Real Estate Market Analysis of Melbourne
A Statistical Perspective

Koldyshev Ilia, Negri Giacomo, Omizzolo Matteo,
Ruzzante Leonardo

January 11, 2025

1 Introduction

We explored a dataset containing housing sales data in Melbourne for 2016 and 2017. This dataset is extensive, comprising 13,580 observations. It includes information such as house locations, characteristics, and sales details, such as the real estate operator, sale typology, and date of sale. After a preliminary analysis, we decided to exclude the columns `BuildingArea` and `YearBuilt` due to their high proportion of missing values: 6,450 and 5,330, respectively. Furthermore, we identified 226 possibly relisted houses by detecting duplicate matching of addresses and suburb. These entries were excluded from further analysis. Interestingly, no sales were recorded in January 2017, an anomaly likely attributed to data collection gaps, business trends, or other unknown commercial factors.

2 Exploratory Data Analysis (EDA)

The dataset exhibits notable concentration across various dimensions, including spatial distribution, housing types, sellers, and sales methods. These insights are crucial for comprehending pricing dynamics and preparing the dataset for further analysis.

Missing values and imputations Besides the `BuildingArea` and `YearBuilt` columns another column presented a significant amount of missing values: `CouncilArea`. The number of such values amounted to 1,370 in total. Given the missing value proportion to the total and the fact that they were concentrated in specific section of the original dataset, we decided to impute these value and retain the `CouncilArea` column. To address the missing entries, we imputed them using the most frequent value within groups defined by `RegionName`, `Postcode`, and `Suburb`, since these fields were fully populated. This approach successfully reduced the missing values from 1,370 to just 7. These last remaining `NaN` entries were removed from the dataset. The final cleaning step involved addressing the `Landsize` variable. Due to the presence of outliers, we excluded records with a value of 1000 or more, which are not representative of real city houses, but more of countryside residences, resulting in a final dataset of 12614 records.

Price The price distribution is asymmetric, exhibiting a positive skew, with an average price of 1,070,406 AUD and a median price of 906,000 AUD. On the other hand its log-transformed counterpart (Figure 1 Top-Left) closely approximates a normal distribution, with mean 13.747 and median 13.717. This represent a key transformation of data which will enable us later to improve the model performance, allowing control of skewness and mitigation of the influence of extreme values, such as multimillion dollars house sales (the maximum value recorded is 9,000,000 AUD).

Concentration Two primary types of concentration emerged: spatial and transactional. Spatial concentration is evident in sales predominantly clustered within specific councils, regions, and suburbs. For instance, the councils of Moreland, Boroondara, and Moonee Valley together account for 17% (2,323 sales) of total sales, while three out of eight regions comprise 85% of sales. This spatial pattern is further reflected in varying prices across areas, as illustrated in the heatmap (Figure 1, Top-Right). The Pareto analysis (Figure 1, Bottom-Left) underscores this concentration by displaying cumulative sales and transaction volumes by `Suburb`, sorted from largest to smallest. These patterns suggest that spatial variables, such as `Regionname` and `CouncilArea`, will be critical in predicting the `LogPrice`. Transactional concentration, on the other

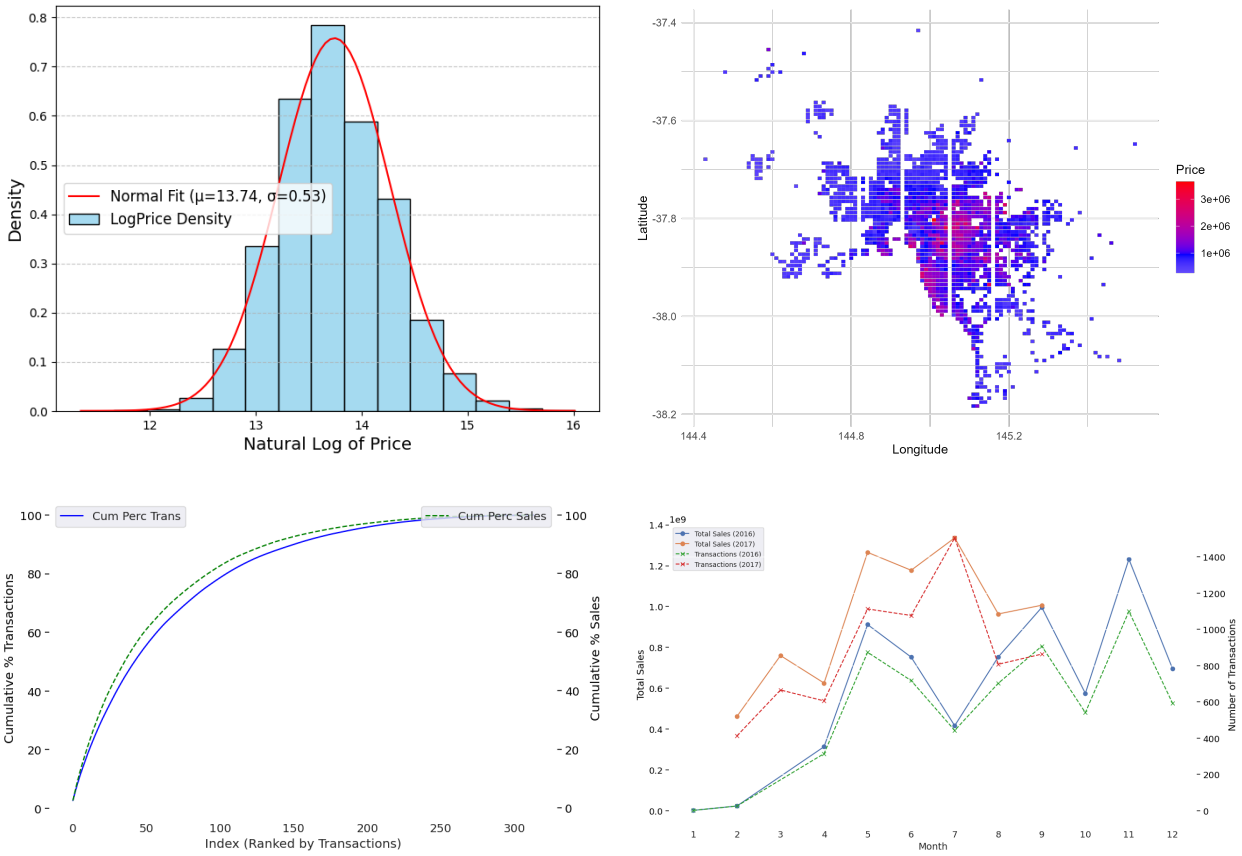

Figure 1: (Top-Left) Histogram of density distribution of `LogPrice`; (Top-Right) Heatmap of sales prices in Melbourne; (Bottom-Left) Cumulative percentage of total sales vs total transactions grouped by `Suburb`; (Bottom-Right) Monthly total sales and number of transactions across years.

hand, reveals that a small number of sellers and selling methods dominate the market. For instance, the top real estate agency accounts for approximately 12% of all Melbourne sales.

Correlation Analysis A notable positive correlation (Figure 3) exists between `LogPrice` and the number of `Rooms` ($r = 0.54$), while `Distance` to Central Business District (CBD) exhibits a consistent negative correlation with LogPrice across its various transformations (standard: $r = -0.18$, logarithmic: $r = -0.14$, squared: $r = -0.19$). Additionally, `Rooms` strongly correlates with `Bedroom2` ($r = 0.95$), moderately with `Bathroom` ($r = 0.59$), and Landsize ($r = 0.55$), reflecting the interconnected nature of housing attributes.

Temporal Analysis Sales exhibit clear seasonal and yearly patterns (Figure 1 Bottom-Right), with an overall Year-over-Year increase in transaction volume and total sales. Average monthly house prices show fluctuations (Figure 4), likely influenced by high-value purchases, as evidenced by the widening gap between monthly transaction volumes and total sales in 2017. Notably, July 2017 stands out as the top-performing month in terms of volume and total sales, despite being in winter (June to August in Australia), typically a slow period for real estate, as observed in 2016. Interestingly, July also recorded the lowest average sale price of 900,784 AUD. Following this peak, sales seem to return to the previous trend, aligning with the seasonal upswing of spring (September to November). These temporal dynamics suggest that time of year, particularly July, may play a significant role in predicting prices.

3 KNN Model

The predictive model employed in this study is a K-Nearest Neighbors (KNN) Regressor, implemented using the `KNeighborsRegressor` class from the Scikit-Learn library. The model estimates the target variable (`LogPrice`) by leveraging the k-nearest instances in the training dataset, where k is a hyperparameter optimized for performance. Given the predominantly geospatial nature of the data, KNN was selected as

it is distance-based, enabling the model to naturally incorporate spatial clustering and leverage the inherent characteristics of the dataset for prediction and intuitive interpretation. This section details the KNN regressor's configuration within the pipeline and its interaction with hyperparameter tuning and feature selection.

Prediction Mechanism The KNN Regressor predicts the target value y for a query instance $\mathbf{x}$ by aggregating the values of the k-nearest neighbors, defined using a distance metric $d(\mathbf{x}, \mathbf{x}_i)$. The prediction $\hat{y}$ is computed using distance-based weights (Equation 1), which prioritize closer neighbors to reduce the impact of noise, as follows:

$$\hat{y} = \frac{\sum_{i=1}^{k} \frac{1}{d(\mathbf{x},\mathbf{x}_i)} y_i}{\sum_{i=1}^{k} \frac{1}{d(\mathbf{x},\mathbf{x}_i)}} \tag{1}$$

Here, $[i = 1 \text{ to } k]$ or $\mathcal{N}(\mathbf{x})$ represents the set of k-nearest neighbors, and $d(\mathbf{x}, \mathbf{x}_i)$ is the distance between $\mathbf{x}$ and training instance $\mathbf{x}_i$. The choice of k, the weighting scheme, and the distance metric are tuned to balance noise resistance and variance control, optimizing the model's predictive performance.

Specific Configuration The KNN model configuration and its hyperparameters were jointly optimized as part of a pipeline, including preprocessing and feature selection, through Bayesian optimization to identify the optimal set for robust predictive performance. The specific configurations determined for the KNN model are as follows:

- **Distance Metric** The Manhattan distance (L_1-norm) was selected:

$$d(\mathbf{x}, \mathbf{x}_i) = \sum_{j=1}^{p} |x_j - x_{i,j}|, \tag{2}$$

  where p denotes the number of selected features. The Manhattan metric was chosen for its robustness in high-dimensional spaces and its ability to handle features with varying distributions effectively, particularly after standardization and encoding.

- **Number of Neighbors (k)** The optimal value of k was determined to be 31, balancing the tradeoff between underfitting and overfitting, thus achieving an optimal bias-variance tradeoff. A smaller k would increase sensitivity to local noise, potentially leading to high variance, while a larger k, incorporating more neighbors, would smooth predictions but lead to a higher bias.

- **Weighting Scheme** A distance-based weighting scheme was selected to prioritize closer neighbors:

$$w_i = \frac{1}{d(\mathbf{x}, \mathbf{x}_i)}, \tag{3}$$

  where w_i is the weight assigned to the i -th neighbor and $d(\mathbf{x}, \mathbf{x}_i)$ is the Manhattan distance. This scheme reduces the impact of distant or less relevant observations, improving robustness against noise and irrelevant neighbors.

Handling Noise and Variance The KNN Regressor addresses noise and variance through careful preprocessing, model design, and hyperparameter optimization. Noise is mitigated by weighting closer neighbors more heavily, minimizing the impact of distant or irrelevant points, and by preprocessing steps like standardization and encoding to ensure proportional contributions from all features. Outliers are removed during preprocessing to maintain consistent neighbor aggregation. Variance is controlled by optimizing the number of neighbors ($k = 31$) via Bayesian search, achieving a bias-variance tradeoff, where smaller k values increase sensitivity to noise, and larger k values smooth predictions. Feature selection further enhances stability by reducing dimensionality and retaining predictors with significant contributions to the target variable.

Bayesian Hyperparameter Optimization with Temporal Validation Bayesian optimization, implemented using the `BayesSearchCV` class from the Scikit-Optimize library, was used to tune the hyperparameters of the KNN regressor and feature selection process. This method is well-suited for high-dimensional search spaces, where hyperparameter interactions significantly influence performance. To account for the temporal nature of the data, `TimeSeriesSplit` was employed for cross-validation, ensuring that validation always occurred on future data. Further details on the Bayesian optimization process, including the mathematical framework, parameter tuning, and configuration for this study, are provided in the appendix B.

Feature Selection and Integration with Model Fitting Feature selection was performed using the `SelectKBest` method, which ranks features based on their correlation with the target variable using the F-statistic from a regression model. The number of features (k_f) was tuned alongside KNN hyperparameters during Bayesian optimization, ensuring an optimal interplay between feature selection and model fitting. To validate the robustness of `SelectKBest`, alternative scoring functions and methods such as stepwise selection and tree-based selectors were tested. However, `SelectKBest` with regression as the scoring method was chosen for its efficiency and strong performance, particularly given the observed linearity between features and the target. Further information on the feature selection process can be found in the appendix C.

4 Results

The KNN model, employing `manhattan` distance with `distance`-based weighting, $k = 31$ neighbors, and 17 features selected by `SelectKBest`, achieved the best performance in predicting `LogPrice`.

Comparison of Metrics Table 4 summarizes the predictive performance of the KNN model and a baseline Linear Regression model on `LogPrice`. Both models were trained on the same dataset using an identical preprocessing pipeline to ensure a fair comparison. The KNN model outperformed Linear Regression across all metrics, demonstrating its superior ability to capture non-linear and local patterns inherent in the data.

Metric	KNN Model	Linear Regression	Percentage Improvement
MAE	0.16	0.22	27.3%
MSE	0.05	0.09	44.4%
RMSE	0.21	0.29	27.6%
R^2	0.81	0.65	24.9%

Table 1: Performance Comparison of KNN and Linear Regression on `LogPrice` with Percentage Improvement

Feature Analysis Both models relied on spatial features (`Distance`, `Regionname_Northern Metropolitan`, `Regionname_Southern Metropolitan`, `Regionname_Western Metropolitan`, `Regionname_Western Victoria`, `Postcode`), property characteristics (`Rooms`, `Bedroom2`, `Bathroom`, `Car`, `Landsize`, `Type_h`, `Type_u`), temporal features (`Month_7`), and auction methods (`Method_S`, `Method_SP`). Additionally, the KNN model included `CouncilArea`, reflecting its ability to capture localized effects, while the Linear Regression model incorporated `SellerG`, highlighting the impact of seller concentration and seller-based pricing strategies. These features align with insights from the exploratory data analysis, where spatial variables highlighted geographic price variations, property characteristics reflected size and functionality, and temporal features captured seasonal trends. Overall, the models identified 17 key predictors that effectively summarize the dataset, aligning with previously observed trends and patterns, and reducing the selected features, post-encoding, from 43 to 17 for both models.

Partial Dependence Plots PDPs illustrate the marginal effects of selected numerical features on predicted housing prices as modeled by the KNN algorithm. These plots are generated by varying one feature over its range while keeping all other features constant and averaging the model's predictions across the dataset. This approach isolates the effect of the selected feature on the target variable, revealing its specific relationship with the predictions. A complete set of PDPs and a more in-depth explanation can be found in the appendix D; here, we focus on two key features. For the former, `Rooms` (Figure 2 Top-Left), the plot shows a steep initial rise, indicating a significant marginal effect up to 4 rooms, after which the curve flattens, suggesting that additional rooms contribute little to the predicted price. This can be explained by the average Australian household size of 2.56 people, which limits the practical value of extra rooms. For the latter, `Distance` (Figure 2 Top-Right), a clear negative relationship emerges, with a consistent downward slope that suggests an almost linear trend across the feature range, with minimal flattening beyond 14 kilometers.

Decision Boundary Analysis This analysis (Figure 2 Bottom) visualizes the decision boundaries of the KNN regression model in the feature space defined by `Rooms` and `Distance`. Due to the applied scaling, negative values are present, and the range, approximately spanning [-3,3], reflects the normality of the distribution when the `StandardScaler` is applied. Scatter points correspond to the standardized training data. It was found that properties with more rooms generally have higher predicted prices, while

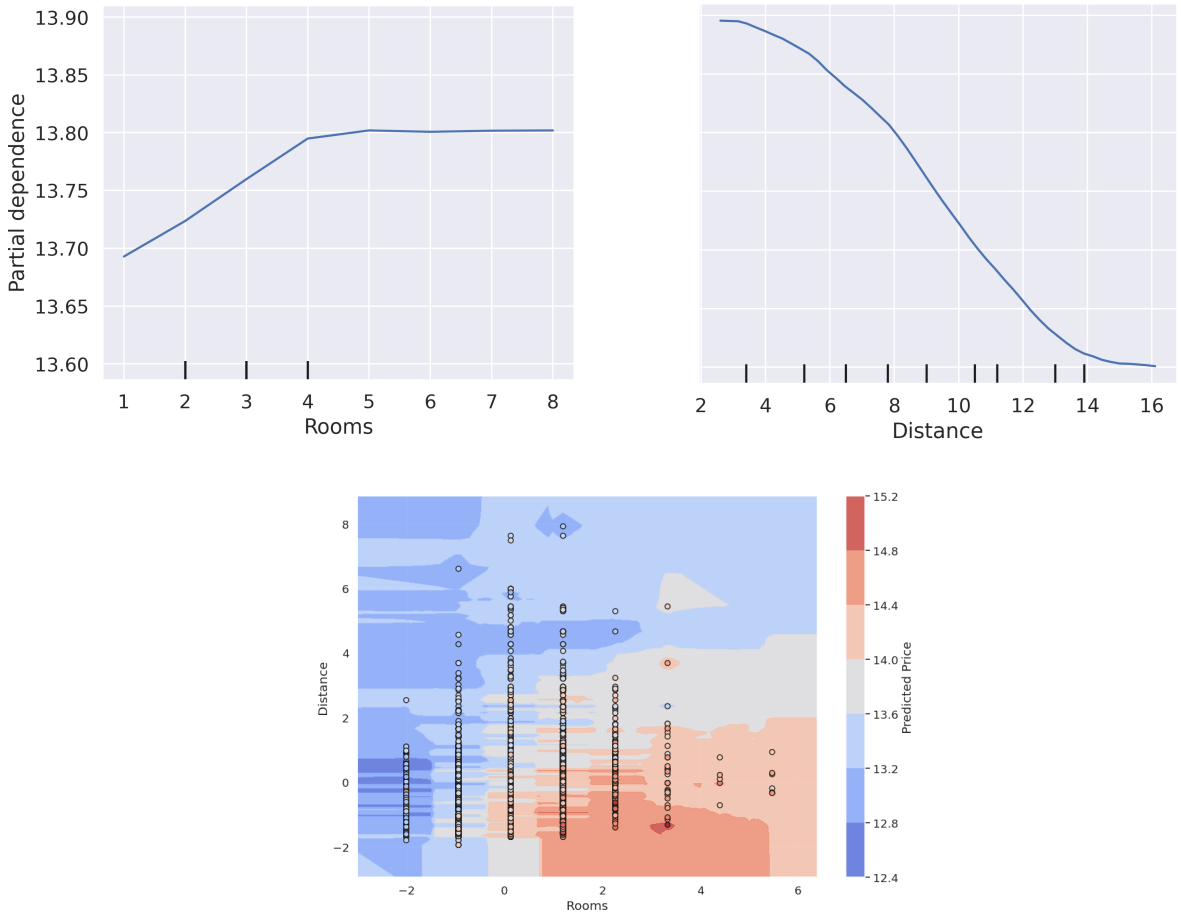

Figure 2: (Top-Left) PDP of `Rooms`; (Top-Right) PDP of `Distance`; (Bottom) Decision Boundaries for `Rooms` and `Distance`

properties farther from the CBD tend to have lower predicted prices. Notably, `Rooms` moderates the effect of `Distance`, as proximity to the CBD alone does not drive high prices unless paired with a sufficient number of rooms, likely representing larger and more desirable properties. The non-linear boundaries highlight KNN's reliance on local relationships and the effect of the distance-based weighting scheme, producing smooth transitions between price zones. However, overlapping boundaries in certain regions (2-4 rooms scaled) suggest interactions or limitations in capturing complex relationships. These insights provide a nuanced understanding of how property size, proximity, and price dynamics interact, aiding in the interpretation of market segmentation and pricing patterns.

5 Conclusions

This study explored spatial, temporal, and transactional patterns in Melbourne housing prices, employing a K-Nearest Neighbors (KNN) Regressor to model these dynamics. Our exploratory analysis identified spatial concentration in specific regions and councils, transactional concentration among sellers and methods, and clear temporal patterns, with July 2017 emerging as a notable anomaly in both sales volume and pricing trends. These insights informed feature selection and preprocessing decisions, ensuring that key predictors like `Rooms` (positively correlated with `LogPrice`) and `Distance` (negatively correlated) were incorporated.

The KNN model, optimized using Bayesian hyperparameter tuning, demonstrated robust performance, achieving an R^2 score of 0.81 and outperforming Linear Regression across all metrics, with a 27.3% improvement in MAE and a 44.4% improvement in MSE. Feature selection reduced the dataset from 43 features to 17, enhancing model interpretability while preserving predictive accuracy. Decision boundary and partial dependence analyses further highlighted the non-linear and local effects of spatial and property features on pricing, showcasing the KNN model's ability to capture complex relationships.

Appendix

A EDA: additional graphs

The first graph displays the correlation among all variables. Specifically, variables with names ending in `_encoded` were label-encoded as they were originally categorical (of type `object`).

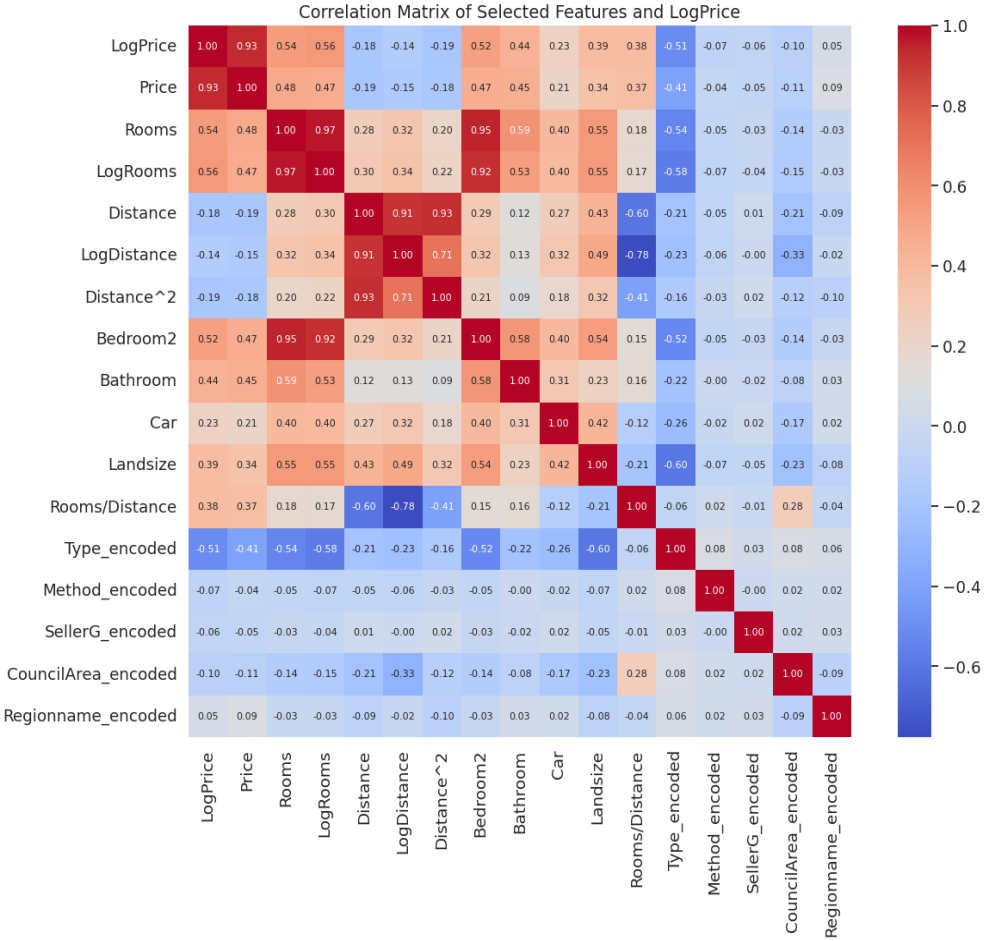

Figure 3: Correlation matrix of variables

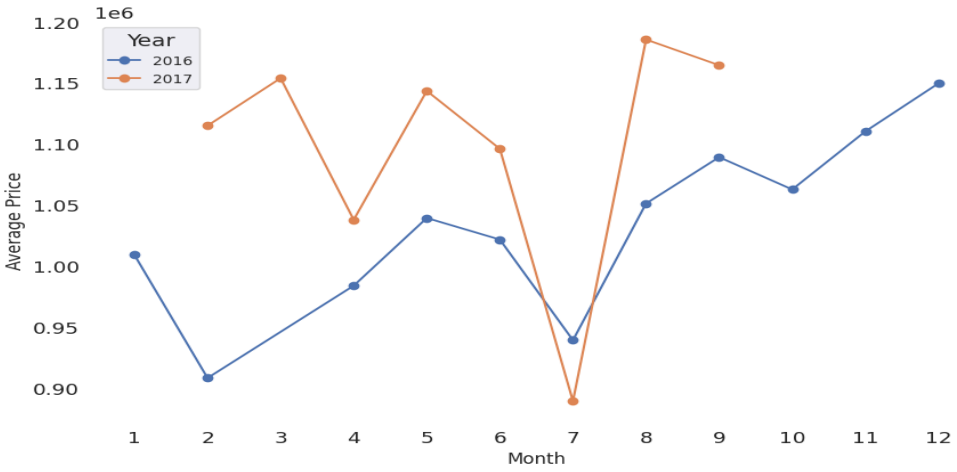

Figure 4: Average house price per year and month

B Bayesian Hyperparameter Optimization with Temporal Validation

Bayesian optimization, implemented using the `BayesSearchCV` class from the Scikit-Optimize library, was employed to systematically tune the hyperparameters of the KNN regressor and the feature selection process. This approach, well-suited for high-dimensional search spaces, effectively captured interactions between hyperparameters that significantly influenced performance. To preserve the temporal structure of the data and avoid data leakage, the `TimeSeriesSplit` strategy was used during inner training, ensuring validation only on future data and maintaining consistency with the temporal nature of predictions.

B.1 Time Series Cross-Validation

Given the temporal dependency in the dataset, the `TimeSeriesSplit` method was employed for cross-validation. This method:

- Chronologically splits the training and validation sets to ensure future data is not used for training.
- Simulates real-world prediction scenarios by training on earlier data and validating on subsequent data.

This strategy minimizes bias, preserves temporal order, and provides realistic performance evaluation during Bayesian optimization.

B.2 Hyperparameters Tuned

Bayesian optimization targeted the tuning of key hyperparameters, with the initial prior distribution of the hyperparameters assumed to be uniform, reflecting the absence of prior knowledge about their optimal values. The process dynamically adjusted the search space and iteration count based on the progress of the optimization. Key adjustments included:

- Expanding the grid when a parameter value approached the boundary of its range, enabling broader exploration of the search space.
- Adding iterations when convergence was detected near the iteration limit, ensuring the surrogate model was thoroughly refined.

The optimization process concluded when further iterations failed to yield significant performance improvements, with 50 iterations identified as the optimal threshold. Below are the search spaces optimally explored:

Feature Selection (k_f)

- $k_f \in [3, 43]$: The number of features selected by `SelectKBest`. The total number of features was expanded due to preprocessing, including one-hot encoding, which transformed categorical variables into multiple binary columns.

KNN Model Parameters

- n_neighbors $\in [1, 200]$: Number of neighbors considered.
- Distance metric: {manhattan, euclidean}.
- Weighting scheme: {uniform, distance}.

B.3 Mathematical Framework of Bayesian Optimization

Bayesian optimization seeks to minimize the objective function $f(\boldsymbol{\theta})$, where $\boldsymbol{\theta}$ represents the hyperparameter configuration, and f is the performance metric ($-$MAE).

Surrogate Model The objective function is approximated by a Gaussian Process (GP):

$$f(\boldsymbol{\theta}) \sim \mathcal{GP}(\mu(\boldsymbol{\theta}), \kappa(\boldsymbol{\theta}, \boldsymbol{\theta}')),$$

where $\mu(\boldsymbol{\theta})$ is the mean function representing the expected value of f, and $\kappa(\boldsymbol{\theta}, \boldsymbol{\theta}')$ is the covariance kernel quantifying uncertainty in predictions. The GP allows Bayesian optimization to model the objective function efficiently, even in high-dimensional spaces.

Acquisition Function The Expected Improvement (EI) function identifies the next hyperparameter configuration:

$$\mathrm{EI}(\boldsymbol{\theta}) = \mathbf{E}\left[\max(0, f(\boldsymbol{\theta}^*) - f(\boldsymbol{\theta}))\right],$$

where $f(\boldsymbol{\theta}^*)$ is the best observed value. By maximizing EI, the algorithm balances exploration (searching less-certain areas) and exploitation (refining around known optimal regions).

Iterative Refinement At each iteration:

- The GP model is updated with results from previous configurations.
- The acquisition function selects the next candidate configuration, $\boldsymbol{\theta}_{\text{next}}$.
- The objective function is evaluated at $\boldsymbol{\theta}_{\text{next}}$, refining the surrogate model for subsequent iterations.

This iterative process continues until convergence or the predefined iteration limit of 50 is reached.

B.4 Application in This Study

In this study, Bayesian optimization was implemented with the following key configurations:

- **Implementation:** `BayesSearchCV` integrated with the Scikit-Learn API.
- **Initial Sampling:** Latin Hypercube Sampling (LHS) for diverse and representative initialization of the search space. The initial prior distribution of the hyperparameters was assumed to be uniform, reflecting the absence of prior knowledge about their optimal values.
- **Dynamic Adjustments:** Adaptive expansion of the parameter space and iteration count to ensure comprehensive exploration.
- **Temporal Validation:** `TimeSeriesSplit` respected the chronological order of the data, aligning with the temporal dependency inherent in the dataset.

B.5 Outcome of Bayesian Optimization

The optimal hyperparameter configuration identified was:

- $k_f = 17$: Number of features selected.
- n_neighbors = 31: Number of neighbors.
- **Distance Metric:** Manhattan distance (L_1-norm).
- **Weighting Scheme:** Distance-based weighting.

This configuration achieved a balance between bias and variance, resulting in robust predictive performance, as detailed in the results section.

C Feature Selection and Integration with Model Fitting

Feature Selection with `SelectKBest` Feature selection was performed using the `SelectKBest` method, which ranks features based on their correlation with the target variable using the F-statistic from a regression model. This process was tightly integrated with Bayesian optimization, ensuring that the number of selected features (k_f) was optimized in conjunction with the KNN hyperparameters. This synergy between feature selection and model fitting allowed the optimization process to identify predictors that complemented the distance metric and weighting scheme, resulting in an optimal configuration tailored to the dataset.

Evaluating Robustness of Feature Selection To evaluate the robustness of `SelectKBest`, alternative methods such as stepwise selection and tree-based selectors were explored. Stepwise selection iteratively added or removed features to maximize model performance, while tree-based selectors utilized feature importance from tree-based models to capture interaction effects. Despite their advantages, the observed strong linear relationships between individual features and the target variable made `SelectKBest` with regression scoring the most efficient and accurate choice. It provided an effective balance between computational cost and predictive performance, aligning well with the characteristics of the data.

Integration with the Preprocessing Pipeline Feature selection was seamlessly integrated into the preprocessing pipeline, preceding the model fitting stage and applied independently to each cross-validation fold. This approach ensured that selected features dynamically adapted to the training data in each fold, avoiding overfitting or data leakage. By closely linking feature selection with hyperparameter tuning and model fitting, the pipeline achieved a cohesive and efficient framework for selecting the most relevant predictors while optimizing model performance.

D Partial Dependence Plots (PDPs)

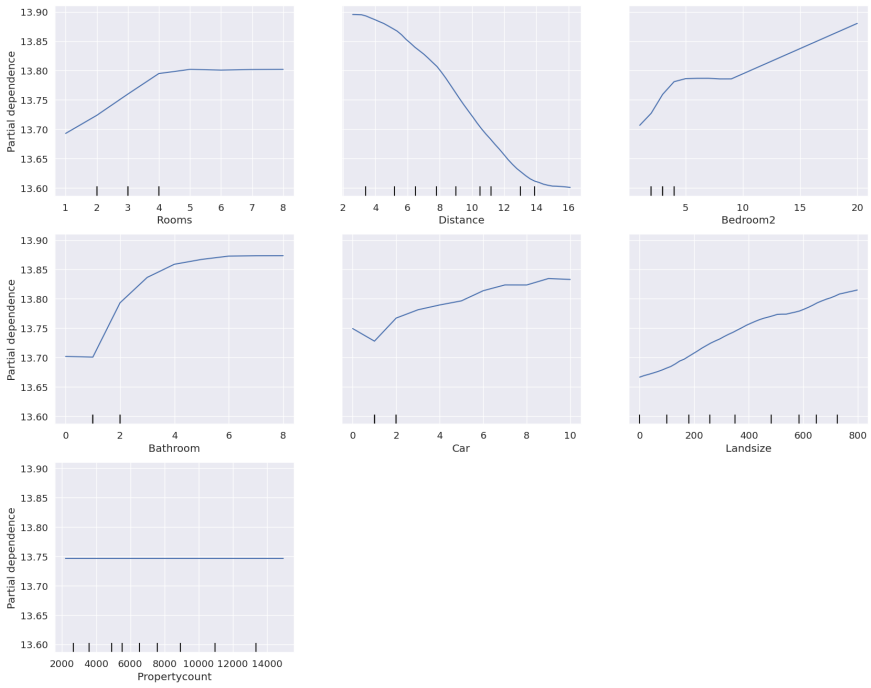

Figure 5: Partial Dependence Plots (PDPs).

Partial Dependence Plots (PDPs) illustrate the relationship between the predicted response and a selected subset of features, referred to as $\mathtt{S}$, while averaging out the influence of all other features, referred to as $\mathtt{C}$. Intuitively, PDPs can be interpreted as the expected target response as a function of the features of interest.

Mathematical definition Given $\mathtt{S}$, the set of input features of interest, and $\mathtt{C}$, its complement, the partial dependence function for the predicted variable $\hat{y}$ at a specific point $\mathbf{x}_\mathtt{S}$ is defined as:

$$PD(\mathbf{x}_\mathtt{S}) = \mathbf{E}_{\mathbf{x}_\mathtt{C}}\left[f(\mathbf{x}_\mathtt{S}, \mathbf{x}_\mathtt{C})\right] = \int f(\mathbf{x}_\mathtt{S}, \mathbf{x}_\mathtt{C}) p(\mathbf{x}_\mathtt{C})\, d\mathbf{x}_\mathtt{C},$$

where f represents the model's prediction function and $p(\mathbf{x}_\mathtt{C})$ is the probability density function of the complement features. This expectation marginalizes over the distribution of the features in $\mathtt{C}$, isolating the effect of the features in $\mathtt{S}$. The resulting function $PD(\mathbf{x}_\mathtt{S})$ depends only on the features in $\mathtt{S}$, making it possible to plot their influence on the predictions.

Computation methods The integral in the partial dependence definition is typically approximated using the *Brute method*: this computes the expectation by averaging over the dataset X:

$$PD(\mathbf{x}_\mathtt{S}) = \frac{1}{N}\sum_{i=1}^{N} f(\mathbf{x}_\mathtt{S}, \mathbf{x}_{i,\mathtt{C}}),$$

where $\mathbf{x}_{i,\mathtt{C}}$ represents the values of the complement features for the i-th sample. For each grid value of $\mathbf{x}_\mathtt{S}$, the model's predictions are averaged across all samples.