End-to-End ML Project
– California Housing Price Prediction

Gauranga Kumar Baishya

August 27, 2025

Outline

Project Goal & The Data Pipeline

Primary Objective

To build a model that predicts the median housing price in California districts using 1990 census data.

Business Context

The model's output will be a crucial **signal** for a downstream investment analysis system. The accuracy of this prediction directly impacts business revenue.

Data Pipeline

This project involves creating a full data pipeline: a sequence of data processing components, from raw data to the final actionable insight.

Framing the Problem

1. Learning Type

Supervised Learning: The dataset includes the target labels (median housing prices).

2. Task Type

Regression Task: We are predicting a continuous numerical value. Specifically, it is a **multivariate regression** problem as we use multiple features.

3. Learning Method

Batch Learning: The dataset is small enough to be handled in memory, and there is no need for real-time adaptation.

Selecting a Performance Measure

Root Mean Square Error (RMSE)

The most common metric for regression tasks. It measures the standard deviation of the prediction errors and gives a higher weight to large errors.

$$\mathrm{RMSE}(\mathbf{X}, h) = \sqrt{\frac{1}{m}\sum_{i=1}^{m}\left(h(\mathbf{x}^{(i)}) - y^{(i)}\right)^2}$$

RMSE is sensitive to outliers because of the squaring operation (l_2 norm).

Mean Absolute Error (MAE)

An alternative metric; MAE is less sensitive to outliers..

$$\mathrm{MAE}(\mathbf{X}, h) = \frac{1}{m}\sum_{i=1}^{m}\left|h(\mathbf{x}^{(i)}) - y^{(i)}\right|$$

How does the data look like ?

Summary Statistics

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
count	20640.000000	20640.000000	20640.000000	20640.000000	20433.000000	20640.000000	20640.000000	20640.000000	20640.000000
mean	-119.569704	35.631861	28.639486	2635.763081	537.870553	1425.476744	499.539680	3.870671	206855.816909
std	2.003532	2.135952	12.585558	2181.615252	421.385070	1132.462122	382.329753	1.899822	115395.615874
min	-124.350000	32.540000	1.000000	2.000000	1.000000	3.000000	1.000000	0.499900	14999.000000
25%	-121.800000	33.930000	18.000000	1447.750000	296.000000	787.000000	280.000000	2.563400	119600.000000
50%	-118.490000	34.260000	29.000000	2127.000000	435.000000	1166.000000	409.000000	3.534800	179700.000000
75%	-118.010000	37.710000	37.000000	3148.000000	647.000000	1725.000000	605.000000	4.743250	264725.000000
max	-114.310000	41.950000	52.000000	39320.000000	6445.000000	35682.000000	6082.000000	15.000100	500001.000000

How does the data look like ?

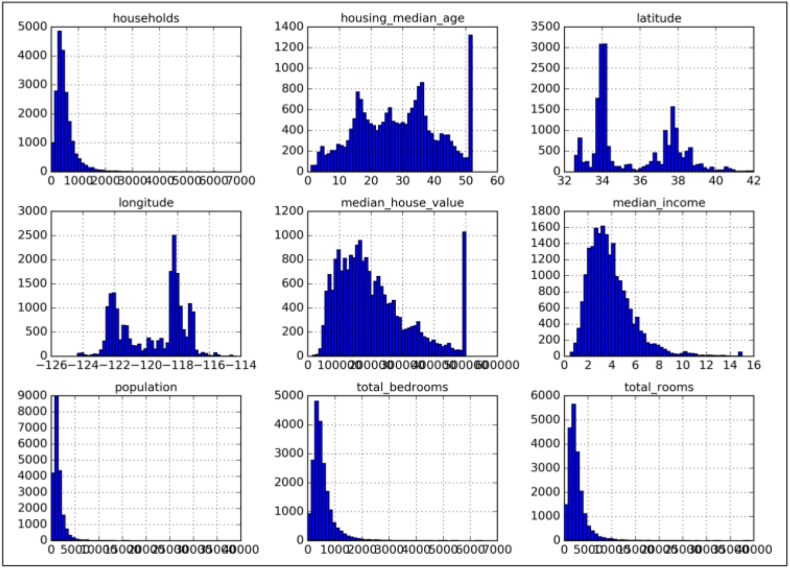


Figure 2-8. A histogram for each numerical attribute

Initial Data Exploration with Pandas

Key Commands

- : Reveals dataset size (e.g., 20,640 entries), data types, and missing values.
- : Shows that ocean_proximity is a categorical feature with 5 distinct categories.
- : Provides a statistical summary of all numerical attributes.
- : Plots histograms to visualize the distribution of each numerical feature.

Key Findings from Histograms

1. **Capped Data**: `housing_median_age` and `median_house_value` are capped. The cap on the target variable at $500,000 is a significant issue.
2. **Scaled Income**: `median_income` is not in USD but is scaled and capped at 15.
3. **Varying Scales**: Attributes have vastly different scales, indicating a need for **feature scaling**.
4. **Tail-Heavy Distributions**: Many attributes are not normally distributed. They may need transformation (e.g., log transform) to help algorithms detect patterns.

Creating a Test Set

Why is this CRITICAL?

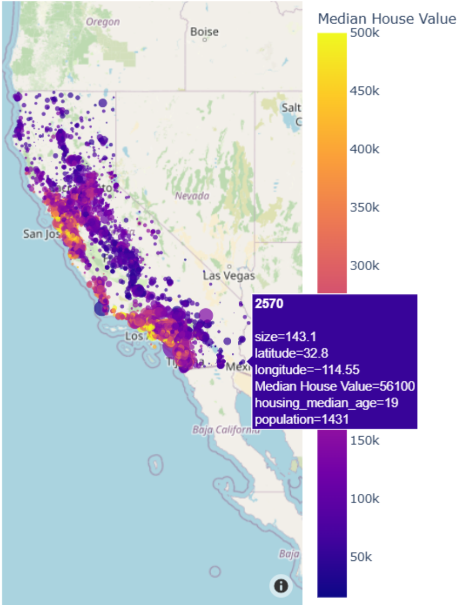
You must create and set aside a test set **before** any deep data exploration. This prevents **data snooping bias**, where you might unconsciously select a model based on patterns you see in the test set, leading to an overly optimistic performance evaluation.

Best Practice: Stratified Sampling

Instead of pure random sampling, **stratified sampling** is used to ensure the test set is representative.

- Since median income is a strong predictor, we stratify based on income categories.
- This creates a test set where the distribution of income categories matches the distribution in the full dataset, avoiding sampling bias.

Geographical Visualization & Correlation



Geographical Visualization & Correlation

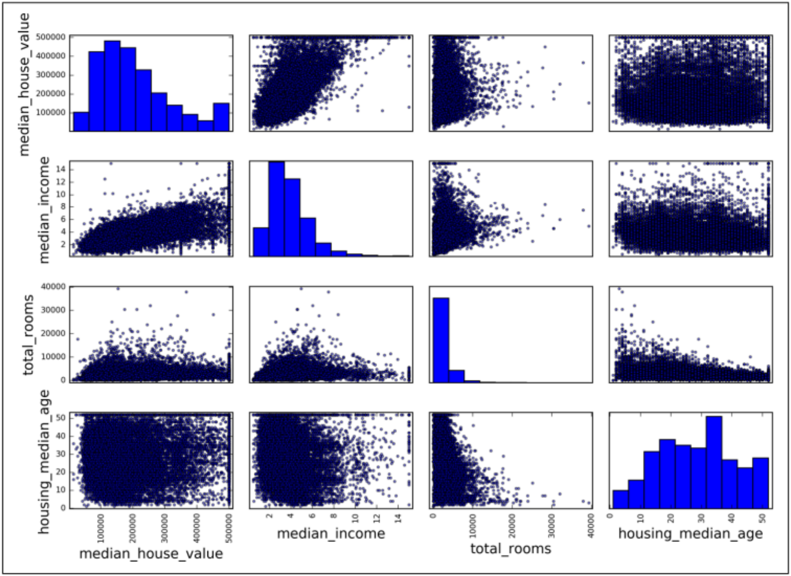


Figure 2-15. Scatter matrix

Geographical Visualization & Correlation

Geographical Plot

A scatterplot of latitude vs. longitude, with point density (‘alpha‘), population size (‘s‘), and price (‘c‘), reveals that prices are highest in dense, coastal areas like the Bay Area and LA.

Correlation Analysis

Using `.corr()`, we find `median_income` has the strongest positive correlation with house value. The `scatter_matrix` confirms this and reveals data quirks, like horizontal lines at price caps.

Attribute Combinations

Creating new features can be very powerful. For example:

- `rooms_per_household`
- `bedrooms_per_room`

These new features proved to be more correlated with house value than the original attributes.

Cleaning, Scaling, and Pipelines

Handling Missing Values

The missing `total_bedrooms` values are handled using Scikit-Learn's `Imputer`, configured to fill them with the column's median.

Handling Categorical Data

The text attribute `ocean_proximity` is converted to numerical data using **One-Hot Encoding**, which creates a binary column for each category. This prevents algorithms from assuming a false ordering.

Feature Scaling

- **Normalization** (`MinMaxScaler`): Scales to a 0-1 range. Sensitive to outliers.
- **Standardization** (`StandardScaler`): Centers to mean 0 with unit variance. More robust to outliers.

Training and Evaluating Models

Model 1: Linear Regression

- A simple baseline model.
- **Result**: High RMSE on the training set, ≈ 68.628.
- **Diagnosis**: The model is **underfitting**. It's too simple to capture the data's complexity.

Model 2: Decision Tree Regressor

- A more powerful, non-linear model.
- **Result**: RMSE of 0 on the training set.
- **Diagnosis**: A classic case of severe **overfitting**. The model has memorized the training data and will not generalize.

Better Evaluation: Cross-Validation

K-Fold Cross-Validation

To get a more robust performance estimate without touching the test set, we use K-fold cross-validation. The training set is split into K folds; the model is trained K times on K-1 folds and evaluated on the remaining fold.

Cross-Validation Results

- **Decision Tree**: The average CV score was worse than Linear Regression, confirming it was overfitting.
- **Random Forest Regressor**: An ensemble model that trains many Decision Trees. It performed much better, with a CV RMSE of ≈ 52.564. This is our most promising model.

Hyperparameter Tuning

Grid Search

Scikit-Learn's `GridSearchCV` automates the process of finding the best hyperparameters.

- You define a "grid" of hyperparameters and values to test.
- It uses cross-validation to evaluate every combination.
- The search improved the Random Forest RMSE to ≈ 49.694.

Analyze Feature Importance

The tuned Random Forest model can report **feature importances**.

- **Most important**: `median_income`.
- **Second most important**: The categorical feature `INLAND`.
- This insight could be used to drop less important features.

Final Evaluation on the Test Set

The Moment of Truth

After all model selection and tuning is complete, the final model is evaluated one last time on the held-out test set.

Final Performance

The final RMSE on the test set was $\approx 47,766$.

Important Rule

You must **not** perform any further tuning based on the test set performance. This result represents the model's expected performance on new, unseen data.

Deployment and Beyond

Launch

Deploy the final model into your production environment, connecting it to live data sources.

Monitor

- Continuously monitor the model's live performance.
- Models can degrade over time as data evolves—a concept known as **"model rot"**.
- Set up alerts to be notified of performance drops.

Maintain

- Automate the process of retraining your model on fresh data.
- Regular retraining ensures the model stays relevant and its performance remains high.

Questions?

Thank You!