model3

November 27, 2024

1 Model 3

1.1 Run XGBoost without any location data, compute averaged neighborhood residuals and then cluster based on these residuals

[ ]:
```
import pandas as pd
import numpy as np
import sklearn as sk
import random
from sklearn.neighbors import NearestNeighbors
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics.pairwise import euclidean_distances
from sklearn.preprocessing import MinMaxScaler
import xgboost as xgb
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import KFold
```

[ ]:
```
pd.set_option('display.max_columns', None)

# Load the dfs
train_df = pd.read_csv('train_processed.csv')
test_df = pd.read_csv('test_processed.csv')

# Find the lengths of the train and test dataframes
train_size = len(train_df)
test_size = len(test_df)

# Drop the latitude and longitude columns
lat_column = train_df['latitude']
long_column = train_df['longitude']
```

```python
train_df = train_df.drop(columns=['latitude', 'longitude'])

train_df
```

```
[ ]:        price  host_since  host_response_time  host_response_rate  \
     0          4       16578                 2.0               100.0
     1          3       19614                 1.0               100.0
     2          3       19204                 1.0               100.0
     3          0       15563                 1.0                99.0
     4          2       16427                 1.0                93.0
     …        …           …                   …                   …
     15691      5       18866                 1.0                99.0
     15692      0       17734                 1.0               100.0
     15693      5       17596                 1.0               100.0
     15694      5       17151                 1.0               100.0
     15695      1       18458                 1.0               100.0

            host_acceptance_rate  host_is_superhost  host_listings_count  \
     0                     100.0                  1                  2.0
     1                      98.0                  1                  1.0
     2                     100.0                  0                 52.0
     3                      23.0                  0                727.0
     4                      95.0                  0                707.0
     …                       …                  …                   …
     15691                  99.0                  1                 15.0
     15692                  67.0                  0                  2.0
     15693                  98.0                  1                 28.0
     15694                  96.0                  0               4494.0
     15695                  83.0                  1                  6.0

            host_total_listings_count  host_has_profile_pic  \
     0                            2.0                     1
     1                            1.0                     1
     2                           55.0                     1
     3                         1336.0                     1
     4                         2453.0                     1
     …                           …                     …
     15691                       15.0                     1
     15692                        2.0                     1
     15693                       35.0                     1
     15694                     4784.0                     1
     15695                        9.0                     1

            host_identity_verified  calculated_host_listings_count  accommodates  \
     0                           1                               1             4
     1                           1                               1             2
     2                           1                              52             2
```

```
3                           1                               719           1
4                           1                                73           2
…                       …                               …           …
15691                       1                                15           4
15692                       1                                 2           2
15693                       1                                16           3
15694                       1                               876           2
15695                       1                                 6           1

       bathrooms  bedrooms  beds  has_availability  availability_30  \
0            2.0       2.0   2.0                 1               12
1            1.0       1.0   2.0                 1               10
2            1.0       0.0   1.0                 1               17
3            1.5       4.0   1.0                 1                0
4            1.0       1.0   1.0                 1                4
…             …         …     …                 …               …
15691        1.0       1.0   4.0                 1                6
15692        1.0       1.0   1.0                 1                0
15693        1.0       1.0   1.0                 1               24
15694        1.0       1.0   1.0                 1                0
15695        1.0       1.0   1.0                 1               29

       availability_60  availability_90  availability_365  instant_bookable  \
0                   42               70                70             False
1                   20               49               324             False
2                   44               70               146              True
3                    0                0               111             False
4                   13               22               241              True
…                  …               …                …               …
15691               12               22               139              True
15692                0                0               216             False
15693               27               57               331              True
15694                0                0               247             False
15695               59               89               179             False

       minimum_nights  maximum_nights  number_of_reviews  \
0                  30            1125                 34
1                   1              29                 30
2                   1              29                  5
3                  30             365                  0
4                   1            1125                  0
…                 …             …                 …
15691               1             365                 32
15692              30            1125                  6
15693               1             365                  1
15694              31            1125                  0
15695              30             160                 21
```

```
       number_of_reviews_ltm  number_of_reviews_l30d  first_review  \
0                          5                       1         18014
1                         30                       6         19735
2                          5                       2         19901
3                          0                       0         20049
4                          0                       0         20049
…                        …                       …           …
15691                     13                       1         19372
15692                      2                       0         19463
15693                      0                       0         19590
15694                      0                       0         20049
15695                      0                       0         18565

       last_review  review_scores_rating  review_scores_accuracy  \
0            19945              5.000000                5.000000
1            19968              4.830000                4.870000
2            19952              4.600000                4.800000
3            20049              4.719393                4.742812
4            20049              4.719393                4.742812
…              …                   …                      …
15691        19949              4.940000                4.940000
15692        19679              4.330000                4.330000
15693        19590              5.000000                5.000000
15694        20049              4.719393                4.742812
15695        19559              4.810000                4.900000

       review_scores_cleanliness  review_scores_checkin  \
0                       4.970000                5.00000
1                       4.930000                4.80000
2                       4.200000                4.80000
3                       4.679642                4.82631
4                       4.679642                4.82631
…                          …                      …
15691                   4.910000                4.97000
15692                   4.170000                4.17000
15693                   5.000000                5.00000
15694                   4.679642                4.82631
15695                   4.950000                4.86000

       review_scores_communication  review_scores_location  \
0                         5.000000                4.710000
1                         4.900000                4.900000
2                         4.800000                4.800000
3                         4.808233                4.721844
4                         4.808233                4.721844
…                            …                      …
```

```
15691                    4.810000                4.000000
15692                    4.330000                4.000000
15693                    5.000000                5.000000
15694                    4.808233                4.721844
15695                    4.860000                4.620000

       review_scores_value  reviews_per_month  room_Entire home/apt  \
0                 4.940000           0.520000                     1
1                 4.630000           3.810000                     0
2                 4.200000           2.140000                     1
3                 4.609505           1.245801                     0
4                 4.609505           1.245801                     0
…                      …                  …                   …
15691             4.720000           1.600000                     1
15692             4.330000           0.350000                     0
15693             5.000000           0.080000                     0
15694             4.609505           1.245801                     1
15695             4.810000           0.450000                     0

       room_Hotel room  room_Private room  room_Shared room  Air conditioning  \
0                    0                  0                 0                 0
1                    0                  1                 0                 1
2                    0                  0                 0                 1
3                    0                  1                 0                 0
4                    1                  0                 0                 1
…                  …                …                …                …
15691                0                  0                 0                 1
15692                0                  1                 0                 0
15693                0                  1                 0                 1
15694                0                  0                 0                 1
15695                0                  1                 0                 1

       Kitchen  Dedicated workspace  Heating  Hot water  Refrigerator  \
0            1                    0        1          1             1
1            1                    0        0          1             0
2            0                    1        1          1             0
3            1                    1        1          1             1
4            0                    1        1          1             0
…          …                  …       …        …           …
15691        0                    1        1          1             1
15692        1                    1        0          1             1
15693        1                    0        1          0             0
15694        1                    0        1          1             1
15695        1                    1        1          1             0

       Free street parking  Self check-in  Shampoo  Washer
0                        1              1        1       0
```

```
1                        1              1        1       0
2                        0              1        1       0
3                        0              0        0       1
4                        0              1        1       0
…                    …            …      …      …
15691                    0              1        1       0
15692                    1              1        1       0
15693                    0              1        1       0
15694                    0              1        1       0
15695                    0              1        0       0

[15696 rows x 50 columns]
```

1.2 Train XGBoost without the location data

```
[ ]: # Split features and target
X = train_df.drop(columns=['price'])
y = train_df['price']

# Set up parameter grid for grid search
param_grid = {
    'n_estimators': [100, 500, 1000, 1500],
    'learning_rate': [0.01, 0.1, 0.2],
    'max_depth': [5, 7, 10, 15],
    'subsample': [0.8, 1.0],
    'colsample_bytree': [0.8, 1.0],
}

# Define the XGBoost Regressor
xgb_model = xgb.XGBRegressor()

# Perform GridSearchCV with 5-fold cross-validation
grid_search = GridSearchCV(
    estimator=xgb_model,
    param_grid=param_grid,
    cv=5,
    scoring='neg_mean_squared_error',  # Using neg_mean_squared_error for RMSE␣
 ↪calculation
    verbose=1,
    n_jobs=-1
)

# Fit the grid search
grid_search.fit(X, y)

# Retrieve the best parameters and model
best_params = grid_search.best_params_
```

```
print("Best Parameters:", best_params)

# Compute the best average RMSE
best_avg_rmse = np.sqrt(-grid_search.best_score_)
print(f"Best Average RMSE over folds: {best_avg_rmse:.4f}")

# Train the model with the best parameters
best_model = xgb.XGBRegressor(**best_params)
best_model.fit(X, y)

# Predict on the same data
y_pred = best_model.predict(X)

# Compute the differences
price_diff = y - y_pred

# Create a DataFrame with location columns, actual prices, predicted prices,␣
 ↪and differences
surprise_df = pd.DataFrame({
    'latitude': lat_column,
    'longitude': long_column,
    'price_difference': price_diff
})

surprise_df
```

```
Fitting 5 folds for each of 192 candidates, totalling 960 fits
```

```
/opt/anaconda3/envs/671/lib/python3.10/site-
packages/joblib/externals/loky/process_executor.py:752: UserWarning: A worker
stopped while some jobs were given to the executor. This can be caused by a too
short worker timeout or by a memory leak.
  warnings.warn(
```

```
Best Parameters: {'colsample_bytree': 0.8, 'learning_rate': 0.01, 'max_depth':
10, 'n_estimators': 1500, 'subsample': 0.8}
Best Average RMSE over folds: 0.8285
```

```
[ ]:        latitude  longitude  price_difference
     0      40.684560 -73.939870          0.081423
     1      40.638991 -73.965739          0.047657
     2      40.618810 -74.032380         -0.052605
     3      40.673970 -73.953990         -0.035475
     4      40.747180 -73.985390         -0.150916
     …            …          …               …
     15691  40.704777 -74.006425         -0.010690
     15692  40.881490 -73.910130         -0.111266
     15693  40.765440 -73.976508          0.113807
```

```
15694   40.735635 -74.005740          0.073095
15695   40.628170 -74.079650          0.120551

[15696 rows x 3 columns]
```

2 KNN to find average surprise of neighbor properties

[ ]:
```
def knn(surprise_df, k1):

    # Extract longitude and latitude for KNN
    coordinates = surprise_df[['longitude', 'latitude']].values

    # Initialize the KNN model (for k1 neighbors)
    knn = NearestNeighbors(n_neighbors=k1)

    # Fit the KNN model
    knn.fit(coordinates)

    # Find the k1 nearest neighbors for each property
    distances, indices = knn.kneighbors(coordinates)

    # Initialize an array to store the averaged price differences
    average_price_differences = np.zeros(surprise_df.shape[0])

    # For each property, calculate the average price difference of its k1␣
 ↪nearest neighbors
    for i in range(surprise_df.shape[0]):
        # Get the indices of the neighbors (including the property itself)
        neighbor_indices = indices[i]

        # Extract the price differences for the neighbors
        neighbor_price_differences = surprise_df.
 ↪iloc[neighbor_indices]['price_difference']

        # Compute the average price difference of the neighbors
        average_price_differences[i] = neighbor_price_differences.mean()

    # Append the column to surprise_df
    surprise_df['average_neighbor_price_difference'] = average_price_differences

    return surprise_df

# Examples
surprise0_df = knn(surprise_df, 500)

surprise0_df
```

```
[ ]:         latitude  longitude  price_difference  \
      0      40.684560 -73.939870          0.081423
      1      40.638991 -73.965739          0.047657
      2      40.618810 -74.032380         -0.052605
      3      40.673970 -73.953990         -0.035475
      4      40.747180 -73.985390         -0.150916
      …            …          …                …
      15691  40.704777 -74.006425         -0.010690
      15692  40.881490 -73.910130         -0.111266
      15693  40.765440 -73.976508          0.113807
      15694  40.735635 -74.005740          0.073095
      15695  40.628170 -74.079650          0.120551

             average_neighbor_price_difference
      0                              -0.005654
      1                              -0.057846
      2                              -0.039686
      3                              -0.001779
      4                               0.028033
      …                                    …
      15691                           0.044174
      15692                          -0.066673
      15693                           0.052566
      15694                           0.109589
      15695                          -0.054494

      [15696 rows x 4 columns]
```

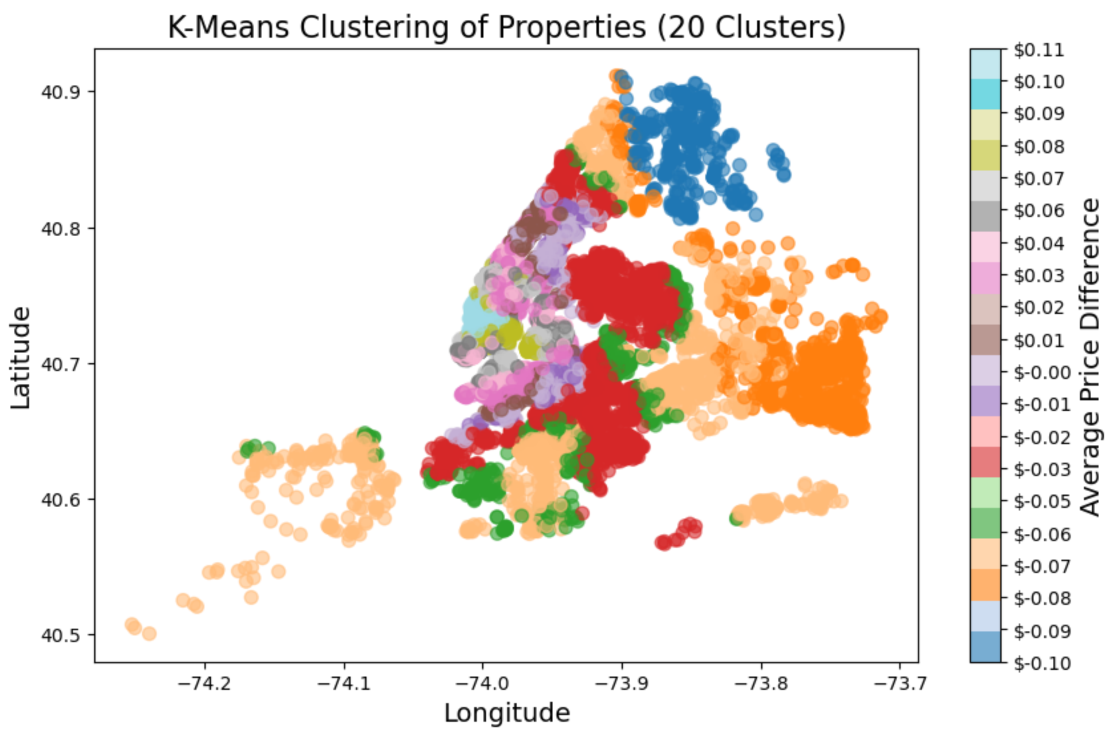
2.1 K Means to find clusters with similar amounts of surprise (price difference)

```
[ ]: def k_means(k2, surprise_df):
         # Apply KMeans clustering on the averaged price difference
         kmeans = KMeans(n_clusters=k2, random_state=42)
         surprise_df['cluster'] = kmeans.
      ↪fit_predict(surprise_df[['average_neighbor_price_difference']])  # Use only␣
      ↪the averaged price difference

         # Compute the average price difference for each cluster
         cluster_means = surprise_df.groupby('cluster')['price_difference'].mean()

         # Map the cluster average back to each row in the DataFrame
         surprise_df['cluster_avg_price_difference'] = surprise_df['cluster'].
      ↪map(cluster_means)

         return surprise_df

     def plot_cluster(df, k2):
```

```
    # Plot the clusters and their centroids
    plt.figure(figsize=(10, 6))

    # Define a colormap with more distinct colors
    cmap = plt.cm.get_cmap('tab20', k2)  # Use the 'tab20' colormap with k2␣
↪distinct colors

    # Create an array of colors for the scatter plot, corresponding to the␣
↪average price differences
    cluster_avg_diff = df.groupby('cluster')['cluster_avg_price_difference'].
↪mean()
    cluster_colors = df['cluster'].map(cluster_avg_diff)  # Map average price␣
↪difference to clusters

    # Scatter plot of the properties with color corresponding to average price␣
↪difference
    scatter = plt.scatter(
        df['longitude'],
        df['latitude'],
        c=cluster_colors,
        cmap=cmap,
        alpha=0.6,
        s=50,
        label='Properties'
    )

    # Create a colorbar based on the average price difference
    cbar = plt.colorbar(scatter)
    cbar.set_label('Average Price Difference', fontsize=14)

    # Create a list of labels for the colorbar corresponding to the average␣
↪price differences
    cbar.set_ticks(np.linspace(min(cluster_avg_diff), max(cluster_avg_diff),␣
↪k2))
    cbar.set_ticklabels([f'${avg_diff:.2f}' for avg_diff in np.
↪linspace(min(cluster_avg_diff), max(cluster_avg_diff), k2)])

    # Add labels and title
    plt.xlabel('Longitude', fontsize=14)
    plt.ylabel('Latitude', fontsize=14)
    plt.title(f'K-Means Clustering of Properties ({k2} Clusters)', fontsize=16)

    # Display the plot
    plt.show()

# Run an example with k2 = 20
```

```
surprise1_df = k_means(20, surprise0_df)
plot_cluster(surprise1_df, 20)
```

```
/var/folders/2f/7sxq51rj0xd8mgyvzzj2tgy80000gn/T/ipykernel_88848/2310280613.py:1
9: MatplotlibDeprecationWarning: The get_cmap function was deprecated in
Matplotlib 3.7 and will be removed in 3.11. Use ``matplotlib.colormaps[name]``
or ``matplotlib.colormaps.get_cmap()`` or ``pyplot.get_cmap()`` instead.
  cmap = plt.cm.get_cmap('tab20', k2)  # Use the 'tab20' colormap with k2
distinct colors
```



2.2 Retrain XGBoost with Neighborhood Price Difference Data

Homeade cross validation to incorporate knn and kmeans

```
[ ]: knn_neighbors = [100, 400, 600, 800]
     kmeans_clusters = [5, 10, 20, 40]

     # XGBoost hyperparameters to tune
     learning_rates = [0.01, 0.1, 0.3]
     max_depths = [8, 10]
     n_estimators = [500, 1000]

     n_folds = 5
```

```
# Generate a new random state dynamically
kf = KFold(n_splits=n_folds, shuffle=True, random_state=42)

# Placeholder for storing best results
best_params = None
best_score = float('inf')  # MSE should be minimized

train_df = train_df.merge(lat_column, left_index=True, right_index=True)
train_df = train_df.merge(lat_column, left_index=True, right_index=True)
```

```
[ ]: def grid_search_cv(train_df, surprise_df):
         global best_params, best_score
         k = 0
         # Iterate over all combinations of XGBoost hyperparameters
         for k1 in knn_neighbors:

             # Run KNN to get price difference averaged over neighbors
             surprise_df = knn(surprise_df, k1)

             for k2 in kmeans_clusters:

                 # Run K-means to get neighborhood price surprise
                 surprise_df = k_means(k2, surprise_df)

                 # drip the cluster_avg_price_difference column if it already exists
                 if 'cluster_avg_price_difference' in train_df.columns:
                     train_df = train_df.drop(columns='cluster_avg_price_difference')

                 train_df = train_df.
     ↪merge(surprise_df['cluster_avg_price_difference'], left_index=True,␣
     ↪right_index=True)

                 y = train_df['price']
                 X = train_df.drop(columns='price')

                 for lr in learning_rates:
                     for depth in max_depths:
                         for ne in n_estimators:

                             # Set XGBoost regression hyperparameters
                             params = {
                                 'objective': 'reg:squarederror',  # Regression␣
     ↪objective
                                 'eta': lr,
                                 'max_depth': depth,
                                 'n_estimators': ne,
                                 'verbosity': 0
```

```
                        }

                        # Store RMSE for each fold
                        fold_rmse = []

                        # 5-fold Cross Validation
                        for train_index, val_index in kf.split(X):
                            X_train, X_val = X.iloc[train_index], X.
↪iloc[val_index]
                            y_train, y_val = y.iloc[train_index], y.
↪iloc[val_index]

                            # Train XGBoost regression model
                            model = xgb.XGBRegressor(**params)
                            model.fit(X_train, y_train)

                            # Predict continuous values on validation set
                            y_pred_continuous = model.predict(X_val)

                            # Round predictions and clip to valid class range␣
↪(e.g., 0-5)
                            y_pred = np.round(y_pred_continuous).clip(0, 5)

                            # Calculate Root Mean Squared Error (RMSE) for this␣
↪fold
                            rmse = np.sqrt(mean_squared_error(y_val, y_pred))
                            fold_rmse.append(rmse)

                        # Calculate average RMSE for this set of hyperparameters
                        avg_rmse = np.mean(fold_rmse)
                        print(f"Iteration {k}: Avg RMSE = {avg_rmse}")
                        k += 1

                        # Update best parameters based on the lowest average␣
↪RMSE
                        if avg_rmse < best_score:
                            params['k1'] = k1
                            params['k2'] = k2
                            best_score = avg_rmse
                            best_params = params

                            # Print the current best hyperparameters and score
                            print(f"Current Best Parameters: {best_params}")
                            print(f"Current Best RMSE: {best_score}")

    # Print the best hyperparameters and score
    print(f"FINAL Best Parameters: {best_params}")
```

```python
    print(f"FINAL Best RMSE: {best_score}")

# Run the grid search cross-validation
grid_search_cv(train_df, surprise_df)
```

```
Iteration 0: Avg RMSE = 0.8196719497861606
Current Best Parameters: {'objective': 'reg:squarederror', 'eta': 0.01,
'max_depth': 8, 'n_estimators': 500, 'verbosity': 0, 'k1': 100, 'k2': 5}
Current Best RMSE: 0.8196719497861606
Iteration 1: Avg RMSE = 0.8050729502578055
Current Best Parameters: {'objective': 'reg:squarederror', 'eta': 0.01,
'max_depth': 8, 'n_estimators': 1000, 'verbosity': 0, 'k1': 100, 'k2': 5}
Current Best RMSE: 0.8050729502578055
Iteration 2: Avg RMSE = 0.814396550002256
Iteration 3: Avg RMSE = 0.8079308382332032
Iteration 4: Avg RMSE = 0.8012015455717096
Current Best Parameters: {'objective': 'reg:squarederror', 'eta': 0.1,
'max_depth': 8, 'n_estimators': 500, 'verbosity': 0, 'k1': 100, 'k2': 5}
Current Best RMSE: 0.8012015455717096
Iteration 5: Avg RMSE = 0.8028931528276999
Iteration 6: Avg RMSE = 0.8140583813437294
Iteration 7: Avg RMSE = 0.8147633916849081
Iteration 8: Avg RMSE = 0.8334882722675818
Iteration 9: Avg RMSE = 0.833342982669808
Iteration 10: Avg RMSE = 0.8450796068984721
Iteration 11: Avg RMSE = 0.8450796068984721
Iteration 12: Avg RMSE = 0.8153787055541392
Iteration 13: Avg RMSE = 0.8024831907502549
Iteration 14: Avg RMSE = 0.8146898031173819
Iteration 15: Avg RMSE = 0.8066338962931411
Iteration 16: Avg RMSE = 0.8018215623472003
Iteration 17: Avg RMSE = 0.8042782526165603
Iteration 18: Avg RMSE = 0.807393621534535
Iteration 19: Avg RMSE = 0.8068081975162459
Iteration 20: Avg RMSE = 0.8289496941085854
Iteration 21: Avg RMSE = 0.828758882662816
Iteration 22: Avg RMSE = 0.844653802149834
Iteration 23: Avg RMSE = 0.844653802149834
Iteration 24: Avg RMSE = 0.8119215100160977
Iteration 25: Avg RMSE = 0.796146860302196
Current Best Parameters: {'objective': 'reg:squarederror', 'eta': 0.01,
'max_depth': 8, 'n_estimators': 1000, 'verbosity': 0, 'k1': 100, 'k2': 20}
Current Best RMSE: 0.796146860302196
Iteration 26: Avg RMSE = 0.8111732265534481
Iteration 27: Avg RMSE = 0.8045135357739023
Iteration 28: Avg RMSE = 0.7997161555056181
Iteration 29: Avg RMSE = 0.8011896936065698
Iteration 30: Avg RMSE = 0.807036994188757
Iteration 31: Avg RMSE = 0.8067220524905648
Iteration 32: Avg RMSE = 0.8320483960904529
Iteration 33: Avg RMSE = 0.8324724806453216
Iteration 34: Avg RMSE = 0.8376349333147731
Iteration 35: Avg RMSE = 0.8376349333147731
Iteration 36: Avg RMSE = 0.8124799965868832
Iteration 37: Avg RMSE = 0.7993943706381972
Iteration 38: Avg RMSE = 0.8148032538112651
Iteration 39: Avg RMSE = 0.8083641250938232
Iteration 40: Avg RMSE = 0.7995246128782326
Iteration 41: Avg RMSE = 0.7991171587206564
Iteration 42: Avg RMSE = 0.8106276943421389
Iteration 43: Avg RMSE = 0.8105522445092722
Iteration 44: Avg RMSE = 0.8407613802204708
Iteration 45: Avg RMSE = 0.8408312792814323
Iteration 46: Avg RMSE = 0.8425320206473396
Iteration 47: Avg RMSE = 0.8425320206473396
Iteration 48: Avg RMSE = 0.8173235511611034
Iteration 49: Avg RMSE = 0.8024767079202938
Iteration 50: Avg RMSE = 0.8140997351959107
Iteration 51: Avg RMSE = 0.8064916989488502
Iteration 52: Avg RMSE = 0.8022561570084659
Iteration 53: Avg RMSE = 0.8021708608230789
Iteration 54: Avg RMSE = 0.8078524177747314
Iteration 55: Avg RMSE = 0.8071806037754377
Iteration 56: Avg RMSE = 0.8314512103466922
Iteration 57: Avg RMSE = 0.8319154495031347
Iteration 58: Avg RMSE = 0.8441409115938379
Iteration 59: Avg RMSE = 0.8441409115938379
Iteration 60: Avg RMSE = 0.8118794156797288
Iteration 61: Avg RMSE = 0.8000696335469388
Iteration 62: Avg RMSE = 0.8107593312341317
Iteration 63: Avg RMSE = 0.80157683827554
Iteration 64: Avg RMSE = 0.7971731767037429
Iteration 65: Avg RMSE = 0.7974556064764922
Iteration 66: Avg RMSE = 0.8076609534313878
Iteration 67: Avg RMSE = 0.808047736284599
Iteration 68: Avg RMSE = 0.8292854955311165
Iteration 69: Avg RMSE = 0.8294414152409679
Iteration 70: Avg RMSE = 0.8407490025364759
Iteration 71: Avg RMSE = 0.8407490025364759
Iteration 72: Avg RMSE = 0.811890498576037
Iteration 73: Avg RMSE = 0.8010669258978196
Iteration 74: Avg RMSE = 0.8094657479321606
Iteration 75: Avg RMSE = 0.8001217858773251
Iteration 76: Avg RMSE = 0.7941839122471276
Current Best Parameters: {'objective': 'reg:squarederror', 'eta': 0.1,
'max_depth': 8, 'n_estimators': 500, 'verbosity': 0, 'k1': 400, 'k2': 20}
```

```
Current Best RMSE: 0.7941839122471276
Iteration 77: Avg RMSE = 0.7960687238299863
Iteration 78: Avg RMSE = 0.805284457752349
Iteration 79: Avg RMSE = 0.8040119957243602
Iteration 80: Avg RMSE = 0.8370148930713514
Iteration 81: Avg RMSE = 0.837049093296975
Iteration 82: Avg RMSE = 0.8293985901234692
Iteration 83: Avg RMSE = 0.8293985901234692
Iteration 84: Avg RMSE = 0.8117591869852621
Iteration 85: Avg RMSE = 0.7956693062145861
Iteration 86: Avg RMSE = 0.8091610770379563
Iteration 87: Avg RMSE = 0.8014610753457692
Iteration 88: Avg RMSE = 0.7969853820795991
Iteration 89: Avg RMSE = 0.8003705371319386
Iteration 90: Avg RMSE = 0.8031284832399599
Iteration 91: Avg RMSE = 0.8036417391152234
Iteration 92: Avg RMSE = 0.8298996919767975
Iteration 93: Avg RMSE = 0.8298601748709459
Iteration 94: Avg RMSE = 0.8372737765984957
Iteration 95: Avg RMSE = 0.8372737765984957
Iteration 96: Avg RMSE = 0.8152324612515456
Iteration 97: Avg RMSE = 0.799856583134755
Iteration 98: Avg RMSE = 0.8116485184037975
Iteration 99: Avg RMSE = 0.8011770921059445
Iteration 100: Avg RMSE = 0.8008022834711065
Iteration 101: Avg RMSE = 0.8008885510446768
Iteration 102: Avg RMSE = 0.8097810255134708
Iteration 103: Avg RMSE = 0.8096744010652321
Iteration 104: Avg RMSE = 0.8338100835599423
Iteration 105: Avg RMSE = 0.8345371341499138
Iteration 106: Avg RMSE = 0.8355352139004678
Iteration 107: Avg RMSE = 0.8355352139004678
Iteration 108: Avg RMSE = 0.8133409929760372
Iteration 109: Avg RMSE = 0.8007463660893333
Iteration 110: Avg RMSE = 0.807757420261475
Iteration 111: Avg RMSE = 0.80190100617738
Iteration 112: Avg RMSE = 0.7949487263878179
Iteration 113: Avg RMSE = 0.7967790679279999
Iteration 114: Avg RMSE = 0.8007021499114497
Iteration 115: Avg RMSE = 0.8006639468838804
Iteration 116: Avg RMSE = 0.8301905420317144
Iteration 117: Avg RMSE = 0.8303486288391675
Iteration 118: Avg RMSE = 0.843711054837317
Iteration 119: Avg RMSE = 0.843711054837317
Iteration 120: Avg RMSE = 0.8105023362740486
Iteration 121: Avg RMSE = 0.7943315732055891
Iteration 122: Avg RMSE = 0.8074503178882052
Iteration 123: Avg RMSE = 0.8005559042196542
Iteration 124: Avg RMSE = 0.7952508156677454
Iteration 125: Avg RMSE = 0.7976440826020922
Iteration 126: Avg RMSE = 0.8037859105548255
Iteration 127: Avg RMSE = 0.8045866264515343
Iteration 128: Avg RMSE = 0.8341349518585167
Iteration 129: Avg RMSE = 0.8340520039156754
Iteration 130: Avg RMSE = 0.839280891067132
Iteration 131: Avg RMSE = 0.839280891067132
Iteration 132: Avg RMSE = 0.8101856990557297
Iteration 133: Avg RMSE = 0.7963886680228194
Iteration 134: Avg RMSE = 0.8091010670797676
Iteration 135: Avg RMSE = 0.8013796605790222
Iteration 136: Avg RMSE = 0.7899443813479874
Current Best Parameters: {'objective': 'reg:squarederror', 'eta': 0.1,
'max_depth': 8, 'n_estimators': 500, 'verbosity': 0, 'k1': 600, 'k2': 40}
Current Best RMSE: 0.7899443813479874
Iteration 137: Avg RMSE = 0.7905167235403568
Iteration 138: Avg RMSE = 0.8049479236602133
Iteration 139: Avg RMSE = 0.8047063505538119
Iteration 140: Avg RMSE = 0.8319754261506119
Iteration 141: Avg RMSE = 0.8320181801363151
Iteration 142: Avg RMSE = 0.833437918310594
Iteration 143: Avg RMSE = 0.833437918310594
Iteration 144: Avg RMSE = 0.8096741363932883
Iteration 145: Avg RMSE = 0.7966604940696771
Iteration 146: Avg RMSE = 0.8110199513140746
Iteration 147: Avg RMSE = 0.8043837129325215
Iteration 148: Avg RMSE = 0.7994756374771098
Iteration 149: Avg RMSE = 0.8011873643245426
Iteration 150: Avg RMSE = 0.8057235800450965
Iteration 151: Avg RMSE = 0.805922873424587
Iteration 152: Avg RMSE = 0.8279826147642055
Iteration 153: Avg RMSE = 0.8278310481872733
Iteration 154: Avg RMSE = 0.837147049985273
Iteration 155: Avg RMSE = 0.837147049985273
Iteration 156: Avg RMSE = 0.8083787458226712
Iteration 157: Avg RMSE = 0.7968957435621015
Iteration 158: Avg RMSE = 0.802899681045741
Iteration 159: Avg RMSE = 0.7931667765458326
Iteration 160: Avg RMSE = 0.7974575585235949
Iteration 161: Avg RMSE = 0.7968608002424994
Iteration 162: Avg RMSE = 0.7995087564935275
Iteration 163: Avg RMSE = 0.799676521991754
Iteration 164: Avg RMSE = 0.8288838629185774
Iteration 165: Avg RMSE = 0.8286532438657532
Iteration 166: Avg RMSE = 0.833882049616378
Iteration 167: Avg RMSE = 0.833882049616378
Iteration 168: Avg RMSE = 0.8096440467551245
Iteration 169: Avg RMSE = 0.7980077146942863
Iteration 170: Avg RMSE = 0.8067302692644297
Iteration 171: Avg RMSE = 0.8011613597119682
Iteration 172: Avg RMSE = 0.794759541044524
Iteration 173: Avg RMSE = 0.7942651343777468
Iteration 174: Avg RMSE = 0.8065753304778145
Iteration 175: Avg RMSE = 0.805862036271639
Iteration 176: Avg RMSE = 0.8240702895198488
Iteration 177: Avg RMSE = 0.8243763718603262
Iteration 178: Avg RMSE = 0.8357572189351299
Iteration 179: Avg RMSE = 0.8357572189351299
Iteration 180: Avg RMSE = 0.8091610036035888
Iteration 181: Avg RMSE = 0.7973469364171761
Iteration 182: Avg RMSE = 0.805387630916331
Iteration 183: Avg RMSE = 0.7954899365943691
Iteration 184: Avg RMSE = 0.7962962892481981
Iteration 185: Avg RMSE = 0.7966583677680116
Iteration 186: Avg RMSE = 0.8020001395203098
Iteration 187: Avg RMSE = 0.8020782296972975
Iteration 188: Avg RMSE = 0.8287394732037552
Iteration 189: Avg RMSE = 0.8288183852063741
Iteration 190: Avg RMSE = 0.8376743601620351
Iteration 191: Avg RMSE = 0.8376743601620351
FINAL Best Parameters: {'objective': 'reg:squarederror', 'eta': 0.1,
'max_depth': 8, 'n_estimators': 500, 'verbosity': 0, 'k1': 600, 'k2': 40}
FINAL Best RMSE: 0.7899443813479874
```

[ ]:
```
 # Run KNN to get price difference averaged over neighbors
surprise_df = knn(surprise_df, best_params['k1'])

# Run K-means to get neighborhood price surprise
surprise_df = k_means(best_params['k2'], surprise_df)

train_df = train_df.merge(surprise_df['cluster_avg_price_difference'],␣
 ↪left_index=True, right_index=True)

y = train_df['price']
X = train_df.drop(columns='price')

params = {
        'objective': 'reg:squarederror',  # regression
        'eta': best_params['eta'],
        'max_depth': best_params['max_depth'],
        'n_estimators': best_params['n_estimators'],
        'verbosity': 0
    }
```

```
# Train XGBoost model
model = xgb.XGBRegressor(**params)
model.fit(X, y)
```

[ ]:
```
# Get feature importance scores
feature_importances = model.get_booster().get_score(importance_type='gain')  #␣
↪'weight', 'gain', 'cover', etc.

# Convert to a sorted list of tuples for better readability
sorted_importances = sorted(feature_importances.items(), key=lambda x: x[1],␣
↪reverse=True)

print("Feature Importances (sorted by gain):")
for feature, importance in sorted_importances:
    print(f"{feature}: {importance}")
```

[ ]:
```
# Get clusters for test data
```

[ ]:
```
id_col = test_df['id']
predict_df = test_df.drop(columns='id')

pred_cont = model.predict(predict_df)
y_pred = np.round(pred_cont).clip(0, 5)

predicted_prices = y_pred.astype(int)
ids = id_col.astype(int)

# Create a DataFrame with 'id' and 'price' columns
result_df = pd.DataFrame({
    'id': ids,
    'price': predicted_prices
})

# Save the DataFrame to a CSV file
result_df.to_csv('predictions3.csv', index=False)
```