

House Price Prediction System Using Machine Learning

M Angel T Maria Ruban*, Dr. Manoj Singh**

*(MSc Computer Science Part II, SIES College of Arts, Science and Commerce, Sion, Mumbai, India.

Email: angelruban147@gmail.com)

**(Department of Computer Science, SIES College of Arts, Science and Commerce, Sion, Mumbai, India.

Email: manojks@sies.edu.in)

Abstract:

Estimating the market value of a home is a task on which buyers, sellers, lenders and agents still rely heavily on informal judgment. This research project develops and evaluates a web-based House Price Prediction System in which supervised regression models are served through a Django application. Users enter six property attributes (built-up area, bedrooms, bathrooms, year of construction, urban or rural location and garage availability) and receive an instant estimate, while administrators manage users, review stored predictions and produce date-range reports. Linear regression, decision tree regression and random forest regression were trained on a controlled dataset of 2,000 records and compared on a held-out 20 percent test split and with five-fold cross-validation. The random forest gave the lowest errors, with a test coefficient of determination (R-squared) of 0.997 and a mean absolute error of about 0.23 million rupees, whereas the linear model reached 0.930. Feature-importance and ablation analyses show that predictions depend mainly on area and location, and a scalability test shows that the serialized forest grows linearly with training size. Because the data are synthetic, the reported accuracy shows that the pipeline works as designed and should not be read as real-market performance; limitations and the steps needed to move to real transaction data are discussed.

Keywords — house price prediction, machine learning, random forest, regression, Django, web application

I. INTRODUCTION

For most households, buying a home is the largest financial commitment they will make, and the asking price is often the first number that frames the negotiation. Professional valuation traditionally rests on comparable sales and on hedonic regression, in which the price of a dwelling is explained as the sum of the implicit prices of its attributes [1], [2]. These approaches work, but they depend on an expert's time, they are hard to repeat consistently, and they rarely give an ordinary buyer a quick answer to the question "what is a fair price for this house?"

Machine learning offers a more flexible alternative. Methods that learn non-linear relationships and interactions directly from data have been argued to suit prediction problems of this type well [3]. However, much of the published work stops at a

notebook experiment: a model is fitted, a score is reported, and the model is never placed in front of the people who would use it.

This research project addresses that gap by building a complete house price prediction system, from data preparation and model comparison to a web application with separate user and administrator modules. The rest of the paper is organized as follows: Section II states the problem, Section III lists the objectives, Section IV reviews the literature, Section V describes the methodology, Section VI presents the system architecture, Section VII describes the implementation, Section VIII reports the results, Section IX records the observations, and Section X concludes.

II. PROBLEM STATEMENT

Buyers, sellers and small investors usually have no inexpensive way to obtain a defensible price estimate for a residential property. Manual valuation is slow and subjective; informal advice from brokers may be biased; and published prediction models, where they exist, are seldom packaged so that a non-technical person can use them. In addition, a model that is deployed without a record of its inputs, outputs and version cannot be audited or improved. The problem addressed here is therefore to design and evaluate a system that (i) predicts the price of a residential property from a small set of attributes with low error, (ii) explains which attributes matter most, and (iii) delivers the predictions through a secure, auditable web interface.

III. OBJECTIVES

The objectives of this research project are:

- To develop a machine learning model that predicts residential property prices from key property attributes.
- To compare linear regression, decision tree regression and random forest regression under one fixed, reproducible evaluation protocol.
- To identify the attributes that most strongly influence the predicted price.
- To provide a web application in which users can submit property details, obtain an estimate and review their prediction history.
- To provide an administrator module for managing users, inspecting predictions and generating date-range reports.
- To assess the limitations and scalability of the proposed system and to outline the path to validation on real data.

IV. LITERATURE REVIEW

Hedonic pricing theory, formalized by Rosen [1], remains the conceptual basis of property valuation, and was applied to housing data in the well-known study of Harrison and Rubinfeld [2]. Its linear, additive form is easy to interpret but cannot represent interactions, such as a premium for floor area that differs between neighborhoods, unless they are specified by hand.

Data-driven models have been compared with the hedonic approach many times. Limsombunchai [4]

compared a hedonic price model with an artificial neural network for house prices in New Zealand and reported results that support the potential of the neural network. Park and Bae [5] compared four classification algorithms on housing records from Fairfax County, Virginia, and reported that a rule-induction method outperformed the alternatives on their data. Public benchmarks have since made such comparisons easier to repeat; the Ames, Iowa dataset, for example, was proposed as a richer replacement for the older Boston data [6].

Ensemble methods occupy a prominent place in this literature. Random forests [7] average many decorrelated trees and so reduce the variance of a single tree, and gradient-boosted trees [8] are widely used for tabular prediction.

Research gap. The studies above concentrate on model accuracy. They generally do not describe how the model is exposed to end users, how predictions are stored for audit, or how an administrator supervises the service. The present work fills this gap by reporting the surrounding system together with the experiments.

V. METHODOLOGY

A. Research Workflow

The work followed five steps: (1) generate a controlled dataset with a known price structure; (2) preprocess the attributes inside a single pipeline; (3) train three regression models; (4) evaluate them on held-out data and by cross-validation, and analyze feature relevance; and (5) serialize the best pipeline and integrate it into a Django web application.

B. Dataset

Real transaction records were not available for this project, so a controlled dataset of 2,000 houses was generated with a fixed random seed (42). Table I lists the attributes and how each was sampled. The sale price in rupees is produced by a rule that follows the additive-attribute logic of hedonic pricing [1] but includes a multiplicative interaction; this rule was defined by the authors for this study and is not taken from any published model:

$$P = A \cdot r \cdot m - 1000 \cdot (2022 - Y) + 50000 \cdot G + \varepsilon \quad (1)$$

where A is the area in square feet, $r = 3000 + 800 \cdot \text{BHK}$ is a rate that grows with the number of bedrooms (BHK), Y is the year built, G the garage

indicator, m a location multiplier (1.3 for urban and 0.8 for rural) and ϵ Gaussian noise with a standard deviation of 50,000 rupees. The resulting prices average 15.34 million rupees and range from 1.09 to 44.45 million. The number of bathrooms is generated but does not enter the rule. A synthetic design has two advantages: the ground truth is known, so the behavior of the models can be checked against it, and the experiment can be repeated exactly. Its disadvantage, discussed in Section IX, is that it says little about real market behavior.

TABLE I
ATTRIBUTES OF THE DATASET

Attribute	Type	How it was generated
Area (sq ft)	Numeric	Uniform, 300–5000
Bedrooms (BHK)	Integer	1–5, mode at 3
Bathrooms	Integer	BHK plus Poisson(1), clipped to 1–6
Year built	Integer	Uniform, 1950–2021
Location	Binary	Urban 60%, rural 40%
Garage	Binary	Present in 30%
Price (target)	Numeric	Equation (1)

C. Preprocessing

All inputs are numeric or binary, so categorical encoding reduces to mapping “urban” to 1 and “garage: yes” to 1. Each numeric feature x is standardized with the usual z-score transformation implemented in scikit-learn [9]:

$$z = (x - \mu) / \sigma \quad (2)$$

where μ and σ are the mean and standard deviation of that feature in the training data. The scaler sits inside the same pipeline as the regressor, so it is fitted on training data only, which prevents leakage from the test set, and the identical transformation is applied at prediction time. Standardization does not change tree-based models but is kept for all three so that the pipelines are structurally identical. In the web application, missing or out-of-range entries are rejected by form validation before they reach the model.

D. Models

Three regressors of increasing flexibility were compared. *Linear regression* fits an additive model by ordinary least squares and serves as the hedonic-style baseline [10]. *Decision tree regression* grows a fully expanded tree by recursive binary splitting [11] and can model non-linear effects and interactions but is prone to high variance. *Random forest regression*

[7] averages the predictions of B trees, each grown on a bootstrap sample with random feature subsets at each split:

$$\hat{y}(x) = (1/B) \sum T_b(x) \quad (3)$$

where T_b is the b -th tree; here $B = 100$. All models come from scikit-learn [9] with default hyperparameters and a fixed random state; no tuning was performed, so that the comparison reflects the model families rather than the effort spent on each.

E. Evaluation Metrics

The data were divided into 1,600 training and 400 test records (80/20). Performance on the test set is reported with the mean absolute error (MAE), the root mean squared error (RMSE) and the coefficient of determination (R-squared). For n test samples with actual prices y_i , predictions p_i and mean actual price $\bar{y}$, these are defined by the standard formulas [12], [13], [10]:

$$\text{MAE} = (1/n) \sum |y_i - p_i| \quad (4)$$

$$\text{RMSE} = \sqrt{(1/n) \sum (y_i - p_i)^2} \quad (5)$$

$$R^2 = 1 - \sum (y_i - p_i)^2 / \sum (y_i - \bar{y})^2 \quad (6)$$

MAE measures the average size of the error in rupees, RMSE penalizes large errors more heavily and is never smaller than MAE [13], and R-squared expresses the share of price variance explained. As a check on the dependence of the result on one particular split, R-squared was also computed by shuffled five-fold cross-validation on all 2,000 records.

VI. SYSTEM ARCHITECTURE

A. Architectural Layers

The software is organized into four layers and adopts the Model–View–Template (MVT) convention of Django [14], as shown in Fig. 1. The presentation layer, written with HTML, CSS and Bootstrap, provides registration, login, a prediction form, a prediction history and an administrator console. The application layer validates input, enforces authentication and role-based access, calls the model and renders results. The machine learning layer holds a serialized scikit-learn pipeline that is loaded by the application when a prediction is requested. The data layer, implemented with SQLite for development, stores user accounts and every prediction together with its inputs, output, model version and timestamp. Model training is performed

offline by a separate script, so retraining does not interrupt the web service.

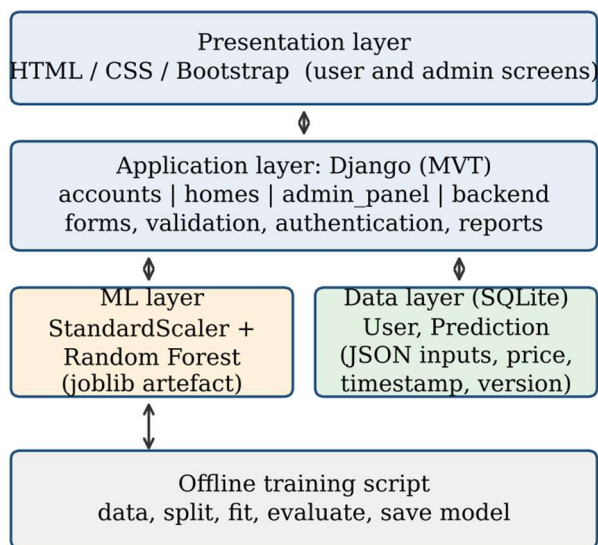


Fig. 1 Layered architecture of the House Price Prediction System.

B. User and Administrator Modules

Two roles are supported. An ordinary user can register, submit property details, view the estimate and browse or open earlier predictions, as well as edit a profile and change a password. An administrator, identified as a Django superuser, sees a dashboard with user and prediction counts, can list, search and inspect users and predictions, delete records, and generate reports for a chosen date range.

C. Technology Stack

Table II lists the technologies used to build the system. The web tier is a conventional Django application; the machine learning tier is a scikit-learn pipeline that Django loads through joblib, so the model and the web application remain loosely coupled.

TABLE II
TECHNOLOGY STACK OF THE HOUSE PRICE PREDICTION SYSTEM

Layer	Technology
Programming language	Python
Web framework	Django
Frontend	HTML, CSS, JavaScript, Bootstrap
Data processing	Pandas, NumPy
Machine learning	Scikit-learn
Data visualization	Matplotlib, Seaborn
Model serialization	Joblib
Database	SQLite
Development tools	PyCharm, VS Code, Jupyter Notebook

VII. IMPLEMENTATION

A. Development Environment

The system was implemented in Python with the technologies listed in Table II. Development was carried out in Jupyter Notebook and a code editor, and the application runs in a standard web browser. A computer with an Intel Core i3 processor, 4 GB of RAM (8 GB for larger datasets) and 100 GB of free disk space was sufficient.

B. Model Training and Serialization

Training is done offline by a stand-alone script. It generates the dataset of Section V, splits it 80/20, fits the scaler-plus-random-forest pipeline, prints the test R-squared and saves the fitted pipeline to a versioned file (house_model_v1.joblib) in a dedicated model folder. Because the scaler is part of the saved pipeline, the web application does not need to repeat any preprocessing code, and replacing the file is enough to deploy a new model.

C. Django Application Structure

The project is divided into four apps: accounts (registration, login, logout, profile), homes (prediction form, result page and the Prediction model), admin_panel (dashboard, user and prediction management, reports, search) and backend (project configuration). A single Prediction table stores the owning user, the submitted features as a JSON field, the predicted price, a model-version tag and a creation time. Keeping the inputs as JSON means that new features can be added without a schema migration, and the version tag allows predictions to be traced to the model that made them.

D. Prediction Workflow

A prediction request proceeds in five steps: the form is validated, the cleaned values are converted into the feature vector expected by the model, the serialized pipeline is loaded with joblib and asked for a price, the result is written to the database linked to the logged-in user, and the user is redirected to a result page. If the model file cannot be loaded, the view does not crash but stores a price of zero; an explicit error message should replace this behavior.

E. Access Control and Administration

After login, superusers are redirected to the administrator dashboard and other users to the

prediction page. Most administrative views are guarded by a decorator that admits superusers only (see Section IX-B for exceptions), and long lists are paginated. The report view filters predictions between two dates, and the search view matches on user name, location, bedrooms and bathrooms.

F. Working System Snapshots

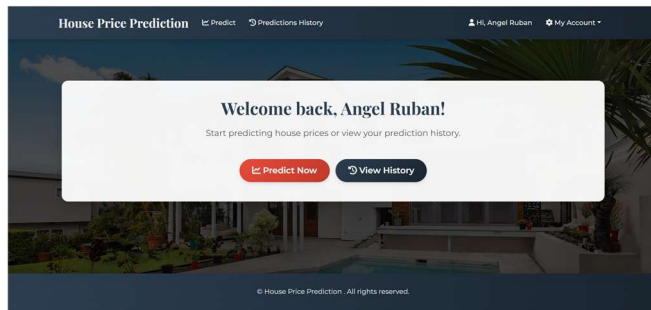


Fig. 2 Home page of the user module.

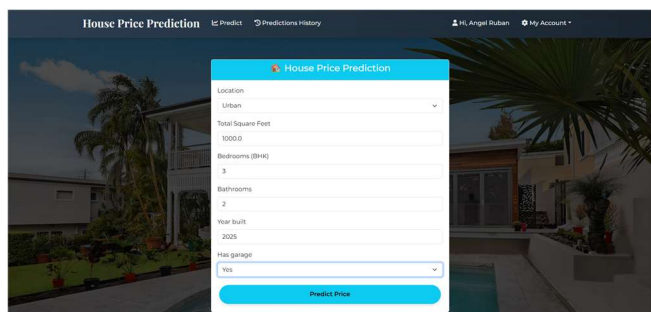


Fig. 3 Prediction form for entering property details.

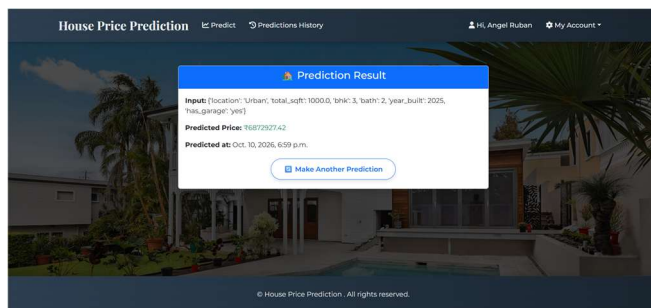


Fig. 4 Prediction result showing the stored inputs, estimated price and time.

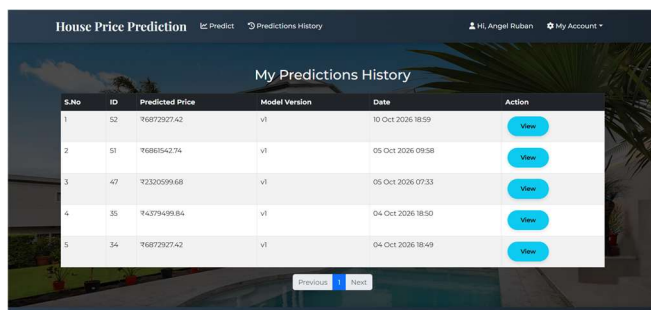


Fig. 5 User's prediction history with model version and date.

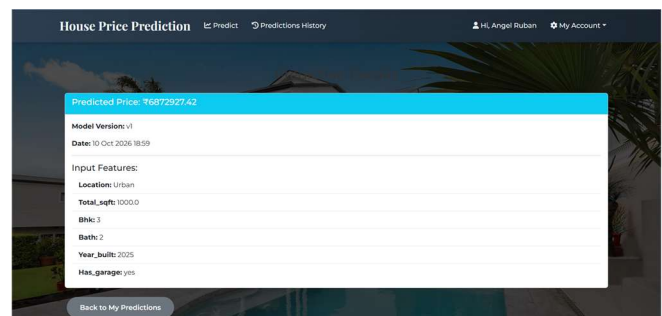


Fig. 6 Prediction details page showing the model version, date and input features.

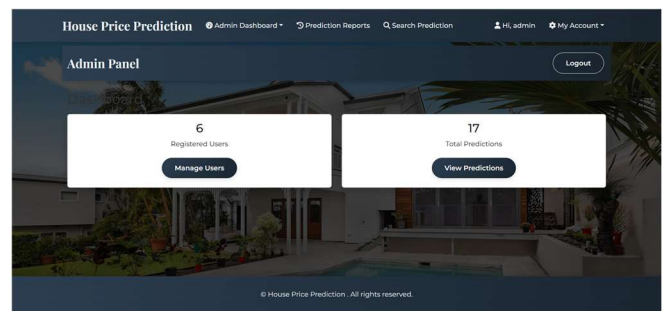


Fig. 7 Administrator dashboard showing registered users and total predictions.

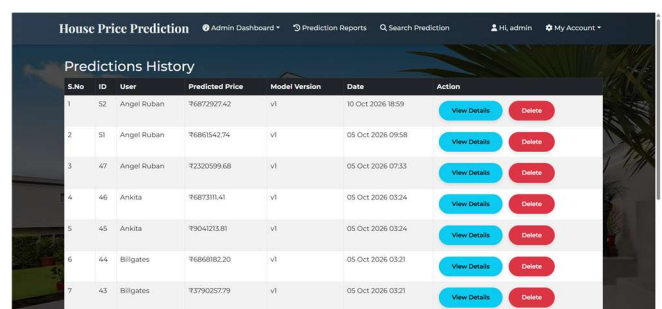


Fig. 8 Administrator view of all predictions.

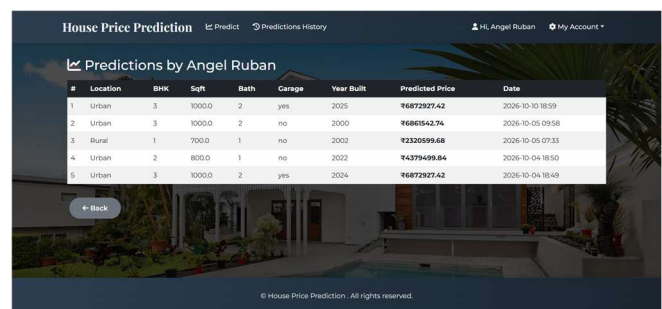


Fig. 9 Prediction details listing the stored inputs and predicted price.

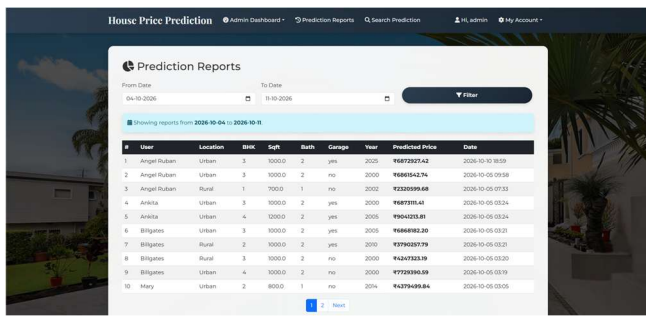


Fig. 10 Administrator report of predictions between two dates.

VIII. RESULTS AND DISCUSSION

A. Predictive Performance

TABLE III
TEST-SET PERFORMANCE OF THE THREE MODELS

Model	MAE	RMSE	R ²	CV R ²
Linear Regression	1.946	2.511	0.930	0.941 ± 0.005
Decision Tree	0.251	0.640	0.995	0.998 ± 0.001
Random Forest	0.231	0.518	0.997	0.999 ± 0.001

MAE and RMSE in million rupees, computed on the 400 held-out records. CV R²: mean ± standard deviation over five folds.

Table III summarizes the results. The random forest achieved the best scores on every measure, with an MAE of 0.231 million rupees, an RMSE of 0.518 million and an R-squared of 0.997. Its median relative error was 0.71 percent. The single decision tree came close (R-squared 0.995), but the forest lowered its RMSE by about 19 percent and its MAE by about 8 percent, which is consistent with the variance-reduction argument for ensembles [7]. Linear regression was clearly behind, with an RMSE nearly five times larger than the forest. Cross-validated scores agree with the test split and show small standard deviations, so the ranking does not depend on the particular partition.

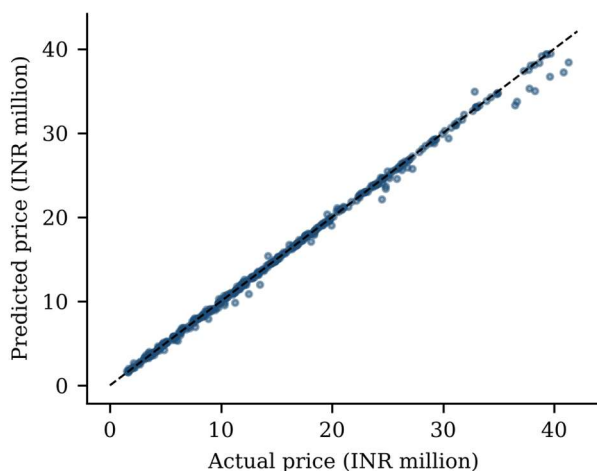


Fig. 11 Predicted versus actual prices for the random forest on the test set. The dashed line marks perfect prediction.

Fig. 11 shows that predictions lie tightly along the diagonal across the whole price range. The weaker performance of the linear model is explained by the data-generating rule in (1): price depends on the product of area and a bedroom- and location-dependent rate, an interaction that a purely additive model cannot express but trees can approximate by successive splits.

To see how much data the forest needs, it was refitted on the first 10, 25, 50, 75 and 100 percent of the training set. The test R-squared rose from 0.965 with 160 records to 0.987, 0.995, 0.997 and 0.997 respectively, so most of the benefit is obtained from a few hundred records on this problem.

B. Feature Relevance

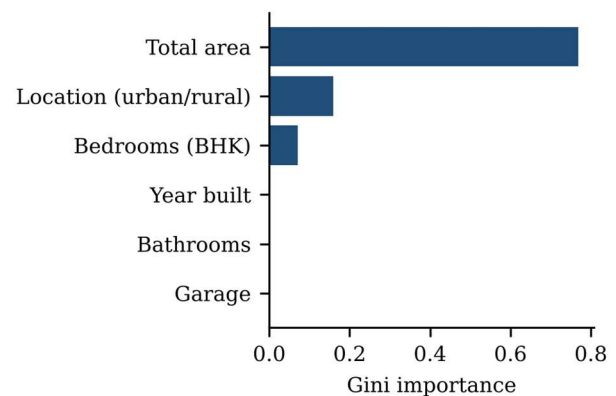


Fig. 12 Impurity-based feature importance of the random forest.

Fig. 12 gives the forest's impurity-based importances. Area accounts for 76.9 percent, location for 15.9 percent and bedrooms for 7.1 percent; year built, bathrooms and garage together contribute about 0.1 percent. Because impurity importance can be biased, an ablation was also run in which the forest was retrained with one feature removed (Table IV).

TABLE IV
TEST R² AFTER REMOVING ONE FEATURE

Feature removed	Test R ²
None (full model)	0.997
Area	-0.064
Location	0.779
Bedrooms (BHK)	0.944
Bathrooms	0.997
Year built	0.998
Garage	0.997

Random forest, same split and settings as Table III.

Removing area destroys the model (R-squared falls below zero), removing location lowers it to 0.779, and removing the bedroom count lowers it to 0.944. Dropping bathrooms, year built or garage leaves the score essentially unchanged. These outcomes agree with the generating rule: bathrooms are not part of it, and the age and garage terms are small next to a price that averages more than 15 million rupees. The forest therefore recovers the structure of the data, which is what one wants from a model before it meets real data. It also warns that in real data a feature with a small coefficient may appear unimportant simply because of scale.

IX. OBSERVATIONS

A. Overall Performance Evaluation

Across the 400 held-out records the random forest was the most accurate of the three models, with a test R-squared of 0.997, a mean absolute error of about 0.23 million rupees (roughly 1.5 percent of the average price) and a median relative error of 0.71 percent. The decision tree was a close second and linear regression a distant third. The cross-validated scores confirm that this ranking is stable. At the system level, the project's unit, integration and system tests found the core workflow, from registration and login to prediction, history and the administrator reports, working as intended; however, no formal user study or concurrent-load test was carried out, so claims about usability and response time under load are not made. These results are obtained on synthetic data and demonstrate that the pipeline learns the structure it is given rather than real-market accuracy.

B. Limitations of the Proposed System

Synthetic data. This is the most important limitation. Because prices follow a known formula with small noise, near-perfect scores are expected for flexible models and should not be compared with results on real sales, where unobserved factors such as condition, view, school quality and negotiation leave much larger residual error.

Limited features and coarse location. Location is a single urban/rural flag. Real valuation depends on neighborhood, proximity to services and temporal market conditions, none of which are represented. The split was random rather than chronological, so

the experiment does not test robustness to price drift. The system is also restricted to residential properties.

Scope of the evaluation. Only one dataset was used, results are from a single train/test split plus cross-validation without statistical significance tests, and the models were not compared with published baselines on a public benchmark. *No tuning or uncertainty estimates.* Default hyperparameters were used, and the system reports a single point estimate without a prediction interval, which may convey more certainty than is justified. Users should treat the output as guidance rather than an appraisal.

Operational concerns. The model is stored as a file and is loaded on demand for each request; monitoring for drift and scheduled retraining are not yet in place. In the current code the user-detail, user-deletion and per-user history views do not yet carry the superuser check, and deletions are triggered by GET requests; they should be restricted to administrators and moved to POST requests with CSRF protection. SQLite is suitable only for development.

C. Advantages of the Proposed System

- *Accuracy on the studied problem:* the random forest explains 99.7 percent of the price variance on unseen data and recovers the factors that actually determine price.
- *Ease of use:* a non-technical user obtains an estimate from one short form, and earlier estimates remain available in a personal history.
- *Auditability:* each prediction is stored with its inputs, output, model version and timestamp, so results can be traced and reproduced.
- *Administrative control:* administrators can monitor users and predictions, search records and generate reports for any date range.
- *Modularity and reproducibility:* training is separate from serving, the whole experiment is seeded, and the model can be replaced without changing the web application.
- *Low cost:* the stack uses only open-source components and runs on an ordinary computer.

D. Scalability Analysis

A small benchmark was run to examine how the random forest pipeline behaves as the amount of data grows (Table V). The measurements were taken on a two-core cloud container with default settings and

are indicative only; they cover the model, not the Django application or the database, which were not load tested.

TABLE V
SCALING OF THE RANDOM FOREST PIPELINE

Training records	Training time (s)	Model file (MB)
2,000	0.4	18
10,000	1.9	91
50,000	10.2	455
100,000	22.0	910

Single-threaded training of 100 fully grown trees.

Training time grows roughly linearly with the number of records (0.4 s for 2,000 records and about 22 s for 100,000), which suggests that retraining remains practical for far larger datasets. Prediction is cheap: a single prediction took about 5.8 ms, and batch prediction ran at roughly 89 thousand records per second. The main concern is memory and storage. Because the trees are fully grown, the serialized model also grows linearly, from about 18 MB to about 910 MB, and loading the 2,000-record model from disk took about 27 ms. Since the present application loads the model for each request, a larger model would add noticeable delay to every prediction. Three measures would address this: loading the model once at server start-up, limiting tree depth or leaf size to bound model size, and moving from SQLite to a server database with a production web server for concurrent users. Feature growth is handled gracefully because inputs are stored as JSON, but adding regions or property types would require retraining on correspondingly richer data.

X. CONCLUSION AND FUTURE WORK

This research project developed a house price prediction system in which a random forest regression pipeline is served through a Django application with user and administrator modules. On a controlled dataset, the forest outperformed a decision tree and a linear model, reaching a test R-squared of 0.997 and a median relative error of under one percent, and importance and ablation analyses confirmed that it relied on the attributes that actually determine price in the data. A scalability test showed linear growth in training time and model size. The main outcome is a documented, reproducible system

and evaluation method; the accuracy figures themselves apply only to the synthetic setting.

Future work will concentrate on validation with real data, such as the Ames dataset [6] or local listing and registry records; richer location features; gradient-boosted models [8] and hyperparameter search; prediction intervals and per-prediction explanations for users; model caching and depth limits; an administrator interface for retraining and versioning models; and deployment with a production database and monitoring.

ACKNOWLEDGMENT

The authors thank Dr. Manoj Singh and the Department of Computer Science, SIES College of Arts, Science and Commerce, for guidance and support.

REFERENCES

- [1] S. Rosen, "Hedonic prices and implicit markets: Product differentiation in pure competition," *J. Polit. Econ.*, vol. 82, no. 1, pp. 34–55, 1974.
- [2] D. Harrison, Jr. and D. L. Rubinfeld, "Hedonic housing prices and the demand for clean air," *J. Environ. Econ. Manage.*, vol. 5, no. 1, pp. 81–102, 1978.
- [3] S. Mullainathan and J. Spiess, "Machine learning: An applied econometrics approach," *J. Econ. Perspect.*, vol. 31, no. 2, pp. 87–106, 2017.
- [4] V. Limsombunchai, "House price prediction: Hedonic price model vs. artificial neural network," in *Proc. New Zealand Agricultural and Resource Economics Society Conf.*, Blenheim, New Zealand, 2004.
- [5] B. Park and J. K. Bae, "Using machine learning algorithms for housing price prediction: The case of Fairfax County, Virginia housing data," *Expert Syst. Appl.*, vol. 42, no. 6, pp. 2928–2934, 2015.
- [6] D. De Cock, "Ames, Iowa: Alternative to the Boston housing data as an end of semester regression project," *J. Stat. Educ.*, vol. 19, no. 3, 2011.
- [7] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, 2001.
- [8] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, San Francisco, CA, USA, 2016, pp. 785–794.
- [9] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, 2011.
- [10] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, 2nd ed. New York, NY, USA: Springer, 2009.
- [11] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees*. Belmont, CA, USA: Wadsworth, 1984.
- [12] C. J. Willmott and K. Matsuura, "Advantages of the mean absolute error (MAE) over the root mean square error (RMSE) in assessing average model performance," *Climate Res.*, vol. 30, no. 1, pp. 79–82, 2005.
- [13] T. Chai and R. R. Draxler, "Root mean square error (RMSE) or mean absolute error (MAE)? – Arguments against avoiding RMSE in the literature," *Geosci. Model Dev.*, vol. 7, no. 3, pp. 1247–1250, 2014.
- [14] Django Software Foundation, "Django documentation." [Online]. Available: <https://docs.djangoproject.com/>