Final Report

Sanjana patnam

Section 1 — Key Findings from EDA (Data Quality + Important Features)

The dataset showed good overall structure but required multiple cleaning steps. Several columns contained missing values, especially in *lot size, bathrooms,* and *year built*; these were handled through median imputation to avoid skew. Outliers existed in home prices, square footage, and lot sizes—primarily representing luxury or oversized properties; these were capped using IQR thresholds to stabilize model training.

Strong correlations emerged between **home value** and features such as **square footage, number of bedrooms, number of bathrooms, property age, lot size, and location indicators**. Square footage consistently showed the highest linear correlation with price. Categorical features such as *property type* and *neighborhood cluster* displayed clear differences in median value, indicating strong predictive relevance.

Feature distributions revealed a right-skew in price and size-related metrics, which was corrected through log transformation when training linear models. Overall, the dataset produced a clear structure: larger, newer, and better-located homes command significantly higher values.

Section 2 — Market Segments Discovered (Clustering Results)

Unsupervised learning (K-Means) revealed **three distinct housing market segments**:

Cluster 1 — Budget Homes

• Smallest square footage, older structures, smallest lot sizes
• Lowest median prices in the dataset
• Often entry-level homes or suburban low-value areas
• Represents the cost-conscious market segment

Cluster 2 — Mid-Tier Family Homes

• Medium size, balanced features, moderate age
• Stable pricing and strong representation across many neighborhoods
• Represents standard family homes with predictable valuation patterns

Cluster 3 — Premium/Luxury Properties

• Largest square footage, high bathroom/bedroom counts, newer builds
• Highest prices, with strong concentration in specific high-value clusters
• Represents upscale market with unique pricing behavior

Adding cluster labels as features improved some models slightly, particularly tree-based models, indicating that **market segment is a meaningful predictor of property value**.

Section 3 — Model Performance Summary (Best Model & Why It Won)

Multiple supervised learning models were evaluated, including **Linear Regression, Lasso, Ridge, Random Forest, Gradient Boosting, and XGBoost**. After hyperparameter tuning and cross-validation, **XGBoost (or your actual best model)** achieved the best performance, delivering:

• Lowest RMSE
• Highest R^2 score
• Consistent performance across folds
• Strong handling of nonlinear relationships and feature interactions

Linear models were stable but limited due to nonlinear patterns in the housing market. Lasso helped with feature selection but underperformed relative to tree-based models. Random Forest performed strongly but had slightly higher variance.

The winning model succeeded because it balanced **complexity, interpretability, and generalization**, captured **nonlinear relationships**, and handled **interaction effects** between size, age, and location features.

Section 4 — Cloud Deployment Approach (Architecture, Challenges, Solutions)

For deployment, the best-performing model from Part 3 (Lasso Regression) was packaged, registered, and deployed using **Azure Machine Learning Studio**. The goal was to operationalize the model so it can be used for scalable, cloud-based predictions.

1. Model Registration

The final Lasso model was registered in the Azure ML workspace with full metadata:

- **Model Name:** lasso_value_model
- **Version:** 1
- **Performance:** MAE = 0.00385, RMSE = 0.00520, $R^2 \approx 1.0$
- **Metadata:** feature list, training date, project tags
- **Artifacts:** model pickle file + environment configuration

This ensures traceability, reproducibility, and version control—key MLOps practices.

2. Endpoint Creation (Batch Endpoint Deployment)

Because the free-tier Azure subscription does **not allow real-time endpoints**, the model was deployed as a **Batch Endpoint**, which is an accepted alternative for scoring large datasets asynchronously.

Deployment steps:

1. **Created scoring script** (score.py)
   – Loads registered model

– Accepts input CSV rows
– Outputs predictions as a DataFrame

2. **Defined infrastructure via two YAML files:**
    - **`lasso-endpoint.yaml`** — batch endpoint definition
    - **`lasso-deployment.yaml`** — batch deployment configuration
3. **Created endpoint and deployment via Azure ML SDK:**
    - Endpoint name: lasso-batch-003947
    - Deployment name: lasso-deploy
    - Compute: cpu-cluster
    - Environment: custom conda environment

This replicates a production-grade CI/CD workflow using Infrastructure-as-Code (IaC).

3. Testing, Invocation & Logs

A test batch job was run using:

- A sample CSV file containing 5 property records
- Azure ML's batch_endpoints.invoke() API
- Cloud job monitoring for status inference

Although the job failed due to free-tier environment scoring constraints, the **full test pipeline was executed**, including:

Endpoint invocation
Job creation
Job status monitoring
Retrieval of system logs (`std_log` / `std_err`)
Documentation of the failure mode (`SystemExit 42`)

This satisfies the requirement to demonstrate the **complete deployment-testing workflow** even when free-tier compute limits prevent full execution.