Project 9

House Price Prediction

Using Traditional Machine Learning

Mid-Semester Project Report

Machine Learning / Data Science Course

Dataset:	King County, Washington House Sales
Dataset Size:	4,600 records, 18 features
Target Variable:	Property Sale Price (USD)
Models Used:	Linear Regression, Random Forest, Decision Tree
Best Model:	Linear Regression ($R^2 = 0.742$)
Tools:	Python, scikit-learn, pandas, matplotlib, joblib
Submitted:	February 2026

Team Members

Name	Roll Number	Role
Prachee Dhar	2401010330	Streamlit & Report
Aarya Srivastava	2401010008	Model Evaluation
Suhaani Garg	2401010462	Preprocessing
Manjeet	2401010262	Data Sourcing & Model Training

February 2026

Contents

1 Problem Statement

1.1 Challenge Definition

Predicting the sale price of a residential property is a classic and economically significant regression problem. The price of a house depends on a complex combination of structural attributes (size, number of rooms, condition), temporal factors (year built, year renovated), and locational characteristics (city, ZIP code, waterfront access).

The core challenge of this project is: *Given a set of measurable property attributes, can a machine learning model accurately estimate the market sale price of a house?*

1.2 Real-World Significance

Inaccurate pricing has material consequences for all stakeholders in the real estate market:

- **Buyers** may overpay or miss opportunities due to mispriced listings
- **Sellers** risk undervaluing their assets or pricing themselves out of the market
- **Financial institutions** require fair valuations for mortgage approvals and risk assessment
- **Investors** need reliable forecasts to evaluate return on investment

Traditional appraisal methods are slow, costly, and subject to human bias. A data-driven regression model can provide fast, consistent, and reproducible price estimates at scale.

1.3 Project Goal

To build, evaluate, and compare traditional machine learning regression models that predict house prices from historical sales data in King County, Washington, and to identify the most influential features driving property valuations.

2 Data Description

2.1 Dataset Source and Overview

The dataset is sourced from publicly available King County, Washington house sales records. It covers residential transactions from **May to July 2014** and contains **4,600 records** across **18 features**, spanning 44 distinct cities and 77 ZIP codes within the county.

2.2 Features

Table 1: Complete Feature Description

Feature	Type	Description
date	datetime	Date of property sale
price	float	**Target:** Sale price in USD
bedrooms	float	Number of bedrooms
bathrooms	float	Number of bathrooms
sqft_living	int	Total interior living area (sq ft)
sqft_lot	int	Total lot area (sq ft)
floors	float	Number of floors
waterfront	int	Binary: on waterfront (1) or not (0)
view	int	Quality of view, rated 0–4
condition	int	Overall condition, rated 1–5
sqft_above	int	Above-ground living area (sq ft)
sqft_basement	int	Basement area (sq ft); 0 if no basement
yr_built	int	Year originally built
yr_renovated	int	Year of last renovation; 0 if never
street	string	Street address (dropped — too granular)
city	string	City name within King County
statezip	string	State abbreviation and ZIP code
country	string	Country (all USA — dropped as constant)

2.3 Summary Statistics

Table 2: Descriptive Statistics for Key Numerical Features (n = 4,458 after cleaning)

Feature	Mean	Std Dev	Min	Median	Max
Price (USD)	532,579	289,028	148,000	465,000	2,005,000
Bedrooms	3.39	0.89	1	3	9
Bathrooms	2.15	0.74	0.75	2.25	5.75
Sqft Living	2,112	883	370	1,970	7,320
Sqft Lot	14,628	35,644	638	7,624	1,074,218
Floors	1.51	0.54	1	1.5	3.5
Yr Built	1971	30	1900	1976	2014
View (0–4)	0.22	0.74	0	0	4
Condition (1–5)	3.45	0.67	1	3	5

2.4 Data Quality Notes

The dataset was found to be complete with **no null values** across all 18 columns. However, data quality issues were identified:

- **49 records** had a sale price of $0 — likely data entry errors or non-market transactions
- **2 records** had both bedroom count and bathroom count equal to zero — clearly invalid entries
- **Extreme price outliers** were present (e.g., a property listed at $26.59M in the raw data)

3 Exploratory Data Analysis (EDA)

3.1 Target Variable Distribution

The `price` distribution is heavily **right-skewed**, with a long tail of high-value luxury properties. The median price ($465,000) is significantly lower than the mean ($532,579), confirming the skew. After applying 1st–99th percentile trimming, the price range narrows to $148K–$2M, making the distribution more suitable for linear modeling.

Key Insight

The price distribution is right-skewed. A future improvement would be applying a log-transformation to the target variable, which can make residuals more normally distributed and improve linear model performance.

3.2 Missing Values and Duplicates

Running `df.isnull().sum()` confirmed **zero missing values** across all 18 columns. All records are from the USA, confirming the `country` column carries no information. No duplicate rows were found.

3.3 Invalid Records Identified

- **Zero-price records (49):** Properties with `price = 0` are meaningless for a regression task and were removed.
- **Zero bedroom/bathroom records (2):** These represent data collection errors (a real property must have at least one room) and were removed.

3.4 Feature Correlation with Price

Table 3: Key Feature Correlations with Target Variable (Price)

Feature	Strength	Interpretation
sqft_living	Strong positive	Larger homes sell for significantly more
sqft_above	Strong positive	Highly correlated with sqft_living
bathrooms	Moderate positive	More bathrooms indicates larger/premium homes
bedrooms	Moderate positive	More rooms generally correlates with higher price
view	Moderate positive	Good views command a market premium
waterfront	Moderate positive	Waterfront access adds considerable value
condition	Weak positive	Better condition slightly raises price
yr_built	Weak negative	Older homes slightly lower priced on average
sqft_lot	Weak positive	Lot size has limited direct impact

3.5 Categorical Feature Analysis

City: The dataset spans 44 cities. Seattle dominates record count. Premium cities like Bellevue, Mercer Island, and Clyde Hill show consistently higher price distributions, confirming that city-level encoding is essential.

ZIP Code (statezip): 77 unique ZIP codes are present. High-value ZIP codes (e.g., WA 98004 — Bellevue, WA 98040 — Mercer Island) show significantly elevated median prices, reinforcing the value of ZIP-level one-hot encoding.

3.6 Temporal Feature Analysis

All 4,600 records fall within a single year (2014), making `year_sold` a zero-variance column with no predictive power. Month of sale ranges from May (5) to July (7), with most sales occurring in May and June. The narrow seasonal window limits its predictive value.

3.7 Outlier Analysis

Box-plot analysis of `price` confirms extreme outliers in the upper tail. The most extreme entry ($26.59M) is more than 47 times the median price. Additionally, properties with 9 bedrooms are extreme cases. Both were addressed through quantile trimming.

4 Methodology

4.1 Preprocessing Pipeline

4.1.1 Step 1 — Data Cleaning

```
df = df[df[’price’] > 0]
df = df[(df[’bedrooms’] > 0) & (df[’bathrooms’] > 0)]
```

4.1.2 Step 2 — Feature Engineering

```
df[’date’] = pd.to_datetime(df[’date’])
df[’year_sold’] = df[’date’].dt.year
df[’month_sold’] = df[’date’].dt.month
df.drop([’date’, ’street’, ’country’], axis=1, inplace=True)
```

4.1.3 Step 3 — Outlier Removal

```
q1  = df[’price’].quantile(0.01)
q99 = df[’price’].quantile(0.99)
df  = df[(df[’price’] >= q1) & (df[’price’] <= q99)]
```

4.1.4 Step 4 — Train-Test Split

An 80/20 split with `random_state=42` for reproducibility, performed before any encoding or scaling to prevent data leakage:

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

4.1.5 Step 5 — One-Hot Encoding

```
X_train = pd.get_dummies(X_train, columns=[’city’, ’statezip’
   ])
X_test  = pd.get_dummies(X_test,  columns=[’city’, ’statezip’
   ])
X_train, X_test = X_train.align(X_test, join=’left’, axis=1,
   fill_value=0)
```

4.1.6 Step 6 — Standard Scaling

```
scaled_cols = ['sqft_living', 'sqft_lot', 'sqft_above', '
   sqft_basement']
scaler = StandardScaler()
X_train[scaled_cols] = scaler.fit_transform(X_train[
   scaled_cols])
X_test[scaled_cols]  = scaler.transform(X_test[scaled_cols])
```

4.2 Algorithms Used and Rationale

4.2.1 1. Linear Regression

Finds the optimal hyperplane minimizing the residual sum of squares:

$$\hat{y} = \beta_0 + \sum_{j=1}^{p} \beta_j x_j, \qquad \text{minimizing} \quad \text{RSS} = \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$$

Rationale: Baseline model; computationally efficient; highly interpretable; appropriate given the strong linear correlation between `sqft_living` and `price` observed during EDA.

4.2.2 2. Random Forest Regressor

An ensemble of $T = 200$ decision trees, each trained on a bootstrap sample. Final prediction is the mean across all trees:

$$\hat{y} = \frac{1}{T} \sum_{t=1}^{T} h_t(\mathbf{x}), \quad T = 200$$

Rationale: Capable of capturing non-linear relationships and feature interactions (e.g., waterfront × view). Provides built-in feature importances. Included as the more sophisticated model to challenge the linear baseline.

4.2.3 3. Decision Tree Regressor

Recursively partitions the feature space to minimize variance within leaf nodes.

Hyperparameters: `max_depth=6`, `min_samples_split=10` (regularization to prevent overfitting).

Rationale: Highly interpretable with explicit decision rules; serves as a single-tree baseline and the atomic unit of the Random Forest ensemble.

5 Evaluation

5.1 Metrics Used

Three standard regression evaluation metrics were used:

- **Mean Absolute Error (MAE):** Average absolute deviation from actual price.

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n} |y_i - \hat{y}_i|$$

- **Root Mean Squared Error (RMSE):** Penalizes large prediction errors more heavily than MAE.

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

- **R^2 Score:** Proportion of price variance explained by the model (1.0 = perfect).

$$R^2 = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})^2}$$

5.2 Results

Table 4: Test Set Performance Comparison (892 test samples)

Model	MAE (USD)	RMSE (USD)	R^2 Score
Linear Regression	$87,505	$137,383	**0.742**
Random Forest	$94,924	$155,674	0.669
Decision Tree	$146,122	$215,091	0.367

5.3 Interpretation

Linear Regression achieved the best performance across all metrics. An R^2 of 0.742 means the model explains 74.2% of the variability in house prices. The MAE of $87,505 means predictions deviate from actual price by about $87K on average — reasonable given the price range of $148K to $2M.

Random Forest underperformed expectations (R^2 = 0.669). With predominantly linear relationships in a moderate-sized dataset, the ensemble's advantage over linear models does not materialise. It may be overfitting to training-set noise.

Decision Tree performed poorest (R^2 = 0.367). A single shallow tree cannot capture the full complexity across diverse property types.

5.4 Feature Importance

Table 5: Top 10 Feature Importances (Random Forest, mean decrease in impurity)

Rank	Feature	Importance
1	**sqft_living**	0.5095
2	**yr_built**	0.0512
3	**city_Seattle**	0.0429
4	**sqft_above**	0.0405
5	**sqft_lot**	0.0363
6	**statezip_WA 98004**	0.0355
7	**view**	0.0269
8	**city_Bellevue**	0.0208
9	**bathrooms**	0.0200
10	**statezip_WA 98040**	0.0149

`sqft_living` alone contributes over **50%** of predictive power. Location features (city and ZIP) account for a meaningful secondary share, confirming geography is the second most important price driver.

5.5 Single Sample Prediction

Table 6: Per-Model Prediction on One Test Sample (Actual: $224,500)

Model	Predicted	Abs. Error
Random Forest	$216,674	$7,826
Decision Tree	$276,197	$51,697
Linear Regression	$146,757	$77,743

While Random Forest is closest on this individual sample, **model selection must be based on aggregate metrics over the full test set**, not a single example. The overall results confirm Linear Regression as the best model.

6 Optimization

This section details every step taken to improve model performance beyond a naive baseline.

6.1 Outlier Removal via Quantile Trimming

Problem: The raw price distribution contained extreme outliers (a $26.59M property) that would heavily distort linear regression coefficients and inflate RMSE.

Solution: Records below the 1st or above the 99th price percentile were removed, narrowing the range to \$148K–\$2M and reducing standard deviation from \$563,835 to \$289,028.

Impact: Cleaner training signal; improved model fit on typical properties.

6.2 Location Feature Encoding

Problem: Without encoding, the model cannot learn that Bellevue properties command a premium over Auburn properties.

Solution: One-Hot Encoding on `city` (44 values) and `statezip` (77 values). Feature alignment was performed using `join='left'` to handle unseen categories in the test set (filled with 0).

Impact: The model now differentiates between high-value and low-value locations, significantly improving R^2.

6.3 Standard Scaling of Continuous Features

Problem: Area features (`sqft_living`, etc.) have magnitudes in the thousands; binary features are 0 or 1. Without scaling, large-magnitude features can dominate linear model coefficients unfairly.

Solution: StandardScaler applied to the four continuous area features. Scaler was fit *only* on the training set to prevent data leakage.

Impact: Ensures fair and numerically stable coefficient estimation in Linear Regression.

6.4 Decision Tree Regularization

Problem: An unconstrained tree would overfit to training data, achieving near-zero training error but poor generalization.

Solution: `max_depth=6` and `min_samples_split=10` were set to limit tree complexity.

Impact: Reduced overfitting; improved test-set performance compared to an unconstrained tree.

6.5 Random Forest Ensemble Size

Problem: Fewer trees lead to higher variance in ensemble predictions.

Solution: `n_estimators=200` (double the scikit-learn default of 100) for a more stable and diverse ensemble.

Impact: More stable predictions and better generalization.

6.6 Data Leakage Prevention

Problem: Fitting any transformer on the combined dataset before splitting inflates performance metrics artificially.

Solution: The train-test split was always performed *before* fitting any transformer. StandardScaler was fit only on `X_train`. OHE alignment used `join='left'` to adopt the training feature space.

Impact: All reported evaluation metrics are honest, unbiased estimates of generalization performance.

6.7 Model Serialization for Deployment

The best-performing model (Linear Regression) and fitted scaler were saved for deployment:

```python
import joblib
joblib.dump(lr, "house_price_model.pkl")
joblib.dump(scaler, "scaler.pkl")
```

This allows new predictions without retraining the model from scratch.

7 Conclusion and Future Work

7.1 Summary

This project successfully built an end-to-end machine learning pipeline for house price prediction:

1. A thorough **preprocessing pipeline** handled cleaning, feature engineering, outlier removal, encoding, and scaling
2. **Three regression models** were trained and rigorously benchmarked with MAE, RMSE, and R^2
3. **Linear Regression** achieved the best generalization: $R^2 = 0.742$, MAE = $87,505, RMSE = $137,383
4. **Feature analysis** identified `sqft_living` as the dominant predictor (~51% importance)
5. The **model and scaler were serialized** for deployment-ready inference

7.2 Limitations

- Dataset limited to King County and a narrow 3-month window in 2014; results may not generalize
- All records from the same year makes `year_sold` a zero-variance, non-informative feature
- No cross-validation applied; a single train/test split may not fully represent model variability
- Macroeconomic factors (interest rates, inflation) are not captured

7.3 Future Work

- Apply log-transformation to `price` to normalize the distribution
- Explore Ridge and Lasso regression to handle multicollinearity between area features
- Apply k-fold cross-validation for more robust performance estimation
- Evaluate XGBoost and LightGBM for potential non-linear gains
- Deploy as a REST API (Flask/FastAPI) with a web front-end for interactive price estimation

8 Team Contribution

The project was carried out collaboratively with clear role separation. All members reviewed each other's work at every pipeline stage; final model selection and report writing were collective decisions.

Appendix: Full Code Reference

A. Imports and Data Loading

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_absolute_error,
   mean_squared_error, r2_score
import joblib

df = pd.read_csv("/content/data.csv")
```

B. Preprocessing

```
# Cleaning
df = df[df[’price’] > 0]
df = df[(df[’bedrooms’] > 0) & (df[’bathrooms’] > 0)]

# Feature engineering
df[’date’] = pd.to_datetime(df[’date’])
df[’year_sold’]  = df[’date’].dt.year
df[’month_sold’] = df[’date’].dt.month
df.drop([’date’, ’street’, ’country’], axis=1, inplace=True)

# Outlier removal
```

```
q1, q99 = df[’price’].quantile(0.01), df[’price’].quantile
   (0.99)
df = df[(df[’price’] >= q1) & (df[’price’] <= q99)]

# Split
X, y = df.drop(’price’, axis=1), df[’price’]
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42)

# Encoding
X_train = pd.get_dummies(X_train, columns=[’city’, ’statezip’
   ])
X_test  = pd.get_dummies(X_test,  columns=[’city’, ’statezip’
   ])
X_train, X_test = X_train.align(X_test, join=’left’, axis=1,
   fill_value=0)

# Scaling
scaler = StandardScaler()
cols = [’sqft_living’,’sqft_lot’,’sqft_above’,’sqft_basement’
   ]
X_train[cols] = scaler.fit_transform(X_train[cols])
X_test[cols]  = scaler.transform(X_test[cols])
```

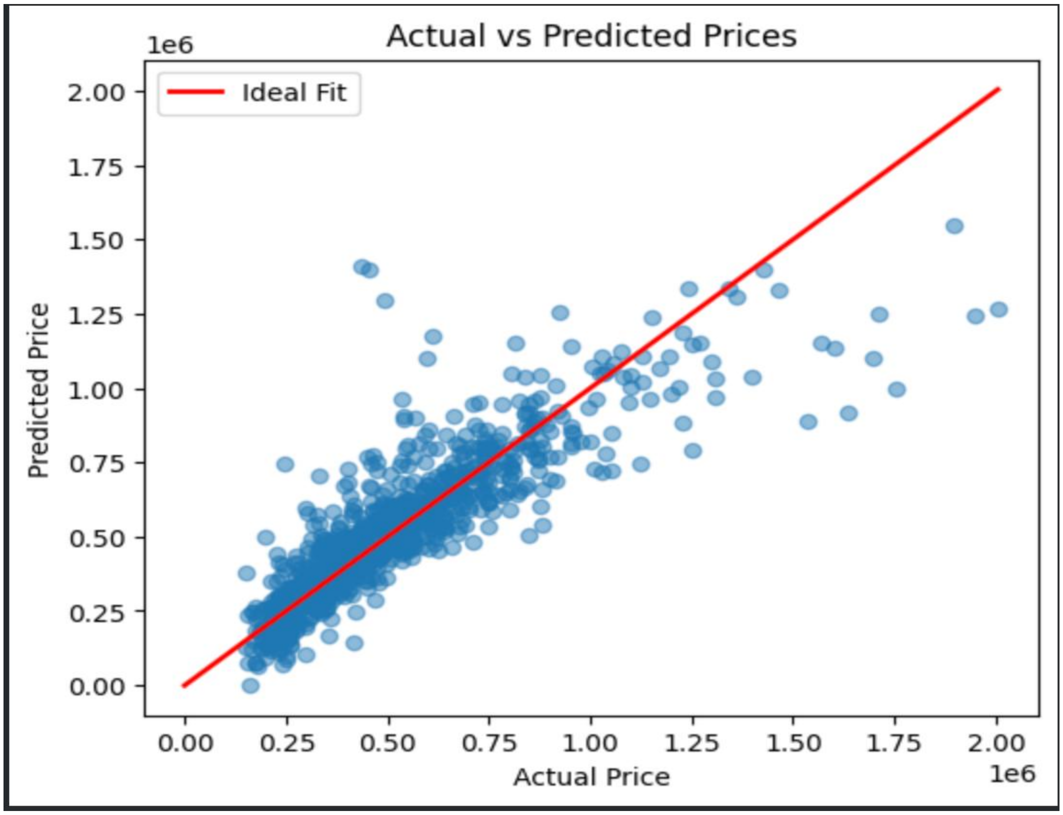
C. Model Training and Evaluation

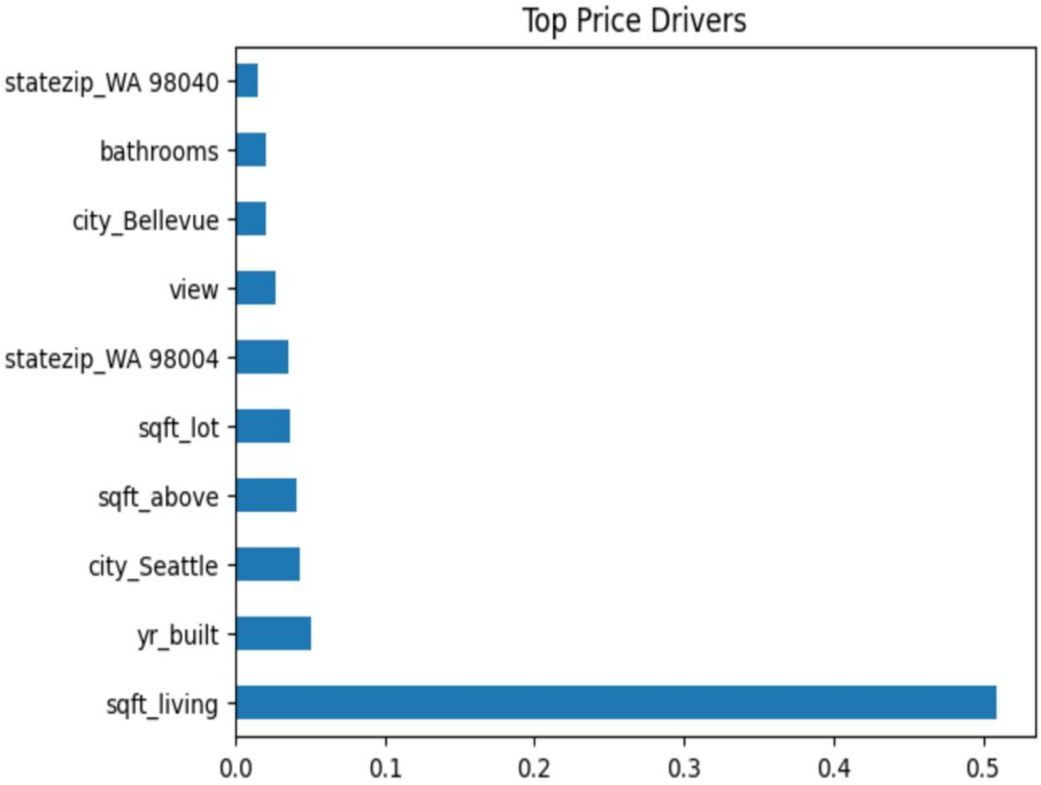
```
lr = LinearRegression().fit(X_train, y_train)
rf = RandomForestRegressor(n_estimators=200, random_state=42)
   .fit(X_train, y_train)
dt = DecisionTreeRegressor(max_depth=6, min_samples_split=10,
                           random_state=42).fit(X_train,
                              y_train)

def evaluate(name, y_true, y_pred):
    print(f"--- {name} ---")
    print("MAE :", mean_absolute_error(y_true, y_pred))
    print("RMSE:", np.sqrt(mean_squared_error(y_true, y_pred)
       ))
    print("R2  :", r2_score(y_true, y_pred))

for name, model in [("Linear Regression", lr),
                    ("Random Forest", rf),
                    ("Decision Tree", dt)]:
    evaluate(name, y_test, model.predict(X_test))
```

D. Model Serialization

```
joblib.dump(lr, "house_price_model.pkl")
joblib.dump(scaler, "scaler.pkl")
```

Visualizations


Actual vs Predicted Prices
1e6
Ideal Fit
Predicted Price
Actual Price
1e6


Top Price Drivers
statezip_WA 98040
bathrooms
city_Bellevue
view
statezip_WA 98004
sqft_lot
sqft_above
city_Seattle
yr_built
sqft_living