Automated Property Value Prediction using MLOps

Stefano Chen
Software Engineer
University of Trieste
Piazzale Europa, 1, 34127 Trieste TS
stefano.chen@studenti.units.it

ABSTRACT

This paper presents an MLOps-based approach for predicting residential property values in California. A local real estate agency relies on manual estimations with an average error rate of 20%. To improve accuracy, we built a complete machine learning pipeline incorporating experiment tracking, model selection, dataset versioning, and web deployment. Among five regression models tested, Random Forest achieved the best results and was further optimized using Optuna. The final model was deployed via a Streamlit web app integrated with MongoDB to log user inputs and predictions. All experiments, datasets, and models were tracked using CometML. The deployed system improves accuracy, transparency, and reproducibility over manual approaches.

CCS Concepts

• **Information system → Geographic information systems**

• **Computing methodologies → Machine learning**

• **Software and its engineering → Software creation and management**

Keywords

MLOps; property valuation; model deployment; CometML; Streamlit; regression models; real estate; Github; Optuna; MongoDB

1. INTRODUCTION

Accurate property valuation is vital in the real estate market, particularly in high-demand regions like California.

A local agency relied on manual expert assessments, which were often time-consuming, inconsistent, and exhibited high error margins (≈20%).

We collaborated with the agency to create a machine learning-based solution supported by an MLOps workflow. The system leverages historical housing data and modern tooling to ensure reproducibility, model traceability, and fast deployment. Our pipeline includes dataset versioning, experiment tracking with CometML, hyperparameter tuning using Optuna, and real-time deployment through Streamlit Community Cloud. Additionally, MongoDB was used to log requests and predictions for future auditing and improvement.

The codebase is available at: GitHub Repository

The live app is accessible at: Streamlit App

2. RELATED WORK

Several machine learning approaches have been applied to real estate valuation, ranging from simple linear models to advanced ensemble methods. While earlier works emphasized statistical modeling, recent research highlights the importance of operational practices like experiment tracking, versioning, and deployment pipelines. Despite this, few studies integrate the full MLOps lifecycle in practical housing valuation systems. Our approach fills this gap by combining predictive modeling with modern MLOps tools to deliver a reproducible and scalable solution.

3. METHODOLOGY

3.1 KPI

To evaluate the effectiveness of the system, the following Key Performance Indicators (KPIs) were defined:

• **MAPE < 15%**: To significantly improve the property evaluation accuracy.

• **System/Model latency < 2 seconds**: For responsive real-time predictions in the deployed web application.

• **100% System Reproducibility**: Ensured through consistent tracking of dataset versions, code, and model parameters.

3.2 Agile Development

The development process followed an Agile Kanban methodology, which emphasized continuous delivery. Tasks were managed using Jira's Kanban board, facilitating real-time visibility and efficient workflow control.

The board included the following stages (columns):

• **To Do**: Approved and prioritized tasks

• **In Progress**: Currently active developments

• **Review/Testing**: Completed tasks under validation

• **Done**: Finished and accepted work

This approach enabled dynamic task prioritization and minimized bottlenecks, supporting a smooth integration of new features and improvements.

3.3 Versioning & CI/CD strategy

We employed Github for source code versioning, ensuring collaborative development and reproducibility. CometML was adopted to track experiments, dataset versions, environment configurations, model performance metrics (RMSE, MAE, MAPE and R2) and model registration. This setup enabled efficient comparison and reproducibility.

To streamline development and deployment, a CI/CD pipeline was implemented using **GitHub Actions**, automating key steps in the project workflow and ensuring reliability.

GitHub Actions was configured to automatically trigger workflows on each push or pull request. The CI pipeline included: 1) **Linting**, to ensure adherence to Python coding standards. 2) **Unit Test**, to validate critical functions and utility components. 3) **Dependency Check**, to confirm a consistent and buildable environment.

The CD pipeline was achieved by designing the Streamlit app to always pull the latest production-ready model from the CometML model registry, and to redeploy automatically to Streamlit Community Cloud.

3.4 Data Cleaning and Transformation

The dataset (20640 entries with 14 features) used in this project contains detailed records of residential properties across California, featuring variables such as geographical coordinates, house attributes, and proximity to the ocean and major cities. To prepare the data for our regression task, we carried out several cleaning and transformation steps. To address missing values - both within the dataset and in potential user input - we applied a mean imputation. Several numerical features showed right-skewed distribution, and presence of outliers, so we applied log transformation to reduce skewness and enhance the model's sensitivity to variation across a more balanced range. Finally, we standardized all numerical features using a Standard Scaler to improve convergence during model training.

3.5 Feature Engineering

To enhance the model's predictive performance, several composite features were engineered based on domain knowledge and Exploratory Data Analysis (EDA):

• **Rooms_Per_House**: Represents the average number of rooms per house, serving as a proxy for house size.

• **Bedrooms_Ratio**: Indicates the proportion of bedrooms to total rooms, highlighting property layout and potential luxury level.

• **People_Per_House**: Captures neighborhood density and may reflect housing demand in a given area.

These derived features help capture more abstract relationships and interactions within the data. Initial correlation analysis showed that these engineered features had stronger predictive power than some original features.

3.6 Preprocessing Pipeline

A consistent preprocessing pipeline was implemented to ensure that the input data is properly transformed before being passed to the machine learning model. This pipeline was critical during both training and inference, guaranteeing that any input – whether from the dataset or from the user – is processed in the same way.

The preprocessing pipeline was implemented using Scikit-learn's **Pipeline** and **ColumnTransformer**. and consist of the following steps:

• **Missing Value Imputation:** Mean imputation was applied to handle missing values in both the training data and user-provided inputs.

• **Log transformation** was applied to reduce skewness and mitigate the impact of outliers.

• A **Standard Scaler** was used to standardize numerical features, improving model convergence during training.

3.7 Model Selection

Five regression algorithms – Linear Regression, Decision Tree, K-Nearest Neighbors, Support Vector Machine, and Random Forest – were evaluated using 10-fold cross validation with default parameters. RMSE and MAPE were the primary evaluation metrics.

3.8 Hyperparameter Tuning

Random Forest, the top-performing algorithm, underwent hyperparameter optimization with **Optuna**. Key parameters such as number of estimators, criterion, and minimum samples per leaf were tuned over 100 trials. Every tuning experiment was versioned and tracked in CometML, ensuring traceability and reproducibility. The best fitted model was serialized along with its preprocessing pipeline into a **single pickle file**, which was registered in the CometML's Model Registry for deployment.

3.9 Web Deployment

The final model was deployed using Streamlit web application, offering an intuitive web interface for real-time property value predictions. The app retrieves production-ready models from CometML's model registry.

3.10 Logging, Monitoring, and Alerting

To ensure system robustness and maintain prediction accuracy over time, we implemented a logging, monitoring, and alerting pipeline combining **MongoDB Atlas**, **GitHub Actions**, and **Jira**.

The deployed Streamlit web application logs all user inputs and corresponding model predictions to a MongoDB Atlas database.

A **scheduled** GitHub Actions workflow was designed to run at fixed intervals (1st and 16th of each month) and execute a drift detection and alerting routine composed of:

1. **Data Retrieval**: Fetches logged user input data from MongoDB Atlas and downloads the training dataset from the CometML Artifact Registry.

2. **Data Drift Detection**: The fetched data is passed to a custom drift detection script powered by Evidently AI, which generates statistical drift reports. If any feature shows significant deviation from the training distribution, the script flags it as data drift.

3. **Alerting via Jira**: When drift is detected, the GitHub actions pipeline automatically triggers the **Jira REST API** to create a new bug issue labeled "Data Drift Alert". Jira is configured to notify the team via automated email alerts upon bug issue creation.

3.10.1 Understanding Model and Data Drift

Data Drift (also known as covariate shift) occurs when the statistical distribution of input feature changes over time compared to the training data. In our specific case, this shift can happen gradually or suddenly dur to various external factors such as economic fluctuations, policy changes, etc.

Model Drift (also called concept drift) happens when the relationship between the input feature and the target variable evolves. This means that even if the input data distribution stays the same, the model's assumptions about how features affect the outcome no longer hold.

In this project, due to the delayed availability of actual sale price (labels), model drift cannot be directly monitored in real-time. Instead, the system emphasizes data drift detection as an early-warning mechanism. If a significant shift is detected in user inputs, it may indicate the need for retraining.

By continuously monitoring data drift, we proactively safeguard the model against silent degradation, ensuring more consistent and trustworthy predictions in production.

4. RESULTS

4.1 Model Selection Outcome

To determine the most suitable model for property value prediction, we conducted a comparative analysis of five widely used regression algorithms: Linear Regression, Decision Tree, K-

Nearest Neighbors (KNN), Support Vector Machine (SVM), and Random Forest. Each model was evaluated using the default parameters and a 10-fold cross-validation to ensure robustness and minimize overfitting cases.

The evaluation metrics used are Root Mean Squared Error (RMSE) [Figure 1] and Mean Absolute Percentage Error (MAPE) [Figure 2].

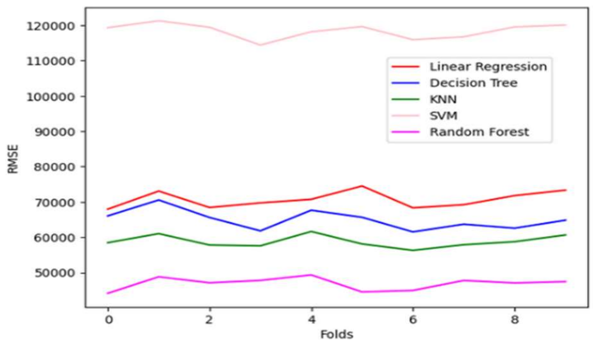

Figure 1. RMSE plot during model selection.

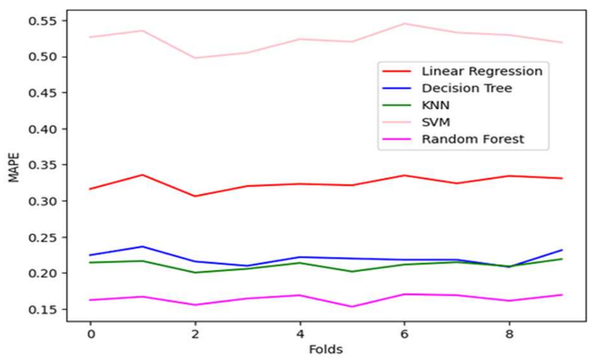

Figure 2. MAPE plot during model selection.

Table 1. Model Selection Metrics

Model	RMSE (avg)	MAPE (avg)
Linear Regression	70700	32%
Decision Tree	65000	22%
KNN	59000	21%
SVM	118400	52%
Random Forest	47000	17%

Linear Regression showed the second poorest performance, likely due to its inability to capture the complex, non-linear relationship within the dataset.
Decision Tree and **KNN** performed moderately well.
SVM struggled with the dimensionality of the dataset, resulting in the poorest performance relative to the other models.
Random Forest achieved the best results across both the metrics, making it the most suitable model for this problem.

4.2 Hyperparameter Tuning Outcome

Using Optuna, we performed automated hyperparameter optimization on the Random Forest model. The search space included:

• n_estimators: 100 – 300

• criterion: [“squared_error”, “absolute_error”, “friedman_mse”, “poisson”]

• min_samples_leaf: 1 – 50

After 100 optimization trials the best configuration found was:

Table 2. Best Hyperparameter

Hyperparameter	Value
Number of Estimators	135
Criterion	“poisson”
Min Samples per Leaf	2

This configuration scored the following metrics value:

Table 3. Best Configuration metrics value

Metric	Value
RMSE	46395.6
MAPE	16.4%
MAE	29509.7
R2	83.6%

The hyperparameter tuning improved marginally the performance of the model, this is most likely due to the low number of optimization trials and the complexity of the problem.

5. LIMITATIONS

While the system is both deployable and modular, it currently lacks an automated retraining mechanism. This limitation is caused by the delayed availability of ground truth labels (i.e., actual property sale prices) for the logged predictions, which are essential for supervised model updates. Consequently, retraining must be performed manually once new labeled data becomes available.

6. CONCLUSION

We developed and deployed a reproducible machine learning workflow to predict houses prices in California.

Random Forest, selected through rigorous model comparison and tuned via Optuna, achieved acceptable performance, allowing the real estate agency to shift resources away from the labor-intensive task of manually estimating house prices.

Future enhancements will focus on integrating a CI/CD pipeline to support automated retraining and further refining hyperparameter optimization strategies.

7. ACKNOWLEDGMENTS
We thank the local real estate agency for providing domain insights and the open-source community for maintaining tools such as Streamlit, Scikit-Learn, and Optuna

8. REFERENCES
[1] Scikit-learn. Random Forest Regressor Documentation. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html.

[2] Google Cloud. MLOps: Continuous delivery and automation pipelines in machine learning. https://cloud.google.com/architecture/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning

[3] Treveil, M., & Pellegrino, A. (2022). *Practical MLOps: Operationalizing Machine Learning Models*. O'Reilly Media.

[4] Soriano, F. (2023). California Housing Prices – Extra Features. https://www.kaggle.com/datasets/fedesoriano/california-housing-prices-data-extra-features.