**House Price Prediction Project Report**

**Poojitha Rao Gadela, Nawaz Shareef Sheik**

**College of Business, Dallas Baptist University**

**MSITM -6341: Python Programming**

**Professor: - Dean-Yuan Wang**

**Introduction**

Predicting house prices is a crucial challenge in real estate, influencing both buyers and sellers. This project aims to develop a machine learning model that can accurately predict house prices based on various features using the Ames Housing dataset. The project includes data ingestion, cleaning, feature engineering, exploratory data analysis, model training, evaluation, and deployment.

**Dataset Description**

The dataset used in this project is the **Ames Housing dataset**, which contains detailed information on residential homes in Ames, Iowa. It includes 82 variables describing different aspects of residential homes, such as:

* **Structural features** (e.g., GrLivArea, GarageCars, YearBuilt)
* **Quality indicators** (e.g., OverallQual, ExterQual, KitchenQual)
* **Amenities** (e.g., Fireplaces, PoolArea, ScreenPorch)
* **Sale price** (SalePrice) as the target variable

We began with over 2900 records and selected features that are most predictive for our target.

**Data Pipeline**

We implemented the data pipeline in modular stages. Each file inside the src/ directory has a defined responsibility.

**Data Loading**

Implemented in data\_loader.py. It loads the dataset from data/AmesHousing.csv into a pandas DataFrame. It includes error handling and logging using a custom logger class (logger.py).

**Data Cleaning**

Implemented in data\_cleaning.py. The script performs the following steps:

* Drops columns with more than 50% missing values
* Fills missing numeric values with median
* Fills missing categorical values with mode
* Saves cleaned data to data/cleaned\_ameshousing.csv

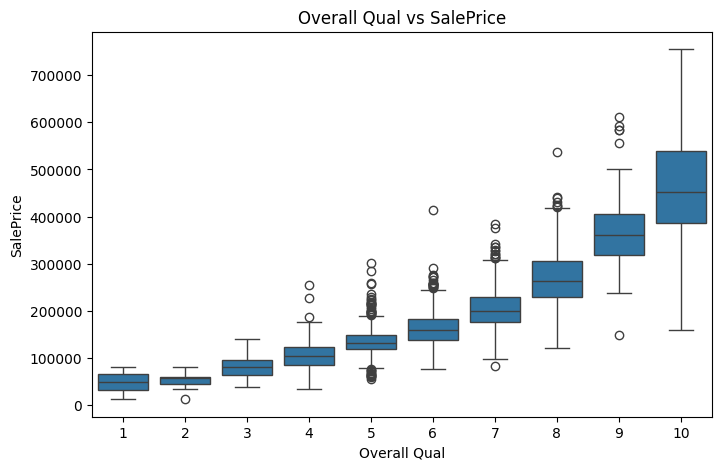
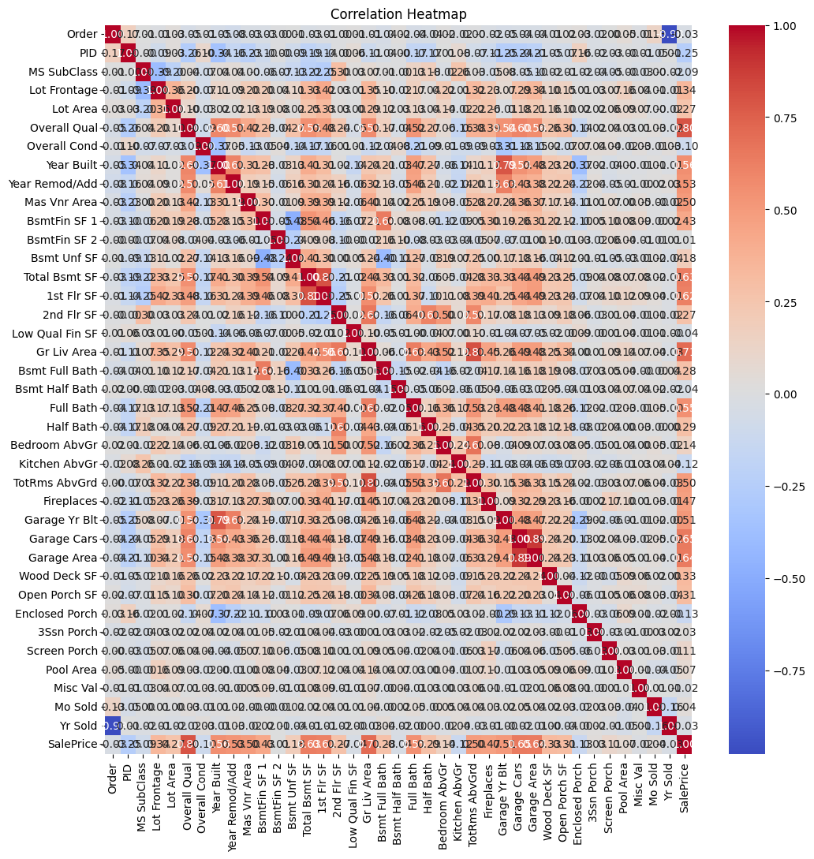
**Exploratory Data Analysis (EDA)**

Visualizations such as distribution plots, correlation heatmaps, and feature relationship plots are created to identify trends, outliers, and relationships. This helps in selecting the most relevant features for the model. Charts include distribution of SalePrice, correlation heatmaps, and scatter plots of GrLivArea vs SalePrice.

Key insights from EDA (performed in EDA.ipynb):

* OverallQual and GrLivArea have the strongest positive correlation with SalePrice
* YearBuilt has shown positive correlation as expected
* Outliers in GrLivArea and GarageArea were detected visually using scatter plots
* Heatmap of top correlated features revealed useful predictors
* Box plots showed the distribution of SalePrice across categorical variables

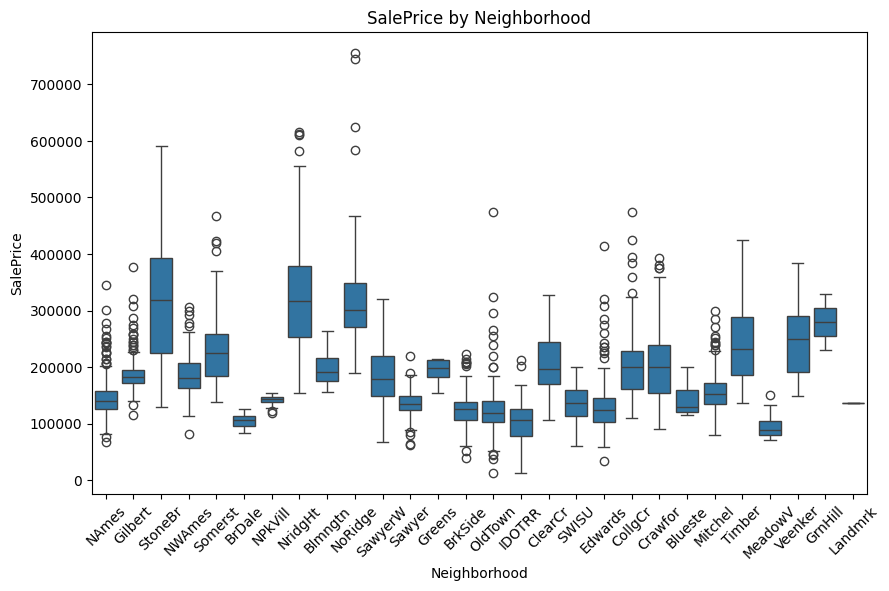
These visualizations helped in feature selection.



**Feature Engineering**

Implemented in feature\_engineer.py. It creates new features using domain knowledge and one-hot encodes categorical variables. Output is saved to data/engineered\_ameshousing.csv.

|  |  |
| --- | --- |
|  |  |



**Model Training**

Implemented in both model\_trainer.py and train\_model.py. This script:

* Loads the engineered dataset
* Selects the top 10 predictive features
* Splits the data (80/20 split)
* Trains multiple models
* Evaluates their performance
* Saves the best model as house\_price\_model.pkl

**Algorithms Used and Justification**

We evaluated **five different regression algorithms**. Below is a detailed explanation of each and why it was included in our pipeline.

**Linear Regression**

Linear regression is the baseline algorithm used to establish a reference point. It assumes a linear relationship between the features and the target. It performed decently but was limited in capturing non-linear trends. Easy to interpret, low complexity but Struggles with outliers and non-linear patterns

**Ridge Regression**

Ridge regression is a regularized version of linear regression that adds L2 penalty to the loss function. It helps avoid overfitting. Improves generalization and prevents overfitting by penalizing large coefficients. Still assumes linearity and can't capture feature interactions

**Lasso Regression**

Lasso uses L1 regularization and has the added benefit of performing feature selection by shrinking some coefficients to zero. Identifies and eliminates irrelevant features automatically. May underfit if regularization is too strong

**Decision Tree Regressor**

A non-linear model that splits the data based on feature thresholds. It captures non-linear interactions well. Handles both numerical and categorical features. Captures non-linearity. Prone to overfitting if not properly tuned

**Gradient Boosting Regressor**

An ensemble technique that builds trees sequentially to minimize error. It gave the best performance in our evaluation. High accuracy, handles non-linearity, and reduces overfitting through boosting. Takes longer to train.

**Model Comparison (Evaluation Metrics)**

| **Model** | **R2 Score** | **RMSE** | **MAE** | **Accuracy (±10%)** |
| --- | --- | --- | --- | --- |
| Gradient Boosting | 0.8868 | 30123.08 | 18590.13 | 62.12% |
| Lasso Regression | 0.8027 | 39777.26 | 25178.97 | 47.10% |
| Ridge Regression | 0.8027 | 39776.94 | 25177.49 | 47.10% |
| Linear Regression | 0.8026 | 39777.73 | 25179.40 | 47.10% |
| Decision Tree | 0.7387 | 45775.14 | 26070.07 | 51.02% |

Among all the models, the Gradient Boosting Regressor outperformed the others by achieving the highest R2 score, lowest RMSE and MAE, and the highest accuracy within a ±10% tolerance. This makes it the most suitable model for deployment in our house price prediction system. Simpler models like Linear, Ridge, and Lasso performed similarly but were less accurate in capturing non-linear relationships. Decision Tree Regressor showed decent results but was slightly less accurate compared to Gradient Boosting.

**Deployment Plan**

We used **FastAPI** to serve the model as a REST API.

* Endpoint /predict accepts JSON input of selected features
* The API loads the trained model (house\_price\_model.pkl) and returns a prediction
* Uvicorn is used to run the server locally

uvicorn app.main:app --reload

**Challenges Faced**

* Managing missing values for over 30 columns
* Dealing with outliers in GrLivArea, GarageArea
* Feature encoding for over 40 categorical columns
* Handling git merge issues in collaborative workflow
* Separating pipelines and contributions per team member (A and B)

**Learnings**

* Strong understanding of data preprocessing workflows
* Practical exposure to regression algorithms and their tuning
* Real-world practice with project modularization
* Use of FastAPI for serving ML models
* Version control in collaborative projects

**Conclusion**

This project demonstrates how machine learning can effectively predict house prices using a structured approach. We built an end-to-end pipeline starting from raw data ingestion to deployment-ready models.

Key takeaways:

* Data quality and feature selection are paramount
* Ensemble models like Gradient Boosting outperform simple models on complex data
* FastAPI provides a lightweight and fast solution for deploying models
* A modular pipeline helps in collaboration and debugging

The final deployed model gives a **R2 score of 0.8868** and can be further improved with hyperparameter tuning and additional features.