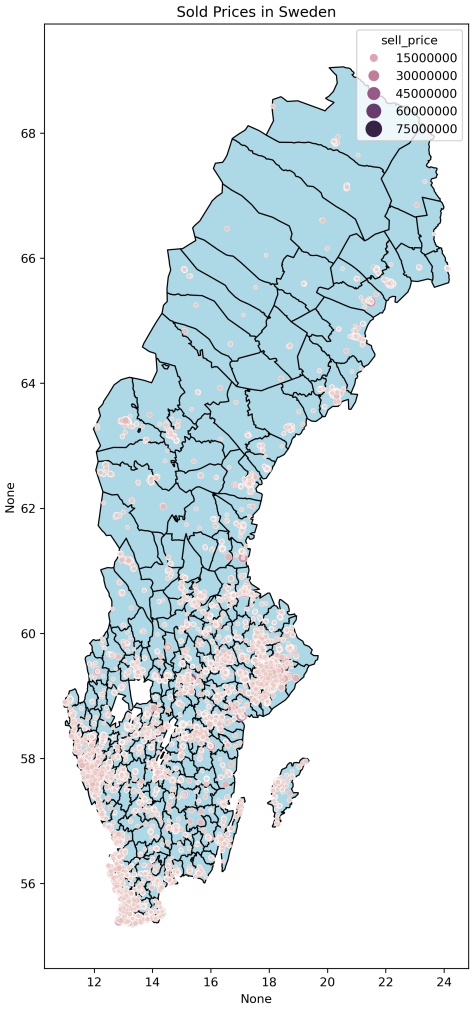
Swedish Housing Market Prediction 2024 - Model Validation File

Hoang Nguyen

December 16, 2024



Contents

1 Introduction **4**

2 Exploratory Data Analysis **4**

3 Feature Engineering **7**
3.1 Missing Values . . . 7
3.2 Drop unnecessary Rows and Columns . . . 8
3.3 One-Hot-Encoding . . . 8
3.4 Extreme Observations . . . 8
3.5 Sparse Values . . . 8
3.6 Feature Selection . . . 9

4 Model Selection **9**
4.1 Baseline Models . . . 9
4.2 Model A: XGBoost . . . 11
4.2.1 No Hyperparameter Tuning . . . 11
4.2.2 With Hyperparameter Tuning . . . 11
4.3 Model B: Decision Tree . . . 12
4.3.1 Untuned Model . . . 12
4.3.2 Tuned Model . . . 13

5 Evaluation **14**

6 Conclusion **16**

A Appendix **18**

List of Figures

1 Median and Average Asking and Sell Prices per Year . . . 5
2 Sell Price per Location and Object Type . . . 6
3 Count of Sales by Object Type . . . 6
4 Distribution of the Price Variables . . . 7
5 Predicted vs. Actual Sell Price using Gradient Boosting Regression . . . 10
6 Predicted vs. Actual without Hyperparameter Tuning . . . 11
7 Predicted vs Actual Sell Price with Hyperparameter Tuning . . . 12
8 Predicted vs. Actual Sell Price using a Decision Tree Model with depth=20 . . . 13
9 Feature Importance for the XGBoost Model . . . 14
10 Sell Price Distribution of Apartments . . . 15
11 MAPE by Price Range . . . 15
12 MAPE by Location . . . 16
13 Distribution of Apartments by Location . . . 16
14 MAPE by Object Type . . . 16
A Visualized Correlations of Key Features with Target Feature . . . 18
B Predicted vs. Actual Sell Price for the Dataset. Note: this model was run on non-outlier corrected data. . . . 20

List of Tables

1 Correlation with Sell Price 4
2 Number of Sell Prices Per Year 5
3 Feature Sets Overview 9
4 Model Performance Metrics for Different Feature Sets 10
5 Evaluation Metrics for XGB without Hyperparameter Tuning 11
6 Evaluation Metrics for XGB with Hyperparameter Tuning 11
7 Decision Tree Evaluation Metrics 13
8 Decision Tree Evaluation Metrics 14
A Missing Data Percentage 19
B Performance Metrics for Using Linear Regression 19
C Train and Test Metrics for Different `max_depth` Values 19

1 Introduction

The aim of the assignment was to leverage a dataset (provided, but scraped from booli.se) to predict real-estate sell prices in Sweden using different machine learning methods.
The nature of the dataset, as it only contains properties that have been sold, thus excludes properties that were listed, but not sold, quite certainly limits the extent to which this analysis can be generalized. In addition, evaluating the entire Swedish real estate market in one model may lead to generalization of important regional or even local differences. In including features describing location to the highest extent possible (balancing overfitting), we, however, try to prevent overgeneralization.

2 Exploratory Data Analysis

The aim is to predict sell prices of real-estate proprieties in Sweden, therefore the variable *sell_price* is defined as the target.

Table 1: Correlation with Sell Price

Variable	Correlation
sell_price	1.000000
asking_price	0.989878
living_area	0.348326
rooms	0.340767
longitude	0.218686
fiscal_year	0.193119
operating_cost	0.183008
floor	0.164208
housing_association_fee	0.134286
long_term_real_estate_debt	0.062723
total_loan	0.062458
latitude	0.025547
total_rental_area	0.021604
number_of_rental_units	0.008399
long_term_debt_other	-0.000639
total_commercial_area	-0.003366
plot_is_leased	-0.005048
plot_area	-0.026836
housing_association_savings	-0.031489
additional_area	-0.035389
construction_year	-0.050007
total_living_area	-0.104819
number_of_units	-0.115836
total_plot_area	-0.127311

Table 1 shows the correlation with the target (sell price) for a selection of the numerical variables in the dataset. Unsurprisingly, a strong correlation is observed with the asking price. In addition, we observe a notable positive correlation with living area and number of rooms. Longitude (measures how far east or west a location is from the prime meridian) shows a relatively strong correlation. This observation is not surprising, as Sweden is more densely populated toward the coast, where the values for latitude are higher. On the negative correlation side, we observe moderate correlations with some of the variables contained in the *HousingAssociation* and *AnnualReport* datasets. Note however, that data coverage is significantly worse, thus we refrain from drawing premature conclusions at this point. Figure A in the appendix further illustrates some key relationships.

Table 2: Number of Sell Prices Per Year

Year	Number of Sell Prices
2006	1
2007	2
2008	554
2009	1214
2010	1188
2011	576
2012	8232
2013	35901
2014	41616
2015	38356
2016	39423
2017	51059
2018	54022
2019	62916
2020	69244
2021	72755
2022	59294
2023	59845
2024	53348

Table 2 shows the number of apartments sold for each year in the data set. We observe that for the first years, fewer observations are available. The sample for the first years may be a bit too small to make meaningful predictions. When running our first models, we observed noticeable performance differences for the early years of the models. Therefore, combined with the observations in table 2 and figure 2, we decided to use data from after 2012 to train our models.

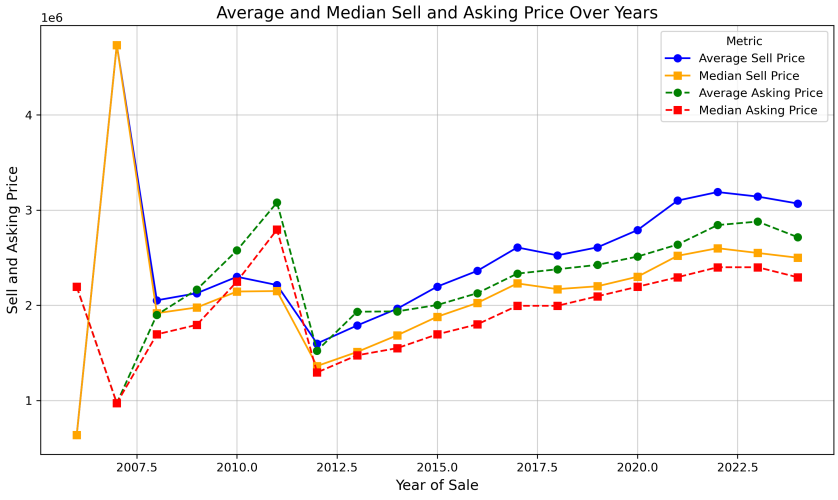


Figure 1: Median and Average Asking and Sell Prices per Year

Figure 1 further reveals some interesting information about prices. We see that prices appeared more volatile in early years (i.e., 2007-2012), whereas all the price measures appear to follow a stable upward trend, cumulating in a slight decrease in recent years, probably related to the COVID-19 pandemic and the following high interest rate period, while also showing a somewhat higher price level compared to all other regions.[1]

[1]Note that variable is called *location* in the dataset, however aggregates over larger *regions* in Sweden.

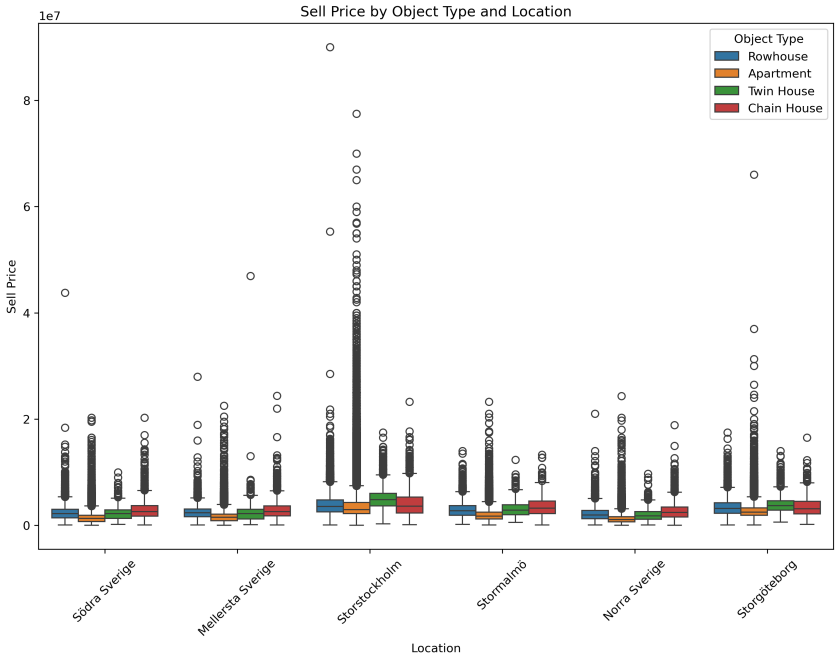


Figure 2: Sell Price per Location and Object Type

Figure 2 illustrates the sell prices for regions, broken down into different object types. Upon closer inspection, we observe, that differences between object types exist (as expected). Additionally, we observe that the Stockholm region (i.e., *Storstockholm*) appears to have a large number of outliers for the apartment category (see *Extreme Values Section*).

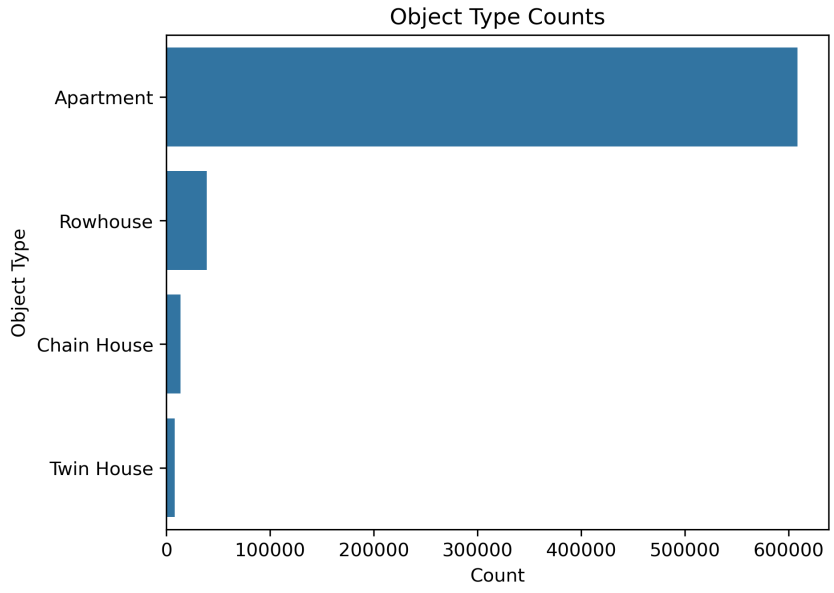


Figure 3: Count of Sales by Object Type

Figure 3 reveals that the majority of our data consists of sold apartments, while the different house types make up smaller sections of the data.

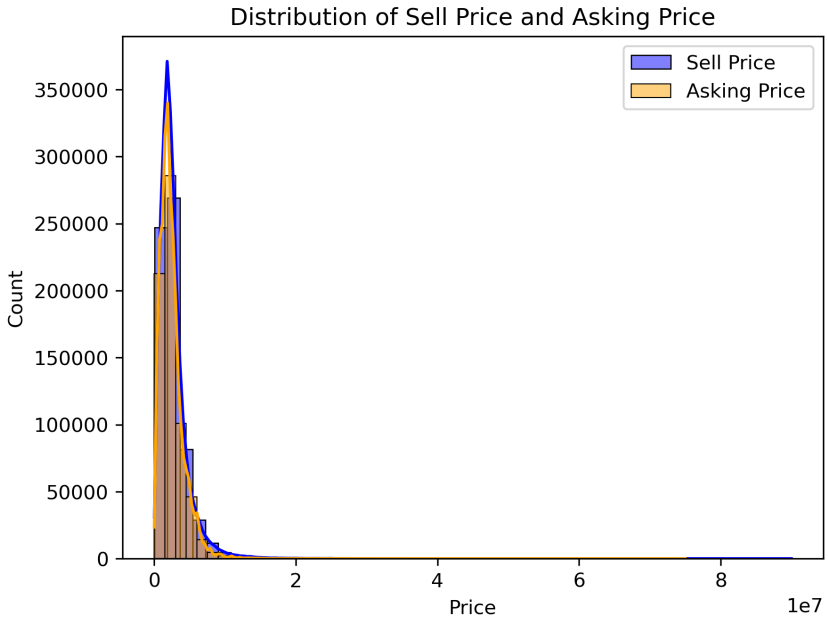


Figure 4: Distribution of the Price Variables

Looking at the distribution of the price variables, figure 4 reveals that the sell- and asking price have similar distributions. The main differences appear to be that the sell price contains some more extreme values, while the density around the median is slightly higher as well.

3 Feature Engineering

3.1 Missing Values

As the dataset contained a lot of missing values, we decided not to make use of columns that contained a large share of missing values (i.e. $> 70\%$). Table A in the appendix provides an overview over the missing values.
For variables that exhibit better coverage, but still contained missing values, we chose the following imputation method:

- **asking_price:** Impute by the median postcode price, remainder imputed by the median object_type price.
- **rooms**: impute by the median number of rooms
- **association_tax_liability**: impute by *lowered_tax_liability*, as per advise given in help session.[2]
- **floor**: impute by median floor
- **region**: impute by mode
- **location**: impute by mode
- **living_area**: impute by median
- **housing_association_fee**: impute by median

[2]98% of Swedish housing associations enjoy preferred tax treatment, giving them an edge in the market. As 98% is close enough to 100%, we assume that the HA that don't have preferred treatment, have been market in the data.

3.2 Drop unnecessary Rows and Columns

From figure 1 we observe that the median and asking prices follow a significantly different pattern for the first years for the dataset, only after 2013 we see the stable upwards trend in Sell % Asking Price. Table 2 further indicates that the we have fewer observations for the first years contained in the data (from 2007-2012 with a maximum of 8000 observations per year). Therefore, to not include any non-indicative trends in our models, we decided to remove observations dated before 2013 to reflect the more recent trends more accurately.

3.3 One-Hot-Encoding

Categorical features included in the model were hot-one encoded, no features were used that had a order assigned to them (e.g., energy class).
We decided to encode 3 features that we plan to use them as variables for our prediction model:

- object_type
- location
- association_tax_liability

These particular features were chosen based on some insights in the Swedish real estate market (location and tax liability are important in determining sell prices). Another factor weighting in for the features above was the relatively low number of unique values, reducing problems with cardinality.

3.4 Extreme Observations

Upon closer inspection of the data we realized that almost all variables included some outliers. As (at least the bulk) of the data is assumed to be populated by people, we infer that some degree of human error is present in the dataset.

A good example of such errors are apartments with 39 rooms or an apartment on floor 100 of a building (a quick Google search reveals that the highest building in Sweden has 54 floors). Other observations labeled as outliers by a simple IQR approach, however, are more involved. In particular, for the price variables (i.e., asking price), it was difficult to differentiate between real-world extreme observations, and outliers that could potentially interfere with our model. As shown in table 1, asking price appears to be an important predictor in our model, therefore outliers should be treated with caution.

However, as the distribution of our target (*sell_price* appears to have a relatively long right tail, we can also avoid training our model on extreme observations by dropping the highest sell prices.[3]

3.5 Sparse Values

The data contained both numerical and categorical features with only sparse values (sparse values contribute 70% of all values). Hot-one encoding, thus creating binary features for each categorical variable futher aggravates the problem of sparse values. To counter this, we chose to run models that handle sparse values and cardinality well.

[3]We ended up dropping the 5% highest sell prices for each year to balance our training and testing sets.

3.6 Feature Selection

Due to the sheer amount of available variables, feature selection posed to be a challenge in the beginning. In the first days of the assignment, we focused too much on getting *smaller* features right, not testing running models on the bigger, probably more important features. After realizing our mistake and pivoting our approach, based on the correlation metrics with sell_price and also some business sense about the importance of location features, we defined some of the more important features, split into four feature sets, as shown in table 3.
When inspecting some early models for performance over time, we observed a noticeable increase in error during the years of the pandemic. While this observation did not surprise us, we wanted to come up with a way to address it in the model. Therefore, *is_covid* is a binary variable included to indicate the years of the COVID-19 pandemic and the unusual real-estate market during the time (and right after it).

Table 3: Feature Sets Overview

Feature Set	**Features**
Set 1	• asking_price • living_area • longitude • latitude • year_of_sale • rooms • floor • housing_association_fee • is_covid
Set 2	All features from Set 1 plus: • location_Storgöteborg • location_Södra Sverige • location_Stormalmö • location_Storstockholm • location_Norra Sverige
Set 3	All features from Set 1 plus: • object_type_Rowhouse • object_type_Chain House • object_type_Twin House • association_tax_liability_other • association_tax_liability_standard_tax_liability
Set 4	All features from Set 1, Set 2, and Set 3 combined.

4 Model Selection

4.1 Baseline Models

Linear Regression:

To establish a baseline to compare later predictions to, we decided to use a linear regression

model. Given the data provided, a key challenge was to work around the missing value problem. We run the model on the different feature set defined above and obtained the results presented in table B in the appendix.
However, running a linear regression without restricting parameters, however, results in some negative guesses, which, of course, are not feasible (see figure B in the appendix).

Gradient Boosting
As we did not want to establish a baseline with negative coefficients, we attempted to run a Gradient Boosting Regression Model on the same features and obtained the following results:

Table 4: Model Performance Metrics for Different Feature Sets

Feature Set	Train MAPE (%)	Test MAPE (%)	Train MAE	Test MAE	Train R^2	Test R^2
Set 1	15.60	17.87	270,112.01	378,696.10	0.8635	0.8473
Set 2 (Loc)	15.59	18.67	269,878.64	385,876.41	0.8636	0.8475
Set 3 (obj. type, tax)	15.50	18.66	268,745.66	385,108.49	0.8661	0.8456
Set 4 (all combined)	15.44	18.12	267,992.03	379,791.85	0.8666	0.8454

We observe that the results shown in table 4 show in a significant improvement in terms of error over the standard linear model presented in table B. Furthermore, the error does not appear to drop significantly between the training and testing set, which indicates the model is not overfitted. In terms of feature sets, we do not appear to be gaining massive performance when adding some more variables, However, at this point, it was yet unclear whether this was a model specific observation. Also note that, using the gradient boosting method, we worked around the negative predictions problem (as shown in figure 5. However, the results still appear to leave room for improvement.

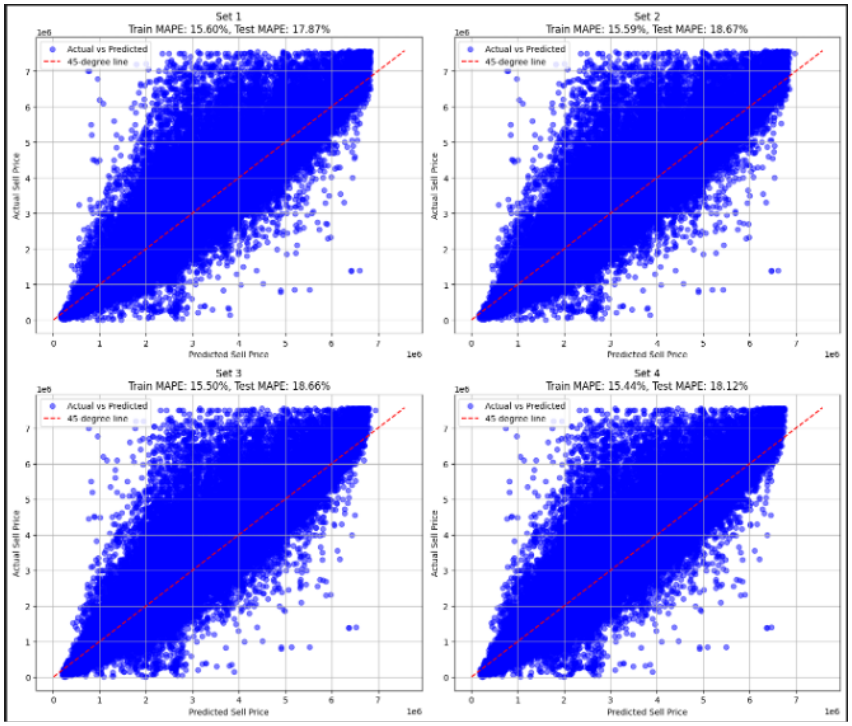


Figure 5: Predicted vs. Actual Sell Price using Gradient Boosting Regression

Further, after establishing a baseline for the different feature sets, we determined feature set 4 to be the best set for future use. While the difference in model performance was not vast, running all sets on the models below would have significantly increased the run-times.

4.2 Model A: XGBoost

4.2.1 No Hyperparameter Tuning

Moving on, we ran an XGBoost model to test against our baseline. In a first step, we ran the XGB model with default parameters and obtained the evaluation metrics presented in table 5.

Metric	Train	Test
MAE	219755.66	329920.08
MAPE (%)	12.38	16.36
R^2	0.91	0.88

Table 5: Evaluation Metrics for XGB without Hyperparameter Tuning

Compared to the baseline, we were able to improve the model error by about 2 percentage points.

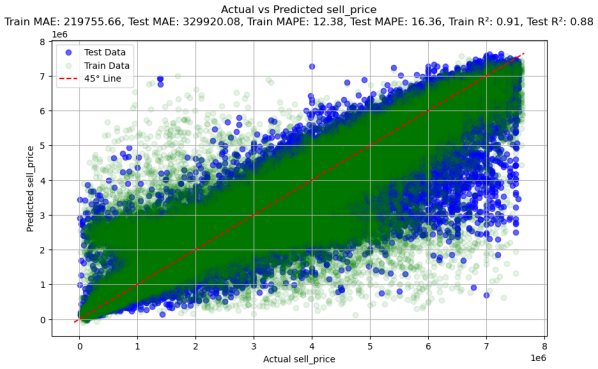


Figure 6: Predicted vs. Actual without Hyperparameter Tuning

4.2.2 With Hyperparameter Tuning

To improve on the untuned performance, we ran a Randomized Search for hyperparameter optimization with 3-fold cross validation. The evaluation metrics presented in table 6 show another 1.1 percentage point improvement over the untuned model.

Metric	Train	Test
MAE	185033.02	306822.38
MAPE (%)	10.72	15.27
R^2	0.93	0.90

Table 6: Evaluation Metrics for XGB with Hyperparameter Tuning

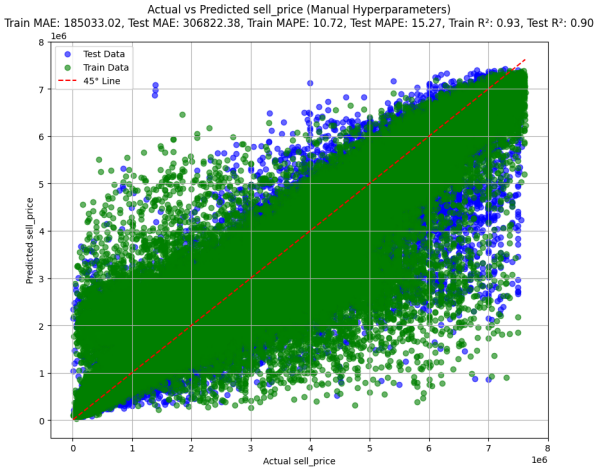


Figure 7: Predicted vs Actual Sell Price with Hyperparameter Tuning

We decide to use `RandomSearchCV` to tune some parameters from XGBoost rather than `GridSearchCV` because it saves us a lot of computing resources and time. We've got the best parameters tuned as follow:

- **subsample:** 1.0, equal to all data is used without sampling, which often improves stability in small datasets.
- **reg_lambda:** 100, L2 regularization parameter to penalize large weights and prevent overfitting. A high value like 100 suggests strong regularization was effective in controlling complexity and improving generalization.
- **reg_alpha:** 0.1, L1 regularization parameter, indicates minimal sparsity was introduced, favoring a balanced model.
- **n_estimators:** 300 boosting rounds (trees), balances computational cost and model performance, avoiding overfitting from too many trees.
- **max_depth:** 13 allows the model to capture complex patterns but avoids being overly deep, which could lead to overfitting.
- **learning_rate:** 0.05, controls the contribution of each tree to the model. A smaller value ensures smoother optimization, avoiding drastic changes and improving convergence.
- **gamma:** 0.1, the minimum loss reduction to create a new tree split, promotes splits for slight improvements, aiding in fine-tuned tree growth.
- **colsample_bytree:** 1.0 – equal to using all features, ensures the model has full information to find splits, likely useful given the dataset features.

4.3 Model B: Decision Tree

4.3.1 Untuned Model

Decision Tree is also a model that satisfy dataset with highly sparse-value and unscaled features. Initially we tried the model without tuning first and get the results in table 7.

Table 7: Decision Tree Evaluation Metrics

Metric	Value
Train MAE	944.2
Train MAPE (%)	0.04
Train R^2	0.99
Test MAE	442,275.2
Test MAPE (%)	21.3
Test R^2	0.76

Regarding MAPE, we see that Train MAPE is only 0.04%. Because we do not limit max_depth so the tree keeps splitting until it will quite fit with the training set. However, in the test set, the Test MAPE up to 21.3% - 21% gap vs. MAPE Train set. Decision tree without tuning is facing overfit problem, which performs quite well on training set but perform poorly on test set. Therefore, we decided to go for Decision tree with some tune in max_depth.

4.3.2 Tuned Model

For Decision tree, we understand that to avoid overfitting, we should limit the max_depth. Therefore, we tried to test a loop for max_depth from 20 to 30 to see which max_depth show the best results. We do not want to test over 30 because the more depth, the more likely the model will overfit with the training set and also cause computation heavy (overview shown in table C in the appendix).
The results showed that max_depth = 20 is the best one, which show the least Test_MAPE overall (19.8%), and the gap vs. Train_MAPE (7.1%) is also smallest in all results. Indeed, the test MAPE decrease 2%, and the Train_MAPE, though increase, but overall show less overfitting problem vs. decision before-tuning. Yet, the visualization in figure 8 and the relatively large gap between the train and test error suggest that the model may suffer from overfitting.

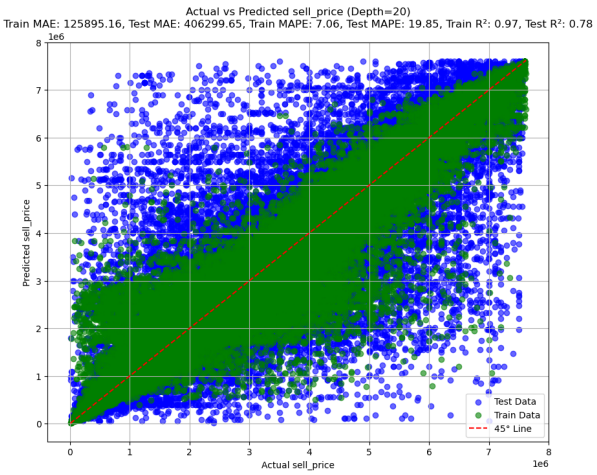


Figure 8: Predicted vs. Actual Sell Price using a Decision Tree Model with depth=20

Table 8: Decision Tree Evaluation Metrics

Metric	Value
Train MAE	125895.16
Train MAPE (%)	7.06
Train R^2	0.97
Test MAE	406299.65
Test MAPE (%)	19.85
Test R^2	0.78

5 Evaluation

What is the best performing model?
XGBoost with hyperparameter tuning shows a quite good result among Linear Regression, Gradient Boosting (baseline) and Decision Tree (challenger). The MAPE on Test is 15.27% - the lowest among all and also has the low gaps vs. Train set (5%), indicating that the model is not overfitting. Besides, MAPE 15.27% on test set is an acceptable result as we are predicting the housing price based on a big and uncleaned real dataset. While it involved a bit of tuning, the time to run the RandomSearchCV to find the best parameters did not take long and high computation resources – actually it took around 15 minutes on my computer. Therefore, compared with the results we've for simpler models like Decision Tree or Gradient Boosting, XGBoost with some tuning is actually a good choice to apply for the housing prediction problem.

What are the most important features in the selected model?
Based on our XGBoost – Tuning model, we run feature importance to see which features that have the most impact on sell_price.

- **Asking_price:** Of course, asking_price seems to be the most important one. From our help sessions, we recognize that the asking_price is the first price that the buyers and sellers come to contact each other. Therefore, sell_price should be somehow near around the asking_price.
- **Latitude and Longitude:** These show the exact location of the apartment and are also the 2nd most important features that impact the sell_price.
- **Housing_association_fee:** Based on the Swedish housing insights, this is also the 3rd important feature that impacts the decision of customers on buying an apartment.
- **Categorical Features (e.g., location):** These seem to be not quite highly important, because the location covers a wide range, not as specific as latitude and longitude.

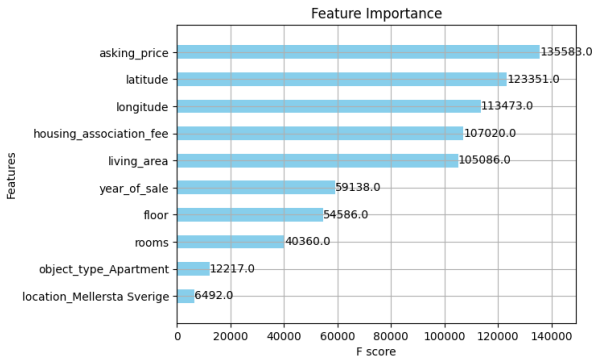


Figure 9: Feature Importance for the XGBoost Model

Which segments does our model perform well on?
Generally, we expect out model to perform well on the segments where a lot of training data was available. Therefore, in terms of price range, most properties were priced above SEK 1 million, therefore, we expect our model to perform well on that segment. The same observation can be made in terms of location, where 42% of properties are located in Stockholm, with the rest being split between the remaining regions. Furthermore, knowing that the dataset consists of 90% apartment sales (see figure 3), we expect our model to have significantly higher performance on apartments compared to other object types.

Figure 10: Sell Price Distribution of Apartments

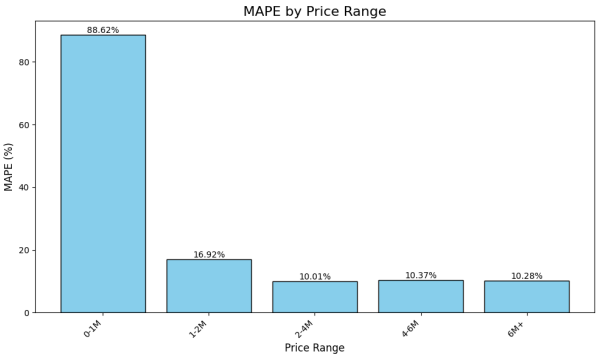


Figure 11: MAPE by Price Range

Performance Results by Feature:
The figures below give some indication of the model performance by feature. Figure 11 shows that our models work best on the properties with prices above SEK 1 million, which contribute approximately 90% of the observations in our dataset. For properties below that threshold, the model performs poorly.
In terms of location, we see some more variation. For the regions with a closer definition (i.e., Stockholm, Gothenburg, and Malmö), the model appears to perform quite well, while the larger areas see a drop in performance.

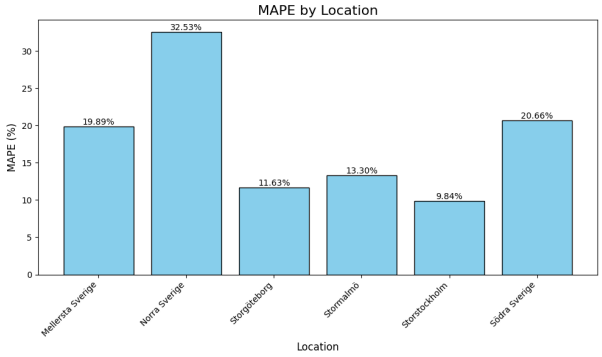


Figure 12: MAPE by Location

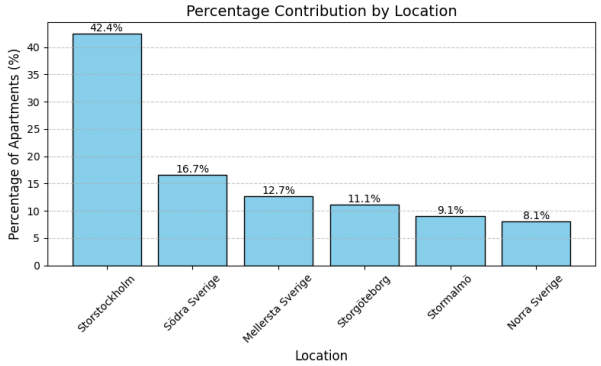


Figure 13: Distribution of Apartments by Location

The pattern continues when looking at the error for object type. The model performs well on apartments, row- and twin houses, while the performance is worse on chain houses, for which the sample size is smaller (see figure 3).

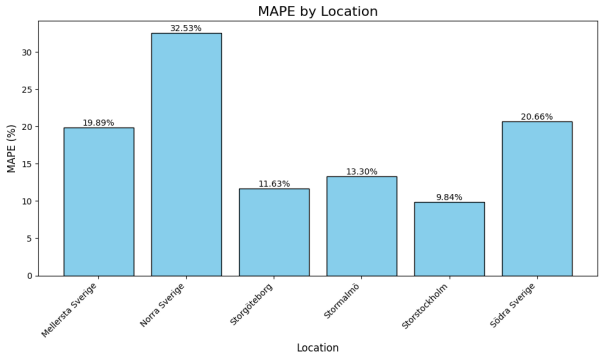


Figure 14: MAPE by Object Type

6 Conclusion

In conclusion, XGBoost with tuning appears to work well on the data provided. The error of approximately 15% is fairly good based on industry standard, because we run the model based on the real dataset of Swedish housing market, which is not perfectly cleaned and standardized in the beginning.
There are some potential weaknesses that the model underperformed for less represented features in the data (e.g., object_type Chain House exhibits quite a high MAPE, but it represented

a small share (2%) of all observations. We also note that not all determinants of sell price can be observed in the data (e.g., condition of the apartment, division of rooms, etc.). Regardless, the model shows low error in the majority categories (such as Location Stockholm, Object Type Apartment).
For future improvement, we will try more features more advanced models such as Neural Networks, or try some dimension reducing methods such as PCA.

A Appendix

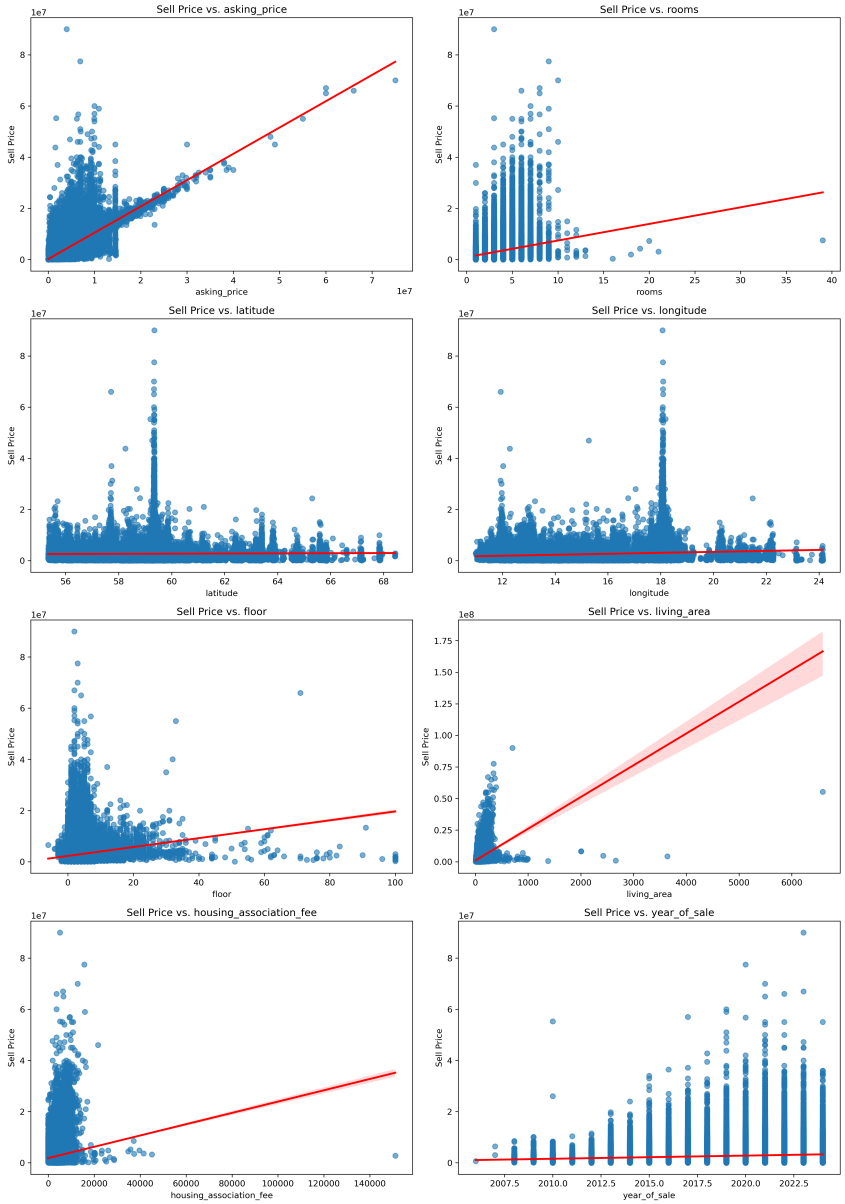


Figure A: Visualized Correlations of Key Features with Target Feature

Table A: Missing Data Percentage

Column	Missing Percentage (%)
has_balcony	99.97
has_patio	99.97
has_fireplace	99.97
cover_photo_description	99.95
width	99.91
height	99.91
is_new_construction	99.91
has_solar_panels	99.91
key	99.45
additional_area	96.10
plot_area	95.60
heating	93.60
energy_class	91.76
amenities	86.96
long_term_debt_other	80.00
total_rental_area	67.56
total_plot_area	67.37
total_commercial_area	67.06
construction_year	66.95
district	64.80
number_of_rental_units	64.17
plot_is_leased	63.84
association_tax_liability	63.83
housing_association_savings	63.82
housing_association_name	63.65
total_living_area	63.65
long_term_real_estate_debt	63.65
number_of_units	63.65
housing_coop_id	63.65
total_loan	63.65

Table B: Performance Metrics for Using Linear Regression

Feature Set	Train MAE	Test MAE	Train MAPE (%)	Test MAPE (%)	Train R^2	Test R^2
Set 1	551284.74107	545560.31499	26.36104	26.61669	0.71573	0.71983
Set 2	546795.94674	541014.10886	26.46977	26.71986	0.71899	0.72302
Set 3	558513.57070	552670.40457	27.35444	27.60448	0.71894	0.72286
Set 4	553599.70959	547813.20251	27.43344	27.68471	0.72224	0.72614

Table C: Train and Test Metrics for Different `max_depth` Values

max_depth	**train_mae**	**train_mape**	**train_r2**	**test_mae**	**test_mape**	**test_r2**
20	125,895.16	7.1%	0.97	406,299.65	19.8%	0.78
21	110,449.59	6.3%	0.97	408,239.71	19.8%	0.78
22	95,434.73	5.5%	0.98	417,226.46	20.4%	0.77
23	81,577.56	4.7%	0.98	423,006.31	20.8%	0.76
24	68,836.60	4.0%	0.99	429,942.17	21.3%	0.76
25	57,472.03	3.4%	0.99	430,985.44	20.6%	0.76
26	47,408.56	2.8%	0.99	434,074.31	21.3%	0.76
27	38,797.12	2.3%	0.99	432,763.87	21.1%	0.76
28	31,418.20	1.9%	1.00	438,791.61	21.3%	0.75
29	25,204.14	1.5%	1.00	435,175.48	21.0%	0.76
30	20,131.86	1.2%	1.00	444,550.17	22.0%	0.75

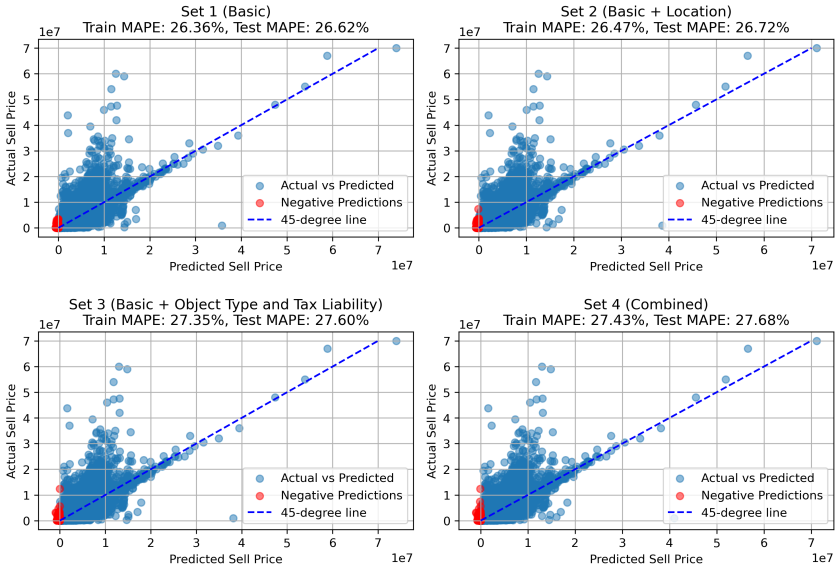


Figure B: Predicted vs. Actual Sell Price for the Dataset. Note: this model was run on non-outlier corrected data.