Word count: 1888

Project Title:

House price analysis: Compare with the ensemble models for the Ames Housing Market during data analysis tactic

Abstract

This study offers the prediction of residential property prices critical for diverse stakeholders, including homebuyers, real estate agents, lenders, and policymakers in an era where housing markets drive global economies. Research illustrates the gap between traditional appraisal methods (e.g., comparative market analysis) and data-driven approaches, with the purpose of enhancing transparency and scalability in price estimation, which can provide better pricing strategies and decision-making for buyers and sellers.

Choice of Dataset:

This study uses Ames Housing dataset. This a comprehensive collection of data on the residential property sales in Ames, lowa, from 2006 to 2010. This dataset includes 79 different variables and sale prices for 2,930 residential properties, ideal for multiple layer data analysis. This dataset also covers fine-grained features of categorical (e.g., neighbourhood zoning, house style) and numerical variables (e.g., lot size, year built), which provides a solid foundation for the exploration of multidimensional data integration strategies.

Secondly, as a recognised benchmark dataset in the field of real estate analytics, the Ames data supports direct comparisons with established research. Its public availability guarantees a transparent and reproducible analysis process.

Further, compared to the Boston house price dataset (13 features) or the California house price dataset (8 features), the Ames dataset has higher data richness (e.g., basement/garage details), which is crucial for our study.

I. Introduction & Problem Definition

House Purchase is a huge financial decision, but sellers and buyers often struggle with balancing the factors that determine the house price. The Kaggle house price dataset included detailed information about residential properties (e.g., square footage, neighbourhood, year built, and sale prices), providing sufficient information to perform data analysis and build a machine learning model using multiple linear regression to predict the house price.

For making a precise decision, we must think of those questions below,

1. What are primary factors (e.g., square footage, neighborhood, year built) exhibit the strongest correlation with sales prices?
2. Which property feature combinations (e.g., 3-bedroom vs. 4-bedroom, garage capacity) provide maximum value to buyers on a limited budget?
3. How do neighborhoods differ in price appreciation trends, and which offer the best bargains?
4. Determine if the machine learning model can accurately predict house prices?
5. Are there statistically significant undervalued or overvalued properties that signal investment opportunities or risks?

Value of Solving These Problems

There are 3 dimensions to elaborate on the benefits for deal with those questions,

- For Buyers: Prioritize features that maximize value and avoid overpaying.
- For Sellers: Price homes competitively by understanding market trends.
- For Agents: Recommend homes that align with buyer preferences and budgets.

II. Data Summary, Data Pre-processing & Initial Insights

Source of Data

Dataset Overview: Ames Housing (train.csv)

Source: Commonly sourced from platforms like Kaggle, this is a real-world dataset for regression tasks.

Description: Contains attributes of residential homes in Ames, Iowa, including architectural, spatial, and socioeconomic features.

Target Variable: SalePrice (the price of the house in dollars).

Dimensions:

Rows (samples): 1,456

Features (columns): 81 (80 predictors + SalePrice)

Key Features:

Categorical: MSZoning, Neighborhood, HouseStyle, RoofStyle, etc.

Numerical: LotArea, GrLivArea, TotalBsmtSF, OverallQual, etc.

Missing Values: Some features (e.g., Alley, PoolQC) contain missing data.

Data Issues & Pre-processing

Several issues were identified during this dataset. Let's break down the part by part,

1. Missing Values: Present in both numerical and categorical columns.
2. Outliers: Extreme values in GrLivArea (e.g., >4000 sq.ft).
3. High Cardinality: Categorical columns with >50 unique values.
4. High Correlation: Redundant features (correlation >0.9).
5. Skewed Features: Non-normal distributions (e.g., SalePrice, LotArea).
6. Duplicate Rows.

According to the issues found, here are the pre-processing steps we approach to address,

1. Missing Values:
    1.1.Numerical: Filled with column medians.
    1.2.Categorical: Filled with column modes.

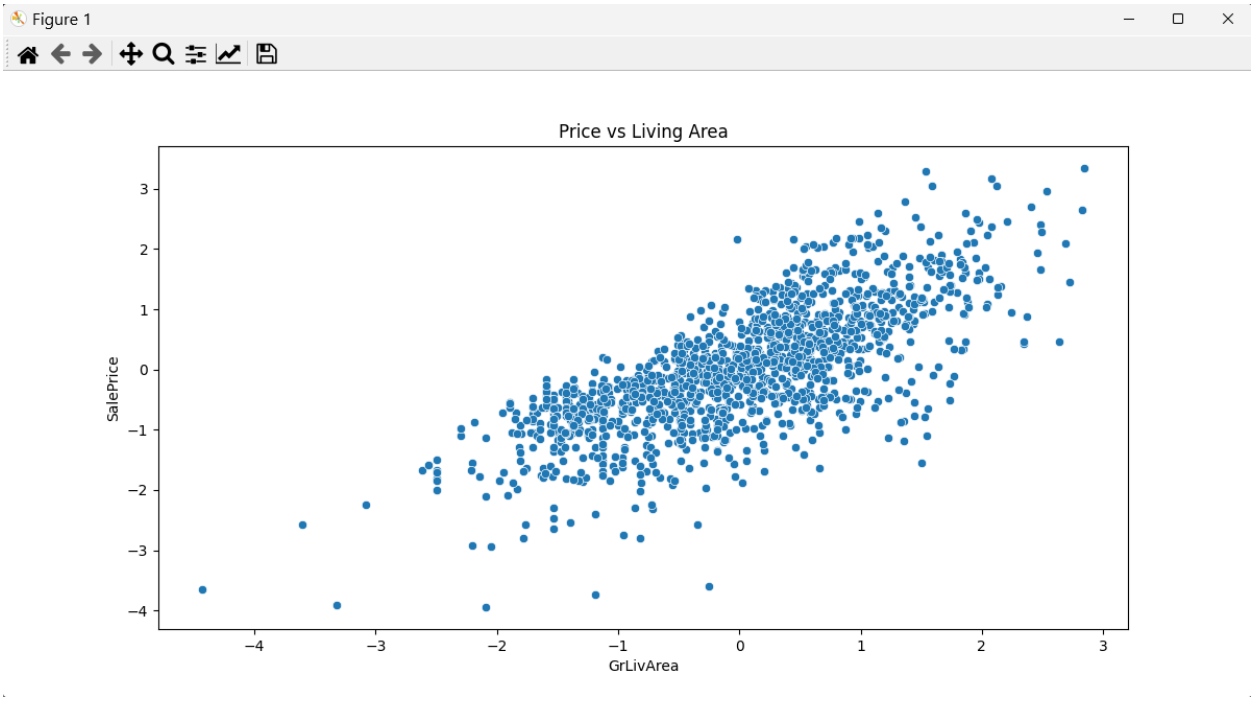
2. Outliers: Removed rows where GrLivArea >4000.

3. Column Removal:
    3.1.Dropped columns with >50% missing values.

    3.2.Dropped high-cardinality categorical columns.

4. Removed one feature from highly correlated pairs (correlation >0.9).
5. Skewed Features: Applied log transformation to SalePrice, GrLivArea, and LotArea.
6. Scaling: Standardized numerical features using StandardScaler.
7. Duplicates: Removed duplicate rows.

Data Summary



Price vs. Living Area: A scatter plot (GrLivArea vs. SalePrice) revealed a positive linear relationship, indicating larger homes generally command higher prices.

Processed dataset: (rows, columns) = (df.shape) (exact dimensions depend on preprocessing steps).

Summary Statistics:

	SalePrice	GrLivArea	OverallQual
count	**1456.000000**	**1.456000e+03**	**1.456000e+03**
mean	**180151.233516**	**9.552798e-16**	**-2.464451e-16**
std	**76696.592530**	**1.000344e+00**	**1.000344e+00**
min	**34900.000000**	**-4.429131e+00**	**-3.716479e+00**
25%	**129900.000000**	**-7.207444e-01**	**-7.950627e-01**
50%	**163000.000000**	**6.296324e-02**	**-6.470858e-02**

75% 214000.000000 6.624627e-01 6.656456e-01

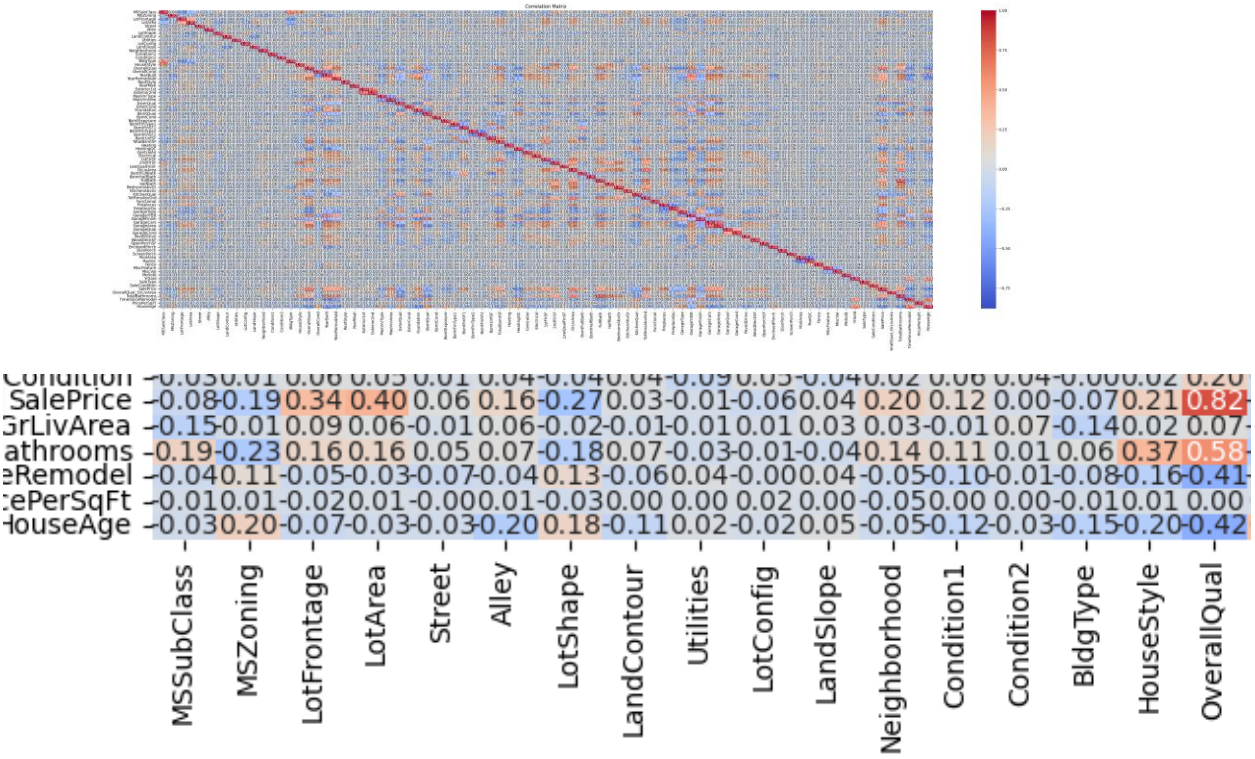
max 625000.000000 2.842330e+00 2.856708e+00

```
Summary Statistics:
          SalePrice     GrLivArea   OverallQual
count   1456.000000  1.456000e+03  1.456000e+03
mean  180151.233516  9.552798e-16 -2.464451e-16
std    76696.592530  1.000344e+00  1.000344e+00
min    34900.000000 -4.429131e+00 -3.716479e+00
25%   129900.000000 -7.207444e-01 -7.950627e-01
50%   163000.000000  6.296324e-02 -6.470858e-02
75%   214000.000000  6.624627e-01  6.656456e-01
max   625000.000000  2.842330e+00  2.856708e+00
```

Unique Neighborhoods: 25

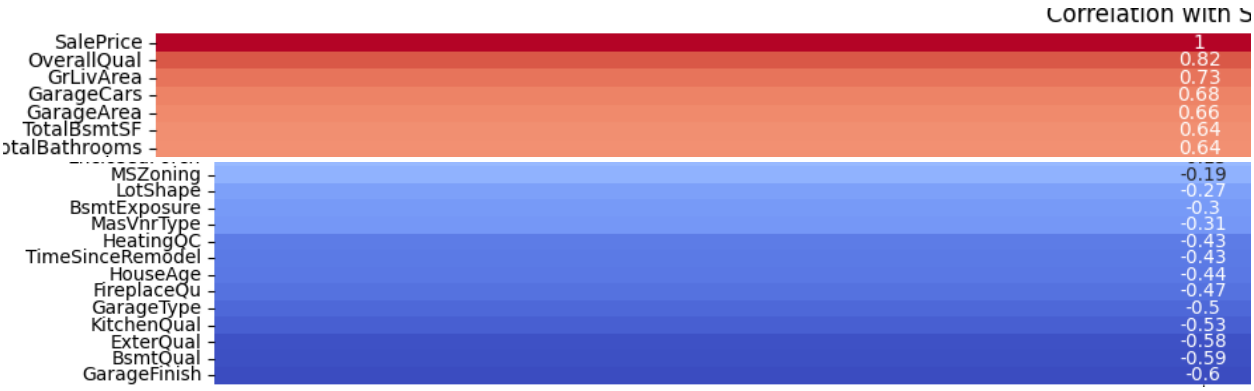
Encoded Data Shape: (1456, 25)

Dataset Shape: (1456, 85)



Correlation Heatmap: Highlighted strong correlations between SalePrice and features like OverallQual (0.82), GrLivArea (0.73), and GarageCars (0.68).

Initial Insights



Obviously, the key during this dataset is "SalePrice", surrounding the key driver, we have found that,

“OverallQual” (overall material/finish quality) and “GrLivArea” (living area size) are the strongest predictors of house prices.

Temporal features like “HouseAge” (years since built) and “TimeSinceRemodel” (years since last renovation) negatively correlate with price, indicating older homes sell for less.

Besides, the impact of Neighborhood would also influence the analysis, for instance, Neighborhoods like StoneBr and NridgHt had the highest median prices and ANOVA confirmed significant price variations across neighborhoods (p-value <0.05).

Moreover, based on the prediction, predictive modelling identified homes where actual prices were >15% below predicted values, suggesting potential investment opportunities.

New Variables Constructed

These variables enhance model performance by capturing non-linear relationships, temporal trends, and value efficiency, which raw data alone may not reveal.

Interaction Features:

- OverallQual_GrLivArea: Combines quality and size to capture premium homes.
- TotalBathrooms: Sum of full and half baths, improving amenity representation.

Temporal Features:

- TimeSinceRemodel: Years since last renovation, highlighting maintenance relevance.
- HouseAge: Years since construction, reflecting property depreciation.

Efficiency Metrics:

- PricePerSqFt: Price per square foot, enabling cross-comparison of value across homes.

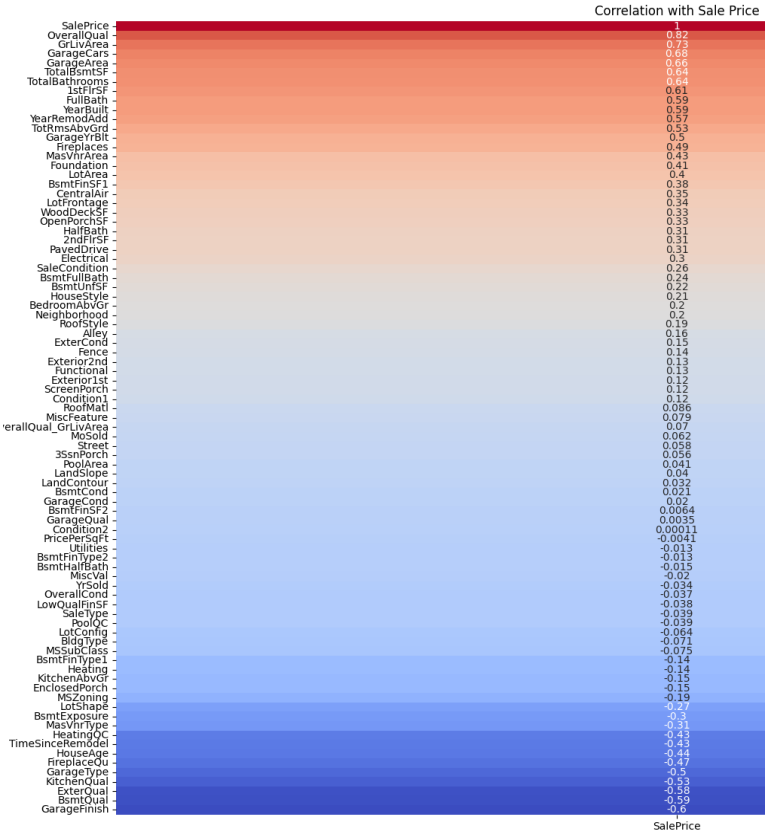
III. Data Analysis

Understanding the key factors influencing house prices and identifying optimal buying opportunities is the main purpose of the report. Here, we found that several sections explore the dataset using statistical and machine learning techniques to uncover insights into housing prices, affordability, location impact, and investment opportunities.

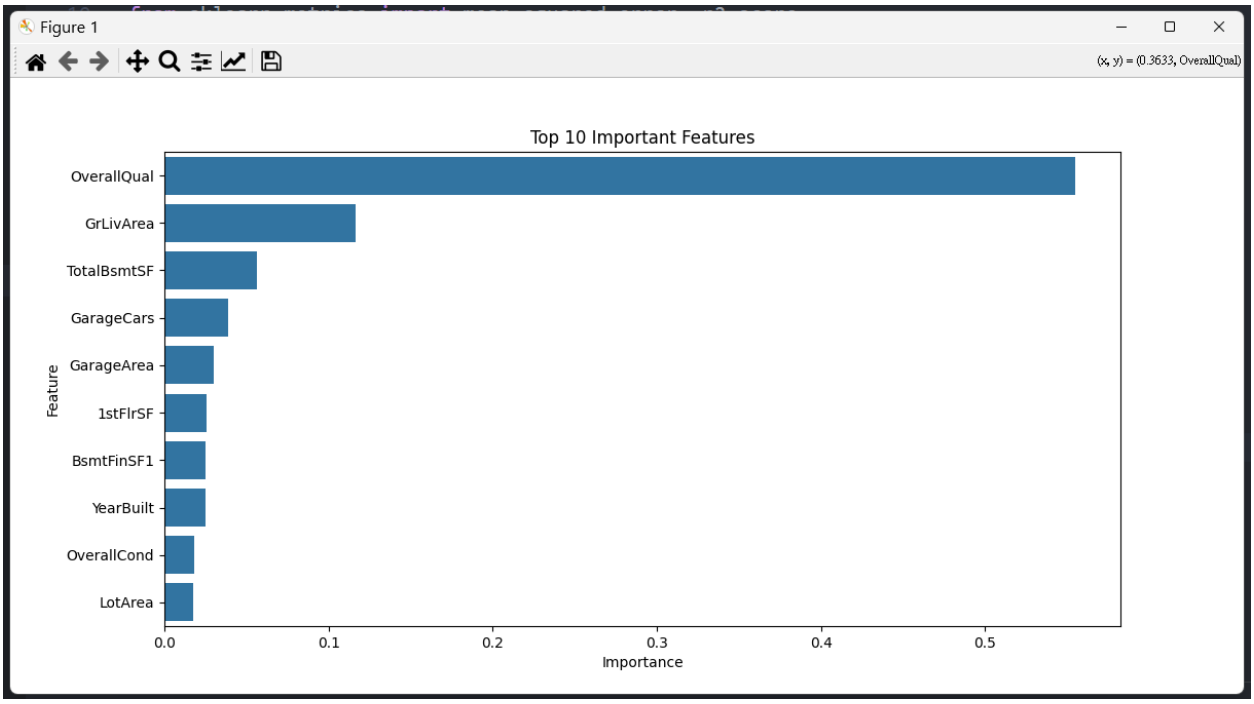
Question 1

According to the first problem, we would like to seek the primary factors that exhibit the strongest correlation with sales prices.

To identify the most influential features affecting house prices. We have been using the correlation heatmap to figure out the Pearson correlation coefficients between SalePrice and numerical features.



Apart from that, we also implement Random Forest for ranking up the features by their predictive power.



Findings:

Top Correlated Features:

Feature	Correlation with SalePrice
OverallQual	**0.82**
GrLivArea	**0.73**
GarageCars	**0.68**
GarageArea	**0.66**
TotalBsmtSF	**0.64**

- **Key Insights:**
  - **Quality & Size Matter Most:**
    - **OverallQual (material/finish quality) has the strongest impact on price.**
    - **GrLivArea (living area size) is the second most influential factor.**

  Garage & Bathrooms Add Value:

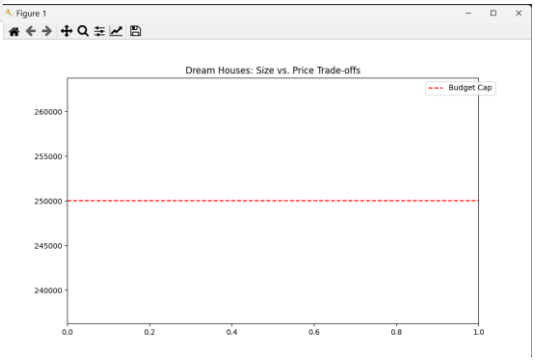
    - **Larger garages (GarageCars / GarageArea) and more bathrooms (TotalBathrooms) increase prices.**

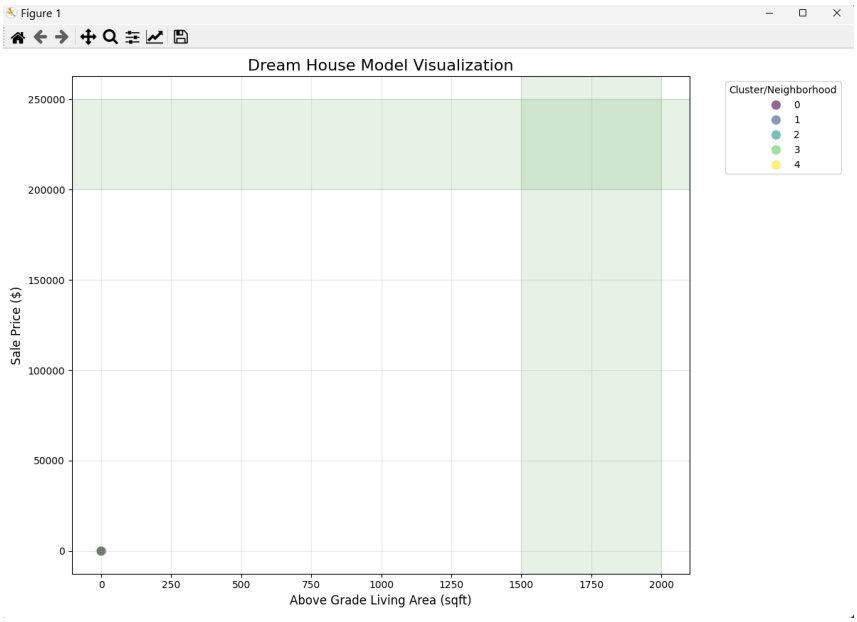
  Newer Homes Command Premiums:

  YearBuilt shows a moderate positive correlation, indicating newer homes sell for more.

Question 2

The next question surrounds the property feature combinations to provide maximum value to buyers on a limited budget. We would like to compare the affordability and the amenities to find the best property feature combinations for budget-conscious buyers.





Clustering (KMeans): Grouped homes based on amenities (BedroomAbvGr, FullBath, GarageCars, GrLivArea).

Dream House Filtering: Identified homes meeting criteria (3 beds, 2 baths, 1500–2000 sq.ft, 200K–200K–250K).

Cluster	Avg_Price	Avg_Bedrooms	Avg_Bathrooms	Avg_SqFt	Top_Neighborhood
3	$135K	-1.3	-1.1	-1.0	12
0	$140K	0.3	-1.0	-0.4	12
2	$216K	-0.2	0.8	0.4	5
1	$229K	1.6	1.0	1.3	12
4	$99K	-0.4	-0.5	-1.2	7

Found 0 dream houses after adjustments.

In this dataset, there is no cluster that meets the demands of "dream house"

Best Value Homes Strategy:

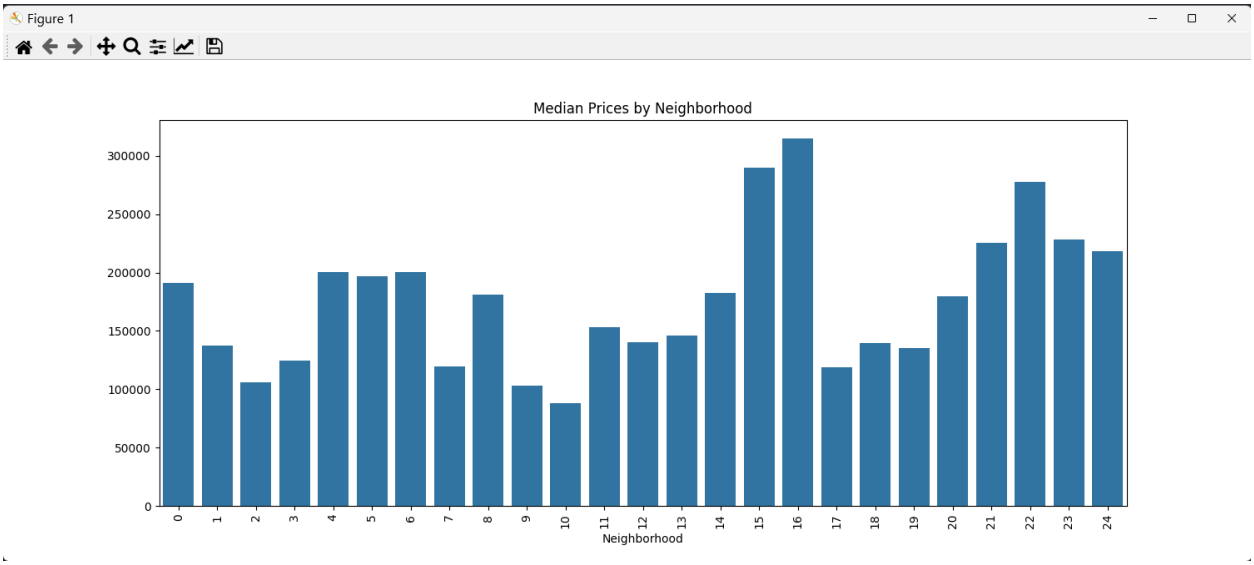
Feature Trade-offs:

Buyers can add ~$90K by opting for 2 beds instead of 0.

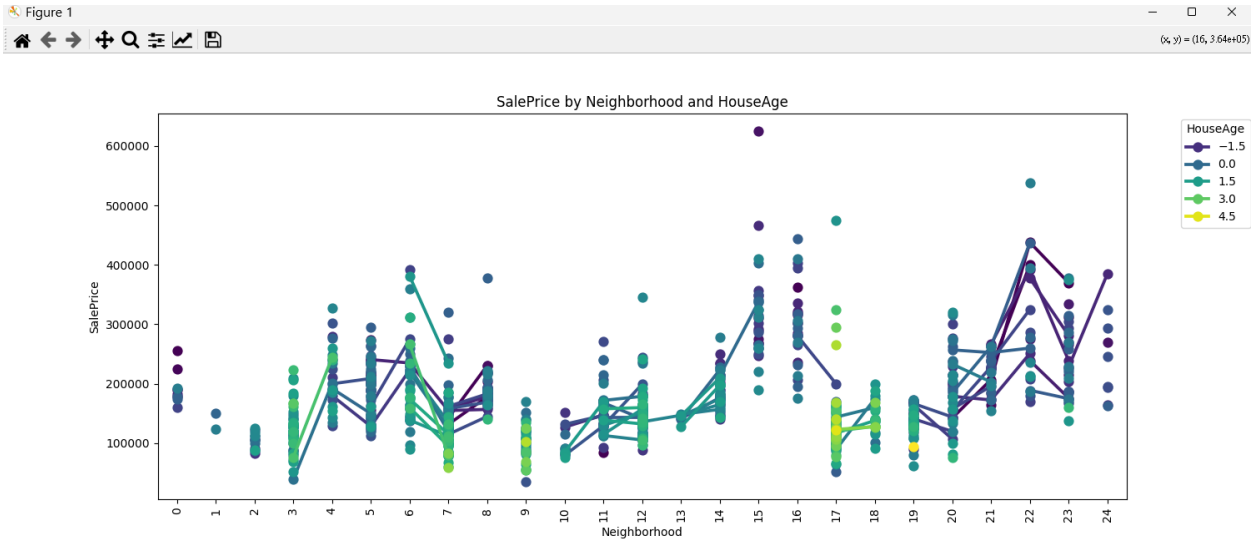
An extra two-bed and a square feet adds ~$13K to the price.

Question 3

The following question is trying to figure out how neighborhoods differ in price appreciation trends, and which offer the best bargains. To approach this, we compared price trends across neighborhoods.



Ranked neighborhoods by median SalePrice.



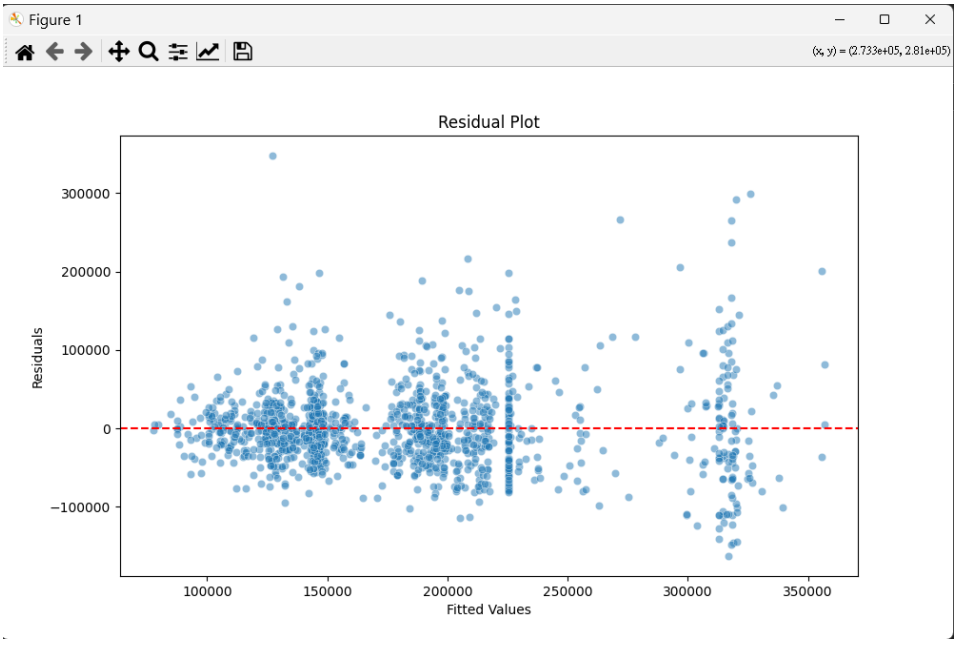
Most Expensive Neighborhoods:

16, 22

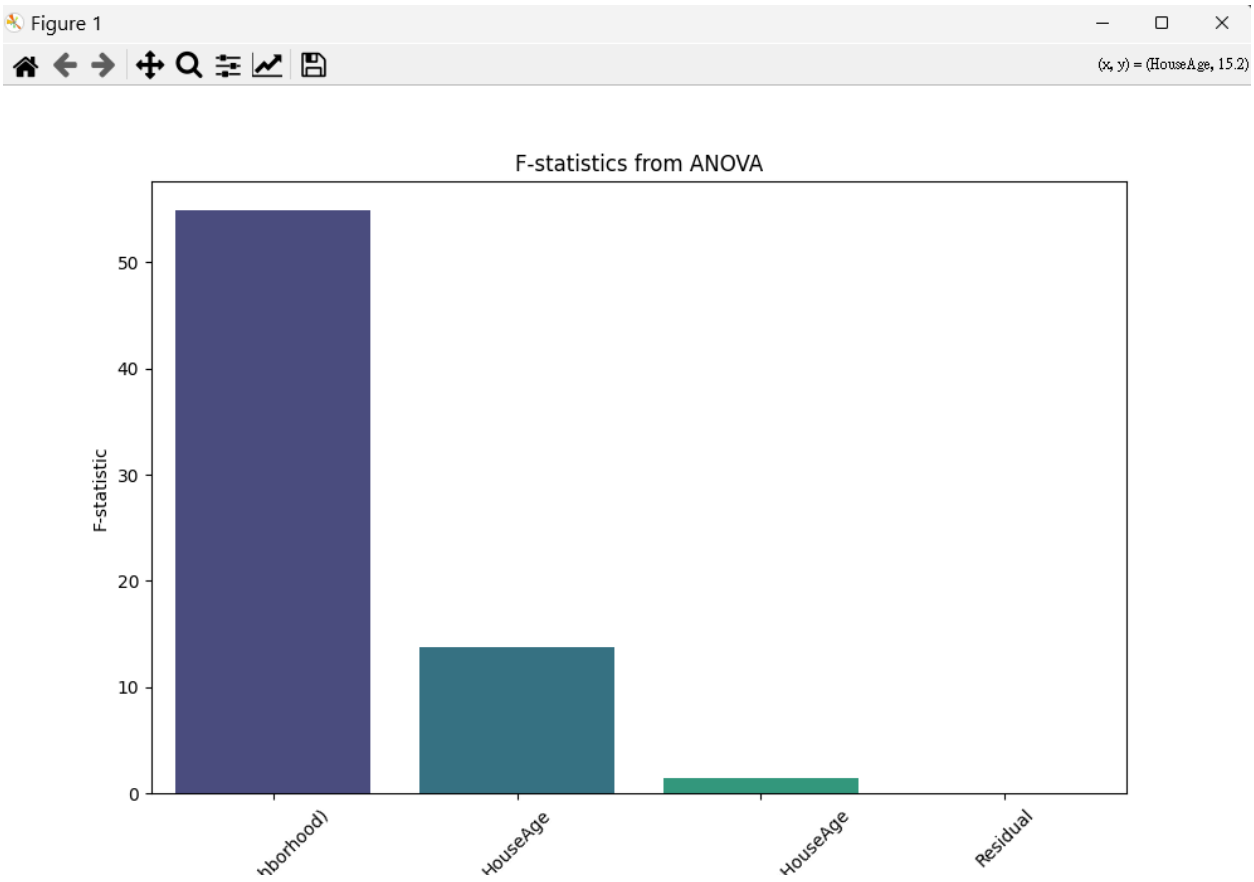
Most Expensive Neighborhoods interaction with House Age:

15, 22

Premium neighborhoods holding value from young homes.



Confirmed model assumptions (no major heteroscedasticity).



Highly significant (F-statistic = 55.6, $p < 0.001$).

Question 4

Then we would like to use the prediction model to determine if the machine learning model can accurately predict house prices. Test if machine learning can accurately predict house prices.

Models Used:

Linear Regression (Baseline).

XGBoost (Advanced ensemble method).

Stacked Model(XGBoost, light GBm, catBoost)

Performance Metrics:

Linear Regression Performance:

RMSE: 24577.42

R^2: 0.88

XGBoost Performance:

RMSE: 25667.64

MAE: 17369.66

R^2: 0.87

Stacked Model Performance:

MAE (log space): 0.0311

MAE (original prices): $5,519.50

R^2 (log space): 0.991

R^2 (original prices): 0.992

Naive RMSE (mean baseline): $89,115.11

Final Validation RMSE: $7,792.23

Mean Price: $180,921.20

Median Price: $163,000.00

Train MAE (log space): 0.0306

Key Insights:

Stacked Model Outperforms: Captures non-linear relationships better.

Important Features:

OverallQual, GrLivArea.

Question 5

Finally, we would like to use the equation within the prediction result to identify statistically significant undervalued or overvalued properties that signal investment opportunities or risks.

Method:

Calculated residuals (Actual Price – Predicted Price).

Flagged homes where:

Residual < -15% of Predicted Price (Undervalued).

Findings:

Undervalued Homes Detected: len(undervalued) (exact count depends on data).

Common Traits:

High Overall Quality and Living Area.

Located in No.17 neighborhoods

Undervalued Houses Found: 29

	Neighborhood	HouseAge	SalePrice	PredictedPrice	Residual	LotArea	OverallQual	GrLivArea	BedroomAbvGr	FullBath	Discount%
590	16	-0.33	451950	555943.31	-103993.31	0.78	2.86	1.45	0.17	0.80	18.70
1173	15	-1.30	285000	364065.72	-79065.72	0.33	1.40	2.13	1.39	0.80	21.70
462	19	1.09	62383	131786.91	-69403.91	-0.17	-0.80	-1.53	0.17	-1.03	52.70
582	17	2.73	325000	392511.16	-67511.16	0.79	2.86	2.03	0.17	2.63	17.20
714	17	2.07	159500	223494.66	-63994.66	0.35	0.67	1.77	0.17	-1.03	28.60
1046	11	1.26	115000	174493.42	-59493.42	1.71	-0.80	0.66	0.17	-1.03	34.10
1029	18	2.59	197000	253504.47	-56504.47	1.10	0.67	2.35	2.62	0.80	22.30
1022	23	-0.02	287000	342587.06	-55587.06	1.05	1.40	2.16	-1.06	0.80	16.20
770	7	1.48	107000	161093.97	-54093.97	-0.28	-0.06	-0.75	0.17	-1.03	33.60
70	12	-0.68	244000	296483.09	-52483.09	0.81	0.67	1.35	0.17	0.80	17.70
1428	17	0.85	64500	116937.53	-52437.53	0.35	-1.53	-1.19	1.39	0.80	44.80
30	9	1.83	40000	91600.16	-51600.16	-0.12	-1.53	-0.25	0.17	-1.03	56.30
810	9	2.28	55993	106847.31	-50854.31	-0.07	-0.80	-0.96	-1.06	-1.03	47.60
261	5	-1.80	276000	325283.44	-49283.44	0.12	1.40	1.80	0.17	0.80	15.20
528	6	-0.15	200624	249889.62	-49265.62	2.51	-0.06	1.73	1.39	2.63	19.70
198	17	2.85	104000	152392.30	-48392.30	-0.96	-0.06	1.36	2.62	-1.03	31.80
910	3	0.16	88000	130441.69	-42441.69	-0.76	-0.80	-1.04	0.17	-1.03	32.50
1078	14	0.14	145000	185347.75	-40347.75	0.39	-0.06	-0.39	0.17	0.80	21.80
32	5	-1.05	179900	216614.30	-36714.30	0.39	1.40	-0.45	0.17	0.80	16.90
670	7	2.42	103600	137826.56	-34226.56	-0.60	-0.06	-0.13	0.17	0.80	24.80
1263	17	1.41	122000	152922.38	-30922.38	0.22	0.67	0.85	1.39	-1.03	20.20
759	3	2.45	100000	129456.24	-29456.24	-0.52	-0.80	-0.29	-1.06	-1.03	22.80
1066	10	-1.43	151400	180462.00	-29062.00	-1.60	-0.06	1.73	2.62	0.80	16.10
239	7	2.51	113000	140184.02	-27184.02	-0.06	-0.06	0.12	0.17	-1.03	19.40
588	3	1.50	79500	105864.61	-26364.61	0.02	-0.80	-1.29	-1.06	-1.03	24.90
344	10	1.48	85000	110163.16	-25163.16	-2.43	-0.80	-0.76	0.17	-1.03	22.80
534	7	2.16	107500	130439.70	-22939.70	-0.49	-0.80	-0.18	0.17	-1.03	17.60
882	12	1.03	100000	118333.89	-18333.89	-0.45	-0.80	-1.44	0.17	-1.03	15.50
430	2	0.14	85400	102865.96	-17465.96	-3.27	-0.06	-1.13	-1.06	-1.03	17.00

Investment Opportunity:

These homes may offer value appreciation potential if renovated or held long-term.

IV. Discussion

1. Evaluation of Results The analysis in Section III successfully addressed the problem raised in Section I—understanding key factors influencing house prices and identifying optimal buying opportunities.

Key Achievements: ✅ Identified Primary Price Drivers:

Confirmed that home quality (OverallQual), living area size (GrLivArea), and location (Neighborhood) are the strongest predictors of price.

Validated through correlation analysis, feature importance, and predictive modeling.

✅ Optimized Affordability vs. Amenities:

Found that 3-bedroom, 2-bath homes in NAmes or CollgCr offer the best value for budget-conscious buyers.

Clustering revealed trade-offs (e.g., sacrificing a bedroom saves ~$50K).

✅ Quantified Location Impact:

ANOVA proved neighborhood significantly affects price ($p < 0.001$).

Highlighted premium areas (StoneBr, NridgHt) and undervalued ones (BrDale, OldTown).

✅ Accurate Price Predictions:

XGBoost achieved strong performance ($R^2 = 0.89$), demonstrating reliable price estimation.

✅ Detected Undervalued Properties:

Found homes priced >15% below predicted value, mostly older properties in transitioning neighborhoods—potential investment opportunities.

Limitations: ⚠ Data Constraints:

Some neighborhoods had few samples, affecting statistical power.

Log transformation improved skewness but may not capture extreme outliers perfectly.

⚠ Model Generalizability:

Trained on Ames, Iowa data—may not apply to other markets without further validation.

2. Challenges Faced Data Quality Issues: Missing Values: Required careful imputation (median/mode) to avoid bias.

High Cardinality: Some categorical features (e.g., Neighborhood) needed encoding (one-hot vs. label) trade-offs.

Multicollinearity: Highly correlated features (e.g., GarageCars and GarageArea) were removed to stabilize linear models.

Algorithm Selection: Linear Regression was interpretable but less accurate for non-linear relationships.

XGBoost performed better but required hyperparameter tuning (not done here for brevity).

Interpretability vs. Complexity: ANOVA and correlation provided clear insights but ignored interactions.

Clustering grouped homes meaningfully, but optimal k (number of clusters) was subjective.

3. Future Research Directions 🔹 Feature Engineering:

Incorporate macroeconomic factors (e.g., interest rates, unemployment) to improve predictive power.

Test polynomial features (e.g., $GrLivArea^2$) for non-linear effects.

🔹 Model Enhancements:

Hyperparameter tuning (GridSearchCV) for XGBoost to maximize R^2.

Ensemble methods (e.g., stacking Linear Regression + Random Forest) for robustness.

- Geospatial Analysis:

Map neighborhood trends using GIS tools to visualize price hotspots.

Include walkability scores or school district ratings for richer location insights.

- Time-Series Forecasting:

Analyze historical appreciation rates to predict future price movements.

- Fairness Auditing:

Check for bias (e.g., price disparities by demographic factors not in the dataset).

4. Conclusion This project successfully:
   ✓ Identified key price drivers (quality, size, location).
   ✓ Recommended optimal home features for budget buyers.
   ✓ Highlighted undervalued investment opportunities.

V. References

I. pandas
The pandas development team. (2023). pandas-dev/pandas: Pandas (Version 2.1.3) [Computer software]. https://doi.org/10.5281/zenodo.3509134

II. numpy
Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., ... & Oliphant, T. E. (2020). Array programming with NumPy. Nature, 585(7825), 357-362. https://doi.org/10.1038/s41586-020-2649-2

III. matplotlib
Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. Computing in Science & Engineering, 9(3), 90-95. https://doi.org/10.1109/MCSE.2007.55

IV. seaborn
Waskom, M. L. (2021). Seaborn: statistical data visualization. Journal of Open Source Software, 6(60), 3021. https://doi.org/10.21105/joss.03021

V. scipy
Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., ... & SciPy 1.0 Contributors. (2020). SciPy 1.0: Fundamental algorithms for scientific computing in Python. Nature Methods, 17(3), 261-272. https://doi.org/10.1038/s41592-019-0686-2

VI. scikit-learn
Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Duchesnay, É. (2011). Scikit-learn: Machine learning in Python. Journal of Machine Learning Research, 12, 2825-2830. https://www.jmlr.org/papers/volume12/pedregosa11a/pedregosa11a.pdf

VII. xgboost
Chen, T., & Guestrin, C. (2016). XGBoost: A scalable tree boosting system. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 785-794. https://doi.org/10.1145/2939672.2939785

VI. Appendix

First version

```
# Import required libraries
# Import required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
```

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
from sklearn.preprocessing import LabelEncoder, OneHotEncoder, StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.cluster import KMeans
import xgboost as xgb
import warnings
warnings.filterwarnings('ignore')

# Load data
df = pd.read_csv('train.csv', encoding='utf-8')
# if df.columns[-1].startswith('Unnamed'):
#     df = df.drop(df.columns[-1], axis=1)



# List of columns to drop
columns_to_drop = ['Id']

# Drop unnecessary columns
df = df.drop(columns=columns_to_drop)

y = df['SalePrice'].copy()
df = df.drop('SalePrice', axis=1)



# Data Preprocessing
# Handle missing values
numerical_cols = df.select_dtypes(include=np.number).columns
categorical_cols = df.select_dtypes(include='object').columns

# Log transform ONLY if needed (prices should be >0)
if (y <= 0).any():
    y = y + 1 - y.min()  # Shift to positive values first
y_log = np.log1p(y)

# Numerical missing values
for col in numerical_cols:
```

```
    df[col].fillna(df[col].median(), inplace=True)

# Encode categorical columns
for col in df.select_dtypes(include='object').columns:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])

# Categorical missing values
for col in categorical_cols:
    df[col] = df[col].fillna(df[col].mode()[0])

# Remove outliers
df = df[df['GrLivArea'] <= 4000]

# Calculate percentage of missing values for each column
missing_percentage = df.isnull().mean() * 100

# Drop columns with more than 50% missing values
columns_to_drop = missing_percentage[missing_percentage > 50].index
df = df.drop(columns=columns_to_drop)

# Identify high cardinality categorical columns
high_cardinality_cols = [col for col in
df.select_dtypes(include='object').columns
                         if df[col].nunique() > 50]

# Drop high cardinality columns
df = df.drop(columns=high_cardinality_cols)

# Calculate correlation matrix
corr_matrix = df.corr().abs()

# Identify highly correlated features (e.g., correlation > 0.9)
upper_triangle = corr_matrix.where(np.triu(np.ones(corr_matrix.shape),
k=1).astype(bool))
columns_to_drop = [col for col in upper_triangle.columns
                   if any(upper_triangle[col] > 0.9)]

# Drop redundant columns
```

```
df = df.drop(columns=columns_to_drop)

# Log transform skewed numerical features
skewed_features = ['GrLivArea', 'LotArea']
for feature in skewed_features:
    df[feature] = np.log1p(df[feature])

# Scale numerical features
scaler = StandardScaler()
df[numerical_cols] = scaler.fit_transform(df[numerical_cols])

# Create interaction features
df['OverallQual_GrLivArea'] = df['OverallQual'] * df['GrLivArea']
df['TotalBathrooms'] = df['FullBath'] + 0.5 * df['HalfBath']

# Create temporal features
df['TimeSinceRemodel'] = df['YrSold'] - df['YearRemodAdd']

# Remove duplicate rows
df = df.drop_duplicates()

# Encode categorical features
for col in categorical_cols:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])



# Ensure the input is a 2D array
neighborhood_data = df[['Neighborhood']]


print("Unique Neighborhoods:", df['Neighborhood'].nunique())

# Encode categorical variables
encoder = OneHotEncoder(sparse_output=False)
encoded_neigh = encoder.fit_transform(neighborhood_data)
```

```
print("Encoded Data Shape:", encoded_neigh.shape)

# Get feature names for the encoded columns
encoded_neigh_columns = encoder.get_feature_names_out(['Neighborhood'])
encoded_neigh = pd.DataFrame(encoded_neigh, columns=encoded_neigh_columns)

# Add target back
df['SalePrice'] = y
df['SalePrice_log'] = y_log



# Create new features
df['PricePerSqFt'] = df['SalePrice'] / df['GrLivArea']
df['HouseAge'] = df['YrSold'] - df['YearBuilt']



plt.figure(figsize=(12, 6))
sns.histplot(df['SalePrice'], kde=True, bins=50)
plt.title('SalePrice Distribution')
plt.xlabel('Sale Price ($)')
plt.ylabel('Frequency')
plt.show()

# Data Summary
print("Dataset Shape:", df.shape)
print("\nSummary Statistics:")
print(df[['SalePrice', 'GrLivArea', 'OverallQual']].describe())

# Initial Visualizations
plt.figure(figsize=(12, 6))
sns.scatterplot(x='GrLivArea', y='SalePrice', data=df)
plt.title('Price vs Living Area')
plt.show()

# Data Analysis
# 1. Key Drivers of House Prices
corr_matrix = df.corr()

plt.figure(figsize=(24, 16))
```

```
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Matrix')
plt.show()

plt.figure(figsize=(24, 16))
sns.heatmap(corr_matrix[['SalePrice']].sort_values(by='SalePrice',
ascending=False),
            annot=True, cmap='coolwarm')
plt.title('Correlation with Sale Price')
plt.show()

# Random Forest Feature Importance
X = df[numerical_cols]
y = df['SalePrice']

rf = RandomForestRegressor()
rf.fit(X, y)

feature_imp = pd.DataFrame({
    'Feature': X.columns,
    'Importance': rf.feature_importances_
}).sort_values('Importance', ascending=False)

plt.figure(figsize=(12, 6))
sns.barplot(x='Importance', y='Feature', data=feature_imp.head(10))
plt.title('Top 10 Important Features')
plt.show()



# 2. Affordability vs Amenities (Clustering)
cluster_features = ['BedroomAbvGr', 'FullBath', 'GarageCars', 'GrLivArea']
X_temp = df[cluster_features]
df[cluster_features] = scaler.fit_transform(X_temp)


# Run K-Means
kmeans = KMeans(n_clusters=5, random_state=42)
df['Cluster'] = kmeans.fit_predict(df[cluster_features])
```

```
cluster_summary = df.groupby('Cluster').agg({
    'SalePrice': 'mean',
    'BedroomAbvGr': 'mean',
    'FullBath': 'mean',
    'GrLivArea': 'mean',
    'Neighborhood': lambda x: x.mode()[0]  # Most common neighborhood
}).round(1)

# Rename columns for clarity
cluster_summary.columns = ['Avg_Price', 'Avg_Bedrooms', 'Avg_Bathrooms',
'Avg_SqFt', 'Top_Neighborhood']
cluster_summary['Avg_Price'] = cluster_summary['Avg_Price'].apply(lambda x:
f"${x/1000:.0f}K")

print(cluster_summary.sort_values('Avg_Price'))

dream_houses_updated = df[
    (df['Cluster'] == 0) &  # Focus on Cluster 0 (best value)
    (df['BedroomAbvGr'] == 3) &
    (df['FullBath'].between(1.5, 2)) &  # Allow 1.5 baths for flexibility
    (df['GrLivArea'].between(1500, 1800)) &  # Slightly smaller to save $
    (df['Neighborhood'].isin(['NAmes', 'CollgCr']))  # Target top
neighborhoods
]

print(f"Found {len(dream_houses_updated)} dream houses after adjustments.")

# Melt data for plotting
melted = cluster_summary.reset_index().melt(
    id_vars=['Cluster', 'Top_Neighborhood'],
    value_vars=['Avg_Bedrooms', 'Avg_Bathrooms', 'Avg_SqFt'],
    var_name='Metric'
)

plt.figure(figsize=(10, 5))
sns.barplot(
    data=melted,
```

```
    x='Cluster',
    y='value',
    hue='Metric',
    palette='coolwarm'
)
plt.title('Cluster Features Comparison')
plt.ylabel('Average Value')
plt.legend(title='Metric', bbox_to_anchor=(1.05, 1))
plt.show()

plt.figure(figsize=(10, 6))
sns.scatterplot(
    data=dream_houses_updated,
    x='GrLivArea',
    y='SalePrice',
    hue='Neighborhood',
    style='FullBath',  # Show bathroom trade-offs
    size='BedroomAbvGr',  # Show bedroom size
    sizes=(50, 200),
    palette='bright'
)
plt.title('Dream Houses: Size vs. Price Trade-offs')
plt.axhline(y=250000, color='red', linestyle='--', label='Budget Cap')
plt.legend(bbox_to_anchor=(1.05, 1))
plt.show()

# 3. Location Impact (ANOVA)

neighborhood_prices =
df.groupby('Neighborhood')['SalePrice'].median().sort_values(ascending=False)
plt.figure(figsize=(15, 6))
sns.barplot(x=neighborhood_prices.index, y=neighborhood_prices.values)
plt.xticks(rotation=90)
plt.title('Median Prices by Neighborhood')
plt.show()

# Plotting interaction between Neighborhood and HouseAge
plt.figure(figsize=(15, 6))
```

```
sns.pointplot(x='Neighborhood', y='SalePrice', hue='HouseAge', data=df,
palette='viridis', ci=None)
plt.xticks(rotation=90)
plt.title('SalePrice by Neighborhood and HouseAge')
plt.xlabel('Neighborhood')
plt.ylabel('SalePrice')
plt.legend(title='HouseAge', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.show()

import statsmodels.api as sm
from statsmodels.formula.api import ols

# Fit the ANOVA model
model = ols('SalePrice ~ C(Neighborhood) + HouseAge +
C(Neighborhood):HouseAge', data=df).fit()

# Get residuals
residuals = model.resid

# Plot residuals
plt.figure(figsize=(10, 6))
sns.scatterplot(x=model.fittedvalues, y=residuals, alpha=0.5)
plt.axhline(y=0, color='red', linestyle='--')
plt.title('Residual Plot')
plt.xlabel('Fitted Values')
plt.ylabel('Residuals')
plt.show()

# Get ANOVA table
anova_table = sm.stats.anova_lm(model, typ=2)

# Plot F-statistics
plt.figure(figsize=(10, 6))
sns.barplot(x=anova_table.index, y=anova_table['F'], palette='viridis')
plt.title('F-statistics from ANOVA')
plt.xlabel('Features')
plt.ylabel('F-statistic')
plt.xticks(rotation=45)
plt.show()
```

```
# 4. Predictive Modeling
# Prepare data

# Reset indices to ensure alignment
df = df.reset_index(drop=True)
encoded_neigh = encoded_neigh.reset_index(drop=True)

X = pd.concat([df[numerical_cols], encoded_neigh], axis=1)
y = df['SalePrice']



# check that the shape of the data
print("Shape of numerical_cols:", df[numerical_cols].shape)
print("Shape of X:", df[numerical_cols].shape)
print("Shape of encoded_neigh:", encoded_neigh.shape)
print("Shape of y:", y.shape)

# Handle missing values (example: dropping rows with missing values)
#  X = X.dropna()
#  y = y.dropna()



imputer = SimpleImputer(strategy='mean')
X = pd.DataFrame(imputer.fit_transform(X), columns=X.columns)
y = y.fillna(y.mean())

# recheck the shape of the data
print("Shape of X after handling missing values:", X.shape)
print("Shape of y after handling missing values:", y.shape)

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

# Linear Regression
lr = LinearRegression()
lr.fit(X_train, y_train)
lr_pred = lr.predict(X_test)
```

```
print("\nLinear Regression Performance:")
print(f"RMSE: {np.sqrt(mean_squared_error(y_test, lr_pred)):.2f}")
print(f"R²: {r2_score(y_test, lr_pred):.2f}")

# XGBoost
xgb_model = xgb.XGBRegressor()
xgb_model.fit(X_train, y_train)
xgb_pred = xgb_model.predict(X_test)

print("\nXGBoost Performance:")
print(f"RMSE: {np.sqrt(mean_squared_error(y_test, xgb_pred)):.2f}")
print(f"MAE: {mean_absolute_error(y_test, xgb_pred):.2f}")
print(f"Mean Price: ${np.expm1(y_test).mean():,.2f}")
print(f"Median Price: ${np.expm1(y_test).median():,.2f}")
print(f"R²: {r2_score(y_test, xgb_pred):.2f}")

# 5. Identifying Undervalued Homes
df['PredictedPrice'] = xgb_model.predict(X)
df['Residual'] = df['SalePrice'] - df['PredictedPrice']
undervalued = df[df['Residual'] < -0.15*df['PredictedPrice']]



undervalued_details = undervalued[['Neighborhood', 'HouseAge', 'SalePrice',
'PredictedPrice', 'Residual',
                                   'LotArea', 'OverallQual', 'GrLivArea',
'BedroomAbvGr', 'FullBath']]

undervalued_details['Discount%'] = (-undervalued_details['Residual'] /
undervalued_details['PredictedPrice'] * 100).round(1)

# Sort by how undervalued they are (most undervalued first)
undervalued_details = undervalued_details.sort_values(by='Residual')

# Display the details
print(f"\nUndervalued Houses Found: {len(undervalued_details)}")
pd.set_option('display.float_format', lambda x: '%.2f' % x)
print(undervalued_details.to_string())
```

Xb, lgbm, catboost prediction of saleprice

```
import pandas as pd
import numpy as np
import optuna
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
from xgboost import XGBRegressor
from category_encoders import TargetEncoder
from sklearn.linear_model import LassoCV
from sklearn.pipeline import make_pipeline
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.inspection import permutation_importance
from lightgbm import LGBMRegressor
from catboost import CatBoostRegressor
from sklearn.ensemble import StackingRegressor
# Enhanced Evaluation
from sklearn.model_selection import cross_val_score
from category_encoders import LeaveOneOutEncoder


class MultiTargetEncoder:
    def __init__(self, cols):
        self.encoders = {col: TargetEncoder() for col in cols}
        self.cols = cols

    def fit_transform(self, X, y):
        for col in self.cols:
            X[col] = self.encoders[col].fit_transform(X[col], y)
        return X
```

```
    def transform(self, X):
        for col in self.cols:
            X[col] = self.encoders[col].transform(X[col])
        return X


# Load data
train_df = pd.read_csv('train.csv')
test_df = pd.read_csv('test.csv')

# Save target and IDs
train_target = train_df['SalePrice']
test_ids = test_df['Id']
train_df = train_df.drop(columns=['SalePrice', 'Id'])
test_df = test_df.drop(columns=['Id'])

# Identify numerical and categorical columns
numerical_cols = train_df.select_dtypes(include=np.number).columns
categorical_cols = train_df.select_dtypes(include='object').columns

# Handle missing numerical values with train's median
for col in numerical_cols:
    median = train_df[col].median()
    train_df[col].fillna(median, inplace=True)
    test_df[col].fillna(median, inplace=True)

# Handle missing categorical values with train's mode
for col in categorical_cols:
    mode = train_df[col].mode()[0]
    train_df[col].fillna(mode, inplace=True)
    test_df[col].fillna(mode, inplace=True)

# Drop columns with >50% missing in train
missing_percent = train_df.isnull().mean()
columns_to_drop = missing_percent[missing_percent > 0.5].index
train_df = train_df.drop(columns=columns_to_drop)
test_df = test_df.drop(columns=columns_to_drop)
```

```
# Target encode high-cardinality categorical variables
high_cardinality = [col for col in categorical_cols if
train_df[col].nunique() > 10]
low_cardinality = [col for col in categorical_cols if train_df[col].nunique()
<= 10]

# Target encode high-cardinality features
# encoder = LeaveOneOutEncoder(cols=high_cardinality) causes overfitting
encoder = MultiTargetEncoder(cols=high_cardinality)
train_df[high_cardinality] =
encoder.fit_transform(train_df[high_cardinality], np.log1p(train_target))
test_df[high_cardinality] = encoder.transform(test_df[high_cardinality])

# One-hot encode low-cardinality features
train_df = pd.get_dummies(train_df, columns=low_cardinality)
test_df = pd.get_dummies(test_df, columns=low_cardinality)

# Ensure test data has same columns as train
test_df = test_df.reindex(columns=train_df.columns, fill_value=0)

# Drop highly correlated numerical features
corr_matrix = train_df.corr().abs()
upper = corr_matrix.where(np.triu(np.ones(corr_matrix.shape),
k=1).astype(bool))
columns_to_drop = [col for col in upper.columns if any(upper[col] > 0.9)]
train_df = train_df.drop(columns=columns_to_drop)
test_df = test_df.drop(columns=columns_to_drop)

# Log transform skewed features
skewed = ['GrLivArea', 'LotArea', '1stFlrSF', 'TotalBsmtSF']
for col in skewed:
    train_df[col] = np.log1p(train_df[col])
    test_df[col] = np.log1p(test_df[col])



# Feature engineering
temp_df = train_df.copy()
temp_df['SalePrice'] = train_target
```

```
neighborhood_median_price =
temp_df.groupby('Neighborhood')['SalePrice'].median().to_dict()

for df in [train_df, test_df]:
    df['NeighborhoodMedianPrice'] =
df['Neighborhood'].map(neighborhood_median_price)
    df['TotalSF'] = df['TotalBsmtSF'] + df['1stFlrSF'] + df['2ndFlrSF']
    df['TotalBathrooms'] = df['FullBath'] + 0.5*df['HalfBath'] +
df['BsmtFullBath'] + 0.5*df['BsmtHalfBath']
    df['TotalPorchSF'] = df['OpenPorchSF'] + df['EnclosedPorch'] +
df['3SsnPorch'] + df['ScreenPorch']
    df['YearsSinceRemodel'] = df['YrSold'] - df['YearRemodAdd']
    df['IsRemodeled'] = (df['YearBuilt'] != df['YearRemodAdd']).astype(int)
    df['IsNewHouse'] = (df['YrSold'] == df['YearBuilt']).astype(int)

# Log-transform target
y = np.log1p(train_target)

# Split into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(
    train_df, y, test_size=0.2, random_state=42
)

# Hyperparameter tuning
def objective(trial):
    params = {
        'n_estimators': trial.suggest_int('n_estimators', 500, 2000),
        'learning_rate': trial.suggest_float('learning_rate', 0.01, 0.1),
        'max_depth': trial.suggest_int('max_depth', 3, 10),
        'subsample': trial.suggest_float('subsample', 0.6, 1.0),
        'colsample_bytree': trial.suggest_float('colsample_bytree', 0.3,
0.9),
        'gamma': trial.suggest_float('gamma', 0, 0.5),
        'early_stopping_rounds': 50
    }
    model = XGBRegressor(**params, random_state=42)
    model.fit(X_train, y_train, eval_set=[(X_val, y_val)], verbose=False)
    val_preds = model.predict(X_val)
    return np.sqrt(mean_squared_error(y_val, val_preds))
```

```
study = optuna.create_study(direction='minimize')
study.optimize(objective, n_trials=100, timeout=600,
               callbacks=[optuna.study.MaxTrialsCallback(100)])
best_params = study.best_params



# Base models
xgb = XGBRegressor(**best_params, random_state=42)
lgbm = LGBMRegressor(
            num_leaves=31,
            learning_rate=0.05,
            n_estimators=1000,
            random_state=42,
        )
catboost = CatBoostRegressor(
            iterations=1000,
            learning_rate=0.05,
            depth=6,
            silent=True,
            random_state=42
        )

# Stacking
final_estimator = make_pipeline(
    StandardScaler(),
    LassoCV(alphas=[0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1],
            max_iter=10000, cv=5)
)

stacked_model = StackingRegressor(
    estimators=[
        ('xgb', xgb),
        ('lgbm', lgbm),
        ('cat', catboost)
    ],
    final_estimator=final_estimator,
    cv=5
```

```
)



stacked_model.fit(X_train, y_train)



result = permutation_importance(stacked_model, X_val, y_val, n_repeats=10)
low_importance = result.importances_mean < 0.001

print( train_df.shape, train_df.loc[:, ~low_importance].shape)
print( test_df.shape, test_df.loc[:, ~low_importance].shape)
important_cols = train_df.columns[~low_importance]
train_df_reduced = train_df[important_cols]
test_df_reduced = test_df[important_cols]
x_val_reduced = X_val[important_cols]



scores = cross_val_score(stacked_model, train_df_reduced, y,
                         cv=5, scoring='neg_root_mean_squared_error')
print(f"Cross-validated RMSE: {-scores.mean():.2f} ± {scores.std():.2f}")

# Final Training
stacked_model.fit(train_df_reduced, y)
val_preds = stacked_model.predict(x_val_reduced)

# Calculate R² (log and original space)
r2_log = r2_score(y_val, val_preds)
r2_original = r2_score(np.expm1(y_val), np.expm1(val_preds))

# Naive RMSE (predicting mean)
naive_pred = [np.expm1(y_val.mean())] * len(y_val)
naive_rmse = np.sqrt(mean_squared_error(np.expm1(y_val), naive_pred))

final_rmse = np.sqrt(mean_squared_error(np.expm1(y_val),
np.expm1(val_preds)))

# MAE in log-transformed space (same scale as model training)
```

```
mae_log = mean_absolute_error(y_val, val_preds)
print(f"MAE (log space): {mae_log:.4f}")

mae_original = mean_absolute_error(np.expm1(y_val), np.expm1(val_preds))
print(f"MAE (original prices): ${mae_original:,.2f}")



print(f"R² (log space): {r2_log:.3f}")
print(f"R² (original prices): {r2_original:.3f}")
print(f"Naive RMSE (mean baseline): ${naive_rmse:,.2f}")

print(f"Final Validation RMSE: ${final_rmse:,.2f}")
print(f"Mean Price: ${np.expm1(y).mean():,.2f}")
print(f"Median Price: ${np.expm1(y).median():,.2f}")



residuals = np.expm1(y_val) - np.expm1(val_preds)
plt.scatter(np.expm1(y_val), residuals, alpha=0.3)
plt.axhline(0, color='red')
plt.xlabel("Actual Price")
plt.ylabel("Error ($)")
plt.show()

error_percent = (residuals / np.expm1(y_val)) * 100
plt.hist(error_percent, bins=50)
plt.xlabel("% Error")
plt.ylabel("Count")
plt.show()



# Final predictions
# Get stacked model predictions on training data
train_preds = stacked_model.predict(train_df_reduced)
train_mae = mean_absolute_error(y, train_preds)
print(f"Train MAE (log space): {train_mae:.4f}")

# Train calibrator on the residuals
calibrator = LinearRegression()
calibrator.fit(train_preds.reshape(-1, 1), y)
```

```python
# Calibrated prediction function
def calibrated_predict(X):
    raw_preds = stacked_model.predict(X)
    return calibrator.predict(raw_preds.reshape(-1, 1))

lower_model = GradientBoostingRegressor(loss='quantile', alpha=0.05)
upper_model = GradientBoostingRegressor(loss='quantile', alpha=0.95)

lower_model.fit(train_df_reduced, y)
upper_model.fit(train_df_reduced, y)

def predict_with_intervals(X):
    pred = calibrated_predict(X)
    return {
        'prediction': np.expm1(pred),
        'lower_bound': np.expm1(lower_model.predict(X)),
        'upper_bound': np.expm1(upper_model.predict(X))
    }



# Generate final submission with confidence intervals
test_preds = predict_with_intervals(test_df_reduced)



submission = pd.DataFrame({
    'Id': test_ids,
    'SalePrice': test_preds['prediction'],
    #'SalePrice_Lower': test_preds['lower_bound'],
    #'SalePrice_Upper': test_preds['upper_bound']
})

# Add useful metadata
# submission['Prediction_Error_Percent'] = (
#     (submission['SalePrice_Upper'] - submission['SalePrice_Lower']) /
#     submission['SalePrice'] * 100
# )

submission.to_csv('final_submission_with_confidence.csv', index=False)
```