House Prices Predictor System

End-to-End Machine Learning Project

By

Amaravathi Veerendra Nath

1. INTRODUCTION

1.1 Theoretical Background

Machine Learning (ML) is a subfield of Artificial Intelligence that focuses on building systems that can learn from data and improve performance over time. Predictive modeling, especially regression analysis, is one of its most widely used applications. Real estate, being a data-rich sector, provides ample opportunities for applying ML models to estimate housing prices based on multiple parameters.

Traditional valuation methods relied on human expertise and statistical regression techniques. However, these approaches often fail to capture the complexity of non-linear relationships between property features and their sale prices. Modern ML algorithms like Random Forests, Gradient Boosting, and Neural Networks offer robust alternatives by modeling complex patterns, reducing bias, and improving predictive accuracy.

This project leverages ML pipelines to automate preprocessing, model training, evaluation, and deployment, while also integrating MLflow for experiment tracking and reproducibility.

1.2 Motivation

The motivation behind this project stems from the importance of real estate in the global economy. Accurate property valuation plays a vital role in the decision-making processes of individuals, businesses, and governments. Buyers rely on fair prices to make informed investments, sellers expect accurate estimates, and financial institutions use property valuation for mortgage approvals.

Manual valuation is prone to bias, time-consuming, and inconsistent. By applying ML, we can provide faster, more objective, and data-driven predictions. Additionally, this project aims to mimic industry practices by incorporating reproducibility, scalability, and deployment readiness — features often missing in academic projects.

1.3 Problem Definition

The central problem is predicting housing prices based on multiple structured features. Challenges include:

- Handling missing data
- Encoding categorical variables
- Managing multicollinearity among predictors
- Preventing overfitting in models
- Ensuring reproducibility and scalability for real-world use

1.4 Aim of the Proposed Work

The primary aim is to build a modular and scalable ML pipeline for predicting housing prices with high accuracy. Specific objectives include:

- Automating preprocessing (missing value imputation, encoding, scaling)
- Experimenting with multiple regression algorithms
- Logging experiments using MLflow for reproducibility
- Building pipelines for training and deployment
- Testing with sample predictions to ensure usability
- Creating a foundation for future cloud deployment and GUI integration

Datasets

The Ames Housing dataset is used, which contains 79 explanatory variables describing residential homes in Ames, Iowa. It is considered more detailed and realistic compared to the Boston Housing dataset. Features include Lot Area, Year Built, Quality, Condition, Number of Rooms, and Garage details.

This dataset is challenging because:
• It contains both numerical and categorical features
• Missing values are present in several columns
• Some features have high correlation, leading to multicollinearity
• The distribution of Sale Prices is skewed, requiring transformations

2. LITERATURE SURVEY

2.1 Survey of the Existing Models/Work

Several works have attempted to predict housing prices using ML and statistical methods:
• Linear Regression: Offers simplicity but poor handling of non-linear features.
• Ridge & Lasso: Useful for reducing multicollinearity and feature selection.
• Decision Trees: Can capture non-linearities but overfit without pruning.
• Random Forest: Improves stability by ensembling trees, but lacks interpretability.
• Gradient Boosting (XGBoost, LightGBM): Delivers high accuracy and is widely used in competitions.
• Neural Networks: Useful for very large datasets but prone to overfitting.

Limitations in prior works:
• Lack of reproducibility and experiment tracking
• Absence of deployment-ready pipelines
• Limited handling of missing data and categorical encoding

2.2 Summary Identified in the Survey

While existing works achieve good predictive performance, they often stop at model evaluation. This project builds on prior research by integrating experiment tracking (MLflow), modular pipelines, and deployment workflows, making it closer to real-world systems.

3. PROPOSED SYSTEM

The proposed system is an end-to-end ML pipeline consisting of data preprocessing, model training, evaluation, experiment tracking, and deployment. Its design ensures automation, modularity, and scalability for real-world applications.

Advantages of the proposed system:

- Modular design
- Automated preprocessing
- Multiple model experimentation
- MLflow-based experiment tracking
- Deployment-ready pipeline

Figure 2: Proposed Workflow (Pipeline Flowchart)

4. PROPOSED WORK

The proposed work involves building a modular pipeline that can be divided into several stages:
• Data Ingestion: Loading raw datasets.
• Preprocessing: Handling missing values, encoding, scaling, transformations.
• Model Training: Training multiple regression models.
• Evaluation: Comparing models based on metrics such as RMSE and R^2.
• Experiment Tracking: Logging results in MLflow.
• Deployment: Saving best models and enabling prediction pipeline.

This modular design ensures each component can be updated independently without disrupting the whole system.

5. MODULES

The system has several modules:
• Data Handling Module
• Preprocessing Module
• Training Module
• Evaluation Module
• Deployment Module
• GUI Module (optional)

5.1 GUI

A user-friendly interface can be developed using Streamlit or Flask. This interface allows users to enter property details and receive predicted prices instantly. This improves accessibility for non-technical users.

6. REQUIREMENTS

6.1 Software Requirements

• Python 3.x
• Pandas, NumPy
• Scikit-learn
• MLflow
• PyYAML
• Jupyter Notebook
• Streamlit/Flask (for GUI)

6.2 System Requirements

• OS: Windows/Linux/MacOS
• Processor: Intel i5 or higher
• RAM: Minimum 8 GB
• Storage: Minimum 2 GB free

Project Code Explanation - Short Summary

This document provides a concise summary of the Python files in the House Prices Predictor System project. It is designed for quick review and interview preparation.

Main Execution Files

• run_pipeline.py – Orchestrates the training pipeline (ingestion → preprocessing → training → evaluation).
• run_deployment.py – Runs the deployment pipeline, loading the best model for predictions.
• sample_predict.py – Tests predictions by inputting house features and outputting predicted price.

Analysis Scripts

• basic_data_inspection.py – Initial dataset inspection.
• univariate_analysis.py – Analyzes individual features.
• bivariate_analysis.py – Studies feature-target relationships.
• multivariate_analysis.py – Analyzes multiple variables at once.
• missing_values_analysis.py – Explores missing data patterns.

Pipelines

• training_pipeline.py – Full training process (from raw data to trained model).
• deployment_pipeline.py – Prepares and serves trained models for real use.

Steps

Pipeline is divided into modular steps:
• data_ingestion_step.py – Loads raw data.
• handle_missing_values_step.py – Handles missing values.
• feature_engineering_step.py – Encodes, scales, creates new features.
• outlier_detection_step.py – Detects/removes anomalies.
• data_splitter_step.py – Splits into train/test sets.
• model_building_step.py – Builds ML models.

• model_evaluator_step.py – Evaluates model performance.
• predictor.py – Generates predictions using trained model.

Core Source Files

Reusable implementations of each step:
• ingest_data.py – Core ingestion logic.
• handle_missing_values.py – Missing data handling.
• feature_engineering.py – Feature transformations.
• outlier_detection.py – Outlier detection methods.
• data_splitter.py – Splitting logic.
• model_building.py – Model training functions.
• model_evaluator.py – Metrics and comparison.

Design Pattern Examples

• strategy_design_pattern.py – Example of interchangeable algorithms.
• template_design_pattern.py – Example workflow skeleton.
• factory_design_patter.py – Example of object creation pattern.

Quick Insight

The project follows a modular design: analysis scripts for data understanding, core source files for reusable ML functions, step files for modular pipeline stages, and pipeline scripts for orchestrating training and deployment. This makes the project production-ready, reproducible, and easy to extend.

Project Code Explanation - House Prices Predictor System

This document provides detailed explanations of each Python (.py) file in the project. The purpose, functions, and connections of each file to the overall system are described for better understanding and readability.

Data Preprocessing Pipeline:

```
        ┌──────────────────────┐
        │   Raw Dataset        │
        │  (AmesHousing.csv)   │
        └──────────┬───────────┘
                   │
                   ▼
      ┌──────────────────────┐
      │ Basic Inspection     │
      │ (shape, dtypes)      │
      └──────────┬───────────┘
                 │
                   ▼
      ┌──────────────────────┐
      │ Missing Value        │
      │   Analysis           │
      └──────────┬───────────┘
                 │
                   ▼
      ┌──────────────────────┐
      │ Feature Engineering  │
      │  - Encoding          │
      │  - Scaling           │
      │  - New features      │
      └──────────┬───────────┘
                 │
                   ▼
      ┌──────────────────────┐
      │ Cleaned Dataset      │
      └──────────────────────┘
```

Main Execution Files

run_pipeline.py

Purpose: Entry point to run the complete ML training pipeline.

Explanation:This script executes the training pipeline which includes data ingestion, preprocessing, feature engineering, model training, and evaluation. It acts as the orchestrator of all steps.

run_deployment.py

Purpose: Entry point for deploying the trained model.

Explanation:This script triggers the deployment pipeline. It loads the trained model, sets up the prediction service, and ensures the model is ready to handle new inputs.

sample_predict.py

Purpose:for testing predictions with a trained model.

Explanation: It loads the deployed prediction service and allows a user to input a sample house configuration. The script returns the predicted house price.

Explanations (Design Patterns)

strategy_design_pattern.py

Purpose: Explains and demonstrates the strategy design pattern.

Explanation: This file provides an educational example of applying the Strategy Pattern in ML workflows, allowing interchangeable algorithms to be selected at runtime.

template_design_pattern.py

Purpose: Explains the template method design pattern.

Explanation: Shows how the template method can define the workflow skeleton, leaving some steps for subclasses to implement. Demonstrates modular design concepts used in this project.

factory_design_patter.py

Purpose: Explains the factory design pattern.

Explanation: This file illustrates how objects (like ML models) can be created using a factory class without exposing the creation logic directly.

Analysis Scripts

basic_data_inspection.py

Purpose: Initial inspection of dataset.

Explanation: Provides functions to explore dataset structure, data types, and detect anomalies in raw data.

univariate_analysis.py

Purpose: Performs univariate analysis.

Explanation: Examines single feature distributions, identifying skewness, outliers, and feature importance.

bivariate_analysis.py

Purpose: Performs bivariate analysis.

Explanation: Studies relationships between two features, especially correlation with target variable (SalePrice).

multivariate_analysis.py

Purpose: Performs multivariate analysis.

Explanation: Examines interaction among multiple variables simultaneously, detecting complex patterns.

missing_values_analysis.py

Purpose: Handles missing data exploration.

Explanation: Identifies missing values, their patterns, and suggests strategies for imputation.s

Pipelines

training_pipeline.py

Purpose: Defines the ML training pipeline.

Explanation: Integrates ingestion, preprocessing, feature engineering, splitting, model building, and evaluation.

```
        ┌──────────────────────┐
        │  Training Pipeline   │
        └──────────┬───────────┘
                   │
                   ▼
        ┌──────────────────────┐
        │ Choose Algorithm     │
        │ (config.yaml)        │
        └──────────┬───────────┘
                   │
    ┌──────────────┼──────────────┐
    ▼              ▼              ▼
LinearReg    RandomForest   GradientBoost
    │              │              │
    ▼              ▼              ▼
 Train & Eval   Train & Eval   Train & Eval
    │              │              │
    └─────┬────────┴──────┬───────┘
          │               │
          ▼               ▼
   Compare Scores    Log in MLflow
          │
          ▼
   Best Model Selected
```

deployment_pipeline.py

Purpose: Defines the deployment pipeline.

Explanation: Focuses on loading the best model, preparing the environment, and serving predictions.

```
    ┌──────────────────────┐
    │  Trained Model       │
    │  (best.pkl)          │
    └─────────┬────────────┘
              ▼
  ┌──────────────────────┐
  │ Deployment Pipeline  │
  └─────────┬────────────┘
              ▼
  ┌──────────────────────┐
  │ Receive Input Data   │
  │ (from user/API)      │
  └─────────┬────────────┘
              ▼
  ┌──────────────────────┐
  │ Preprocess Input     │
  │ (same as training)   │
  └─────────┬────────────┘
              ▼
  ┌──────────────────────┐
  │ Model Prediction     │
  └─────────┬────────────┘
              ▼
  ┌──────────────────────┐
  │ Return Price Output  │
  └──────────────────────┘
```

Pipeline Steps

data_ingestion_step.py

Purpose: Loads raw data from CSV or other sources.

Explanation: Implements a key step in the modular ML pipeline. Each step is reusable and can be modified independently.

handle_missing_values_step.py

Purpose: Imputes or removes missing values in dataset.

Explanation: Implements a key step in the modular ML pipeline. Each step is reusable and can be modified independently.

feature_engineering_step.py

Purpose: Creates new features, encodes categorical data, applies scaling.

Explanation: Implements a key step in the modular ML pipeline. Each step is reusable and can be modified independently.

outlier_detection_step.py

Purpose: Detects and removes anomalies/outliers that distort training.

Explanation: Implements a key step in the modular ML pipeline. Each step is reusable and can be modified independently.

data_splitter_step.py

Purpose: Splits dataset into training, validation, and test sets.

Explanation: Implements a key step in the modular ML pipeline. Each step is reusable and can be modified independently.

model_building_step.py

Purpose: Trains ML models such as Linear Regression, Random Forest, etc.

Explanation: Implements a key step in the modular ML pipeline. Each step is reusable and can be modified independently.

model_evaluator_step.py

Purpose: Evaluates trained models using metrics like RMSE, R^2, MAE.

Explanation: Implements a key step in the modular ML pipeline. Each step is reusable and can be modified independently.

model_loader.py

Purpose: Loads saved trained models for further use.

Explanation: Implements a key step in the modular ML pipeline. Each step is reusable and can be modified independently.

predictor.py

Purpose: Executes predictions using the trained model pipeline.

Explanation: Implements a key step in the modular ML pipeline. Each step is reusable and can be modified independently.

prediction_service_loader.py

Purpose: Sets up the prediction service to handle requests.

Explanation: Implements a key step in the modular ML pipeline. Each step is reusable and can be modified independently.

dynamic_importer.py

Purpose: Dynamically loads modules or classes at runtime for flexibility.

Explanation: Implements a key step in the modular ML pipeline. Each step is reusable and can be modified independently.

Core Source Code

ingest_data.py

Purpose: Core implementation of data ingestion functions.

Explanation: This file contains reusable functions that are called by steps and pipelines for execution.

handle_missing_values.py

Purpose: Functions for missing value imputation/removal.

Explanation: This file contains reusable functions that are called by steps and pipelines for execution.

feature_engineering.py

Purpose:Contains transformations and feature engineering logic.

Explanation: This file contains reusable functions that are called by steps and pipelines for execution.

outlier_detection.py

Purpose: Implements methods for detecting and treating outliers.

Explanation: This file contains reusable functions that are called by steps and pipelines for execution.

data_splitter.py

Purpose: Implements train-test splitting logic.

Explanation: This file contains reusable functions that are called by steps and pipelines for execution.

model_building.py

Purpose: Defines functions to build/train ML models.

Explanation: This file contains reusable functions that are called by steps and pipelines for execution.

model_evaluator.py

Purpose: Implements metrics calculation and model comparison logic.

Explanation: This file contains reusable functions that are called by steps and pipelines for execution.

Experiment Tracking Flowchart (MLflow)

```
┌──────────────────────┐
│   Start Training     │
└──────────┬───────────┘
           │
           ▼
┌──────────────────────┐
│   Run Experiment     │
└──────────┬───────────┘
           │
           ▼
┌──────────────────────┐
│ Log Parameters       │
│ (model, hyperparams)│
└──────────┬───────────┘
           │
           ▼
┌──────────────────────┐
│ Log Metrics          │
│ (RMSE, R², MAE)      │
└──────────┬───────────┘
           │
           ▼
┌──────────────────────┐
│ Save Artifacts       │
│ (model.pkl, plots)   │
└──────────┬───────────┘
           │
           ▼
┌──────────────────────┐
│ MLflow Dashboard     │
│ Compare Experiments  │
└──────────────────────┘
```

Design Pattern Usage Flowchart

```
  ┌──────────────────────────────┐
  │      Training Pipeline       │
  └────────────┬─────────────────┘
               │
               ▼
┌───────────────────────────────┐
│ Factory Pattern               │
│ - Creates model based on      │
│   config.yaml (LR, RF, etc.)│
└────────────┬──────────────────┘
             │
             ▼
┌───────────────────────────────┐
│ Strategy Pattern              │
│ - Selects training strategy │
│   (cross-val, holdout, etc.)│
└────────────┬──────────────────┘
             │
             ▼
┌───────────────────────────────┐
│ Template Pattern              │
│ - Defines training skeleton │
│   (load→process→train→eval) │
└───────────────────────────────┘
```

7. DESIGN OF THE PROJECT

The project follows a modular architecture, ensuring that each stage of the pipeline is independent. This enables easy debugging, testing, and deployment. Key design features:
• Separate training and deployment pipelines
• Configurable parameters via YAML
• MLflow integration for experiment logging
• Clear separation of concerns across modules

Data Insights

- The dataset contained missing values in several categorical and numerical columns (like LotFrontage, GarageYrBlt, PoolQC). Imputation strategies (mean, mode, and special category “None”) improved dataset completeness.

- Outliers (e.g., houses with extremely high square footage but low price) negatively impacted regression performance. Removing them improved stability.
- Feature Engineering (like log-transforming skewed variables, encoding categorical features, and creating new interaction features) significantly improved model accuracy.
- Strong correlations were found between OverallQual, GrLivArea, GarageCars and the target variable SalePrice
- these were the most influential predictors.

Model Training & Evaluation Findings

- Baseline models like Linear Regression captured basic relationships but underperformed due to multicollinearity and non-linearity in data.
- Tree-based models (Random Forest, Gradient Boosting, XGBoost) provided much better predictive accuracy, handling outliers and feature interactions effectively.
- Cross-validation and MLflow tracking showed that Gradient Boosting achieved the best balance between accuracy and generalization.
- Feature importance plots confirmed that living area, overall quality, and number of cars in garage are key drivers of house prices.

Pipeline & System Insights

- The modular training pipeline (data ingestion → preprocessing → feature engineering → training → evaluation) ensured reusability and scalability.
- The deployment pipeline allowed seamless loading of the best-trained model and serving predictions in real-time.
- By structuring the project into steps and pipelines, it became easy to extend (e.g., adding new models or preprocessing steps without rewriting code).

- The use of design patterns (Strategy, Factory, Template) demonstrated industry-grade software engineering principles making the project stand out compared to typical ML projects.

Overall Conclusions

- Data preprocessing (handling missing values, outliers, feature engineering) had a **bigger impact on accuracy** than simply trying more complex models.
- Gradient Boosting (or XGBoost) emerged as the most reliable model for predicting house prices.
- The system is **production-ready** — with pipelines, modularity, and deployment — not just an academic ML script.
- The project shows strong **practical ML engineering skills**: reproducibility, modular design, and deployment focus.

8. FUTURE SCOPE

Potential improvements include:

• Hyperparameter tuning with GridSearch or Bayesian Optimization
• Use of advanced ML models like XGBoost, LightGBM, and Deep Neural Networks
• Full REST API or Web App deployment
• Containerization with Docker
• Cloud deployment on AWS/GCP/Azure
• Model explainability using SHAP or LIME

9. CONCLUSION

The House Prices Predictor System successfully demonstrates the application of ML in real estate. It provides a modular, scalable, and reproducible pipeline that integrates preprocessing, model training, evaluation, and deployment. By using MLflow, it ensures that experiments are tracked and reproducible. This project is closer to an industry-grade solution compared to typical academic projects.

Achievements:

1. **Comprehensive Data Handling**
    - Missing values analyzed & imputed.
    - Categorical and numerical features processed with appropriate encoding/scaling.
    - Exploratory Data Analysis (EDA) provided deep insights into housing market trends.
2. **Pipeline Automation**
    - Training and deployment pipelines designed to ensure **reusability and automation**.
    - MLflow integrated for **experiment tracking**, allowing comparison of models & metrics.
3. **Design Patterns for Scalability**
    - Strategy, Template, and Factory patterns used to make the system **flexible and extensible** (e.g., easily switching models).
4. **Model Performance**
    - Multiple ML models tested (Linear Regression, Ridge, Random Forest, etc.).
    - Evaluation metrics (RMSE, R^2) tracked to choose the **best performing model**.
5. **Deployment Ready**
    - Deployment pipeline ensures that trained models can be saved, loaded, and used for real-world predictions.
    - sample_predict.py provides a simple interface for testing predictions on new data.

10. REFERENCES

1. Ames Housing Dataset - Kaggle
2. Scikit-learn Documentation
3. MLflow Documentation
4. Research papers on regression models for real estate prediction
5. XGBoost and LightGBM official papers