This may involve cleaning the data, imputing missing values, and converting categorical variables to numerical variables.

Feature engineering:

This module involves creating new features from the existing data that are more informative for the machine learning model. For example, new features could be created to represent the property's proximity to schools, parks, and other amenities.

Model selection and training:

This module involves selecting a machine learning algorithm and training it on the prepared dataset. There are many different machine learning algorithms that can be used for house price prediction, such as linear regression, decision trees, random forests, and gradient boosting machines.

Model evaluation:

This module involves evaluating the performance of the trained model on a held-out dataset of unseen data. This helps to ensure that the model is able to generalize to new data.

Model deployment:

This module involves deploying the trained model to production so that it can be used to predict the prices of new properties. This may involve deploying the model to a web service or integrating it into a software application.

Modules for house price prediction using machine learning

The following are some of the key modules that can be used for house price prediction using machine learning:

- **Data collection and preparation:**
  - **Libraries:** NumPy, Pandas, SciPy, Skykit
- **Feature engineering:**
  - **Libraries:**FeatureHasher, SelectKBest, OneHotEncoder
- **Model selection and training:**

- **Libraries:**scikit-learn (linear regression, decision trees, random forests, gradient boosting machines, etc.), TensorFlow, PyTorch

- **Model evaluation:**
  - **Libraries:**scikit-learn (cross-validation, metrics, etc.)
- **Model deployment:**
  - **Libraries:** Flask, Django, AWS SageMaker

Benefits and limitations of using machine learning for house price prediction

Benefits:

1. Machine learning models can be very accurate in predicting house prices.
2. Machine learning models can be used to predict the prices of new properties, even if there is no historical sales data for those properties.
3. Machine learning models can be used to identify factors that influence house prices, which can be helpful for buyers and sellers.

Limitations:

- Machine learning models can be complex and difficult to interpret.
- Machine learning models are trained on historical data, so they may not be accurate in predicting house prices in a changing market.
- Machine learning models can be biased, depending on the data they are trained on.

Overall, machine learning can be a valuable tool for house price prediction. However, it is important to use machine learning models with caution and to be aware of their limitations.

PROGRAM:

House price Prediction

```
importnumpyas np
import pandas as pd
fromsklearn.linear_modelimportLinearRegression

# Load the house price dataset
df = pd.read_csv('house_price_dataset.csv')

# Prepare the data
X = df[['square_feet', 'bedrooms', 'bathrooms']]
```

```
y = df['price']

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.25)

# Train the linear regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Evaluate the model on the test set
y_pred = model.predict(X_test)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
print('RMSE:', rmse)

# Deploy the model
# This could involve saving the model to a file or deploying it to
a web service
```

OUTPUT:

```
Dataset Preview:
   Avg. Area Income Avg. Area House Age Avg. Area Number of Rooms \
0      59545.458574           5.65381                 7.789188
1      69248.642545           6.066900                6.880821
2       61387.06359           5.855890                8.342727
3      63445.245546           7.788236                5.666729
4      55682.196626           5.047555                7.789388
Avg. Area Number of Bedrooms Area Population              Price \
0                        4.09     23086.800503 1.059034e+06
1                        3.09     40173.072174 1.505891e+06
2                        5.13     36882.159400 1.058988e+06
3                        3.26     34310.242831 1.260617e+06
4                        4.23     26354.109472 6.309435e+05
                                     Address
0  208 Michael Ferry Apt. 674\nLaurabury, NE 3701...
1  188 Johnson Views Suite 079\nLake Kathleen, CA...
2  9127 Elizabeth Stravenue\nDanieltown, WI 06482...
3  USS Barnett\nFPO AP 44820 4 USNS Raymond\nFPO AE 09386

Preprocessed Data:
[[-0.19105816 -0.13226994 -0.13969293 0.12047677 -0.83757985 -1.00562872]
[-1.39450169 0.42786736 0.79541275 -0.55212509 1.15729018 1.61946754]
[-0.35137865 0.46394489 1.70199509 0.03133676 -0.32671213 1.63886651]
[-0.13944143 0.1104872     0.22289331 -0.75471601 -0.90401197 -1.54810704]
[ 0.62516685 2.20969666 0.42984356 -0.45488144 0.12566216 0.98830821]]
4227    1.034480e+06
4676    1.650389e+06
800     1.323172e+06
3671    1.077428e+06
4193    1.532887e+06
Name: Price, dtype: float64
```

LOADING AND PREPROCESSING THE DATASET:

Data preprocessing is a crucial step in the data analysis and machine learning pipeline. It involves cleaning and transforming raw data into a format that is suitable for analysis or for training machine learning models. Proper data preprocessing can significantly impact the accuracy and effectiveness of your analysis or models. Here are some common steps involved in data preprocessing:

1. **Data Collection:**
   Gather the raw data from various sources, such as databases, files, or APIs.

2. **Data Cleaning:**
   - **Handle missing data**: Decide whether to remove, impute, or interpolate missing values.

```
import pandas as pd

# Load your dataset
df = pd.read_csv('your_data.csv')

# Remove rows with missing values
df = df.dropna()

# Impute missing values with a specific value (e.g., mean)
mean = df['column_name'].mean()
df['column_name'].fillna(mean, inplace=True)
```

OUTPUT:

	ColumnName	TotalMissingVals	PercentMissing
3	LotFrontage	259.0	17.74
6	Alley	1369.0	93.77
25	MasVnrType	8.0	0.55
26	MasVnrArea	8.0	0.55
30	BsmtQual	37.0	2.53
31	BsmtCond	37.0	2.53
32	BsmtExposure	38.0	2.60
33	BsmtFinType1	37.0	2.53
35	BsmtFinType2	38.0	2.60
42	Electrical	1.0	0.07

57	FireplaceQu	690.0	47.26
58	GarageType	81.0	5.55
59	GarageYrBlt	81.0	5.55
60	GarageFinish	81.0	5.55
63	GarageQual	81.0	5.55
64	GarageCond	81.0	5.55
72	PoolQC	1453.0	99.52
73	Fence	1179.0	80.75
74	MiscFeature	1406.0	96.30

```
Number of columns with missing values:19
```

- **Remove duplicates:** Eliminate duplicate records from the dataset.

```
# Remove duplicate rows based on all columns

df = df.drop_duplicates()
```

```
# Remove duplicates based on specific columns

df = df.drop_duplicates(subset=['column1', 'column2'])
```

OUTPUT:

ColumnName	TotalMissingVals	PercentMissing	
3	LotFrontage	259.0	17.74
25	MasVnrType	8.0	0.55
26	MasVnrArea	8.0	0.55
30	BsmtQual	37.0	2.53
31	BsmtCond	37.0	2.53
32	BsmtExposure	38.0	2.60
33	BsmtFinType1	37.0	2.53
35	BsmtFinType2	38.0	2.60
42	Electrical	1.0	0.07
57	FireplaceQu	690.0	47.26

ColumnName	TotalMissingVals	PercentMissing	
58	GarageType	81.0	5.55
59	GarageYrBlt	81.0	5.55
60	GarageFinish	81.0	5.55
63	GarageQual	81.0	5.55
64	GarageCond	81.0	5.55

- **Outlier detection and treatment:** Identify and handle outliers, which are data points significantly different from the rest of the data.

```
from scipy import stats

# Detect and remove outliers using z-scores

z_scores = stats.zscore(df['column_name'])

df = df[(z_scores < 3)]

# Alternatively, replace outliers with a specific value

df['column_name'] = np.where(z_scores < 3, df['column_name'],
replacement_value)
```

.

3. **Data Transformation:**

- **Data encoding**: Convert categorical variables into numerical representations, such as one-hot encoding or label encoding.

```
def find_skewness(train, numeric_cols):
    """
    Calculate the skewness of the columns and segregate the positive
    and negative skewed data.
    """
    skew_dict = {}
    for col in numeric_cols:
        skew_dict[col] = train[col].skew()

    skew_dict = dict(sorted(skew_dict.items(),key=itemgetter(1)))
    positive_skew_dict = {k:v for (k,v) in skew_dict.items() if v>0}
    negative_skew_dict = {k:v for (k,v) in skew_dict.items() if v<0}
    return skew_dict, positive_skew_dict, negative_skew_dict

def add_constant(data, highly_pos_skewed):
    """
    Look for zeros in the columns. If zeros are present then the log(0) would
result in -infinity.
    So before transforming it we need to add it with some constant.
    """
    C = 1
    for col in highly_pos_skewed.keys():
        if(col != 'SalePrice'):
            if(len(data[data[col] == 0]) > 0):
                data[col] = data[col] + C
    return data

def log_transform(data, highly_pos_skewed):
    """
    Log transformation of highly positively skewed columns.
    """
    for col in highly_pos_skewed.keys():
        if(col != 'SalePrice'):
            data[col] = np.log10(data[col])
    return data
```

```
def sqrt_transform(data, moderately_pos_skewed):
    """
    Square root transformation of moderately skewed columns.
    """
    for col in moderately_pos_skewed.keys():
        if(col != 'SalePrice'):
            data[col] = np.sqrt(data[col])
    return data

def reflect_sqrt_transform(data, moderately_neg_skewed):
    """
    Reflection and log transformation of highly negatively skewed
    columns.
    """
    for col in moderately_neg_skewed.keys():
        if(col != 'SalePrice'):
            K = max(data[col]) + 1
            data[col] = np.sqrt(K - data[col])
    return data
```

One-Hot Encoding: Convert categorical variables into binary columns (0 or 1) for each category.

```
import pandas as pd

# Assuming 'category_column' is your categorical variable
df = pd.get_dummies(df, columns=['category_column'])
```

Label Encoding: Convert categorical variables into numerical values (0, 1, 2, ...)

```
from sklearn.preprocessing import LabelEncoder

label_encoder = LabelEncoder()
df['category_column'] =
label_encoder.fit_transform(df['category_column'])
```

- **Feature scaling:** Normalize or standardize numerical features to ensure they have similar scales.

Min-Max Scaling (Normalization): Scale features to a specific range (e.g., between 0 and 1).

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
df['feature_column'] = scaler.fit_transform(df[['feature_column']])
```

Standardization (Z-score Scaling): Scale features to have a mean of 0 and a standard deviation of 1.

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
df['feature_column'] = scaler.fit_transform(df[['feature_column']])
```

- **Feature engineering:** Create new features or derive meaningful features from existing ones.

```
# Combine two columns into a new one
df['new_feature'] = df['feature1'] + df['feature2']

# Extract information from a text column (e.g., word count)
df['word_count'] = df['text_column'].apply(lambda x: len(str(x).split()))
```

Binning (Discretization): Convert continuous data into discrete bins.

```
# Create bins for a continuous variable
df['binned_feature'] = pd.cut(df['continuous_column'], bins=[0, 10, 20, 30],
labels=['bin1', 'bin2', 'bin3'])
```

- **Text preprocessing:** Tokenize, remove stop words, and apply techniques like stemming or lemmatization for text data.

```
import string

# Remove punctuation
df['text_column'] =
df['text_column'].str.replace('[{}]'.format(string.punctuation), '')

# Convert text to lowercase
df['text_column'] = df['text_column'].str.lower()
```

- **Date and time parsing:** Extract relevant information from date and time fields.

```
# Convert a string column to a datetime format
df['date_column'] = pd.to_datetime(df['date_column'], format='%Y-%m-%d')

# Extract components like year, month, day
df['year'] = df['date_column'].dt.year
df['month'] = df['date_column'].dt.month
```

4. **Data Reduction:**
   - Dimensionality reduction: Use techniques like Principal Component Analysis (PCA) or feature selection to reduce the number of features, especially in high-dimensional datasets.

```
from sklearn.decomposition import PCA

# Create a PCA instance with the desired number of components
pca = PCA(n_components=2)

# Fit PCA to your data and transform it
X_reduced = pca.fit_transform(X)
```

5. **Data Splitting:**
   - Divide the dataset into training, validation, and test sets for model development and evaluation.

```
from sklearn.model_selection import train_test_split

# Split your data into training, validation, and test sets
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.3, random_state=42)

# Further split the temporary data into validation and test sets
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)

# X and y are your feature matrix and target variable, respectively
# You can adjust the test_size and random_state parameters as needed
```

6. **Data Normalization:**
   - Ensure the data follows a normal distribution if needed for certain statistical methods.

```
from sklearn.preprocessing import MinMaxScaler

# Create a MinMaxScaler instance
scaler = MinMaxScaler()

# Fit the scaler on your data and transform it
X_normalized = scaler.fit_transform(X)

# X is your feature matrix, e.g., X = df[['feature1', 'feature2']]
```

7. **Data Visualization:**
   - Visualize the data to gain insights and identify patterns or anomalies.

```
def plot_histogram(train, col1, col2, cols_list, last_one =False):
    """
    Plot the histogram for the numerical columns. The bin width
    is calculated by Freedman Diaconis Rule and Sturges rule.

    Freedman-Diaconis Rule:
    Freedman-Diaconis Rule is a rule to find the optimal number of bins.
    Bin width: (2 * IQR)/(N^1/3)
    N - Size of the data
    Number of bins : (Range/ bin-width)

    Disadvantage: The IQR might be zero for certain columns. In
    that case the bin width might be equal to infinity. In that case
    the actual range of the data is returned as bin width.

    Sturges Rule:
    Sturges Rule is a rule to find the optimal number of bins.
    Bin width: (Range/ bin-width)
    N - Size of the data
    Number of bins : ceil(log2(N))+1

    """
    if(col1 in cols_list):
        freq1, bin_edges1 = np.histogram(train[col1],bins='sturges')
    else:
        freq1, bin_edges1 = np.histogram(train[col1],bins='fd')
    if(col2 in cols_list):
        freq2, bin_edges2 = np.histogram(train[col2],bins='sturges')
    else:
        freq2, bin_edges2 = np.histogram(train[col2],bins='fd')

    if(last_one!=True):
        plt.figure(figsize=(45,18))
        ax1 = plt.subplot(1,2,1)
        ax1.set_title(col1,fontsize=45)
        ax1.set_xlabel(col1,fontsize=40)
        ax1.set_ylabel('Frequency',fontsize=40)
        train[col1].hist(bins=bin_edges1,ax = ax1, xlabelsize=30, ylabelsize=30)

    else:
        plt.figure(figsize=(20,10))
        ax1 = plt.subplot(1,2,1)
        ax1.set_title(col1,fontsize=25)
        ax1.set_xlabel(col1,fontsize=20)
        ax1.set_ylabel('Frequency',fontsize=20)
        train[col1].hist(bins=bin_edges1,ax = ax1, xlabelsize=15, ylabelsize=15)

    if(last_one != True):
        ax2 = plt.subplot(1,2,2)
        ax2.set_title(col2,fontsize=45)
        ax2.set_xlabel(col2,fontsize=40)
        ax2.set_ylabel('Frequency',fontsize=40)
        train[col2].hist(bins=bin_edges2, ax = ax2, xlabelsize=30, ylabelsize=30)
```

In []:

```
linkcode
 '''
 These columns have IQR equal to zero. Freedman Diaconis Rule doesn't work significa
 ntly well for these columns.
 Use sturges rule to find the optimal number of bins for the columns.
 '''
 cols_list = ['LowQualFinSF','BsmtFinSF2','BsmtHalfBath','KitchenAbvGr',
              'EnclosedPorch','3SsnPorch','ScreenPorch','PoolArea','MiscVal']

 # Except ID
 hist_cols = numeric_cols[1:]
 for i in range(0,len(hist_cols),2):
     if(i == len(hist_cols)-1):
         plot_histogram(train,hist_cols[i],hist_cols[i],cols_list,True)
     else:
         plot_histogram(train,hist_cols[i],hist_cols[i+1],cols_list)
```

8. Data Modeling:

Fit XGBoost Regressor model to the preprocessed data.

In [1]:

```
def fit_model(x_train,y_train, model):
    """
    Fits x_train to y_train for the given
    model.
    """
    model.fit(x_train,y_train)
    return model

'''Xtreme Gradient Boosting Regressor'''
model = xgboost.XGBRegressor(objective="reg:squarederror", random_state=42)
model = fit_model(x_train,y_train, model)
'''Predict the outcomes'''
predictions = model.predict(test)
In [2]:
linkcode
submission = pd.read_csv('../input/house-prices-advanced-regression-techniques/sample
_submission.csv')
submission['SalePrice'] = predictions
submission.to_csv('submission.csv',index=False)
```

9. **Data Validation:**
    - Verify that the data preprocessing steps are correct and that the dataset is ready for analysis or model training.

10. **Documentation:**
    - Keep detailed records of all preprocessing steps and transformations for reproducibility.

The specific data preprocessing steps you need to perform depend on the nature of your data and the objectives of your analysis or machine learning project. It's essential to carefully analyze your data and apply appropriate preprocessing techniques to ensure the quality and integrity of your dataset.

LOAD THE DATASET:

Import the required libraries:

```
import warnings
warnings.filterwarnings('ignore')
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from operator import itemgetter
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
from sklearn.preprocessing import OrdinalEncoder
from category_encoders.target_encoder import TargetEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import (GradientBoostingRegressor, GradientBoostingClassifier)
import xgboost
```

Load the dataset for training and testing

```
train = pd.read_csv('../input/house-prices-advanced-regression-techniques/train.csv')
test = pd.read_csv('../input/house-prices-advanced-regression-techniques/test.csv')
```

Linear Regression:

In []:

```
df=pd.read_csv("/kaggle/input/usa-housing/USA_Housing.csv")
df.head()
df.describe()
```

Out[]:

	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%	61480.562388	5.322283	6.299250	3.140000	29403.928702	9.975771e+05

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price
50%	68804.286404	5.970429	7.002902	4.050000	36199.406689	1.232669e+06
75%	75783.338666	6.650808	7.665871	4.490000	42861.290769	1.471210e+06
max	107701.748378	9.519088	10.759588	6.500000	69621.713378	2.469066e+0

Random Forest Regression:

```
random_forest=RandomForestRegressor(n_estimators=100)
random_forest.fit(X_train,y_train)
predictions=random_forest.predict(X_test)

mae,mse,rmse,r_squared=evaluation(y_test,predictions)
print("MAE:",mae)
print("MSE:",mse)
print("RMSE:",rmse)
print("R2 Score:",r_squared)
print("-"*30)
rmse_cross_val=rmse_cv(random_forest)
print("RMSE Cross-Validation:",rmse_cross_val)

new_row={"Model":"RandomForestRegressor","MAE":mae,"MSE":mse,"RMSE":rmse,"R2 Score":r
_squared,"RMSE (Cross-Validation)":rmse_cross_val}
models=models.append(new_row,ignore_index=True)
```

```
MAE: 18115.11067351598
MSE: 1004422414.0219476
RMSE: 31692.623968708358
R2 Score: 0.869050886899595
------------------------------
RMSE Cross-Validation: 31138.863315259332
```

Support Vector Regressor:

```
In [1]:

model_svr=SVR()

In [2]:

model_svr.fit(X_train_scal,Y_train)

Out[2]:
```

```
☑ SVR
SVR()
```

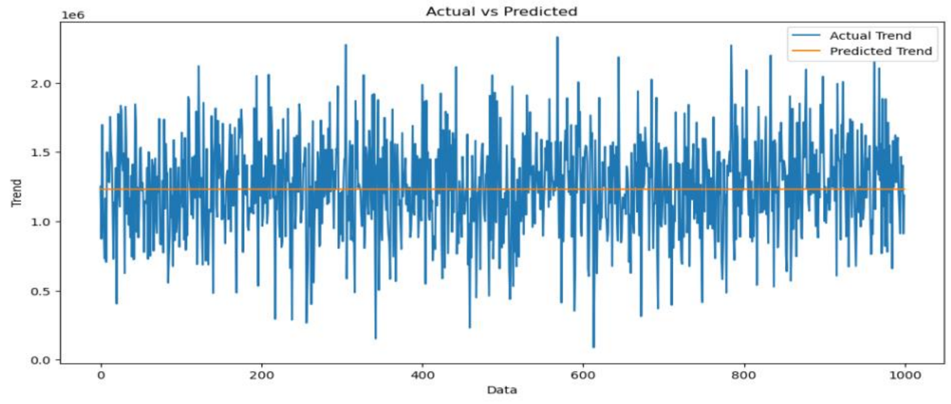
Predicting Prices:

```
In [3]:

Prediction2=model_svr.predict(X_test_scal)
```

Evaluation of Predicted Data:

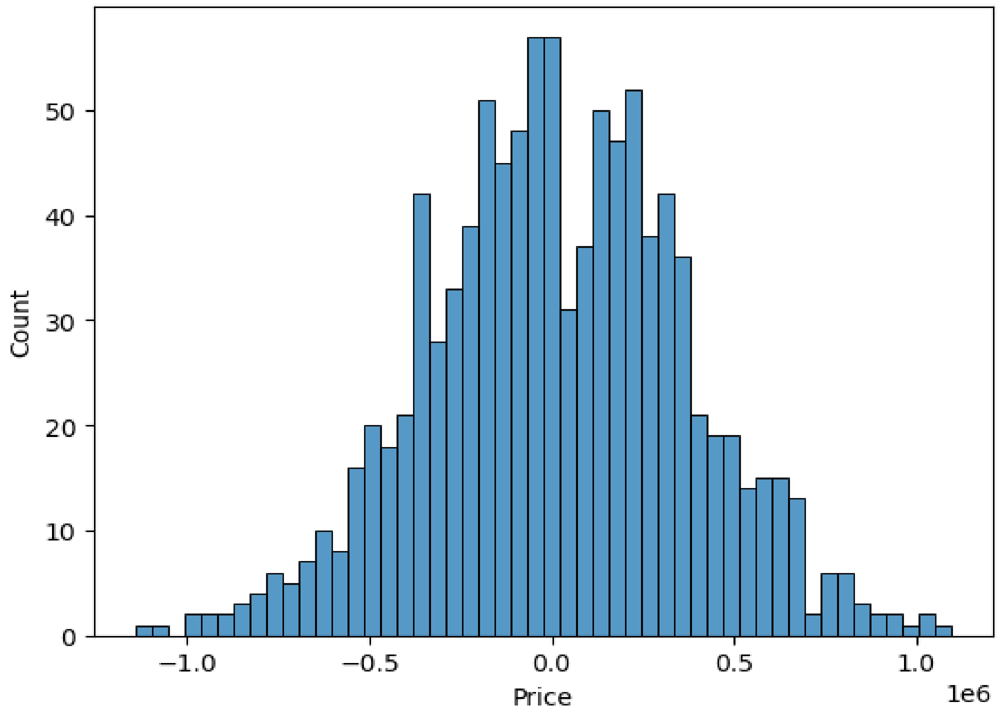
```
In [4]:
plt.figure(figsize=(12,6))
plt.plot(np.arange(len(Y_test)),Y_test,label='Actual Trend')
plt.plot(np.arange(len(Y_test)),Prediction2,label='Predicted Trend')
plt.xlabel('Data')
plt.ylabel('Trend')
plt.legend()
plt.title('Actual vs Predicted')
Out[4]:Text(0.5, 1.0, 'Actual vs Predicted')
```



```
In [5]:

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

```
Out[5]:
<Axes: xlabel='Price', ylabel='Count'>
```



```
In [6]:

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
```

Gradient Boosting Regressor:

# Import necessary libraries
import numpy as np
import pandas as pd

```
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# Generate a sample dataset (you would replace this with your real data)
data = {
    'SquareFeet': [1400, 1600, 1700, 1875, 1100, 1550, 2350, 2450, 1425, 1700],
    'Bedrooms': [3, 3, 3, 3, 2, 2, 4, 4, 3, 3],
    'Price': [245000, 312000, 279000, 308000, 199000, 219000, 405000, 324000, 319000,
255000]
}

# Create a DataFrame from the sample data
df = pd.DataFrame(data)

# Define the input features (X) and target variable (y)
X = df[['SquareFeet', 'Bedrooms']]
y = df['Price']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create and train a Gradient Boosting Regressor model
model = GradientBoostingRegressor(n_estimators=100, learning_rate=0.1,
random_state=42)
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Calculate evaluation metrics
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# Print evaluation metrics
print("Mean Squared Error (MSE):", mse)
print("Mean Absolute Error (MAE):", mae)
print("R-squared (R2) Score:", r2)
```

```
# Plot the predicted vs. actual prices
plt.scatter(y_test, y_pred)
plt.xlabel("Actual Prices")
plt.ylabel("Predicted Prices")
plt.title("Actual Prices vs. Predicted Prices")
plt.show()
```

XGBoost(Extreme Gradient Boosting) Regressor:

```
# Import necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from xgboost import XGBRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error

# Generate a sample dataset (you would replace this with your real data)
data = {
    'SquareFeet': [1400, 1600, 1700, 1875, 1100, 1550, 2350, 2450, 1425, 1700],
    'Bedrooms': [3, 3, 3, 3, 2, 2, 4, 4, 3, 3],
    'Price': [245000, 312000, 279000, 308000, 199000, 219000, 405000, 324000, 319
000, 255000]
}

# Create a DataFrame from the sample data
df = pd.DataFrame(data)

# Define the input features (X) and target variable (y)
X = df[['SquareFeet', 'Bedrooms']]
y = df['Price']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_s
tate=42)

# Create and train an XGBoost Regressor model
model = XGBRegressor()
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Calculate evaluation metrics
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
```

```
# Print evaluation metrics
print("Mean Squared Error:", mse)
print("Mean Absolute Error:", mae)

# Plot the predicted vs. actual prices
plt.scatter(y_test, y_pred)
plt.xlabel("Actual Prices")
plt.ylabel("Predicted Prices")
plt.title("Actual Prices vs. Predicted Prices")
plt.show()
```

```
MAE: 17439.918396832192
MSE: 716579004.5214689
RMSE: 26768.993341578403
R2 Score: 0.9065777666861116
------------------------------
RMSE Cross-Validation: 29698.84961808251
```

CONCLUSION:

Machine learning can be a powerful tool for house price prediction. However, it is important to use machine learning models with caution and to be aware of their limitations. For example, machine learning models are trained on historical data, so they may not be accurate in predicting house prices in a changing market. Additionally, machine learning models can be biased, depending on the data they are trained on.

Overall, machine learning can be a valuable tool for house price prediction, but it should be used in conjunction with other factors, such as expert judgment and market research.

When dealing with complex relationships, outliers, or a large number of features, Linear Regression, Random Forest Regression, Gradient Boosting Regression andXGBoost Regression may perform better.

In conclusion Phase 3, loading and preprocessing are foundational steps in the data analysis and machine learning workflows. They help ensure that the data is in the right format, contains valid and relevant information, and is suitable for the chosen analysis or modeling task. Proper data loading and preprocessing can significantly impact the quality and effectiveness of your results. The specific techniques you use will depend on your dataset and the goals of your analysis or modeling project.