Real-Estate House Price Prediction

Jay Patel (200538083), 200538083@student.georgianc.on.ca (mailto:200538083@student.georgianc.on.ca)

Deven Modi (200539304), 200539304@student.georgianc.on.ca (mailto:200539304@student.georgianc.on.ca)

Machine Learning Programming - Final Project

Introduction ¶

Overview

This project is centered around the development and application of a novel method known as Geo-Spatial Network Embedding (GSNE) for predicting house prices. The GSNE technique leverages the power of graph neural networks to capture the geo-spatial context of a neighborhood, thereby addressing the limitations of existing real estate price prediction models. These traditional models often overlook the influence of neighborhood amenities on house prices, such as proximity to a train station, a highly-ranked school, or a shopping center. By incorporating these factors, the GSNE method provides a more comprehensive and accurate approach to predicting house prices.

The GSNE method represents houses and various types of Points of Interest (POIs) as attributed nodes in multipartite networks, with the relationships between them represented as edges. This allows for the learning of embeddings of houses and POIs, which are then used to improve the performance of the house price prediction task. Extensive experiments have shown that the embeddings produced by the GSNE technique consistently enhance the accuracy of house price predictions, regardless of the downstream regression model used. Through this innovative approach, the project aims to benefit various stakeholders in the real estate market by providing a more context-aware method for predicting house prices.s.

Problem Description

The project addresses a key problem in real estate price prediction: the lack of consideration for the geo-spatial context of neighborhood amenities in existing models. Traditional models often overlook factors such as proximity to a train station, a highly-ranked school, or a shopping center, which can significantly influence a house's price. This project proposes a novel method, the Geo-Spatial Network Embedding (GSNE), which leverages graph neural networks to capture this geo-spatial context, thereby aiming to enhance the accuracy of house price predictions.

Context of the Problem

The context of the problem lies in the field of real estate price prediction. Real estate significantly contributes to all major economies around the world, and house prices have a direct impact on stakeholders, ranging from house buyers to financing companies. A variety of techniques have been developed for real estate price prediction, most of which rely on different house features to build a variety of prediction models. Some later works introduced spatial regression models to improve prediction performance, recognizing the effect of spatial dependence on house prices. However, these models often fail to consider the geo-spatial context of neighborhood amenities, such as how close a house is to a train station, a highly-ranked school, or a shopping center. Such contextual information can significantly influence a user’s interest in a house and thereby its price. This lack of contextual information in current models is the key problem that this project aims to address.89-x.

Limitation about approach

Traditional real estate price prediction models have several limitations. Most of these models rely heavily on house features to predict prices, such as the size of the house, the number of rooms, the age of the house, and so on. While these features are undoubtedly important, they do not provide a complete picture of what drives house prices.

One significant limitation is that these models often overlook the effect of spatial dependence on house prices. Some later works introduced spatial regression models to improve prediction performance, recognizing this effect. However, even these models fail to consider the geo-spatial context of neighborhood amenities. For example, the proximity of a house to a train station, a highly-ranked school, or a shopping center can significantly influence a user's interest in a house and thereby its price. This lack of contextual information in current models is a key problem that needs to be addressed for more accurate and comprehensive real estate price predictin

Another limitation of traditional models is their inability to effectively capture the complex relationships between different neighborhood amenities and house prices. The value of a house is not only determined by its own features but also by the features of its surrounding environment. For example, a house located near a noisy factory may be less desirable than a similar house located near a quiet park. Traditional models often fail to capture these complex relationships, leading to inaccurate predictions. The proposed Geo-Spatial Network Embedding (GSNE) method addresses this limitation by learning the embeddings of houses and various Points of Interest (POIs) in multipartite networks, thereby capturing the complex relationships between houses and their surrounding amenities. This approach significantly improves the performance of the house price prediction task, providing a more comprehensive and accurate model for real estate price prediction.0789-x.

Solutions

Machine learning applications are disrupting the real estate industry by incorporating non-traditional data sources into price prediction models. This approach helps answer questions like: Which house should I buy or build to maximize my return? Where or when should I do so? What is its optimum rent or sale price? By using machine learning algorithms, these models can learn from past data and make accurate predictions about future prices.

Another approach involves the use of data mining and machine learning techniques to adjust pricing models and processes, and realize trend estimation that changes over time. This results in a pricing model with advantages such as dynamics, accuracy, and flexibility over the original model. Data mining techniques can uncover hidden patterns and relationships in the data, which can then be used to make more accurate price predictions.

Deep learning, a subset of machine learning, can automatically learn complex patterns in the data and make more accurate predictions. This approach overcomes the limitations of traditional methods, which may be prone to human error and may not be able to capture complex patterns in the data. Deep learning models can process a large amount of data and learn from it, improving their accuracy over time.

One innovative approach is to calculate and analyze the entropy and information gain of various influencing factors of house prices and extract the main factors affecting real estate prices. This method facilitates the analysis of the relationship between the main factors and establishes a multiple linear regression model according to the relationship between each factor and the real estate price. This approach provides a more scientific and rational basis for real estate price prediction. These solutions leverage advanced technologies and innovative approaches to address the limitations of traditional models, providing more accurate and comprehensive real estate price predictions. They represent the future of real estate price prediction, offering potential benefits for buyers, sellers, and the real estate industry as a whole.

The Background of the Project

Reference	Explanation	Dataset	Future Improvement
Basu, S., Thibodeau, T.G.: Analysis of spatial autocorrelation in house prices. The Journal of Real Estate Finance and Economics 17(1), 61–85 (1998)	This paper analyzes the spatial autocorrelation in house prices, which is a crucial factor in real estate price prediction.	Not specified	Future work could explore other factors that might influence spatial autocorrelation in house prices.
Bojchevski, A., G¨unnemann, S.: Deep gaussian embedding of attributed graphs: Unsupervised inductive learning via ranking. arXiv preprint arXiv:1707.03815 (2017)	This paper presents a method for deep Gaussian embedding of attributed graphs, which could be useful for capturing complex relationships in data.	Not specified	Future improvements could include applying this method to other types of data or problems.
Bourassa, S., Cantoni, E., Hoesli, M.: Predicting house prices with spatial dependence: a comparison of alternative methods. Journal of Real Estate Research 32(2), 139–159 (2010)	This paper compares different methods for predicting house prices with spatial dependence, providing valuable insights for real estate price prediction.	Not specified	Future work could involve developing new methods or improving existing ones for predicting house prices with spatial dependence.

Reference	Explanation	Dataset	Future Improvement
Bourassa, S.C., Cantoni, E., Hoesli, M.: Spatial dependence, housing submarkets, and house price prediction. The Journal of Real Estate Finance and Economics 35(2), 143–160 (2007)	This paper discusses the concept of spatial dependence and housing submarkets in the context of house price prediction.	Not specified	Future work could explore the impact of various housing submarkets on spatial dependence and house price prediction.
Bourassa, S.C., Hoesli, M., Peng, V.S.: Do housing submarkets really matter? Journal of Housing Economics 12(1), 12–28 (2003)	This paper investigates the significance of housing submarkets in the real estate industry.	Not specified	Future research could delve deeper into the role and impact of different housing submarkets.
Cai, H., Zheng, V.W., Chang, K.C.C.: A comprehensive survey of graph embedding: Problems, techniques, and applications. IEEE Transactions on Knowledge and Data Engineering 30(9), 1616–1637 (2018)	This paper provides a comprehensive survey of graph embedding, discussing various problems, techniques, and applications.	Not specified	Future work could focus on addressing the problems identified and improving the techniques discussed.
Case, B., Clapp, J., Dubin, R., Rodriguez, M.: Modeling spatial and temporal house price patterns: A comparison of four models. The Journal of Real Estate Finance and Economics 29(2), 167–191 (2004)	This paper compares four models for modeling spatial and temporal house price patterns, contributing to the field of real estate price prediction.	Not specified	Future research could develop new models or improve the existing ones for better prediction of spatial and temporal house price patterns.
Chen, T., Guestrin, C.: Xgboost: A scalable tree boosting system. In: Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining, pp. 785–794 (2016)	This paper presents XGBoost, a scalable tree boosting system, which could be used for various prediction tasks, including real estate price prediction.	Not specified	Future work could explore the application of XGBoost in different domains and for different prediction tasks.
Chen, X., Wei, L., Xu, J.: House price prediction using lstm. arXiv preprint arXiv:1709.08432 (2017)	This paper discusses the use of Long Short-Term Memory (LSTM) for house price prediction.	Not specified	Future research could focus on improving the LSTM model for better house price prediction.
Clevert, D.A., Unterthiner, T., Hochreiter, S.: Fast and accurate deep network learning by exponential linear units (ELUs). arXiv preprint arXiv:1511.07289 (2015)	This paper presents a method for fast and accurate deep network learning using Exponential Linear Units (ELUs), which could be applied to various learning tasks, including real estate price prediction.	Not specified	Future work could focus on improving the ELU-based deep learning method for better performance.
Dubin, R.A.: Predicting house prices using multiple listings data. The Journal of Real Estate Finance and Economics 17(1),	This paper discusses the use of multiple listings data for predicting house prices, providing a new perspective in the field of real estate price	Not specified	Future research could explore other types of data for house price prediction

Methodology

To implement the gradient boosting framework on both categorical and numerical data, here XGBoost is used for the implementation of the concept. This approach allows us to leverage the power of XGBoost's speed and performance for efficient and effective real estate price prediction. It handles both categorical and numerical features, making it a versatile tool for this task.

XGBoost

XGBoost, short for eXtreme Gradient Boosting, is a powerful machine learning algorithm widely used in the field of data science and machine learning. It's an implementation of gradient-boosting machines, designed to be highly efficient, flexible, and portable. In the context of house price prediction, XGBoost can be particularly useful due to its ability to handle a variety of data types, including both categorical and numerical data, and its capability to model complex non-linear relationships. It works by building an ensemble of decision trees in a sequential manner, where each new tree is built to correct the errors made by the existing ensembl T

The algorithm also includes several regularization parameters to prevent overfitting, making it robust to noise and outliers in the data. This makes XGBoost a great choice for building a house price prediction model, as it can effectively capture the intricate patterns in the data and provide accurate predictions.

Insights and Execution

Detailed Insights The current version includes the following steps:

- **Loading Datasets**: Importing datasets in CSV or other formats that contain historical house prices and related features.
- **Preprocessing of Data**: Cleaning the data, handling missing values, outliers, and performing exploratory data analysis. This step also involves feature engineering where new features are created from the existing data that might help improve the performance of the prediction model.
- **Model Training**: Training the XGBoost model using a subset of the collected data. This involves feeding the data into the model so it can learn the relationships between the features and the target variable (house prices).
- **Model Evaluation**: Evaluating the performance of the model using appropriate metrics, such as Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), or R-squared. This helps us understand how well our model is performing.
- **Hyperparameter Tuning**: Tuning the hyperparameters of the XGBoost model to improve its performance. This could involve techniques like Grid Search or Random Search.
- **Prediction**: Using the trained model to predict house prices for new data. This is where we can see the results of our work and use it for practical applications.
- **Feature Importance Analysis**: Analyzing the feature importance scores provided by XGBoost to understand which features are most influential in predicting house prices.
- **Model Deployment (Optional)**: Deploying the model in a suitable environment where it can be used to make predictions in real-time. This could be a local server, cloud platform, or even embedded in a mobile or web application.
- **Monitoring and Updating (Optional)**: Continuously monitoring the model's performance over time and updating or retraining it as necessary. This ensures that the model remains effective as new data comes in.in a mobile or.new data comes in.

Reference: https://arxiv.org/pdf/2009.00254v1.pdf (https://arxiv.org/pdf/2009.00254v1.pdf)

Exploratory Data Analysis (EDA).

In [1]:
```
import numpy as np                                  # Importing Numpy important pytho
import pandas as pd                                # Importing Pandas library for EDA
import matplotlib.pyplot as plt
import seaborn as sns                             # Importing Seaborn for Ploting gra

import warnings
warnings.filterwarnings("ignore")

from sklearn.model_selection import KFold
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score,classification_report,confusion_mat
```

In [2]:
```
df = pd.read_csv("dataset_ori.csv")    # Fetching Dataset using python Library
df.head()
```

Out[2]:

	Home	SqFt	Bedrooms	Bathrooms	Offers	Neighborhood	Price
0	1	1790	2	2	2	East	114300
1	2	2030	4	2	3	East	114200
2	3	1740	3	2	1	East	114800
3	4	1980	3	2	3	East	94700
4	5	2130	3	3	3	East	119800

Shape Of Dataset..

In [3]:
```
df.shape
```

Out[3]:
```
(467, 7)
```

Explore Continues Features.

In [4]:
```
df.describe()
```

Out[4]:

	Home	SqFt	Bedrooms	Bathrooms	Offers	Price
count	467.000000	467.000000	467.000000	467.000000	467.000000	467.000000
mean	234.000000	2023.832976	3.021413	2.498929	2.599572	132572.591006
std	134.955548	210.089083	0.705262	0.525630	1.040097	27144.107920
min	1.000000	1450.000000	2.000000	2.000000	1.000000	69100.000000
25%	117.500000	1910.000000	3.000000	2.000000	2.000000	111600.000000
50%	234.000000	2020.000000	3.000000	2.000000	3.000000	127700.000000
75%	350.500000	2150.000000	3.000000	3.000000	3.000000	150200.000000
max	467.000000	2590.000000	5.000000	4.000000	6.000000	211200.000000

Inorder to understand Data Types Of Entire Data.

In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 467 entries, 0 to 466
Data columns (total 7 columns):
 #   Column        Non-Null Count  Dtype 
---  ------        --------------  ----- 
 0   Home          467 non-null    int64 
 1   SqFt          467 non-null    int64 
 2   Bedrooms      467 non-null    int64 
 3   Bathrooms     467 non-null    int64 
 4   Offers        467 non-null    int64 
 5   Neighborhood  467 non-null    object
 6   Price         467 non-null    int64 
dtypes: int64(6), object(1)
memory usage: 25.7+ KB
```

Null data Check in Database.

In [6]:
```
df.isnull().sum()     # inorder to check missing value in dataset
```

Out[6]:
```
Home            0
SqFt            0
Bedrooms        0
Bathrooms       0
Offers          0
Neighborhood    0
Price           0
dtype: int64
```

----} Here, Dataset has no null values.

Duplicate Data Check in DataBase.

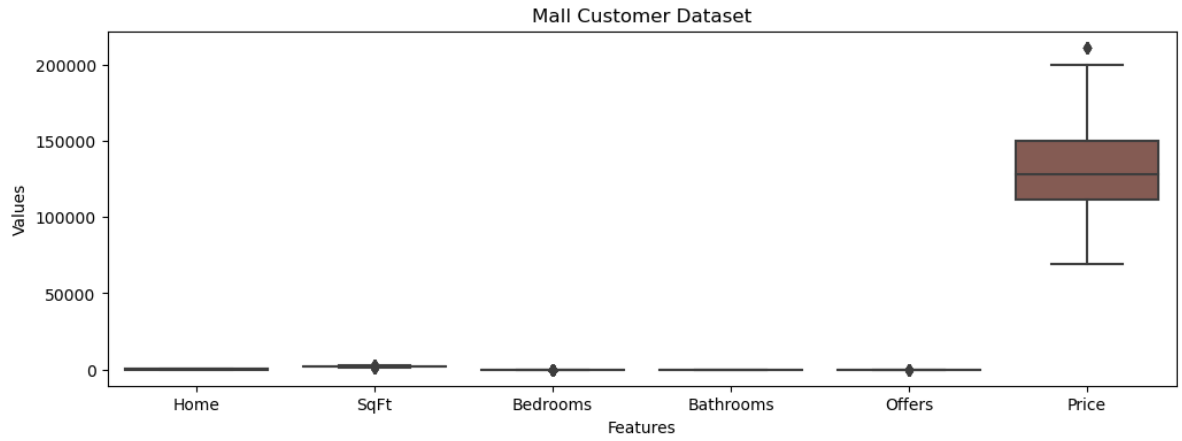
In [7]:
```
df.duplicated().sum()      # Provides total number of duplicate data in databa
```

Out[7]:
```
0
```

----} Here, Dataset has no Duplicate data.

Outliers Check and Handling.

In [8]:
```
# Create a box plot of all features using Seaborn.

plt.figure(figsize = (12,4))          # Sets plot size.

sns.boxplot(data = df)

# Title of Dataset.
plt.title('Mall Customer Dataset')

plt.xlabel('Features')
plt.ylabel('Values')

# Display the plot.
plt.show()
```

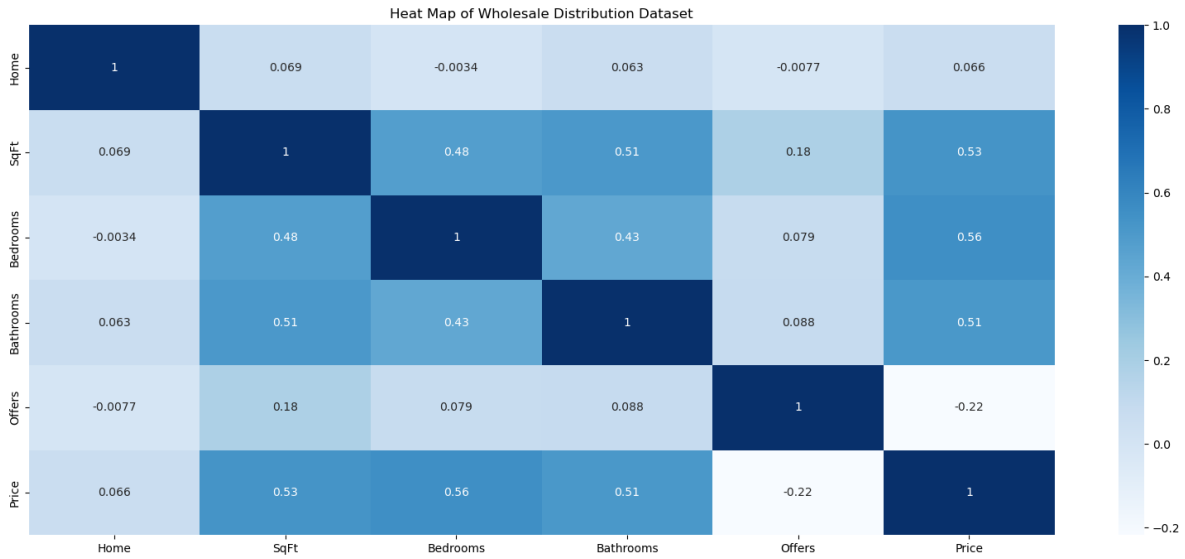


----} Here, it can be observe that outliers in Mall Customer dataset is not present.

Implementing Correlation Heat Map.

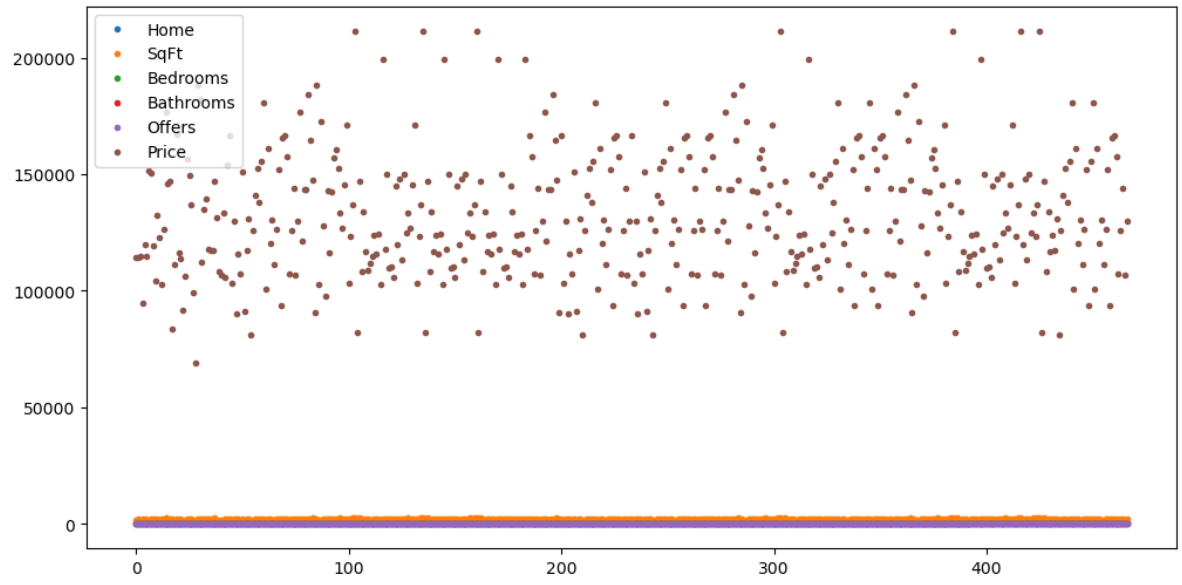
In [9]:
```
# Using SNS implementing Heatmap inorder to check Corelation amongs Features.

plt.figure(figsize = (20,8))
sns.heatmap(df.corr(), annot = True, cmap="Blues")
plt.title('Heat Map of Wholesale Distribution Dataset')
```

Out[9]: Text(0.5, 1.0, 'Heat Map of Wholesale Distribution Dataset')



Analysing Trends Among the features.

In [10]:
```
# No obvious trends in the data as sequences
%matplotlib inline
df.plot(figsize=(12,6), style='.');
```



In [11]:
```
df.head()
```

Out[11]:

	Home	SqFt	Bedrooms	Bathrooms	Offers	Neighborhood	Price
0	1	1790	2	2	2	East	114300
1	2	2030	4	2	3	East	114200
2	3	1740	3	2	1	East	114800
3	4	1980	3	2	3	East	94700
4	5	2130	3	3	3	East	119800

Label Encoding.

In [12]:
```
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
df["Neighborhood"] = le.fit_transform(df["Neighborhood"])
# df["mainroad"] = le.fit_transform(df["mainroad"])
# df["guestroom"] = le.fit_transform(df["guestroom"])
# df["basement"] = le.fit_transform(df["basement"])
# df["hotwaterheating"] = le.fit_transform(df["hotwaterheating"])
# df["airconditioning"] = le.fit_transform(df["airconditioning"])
# df["prefarea"] = le.fit_transform(df["prefarea"])
# df["furnishingstatus"] = le.fit_transform(df["furnishingstatus"])
```

In [13]:
```
df.head()
```

Out[13]:

	Home	SqFt	Bedrooms	Bathrooms	Offers	Neighborhood	Price
0	1	1790	2	2	2	0	114300
1	2	2030	4	2	3	0	114200
2	3	1740	3	2	1	0	114800
3	4	1980	3	2	3	0	94700
4	5	2130	3	3	3	0	119800

Implementing K-Fold Cross & Gaussian NB Model.

In [14]:
```
# Split the data into features (X) and target (y)

x = df.iloc[:, :-1]
y = df.iloc[:, -1]
```

In [15]:
```
kfold = KFold(n_splits = 11, shuffle=True, random_state=0)
```

In [16]:
```
accuracy_scores = []
```

In [17]:
```
for train_index, test_index in kfold.split(x):

    x_train, x_test = x.iloc[train_index], x.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]

    model = GaussianNB()
    model.fit(x_train, y_train)

    y_pred = model.predict(x_test)

    accuracy = accuracy_score(y_test, y_pred)

    accuracy_scores.append(accuracy)
```

In [18]:
```
mean_accuracy = np.mean(accuracy_scores)
print("Mean accuracy score:", mean_accuracy*100)
```

```
Mean accuracy score: 69.78254303835699
```

In [19]:
```
print(accuracy_score(y_test, y_pred)*100)
```

```
76.19047619047619
```

In [20]:
```
print(classification_report(y_test, y_pred))
```

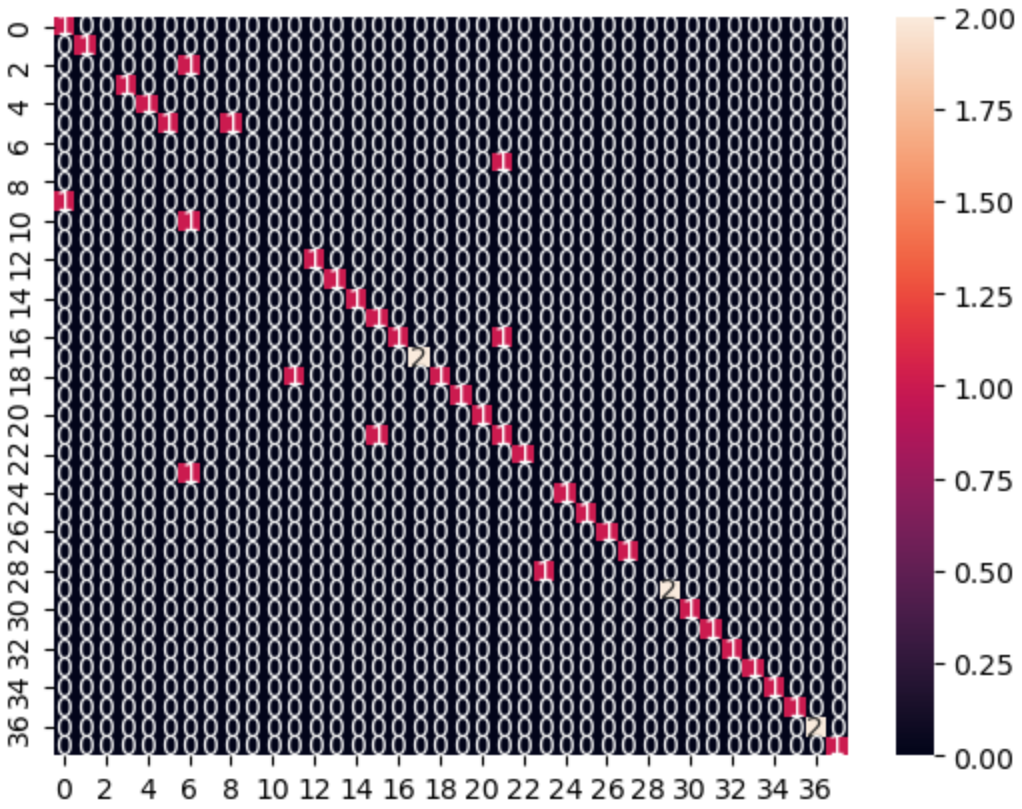
```
              precision    recall  f1-score   support

       81300       0.50      1.00      0.67         1
       82300       1.00      1.00      1.00         1
       90300       0.00      0.00      0.00         1
       93600       1.00      1.00      1.00         1
      100900       1.00      1.00      1.00         1
      102500       1.00      0.50      0.67         2
      103200       0.00      0.00      0.00         0
      104000       0.00      0.00      0.00         1
      105600       0.00      0.00      0.00         0
      107300       0.00      0.00      0.00         1
      108200       0.00      0.00      0.00         1
      108500       0.00      0.00      0.00         0
      110400       1.00      1.00      1.00         1
      111100       1.00      1.00      1.00         1
      117000       1.00      1.00      1.00         1
      117800       0.50      1.00      0.67         1
      119700       1.00      0.50      0.67         2
      120500       1.00      1.00      1.00         2
      121300       1.00      0.50      0.67         2
      123600       1.00      1.00      1.00         1
      124500       1.00      1.00      1.00         1
      125700       0.33      0.50      0.40         2
      127700       1.00      1.00      1.00         1
      129800       0.00      0.00      0.00         1
      130800       1.00      1.00      1.00         1
      134000       1.00      1.00      1.00         1
      142600       1.00      1.00      1.00         1
      143400       1.00      1.00      1.00         1
      149300       0.00      0.00      0.00         1
      150200       1.00      1.00      1.00         2
      151900       1.00      1.00      1.00         1
      161300       1.00      1.00      1.00         1
      164800       1.00      1.00      1.00         1
      166700       1.00      1.00      1.00         1
      171000       1.00      1.00      1.00         1
      172500       1.00      1.00      1.00         1
      176500       1.00      1.00      1.00         2
      188300       1.00      1.00      1.00         1

    accuracy                           0.76        42
   macro avg       0.72      0.71      0.70        42
weighted avg       0.80      0.76      0.77        42
```

In [21]:
```
print(confusion_matrix(y_test, y_pred))
```

```
[[1 0 0 ... 0 0 0]
 [0 1 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 ...
 [0 0 0 ... 1 0 0]
 [0 0 0 ... 0 2 0]
 [0 0 0 ... 0 0 1]]
```

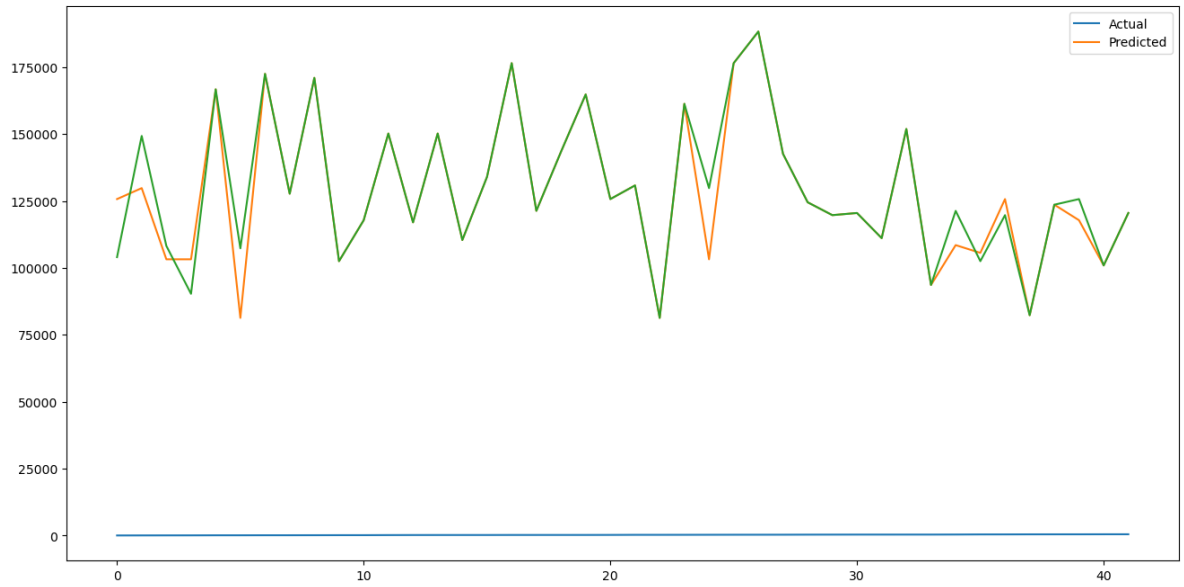
In [22]:
```
sns.heatmap(confusion_matrix(y_test, y_pred),annot =True, fmt='.2g')
```

Out[22]: <AxesSubplot:>



In [23]:
```
#visualizatation:
test = pd.DataFrame({'Pred_values':y_pred,'Actual':y_test})
fig=plt.figure(figsize=(16,8))
test = test.reset_index()
plt.plot(test[:50])
plt.legend(['Actual','Predicted'])
```

Out[23]:
```
<matplotlib.legend.Legend at 0x12946f27610>
```



Price Prediction.

In [24]:
```
test = pd.read_csv("test_dataset_ori.csv")
```

In [25]:
```
test.head()
```

Out[25]:

	home	SqFt	Bedrooms	Bathrooms	Offers	Neighborhood	Price
0	1	1930	4	2	3	West	100000
1	2	1678	5	5	4	East	120000
2	3	2903	6	3	2	West	80000
3	4	832	3	2	1	North	76000
4	5	8732	2	1	1	West	135000

Label Encoding.

In [26]:
```
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()

test["Neighborhood"] = le.fit_transform(test["Neighborhood"])
```

In [27]:
```
test = test.drop(['Price'], axis=1)
```

In [28]:
```
y_pred = model.predict(test)
y_pred
```

Out[28]:
```
array([129800, 125700, 129800, 103200, 103200], dtype=int64)
```

In [29]:
```
series = pd.Series(y_pred)
series
```

Out[29]:
```
0    129800
1    125700
2    129800
3    103200
4    103200
dtype: int64
```

Implement Grid Search CV to find optimal hyperparameters for Following Models LR, SVM, MLP, RF, DTC.

---} Splitting Dataset into 80:10:10 Train, Validation & Test Dataset Ratio.

In [30]:
```
# Split the data into features (X) and target (y)

test = pd.read_csv("test_dataset_ori.csv")
test["Neighborhood"] = le.fit_transform(test["Neighborhood"])
x = df.iloc[:, :-1]
y = df.iloc[:, -1]
```

In [31]:
```
for train_index, test_index in kfold.split(x):

    x_train, x_test = x.iloc[train_index], x.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]
```

In [32]:
```
# from sklearn.model_selection import train_test_split
```

In [33]:
```
# var_x = df.drop('price', axis=1)
# var_y = df['price']

# x_train, x_test, y_train, y_test = train_test_split(var_x, var_y, test_size=
# x_test, x_val, y_test, y_val = train_test_split(x_test, y_test, test_size=0.
```

----} Logistic Regression Model.

In [34]:
```
import joblib                                                    # inorder t
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
from sklearn import metrics
from sklearn.metrics import accuracy_score, precision_score, recall_score, cla
from time import time



import warnings                                                  # importing
warnings.filterwarnings('ignore', category=FutureWarning)
warnings.filterwarnings('ignore', category=DeprecationWarning)
```

In [35]:
```
def print_results(results):
    print('BEST Params: {}\n'.format(results.best_params_))

    mean = results.cv_results_['mean_test_score']
    stds = results.cv_results_['std_test_score']

    for mean, std, params in zip(mean, stds, results.cv_results_['params']):
        print('{} (+/-{}) for {}'.format(round(mean, 3), round(std * 2, 3), pa
```

In [36]:
```
lr = LogisticRegression()
parameters = {
    'C': [10, 50, 100, 500, 1000]

}

cv = GridSearchCV(lr, parameters, cv=5)
cv.fit(x_train, y_train.values.ravel())

print_results(cv)
```

```
BEST Params: {'C': 10}

0.024 (+/-0.033) for {'C': 10}
0.019 (+/-0.028) for {'C': 50}
0.019 (+/-0.028) for {'C': 100}
0.019 (+/-0.028) for {'C': 500}
0.019 (+/-0.028) for {'C': 1000}
```

In [37]:
```
cv.best_estimator_
```

Out[37]:
```
LogisticRegression(C=10)
```

In [38]:
```
joblib.dump(cv.best_estimator_, 'LR_model.pkl')
```

Out[38]:
```
['LR_model.pkl']
```

----} Random Forest Classifier Model.

In [39]:
```
import joblib
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn import metrics
from sklearn.metrics import accuracy_score, precision_score, recall_score, cla
from time import time

import warnings
warnings.filterwarnings('ignore', category=FutureWarning)
warnings.filterwarnings('ignore', category=DeprecationWarning)
```

In [40]:
```
def print_results(results):
    print('BEST Params: {}\n'.format(results.best_params_))

    mean = results.cv_results_['mean_test_score']
    stds = results.cv_results_['std_test_score']

    for mean, std, params in zip(mean, stds, results.cv_results_['params']):
        print('{} (+/-{}) for {}'.format(round(mean, 3), round(std * 2, 3), pa
```

In [41]:
```
rf = RandomForestClassifier()
parameters = {
    'n_estimators': [5, 50, 250],
    'max_depth': [2, 4, 8, 16, 32]

}

cv =GridSearchCV(rf, parameters, cv=5)
cv.fit(x_train, y_train.values.ravel())

print_results(cv)
```

```
BEST Params: {'max_depth': 32, 'n_estimators': 250}

0.127 (+/-0.058) for {'max_depth': 2, 'n_estimators': 5}
0.259 (+/-0.13) for {'max_depth': 2, 'n_estimators': 50}
0.289 (+/-0.066) for {'max_depth': 2, 'n_estimators': 250}
0.278 (+/-0.064) for {'max_depth': 4, 'n_estimators': 5}
0.44 (+/-0.193) for {'max_depth': 4, 'n_estimators': 50}
0.468 (+/-0.163) for {'max_depth': 4, 'n_estimators': 250}
0.506 (+/-0.254) for {'max_depth': 8, 'n_estimators': 5}
0.664 (+/-0.269) for {'max_depth': 8, 'n_estimators': 50}
0.661 (+/-0.303) for {'max_depth': 8, 'n_estimators': 250}
0.579 (+/-0.213) for {'max_depth': 16, 'n_estimators': 5}
0.661 (+/-0.345) for {'max_depth': 16, 'n_estimators': 50}
0.68 (+/-0.352) for {'max_depth': 16, 'n_estimators': 250}
0.628 (+/-0.306) for {'max_depth': 32, 'n_estimators': 5}
0.664 (+/-0.358) for {'max_depth': 32, 'n_estimators': 50}
0.692 (+/-0.324) for {'max_depth': 32, 'n_estimators': 250}
```

In [42]:
```
cv.best_estimator_                                          # Stores best est
```

Out[42]:
```
RandomForestClassifier(max_depth=32, n_estimators=250)
```

In [43]:
```
joblib.dump(cv.best_estimator_, 'RF_model.pkl')
```

Out[43]:
```
['RF_model.pkl']
```

----} Support Vector Machine Model.

In [44]:
```
import joblib
import pandas as pd
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
from sklearn import metrics
from sklearn.metrics import accuracy_score, precision_score, recall_score, cla
from time import time

import warnings
warnings.filterwarnings('ignore', category=FutureWarning)
warnings.filterwarnings('ignore', category=DeprecationWarning)
```

In [45]:
```
def print_results(results):
    print('BEST Params: {}\n'.format(results.best_params_))

    means = results.cv_results_['mean_test_score']
    stds = results.cv_results_['std_test_score']

    for mean,std, params in zip(means, stds, results.cv_results_['params']):
        print('{} (+/-{}) for {}'.format(round(mean, 3), round(std * 2, 3), pa
```

In [46]:
```
svc = SVC()
parameters = {
    'kernel': ['linear', 'rbf'],
    'C': [0.01, 0.1, 10]
}

cv = GridSearchCV(svc, parameters, cv=5)
cv.fit(x_train, y_train.values.ravel())

print_results(cv)
```

```
BEST Params: {'C': 10, 'kernel': 'linear'}

0.412 (+/-0.218) for {'C': 0.01, 'kernel': 'linear'}
0.021 (+/-0.009) for {'C': 0.01, 'kernel': 'rbf'}
0.485 (+/-0.219) for {'C': 0.1, 'kernel': 'linear'}
0.021 (+/-0.009) for {'C': 0.1, 'kernel': 'rbf'}
0.668 (+/-0.299) for {'C': 10, 'kernel': 'linear'}
0.059 (+/-0.021) for {'C': 10, 'kernel': 'rbf'}
```

In [47]:
```
cv.best_estimator_                                          # Stores best est
```

Out[47]:
```
SVC(C=10, kernel='linear')
```

In [48]:
```
joblib.dump(cv.best_estimator_, 'SVM_model.pkl')
```

Out[48]:
```
['SVM_model.pkl']
```

---} Multilayer Perceptron Model.

In [49]:
```
import joblib
import pandas as pd
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import GridSearchCV
from sklearn import metrics
from sklearn.metrics import accuracy_score, precision_score, recall_score, cla
from time import time

import warnings
warnings.filterwarnings('ignore', category=FutureWarning)
warnings.filterwarnings('ignore', category=DeprecationWarning)
```

In [50]:
```
def print_results(results):
    print('BEST Params: {}\n'.format(results.best_params_))

    means = results.cv_results_['mean_test_score']
    stds = results.cv_results_['std_test_score']

    for mean,std, params in zip(means, stds, results.cv_results_['params']):
        print('{} (+/-{}) for {}'.format(round(mean, 3), round(std * 2, 3), pa
```

In [51]:
```
mlp = MLPClassifier()
parameters = {
    'hidden_layer_sizes': [(10,), (50,), (100,)],
    'activation': ['relu', 'tanh', 'logistic'],
    'learning_rate': ['constant', 'invscaling', 'adaptive'],
}

cv = GridSearchCV(mlp, parameters, cv=5)
cv.fit(x_train, y_train.values.ravel())

print_results(cv)
```

```
BEST Params: {'activation': 'tanh', 'hidden_layer_sizes': (10,), 'learning_ra
te': 'constant'}

0.016 (+/-0.012) for {'activation': 'relu', 'hidden_layer_sizes': (10,), 'lea
rning_rate': 'constant'}
0.016 (+/-0.012) for {'activation': 'relu', 'hidden_layer_sizes': (10,), 'lea
rning_rate': 'invscaling'}
0.014 (+/-0.027) for {'activation': 'relu', 'hidden_layer_sizes': (10,), 'lea
rning_rate': 'adaptive'}
0.016 (+/-0.019) for {'activation': 'relu', 'hidden_layer_sizes': (50,), 'lea
rning_rate': 'constant'}
0.014 (+/-0.018) for {'activation': 'relu', 'hidden_layer_sizes': (50,), 'lea
rning_rate': 'invscaling'}
0.005 (+/-0.012) for {'activation': 'relu', 'hidden_layer_sizes': (50,), 'lea
rning_rate': 'adaptive'}
0.014 (+/-0.009) for {'activation': 'relu', 'hidden_layer_sizes': (100,), 'le
arning_rate': 'constant'}
0.014 (+/-0.009) for {'activation': 'relu', 'hidden_layer_sizes': (100,), 'le
arning_rate': 'invscaling'}
0.014 (+/-0.018) for {'activation': 'relu', 'hidden_layer_sizes': (100,), 'le
arning_rate': 'adaptive'}
0.026 (+/-0.009) for {'activation': 'tanh', 'hidden_layer_sizes': (10,), 'lea
rning_rate': 'constant'}
0.026 (+/-0.009) for {'activation': 'tanh', 'hidden_layer_sizes': (10,), 'lea
rning_rate': 'invscaling'}
0.016 (+/-0.012) for {'activation': 'tanh', 'hidden_layer_sizes': (10,), 'lea
rning_rate': 'adaptive'}
0.016 (+/-0.012) for {'activation': 'tanh', 'hidden_layer_sizes': (50,), 'lea
rning_rate': 'constant'}
0.019 (+/-0.019) for {'activation': 'tanh', 'hidden_layer_sizes': (50,), 'lea
rning_rate': 'invscaling'}
0.021 (+/-0.043) for {'activation': 'tanh', 'hidden_layer_sizes': (50,), 'lea
rning_rate': 'adaptive'}
0.014 (+/-0.018) for {'activation': 'tanh', 'hidden_layer_sizes': (100,), 'le
arning_rate': 'constant'}
0.016 (+/-0.019) for {'activation': 'tanh', 'hidden_layer_sizes': (100,), 'le
arning_rate': 'invscaling'}
0.019 (+/-0.024) for {'activation': 'tanh', 'hidden_layer_sizes': (100,), 'le
arning_rate': 'adaptive'}
0.021 (+/-0.009) for {'activation': 'logistic', 'hidden_layer_sizes': (10,),
'learning_rate': 'constant'}
0.021 (+/-0.009) for {'activation': 'logistic', 'hidden_layer_sizes': (10,),
'learning_rate': 'invscaling'}
0.024 (+/-0.0) for {'activation': 'logistic', 'hidden_layer_sizes': (10,), 'l
earning_rate': 'adaptive'}
0.014 (+/-0.018) for {'activation': 'logistic', 'hidden_layer_sizes': (50,),
'learning_rate': 'constant'}
0.016 (+/-0.024) for {'activation': 'logistic', 'hidden_layer_sizes': (50,),
'learning_rate': 'invscaling'}
0.021 (+/-0.018) for {'activation': 'logistic', 'hidden_layer_sizes': (50,),
'learning_rate': 'adaptive'}
0.016 (+/-0.019) for {'activation': 'logistic', 'hidden_layer_sizes': (100,),
'learning_rate': 'constant'}
0.016 (+/-0.019) for {'activation': 'logistic', 'hidden_layer_sizes': (100,),
'learning_rate': 'invscaling'}
```

```
0.014 (+/-0.018) for {'activation': 'logistic', 'hidden_layer_sizes': (100,),
'learning_rate': 'adaptive'}
```

In [52]:
```
cv.best_estimator_                                                    # Stores best est
```

Out[52]:
```
MLPClassifier(activation='tanh', hidden_layer_sizes=(10,))
```

In [53]:
```
joblib.dump(cv.best_estimator_, 'MLP_model.pkl')
```

Out[53]:
```
['MLP_model.pkl']
```

----} Boosting Model.

In [54]:
```
import joblib
import pandas as pd
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import GridSearchCV
from sklearn import metrics
from sklearn.metrics import accuracy_score, precision_score, recall_score, cla
from time import time

import warnings
warnings.filterwarnings('ignore', category=FutureWarning)
warnings.filterwarnings('ignore', category=DeprecationWarning)
```

In [55]:
```
def print_results(results):
    print('BEST Params: {}\n'.format(results.best_params_))

    means = results.cv_results_['mean_test_score']
    stds = results.cv_results_['std_test_score']

    for mean,std, params in zip(means, stds, results.cv_results_['params']):
        print('{} (+/-{}) for {}'.format(round(mean, 3), round(std * 2, 3), pa
```

In [56]:
```
gb = GradientBoostingClassifier()
parameters = {
    'n_estimators': [2, 4, 6, 12],
    'max_depth': [7],
    'learning_rate': [0.01, 0.1, 1, 10]
}

cv = GridSearchCV(gb, parameters, cv=5)
cv.fit(x_train, y_train.values.ravel())

print_results(cv)
```

```
BEST Params: {'learning_rate': 0.1, 'max_depth': 7, 'n_estimators': 4}

0.687 (+/-0.326) for {'learning_rate': 0.01, 'max_depth': 7, 'n_estimators':
2}
0.694 (+/-0.302) for {'learning_rate': 0.01, 'max_depth': 7, 'n_estimators':
4}
0.701 (+/-0.32) for {'learning_rate': 0.01, 'max_depth': 7, 'n_estimators':
6}
0.711 (+/-0.303) for {'learning_rate': 0.01, 'max_depth': 7, 'n_estimators':
12}
0.706 (+/-0.297) for {'learning_rate': 0.1, 'max_depth': 7, 'n_estimators':
2}
0.72 (+/-0.31) for {'learning_rate': 0.1, 'max_depth': 7, 'n_estimators': 4}
0.718 (+/-0.287) for {'learning_rate': 0.1, 'max_depth': 7, 'n_estimators':
6}
0.715 (+/-0.307) for {'learning_rate': 0.1, 'max_depth': 7, 'n_estimators': 1
2}
0.718 (+/-0.287) for {'learning_rate': 1, 'max_depth': 7, 'n_estimators': 2}
0.72 (+/-0.312) for {'learning_rate': 1, 'max_depth': 7, 'n_estimators': 4}
0.704 (+/-0.26) for {'learning_rate': 1, 'max_depth': 7, 'n_estimators': 6}
0.706 (+/-0.297) for {'learning_rate': 1, 'max_depth': 7, 'n_estimators': 12}
0.72 (+/-0.299) for {'learning_rate': 10, 'max_depth': 7, 'n_estimators': 2}
0.718 (+/-0.288) for {'learning_rate': 10, 'max_depth': 7, 'n_estimators': 4}
0.701 (+/-0.294) for {'learning_rate': 10, 'max_depth': 7, 'n_estimators': 6}
0.715 (+/-0.293) for {'learning_rate': 10, 'max_depth': 7, 'n_estimators': 1
2}
```

In [57]:
```
cv.best_estimator_                                          # Stores best est
```

Out[57]:
```
GradientBoostingClassifier(max_depth=7, n_estimators=4)
```

In [58]:
```
joblib.dump(cv.best_estimator_, 'GB_model.pkl')
```

Out[58]:
```
['GB_model.pkl']
```

Analyzing the results on Test Dataset, in order to do validation.

In [59]:
```
test.head()
```

Out[59]:

	home	SqFt	Bedrooms	Bathrooms	Offers	Neighborhood	Price
0	1	1930	4	2	3	2	100000
1	2	1678	5	5	4	0	120000
2	3	2903	6	3	2	2	80000
3	4	832	3	2	1	1	76000
4	5	8732	2	1	1	2	135000

In [60]:
```
# Split the data into features (X) and target (y)
from sklearn.model_selection import train_test_split

X = test.iloc[:, :-1]
Y = test.iloc[:, -1]

x_tr, x_te, y_tr, y_te = train_test_split(X, Y, test_size=0.2, random_state=42
```

---} Types Of Models & Their Best Result.

In [61]:
```
models = {}

for mdl in ['LR', 'MLP', 'SVM', 'RF', 'GB']:
    models[mdl] = joblib.load('{}_model.pkl'.format(mdl))
```

In [62]:
```
models
```

Out[62]:
```
{'LR': LogisticRegression(C=10),
 'MLP': MLPClassifier(activation='tanh', hidden_layer_sizes=(10,)),
 'SVM': SVC(C=10, kernel='linear'),
 'RF': RandomForestClassifier(max_depth=32, n_estimators=250),
 'GB': GradientBoostingClassifier(max_depth=7, n_estimators=4)}
```

In [63]:
```
def evaluate_model(name, model, x, y):
    start = time()
    pred = model.predict(x)
    end = time()

    accuracy = round(accuracy_score(y, pred), 3)
    precision = round(precision_score(y, pred, average = 'weighted'), 3)
    recall = round(recall_score(y, pred, average = 'weighted'), 3)

    print('{} -- Accuracy: {} / precision: {} / Recall: {} / Latency: {}ms'.fo
```

In [64]:
```
for name, mdl in models.items():
    evaluate_model(name, mdl, x_train, y_train)
```

```
LR -- Accuracy: 0.038 / precision: 0.006 / Recall: 0.038 / Latency: 0.0ms
MLP -- Accuracy: 0.024 / precision: 0.002 / Recall: 0.024 / Latency: 0.0ms
SVM -- Accuracy: 0.984 / precision: 0.986 / Recall: 0.984 / Latency: 0.041744
70901489258ms
RF -- Accuracy: 1.0 / precision: 1.0 / Recall: 1.0 / Latency: 0.0362172126770
0195ms
GB -- Accuracy: 1.0 / precision: 1.0 / Recall: 1.0 / Latency: 0.0ms
```

Conclusion

Upon successful execution of the house price prediction project, we can conclude that machine learning provides a powerful tool for predicting house prices based on a variety of features. The insights gained from the feature importance can provide valuable information for home buyers, sellers, and real estate professionals, helping them make informed decisions. Furthermore, the project demonstrates the potential of machine learning in transforming the real estate industry by providing more accurate and reliable house price predictions. This could lead to more efficient markets, better investment decisions, and ultimately, a more robust real estate sector. The project also highlights the importance of continuous model monitoring and updating to ensure the model remains effective as new data comes in, reflecting the dynamic nature of the real estate market. In addition, the project underscores the importance of data quality and preprocessing in achieving accurate predictions. It shows that careful handling of missing values, outliers, and proper feature engineering can significantly improve the model's performance. The project also highlights the potential of machine learning models to uncover hidden patterns and relationships in the data that might not be apparent through traditional analysis methods. Lastly, the project emphasizes the role of continuous learning and adaptation in machine learning models to keep up with the ever-changing dynamics of real estate markets. This adaptability is crucial in ensuring the long-term effectiveness and relevance of the model in real-world applications.

Future Scope

The future directions for a house price prediction project could involve several avenues. One potential direction is the integration of more diverse data sources. While the current model may rely on traditional data sources such as property size, location, and nearby amenities, future models could incorporate more unconventional data. This could include data related to local economic indicators, demographic trends, or even social media sentiment. Incorporating these types of data could help capture the broader societal trends that influence house prices and improve the accuracy of the model.

Another promising direction is the exploration of more advanced machine learning techniques. While the current project uses XGBoost, future work could explore the use of deep learning techniques, which have shown promise in handling complex, high-dimensional data. For instance, convolutional neural networks (CNNs) could be used to analyze images of properties,

while recurrent neural networks (RNNs) or transformers could be used to analyze temporal

In [ ]: