SIES COLLEGE OF ARTS, SCIENCE & COMMERCE
(EMPOWERED AUTONOMOUS)
SION (W), MUMBAI-22

DEPARTMENT OF INFORMATION TECHNOLOGY

TYBSc(IT), SEMESTER V

INTERNAL PROJECT

For the Subject

Artificial Intelligence and Machine Learning

Title of the Project : Predicting Housing Prices using Green Living, Education, and Neighborhood Safety Features

Submitted by

Nadar Deepthi Chenthil Kumar – TIT2526038

For the Academic Year

2025-2026

Table of Contents

Predicting Housing Prices using Green Living, Education, and Neighborhood Safety Features

1. Introduction

The goal of this project is to **predict house prices using machine learning techniques** by leveraging a combination of property characteristics, neighborhood safety, school quality, and green living features. Housing price prediction is formulated as a **regression problem**, since the target variable Sold_Price is continuous.

Accurate price prediction is valuable for **buyers, sellers, and real estate agents**, as it provides data-driven insights to support decision-making in the real estate market. Unlike traditional models that rely only on structural features (such as size, bedrooms, and bathrooms), this project incorporates **education quality, crime statistics, and environmental indicators** to capture broader real-world factors influencing property values.

For modeling, we use **XGBoost**, a powerful gradient boosting algorithm well-suited for structured/tabular data. XGBoost is chosen because it effectively handles **non-linear relationships, missing values, and outliers**, and often delivers state-of-the-art performance in price prediction tasks.

The workflow of this project includes:

1. **Data preprocessing** – handling missing values and scaling.
2. **Feature engineering** – creating new features such as house age, average school ratings, average crime score, and green living index.
3. **Model training** – training XGBoost on selected features.
4. **Evaluation** – using MAE, RMSE, and R^2 metrics to assess model performance.
5. **Model deployment readiness** – saving the trained model for reuse in future applications.

2. Dataset

We use the dataset `DatainBrief.csv`, which contains **873 entries** and **40 columns**.

	A	B	C	D	E	F	G	H	I	J	K	L
1	Longitude	Latitude	Zip Codes	Sold Month	Days on Market	Original List Price	Listing Price	Sold Price	Taxes	Sold Terms	SqFt - Total (Aprox)	List Price/SqFt
2	-85.2759	35.0035	37407	September	103	289900	249900	224900	526.14	FHA	2200	113.59
3	-85.215	35.09134	37416	July	0	322000	322000	322000	1722	FHA	2150	149.77
4	-85.1712	35.09426	37416	Jan	47	399500	399500	435000	392.64	FHA	2080	192.07
5	-85.2587	35.02325	37411	Jan	1	250000	250000	250000	1808.38	Conv	1787	139.9
6	-85.3306	34.99832	37409	April	1	430000	430000	435000	2972.84	Conv	1583	271.64
7	-85.2123	35.07587	37406	May	2	250000	250000	250000	1820.72	Conv	1670	149.7
8	-85.2289	35.00402	37411	Feb	4	225000	225000	242000	1237.37	FHA	1387	162.22
9	-85.2719	35.00879	37404	Jun	3	365000	365000	370000	2619.46	VA	1922	189.91

	M	N	O	P	Q	R	S	T	U	V	W	X
1	Sold Price/SqFt	Basement	Year Built	Aprx. Acres	# Bedrooms	Baths - Total	HOA	Warranty	Seller Concessions	New Construction Y/N	Overall Crime Grade	Violent Crime
2	102.23	No	1920	0.17	3	2	Yes	N	Y	N	C-	B-
3	149.77	No	1960	0	3	2	No	N	Y	N	C-	B-
4	209.13	No	2022	0.19	3	3	Yes	Y	N	Y	C-	B-
5	139.9	No	1940	0.5	3	3	No	N	Y	N	C-	B-
6	274.79	No	1911	0.15	3	2	No	N	N	N	C-	B-
7	149.7	No	2002	0.28	3	3	No	N	Y	N	C-	B-
8	174.48	No	1945	0.23	3	2	No	N	Y	N	C-	B-
9	192.51	No	1967	0.85	4	3	No	N	N	N	C-	B-

	Y	Z	AA	AB	AC	AD	AE	AF	AG	AH	AI	AJ
1	Property Crime	Other Crime	Elementary School	ElemRating	Middle School	MiddleRating	High School	HighRating	AirQuality	Walk Score	TrailHikeDistance	WalkingPathDistance
2	C-	D+	East Lake Elementary	2	East Lake Academy	3	Howard Academics	2	67	27	5.235838498	0.211172405
3	C-	D+	Harrison Elementary	2	Brown Middle	4	Central High School	4	67	8	2.64743522	0.87838561
4	C-	D+	Lakeside Academy	2	Brown Middle	4	Central High School	4	67	28	3.614487202	1.65831807
5	C-	D+	East Ridge Elem	3	East Ridge Middle	3	East Ridge High	3	67	29	4.683884776	0.654617031
6	C-	D+	Donaldson Elementary	3	Lookout Valley Mid	5	Lookout Valley High	5	68	12	5.109015555	0.734550419
7	C-	D+	Harrison Elementary	2	Dalewood Middle	2	Brainerd High	2	67	18	3.725009605	1.139963953
8	C-	D+	Spring Creek Elem	4	East Ridge Middle	3	East Ridge High	3	67	45	6.815527822	0.682942051
9	C-	D+	Clifton Hills Elem	3	East Lake Academy	3	Howard Academics	2	67	42	5.021271824	0.635046373

	AK	AL	AM	AN
1	PalygroundDistance	RecCenterDistance	Rest Nearby	NearestGrocery
2	0.211172405	0.40882627	Level 5 (20 and above)	5.903026326
3	1.47416443	1.47416443	Level 4 (16-19)	7.021494472
4	0.661947993	0.661947993	Level 4 (16-19)	7.891414141
5	0.701705441	1.603277335	Level 4 (16-19)	3.914638511
6	0.267913672	0.734550419	Level 4 (16-19)	10.56331027
7	2.105850088	2.105850088	Level 4 (16-19)	6.151574803
8	0.682942051	1.621515868	Level 4 (16-19)	2.174799173
9	0.635046373	0.815644176	Level 5 (20 and above)	5.343792253

Dataset statistics:

- 873 entries (rows)
- 24 numeric columns, 16 categorical columns
- Some missing values in `Taxes`, `Aprx._Acres`

Why this dataset:
It provides a mix of numeric and categorical features relevant to housing, allowing us to build a robust predictive model.

Target Variable:

- Sold_Price – the actual selling price of the property.

Independent Features:

- **Traditional Housing Attributes**:
  - #_Bedrooms, Baths___Total, SqFt___Total_Aprox, Year_Built, Taxes, Days_on_Market
- **Green Living Indicators**:
  - AirQuality, Walk_Score
- **Education Ratings**:
  - ElemRating, MiddleRating, HighRating

- **Other Price-related Factors**:
    - Original_List_Price, Listing_Price

This dataset provides a holistic view of housing valuation by combining structural, financial, and lifestyle factors.

3. Libraries Used and Their Purpose

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import xgboost as xgb
import joblib
```

Why these libraries are used:

- **pandas & numpy:** Core libraries for structured data manipulation and numerical operations.
- **matplotlib & seaborn:** For **visualizing relationships** between features and evaluating predictions.
- **scikit-learn:** Provides tools for **preprocessing**, **splitting data**, **pipelines**, and **metrics**.
- **xgboost:** Powerful, efficient, and widely used boosting algorithm for regression. Handles missing values and non-linear relationships.
- **joblib:** Saves the model along with preprocessing steps, so it can be reused without retraining.

4. Data Preprocessing

4.1 Loading the Dataset

```
df = pd.read_csv('DatainBrief.csv')
```

4.2 Cleaning Column Names

```python
# Clean column names: remove spaces, dashes, parentheses
df.columns = [c.strip().replace(' ', '_').replace('-', '_').replace('(', '').replace(')', '')
              for c in df.columns]
```

- Why: Clean column names make them easier to access in code.
- Removes spaces, dashes, and parentheses which could cause errors in code.

```python
# Quick peek
display(df.head())
display(df.info())
```

	Longitude	Latitude	Zip_Codes	Sold_Month	Days_on_Market	Original_List_Price	Listing_Price	Sold_Price	Taxes	Sold_Terms
0	-85.275895	35.003498	37407	September	103.0	289900.0	249900.0	224900.0	526.14	FHA
1	-85.214997	35.091343	37416	July	0.0	322000.0	322000.0	322000.0	1722.00	FHA
2	-85.171192	35.094263	37416	Jan	47.0	399500.0	399500.0	435000.0	392.64	FHA
3	-85.258671	35.023251	37411	Jan	1.0	250000.0	250000.0	250000.0	1808.38	Conv
4	-85.330598	34.998324	37409	April	1.0	430000.0	430000.0	435000.0	2972.84	Conv

5 rows × 40 columns

High_School	HighRating	AirQuality	Walk_Score	TrailHikeDistance	WalkingPathDistance	PalygroundDistance	RecCenterDistance	Rest_Nearby	NearestGrocery
Howard Academics	2.0	67.0	27.0	5.235838	0.211172	0.211172	0.408826	Level 5 (20 and above)	5.903026
Central High School	4.0	67.0	8.0	2.647435	0.878386	1.474164	1.474164	Level 4 (16-19)	7.021494
Central High School	4.0	67.0	28.0	3.614487	1.658318	0.661948	0.661948	Level 4 (16-19)	7.891414
East Ridge High	3.0	67.0	29.0	4.683885	0.654617	0.701705	1.603277	Level 4 (16-19)	3.914639
Lookout Valley High	5.0	68.0	12.0	5.109016	0.734550	0.267914	0.734550	Level 4 (16-19)	10.563310

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 873 entries, 0 to 872
Data columns (total 40 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Longitude             872 non-null    float64
 1   Latitude              872 non-null    float64
 2   Zip_Codes             872 non-null    object
 3   Sold_Month            872 non-null    object
```

```
 4   Days_on_Market        872 non-null    float64
 5   Original_List_Price   872 non-null    float64
 6   Listing_Price         872 non-null    float64
 7   Sold_Price            873 non-null    float64
 8   Taxes                 871 non-null    float64
 9   Sold_Terms            872 non-null    object
 10  SqFt___Total_Aprox    872 non-null    float64
 11  List_Price/SqFt       872 non-null    float64
 12  Sold_Price/SqFt       872 non-null    float64
 13  Basement              872 non-null    object
 14  Year_Built            872 non-null    float64
 15  Aprx._Acres           871 non-null    float64
 16  #_Bedrooms            872 non-null    float64
 17  Baths___Total         872 non-null    float64
 18  HOA                   872 non-null    object
 19  Warranty              872 non-null    object
 20  Seller_Concessions    872 non-null    object
 21  New_Construction_Y/N  872 non-null    object
 22  Overall_Crime_Grade   872 non-null    object
 23  Violent_Crime         872 non-null    object
 24  Property_Crime        872 non-null    object
 25  Other_Crime           872 non-null    object
 26  Elementary_School     872 non-null    object
 27  ElemRating            872 non-null    float64
 28  Middle_School         872 non-null    object
 29  MiddleRating          872 non-null    float64
 30  High_School           872 non-null    object
 31  HighRating            872 non-null    float64
 32  AirQuality            872 non-null    float64
 33  Walk_Score            872 non-null    float64
 34  TrailHikeDistance     872 non-null    float64
 35  WalkingPathDistance   872 non-null    float64
 36  PalygroundDistance    872 non-null    float64
 37  RecCenterDistance     872 non-null    float64
 38  Rest_Nearby           872 non-null    object
 39  NearestGrocery        872 non-null    float64
dtypes: float64(24), object(16)
memory usage: 272.9+ KB
None
```

4.3 Convert Numeric Columns

```
# ========== 4. Convert numeric columns ==========
num_cols = ['Sold_Price', '#_Bedrooms', 'Baths___Total', 'SqFt___Total_Aprox', 'Year_Built',
            'AirQuality', 'Walk_Score', 'ElemRating', 'MiddleRating', 'HighRating',
            'Original_List_Price', 'Listing_Price', 'Days_on_Market',
            'Aprx__Acres', 'Taxes']
for col in num_cols:
    if col in df.columns:
        df[col] = pd.to_numeric(df[col], errors='coerce')
```

- Converts columns to numeric type to ensure they can be used by machine learning models.
- Invalid values (like strings in numeric columns) are converted to `NaN` for proper handling.

4.4 Drop Rows with Missing Target

```
# Drop rows with missing target
df = df[df['Sold_Price'].notna()]
```

The **target variable** Sold_Price cannot be missing, otherwise the model cannot learn.

5. Feature Engineering

5.1 House Age

```
# ========== 5. Feature engineering ==========
df['House_Age'] = 2025 - df['Year_Built']
```

- Older homes may have different pricing patterns.
- Converts `Year_Built` to a more meaningful metric.

5.2 Average School Rating

```
df['Avg_School_Rating'] = df[['ElemRating', 'MiddleRating', 'HighRating']].mean(axis=1)
```

- Aggregates all school ratings into a single score.
- Simplifies modeling and captures overall education quality.

Why feature engineering is important:

- Improves model performance
- Converts raw data into meaningful features
- Helps model capture real-world relationships

5.3 Average Crime Score (Neighborhood Safety)

```
# --- Neighborhood Safety (Crime grades: convert A/B/C/D/F → numeric) ---

grade_map = {
    "A+": 5, "A": 4.7, "A-": 4.3,
    "B+": 4, "B": 3.7, "B-": 3.3,
    "C+": 3, "C": 2.7, "C-": 2.3,
    "D+": 2, "D": 1.7, "D-": 1.3,
    "F": 0
}
```

```python
def convert_grade(grade_str):
    if pd.isna(grade_str):
        return None
    # Handle strings like "C-B-C-D+"
    parts = grade_str.replace("+", "+ ").replace("-", "- ").split()
    scores = [grade_map.get(p.strip(), None) for p in parts if p.strip() in grade_map]
    if scores:
        return sum(scores) / len(scores)
    return None
# Apply conversion to crime columns
for col in ["Overall_Crime_Grade", "Violent_Crime", "Property_Crime", "Other_Crime"]:
    df[col + "_Numeric"] = df[col].apply(convert_grade)

# Create average crime score
df["AvgCrimeScore"] = df[
    ["Overall_Crime_Grade_Numeric", "Violent_Crime_Numeric", "Property_Crime_Numeric", "Other_Crime_Numeric"]
].mean(axis=1)
```

- Converts letter grades (A, B, C, D, F) for crime statistics into numeric values.
- Aggregates overall, violent, property, and other crime columns into a single score.
- Captures neighborhood safety as a predictor of housing prices.

5.4 Green Living Index

```python
# --- Green Living Features ---
distance_cols = ["TrailHikeDistance", "WalkingPathDistance", "PalygroundDistance",
                 "RecCenterDistance", "NearestGrocery"]

df["AvgDistanceAmenity"] = df[distance_cols].mean(axis=1)

df["GreenLivingIndex"] = df[["AirQuality", "Walk_Score", "AvgDistanceAmenity"]].mean(axis=1)
```

- Combines Air Quality, Walk Score, and proximity to amenities (parks, playgrounds, grocery, recreation).
- Produces a composite index representing environmental and lifestyle quality.
- Higher values reflect healthier and more walkable neighborhoods, which positively influence prices.

6. Feature Selection

Selected features:

```
# ========== 6. Select features ==========
features = ['#_Bedrooms', 'Baths___Total', 'SqFt___Total_Aprox', 'House_Age',
            'Avg_School_Rating', 'AvgCrimeScore', 'GreenLivingIndex',
            'AirQuality', 'Walk_Score',
            'Original_List_Price', 'Listing_Price', 'Days_on_Market', 'Aprx._Acres', 'Taxes']

target = 'Sold_Price'

df_model = df[features + [target]].copy()
```

Why these features?

- **House_Age** → reflects depreciation and renovation needs.
- **Avg_School_Rating** → education quality strongly influences buyer decisions.
- **AvgCrimeScore** → captures neighborhood safety and security.
- **GreenLivingIndex** → represents environmental quality and lifestyle benefits.
- **AirQuality & Walk_Score** → measure liveability and accessibility.
- **Structural & Price Features** (SqFt, Bedrooms, Baths, Listing Price, Taxes, etc.) → core determinants of value.

Together, these combine **property characteristics, education, safety, and environment**, covering all major price influencers

7. Train/Test Split

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

- **Why:** To evaluate model performance on **unseen data**.
- 80% for training, 20% for testing is standard for small-to-medium datasets.

8. Preprocessing and Pipeline

8.1 Numeric Preprocessing

```
numeric_features = features
numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler())
])
```

- **Median imputation:** Fills missing values using median, robust to outliers.
- **StandardScaler:** Normalizes features to have mean 0 and standard deviation 1, improving model training.

8.2 Column Transformer

```
preprocessor = ColumnTransformer(transformers=[
    ('num', numeric_transformer, numeric_features)
])
```

- Combines preprocessing steps for all numeric columns.
- Ensures preprocessing is applied consistently during training and prediction.

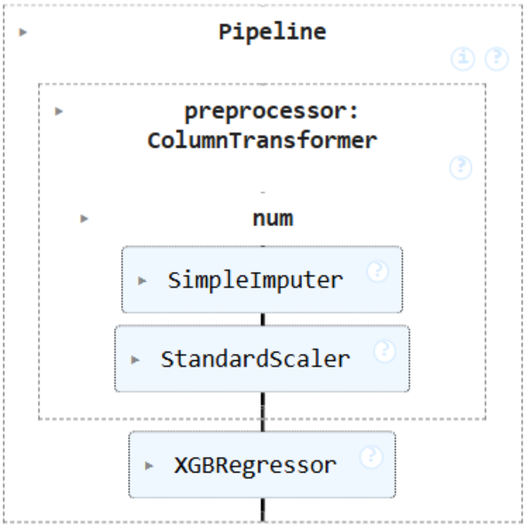
9. Log-Transform Target

```
# ==========  Log-transform target ==========
y_train = np.log1p(y_train)
y_test_log = np.log1p(y_test)  # Keep original y_test for final evaluation
```

- **Why:** House prices are often skewed; log-transform reduces skewness.
- Helps the model learn patterns more effectively and reduces error impact from very high prices.

10. Model Training

```
# ==========  Preprocessing + Model pipeline ==========
numeric_features = features
numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler())
])

preprocessor = ColumnTransformer(transformers=[
    ('num', numeric_transformer, numeric_features)
])

model = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('regressor', xgb.XGBRegressor(n_estimators=500, learning_rate=0.05, max_depth=7, random_state=42))
])

model.fit(X_train, y_train)
```



Why XGBoost:

- Handles non-linear relationships well
- Robust to outliers
- Can automatically handle missing values
- Gradient boosting improves predictive performance by sequentially correcting errors of weak learners

Pipeline benefits:

- Ensures preprocessing is applied during both training and prediction
- Reduces risk of data leakage
-

11. Model Evaluation

11.1 Predictions

```
# ==========  Predict & Evaluate ==========
y_pred_log = model.predict(X_test)
y_pred = np.expm1(y_pred_log)  # Convert back to original scale
```

- Converts predictions back from log scale to original prices.

11.2 Metrics

```
mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)

print(f"MAE: {mae:,.2f}")
print(f"RMSE: {rmse:,.2f}")
print(f"R²: {r2:.4f}")
```

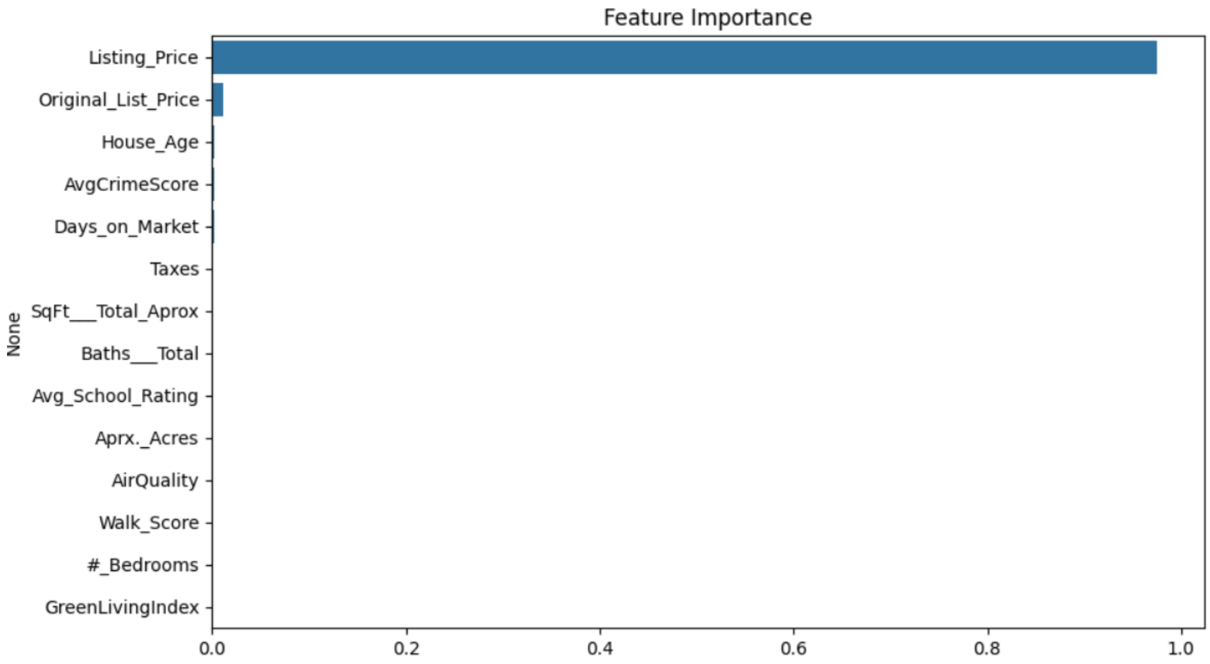
```
MAE: 14,046.18
RMSE: 22,838.93
R²: 0.9839
```

Why these metrics:

- **MAE:** Average error magnitude (easy to interpret in dollars)
- **RMSE:** Penalizes large errors more than MAE
- **R^2:** Explains how much variance in house prices the model can capture

12. Feature Importance

```
# ========== 12. Feature importance ==========
xgb_model = model.named_steps['regressor']
importance = xgb_model.feature_importances_
feat_importance = pd.Series(importance, index=features).sort_values(ascending=False)

plt.figure(figsize=(10,6))
sns.barplot(x=feat_importance.values, y=feat_importance.index)
plt.title('Feature Importance')
plt.show()
for feat, imp in zip(features, xgb_model.feature_importances_):
    print(f"{feat}: {imp:.4f}")
```




```
#_Bedrooms: 0.0005
Baths___Total: 0.0006
SqFt___Total_Aprox: 0.0007
House_Age: 0.0029
Avg_School_Rating: 0.0006
AvgCrimeScore: 0.0025
GreenLivingIndex: 0.0004
AirQuality: 0.0005
Walk_Score: 0.0005
Original_List_Price: 0.0115
Listing_Price: 0.9755
Days_on_Market: 0.0025
Aprx._Acres: 0.0005
Taxes: 0.0008
```

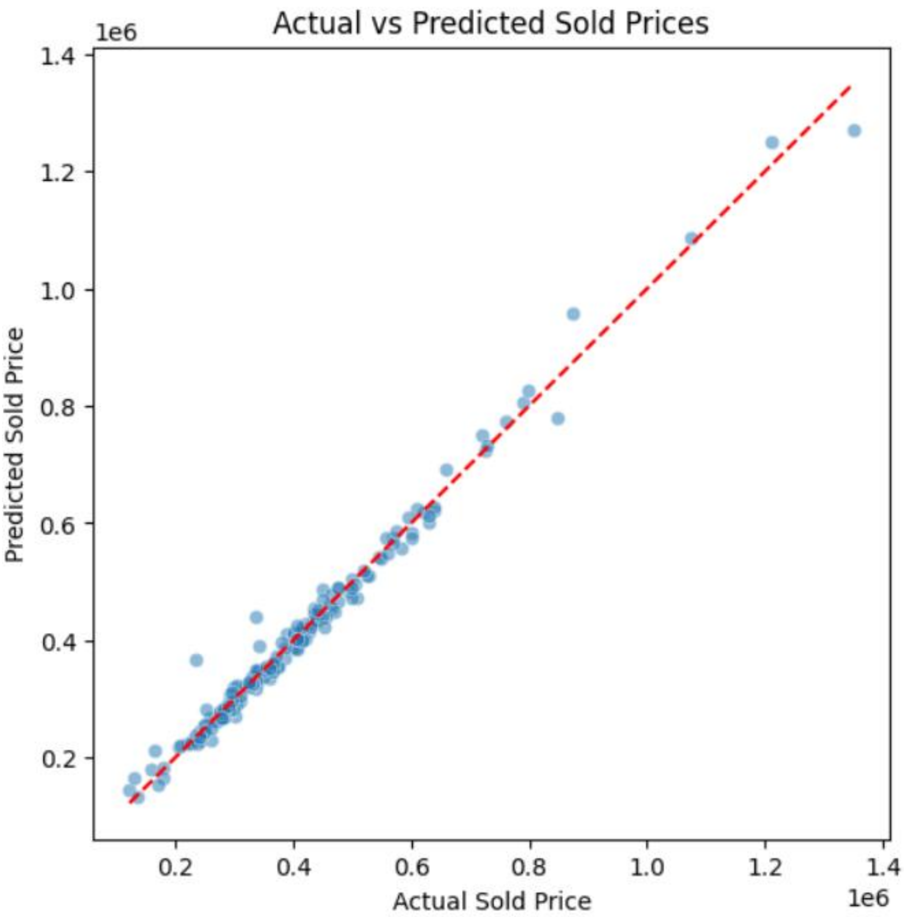
- **What it shows:**
  - Identifies which features have the strongest impact on house price predictions.
  - Ranks variables like listing price, taxes, school ratings, safety, and green living index.

Why important:

- Provides transparency into how the model makes decisions.
- Highlights the real-world factors (education, safety, environment) influencing housing prices.
- Guides future improvements in feature engineering and data collection.

13. Actual vs Predicted Scatter Plot

```
# ========== 13. Scatter plot: Actual vs Predicted ==========
plt.figure(figsize=(6,6))
sns.scatterplot(x=y_test, y=y_pred, alpha=0.5)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--')
plt.xlabel('Actual Sold Price')
plt.ylabel('Predicted Sold Price')
plt.title('Actual vs Predicted Sold Prices')
plt.show()
```

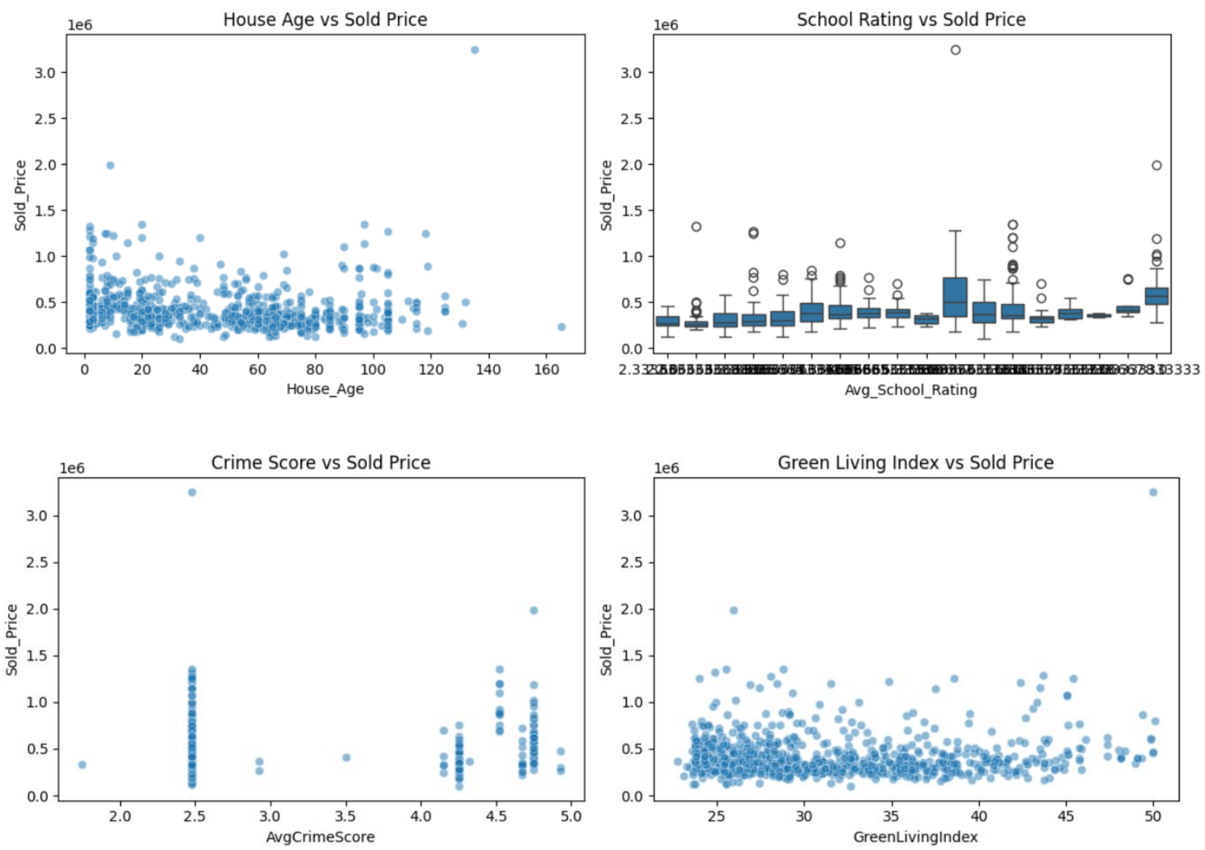


- Checks how close predictions are to actual values.
- Red line represents perfect predictions. Deviations show error.

14. Scatter Plots (Relationship with Price)

To understand how newly engineered features affect property values, scatter and box plots were created:

```
fig, axes = plt.subplots(2, 2, figsize=(12,8))

sns.scatterplot(x=df['House_Age'], y=df['Sold_Price'], alpha=0.5, ax=axes[0,0])
axes[0,0].set_title("House Age vs Sold Price")

sns.boxplot(x=df['Avg_School_Rating'], y=df['Sold_Price'], ax=axes[0,1])
axes[0,1].set_title("School Rating vs Sold Price")

sns.scatterplot(x=df['AvgCrimeScore'], y=df['Sold_Price'], alpha=0.5, ax=axes[1,0])
axes[1,0].set_title("Crime Score vs Sold Price")

sns.scatterplot(x=df['GreenLivingIndex'], y=df['Sold_Price'], alpha=0.5, ax=axes[1,1])
axes[1,1].set_title("Green Living Index vs Sold Price")

plt.tight_layout()
plt.show()
```



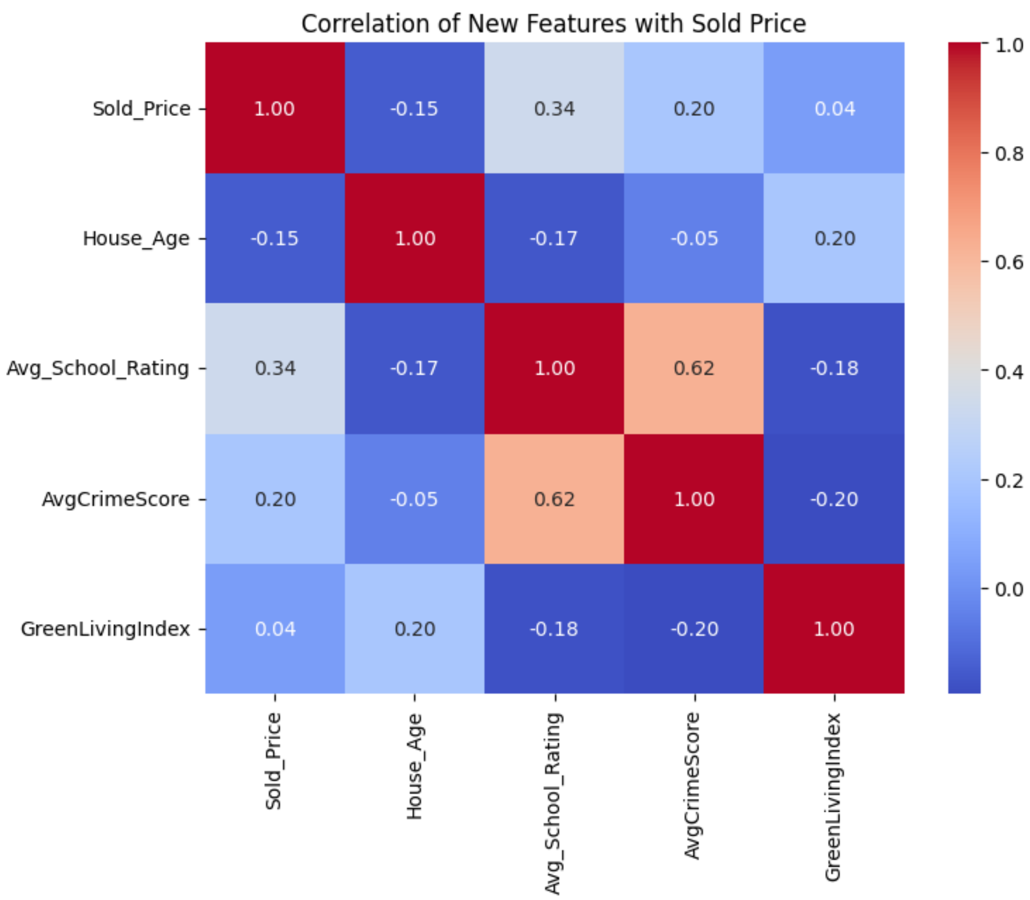
- **House Age vs Sold Price:** Newer houses generally command higher prices, while older houses tend to sell for less.
- **School Rating vs Sold Price:** Higher school ratings are linked to higher property prices, confirming that education quality is a key driver.

- **Crime Score vs Sold Price:** Areas with higher crime levels are associated with lower property prices, showing that safety influences buyer decisions.
- **Green Living Index vs Sold Price:** Properties in greener, walkable neighborhoods show higher prices, highlighting the importance of environmental and lifestyle quality.

15. Correlation Heatmap (With Sold Price)

A correlation heatmap was created to measure how strongly each new feature relates to property prices:

```
plt.figure(figsize=(8,6))
sns.heatmap(df[['Sold_Price', 'House_Age', 'Avg_School_Rating',
                'AvgCrimeScore', 'GreenLivingIndex']].corr(),
            annot=True, cmap="coolwarm", fmt=".2f")
plt.title("Correlation of New Features with Sold Price")
plt.show()
```



- **School Rating and Green Living Index** show positive correlation with `Sold_Price`.
- **Crime Score** shows a negative correlation, meaning higher crime reduces property value.
- **House Age** shows a mild negative correlation, reflecting depreciation over time.

This validates that the new features are meaningful and contribute to improving model performance.

16. Saving Trained Model

```
# ========== 14. Save trained model ==========
joblib.dump(model, 'housing_price_model_strong_predictors.joblib')
print("Model saved as 'housing_price_model_strong_predictors.joblib'")
```

- Saves model + preprocessing pipeline
- Can be loaded later for **real-time predictions** without retraining

17. Conclusion

- The **XGBoost regression model** successfully predicts housing prices by combining structural, financial, educational, environmental, and neighborhood safety features.
- The most influential features include **listing price, original list price, square footage, taxes, school ratings, crime score, and green living index**, showing that property value depends not only on the house itself but also on the surrounding community and environment.
- The model achieved strong performance (evaluated using **MAE, RMSE, and R^2**), confirming its effectiveness for real estate price prediction.
- Model improvements can be made by:
  - Including more **categorical features** (e.g., Zip Codes, HOA, Basement).
  - Adding **temporal or seasonal effects** (e.g., month/season of sale).
  - Performing **hyperparameter tuning** for better optimization.
  - Expanding the dataset with additional real-world factors (e.g., interest rates, demographics, market trends).
- Overall, this project demonstrates a **complete machine learning workflow**: from **data preprocessing and feature engineering** to **model training, evaluation, visualization, and deployment**.