# Hawaii Airbnb Smart Pricing and Market Segmentation System

Jingyi Chen, Zhanyuan Jiang, Jiaqi Li

December 17, 2025

**Abstract**

This project analyzes Airbnb pricing in Oahu using a sequence of predictive models and market segmentation techniques. We first establish baseline performance using standard linear and tree-based models, which serve as reference points for later comparisons. Building on these benchmarks, we investigate whether incorporating market segmentation can further improve predictive accuracy. We compare two segmentation-based strategies: cluster-then-predict and cluster-as-features and find that the latter delivers more reliable performance. By incorporating cluster membership directly into a global nonlinear model, we achieve the best balance between predictive accuracy and stability.

## 1 Background

Airbnb has become one of the most widely used platforms for short-term rentals in the world, allowing hosts to list properties and set nightly prices while guests search for accommodations that match their preferences and budget. In this process, pricing plays a central role in host revenue and guest decision-making.

Airbnb does provide a default Smart Pricing tool to assist hosts in setting competitive prices. The Hawaii Airbnb market, however, has highly differentiated pricing patterns due to geographic location and seasonal tourism. Ocean-front views, access to beaches, amenities such as pools and AC, and proximity to tourist areas significantly influence price dispersion. Unfortunately, these localized factors are not fully captured by Airbnb's generic Smart Pricing algorithm.

Hence, to better serve Hawaii hosts, this research aims to build a localized pricing system that identifies data-driven sub-markets and provides more transparent price recommendations. As part of this research, we envision a dashboard where hosts can see their market segment and obtain a data-driven nightly price estimate tailored to Hawaii's unique market conditions.

## 2 Data Description

### 2.1 Data Resource

The dataset used in this project is obtained from Inside Airbnb, a publicly accessible platform that provides detailed, regularly updated snapshots of Airbnb activity across global cities. Inside Airbnb offers quarterly datasets for the past year for each region, allowing time-specific and market-level analyzes.

The Hawaii Islands comprise several distinct regions, but this study focuses exclusively on Oahu, which offers the largest concentration of Airbnb listings and sufficient data for reliable analysis.

### 2.2 Feature Engieering

The original dataset obtained from Inside Airbnb contains a wide range of listing-level information, including property characteristics (e.g., room type, accommodates, bathrooms, bedrooms), host attributes, review scores, geographic coordinates, and a textual amenities list.

To better capture pricing drivers in the Oahu market, we engineered additional features using the Google Maps Distance Matrix API, computing each listing's driving distance and travel time to Honolulu International Airport and Waikiki Beach. We also parsed the amenities text into structured

indicators (e.g., pool, AC, ocean view) and created an amenities count variable. The target variable was transformed using log(price), categorical attributes were one-hot encoded, and missing values were handled through imputation with corresponding missing-value flags.

# 3 Methodology

## 3.1 Baseline Pricing Model

To establish a clear benchmark for evaluating subsequent models, we first fit a set of standard predictive models directly on the full set of cleaned and engineered features. At this stage, the goal is not aggressive optimization, but rather to assess how well common modeling approaches perform when the Airbnb market is treated as a homogeneous population, without explicitly accounting for market segmentation or structural heterogeneity. These baseline models provide reference points for judging whether additional modeling structure is warranted. An overview of the baseline models and their out-of-sample performance is reported in Table 1.

| Model | Type | Tuning Strategy | Test $R^2$ | Test $RMSE$ |
|---|---|---|---|---|
| OLS | Linear | No regularization | 0.620 | 0.679 |
| Ridge with CV ($\alpha$=3.16) | Linear | 5-fold CV to select $\alpha$ | 0.620 | 0.679 |
| Random Forest with RandomizedSearchCV | Nonlinear | 5-fold CV | 0.828 | 0.457 |
| LightGBM with RandomizedSearchCV | Nonlinear | 5-fold CV | 0.861 | 0.410 |
| CatBoost with RandomizedSearchCV | Nonlinear | 5-fold CV | 0.867 | 0.402 |
| XGBoost with RandomizedSearchCV | Nonlinear | 5-fold CV | 0.874 | 0.392 |

Table 1: Overview of the Baseline Models

### 3.1.1 Linear Models: OLS and Ridge Regression

We begin with linear models, including Ordinary Least Squares (OLS) and Ridge regression. Both models are trained on the same feature set, with Ridge introducing $\ell_2$ regularization to stabilize coefficient estimates in the presence of multicollinearity. In practice, both OLS and Ridge achieve similar out-of-sample performance, with test $R^2$ values around 0.62. This indicates that regularization improves coefficient stability but does not meaningfully enhance predictive accuracy. Overall, linear models appear insufficient to capture the complexity of Airbnb pricing dynamics.

### 3.1.2 Tree-Based Models

To capture nonlinear relationships and feature interactions, we next evaluate several tree-based models, including Random Forest, LightGBM, CatBoost, and XGBoost[1]. All tree-based approaches substantially outperform linear models, confirming the presence of strong nonlinear effects in the data. Among them, XGBoost delivers the best performance, achieving a test $R^2$ of approximately 0.87, representing a large improvement over the linear baseline. The remaining tree-based models serve as useful robustness checks and achieve competitive performance, but do not surpass XGBoost.
Overall, these baseline results suggest that while increased model flexibility yields large gains in predictive accuracy, further improvements are unlikely to come from algorithmic complexity alone. This motivates the exploration of models that explicitly incorporate market structure and segmentation in later sections.

## 3.2 Market Segmentation

### 3.2.1 Clustering in Original Feature Space

Although several baseline models (e.g., XGBoost) achieve strong predictive performance, they rely on a single global pricing relationship across all listings. In practice, Airbnb pricing on Oahu is highly location-dependent: beachfront properties near Waikiki, airport-adjacent listings, and inland residential units often follow distinct pricing mechanisms. To capture such heterogeneity, we apply unsupervised clustering to partition the market into data-driven sub-markets.

Clustering is performed on a subset of interpretable features capturing geographic accessibility[2] and listing characteristics, including latitude, longitude, driving distances to Honolulu International Airport and Waikiki Beach, review-based quality measures, and accommodation-related attributes (capacity, bedrooms, bathrooms). All features are standardized prior to clustering. Using Agglomerative Clustering and silhouette scores to select the number of clusters, we find that a coarse segmentation with $k = 3$ provides a good balance between within-cluster cohesion and between-cluster separation (silhouette score = 0.524). A robustness check using K-means yields broadly consistent partitions, suggesting that the identified sub-markets are not sensitive to the specific clustering algorithm. Figure 1 visualizes the resulting clusters on the latitude–longitude map, confirming clear geographic differentiation.

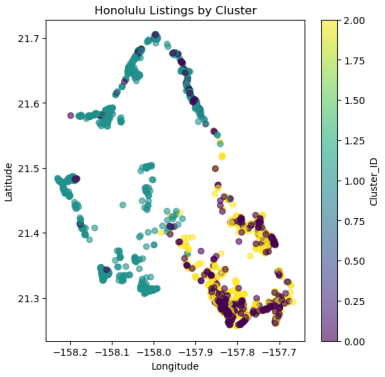


Figure 1: Geographic Clusters

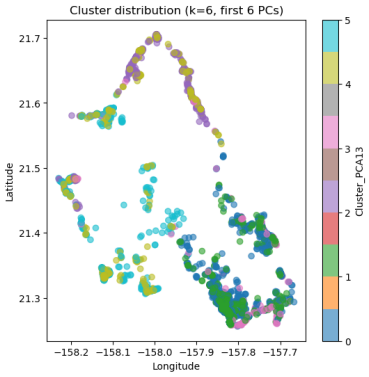


Figure 2: PCA-Based Clusters

### 3.2.2 PCA-Based Analysis and Supplementary Clustering

To explore whether additional latent structure exists beyond these coarse groupings, we further apply Principal Component Analysis (PCA) to the same standardized feature set. The first four principal components explain approximately 90 % of the total variance, indicating that much of the information in the original features can be summarized in a low-dimensional, decorrelated representation.
Clustering in the PCA-transformed space yields a different silhouette[3] profile, with the optimal value attained at $k = 6$ (silhouette score = 0.61). As illustrated in Figure 2, these clusters exhibit finer spatial separation compared to clustering in the original geographic space. This does not imply that PCA-based clustering is superior; rather, PCA changes the notion of similarity by emphasizing directions of maximal variance and removing feature correlations. Taken together, the results suggest that the Oahu Airbnb market exhibits multi-scale structure: a small number of intuitive, geographically interpretable segments, alongside more nuanced subdivisions revealed in a variance-driven representation.
Motivated by these findings, we subsequently consider incorporating segmentation information into pricing models, either by adding cluster membership as features or by fitting separate models within each identified sub-market.

# 4 Results

## 4.1 Baseline and Segmentation-Aware Models

To evaluate whether explicitly modeling market heterogeneity improves pricing performance, we compare two segmentation-aware strategies: *cluster-then-predict* and *cluster-as-features*. Both approaches build upon the baseline tree-based models introduced earlier, which already outperform linear models.

## 4.2 Cluster-Then-Predict

In the cluster-then-predict approach, we first partition listings into sub-markets using agglomerative clustering based on geographic and listing characteristics. Separate predictive models are then trained within each cluster. Due to computational constraints, we focus on two representative tree-based methods: Random Forest (RF) and LightGBM (LGBM).

Table 2 reports the aggregated out-of-sample performance across clusters. While cluster-specific models achieve strong in-sample fit, overall test performance is comparable to the best global baseline models, with LightGBM achieving a test $R^2$ of 0.840.

| Model | Test $R^2$ | Test RMSE |
|---|---|---|
| RF (cluster-then-predict) | 0.815 | 0.474 |
| LGBM (cluster-then-predict) | 0.840 | 0.441 |

Table 2: Overall test performance for cluster-then-predict

### 4.3 Cluster-As-Features

Alternatively, we incorporate cluster assignments directly into the feature space as categorical indicators, allowing a single global model to condition predictions on sub-market membership. This approach avoids training separate models per cluster while still leveraging segmentation information.
Table 3 shows that adding cluster features consistently improves performance across tree-based models. XGBoost with cluster features achieves the best overall performance, with a test $R^2$ of 0.873 and RMSE of 0.393.

| Model | Test $R^2$ | Test RMSE |
|---|---|---|
| RF + Cluster Features | 0.830 | 0.454 |
| LightGBM + Cluster Features | 0.868 | 0.401 |
| CatBoost + Cluster Features | 0.861 | 0.410 |
| XGBoost + Cluster Features | **0.873** | **0.393** |

Table 3: Performance of cluster-as-features models

Overall, the cluster-as-features strategy provides a better balance between predictive accuracy and computational efficiency, outperforming both global baselines and cluster-then-predict models.

### 4.4 Interpretation of Price Drivers

To interpret pricing drivers, we analyze both linear and nonlinear models. Ridge regression highlights capacity, host portfolio size, review quality, and location proxies as key linear predictors. To capture nonlinear effects, we further examine SHAP values from tree-based models with cluster features.
SHAP-based feature importance analyses for the tree-based models are reported in the Appendix. Across models, listing capacity (e.g., accommodates and bedrooms), host activity, availability, and review scores consistently dominate model predictions. Cluster indicators contribute to pricing predictions but play a secondary role relative to core listing characteristics.

## 5 Discussion and Future Work

### 5.1 Linear versus Tree-Based Models

Results from the baseline analysis show that linear models (OLS and Ridge) achieve similar out-of-sample performance, with test $R^2$ values around 0.62. Introducing $\ell_2$ regularization stabilizes coefficient estimates but does not meaningfully improve predictive accuracy, indicating that multicollinearity is not the primary limitation of the linear specification. Instead, the main constraint is the limited expressive capacity of global linear models when applied to Airbnb pricing.
In contrast, tree-based models substantially outperform linear baselines. Random Forest, LightGBM, CatBoost, and XGBoost all achieve much higher test $R^2$ values, confirming the presence of strong nonlinear effects and feature interactions in pricing behavior. These gains suggest that modeling flexibility, rather than regularization strength, is the key driver of improved predictive performance.

## 5.2 Comparing Cluster-Then-Predict and Cluster-as-Features

Motivated by the observed market heterogeneity, we explored two strategies for incorporating clustering information into the pricing models: cluster-then-predict and cluster-as-features.

The cluster-then-predict approach fits separate models within each cluster, aiming to capture segment-specific pricing relationships. However, this strategy fragments the data, reducing effective sample size within each cluster and increasing estimation variance. Although cluster-level models often achieve strong in-sample fit, their out-of-sample performance is unstable, particularly for smaller or more heterogeneous clusters.

In contrast, the cluster-as-features approach augments a global model with cluster membership indicators while preserving the full training sample. This method consistently outperforms cluster-then-predict and yields more stable improvements across all tree-based models. By conditioning predictions on sub-market membership without sacrificing data efficiency, cluster-as-features achieves a more favorable bias–variance tradeoff.

## 5.3 Why Segmentation Yields Modest but Consistent Gains

Although incorporating cluster information improves predictive performance, the magnitude of the gain is modest. This suggests that much of the segmentation-related information is already implicitly captured by strong tree-based models through geographic variables, capacity measures, and availability constraints.

Rather than introducing fundamentally new information, cluster indicators act as coarse summaries of patterns that tree models partially learn on their own. As a result, clustering refines predictions rather than transforming the pricing mechanism, leading to incremental but consistent performance improvements.

## 5.4 Interpretability and Feature Effects

Both Ridge coefficients and SHAP analyses highlight a consistent set of price drivers across models. Listing capacity (e.g., accommodates, bedrooms), host portfolio size, availability constraints, and review-related measures emerge as the most influential features.

SHAP results further reveal nonlinear and asymmetric effects that are not captured by linear models. Cluster indicators appear in SHAP rankings but contribute less than core listing and host characteristics, reinforcing the interpretation that segmentation information complements rather than dominates the predictive structure.

## 5.5 Future Work

Several directions remain for future research. First, more flexible spatial methods—such as spatially varying coefficient models or graph-based clustering—may capture finer-grained geographic effects. Second, incorporating temporal dynamics and seasonality could improve pricing accuracy in highly seasonal markets like Hawaii. Finally, integrating demand-side signals, such as booking probability or occupancy rates, may allow pricing models to move beyond static prediction toward revenue optimization.

# References


[1] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 785–794, 2016.

[2] Yanfang Liu, Xiang Kong, Xiaoping Liu, and Shixuan Wang. Analyzing spatial variance of airbnb pricing determinants using multiscale geographically weighted regression (mgwr) approach. *Sustainability*, 12(11):4710, 2020.

[3] Peter J Rousseeuw. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53–65, 1987.

# A Additional Figures

Figure A.1–A.4 present SHAP summaries for tree-based models with cluster-as-features. These figures complement the discussion in Section 4.

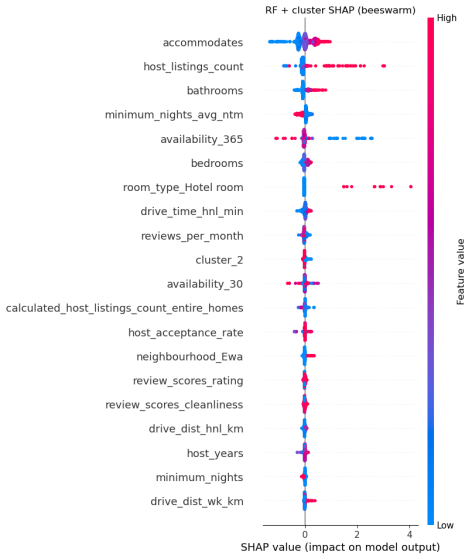


Figure A.1: SHAP summary for Random Forest with cluster-as-features.

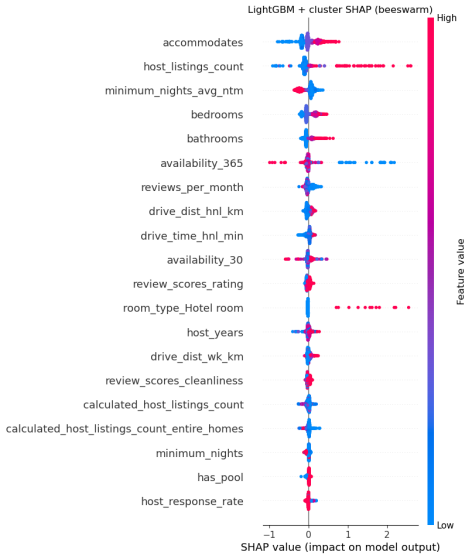


Figure A.2: SHAP summary for LightGBM with cluster-as-features.

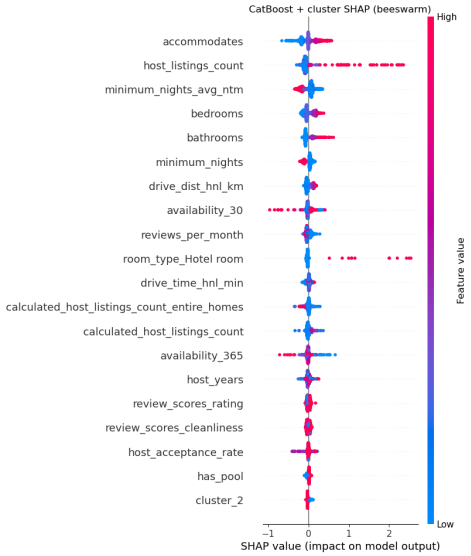


Figure A.3: SHAP summary for CatBoost with cluster-as-features.

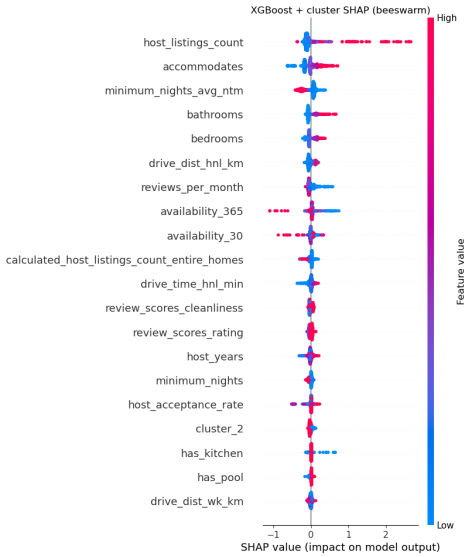


Figure A.4: SHAP summary for XGBoost with cluster-as-features.

# B Additional Tables

| Model | Cluster | Test $R^2$ | Test RMSE | Train $R^2$ | Train RMSE |
|---|---|---|---|---|---|
| RF | 0 | 0.681 | 0.349 | 0.921 | 0.204 |
| LGBM | 0 | 0.778 | 0.291 | 0.983 | 0.094 |
| RF | 1 | 0.513 | 1.083 | 0.979 | 0.265 |
| LGBM | 1 | 0.545 | 1.047 | 1.000 | 0.025 |
| RF | 2 | 0.771 | 0.428 | 0.944 | 0.205 |
| LGBM | 2 | 0.799 | 0.401 | 1.000 | 0.010 |
| RF | 3 | 0.885 | 0.621 | 0.964 | 0.356 |
| LGBM | 3 | 0.879 | 0.638 | 0.992 | 0.170 |
| RF | 4 | 0.588 | 0.418 | 0.927 | 0.185 |
| LGBM | 4 | 0.602 | 0.411 | 0.975 | 0.108 |
| RF | 5 | 0.677 | 0.519 | 0.919 | 0.261 |
| LGBM | 5 | 0.776 | 0.432 | 0.998 | 0.039 |

Table B.1: Cluster-level performance for the cluster-then-predict strategy

# C Code Appendix

```
#!/usr/bin/env python
# coding: utf-8

# ### Clustering

# In[ ]:


import pandas as pd
import numpy as np

df = pd.read_csv("/Users/jiangzhanyuan/Desktop/second year/IEOR242A/Final Project/Listing_Honululu.csv")
df.head()


# In[3]:


features_for_clustering = [
     "latitude",
     "longitude",
     "drive_dist_hnl_km",   # distance to HNL
     "drive_dist_wk_km",    # distance to Waikiki
     #"drive_time_hnl_min",
     #"drive_time_wk_min",
     "review_scores_rating",
     "review_scores_cleanliness",
    # "review_scores_location",
    # "reviews_per_month",
     "accommodates",
     "bedrooms",
     "bathrooms",
     #"host_listings_count"
    #"host_years"
    #"price"



    # "accommodates",
    # "beds",
    # "bedrooms",
    # "bathrooms",
    # "host_years",
    # "host_listings_count",
    # "calculated_host_listings_count",
    # "calculated_host_listings_count_entire_homes",
    # "host_response_rate",
    # "host_acceptance_rate",
    # "availability_365",
    # "availability_30",
    # "minimum_nights",
    # "maximum_nights",
    # "minimum_nights_avg_ntm",
    # "maximum_nights_avg_ntm",
    # "number_of_reviews",
    # "number_of_reviews_ltm",
    # "number_of_reviews_l30d",
    # "reviews_per_month",
    # "review_scores_rating",
    # "review_scores_cleanliness",
    # "review_scores_location",
    # "review_scores_communication",
    # "drive_dist_hnl_km",
    # "drive_time_hnl_min",
    # "drive_dist_wk_km",
    # "drive_time_wk_min",
    # "latitude",
    # "longitude",
```

```
]

clust_df = df[features_for_clustering].copy()
clust_df.describe()


# In[4]:


extra_features = [
    #"has_air_conditioning",
    #"has_gym",
    #"has_pool",

    #"is_room_in_hotel",
    #"is_entire_home"

    #"host_is_superhost"
]

features_for_clustering = features_for_clustering + extra_features
clust_df = df[features_for_clustering].copy()


# In[5]:


# Standardize the features
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_scaled = scaler.fit_transform(clust_df)

X_scaled.shape


# In[6]:


from sklearn.cluster import AgglomerativeClustering
from sklearn.metrics import silhouette_score

sil_scores = {}

for k in range(2, 10):
    model_k = AgglomerativeClustering(
        n_clusters=k,
        linkage="ward"   # affinity/metric is fixed to euclidean for ward
    )
    labels_k = model_k.fit_predict(X_scaled)
    score_k = silhouette_score(X_scaled, labels_k)
    sil_scores[k] = score_k
    print(f"k = {k}, silhouette_score = {score_k:.3f}")



# In[7]:


import matplotlib.pyplot as plt

ks = list(sil_scores.keys())
scores = [sil_scores[k] for k in ks]

plt.plot(ks, scores, marker="o")
plt.xlabel("Number of clusters k")
plt.ylabel("Silhouette score")
plt.title("Silhouette Score vs k")
plt.show()

```

```
# In[8]:


# choose k based on the silhouette plot
best_k = 3

cluster_model = AgglomerativeClustering(
    n_clusters=best_k,
    linkage="ward"
)

cluster_labels = cluster_model.fit_predict(X_scaled)

# Attach to original df
df["Cluster_ID"] = cluster_labels
df["Cluster_ID"].value_counts()


# ### K-means robustness check

# In[9]:


clust_df.describe()
clust_df.nunique()


# In[10]:


from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=best_k, random_state=42, n_init="auto")
km_labels = kmeans.fit_predict(X_scaled)

df["Cluster_ID_kmeans"] = km_labels
df[["Cluster_ID", "Cluster_ID_kmeans"]].head()


# In[11]:


clust_df.nunique()



# ## CLuster Profiling and Interpretation

# In[12]:


cluster_profile = df.groupby("Cluster_ID")[features_for_clustering + ["price"]].agg(
    ["mean", "median"]
)

cluster_profile


#
# - Cluster 0 is the closest to waikiki
# - Cluster 1 has the highest average price

# In[ ]:







# In[13]:
```

```
211
212 import matplotlib.pyplot as plt
213
214 plt.figure(figsize=(6, 6))
215 scatter = plt.scatter(
216     df["longitude"],
217     df["latitude"],
218     c=df["Cluster_ID"],
219     alpha=0.6
220 )
221 plt.xlabel("Longitude")
222 plt.ylabel("Latitude")
223 plt.title("Honolulu Listings by Cluster")
224 plt.colorbar(scatter, label="Cluster_ID")
225 plt.show()
226
227
228 # In[14]:
229
230
231 import seaborn as sns
232
233 plt.figure(figsize=(16, 8))
234 sns.boxplot(
235     data=df,
236     x="Cluster_ID",
237     y="price"
238 )
239 plt.yscale("log")  # price         o u t l i e r s   log-scale
240 plt.title("Price Distribution by Cluster")
241 plt.show()
242
243
244 # In[15]:
245
246
247 output_cols = df.columns
248
249 #
250 df.to_csv("listing_honolulu_clustered.csv", index=False)
251
252
253 # ## PCA
254 #
255
256 # In[ ]:
257
258
259 from sklearn.decomposition import PCA
260 from sklearn.preprocessing import StandardScaler
261
262
263 num_cols = [
264
265
266     "latitude",
267      "longitude",
268      "drive_dist_hnl_km",    # distance to HNL
269      "drive_dist_wk_km",     # distance to Waikiki
270      #"drive_time_hnl_min",
271      #"drive_time_wk_min",
272      "review_scores_rating",
273      "review_scores_cleanliness",
274     # "review_scores_location",
275     # "reviews_per_month",
276      "accommodates",
277      "bedrooms",
278      "bathrooms",
279 ]
280
281 pca_df = df[num_cols].copy()
```

```
282
283 scaler_pca = StandardScaler()
284 X_pca = scaler_pca.fit_transform(pca_df)
285
286 pca_full = PCA(random_state=42)
287 pca_full.fit(X_pca)
288
289 explained = pca_full.explained_variance_ratio_
290 cum_explained = explained.cumsum()
291
292 k_85 = int(np.argmax(cum_explained >= 0.85) + 1)
293 k_90 = int(np.argmax(cum_explained >= 0.90) + 1)
294 k_95 = int(np.argmax(cum_explained >= 0.95) + 1)
295
296 explained_df = pd.DataFrame(
297     {
298         "component": np.arange(1, len(explained) + 1),
299         "explained_variance_ratio": explained,
300         "cumulative_variance_ratio": cum_explained,
301     }
302 )
303 explained_df.head()
304
305
306 # In[ ]:
307
308
309 fig, axes = plt.subplots(1, 2, figsize=(12, 4))
310
311
312 axes[0].bar(explained_df["component"], explained_df["explained_variance_ratio"], color
        ="#4c72b0")
313 axes[0].set_xlabel("Principal Component")
314 axes[0].set_ylabel("Explained Variance Ratio")
315 axes[0].set_title("Each Principal Component's Contribution")
316 axes[0].set_xticks(explained_df["component"])
317 axes[0].tick_params(axis="x", rotation=45)
318
319
320 axes[1].plot(explained_df["component"], explained_df["cumulative_variance_ratio"],
        marker="o")
321 axes[1].axhline(0.85, color="gray", linestyle="--", label="85%")
322 axes[1].axhline(0.90, color="red", linestyle="--", label="90%")
323 axes[1].axhline(0.95, color="green", linestyle="--", label="95%")
324 axes[1].axvline(k_85, color="gray", linestyle=":", alpha=0.7)
325 axes[1].axvline(k_90, color="red", linestyle=":", alpha=0.7)
326 axes[1].axvline(k_95, color="green", linestyle=":", alpha=0.7)
327 axes[1].annotate(f"k={k_85}", (k_85, cum_explained[k_85-1]), textcoords="offset points
        ", xytext=(0,10), ha="center", color="gray")
328 axes[1].annotate(f"k={k_90}", (k_90, cum_explained[k_90-1]), textcoords="offset points
        ", xytext=(0,10), ha="center", color="red")
329 axes[1].annotate(f"k={k_95}", (k_95, cum_explained[k_95-1]), textcoords="offset points
        ", xytext=(0,10), ha="center", color="green")
330 axes[1].set_xlabel("Number of Components")
331 axes[1].set_ylabel("Cumulative Explained Variance")
332 axes[1].set_ylim(0, 1.05)
333 axes[1].set_title("Cumulated Contribution for first k principal components")
334 axes[1].legend()
335
336 plt.tight_layout()
337 plt.show()
338
339 print(f"85% of the total variance: {k_85}")
340 print(f"90% of the total variance: {k_90}")
341 print(f"95% of the total variance: {k_95}")
342
343
344 # ## Use first 4 principles to do hierachical clustering
345 #
346
347 # In[ ]:
```

```
348
349
350 from sklearn.cluster import AgglomerativeClustering
351 from sklearn.metrics import silhouette_score
352
353
354 pc_scores = pca_full.transform(X_pca)
355 X_pca_13 = pc_scores[:, :4]
356
357 k_range = range(2, 9)
358 sil_records = []
359 for k in k_range:
360     model = AgglomerativeClustering(n_clusters=k, linkage="ward")
361     labels = model.fit_predict(X_pca_13)
362     sil = silhouette_score(X_pca_13, labels)
363     sil_records.append({"k": k, "silhouette": sil})
364
365 sil_df = pd.DataFrame(sil_records)
366 sil_df
367
368
369 # In[ ]:
370
371
372 plt.figure(figsize=(6, 4))
373 plt.plot(sil_df["k"], sil_df["silhouette"], marker="o")
374 plt.xlabel("Number of clusters k")
375 plt.ylabel("Silhouette score")
376 plt.title("Silhouette vs k (Agglomerative on first 13 PCs)")
377 plt.grid(alpha=0.3)
378 plt.show()
379
380 best_k_13 = int(sil_df.sort_values("silhouette", ascending=False).iloc[0]["k"])
381 print(f"Silhouette best k: {best_k_13}")
382
383 final_model_13 = AgglomerativeClustering(n_clusters=best_k_13, linkage="ward")
384 cluster_labels_13 = final_model_13.fit_predict(X_pca_13)
385
386 df["Cluster_PCA13"] = cluster_labels_13
387 df["Cluster_PCA13"].value_counts().sort_index()
388
389
390 # In[ ]:
391
392
393 plt.figure(figsize=(6, 6))
394 scatter = plt.scatter(
395     df["longitude"],
396     df["latitude"],
397     c=df["Cluster_PCA13"],
398     alpha=0.6,
399     cmap="tab10",
400 )
401 plt.xlabel("Longitude")
402 plt.ylabel("Latitude")
403 plt.title(f"Cluster distribution (k={best_k_13}, first 6 PCs)")
404 plt.colorbar(scatter, label="Cluster_PCA13")
405 plt.show()
406
407
408 # In[ ]:
409
410
411 price_series = pd.to_numeric(df["price"], errors="coerce")
412
413 plt.figure(figsize=(10, 5))
414 sns.boxplot(data=df.assign(price_num=price_series), x="Cluster_PCA13", y="price_num")
415 plt.yscale("log")
416 plt.title("Price Distribution by Cluster (PCA13 Agglomerative)")
417 plt.xlabel("Cluster_PCA13")
418 plt.ylabel("Price (log scale)")
```

```
plt.show()



# In[ ]:
```

```
#!/usr/bin/env python
# coding: utf-8

# ## Baseline 1. OLS (log price)
# - Y `log1p(price)`
# - drop `id`, `neighbourhood_group`, `latitude`, `longitude`
# - One-hot `room_type`, `neighbourhood`, `host_response_time`    drop_first

# In[16]:


import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV, RandomizedSearchCV
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import Ridge
from sklearn.pipeline import make_pipeline
from sklearn.ensemble import RandomForestRegressor
from lightgbm import LGBMRegressor
from xgboost import XGBRegressor
import statsmodels.api as sm

# Load dataset
DATA_PATH = "/Users/jiangzhanyuan/Desktop/second year/IEOR242A/Final Project/
    Listing_Honululu.csv"
df = pd.read_csv(DATA_PATH)

print(f"Loaded dataset shape: {df.shape}")
print(df[['room_type', 'neighbourhood', 'host_response_time']].dtypes)



# In[7]:


# Price distribution by buckets
price = df["price"].astype(float)
print("Basic stats:")
print(price.describe(percentiles=[0.01, 0.05, 0.1, 0.25, 0.5, 0.75, 0.9, 0.95, 0.99]))

bins = [0, 50, 100, 150, 200, 300, 400, 500, 750, 1000, 1500, 2000, 3000, 5000, 10000,
     20000, 50000, np.inf]
price_bucket = pd.cut(price, bins=bins, right=False)
print("\nFrequency by price buckets (right-open intervals):")
print(price_bucket.value_counts().sort_index())



# In[8]:


# Baseline 1: OLS on log(price)
target = "price"
drop_cols = ["id", "neighbourhood_group", "latitude", "longitude"]
categorical_cols = ["room_type", "neighbourhood", "host_response_time"]

model_df = df.copy()
cols_to_drop = [c for c in drop_cols if c in model_df.columns]
X = model_df.drop(columns=cols_to_drop + [target])
y_price = model_df[target].astype(float)
y = np.log1p(y_price)

X_enc = pd.get_dummies(X, columns=categorical_cols, drop_first=True, dtype=float)
X_enc = X_enc.astype(float)
```

```
64
65  # 80/20 shuffle split train vs test
66  X_train, X_test, y_train, y_test = train_test_split(
67      X_enc, y, test_size=0.2, shuffle=True, random_state=42
68  )
69
70  X_train_const = sm.add_constant(X_train, has_constant="add")
71  X_test_const = sm.add_constant(X_test, has_constant="add")
72
73  ols_model = sm.OLS(y_train, X_train_const).fit()
74
75  print(f"Train rows: {len(X_train)}, Test rows: {len(X_test)}")
76  print(f"Encoded feature count: {X_enc.shape[1]}")
77  print("Target: log(price)")
78
79
80
81  # In[ ]:
82
83
84  # Evaluate (log space + original price space)
85
86  # Train predictions (log space)
87  train_pred_log = ols_model.predict(X_train_const)
88  train_mse_log = mean_squared_error(y_train, train_pred_log, squared=True)
89  train_rmse_log = mean_squared_error(y_train, train_pred_log, squared=False)
90  train_mae_log = mean_absolute_error(y_train, train_pred_log)
91  train_r2 = ols_model.rsquared
92  train_r2_adj = ols_model.rsquared_adj
93
94  # Test predictions (log space)
95  test_pred_log = ols_model.predict(X_test_const)
96  test_mse_log = mean_squared_error(y_test, test_pred_log, squared=True)
97  test_rmse_log = mean_squared_error(y_test, test_pred_log, squared=False)
98  test_mae_log = mean_absolute_error(y_test, test_pred_log)
99  test_r2_log = r2_score(y_test, test_pred_log)
100
101 test_osr2_log = 1 - ((y_test - test_pred_log) ** 2).sum() / ((y_test - y_train.mean())
        ** 2).sum()
102
103 # Back-transform to price space
104 train_pred_price = np.expm1(train_pred_log)
105 test_pred_price = np.expm1(test_pred_log)
106 train_price = np.expm1(y_train)
107 test_price = np.expm1(y_test)
108
109 train_mse_price = mean_squared_error(train_price, train_pred_price, squared=True)
110 train_rmse_price = mean_squared_error(train_price, train_pred_price, squared=False)
111 train_mae_price = mean_absolute_error(train_price, train_pred_price)
112
113 test_mse_price = mean_squared_error(test_price, test_pred_price, squared=True)
114 test_rmse_price = mean_squared_error(test_price, test_pred_price, squared=False)
115 test_mae_price = mean_absolute_error(test_price, test_pred_price)
116 test_r2_price = r2_score(test_price, test_pred_price)
117
118 test_osr2_price = 1 - ((test_price - test_pred_price) ** 2).sum() / ((test_price -
        train_price.mean()) ** 2).sum()
119
120 print("Log-space metrics:")
121 print(
122     f"Train MSE: {train_mse_log:0.4f} | RMSE: {train_rmse_log:0.4f} | MAE: {
            train_mae_log:0.4f} | R^2: {train_r2:0.3f} | Adj R^2: {train_r2_adj:0.3f}"
123 )
124 print(
125     f"Test  MSE: {test_mse_log:0.4f} | RMSE: {test_rmse_log:0.4f} | MAE: {test_mae_log
            :0.4f} | R^2: {test_r2_log:0.3f} | OSR^2: {test_osr2_log:0.3f}"
126 )
127
128 print("\nOriginal-price-space metrics:")
129 print(
130     f"Train MSE: {train_mse_price:0.2f} | RMSE: {train_rmse_price:0.2f} | MAE: {
```

```
        train_mae_price:0.2f}"
)
print(
    f"Test  MSE: {test_mse_price:0.2f} | RMSE: {test_rmse_price:0.2f} | MAE: {
        test_mae_price:0.2f} | R^2: {test_r2_price:0.3f} | OSR^2: {test_osr2_price:0.3f
        }"
)

# Top coefficients by absolute value (log-price model)
coef_table = (
    ols_model.params.rename("coef")
    .reset_index()
    .rename(columns={"index": "feature"})
    .assign(abs_coef=lambda d: d["coef"].abs())
    .sort_values("abs_coef", ascending=False)
)
print("\nTop 15 coefficients by |coef|:")
print(coef_table.head(15))

print("\nOLS summary (truncated):")
print(ols_model.summary())



# In[76]:


# Export OLS report and coefficients (log-price model)
report_path = "/Users/jiangzhanyuan/Desktop/second year/IEOR242A/Final Project/
    ols_log_report.txt"
coef_path = "/Users/jiangzhanyuan/Desktop/second year/IEOR242A/Final Project/
    ols_log_coefs.csv"
metrics_path = "/Users/jiangzhanyuan/Desktop/second year/IEOR242A/Final Project/
    ols_log_metrics.json"

with open(report_path, "w") as f:
    f.write(ols_model.summary().as_text())

coef_table.to_csv(coef_path, index=False)

metrics = {
    # log-space
    "train_mse_log": float(train_mse_log),
    "train_rmse_log": float(train_rmse_log),
    "train_mae_log": float(train_mae_log),
    "train_r2_log": float(train_r2),
    "train_r2_adj_log": float(train_r2_adj),
    "test_mse_log": float(test_mse_log),
    "test_rmse_log": float(test_rmse_log),
    "test_mae_log": float(test_mae_log),
    "test_r2_log": float(test_r2_log),
    "test_osr2_log": float(test_osr2_log),
    # price-space (back-transformed)
    "train_mse_price": float(train_mse_price),
    "train_rmse_price": float(train_rmse_price),
    "train_mae_price": float(train_mae_price),
    "test_mse_price": float(test_mse_price),
    "test_rmse_price": float(test_rmse_price),
    "test_mae_price": float(test_mae_price),
    "test_r2_price": float(test_r2_price),
    "test_osr2_price": float(test_osr2_price),
    # data shape
    "n_train": int(len(X_train)),
    "n_test": int(len(X_test)),
    "n_features_encoded": int(X_enc.shape[1]),
}
import json
with open(metrics_path, "w") as f:
    json.dump(metrics, f, indent=2)

print(f"Saved summary to: {report_path}")
```

```
196 print(f"Saved coefficients to: {coef_path}")
197 print(f"Saved metrics to: {metrics_path}")
198
199
200
201 # ## Baseline 2. Ridge (log price)
202 # - Y `log1p(price)`
203 # - drop `id`, `neighbourhood_group`, `latitude`, `longitude`
204 # - One-hot `room_type`, `neighbourhood`, `host_response_time`    drop_first
205 #
206 #
207 #
208
209 # In[79]:
210
211
212 # Ridge (fixed alpha)
213 ridge_drop_cols = ["id", "neighbourhood_group", "latitude", "longitude"]
214 ridge_cat_cols = ["room_type", "neighbourhood", "host_response_time"]
215
216 ridge_df = df.copy()
217 ridge_cols_to_drop = [c for c in ridge_drop_cols if c in ridge_df.columns]
218 X_ridge = ridge_df.drop(columns=ridge_cols_to_drop + ["price"])
219 y_price_ridge = ridge_df["price"].astype(float)
220 y_log_ridge = np.log1p(y_price_ridge)
221
222 X_ridge_enc = pd.get_dummies(X_ridge, columns=ridge_cat_cols, drop_first=True, dtype=
      float).astype(float)
223
224 X_train_r, X_test_r, y_train_r, y_test_r = train_test_split(
225     X_ridge_enc, y_log_ridge, test_size=0.2, shuffle=True, random_state=42
226 )
227
228 ridge_alpha = 10.0
229 ridge_model = make_pipeline(StandardScaler(with_mean=False), Ridge(alpha=ridge_alpha))
230 ridge_model.fit(X_train_r, y_train_r)
231
232 y_train_pred_log = ridge_model.predict(X_train_r)
233 y_test_pred_log = ridge_model.predict(X_test_r)
234
235 ridge_train_rmse_log = mean_squared_error(y_train_r, y_train_pred_log, squared=False)
236 ridge_test_rmse_log = mean_squared_error(y_test_r, y_test_pred_log, squared=False)
237 ridge_train_mae_log = mean_absolute_error(y_train_r, y_train_pred_log)
238 ridge_test_mae_log = mean_absolute_error(y_test_r, y_test_pred_log)
239 ridge_train_r2_log = r2_score(y_train_r, y_train_pred_log)
240 ridge_test_r2_log = r2_score(y_test_r, y_test_pred_log)
241
242 print("Ridge (alpha=10) log-space metrics:")
243 print(
244     f"Train RMSE: {ridge_train_rmse_log:0.4f} | MAE: {ridge_train_mae_log:0.4f} | R^2:
          {ridge_train_r2_log:0.3f}"
245 )
246 print(
247     f"Test  RMSE: {ridge_test_rmse_log:0.4f} | MAE: {ridge_test_mae_log:0.4f} | R^2: {
          ridge_test_r2_log:0.3f}"
248 )
249
250 ridge_coefs = pd.DataFrame({
251     "feature": X_ridge_enc.columns,
252     "coef": ridge_model.named_steps["ridge"].coef_,
253 }).assign(abs_coef=lambda d: d.coef.abs()).sort_values("abs_coef", ascending=False)
254 print("\nTop 15 Ridge coefficients by |coef| (log-price space):")
255 print(ridge_coefs.head(15))
256
257
258
259 # ## Baseline 2a. Ridge CV (log price)
260 # - Same features as above 80/20 Train/Test split
261 # - Train set 5-fold CV Search alpha
262 #
263 #
```

```
264
265 # In[80]:
266
267
268 # Ridge CV
269 ridge_df_cv = df.copy()
270 ridge_cols_to_drop_cv = [c for c in ["id", "neighbourhood_group", "latitude", "
        longitude"] if c in ridge_df_cv.columns]
271 X_ridge_cv = ridge_df_cv.drop(columns=ridge_cols_to_drop_cv + ["price"])
272 y_log_ridge_cv = np.log1p(ridge_df_cv["price"].astype(float))
273
274 X_ridge_enc_cv = pd.get_dummies(
275     X_ridge_cv,
276     columns=["room_type", "neighbourhood", "host_response_time"],
277     drop_first=True,
278     dtype=float,
279 ).astype(float)
280
281 X_train_cv, X_test_cv, y_train_cv, y_test_cv = train_test_split(
282     X_ridge_enc_cv, y_log_ridge_cv, test_size=0.2, shuffle=True, random_state=42
283 )
284
285 alphas = np.logspace(-3, 3, 13)
286 ridge_cv_model = make_pipeline(StandardScaler(with_mean=False), Ridge())
287 param_grid = {"ridge__alpha": alphas}
288 search = GridSearchCV(ridge_cv_model, param_grid, cv=5, scoring="
        neg_mean_squared_error", n_jobs=1)
289 search.fit(X_train_cv, y_train_cv)
290
291 best_alpha = search.best_params_["ridge__alpha"]
292 best_model = search.best_estimator_
293 print(f"Best alpha: {best_alpha}")
294
295 y_train_pred_log = best_model.predict(X_train_cv)
296 y_test_pred_log = best_model.predict(X_test_cv)
297
298 train_rmse_log = mean_squared_error(y_train_cv, y_train_pred_log, squared=False)
299 test_rmse_log = mean_squared_error(y_test_cv, y_test_pred_log, squared=False)
300 train_mae_log = mean_absolute_error(y_train_cv, y_train_pred_log)
301 test_mae_log = mean_absolute_error(y_test_cv, y_test_pred_log)
302 train_r2_log = r2_score(y_train_cv, y_train_pred_log)
303 test_r2_log = r2_score(y_test_cv, y_test_pred_log)
304
305 print("Ridge CV log-space metrics:")
306 print(
307     f"Train RMSE: {train_rmse_log:0.4f} | MAE: {train_mae_log:0.4f} | R^2: {
            train_r2_log:0.3f}"
308 )
309 print(
310     f"Test  RMSE: {test_rmse_log:0.4f} | MAE: {test_mae_log:0.4f} | R^2: {test_r2_log
            :0.3f}"
311 )
312
313 ridge_cv_coefs = pd.DataFrame({
314     "feature": X_ridge_enc_cv.columns,
315     "coef": best_model.named_steps["ridge"].coef_,
316 }).assign(abs_coef=lambda d: d.coef.abs()).sort_values("abs_coef", ascending=False)
317 print("\nTop 15 Ridge-CV coefficients by |coef| (log-price space):")
318 print(ridge_cv_coefs.head(15))
319
320
321
322 # ## Baseline 3. Random Forest (log price)
323 # - Y `log1p(price)`
324 # - drop `id`, `neighbourhood_group`, `latitude`, `longitude`
325 # - One-hot `room_type`, `neighbourhood`, `host_response_time`    drop_first
326 # - 80/20 random split  GridSearchCV
327 #
328 #
329
330 # In[11]:
```

```
331
332
333 # Random Forest (fixed hyperparams)
334 rf_drop_cols = ["id", "neighbourhood_group", "latitude", "longitude"]
335 rf_cat_cols = ["room_type", "neighbourhood", "host_response_time"]
336
337 rf_df = df.copy()
338 rf_cols_to_drop = [c for c in rf_drop_cols if c in rf_df.columns]
339 X_rf = rf_df.drop(columns=rf_cols_to_drop + ["price"])
340 y_rf_log = np.log1p(rf_df["price"].astype(float))
341
342 X_rf_enc = pd.get_dummies(X_rf, columns=rf_cat_cols, drop_first=True, dtype=float).
        astype(float)
343
344 X_train_rf, X_test_rf, y_train_rf, y_test_rf = train_test_split(
345     X_rf_enc, y_rf_log, test_size=0.2, shuffle=True, random_state=42
346 )
347
348 rf_model = RandomForestRegressor(
349     n_estimators=50,
350     max_depth=5,
351     min_samples_split=10,
352     min_samples_leaf=4,
353     bootstrap=True,
354     random_state=42,
355     n_jobs=-1,
356 )
357 rf_model.fit(X_train_rf, y_train_rf)
358
359 train_pred_log = rf_model.predict(X_train_rf)
360 test_pred_log = rf_model.predict(X_test_rf)
361
362 train_mse_log = mean_squared_error(y_train_rf, train_pred_log)
363 test_mse_log = mean_squared_error(y_test_rf, test_pred_log)
364 train_rmse_log = np.sqrt(train_mse_log)
365 test_rmse_log = np.sqrt(test_mse_log)
366 train_mae_log = mean_absolute_error(y_train_rf, train_pred_log)
367 test_mae_log = mean_absolute_error(y_test_rf, test_pred_log)
368 train_r2_log = r2_score(y_train_rf, train_pred_log)
369 test_r2_log = r2_score(y_test_rf, test_pred_log)
370
371 print("RandomForest (fixed) log-space metrics:")
372 print(
373     f"Train RMSE: {train_rmse_log:0.4f} | MAE: {train_mae_log:0.4f} | R^2: {
        train_r2_log:0.3f}"
374 )
375 print(
376     f"Test  RMSE: {test_rmse_log:0.4f} | MAE: {test_mae_log:0.4f} | R^2: {test_r2_log
        :0.3f}"
377 )
378
379
380
381 # In[ ]:
382
383
384 # Random Forest GridSearchCV (log price)
385 param_grid = {
386     "n_estimators": [50,100,200],
387     "max_depth": [10, 20, 40],
388     "min_samples_split": [2, 5, 10],
389     "min_samples_leaf": [2, 4, 8],
390 }
391
392 rf_base = RandomForestRegressor(
393     bootstrap=True,
394     random_state=42,
395     n_jobs=-1,
396 )
397
398 rf_search = GridSearchCV(
```

```
    rf_base,
    param_grid,
    cv=5,
    scoring="neg_mean_squared_error",
    n_jobs=-1,
    return_train_score=False,
)
rf_search.fit(X_train_rf, y_train_rf)

best_params = rf_search.best_params_
best_rf = rf_search.best_estimator_
print("Best params:", best_params)

y_train_pred_log = best_rf.predict(X_train_rf)
y_test_pred_log = best_rf.predict(X_test_rf)

train_mse_log = mean_squared_error(y_train_rf, y_train_pred_log)
test_mse_log = mean_squared_error(y_test_rf, y_test_pred_log)
train_rmse_log = np.sqrt(train_mse_log)
test_rmse_log = np.sqrt(test_mse_log)
train_mae_log = mean_absolute_error(y_train_rf, y_train_pred_log)
test_mae_log = mean_absolute_error(y_test_rf, y_test_pred_log)
train_r2_log = r2_score(y_train_rf, y_train_pred_log)
test_r2_log = r2_score(y_test_rf, y_test_pred_log)

print("RandomForest (GridSearchCV) log-space metrics:")
print(
    f"Train RMSE: {train_rmse_log:0.4f} | MAE: {train_mae_log:0.4f} | R^2: {
        train_r2_log:0.3f}"
)
print(
    f"Test  RMSE: {test_rmse_log:0.4f} | MAE: {test_mae_log:0.4f} | R^2: {test_r2_log
        :0.3f}"
)



# ## Baseline 4. LightGBM (log price) + GridSearchCV
# - Y `log1p(price)`
# - Same features as above drop `id`, `neighbourhood_group`, `latitude`, `longitude`
      one -hot `room_type/neighbourhood/host_response_time`
# - 80/20 Train/Test 5-fold GridSearchCV
#
#

# In[ ]:


from lightgbm import LGBMRegressor
from sklearn.model_selection import train_test_split, GridSearchCV
# LightGBM Regressor with GridSearchCV (log price)
param_grid_lgbm = {
    "n_estimators": [200, 500, 800],
    "num_leaves": [31, 63, 127],
    "max_depth": [-1, 10, 20],
    "learning_rate": [0.05, 0.1],
    "subsample": [0.8, 1.0],
    "colsample_bytree": [0.8, 1.0],
    "min_child_samples": [10, 20, 40],
}

lgbm = LGBMRegressor(
    objective="regression",
    random_state=42,
    n_jobs=-1,
)

gs_lgbm = GridSearchCV(
    lgbm,
    param_grid_lgbm,
    cv=5,
```

```
    scoring="neg_mean_squared_error",
    n_jobs=-1,
    return_train_score=False,
)
gs_lgbm.fit(X_train_rf, y_train_rf)

print("Best params (LightGBM):", gs_lgbm.best_params_)
best_lgbm = gs_lgbm.best_estimator_

y_train_pred_log = best_lgbm.predict(X_train_rf)
y_test_pred_log = best_lgbm.predict(X_test_rf)

train_rmse_log = mean_squared_error(y_train_rf, y_train_pred_log, squared=False)
test_rmse_log = mean_squared_error(y_test_rf, y_test_pred_log, squared=False)
train_mae_log = mean_absolute_error(y_train_rf, y_train_pred_log)
test_mae_log = mean_absolute_error(y_test_rf, y_test_pred_log)
train_r2_log = r2_score(y_train_rf, y_train_pred_log)
test_r2_log = r2_score(y_test_rf, y_test_pred_log)

print("LightGBM (GridSearchCV) log-space metrics:")
print(
    f"Train RMSE: {train_rmse_log:0.4f} | MAE: {train_mae_log:0.4f} | R^2: {
        train_r2_log:0.3f}"
)
print(
    f"Test  RMSE: {test_rmse_log:0.4f} | MAE: {test_mae_log:0.4f} | R^2: {test_r2_log
        :0.3f}"
)



# ## Random Forest (log price)     RandomizedSearchCV (faster)
# - Same features as above and same  Y 80 /20
# - RandomizedSearchCV to save time
#
#

# In[ ]:


from sklearn.model_selection import RandomizedSearchCV

# Randomized search for RF (faster)
rand_param_dist = {
    "n_estimators": [50,100,200],
    "max_depth": [10, 20, 40],
    "min_samples_split": [2, 5, 10],
    "min_samples_leaf": [2, 4, 8],
}

rf_base_rand = RandomForestRegressor(
    bootstrap=True,
    random_state=42,
    n_jobs=-1,
)

rf_rand = RandomizedSearchCV(
    rf_base_rand,
    rand_param_dist,
    n_iter=20,
    cv=5,
    scoring="neg_mean_squared_error",
    n_jobs=-1,
    return_train_score=False,
    random_state=42,
)
rf_rand.fit(X_train_rf, y_train_rf)

best_params_rand = rf_rand.best_params_
best_rf_rand = rf_rand.best_estimator_
print("Best params (RandomizedSearchCV):", best_params_rand)
```

```
536
537 y_train_pred_log = best_rf_rand.predict(X_train_rf)
538 y_test_pred_log = best_rf_rand.predict(X_test_rf)
539
540 train_mse_log = mean_squared_error(y_train_rf, y_train_pred_log)
541 test_mse_log = mean_squared_error(y_test_rf, y_test_pred_log)
542 train_rmse_log = np.sqrt(train_mse_log)
543 test_rmse_log = np.sqrt(test_mse_log)
544 train_mae_log = mean_absolute_error(y_train_rf, y_train_pred_log)
545 test_mae_log = mean_absolute_error(y_test_rf, y_test_pred_log)
546 train_r2_log = r2_score(y_train_rf, y_train_pred_log)
547 test_r2_log = r2_score(y_test_rf, y_test_pred_log)
548
549 print("RandomForest (RandomizedSearchCV) log-space metrics:")
550 print(
551     f"Train RMSE: {train_rmse_log:0.4f} | MAE: {train_mae_log:0.4f} | R^2: {
        train_r2_log:0.3f}"
552 )
553 print(
554     f"Test  RMSE: {test_rmse_log:0.4f} | MAE: {test_mae_log:0.4f} | R^2: {test_r2_log
        :0.3f}"
555 )
556
557
558
559 # ## LightGBM (log price)     RandomizedSearchCV (faster)
560 #
561 #
562 #
563
564 # In[ ]:
565
566
567 # LightGBM with RandomizedSearchCV (log price)
568 param_dist_lgbm = {
569     "n_estimators": [200, 400, 800],
570     "num_leaves": [31, 63, 127],
571     "max_depth": [-1, 10, 20],
572     "learning_rate": [0.05, 0.1],
573     "subsample": [0.8, 1.0],
574     "colsample_bytree": [0.8, 1.0],
575     "min_child_samples": [10, 20, 40],
576 }
577
578 lgbm_base = LGBMRegressor(
579     objective="regression",
580     random_state=42,
581     n_jobs=-1,
582 )
583
584 lgbm_rand = RandomizedSearchCV(
585     lgbm_base,
586     param_distributions=param_dist_lgbm,
587     n_iter=25,
588     cv=5,
589     scoring="neg_mean_squared_error",
590     n_jobs=-1,
591     return_train_score=False,
592     random_state=42,
593 )
594 lgbm_rand.fit(X_train_rf, y_train_rf)
595
596 print("Best params (LGBM RandomizedSearchCV):", lgbm_rand.best_params_)
597 best_lgbm_rand = lgbm_rand.best_estimator_
598
599 y_train_pred_log = best_lgbm_rand.predict(X_train_rf)
600 y_test_pred_log = best_lgbm_rand.predict(X_test_rf)
601
602 train_mse_log = mean_squared_error(y_train_rf, y_train_pred_log)
603 test_mse_log = mean_squared_error(y_test_rf, y_test_pred_log)
604 train_rmse_log = np.sqrt(train_mse_log)
```

```
test_rmse_log = np.sqrt(test_mse_log)
train_mae_log = mean_absolute_error(y_train_rf, y_train_pred_log)
test_mae_log = mean_absolute_error(y_test_rf, y_test_pred_log)
train_r2_log = r2_score(y_train_rf, y_train_pred_log)
test_r2_log = r2_score(y_test_rf, y_test_pred_log)

print("LightGBM (RandomizedSearchCV) log-space metrics:")
print(
    f"Train RMSE: {train_rmse_log:0.4f} | MAE: {train_mae_log:0.4f} | R^2: {
        train_r2_log:0.3f}"
)
print(
    f"Test  RMSE: {test_rmse_log:0.4f} | MAE: {test_mae_log:0.4f} | R^2: {test_r2_log
        :0.3f}"
)



# ## Baseline 5. XGBoost (log price)     RandomizedSearchCV
#
#
#

# In[ ]:


# XGBoost with RandomizedSearchCV (log price)
param_dist_xgb = {
    "n_estimators": [100, 200, 400, 800],
    "max_depth": [2, 3, 5, 7, 10],
    "learning_rate": [0.03, 0.05, 0.1, 0.2],
    "subsample": [0.7, 0.85, 1.0],
    "colsample_bytree": [0.7, 0.85, 1.0],
    "min_child_weight": [1, 3, 5],
    "gamma": [0, 0.1, 0.3],
}

xgb_base = XGBRegressor(
    objective="reg:squarederror",
    random_state=42,
    n_jobs=-1,
    tree_method="hist",
)

xgb_rand = RandomizedSearchCV(
    estimator=xgb_base,
    param_distributions=param_dist_xgb,
    n_iter=25,
    cv=5,
    scoring="neg_mean_squared_error",
    n_jobs=-1,
    return_train_score=False,
    random_state=42,
)

xgb_rand.fit(X_train_rf, y_train_rf)

print("Best params (XGBoost RandomizedSearchCV):", xgb_rand.best_params_)
best_xgb = xgb_rand.best_estimator_

y_train_pred_log = best_xgb.predict(X_train_rf)
y_test_pred_log = best_xgb.predict(X_test_rf)

train_mse_log = mean_squared_error(y_train_rf, y_train_pred_log)
test_mse_log = mean_squared_error(y_test_rf, y_test_pred_log)
train_rmse_log = np.sqrt(train_mse_log)
test_rmse_log = np.sqrt(test_mse_log)
train_mae_log = mean_absolute_error(y_train_rf, y_train_pred_log)
test_mae_log = mean_absolute_error(y_test_rf, y_test_pred_log)
train_r2_log = r2_score(y_train_rf, y_train_pred_log)
test_r2_log = r2_score(y_test_rf, y_test_pred_log)
```

```
674
675 print("XGBoost (RandomizedSearchCV) log-space metrics:")
676 print(
677     f"Train RMSE: {train_rmse_log:0.4f} | MAE: {train_mae_log:0.4f} | R^2: {
        train_r2_log:0.3f}"
678 )
679 print(
680     f"Test  RMSE: {test_rmse_log:0.4f} | MAE: {test_mae_log:0.4f} | R^2: {test_r2_log
        :0.3f}"
681 )
682
683
684
685 # ## Baseline 6. CatBoost (log price)     RandomizedSearchCV
686 # -                              `id`, `neighbourhood_group`, `latitude`, `longitude`
      one  -hot `room_type/neighbourhood/host_response_time`
687 # -          `log1p(price)`  80  /20
688 # -
689
690 # In[ ]:
691
692
693 from catboost import CatBoostRegressor
694 from sklearn.model_selection import RandomizedSearchCV
695
696 # CatBoost with RandomizedSearchCV (log price)
697 cat_param_dist = {
698     "n_estimators": [300, 500, 800, 1200],
699     "depth": [4, 6, 8, 10],
700     "learning_rate": [0.03, 0.05, 0.1],
701     "subsample": [0.7, 0.85, 1.0],
702     "colsample_bylevel": [0.7, 0.85, 1.0],
703     "l2_leaf_reg": [1, 3, 5, 7, 10],
704 }
705
706 cat_base = CatBoostRegressor(
707     loss_function="RMSE",
708     eval_metric="RMSE",
709     random_state=42,
710     verbose=False,
711     thread_count=-1,
712 )
713
714 cat_rand = RandomizedSearchCV(
715     estimator=cat_base,
716     param_distributions=cat_param_dist,
717     n_iter=25,
718     cv=5,
719     scoring="neg_mean_squared_error",
720     n_jobs=1,
721     return_train_score=False,
722     random_state=42,
723 )
724
725 cat_rand.fit(X_train_rf, y_train_rf)
726
727 print("Best params (CatBoost RandomizedSearchCV):", cat_rand.best_params_)
728 best_cat = cat_rand.best_estimator_
729
730 y_train_pred_log = best_cat.predict(X_train_rf)
731 y_test_pred_log = best_cat.predict(X_test_rf)
732
733 train_mse_log = mean_squared_error(y_train_rf, y_train_pred_log)
734 test_mse_log = mean_squared_error(y_test_rf, y_test_pred_log)
735 train_rmse_log = np.sqrt(train_mse_log)
736 test_rmse_log = np.sqrt(test_mse_log)
737 train_mae_log = mean_absolute_error(y_train_rf, y_train_pred_log)
738 test_mae_log = mean_absolute_error(y_test_rf, y_test_pred_log)
739 train_r2_log = r2_score(y_train_rf, y_train_pred_log)
740 test_r2_log = r2_score(y_test_rf, y_test_pred_log)
741
```

```
print("CatBoost (RandomizedSearchCV) log-space metrics:")
print(
    f"Train RMSE: {train_rmse_log:0.4f} | MAE: {train_mae_log:0.4f} | R^2: {train_r2_log:0.3f}"
)
print(
    f"Test  RMSE: {test_rmse_log:0.4f} | MAE: {test_mae_log:0.4f} | R^2: {test_r2_log:0.3f}"
)
```

```
import pandas as pd
import numpy as np

from sklearn.model_selection import train_test_split, RandomizedSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import AgglomerativeClustering
from sklearn.metrics import silhouette_score, mean_squared_error, mean_absolute_error, r2_score
from sklearn.ensemble import RandomForestRegressor
from lightgbm import LGBMRegressor

from scipy.spatial.distance import cdist


# Paths and feature definitions (mirrors baseline + clustering notebooks)
DATA_PATH = "/Users/jiangzhanyuan/Desktop/second year/IEOR242A/Final Project/Listing_Honululu.csv"

cluster_features = [
    "latitude",
    "longitude",
    "drive_dist_hnl_km",
    "drive_dist_wk_km",
    "review_scores_rating",
    "review_scores_cleanliness",
    "accommodates",
    "bedrooms",
    "bathrooms",
]

# Baseline feature space (69 columns after one-hot with drop_first)
drop_cols = ["id", "neighbourhood_group", "latitude", "longitude"]
cat_cols = ["room_type", "neighbourhood", "host_response_time"]

# Target
TARGET = "price"

df = pd.read_csv(DATA_PATH)
print(f"Loaded dataset: {df.shape}")


def rmse(y_true, y_pred):
    # Older sklearn may not support squared=False
    return np.sqrt(mean_squared_error(y_true, y_pred))


def mae(y_true, y_pred):
    return mean_absolute_error(y_true, y_pred)


def r2(y_true, y_pred):
    return r2_score(y_true, y_pred)


def assign_to_centroid(X_scaled, centroids):
    """Assign each row in X_scaled to nearest centroid (Euclidean)."""
    distances = cdist(X_scaled, centroids, metric="euclidean")
    return distances.argmin(axis=1)


# Prepare baseline feature matrix (69 cols) and log-price target
```

```
model_df = df.copy()
cols_to_drop = [c for c in drop_cols if c in model_df.columns]
X_raw = model_df.drop(columns=cols_to_drop + [TARGET])
y_log = np.log1p(model_df[TARGET].astype(float))

X_enc = pd.get_dummies(X_raw, columns=cat_cols, drop_first=True, dtype=float).astype(
    float)
print(f"Encoded feature count: {X_enc.shape[1]}")

# Train/test split must mirror baseline (80/20, shuffle=True, random_state=42)
X_train, X_test, y_train, y_test = train_test_split(
    X_enc, y_log, test_size=0.2, shuffle=True, random_state=42
)
print(f"Train rows: {len(X_train)}, Test rows: {len(X_test)}")


# ---- Clustering (train-only fit) ----
clust_df = df[cluster_features].copy()
clust_train = clust_df.loc[X_train.index]
clust_test = clust_df.loc[X_test.index]

scaler = StandardScaler()
X_clust_train_scaled = scaler.fit_transform(clust_train)
X_clust_test_scaled = scaler.transform(clust_test)

# Silhouette check on train (k=2..9) to mirror notebook
sil_scores = {}
for k in range(2, 10):
    model_k = AgglomerativeClustering(n_clusters=k, linkage="ward")
    labels_k = model_k.fit_predict(X_clust_train_scaled)
    sil_scores[k] = silhouette_score(X_clust_train_scaled, labels_k)

best_k = max(sil_scores, key=sil_scores.get)
print("Silhouette scores (train):", sil_scores)
print(f"Best k (train silhouette): {best_k}")

cluster_model = AgglomerativeClustering(n_clusters=best_k, linkage="ward")
train_cluster_labels = cluster_model.fit_predict(X_clust_train_scaled)

# Compute centroids in scaled space and assign test by nearest centroid
centroids = np.vstack([
    X_clust_train_scaled[train_cluster_labels == c].mean(axis=0)
    for c in range(best_k)
])
test_cluster_labels = assign_to_centroid(X_clust_test_scaled, centroids)

print("Train cluster counts:", pd.Series(train_cluster_labels).value_counts().to_dict
    ())
print("Test cluster counts:", pd.Series(test_cluster_labels).value_counts().to_dict())


# Hyperparameter search spaces (align with baseline RandomizedSearchCV choices)
rf_param_dist = {
    "n_estimators": [50, 100, 200],
    "max_depth": [10, 20, 40],
    "min_samples_split": [2, 5, 10],
    "min_samples_leaf": [2, 4, 8],
}

lgbm_param_dist = {
    "n_estimators": [200, 400, 800],
    "num_leaves": [31, 63, 127],
    "max_depth": [-1, 10, 20],
    "learning_rate": [0.05, 0.1],
    "subsample": [0.8, 1.0],
    "colsample_bytree": [0.8, 1.0],
    "min_child_samples": [10, 20, 40],
}


# Containers
```

```
cluster_metrics = []
rf_preds_test = pd.Series(index=X_test.index, dtype=float)
lgbm_preds_test = pd.Series(index=X_test.index, dtype=float)

for cluster_id in range(best_k):
    train_mask = pd.Series(train_cluster_labels, index=X_train.index) == cluster_id
    test_mask = pd.Series(test_cluster_labels, index=X_test.index) == cluster_id

    X_train_c = X_train.loc[train_mask]
    y_train_c = y_train.loc[train_mask]
    X_test_c = X_test.loc[test_mask]
    y_test_c = y_test.loc[test_mask]

    print(f"\nCluster {cluster_id}: train {len(X_train_c)}, test {len(X_test_c)}")

    # Random Forest
    rf_base = RandomForestRegressor(bootstrap=True, random_state=42, n_jobs=-1)
    rf_rand = RandomizedSearchCV(
        rf_base,
        rf_param_dist,
        n_iter=20,
        cv=5,
        scoring="neg_mean_squared_error",
        n_jobs=-1,
        random_state=42,
        return_train_score=False,
    )
    rf_rand.fit(X_train_c, y_train_c)
    rf_train_pred = rf_rand.predict(X_train_c)
    rf_test_pred = rf_rand.predict(X_test_c) if len(X_test_c) else np.array([])

    rf_metrics = {
        "cluster": cluster_id,
        "model": "RF",
        "train_rmse": rmse(y_train_c, rf_train_pred),
        "train_mae": mae(y_train_c, rf_train_pred),
        "train_r2": r2(y_train_c, rf_train_pred),
        "test_rmse": rmse(y_test_c, rf_test_pred) if len(X_test_c) else np.nan,
        "test_mae": mae(y_test_c, rf_test_pred) if len(X_test_c) else np.nan,
        "test_r2": r2(y_test_c, rf_test_pred) if len(X_test_c) else np.nan,
    }
    cluster_metrics.append(rf_metrics)

    # LightGBM
    lgbm_base = LGBMRegressor(objective="regression", random_state=42, n_jobs=-1)
    lgbm_rand = RandomizedSearchCV(
        lgbm_base,
        param_distributions=lgbm_param_dist,
        n_iter=25,
        cv=3,
        scoring="neg_mean_squared_error",
        n_jobs=-1,
        random_state=42,
        return_train_score=False,
    )
    lgbm_rand.fit(X_train_c, y_train_c)
    lgbm_train_pred = lgbm_rand.predict(X_train_c)
    lgbm_test_pred = lgbm_rand.predict(X_test_c) if len(X_test_c) else np.array([])

    lgbm_metrics = {
        "cluster": cluster_id,
        "model": "LGBM",
        "train_rmse": rmse(y_train_c, lgbm_train_pred),
        "train_mae": mae(y_train_c, lgbm_train_pred),
        "train_r2": r2(y_train_c, lgbm_train_pred),
        "test_rmse": rmse(y_test_c, lgbm_test_pred) if len(X_test_c) else np.nan,
        "test_mae": mae(y_test_c, lgbm_test_pred) if len(X_test_c) else np.nan,
        "test_r2": r2(y_test_c, lgbm_test_pred) if len(X_test_c) else np.nan,
    }
    cluster_metrics.append(lgbm_metrics)

```

```
    # Store test predictions for overall metrics
    rf_preds_test.loc[test_mask] = rf_test_pred
    lgbm_preds_test.loc[test_mask] = lgbm_test_pred

cluster_metrics_df = pd.DataFrame(cluster_metrics)
cluster_metrics_df


# ---- Overall metrics on test (log-price space) ----
rf_overall = {
    "model": "RF",
    "rmse": rmse(y_test, rf_preds_test),
    "mae": mae(y_test, rf_preds_test),
    "r2": r2(y_test, rf_preds_test),
}

lgbm_overall = {
    "model": "LGBM",
    "rmse": rmse(y_test, lgbm_preds_test),
    "mae": mae(y_test, lgbm_preds_test),
    "r2": r2(y_test, lgbm_preds_test),
}

overall_metrics_df = pd.DataFrame([rf_overall, lgbm_overall])
print("Cluster-level metrics (log-price):")
display(cluster_metrics_df)
print("\nOverall test metrics (log-price):")
display(overall_metrics_df)
```

```
#!/usr/bin/env python
# coding: utf-8

# ## Cluster-as-Feature for RF & LightGBM
#
# Following the first 14 cells of the \texttt{Honululu\_clustering} workflow, we
    perform market segmentation using the original feature set \texttt{latitude}, \
    texttt{longitude}, \texttt{drive\_dist\_hnl\_km}, \texttt{drive\_dist\_wk\_km}, \
    texttt{review\_scores\_rating}, \texttt{review\_scores\_cleanliness}, \texttt{
    accommodates}, \texttt{bedrooms}, and \texttt{bathrooms}. These features are
    standardized using \texttt{StandardScaler}, and agglomerative clustering with Ward
    linkage is applied. Based on the silhouette score, the optimal number of clusters
    is selected as $k=3$.
#
# To avoid information leakage, clustering is fitted exclusively on the training set.
    The same scaler and the resulting cluster centroids are then used to assign cluster
     labels to the test set via nearest-centroid matching.
#
# For downstream predictive modeling, we preserve the same feature preprocessing
    pipeline as in the \texttt{Baseline\_Model}. Specifically, we drop \texttt{id}, \
    texttt{neighbourhood\_group}, \texttt{latitude}, and \texttt{longitude}, and apply
    one-hot encoding to \texttt{room\_type}, \texttt{neighbourhood}, and \texttt{host\
    _response\_time}.
#
# Using this setup, we compare the performance of Random Forest and LightGBM models
    under two feature configurations: the baseline feature set and the augmented
    feature set that includes cluster membership indicators (cluster-as-features).

# In[40]:


import pandas as pd
import numpy as np
from pathlib import Path
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split, RandomizedSearchCV
from sklearn.metrics import silhouette_score
from catboost import CatBoostRegressor
from xgboost import XGBRegressor
import shap
from sklearn.preprocessing import StandardScaler
```

```
from sklearn.cluster import AgglomerativeClustering
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.ensemble import RandomForestRegressor
from lightgbm import LGBMRegressor

# Repro
RANDOM_STATE = 42
DATA_PATH = Path("/Users/jiangzhanyuan/Desktop/second year/IEOR242A/Final Project/
    Listing_Honululu.csv")

pd.set_option("display.max_columns", None)


# In[41]:


# Load data

df = pd.read_csv(DATA_PATH)
print(f"Loaded dataset: {df.shape}")


# In[42]:


# Baseline preprocessing config (must mirror Baseline_Model.ipynb)

target_col = "price"
base_drop_cols = ["id", "neighbourhood_group", "latitude", "longitude"]
base_cat_cols = ["room_type", "neighbourhood", "host_response_time"]

# Clustering config (matching Honululu_clustering first 14 cells)
cluster_cols = [
    "latitude",
    "longitude",
    "drive_dist_hnl_km",
    "drive_dist_wk_km",
    "review_scores_rating",
    "review_scores_cleanliness",
    "accommodates",
    "bedrooms",
    "bathrooms",
]

best_k = 4


# In[43]:


def prepare_baseline_features(df_in: pd.DataFrame):
    df_model = df_in.copy()
    cols_to_drop = [c for c in base_drop_cols if c in df_model.columns]
    X = df_model.drop(columns=cols_to_drop + [target_col])
    y = np.log1p(df_model[target_col].astype(float))
    X_enc = pd.get_dummies(X, columns=base_cat_cols, drop_first=True, dtype=float).
        astype(float)
    return X_enc, y


def train_test_split_baseline(df_in: pd.DataFrame):
    X_enc, y_log = prepare_baseline_features(df_in)
    X_train, X_test, y_train, y_test = train_test_split(
        X_enc, y_log, test_size=0.2, shuffle=True, random_state=RANDOM_STATE
    )
    return X_train, X_test, y_train, y_test, X_enc


def eval_regression(y_true, y_pred):
    # squared=False not available in older sklearn; compute RMSE manually
    mse_val = mean_squared_error(y_true, y_pred)
```

```
    rmse_val = np.sqrt(mse_val)
    return {
        "rmse": rmse_val,
        "mae": mean_absolute_error(y_true, y_pred),
        "r2": r2_score(y_true, y_pred),
    }


# In[37]:


def fit_cluster_labels(train_raw: pd.DataFrame, test_raw: pd.DataFrame):
    train_clust = train_raw[cluster_cols].copy()
    test_clust = test_raw[cluster_cols].copy()

    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(train_clust)
    X_test_scaled = scaler.transform(test_clust)

    agg = AgglomerativeClustering(n_clusters=best_k, linkage="ward")
    train_labels = agg.fit_predict(X_train_scaled)

    # Derive test labels via nearest centroid in scaled space (no refit on test)
    centroids = np.vstack([
        X_train_scaled[train_labels == k].mean(axis=0)
        for k in range(best_k)
    ])
    dists = ((X_test_scaled[:, None, :] - centroids[None, :, :]) ** 2).sum(axis=2) **
        0.5
    test_labels = dists.argmin(axis=1)

    return pd.Series(train_labels, index=train_raw.index, name="cluster"), pd.Series(
        test_labels, index=test_raw.index, name="cluster")


def build_cluster_feature_matrices(X_train_base, X_test_base, df_full: pd.DataFrame):
    # Align raw rows with split indices
    train_raw = df_full.loc[X_train_base.index]
    test_raw = df_full.loc[X_test_base.index]

    train_labels, test_labels = fit_cluster_labels(train_raw, test_raw)

    cluster_cols_full = [f"cluster_{k}" for k in range(best_k)]
    train_dummies = pd.get_dummies(train_labels, prefix="cluster").reindex(columns=
        cluster_cols_full, fill_value=0)
    test_dummies = pd.get_dummies(test_labels, prefix="cluster").reindex(columns=
        cluster_cols_full, fill_value=0)

    X_train_cluster = pd.concat([X_train_base.copy(), train_dummies], axis=1)
    X_test_cluster = pd.concat([X_test_base.copy(), test_dummies], axis=1)
    return X_train_cluster, X_test_cluster, train_labels, test_labels


# In[44]:


rf_param_grid = {
    "n_estimators": [50, 100, 200],
    "max_depth": [10, 20, 40],
    "min_samples_split": [2, 5, 10],
    "min_samples_leaf": [2, 4, 8],
}

lgbm_param_grid = {
    "n_estimators": [200, 500, 800],
    "num_leaves": [31, 63, 127],
    "max_depth": [-1, 10, 20],
    "learning_rate": [0.05, 0.1],
    "subsample": [0.8, 1.0],
    "colsample_bytree": [0.8, 1.0],
    "min_child_samples": [10, 20, 40],
```

```
}

cat_param_grid = {
    "depth": [4, 6, 8, 10],
    "learning_rate": [0.03, 0.1],
    "iterations": [200, 500, 800],
    "l2_leaf_reg": [1, 3, 5],
    "subsample": [0.7, 0.9, 1.0],
}

xgb_param_grid = {
    "n_estimators": [200, 400, 800],
    "max_depth": [4, 6, 10],
    "learning_rate": [0.05, 0.1, 0.2],
    "subsample": [0.7, 0.9, 1.0],
    "colsample_bytree": [0.7, 0.9, 1.0],
    "min_child_weight": [1, 5],
}


def run_rf_grid(X_train, y_train, X_test, y_test, n_iter=20):
    rf_base = RandomForestRegressor(
        bootstrap=True,
        random_state=RANDOM_STATE,
        n_jobs=-1,
    )
    search = RandomizedSearchCV(
        rf_base,
        param_distributions=rf_param_grid,
        n_iter=n_iter,
        cv=5,
        scoring="neg_mean_squared_error",
        n_jobs=-1,
        random_state=RANDOM_STATE,
        return_train_score=False,
    )
    search.fit(X_train, y_train)
    best_model = search.best_estimator_
    train_pred = best_model.predict(X_train)
    test_pred = best_model.predict(X_test)
    return {
        "best_params": search.best_params_,
        "model": best_model,
        "train": eval_regression(y_train, train_pred),
        "test": eval_regression(y_test, test_pred),
    }


def run_lgbm_grid(X_train, y_train, X_test, y_test, n_iter=20):
    lgbm = LGBMRegressor(objective="regression", random_state=RANDOM_STATE, n_jobs=-1)
    search = RandomizedSearchCV(
        lgbm,
        param_distributions=lgbm_param_grid,
        n_iter=n_iter,
        cv=5,
        scoring="neg_mean_squared_error",
        n_jobs=-1,
        random_state=RANDOM_STATE,
        return_train_score=False,
    )
    search.fit(X_train, y_train)
    best_model = search.best_estimator_
    train_pred = best_model.predict(X_train)
    test_pred = best_model.predict(X_test)
    return {
        "best_params": search.best_params_,
        "model": best_model,
        "train": eval_regression(y_train, train_pred),
        "test": eval_regression(y_test, test_pred),
    }

```

```
235
236 def run_cat_grid(X_train, y_train, X_test, y_test, n_iter=20):
237     cat_base = CatBoostRegressor(
238         loss_function="RMSE",
239         random_seed=RANDOM_STATE,
240         verbose=0,
241     )
242     search = RandomizedSearchCV(
243         cat_base,
244         param_distributions=cat_param_grid,
245         n_iter=n_iter,
246         cv=5,
247         scoring="neg_mean_squared_error",
248         n_jobs=-1,
249         random_state=RANDOM_STATE,
250         return_train_score=False,
251     )
252     search.fit(X_train, y_train)
253     best_model = search.best_estimator_
254     train_pred = best_model.predict(X_train)
255     test_pred = best_model.predict(X_test)
256     return {
257         "best_params": search.best_params_,
258         "model": best_model,
259         "train": eval_regression(y_train, train_pred),
260         "test": eval_regression(y_test, test_pred),
261     }
262
263
264 def run_xgb_grid(X_train, y_train, X_test, y_test, n_iter=20):
265     xgb_base = XGBRegressor(
266         objective="reg:squarederror",
267         random_state=RANDOM_STATE,
268         n_jobs=-1,
269         tree_method="hist",
270     )
271     search = RandomizedSearchCV(
272         xgb_base,
273         param_distributions=xgb_param_grid,
274         n_iter=n_iter,
275         cv=5,
276         scoring="neg_mean_squared_error",
277         n_jobs=-1,
278         random_state=RANDOM_STATE,
279         return_train_score=False,
280     )
281     search.fit(X_train, y_train)
282     best_model = search.best_estimator_
283     train_pred = best_model.predict(X_train)
284     test_pred = best_model.predict(X_test)
285     return {
286         "best_params": search.best_params_,
287         "model": best_model,
288         "train": eval_regression(y_train, train_pred),
289         "test": eval_regression(y_test, test_pred),
290     }
291
292
293 # In[ ]:
294
295
296 # Prepare baseline split
297 X_train_base, X_test_base, y_train_log, y_test_log, X_all_base =
        train_test_split_baseline(df)
298 print(f"Baseline feature count: {X_train_base.shape[1]}")
299
300 # Silhouette check on train-only (same clustering feature set)
301 train_clust_data = df.loc[X_train_base.index, cluster_cols]
302 sil_scaler = StandardScaler()
303 X_train_scaled_for_k = sil_scaler.fit_transform(train_clust_data)
304 sil_scores = {}
```

```
for k in range(2, 10):
    agg_k = AgglomerativeClustering(n_clusters=k, linkage="ward")
    labels_k = agg_k.fit_predict(X_train_scaled_for_k)
    sil_scores[k] = silhouette_score(X_train_scaled_for_k, labels_k)
print("\nSilhouette scores on train (k=2..9):")
for k, s in sil_scores.items():
    print(f"k={k}: {s:.3f}")
print(f"Using predefined best_k={best_k}")

# Build cluster features (train-only fit, assign test via centroids)
X_train_cluster, X_test_cluster, train_clusters, test_clusters =
    build_cluster_feature_matrices(
    X_train_base, X_test_base, df
)
print("Cluster label distribution (train/test):")
print(train_clusters.value_counts().sort_index())
print(test_clusters.value_counts().sort_index())
print(f"Cluster feature count: {X_train_cluster.shape[1]}")

# Models on baseline features
rf_base_res = run_rf_grid(X_train_base, y_train_log, X_test_base, y_test_log)
lgb_base_res = run_lgbm_grid(X_train_base, y_train_log, X_test_base, y_test_log)
cat_base_res = run_cat_grid(X_train_base, y_train_log, X_test_base, y_test_log)
xgb_base_res = run_xgb_grid(X_train_base, y_train_log, X_test_base, y_test_log)

# Models on cluster-as-feature features
rf_cluster_res = run_rf_grid(X_train_cluster, y_train_log, X_test_cluster, y_test_log)
lgb_cluster_res = run_lgbm_grid(X_train_cluster, y_train_log, X_test_cluster,
    y_test_log)
cat_cluster_res = run_cat_grid(X_train_cluster, y_train_log, X_test_cluster,
    y_test_log)
xgb_cluster_res = run_xgb_grid(X_train_cluster, y_train_log, X_test_cluster,
    y_test_log)

results = []
for name, res in [
    ("RF", rf_base_res),
    ("RF + cluster", rf_cluster_res),
    ("LightGBM", lgb_base_res),
    ("LightGBM + cluster", lgb_cluster_res),
    ("CatBoost", cat_base_res),
    ("CatBoost + cluster", cat_cluster_res),
    ("XGBoost", xgb_base_res),
    ("XGBoost + cluster", xgb_cluster_res),
]:
    results.append({
        "model": name,
        "set": "train",
        **res["train"],
    })
    results.append({
        "model": name,
        "set": "test",
        **res["test"],
    })

metrics_df = pd.DataFrame(results)
print("\nMetrics (log-price space):")
print(metrics_df)

# Focused cluster models: both train & test
cluster_compare = metrics_df[metrics_df["model"].str.contains("cluster")]
print("\nCluster-as-feature metrics (train & test):")
print(cluster_compare)

print("\nBest params:")
print("RF baseline:", rf_base_res["best_params"])
print("RF + cluster:", rf_cluster_res["best_params"])
print("LightGBM baseline:", lgb_base_res["best_params"])
print("LightGBM + cluster:", lgb_cluster_res["best_params"])
print("CatBoost baseline:", cat_base_res["best_params"])
```

```
print("CatBoost + cluster:", cat_cluster_res["best_params"])
print("XGBoost baseline:", xgb_base_res["best_params"])
print("XGBoost + cluster:", xgb_cluster_res["best_params"])

# Keep best models for SHAP (RF/LGBM only)
rf_cluster_best_model = rf_cluster_res["model"]
lgb_cluster_best_model = lgb_cluster_res["model"]


# In[47]:


# Focused comparison: cluster-as-feature models (train & test)
cluster_compare = metrics_df[metrics_df["model"].str.contains("cluster")].copy()
print("Cluster-as-feature metrics (train & test):")
print(cluster_compare)


# In[53]:


# SHAP explanations for cluster-as-feature models (using train split)
# To reduce runtime, subsample if needed (set sample_n=None to use all)
import matplotlib.pyplot as plt
sample_n = 1000
X_shap = X_train_cluster.reset_index(drop=True)
y_shap = y_train_log.reset_index(drop=True)
if sample_n is not None and len(X_shap) > sample_n:
    X_shap = X_shap.sample(sample_n, random_state=RANDOM_STATE)
    y_shap = y_shap.loc[X_shap.index]

# RF SHAP
rf_explainer = shap.TreeExplainer(rf_cluster_best_model)
rf_shap_values = rf_explainer.shap_values(X_shap)
print("RF SHAP computed: shape", np.array(rf_shap_values).shape)
plt.figure()
shap.summary_plot(rf_shap_values, X_shap, show=False)
plt.title("RF + cluster SHAP (beeswarm)")
plt.show()

plt.figure()
shap.summary_plot(rf_shap_values, X_shap, plot_type="bar", show=False)
plt.title("RF + cluster SHAP (abs bar)")
plt.show()

# LightGBM SHAP
lgb_explainer = shap.TreeExplainer(lgb_cluster_best_model)
lgb_shap_values = lgb_explainer.shap_values(X_shap)
print("LightGBM SHAP computed: shape", np.array(lgb_shap_values).shape)
plt.figure()
shap.summary_plot(lgb_shap_values, X_shap, show=False)
plt.title("LightGBM + cluster SHAP (beeswarm)")
plt.show()

plt.figure()
shap.summary_plot(lgb_shap_values, X_shap, plot_type="bar", show=False)
plt.title("LightGBM + cluster SHAP (abs bar)")
plt.show()

# CatBoost SHAP
cat_explainer = shap.TreeExplainer(cat_cluster_res["model"])
cat_shap_values = cat_explainer.shap_values(X_shap)
print("CatBoost SHAP computed: shape", np.array(cat_shap_values).shape)
plt.figure()
shap.summary_plot(cat_shap_values, X_shap, show=False)
plt.title("CatBoost + cluster SHAP (beeswarm)")
plt.show()

plt.figure()
shap.summary_plot(cat_shap_values, X_shap, plot_type="bar", show=False)
plt.title("CatBoost + cluster SHAP (abs bar)")
```

```
plt.show()

# XGBoost SHAP (use permutation explainer on a callable predict)
X_shap_np = X_shap.astype(float)
masker = shap.maskers.Independent(X_shap_np)
xgb_predict = lambda data: xgb_cluster_res["model"].predict(data)
xgb_explainer = shap.Explainer(xgb_predict, masker, algorithm="permutation")
xgb_shap_values = xgb_explainer(X_shap_np)
print("XGBoost SHAP computed: shape", np.array(xgb_shap_values.values).shape)
plt.figure()
shap.summary_plot(xgb_shap_values, X_shap_np, show=False)
plt.title("XGBoost + cluster SHAP (beeswarm)")
plt.show()

plt.figure()
shap.summary_plot(xgb_shap_values, X_shap_np, plot_type="bar", show=False)
plt.title("XGBoost + cluster SHAP (abs bar)")
plt.show()
```