# Analysis and Prediction of Apartment Sales in Lund, Sweden

**Zhiwei Chang**

December 28, 2021

# Contents

# 1 Introduction

Nearly everyone cares about housing price at some point in their lives. As the Wall Street legend Peter Lynch wrote in his best-selling book *One Up in Wall Street*: "Before you do invest anything in stocks, you ought to consider buying a house, since a house, after all, is the one good investment that almost everyone manages to make". He also described the customary progression of houses as follows: "You buy a small house (a starter house), then a medium-sized house, then a larger house that eventually you don't need. After the children have moved away, then you sell the big house and revert to a smaller house". Indeed, many people followed this path and I personally witnessed many of my friends made sizable profits in such transitions.

The aim of this project is to provide a thorough analysis and eventually build an effective model using machine learning algorithms to predict the housing prices in the city of Lund where I am currently living. Here we do not discuss economy, interest rates, pandemic, or government policies, etc. that are commonly regarded as main factors affecting housing prices (by the way, the housing market is still booming, regardless), we leave these work to the economists. Instead, we let data talk. I hope this project does not only provide a general overview of the recent Lund housing market, but also answers three main questions for the potential housing buyers/sellers:

1. When is the best time to sell/buy a property?
2. Which brokerage agency (or even whom) you should call if you want to sell?
3. What is the reasonable price for a given property with parameters e.g. location, living area size, number of rooms, floor number, year of build, etc.?

This is a typical regression problem in data science. Everyone might have their own opinions or experiences to these issues, but here we explore the answer to these questions based on rigorous data analysis and model prediction. The dataset used for this project were collected from hemnet.se, the go-to website where people look for housings in Sweden, using Python and a web scrapping module called Beautiful Soup. Note that all data I collected are public information, but for privacy concerns, I did not upload the generated CSV files which contain private info like addresses to my Github repo. These files can be provided upon request.

The outline of this report is as follows. In section 2, we give a brief overview of the apartment sales in Lund, then we try to answer the first two questions brought up in this section using data analysis. In section 3, we will first deal with the missing data, examine the correlation between sale price and each feature, and deal with the outliers. Then we do some feature engineering and data transformation and then split the dataset into train and test part and test several machine learning algorithms using training dataset to predict the apartment price, which would then be validated by the test set. In appendices we also provide the Python code (in the form of Jupyter Notebook) for data collection (section A-D), cleaning (section E), and model training (section F).

**This project is still ongoing.** The majority work has been done. Currently I am building a website to deploy the trained model, and I expect the final product would be a website with API endpoint where you provide the relevant features of a property (e.g. address, size, build year, number of rooms, etc.) or even just a simple Hemnet link to the listing, then with one click, it shows the estimated price. The model should be continuously refined and validated by taking streaming data from Hemnet (using MLOps) as the new listings keep coming in.

## 2 Data Analysis

### 2.1 Overview

For the past year (dated from 2020-10-30 to 2021-10-21), 2500 properties listed on Hemnet were sold in Lund. As seen in Fig. 1 that 1972 (79%) of them are apartment (lägenhet in Swedish). In comparison, there are only 489 houses (villa + radhus) have been sold, out of which 217 are townhouse (radhus). In this report, we focus on apartment sales not only because the corresponding dataset is larger (no surprise, as apartment is much more affordable than house in most cases), but also Hemnet provides more features for apartment than that for house, both of which benefit the later model training.

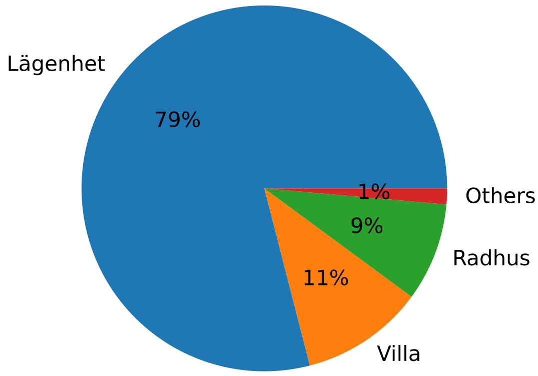



Figure 1: Pie chart of the housing types distribution.

First we present an overview of the apartment sales in Lund for the past year. The market worths 5.485 billion SEK in total during this period. In Table 1 we listed some statistics which may give us a general picture about the market. The first column is about the price. As seen that the average sold price (mean) for an apartment is 2.78 million SEK, and the medium is 2.53 million SEK (indicating that 50% apartments are sold below and the other half are above this price). The cheapest one was sold for 750 thousand SEK while the most expensive one 11.66 million SEK (more expensive than most houses! After checking its location and size, this price starts to make some sense to me...). Fig. 2 shows the histogram of the price and we may find that it is not a normal distribution. The skewness and kurtosis are 2.56 and 11.22, respectively, which indicate a heavy-tailed skewed right distribution. That is also why the medium is smaller than the mean value. We need to do some data transformation later as a preparation for model training.

About the average price, the mean value is 44319 SEK/m$^2$ which is close to the medium (42284 SEK/m$^2$). Taking Chinese cities as comparison, this is equivalent to the price of those "new first-tier cities" i.e. Hangzhou, Tianjin, Nanjing, etc., after calculating SEK/RMB exchange rates. Only 25% apartments were sold below 33604 SEK/m$^2$. But we shall remember that the average price for the smaller apartments are normally higher than that for the larger ones. So if you own an apartment with 100 m$^2$, you should not expect to sell it with this mean or medium average price unless the location is good. We will also investigate this correlation in Section

Table 1: Statistics about Lund apartment market.

| | Price (tKr) | Average (Kr/m$^2$) | Avgift (Kr/month) | Area (m$^2$) | Room# | Year |
|---|---|---|---|---|---|---|
| mean | 2781 | 44319 | 3821 | 67.1 | 2.5 | 1972 |
| min | 750 | 15900 | 0 | 16.7 | 1.0 | 1844 |
| max | 11662 | 128571 | 10626 | 245.0 | 7.0 | 2021 |
| 25% | 2075 | 33604 | 2866 | 47.0 | 2.0 | 1951 |
| 50% | 2525 | 42284 | 3725 | 64.0 | 2.0 | 1967 |
| 75% | 3131 | 53097 | 4688 | 83.5 | 3.0 | 2007 |
| Total | 5485346 | | | | | |

3.1. The monthly fee (avgift in Swedish) ranges between 0 to 10626 SEK per month, with the mean value 3821 SEK/month (medium about the same).

Regarding the size, on average it is about 67 m$^2$ while the smallest one is 16.7 m$^2$ and the largest is 245 m$^2$ (again, larger than most houses!). The medium is 64 m$^2$, close to the mean value. Sixth column indicates that most popular housing types are 2 or 3-room apartment. From the last column we learnt that the oldest apartment sold in last year was built in 1844 (nearly 180 years old, good quality!). Although there are some new properties being developed in Lund, half of the sold ones are over 50 years old.

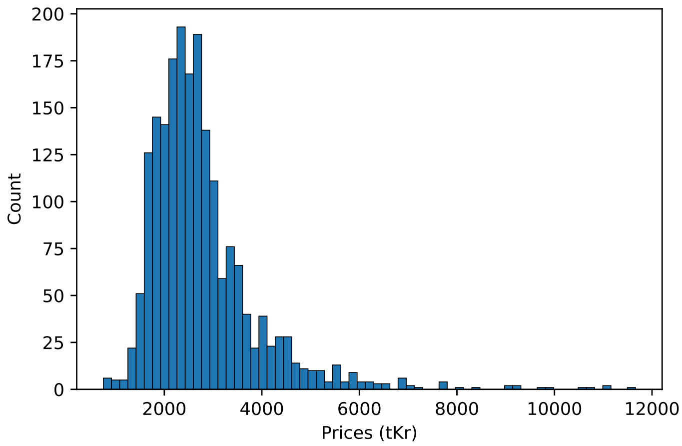



Figure 2: Histogram of sold prices.

## 2.2 Best selling/buying time window

Here we try to answer the first question brought up in section 1 which is to figure out the best time to sell or buy an apartment. We should keep in mind that the following analysis are based on statistics that may vary between each individual case.

The left panel in Fig. 3 shows the apartment sales grouped by months. As expected, the worst sale occurs in December due to the Christmas/New year holiday. Then from January to September the sales are roughly the same, June and July are a little bit down but not so much. The best season is in October and November with the most sales. But if we look at average price in the right panel the situation is different: the highest price occurs in summer (July and August) while the lowest in winter (November, January, and February).

This is interesting and why is that? My guess is that for sellers, since there are many listings available in the market in October/November, they may be willing to take lower price offers due to the competition so they can sell it quickly before holidays in December. For buyers, they may want to settle down before starting new semester or job in September, so they are willing to pay more in Summer. Do you agree with me? If not, what is your theory?

Anyway, the data show that we better sell in summer (July/August) and buy in winter (November, January, February). Suppose you want to buy an apartment and follow this advice, in some extreme scenario you may save up to 5000 SEK/$m^2$, that is 300 thousand SEK for a 60 $m^2$ apartment. Unfortunately Hemnet does not provide the information of listing date which we could use to estimate on average how much time it takes to sell an apartment. But let us give a reasonable estimation about 1 to 2 months. So if you want to sell your apartment for a higher price in July or August, then perhaps you should call your broker and put up your listing on Hemnet in April or May.

I should say that above recommendation is solely based on the observation from recent one-year data, and the trend might fluctuate over time. We could assume that every year it follows the same pattern and there is not much of difference. But it would be really helpful to check out the data from previous years to see the trend and do some time series analysis. In fact, we should do this for the whole project! However, this will be tremendous amount of work even for the data collection part. So unless Hemnet allows me to use their API freely to pull any data I want from their database, we have to rely on these collected data, confine ourselves and be happy about the results we get.

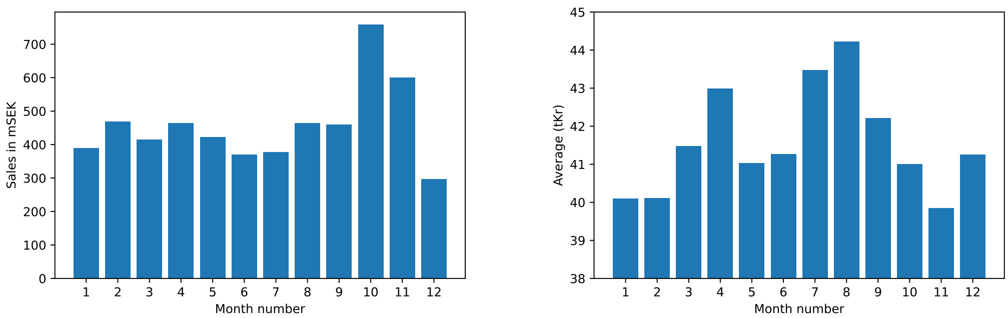



Figure 3: Bar chart of total sales (left) and average price (right) grouped by months.

## 2.3 Whom you should call?

Now we answer the second question: which agency or even whom should you call if you want to sell your apartment? Normally you receive post regularly from the guy who closed the deal for your current apartment, updating the value of your property in the market. But is he the best person to contact if you want to sell? Again, I should declare that this is a hobby project which is not funded by any brokerage company or agent.

We first check out the market share for each agency. For all 1972 sold apartments, there are in total 39 brokerage companies appeared in the list. Fig. 4 shows the top 5 agencies in Lund housing market as well as their market share percentage. As seen that Bjurfors Lund Centrum and Fastighetsbyrån Lund each takes approximately one quarter of the market. Then another Bjurfors branch Bjurfors Lund Väster and Erik Olsson Fastighetsförmedling each takes about 11%, followed by MOHV Lund with about 8% market share. The rest 34 companies share the rest 20%. Although out of these 34 companies, there are agencies from nearby cities like Malmö, Kristianstad, Lomma, Kävlinge, Staffanstorp, etc. which may occasionally do business in Lund, clearly the competition is fierce.

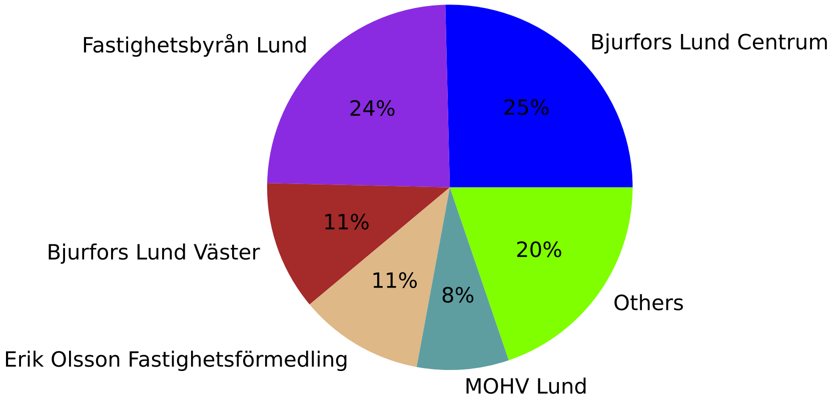



Figure 4: Pie chart of the market shares for the top 5 brokerage agencies.

In Table 2, we present the sale volume and market share for the top 10 brokerage agencies. From this table, we may conclude that Lund apartment sales are dominated by the top 5 companies which take over about 80% market share, especially the top two agencies Bjurfors Lund Centrum and Fastighetsbyrån Lund, which together take over half of the market.

Table 2: Top 10 brokerage agencies in Lund (from 2020.10 to 2021.10).

| Brokerage Agencies | Sales (mKr) | Share (%) |
|---|---|---|
| Bjurfors Lund Centrum | 1393.0 | 25.4 |
| Fastighetsbyrån Lund | 1329.2 | 24.2 |
| Bjurfors Lund Väster | 628.1 | 11.5 |
| Erik Olsson Fastighetsförmedling | 604.3 | 11.0 |
| MOHV Lund | 446.1 | 8.1 |
| Länsförsäkringar Fastighetsförmedling Lund | 278.7 | 5.1 |
| Våningen & Villan Lund | 163.1 | 3.0 |
| Svensk Fastighetsförmedling Lund | 160.6 | 2.9 |
| Mäklarhuset Lund | 113.8 | 2.1 |
| Bülow & Lind Fastighetsförmedling | 100.0 | 1.8 |

Regarding the performance of the brokers, Table 3 shows the number of apartment sold, total sales, and the agencies they belong for the top 10 brokers in Lund for the past year. The honor of Best Broker of the Year goes to Simon, who sold 139 apartments with the sale volume 351 million SEK. Daniel sold a little bit less (337 million SEK), actually both of them are senior partners in Bjurfors. In fact, for the top 3 agents, if they decide to open their own business, immediately their one-man new firm will become number 6 in Table 2. Both Simon and Andreas contributed approximately 1/4 of the total sales in their respective firms. If we check the agencies they belong, we find that except Oskar for Erik Olsson Fastighetsförmedling, the other 9 brokers work either in Bjurfors or Fastighetsbyrån. All these facts indicate the decisive contribution of a good agent to the success of a brokerage firm. If you were to sell your apartment, you can contact either name listed in Table 3 as they are all experienced brokers. Personally I would call Simon not only because of his excellent record, but also he has very good reputation among my friends.

Table 3: Top 10 brokers in Lund (from 2020.10 to 2021.10).

| Broker | Sold # | Sales (mKr) | Agencies |
|---|---|---|---|
| Simon Wall Sanktnovius | 139 | 351.4 | Bjurfors Lund Centrum |
| Daniel Frostmo | 109 | 337.3 | Bjurfors Lund Centrum, Bjurfors Lund Väster |
| Andreas Hansen | 96 | 309.5 | Fastighetsbyrån Lund |
| Yosef Halim | 94 | 268.0 | Bjurfors Lund Centrum, Bjurfors Lund Väster |
| Joacim Ernstsson | 50 | 202.6 | Bjurfors Lund Väster |
| Rasmus Asterhed | 73 | 190.2 | Fastighetsbyrån Lund |
| Oskar Olsson | 73 | 186.3 | Erik Olsson Fastighetsförmedling |
| Kristoffer Cedergren | 60 | 184.8 | Bjurfors Lund Centrum |
| Bardia Ghasemi | 62 | 177.0 | Bjurfors Lund Centrum |
| Jakob Gustafsson | 61 | 166.1 | Fastighetsbyrån Lund |

# 3 Model prediction

## 3.1 Feature engineering

In the previous sections, we analyzed the variables that are important to answer our questions. For each sold property, Hemnet actually provides the following 17 features:

1. Address
2. Size
3. Number of room
4. Whether there is balcony
5. Whether there is patio
6. Whether there is an elevator
7. Floor number
8. Total building floor
9. Monthly fee (avgift)
10. Year of build
11. Asking price
12. Sold price
13. Average price
14. Housing type
15. Broker who sold this property
16. Brokerage agency
17. Sold dates

Now we are going to explore each one of them and their mutual correlations, especially how they contribute to the sold price that we are mostly interested in. But before that, we may remove the broker/agency features. For the same apartment, different brokers may sell it for different prices, but here we assume that the price is mainly reflected by its intrinsic value. Moreover, we delete "Average price" as it is merely sold price divided by size. In addition, we remove the feature "Asking price" which is the primary information we want to provide from our model. Later we can compare our model prediction with this feature to see if we can do better than professional brokers regarding evaluating apartment price.

### 3.1.1 Missing data

We first check the number of missing values for each feature. As seen in Table 4 that for the feature "Patio", there are over 89% values missing. This is not a common feature and we may safely delete it. Then we make the following modifications:

- if the elevator/balcony value is missing, then we treat it as no elevator/balcony;
- we set floor number/total floor missing values as 1;
- fill the missing values in "Build year" and Lat-Lon coordinates with the most common value appeared in the corresponding feature;

Moreover, we convert the categorical features "Balcony" and "Elevator" to numerical (1's and 0's) as most machine learning algorithms are only able to handle numerical features.

Table 4: Missing values.

| | Total | Percent |
|---|---|---|
| Patio | 1762 | 0.894 |
| Elevator | 304 | 0.154 |
| Build year | 296 | 0.150 |
| Balcony | 242 | 0.123 |
| Floor number | 221 | 0.112 |
| Total floor | 221 | 0.112 |
| Lat | 5 | 0.003 |
| Lon | 5 | 0.003 |
| Addresses | 0 | 0.000 |
| area ($m^2$) | 0 | 0.000 |
| # of rooms | 0 | 0.000 |
| Monthly Fees (Kr) | 0 | 0.000 |
| Prices (tKr) | 0 | 0.000 |
| Average | 0 | 0.000 |
| Month | 0 | 0.000 |

#### 3.1.2 Outliers

Since we know the address for each sold apartment, we may investigate the effect of location to the final price. Many things could affect the housing price, but for the same housing type, it is a common sense that the location is usually the decisive factor. Nearly all agents you talked to, would repeat "Location! Location! Location is everything!" (p.s. there are many housing price datasets hosted on Kaggle and some of them listed over 100 features but omitted the location. Maybe out of privacy concerns when they compiled the datasets, in my opinion it is not so useful to explore these data unless for an educational purpose. If you go through the posted analysis, some of them are reasonable like price is strongly correlated to the overall quality of the building, living area size, etc. But some show that whether there is a full bath could also be crucial. A full bath? Really?)

To view the geo-distribution of the data, we first convert all addresses to the corresponding latitude-longitude coordinates by using a Python library named geopy, then we are able to display all our data on the Google map with an API key. Fig. 5 shows such a scattering plot. The left panel shows the housing distribution in city area and on the right panel we see that there are also several apartments sold in Södra Sandby, Dalby, and Veberöd, the three localities of Lund municipality. We can add more functionalities to Fig. 5 by adding the hover tools so that when we move mouse to a data point, a tooltip will appear to give more detailed info of the sold property (right panel of Fig. 6) together with a color bar indicating the sold price.

The dynamic Google map is a nice displaying tool which may be utilized in the final web application, but it is not so useful for the model training. If we look at the left panel in Fig. 6, we may find that as expected, the expensive ones are mostly clustered in the city center and the sold prices are decaying as the distance from city center increases. To investigate the correlation between location and sold price, let us test the simplest assumption: the sold prices are only dependent on such distances in terms of location. So for the same apartment, no matter which region it is located, Norra Fäladen or Linero, as long as its distance with respect to the city center is the same, we assume that it would have same price. Of course this is a very rough assumption, but later we may find that it is actually very powerful. If we set the location of the most expensive apartment as the reference point, Fig. 7 (left) shows the scattering plot

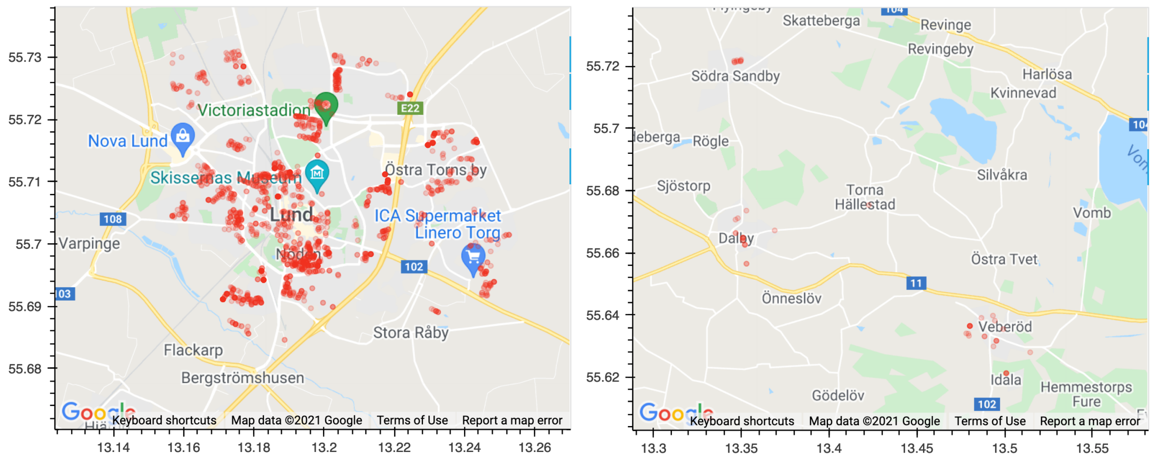



Figure 5: Scattering data plot overlaid on Google map: (left) Lund city; (right) urban area.

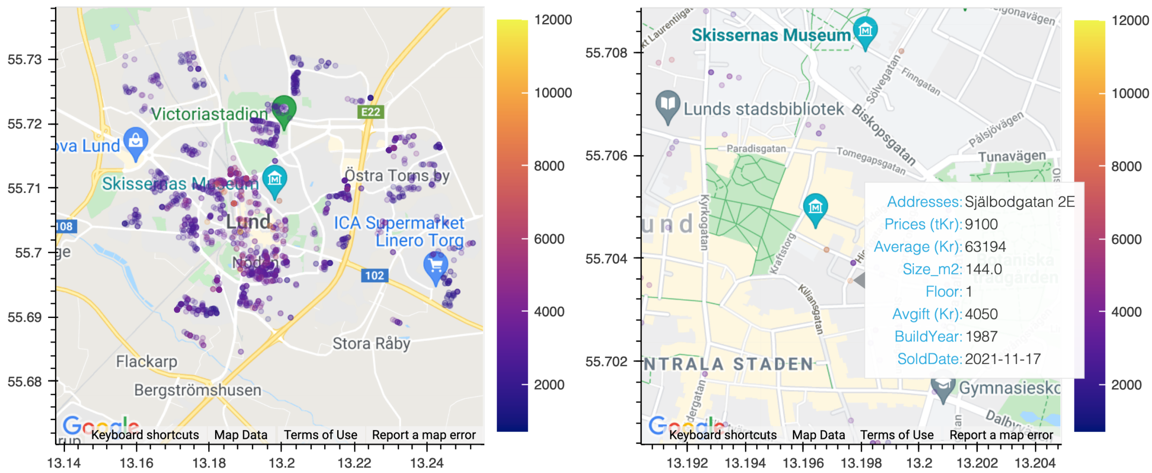



Figure 6: Dynamic Google map with interactive plot and color bar.

of the sold prices versus distances. We may find that all data points are separated into three groups and most of them are clustered within 5 km radius which corresponds to the city area. The other data points belong to the three localities we saw in the right panel of Fig. 5. In order to use the assumption we made to account for the location factor, we have to treat these locality data points as outliers and delete them because each locality region may have their own price-versus-distance correlation. The right panel shows the plot including 1919 data points from such 5-km-radius cutoff. After some data transformations, we may find some linearities between these two variables (see more details in Appendix F).

### 3.1.3 Data transformation

As we mentioned in section 2.1 that the price data are skewed. In this case, we need to do data transformation so that it obeys normal distribution as normality is the most fundamental assumption in multivariate analysis. We apply the log10 transformation (shown in Fig. 8) to the price as well as the size data which also has the skewness issue. Moreover, we do some feature scaling by transforming the lan-lon values to the Cartesian X-Y coordinates. In addition, we manually add a feature "floorRatio" which is the floor number divided by the total building floors. Normally for the same building, the apartment located on the third floor is more expensive than that on the ground floor. Table 5 lists the skewness for each numerical

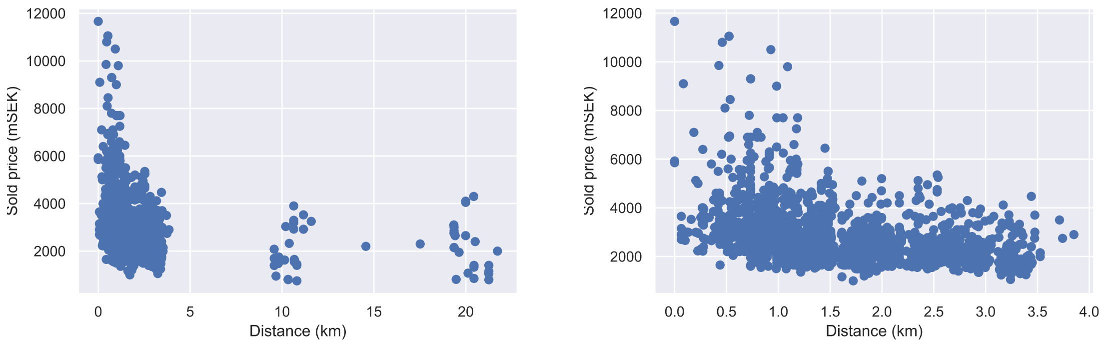



Figure 7: Scattering plot of the sold prices versus distances: (left) Lund municipality including three urban areas; (right) Lund city.

variable and we may find that log10 transformation has reduced the skewness from 2.56 to 0.77 for the price data.

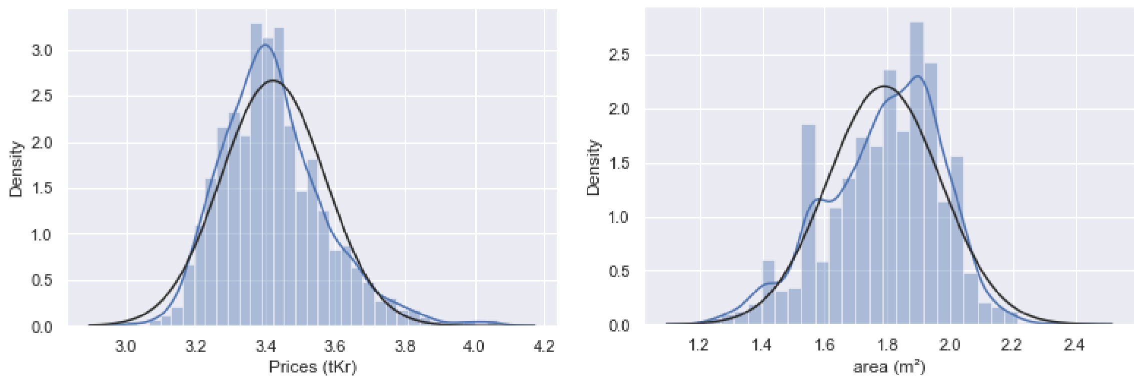



Figure 8: Data distribution after log10 transformation: (left) sold price; (right) size. Black lines show the norm-fitted curves.

### 3.1.4 Correlations

In Fig. 9 we plotted a heatmap showing the correlations between different numerical variables using Seaborn library. As seen that the three variables monthly fee, number of rooms, and size are linearly correlated. This is not surprising as for larger apartment, normally there will be more rooms, and the avgift tends to be correspondingly higher. As a rule of thumb, if the correlation degree between two variables is higher than 0.7, we should only include one of them to avoid multilinearity error in the model training. Therefore, we only retain the size variable and delete monthly fee and number of rooms. In Fig. 9 we also observe that the two most important factors contributing to the sold prices are size and location (distances). The size variable shows a positive while the distance gives a negative correlation to the price. Later in the model training session, we shall compare the models trained using only these two variables with that including all numerical variables to see how big the effect of these two parameters is.

Table 5: Feature skewness (in descending order).

| | Skewness |
|---|---|
| X | 0.770137 |
| Prices (tKr) | 0.769580 |
| distance | 0.572442 |
| Elevator | 0.472850 |
| Y | 0.359745 |
| floorRatio | -0.178422 |
| Month | -0.190101 |
| area ($m^2$) | -0.411013 |
| Build year | -0.482272 |
| Balcony | -0.561480 |

### 3.2 Model training

Finally we come to the part of model training. We first split the data into train (70%) and test (30%) sets, then we test three commonly used ensemble algorithms: gradient boosting, xgboost, and random forest. As seen from Table 6 that if we train the algorithms using only two features size and location, then the mean absolute error (MAE) is about 300 tSEK. If we include all features listed in Table 5, then the model performance is improved by around 10%. We are glad to see that all three models trained with including all numerical features slightly outperform the brokers' evaluation from comparing with the asking price MAE. xgboost and random forest performed especially well.

Here we only used the default parameters for these algorithms provided by scikit-learn, we could definitely do better by fine-tuning these parameters, i.e. learning rate, sample number, etc. We could also try more algorithms like Lasso, Ridge, Elastic-Net regression, CatBoost, LightGBM, etc. Each model may capture certain part of the data pattern, which may have overfitting/underfitting problem. We could blend these models to get a better prediction. However, there are two important factors we can not include our model. First, Hemnet does not provide the overall quality info for the sold property. A well renovated apartment can easily sold for additional 100-200 tSEK compared to a moderated one. Second, the bidding process is often unpredictable, sometimes even unreasonable when the market is hot. Therefore, the MAE of our trained models might be reasonable. Despite missing these two important factors, we can still provide a good estimating price for an apartment given the information we have. Maybe the user could somehow modify the predicted price based on how well his/her apartment is renovated and the hotness of the current housing market.

In Table 7, we listed the asking, final sold prices as well as our model prediction using random forest algorithm for the six randomly picked apartments sold recently. As seen that our model perform pretty well especially when the price is not too high. The next step is to build a website to deploy this model. In addition, I plan to use MLOps to continuously train the model with the streaming data.

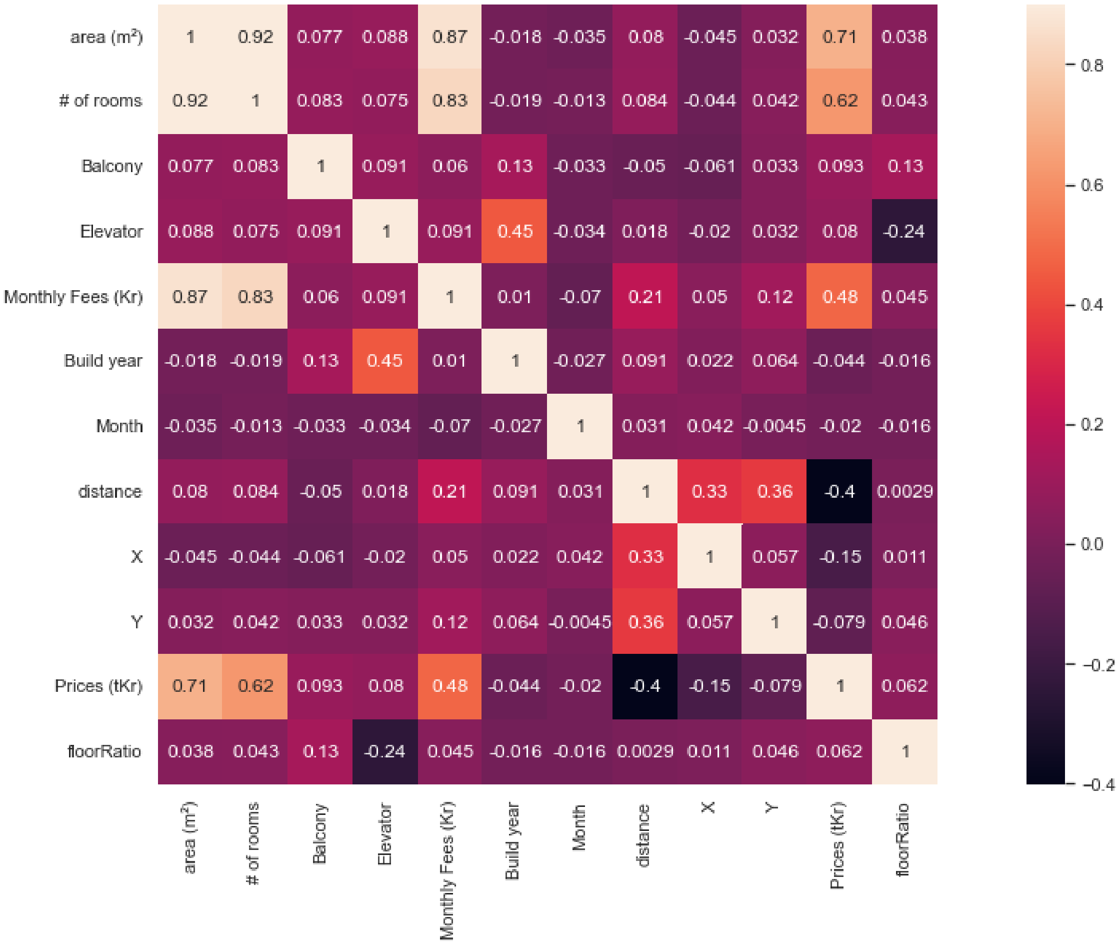



Figure 9: Heat map

Table 6: Mean absolute error (in tSEK) for different algorithms.

| | size + distance | all features | |
|---|---|---|---|
| Gboost | 312.3 | 286.0 | |
| xgb | 301.2 | 277.0 | |
| RF | 300.6 | 273.9 | |
| Asking | | | 287.1 |

Table 7: Price prediction for six randomly picked properties sold recently (price in mSEK)

| Address | Sold date | Model | Asking | Final |
|---|---|---|---|---|
| Södra Esplanaden 21 | 2021-12-08 | 6.34 | 7.20 | 7.30 |
| Kakelvägen 6A | 2021-12-08 | 1.53 | 1.59 | 1.80 |
| Norrängavägen 5A | 2021-12-07 | 1.35 | 1.42 | 1.49 |
| Måsvägen 20C | 2021-12-06 | 2.15 | 2.10 | 2.10 |
| Dag Hammarskjölds Väg 1E | 2021-12-03 | 2.82 | 2.39 | 2.73 |
| Sunnanväg 10H | 2021-12-02 | 3.34 | 2.99 | 3.50 |

# A Web scrapping from a Hemnet sample page

[1]:
```
from bs4 import BeautifulSoup
import pandas as pd
import re
```

## A.1 Scrap info from the html <body> tag

[3]:
```
with open('hemnet_page50.html', 'r') as html_file:
    soup = BeautifulSoup(html_file, 'html.parser')
body = soup.find('body')
```

## A.2 Individual link for each sold property

[4]:
```
# These links provide additional information (e.g. year of build, agent, etc.
↪) which I refered to as second-layer.

links = body.select("li.sold-results__normal-hit a")
actual_links = [link['href'] for link in links]
actual_links[0:6]
```

[4]:
```
['https://www.hemnet.se/salda/lagenhet-3rum-centrum-lunds-kommun-sodra-
esplanaden-5a-1268410',
 'https://www.hemnet.se/salda/lagenhet-4rum-ostra-torn-lunds-kommun-
stralsundsvagen-92-1268390',
 'https://www.hemnet.se/salda/lagenhet-1,5rum-veberod-lunds-kommun-
vildgasvagen-45-1268417',
 'https://www.hemnet.se/salda/villa-7rum-stangby-lunds-kommun-
vallkarratorn-502-1263056',
 'https://www.hemnet.se/salda/lagenhet-1rum-centrum-lunds-kommun-
gronegatan-19b-1268104',
 'https://www.hemnet.se/salda/lagenhet-3rum-norra-faladen-lunds-kommun-
skarpskyttevagen-22-f-1268135']
```

## A.3 Addresses

[6]:
```
# Most important feature in model training which will later be converted to␣
↪lan-longitude coordinates.

addresses = body.select("li.sold-results__normal-hit h2")
str_addresses = [address.get_text().replace('\n', '').strip() for address in␣
↪addresses]
str_addresses[0:4]
```

[6]:
```
['Södra Esplanaden 5A',
 'Stralsundsvägen 92',
```

```
'Vildgåsvägen 45',
'Vallkärratorn 502']
```

### A.4 Property type

```
[7]: types = body.select("li.sold-results__normal-hit title")
     actual_type = [kind.get_text() for kind in types]
     actual_type[0:6]
```

```
[7]: ['Lägenhet', 'Lägenhet', 'Lägenhet', 'Villa', 'Lägenhet', 'Lägenhet']
```

### A.5 Living area

```
[9]: # There are 3 pieces info (i.e. living area, # of rooms, sold price)␣
      ↪embedded
     # in the div class=sold-property-listing__subheading, need to separate them.

     info3 = body.select("div.sold-property-listing__subheading")
     actual_info = [info.get_text().replace('\n', '').replace('\xa0', '').strip()␣
      ↪for info in info3]
```

```
[10]: area = [None for _ in range(50)]
      for i in range(len(actual_info)//2):
          # if i % 2 == 0:
          area[i] = re.search(r'(.*?)m²', actual_info[2 * i]).group(1).replace('  ␣
       ↪                               ', '')
      area[0:6]
```

```
[10]: ['68', '80,5', '37 + 15', '251', '34,6', '95']
```

### A.6 Sold prices (in tKr) & Number of rooms

```
[11]: prices = [0 for _ in range(50)]
      number_of_rooms = [None for _ in range(50)]
      for i in range(len(actual_info)//2):
          prices[i] = int(int(re.search(r'Slutpris(.*?)kr', actual_info[2*i + 1]).
       ↪group(1))/1000)
          m = re.search(r'm²                         (.*?)rum', actual_info[2 *␣
       ↪i])
          if m:  # this if condition is necessary cause group() does not work for␣
       ↪None type
              number_of_rooms[i] = float(m.group(1).replace(',', '.').strip())
      prices[0:6]
```

```
[11]: [3200, 2225, 1000, 4450, 2300, 2065]
```

[12]:
```
number_of_rooms[0:6]
```

[12]:
```
[3.0, 4.0, 1.5, 7.0, 1.0, 3.0]
```

### A.7 Sold dates

[13]:
```
dates = body.select("div.sold-property-listing__sold-date")
actual_date = [date.get_text().replace('\n', '').replace('Såld', '').strip()␣
↪for date in dates]
actual_date[0:6]
```

[13]:
```
['11 oktober 2020',
 '11 oktober 2020',
 '11 oktober 2020',
 '11 oktober 2020',
 '10 oktober 2020',
 '10 oktober 2020']
```

### A.8 Monthly fees (avgift) in Kr

[14]:
```
# There are also 3 pieces info embeded in the div␣
↪class=sold-property-listing__size
# which we only need the monthly fees (avgift)

sizes = body.select("div.sold-property-listing__size")
actual_size = [size.get_text().replace('\n', '').replace('\xa0', '').strip()␣
↪for size in sizes]
```

[15]:
```
# Preset fees as None type cause most houses (villa) do not have monthly fee.
↪

fees = [None for _ in range(50)]
for i in range(len(actual_size)):
    n = re.search(r'rum        (.*?)kr/mån', actual_size[i])
    if n:
        fees[i] = int(n.group(1).strip())
fees[0:6]
```

[15]:
```
[3509, 5809, 2138, None, 2538, 4956]
```

[16]:
```
d = {'Addresses': str_addresses, 'Types': actual_type, 'area (m²)': area, '#␣
↪of rooms': number_of_rooms, 'Monthly Fees (Kr)': fees, 'Sold Dates':␣
↪actual_date,
    'Links': actual_links, 'Prices (tKr)': prices}
df = pd.DataFrame(data=d)
df.to_csv('hemnet50.csv', index=False)

all_data = pd.read_csv("hemnet50.csv")
```

```
all_data.head()
```

```
[16]:              Addresses     Types area (m²)  # of rooms  Monthly Fees (Kr)  \
      0  Södra Esplanaden 5A  Lägenhet        68         3.0             3509.0
      1   Stralsundsvägen 92  Lägenhet      80,5         4.0             5809.0
      2      Vildgåsvägen 45  Lägenhet   37 + 15         1.5             2138.0
      3    Vallkärratorn 502     Villa       251         7.0                NaN
      4        Grönegatan 19B  Lägenhet      34,6         1.0             2538.0

              Sold Dates                                              Links  \
      0  11 oktober 2020  https://www.hemnet.se/salda/lagenhet-3rum-cent...
      1  11 oktober 2020  https://www.hemnet.se/salda/lagenhet-4rum-ostr...
      2  11 oktober 2020  https://www.hemnet.se/salda/lagenhet-1,5rum-ve...
      3  11 oktober 2020  https://www.hemnet.se/salda/villa-7rum-stangby...
      4  10 oktober 2020  https://www.hemnet.se/salda/lagenhet-1rum-cent...

         Prices (tKr)
      0          3200
      1          2225
      2          1000
      3          4450
      4          2300
```

# B Scrapping first-layer info

[3]:
```
from bs4 import BeautifulSoup
import pandas as pd
import glob
import re
```

## B.1 Generate CSV file for each downloaded Hemnet page (50 in total)

[26]:
```
for p in range(49):

    page = 'hemnet_page' + str(p+1) + '.html'
    with open(page, 'r') as html_file:
        soup = BeautifulSoup(html_file, 'html.parser')
    body = soup.find('body')

    links = body.select("li.sold-results__normal-hit a")
    actual_links = [link['href'] for link in links]

    addresses = body.select("li.sold-results__normal-hit h2")
    str_addresses = [address.get_text().replace('\n', '').strip() for␣
↪address in addresses]

    types = body.select("li.sold-results__normal-hit title")
    actual_type = [kind.get_text() for kind in types]

    info3 = body.select("div.sold-property-listing__subheading")
    actual_info = [info.get_text().replace('\n', '').replace('\xa0', '').
↪strip() for info in info3]

    # Apparently there's some exceptions for "area" which gives None value.␣
↪These elements
    # all have the room type as "Gård/Skog".

    area = [None for _ in range(50)]
    for i in range(len(actual_info)//2):
        # if i % 2 == 0:
        n = re.search(r'(.*?)m²', actual_info[2 * i])
        if n:
            area[i] = n.group(1).replace('                                  ␣
↪', '')

    prices = [0 for _ in range(50)]
    number_of_rooms = [None for _ in range(50)]
    for i in range(len(actual_info)//2):
        prices[i] = int(int(re.search(r'Slutpris(.*?)kr', actual_info[2*i +␣
↪1]).group(1))/1000)
```

```
        m = re.search(r'm²                          (.*?)rum', actual_info[2␣
↪* i])
        if m:
            number_of_rooms[i] = float(m.group(1).replace(',', '.').strip())

    dates = body.select("div.sold-property-listing__sold-date")
    actual_date = [date.get_text().replace('\n', '').replace('Såld', '').
↪strip() for date in dates]

    sizes = body.select("div.sold-property-listing__size")
    actual_size = [size.get_text().replace('\n', '').replace('\xa0', '').
↪strip() for size in sizes]

    fees = [None for _ in range(50)]
    for i in range(len(actual_size)):
        n = re.search(r'rum        (.*?)kr/mån', actual_size[i])
        if n:
            fees[i] = int(n.group(1).strip())

    d = {'Addresses': str_addresses, 'Types': actual_type, 'area (m²)':␣
↪area, '# of rooms': number_of_rooms, 'Monthly Fees (Kr)': fees, 'Sold␣
↪Dates': actual_date,
         'Links': actual_links, 'Prices (tKr)': prices}
    df = pd.DataFrame(data=d)

    filename = 'hemnet' + str(p+1) + '.csv'

    df.to_csv(filename, index=False)
```

### B.2 Merge all 50 csvs into 1 file

[ ]:
```
all_files = glob.glob("*.csv")

li = []

for filename in all_files:
    df = pd.read_csv(filename, index_col=None, header=0)
    li.append(df)

frame = pd.concat(li, axis=0, ignore_index=True)
frame.to_csv('hemnet.csv', index=False)
```

### B.3 Check the result

[5]:
```
df = pd.read_csv('hemnet.csv')
print(df.head(5))
```

```
           Addresses     Types area (m²)  # of rooms  Monthly Fees (Kr)  \
```

```
0   Flormansgatan 2A  Lägenhet        43         1.5              2767.0
1  Kastanjegatan 19F  Lägenhet        34         2.0              2415.0
2   Karl XI gatan 47  Lägenhet      87,4         3.0              5787.0
3          Äspet 163     Villa  158 + 22         8.0                 NaN
4  Margaretavägen 3K  Lägenhet        78         3.0              4584.0

          Sold Dates                                              Links  \
0  30 september 2021  https://www.hemnet.se/salda/lagenhet-1,5rum-ce...
1  30 september 2021  https://www.hemnet.se/salda/lagenhet-2rum-jarn...
2  30 september 2021  https://www.hemnet.se/salda/lagenhet-3rum-lund...
3  30 september 2021  https://www.hemnet.se/salda/villa-8rum-lunds-k...
4  30 september 2021  https://www.hemnet.se/salda/lagenhet-3rum-moll...

   Prices (tKr)
0          2370
1          1745
2          4700
3          5350
4          2750
```

### B.4 Check the housing type

[6]:
```python
set(df['Types'])
```

[6]:
```
{'Fritidshus', 'Gård/Skog', 'Lägenhet', 'Radhus', 'Tomt', 'Villa', 'Övrigt'}
```

### B.5 Generate a dataframe for only apartments

[7]:
```python
apart_df = df[df['Types'] == 'Lägenhet']
apart_df.head()
```

[7]:
```
               Addresses     Types area (m²)  # of rooms  Monthly Fees (Kr)  \
0       Flormansgatan 2A  Lägenhet        43         1.5             2767.0
1      Kastanjegatan 19F  Lägenhet        34         2.0             2415.0
2       Karl XI gatan 47  Lägenhet      87,4         3.0             5787.0
4      Margaretavägen 3K  Lägenhet        78         3.0             4584.0
5  Qvantenborgsvägen 4B  Lägenhet        59         2.0             3125.0

          Sold Dates                                              Links  \
0  30 september 2021  https://www.hemnet.se/salda/lagenhet-1,5rum-ce...
1  30 september 2021  https://www.hemnet.se/salda/lagenhet-2rum-jarn...
2  30 september 2021  https://www.hemnet.se/salda/lagenhet-3rum-lund...
4  30 september 2021  https://www.hemnet.se/salda/lagenhet-3rum-moll...
5  29 september 2021  https://www.hemnet.se/salda/lagenhet-2rum-kobj...

   Prices (tKr)
0          2370
1          1745
2          4700
```

```
4            2750
5            2250
```

[ ]:
```
apart_df.to_csv('apart_df.csv')
```

# C Scrapping 2nd layer info from an individual link

[1]:
```python
import requests
from bs4 import BeautifulSoup
import pandas as pd
import re
```

## C.1 Spercify a header to pass the robot detection

[2]:
```python
headers = {'User-Agent': 'Mozilla/5.0 (Windows NT 6.1; WOW64; rv:20.0) Gecko/
 ↪20100101 Firefox/20.0'}
```

## C.2 Scrapping info from an example link

[6]:
```python
df = pd.read_csv('apart_df.csv')

link_ind=1970

r = requests.get(df['Links'][link_ind], headers=headers)

soup = BeautifulSoup(r.content, 'html.parser')

body = soup.find('body')


properties1 = body.select("dd.sold-property__attribute-value")
properties2 = body.select("div.broker-card__info")

for i in range(len(properties1)):
    print(str(properties1[i]).replace('<dd␣
 ↪class="sold-property__attribute-value">', '').replace('</dd>', '')
          .replace('\n','').replace('\xa0','').replace('<i class="fa␣
 ↪fa-arrow-circle-o-up fa-lg price-icon--up"></i>','')
          .strip())

# Here we shall pay special attention to the order of string replacement.␣
 ↪The first replaced string can not be
# a subset of the second string going to be replaced.
print(properties2[0].get_text().replace('\n', '').replace('Kontakta␣
 ↪mäklarkontoret','').replace('Kontakta mäklaren','').replace('Kontakt','').
 ↪strip().split("      "))
```

```
69410kr/m²
10000000kr
+1,05milj. kr (+11%)
Lägenhet
Bostadsrätt
5 rum
```

```
159,2 m²
Ja
3 av 4, hiss finns ej
1903
4868kr/mån
['Joacim Ernstsson', 'Bjurfors Lund Väster']
```

From the above code, we obtain the following info for a specific sold property:

1. Price per square meters
2. Sold price
3. Price increase (compared to the asking price)
4. Housing type
5. Number of room
6. Living area size
7. Whether there is balcony
8. Whether there is patio (now shown in this example)
9. floor number/total building floor, whether there is an elevator
10. Year of build
11. Monthly fee (avgift)
12. Broker who sold this property
13. Brokerage agency

The items (1-6, and 11) have already been provided in the 1st-layer info. Moreover, we noticed that the information of broker/agency is structured differently than other info. Therefore, we shall deal with them separately.

# D Second layer info

## D.1 Items except broker/agency

[1]:
```
import requests
from bs4 import BeautifulSoup
import pandas as pd
import re

headers = {'User-Agent': 'Mozilla/5.0 (Windows NT 6.1; WOW64; rv:20.0) Gecko/
 ↪20100101 Firefox/20.0'}
```

[2]:
```
df = pd.read_csv('apart_df.csv')
```

Here we actually only need the info 7-10. But for some links, not all 13 items are provided, for example, some are missing the year of build, some do not list the avgift, etc. So here we better scrap all info and then clean them later.

[3]:
```
link_len = len(df['Links'])

info0 = [None for _ in range(link_len)]
info1 = [None for _ in range(link_len)]
info2 = [None for _ in range(link_len)]
info3 = [None for _ in range(link_len)]
info4 = [None for _ in range(link_len)]
info5 = [None for _ in range(link_len)]
info6 = [None for _ in range(link_len)]
info7 = [None for _ in range(link_len)]
info8 = [None for _ in range(link_len)]
info9 = [None for _ in range(link_len)]
info10 = [None for _ in range(link_len)]
info11 = [None for _ in range(link_len)]
info12 = [None for _ in range(link_len)]
```

[4]:
```
for link_ind in range(link_len):

    r = requests.get(df['Links'][link_ind], headers=headers)

    soup = BeautifulSoup(r.content, 'html.parser')

    body = soup.find('body')


    properties1 = body.select("dd.sold-property__attribute-value")
    prop_len = len(properties1)
```

```
    info0[link_ind] = str(properties1[0]).replace('<dd␣
↪class="sold-property__attribute-value">', '').replace('</dd>', '').
↪replace('\n','').replace('\xa0','').replace('<i class="fa␣
↪fa-arrow-circle-o-up fa-lg price-icon--up"></i>','').replace('<i class="fa␣
↪fa-arrow-circle-o-up fa-lg price-icon--down"></i>','').strip()
    info1[link_ind] = str(properties1[1]).replace('<dd␣
↪class="sold-property__attribute-value">', '').replace('</dd>', '').
↪replace('\n','').replace('\xa0','').replace('<i class="fa␣
↪fa-arrow-circle-o-up fa-lg price-icon--up"></i>','').replace('<i class="fa␣
↪fa-arrow-circle-o-up fa-lg price-icon--down"></i>','').strip()
    info2[link_ind] = str(properties1[2]).replace('<dd␣
↪class="sold-property__attribute-value">', '').replace('</dd>', '').
↪replace('\n','').replace('\xa0','').replace('<i class="fa␣
↪fa-arrow-circle-o-up fa-lg price-icon--up"></i>','').replace('<i class="fa␣
↪fa-arrow-circle-o-up fa-lg price-icon--down"></i>','').strip()
    info3[link_ind] = str(properties1[3]).replace('<dd␣
↪class="sold-property__attribute-value">', '').replace('</dd>', '').
↪replace('\n','').replace('\xa0','').replace('<i class="fa␣
↪fa-arrow-circle-o-up fa-lg price-icon--up"></i>','').replace('<i class="fa␣
↪fa-arrow-circle-o-up fa-lg price-icon--down"></i>','').strip()
    info4[link_ind] = str(properties1[4]).replace('<dd␣
↪class="sold-property__attribute-value">', '').replace('</dd>', '').
↪replace('\n','').replace('\xa0','').replace('<i class="fa␣
↪fa-arrow-circle-o-up fa-lg price-icon--up"></i>','').replace('<i class="fa␣
↪fa-arrow-circle-o-up fa-lg price-icon--down"></i>','').strip()
    info5[link_ind] = str(properties1[5]).replace('<dd␣
↪class="sold-property__attribute-value">', '').replace('</dd>', '').
↪replace('\n','').replace('\xa0','').replace('<i class="fa␣
↪fa-arrow-circle-o-up fa-lg price-icon--up"></i>','').replace('<i class="fa␣
↪fa-arrow-circle-o-up fa-lg price-icon--down"></i>','').strip()
    info6[link_ind] = str(properties1[6]).replace('<dd␣
↪class="sold-property__attribute-value">', '').replace('</dd>', '').
↪replace('\n','').replace('\xa0','').replace('<i class="fa␣
↪fa-arrow-circle-o-up fa-lg price-icon--up"></i>','').replace('<i class="fa␣
↪fa-arrow-circle-o-up fa-lg price-icon--down"></i>','').strip()
    info7[link_ind] = str(properties1[7]).replace('<dd␣
↪class="sold-property__attribute-value">', '').replace('</dd>', '').
↪replace('\n','').replace('\xa0','').replace('<i class="fa␣
↪fa-arrow-circle-o-up fa-lg price-icon--up"></i>','').replace('<i class="fa␣
↪fa-arrow-circle-o-up fa-lg price-icon--down"></i>','').strip()

    if prop_len == 9:
        info8[link_ind] = str(properties1[8]).replace('<dd␣
↪class="sold-property__attribute-value">', '').replace('</dd>', '').
↪replace('\n','').replace('\xa0','').replace('<i class="fa␣
↪fa-arrow-circle-o-up fa-lg price-icon--up"></i>','').replace('<i class="fa␣
↪fa-arrow-circle-o-up fa-lg price-icon--down"></i>','').strip()
    elif prop_len == 10:
```

```
        info8[link_ind] = str(properties1[8]).replace('<dd␣
↪class="sold-property__attribute-value">', '').replace('</dd>', '').
↪replace('\n','').replace('\xa0','').replace('<i class="fa␣
↪fa-arrow-circle-o-up fa-lg price-icon--up"></i>','').replace('<i class="fa␣
↪fa-arrow-circle-o-up fa-lg price-icon--down"></i>','').strip()
        info9[link_ind] = str(properties1[9]).replace('<dd␣
↪class="sold-property__attribute-value">', '').replace('</dd>', '').
↪replace('\n','').replace('\xa0','').replace('<i class="fa␣
↪fa-arrow-circle-o-up fa-lg price-icon--up"></i>','').replace('<i class="fa␣
↪fa-arrow-circle-o-up fa-lg price-icon--down"></i>','').strip()
    elif prop_len == 11:
        info8[link_ind] = str(properties1[8]).replace('<dd␣
↪class="sold-property__attribute-value">', '').replace('</dd>', '').
↪replace('\n','').replace('\xa0','').replace('<i class="fa␣
↪fa-arrow-circle-o-up fa-lg price-icon--up"></i>','').replace('<i class="fa␣
↪fa-arrow-circle-o-up fa-lg price-icon--down"></i>','').strip()
        info9[link_ind] = str(properties1[9]).replace('<dd␣
↪class="sold-property__attribute-value">', '').replace('</dd>', '').
↪replace('\n','').replace('\xa0','').replace('<i class="fa␣
↪fa-arrow-circle-o-up fa-lg price-icon--up"></i>','').replace('<i class="fa␣
↪fa-arrow-circle-o-up fa-lg price-icon--down"></i>','').strip()
        info10[link_ind] = str(properties1[10]).replace('<dd␣
↪class="sold-property__attribute-value">', '').replace('</dd>', '').
↪replace('\n','').replace('\xa0','').replace('<i class="fa␣
↪fa-arrow-circle-o-up fa-lg price-icon--up"></i>','').replace('<i class="fa␣
↪fa-arrow-circle-o-up fa-lg price-icon--down"></i>','').strip()
    elif prop_len == 12:
        info8[link_ind] = str(properties1[8]).replace('<dd␣
↪class="sold-property__attribute-value">', '').replace('</dd>', '').
↪replace('\n','').replace('\xa0','').replace('<i class="fa␣
↪fa-arrow-circle-o-up fa-lg price-icon--up"></i>','').replace('<i class="fa␣
↪fa-arrow-circle-o-up fa-lg price-icon--down"></i>','').strip()
        info9[link_ind] = str(properties1[9]).replace('<dd␣
↪class="sold-property__attribute-value">', '').replace('</dd>', '').
↪replace('\n','').replace('\xa0','').replace('<i class="fa␣
↪fa-arrow-circle-o-up fa-lg price-icon--up"></i>','').replace('<i class="fa␣
↪fa-arrow-circle-o-up fa-lg price-icon--down"></i>','').strip()
        info10[link_ind] = str(properties1[10]).replace('<dd␣
↪class="sold-property__attribute-value">', '').replace('</dd>', '').
↪replace('\n','').replace('\xa0','').replace('<i class="fa␣
↪fa-arrow-circle-o-up fa-lg price-icon--up"></i>','').replace('<i class="fa␣
↪fa-arrow-circle-o-up fa-lg price-icon--down"></i>','').strip()
        info11[link_ind] = str(properties1[11]).replace('<dd␣
↪class="sold-property__attribute-value">', '').replace('</dd>', '').
↪replace('\n','').replace('\xa0','').replace('<i class="fa␣
↪fa-arrow-circle-o-up fa-lg price-icon--up"></i>','').replace('<i class="fa␣
↪fa-arrow-circle-o-up fa-lg price-icon--down"></i>','').strip()
    elif prop_len == 13:
```

```
        info8[link_ind] = str(properties1[8]).replace('<dd␣
↪class="sold-property__attribute-value">', '').replace('</dd>', '').
↪replace('\n','').replace('\xa0','').replace('<i class="fa␣
↪fa-arrow-circle-o-up fa-lg price-icon--up"></i>','').replace('<i class="fa␣
↪fa-arrow-circle-o-up fa-lg price-icon--down"></i>','').strip()
        info9[link_ind] = str(properties1[9]).replace('<dd␣
↪class="sold-property__attribute-value">', '').replace('</dd>', '').
↪replace('\n','').replace('\xa0','').replace('<i class="fa␣
↪fa-arrow-circle-o-up fa-lg price-icon--up"></i>','').replace('<i class="fa␣
↪fa-arrow-circle-o-up fa-lg price-icon--down"></i>','').strip()
        info10[link_ind] = str(properties1[10]).replace('<dd␣
↪class="sold-property__attribute-value">', '').replace('</dd>', '').
↪replace('\n','').replace('\xa0','').replace('<i class="fa␣
↪fa-arrow-circle-o-up fa-lg price-icon--up"></i>','').replace('<i class="fa␣
↪fa-arrow-circle-o-up fa-lg price-icon--down"></i>','').strip()
        info11[link_ind] = str(properties1[11]).replace('<dd␣
↪class="sold-property__attribute-value">', '').replace('</dd>', '').
↪replace('\n','').replace('\xa0','').replace('<i class="fa␣
↪fa-arrow-circle-o-up fa-lg price-icon--up"></i>','').replace('<i class="fa␣
↪fa-arrow-circle-o-up fa-lg price-icon--down"></i>','').strip()
        info12[link_ind] = str(properties1[12]).replace('<dd␣
↪class="sold-property__attribute-value">', '').replace('</dd>', '').
↪replace('\n','').replace('\xa0','').replace('<i class="fa␣
↪fa-arrow-circle-o-up fa-lg price-icon--up"></i>','').replace('<i class="fa␣
↪fa-arrow-circle-o-up fa-lg price-icon--down"></i>','').strip()
```

[ ]:
```
d = {'info0': info0, 'info1': info1,'info2': info2, 'info3': info3,'info4':␣
↪info4, 'info5': info5,'info6': info6, 'info7': info7, 'info8': info8,␣
↪'info9': info9, 'info10': info10, 'info11': info11, 'info12': info12}
frame = pd.DataFrame(data=d).T
frame.to_csv('unprocessed_sndlayer_info.csv', index=False)
```

## D.2 Broker/Agency

[ ]:
```
brokers = [None for _ in range(link_len)]
agencies = [None for _ in range(link_len)]
```

[ ]:
```
for link_ind in range(link_len):

    r = requests.get(df['Links'][link_ind], headers=headers)

    soup = BeautifulSoup(r.content, 'html.parser')

    body = soup.find('body')


    properties2 = body.select("div.broker-card__info")
```

```
    m = properties2[0].get_text().replace('\n', '').replace('Kontakta␣
↪mäklarkontoret','').replace('Kontakta mäklaren','').replace('Kontakt','').
↪strip().split("      ")

    # Note that some links only provide agency without broker info.
    if len(m) == 2:
        brokers[link_ind] = m[0]
        agencies[link_ind] = m[1]
    else:
        agencies[link_ind] = m[0]
```

[ ]:

```
d = {'Agents': brokers, 'Agencies': third_agency}
agent_df = pd.DataFrame(data=d)
agent_df.to_csv('agent.csv', index=False)
```

# E Data Cleaning

[1]:
```
import pandas as pd
```

## E.1 Clean second layer info except broker/agencies

[2]:
```
snd_df = pd.read_csv('unprocessed_sndlayer_info.csv')
snd_df.head(10)
```

[2]:
```
                     0                    1                    2  \
0          32609kr/m²           15900kr/m²           34043kr/m²
1            695000kr             795000kr             725000kr
2     +55000 kr (+8%)                  NaN     +75000 kr (+10%)
3            Lägenhet             Lägenhet             Lägenhet
4         Bostadsrätt          Bostadsrätt          Bostadsrätt
5               1 rum                2 rum                1 rum
6               23 m²                50 m²              23,5 m²
7                 Nej  2 av 2, hiss finns ej  2 av 2, hiss finns ej
8  1 av 3, hiss finns ej                2004                 1957
9                1956            4011kr/mån           1836kr/mån

                 3                    4                    5  \
0       26129kr/m²           34468kr/m²           19318kr/m²
1         795000kr             750000kr             850000kr
2  +15000 kr (+2%)      +60000 kr (+8%)                  NaN
3         Lägenhet             Lägenhet             Lägenhet
4      Bostadsrätt          Bostadsrätt          Bostadsrätt
5            1 rum                1 rum                2 rum
6            31 m²              23,5 m²                44 m²
7           2 av 2  1 av 2, hiss finns ej  2 av 2, hiss finns ej
8             2018                 1957                 1953
9       1770kr/mån           1786kr/mån           2967kr/mån

                     6                   7           8  \
0          19000kr/m²          31847kr/m²  16667kr/m²
1            975000kr            950000kr   1050000kr
2     -25000 kr (-3%)     +50000 kr (+5%)         NaN
3            Lägenhet            Lägenhet    Lägenhet
4         Bostadsrätt         Bostadsrätt Bostadsrätt
5               1 rum             1,5 rum       2 rum
6               50 m²             31,4 m²       63 m²
7                  Ja                 Nej      2 av 2
8  2 av 2, hiss finns ej  1 av 8, hiss finns        2004
9                1971                1964  4987kr/mån

                     9  ...                 1962        1963  \
0          28514kr/m²  ...          63194kr/m²  44498kr/m²
1            895000kr  ...           7995000kr    9300000kr
2    +160000 kr (+18%)  ...  +1,11milj. kr (+14%)         NaN
```

```
3               Lägenhet  ...              Lägenhet     Lägenhet
4           Bostadsrätt  ...           Bostadsrätt  Bostadsrätt
5                 1 rum  ...                 5 rum        6 rum
6                 37 m²  ...                144 m²       209 m²
7  1 av 3, hiss finns ej  ...                   Nej           Ja
8                  1971  ...                    Ja  10626kr/mån
9           2228kr/mån  ...  1 av 2, hiss finns ej          NaN

           1964                   1965                   1966  \
0   44498kr/m²             64901kr/m²             40204kr/m²
1    9300000kr              6995000kr              8500000kr
2          NaN   +2,81milj. kr (+40%)  +1,35milj. kr (+16%)
3     Lägenhet               Lägenhet               Lägenhet
4  Bostadsrätt  Andel i bostadsförening            Bostadsrätt
5        6 rum                  5 rum                  6 rum
6       209 m²                 151 m²                 245 m²
7  10626kr/mån                    Nej                     Ja
8          NaN    1 av 3, hiss finns ej    2 av 5, hiss finns
9          NaN                   1903                   1890

                  1967                   1968                   1969  \
0           63368kr/m²             63905kr/m²             69410kr/m²
1            9500000kr              9500000kr             10000000kr
2    +1milj. kr (+11%)    +1,3milj. kr (+14%)   +1,05milj. kr (+11%)
3             Lägenhet               Lägenhet               Lägenhet
4          Bostadsrätt            Bostadsrätt            Bostadsrätt
5                6 rum                  5 rum                  5 rum
6             165,7 m²                 169 m²               159,2 m²
7                   Ja                     Ja                     Ja
8  5 av 5, hiss finns  2 av 4, hiss finns ej  3 av 4, hiss finns ej
9           8008kr/mån                   1904                   1903

                   1970                          1971
0            69410kr/m²                    85124kr/m²
1            10000000kr                    11600000kr
2  +1,05milj. kr (+11%)              +62000 kr (+1%)
3              Lägenhet                      Lägenhet
4           Bostadsrätt                   Bostadsrätt
5                 5 rum                         4 rum
6              159,2 m²                        137 m²
7                    Ja                    6909kr/mån
8  3 av 4, hiss finns ej  Brf Kulturkvarteret i Lund
9                  1903                           NaN

[10 rows x 1972 columns]
```

### E.1.1 Asking prices

[3]:
```
asking_price=snd_df.iloc[1].str.replace('kr', '').astype(float)
asking_price.head()
```

[3]:
```
0    695000.0
1    795000.0
2    725000.0
3    795000.0
4    750000.0
Name: 1, dtype: float64
```

### E.1.2 Building year

[6]:
```
build_year = [None for _ in range(1972)]

for i in range(1972):
    snd_df_col = snd_df[str(i)]
    for element in snd_df_col:
        # here we need to check if the element is string because we set␣
↪asking_price data type
        # to be float which does not have length.
        if type(element)==str and len(element) == 4:
            build_year[i] = int(element)

build_year[0:6]
```

[6]:
```
[1956, 2004, 1957, 2018, 1957, 1953]
```

### E.1.3 Wether there is a balcony/patio

**Let's check how many sold apartments provide the info of balcony/patio**

[9]:
```
is_balcony_count = [None for _ in range(1972)]
for i in range(1972):
    snd_df_col = snd_df[str(i)]
    Nej_count = list(snd_df_col).count('Nej')
    Ja_count = list(snd_df_col).count('Ja')
    is_balcony_count[i] = Nej_count + Ja_count
```

[10]:
```
# 242 apartments do not provide this info
is_balcony_count.count(0)
```

[10]:
```
242
```

[11]:
```
# 1520 apartments only provide whether there is a balcony
is_balcony_count.count(1)
```

[11]:
```
1520
```

[12]:
```python
# 210 apartments provide both balcony and patio info
is_balcony_count.count(2)
```

[12]: 210

**Corresponding indices**

[13]:
```python
balcony_index = [i for i, e in enumerate(is_balcony_count) if e == 1]
balcony_patio_index = [i for i, e in enumerate(is_balcony_count) if e == 2]
```

**Now we can get the info from the items with given indices**

[14]:
```python
is_balcony = [None for _ in range(1972)]
is_patio = [None for _ in range(1972)]
```

[18]:
```python
# For the links which only provide balcony info, there is either 'Ja' or␣
↪'Nej'

for ind in balcony_index:
    snd_df_col = snd_df[str(ind)]
    for element in snd_df_col:
        if element == 'Ja' or element == 'Nej':
            is_balcony[ind] = element
```

[19]:
```python
# For the links provide both the balcony and patio info, the 7th element␣
↪gives balcony and 8th element gives patio

for ind in balcony_patio_index:
    snd_df_col = snd_df[str(ind)]
    is_balcony[ind] = snd_df_col[7]
    is_patio[ind] = snd_df_col[8]
```

[20]:
```python
is_balcony[0:10]
```

[20]: ['Nej', None, None, None, None, None, 'Ja', 'Nej', None, None]

[22]:
```python
is_patio[5:15]
```

[22]: [None, None, None, None, None, 'Ja', None, None, None, None]

### E.1.4 Total number of building floors/Apartment floor number/If elevator is available.

[25]:
```python
floor_elevator = [None for _ in range(1972)]

for i in range(1972):
    snd_df_col = snd_df[str(i)]
    for element in snd_df_col:
        if type(element)==str and element.count('av') == 1:
```

```
            floor_elevator[i] = element

floor_elevator[0:5]
```

[25]:
```
['1 av 3, hiss finns ej',
 '2 av 2, hiss finns ej',
 '2 av 2, hiss finns ej',
 '2 av 2',
 '1 av 2, hiss finns ej']
```

**From the above output, we find that building floors/Apartment floor number is separated by the string 'av', and is_elevator is obtained simply by first replacing 'hiss finns ej' as 'No' and 'hiss finns' as 'Yes'.**

[26]:
```
floor_number = [None for _ in range(1972)]
total_floor = [None for _ in range(1972)]
is_elevator = [None for _ in range(1972)]
```

[27]:
```
for i in range(1972):
    if floor_elevator[i]:
        element = floor_elevator[i].split(', ')

        if len(element) == 2:
            is_elevator[i] = element[1]
            floor_info = element[0].split('av')
            floor_number[i] = floor_info[0].strip()
            total_floor[i] = floor_info[1].strip()
        elif len(element) == 1:
            floor_info = element[0].split('av')
            floor_number[i] = floor_info[0].strip()
            total_floor[i] = floor_info[1].strip()
```

[28]:
```
floor_number[0:6]
```

[28]:
```
['1', '2', '2', '2', '1', '2']
```

[29]:
```
total_floor[0:6]
```

[29]:
```
['3', '2', '2', '2', '2', '2']
```

[31]:
```
is_elevator = list(pd.Series(is_elevator).replace('hiss finns ej','No').
 ↪replace('hiss finns','Yes'))
is_elevator[0:10]
```

[31]:
```
['No', 'No', 'No', None, 'No', 'No', 'No', 'Yes', None, 'No']
```

[ ]:
```
d = {'Average': average, 'Asking price': asking_price,'Balcony': is_balcony,␣
 ↪'Patio': is_patio,'Build year': build_year, 'Floor number': floor_number,␣
 ↪'Total floor': total_floor, 'Elevator': is_elevator}
snd_layer_df = pd.DataFrame(data=d)
```

```python
snd_layer_df.to_csv('snd_layer_df_info.csv', index=False)
```

## E.2 Clean first layer info

[152]:
```python
df = pd.read_csv('hemnet.csv')
df.head()
```

[152]:
```
            Addresses     Types area (m²)  # of rooms  Monthly Fees (Kr)  \
0    Flormansgatan 2A  Lägenhet        43         1.5             2767.0
1  Kastanjegatan 19F  Lägenhet        34         2.0             2415.0
2    Karl XI gatan 47  Lägenhet      87,4         3.0             5787.0
3          Äspet 163     Villa  158 + 22         8.0                NaN
4  Margaretavägen 3K  Lägenhet        78         3.0             4584.0

          Sold Dates                                              Links  \
0  30 september 2021  https://www.hemnet.se/salda/lagenhet-1,5rum-ce...
1  30 september 2021  https://www.hemnet.se/salda/lagenhet-2rum-jarn...
2  30 september 2021  https://www.hemnet.se/salda/lagenhet-3rum-lund...
3  30 september 2021  https://www.hemnet.se/salda/villa-8rum-lunds-k...
4  30 september 2021  https://www.hemnet.se/salda/lagenhet-3rum-moll...

   Prices (tKr)
0          2370
1          1745
2          4700
3          5350
4          2750
```

### E.2.1 Separate Apartment (Lägenhet) from other housing types

[153]:
```python
apart_df = df[df['Types'] == 'Lägenhet']
apart_df.head()
```

[153]:
```
               Addresses     Types area (m²)  # of rooms  Monthly Fees (Kr)  \
0       Flormansgatan 2A  Lägenhet        43         1.5             2767.0
1      Kastanjegatan 19F  Lägenhet        34         2.0             2415.0
2       Karl XI gatan 47  Lägenhet      87,4         3.0             5787.0
4      Margaretavägen 3K  Lägenhet        78         3.0             4584.0
5  Qvantenborgsvägen 4B  Lägenhet        59         2.0             3125.0

          Sold Dates                                              Links  \
0  30 september 2021  https://www.hemnet.se/salda/lagenhet-1,5rum-ce...
1  30 september 2021  https://www.hemnet.se/salda/lagenhet-2rum-jarn...
2  30 september 2021  https://www.hemnet.se/salda/lagenhet-3rum-lund...
4  30 september 2021  https://www.hemnet.se/salda/lagenhet-3rum-moll...
5  29 september 2021  https://www.hemnet.se/salda/lagenhet-2rum-kobj...

   Prices (tKr)
```

```
0          2370
1          1745
2          4700
4          2750
5          2250
```

[155]:
```
apart_df.shape
```

[155]:
```
(1972, 8)
```

### E.2.2 Clean the data in the column 'area'

[121]:
```
area = apart_df['area (m²)']
area.head()
```

[121]:
```
0      43
1      34
2    87,4
4      78
5      59
Name: area (m²), dtype: object
```

[122]:
```
# We first clean the values contain '+' sign by removing the number after it.
↪
# By comparing the sold price and price/m², seems these numbers are not␣
↪counted.

irregular_values = apart_df[apart_df['area (m²)'].str.contains('+',␣
↪regex=False)]['area (m²)']
irregular_values.head()
```

[122]:
```
69       60 + 20
91       44 + 20
154    48,4 + 20
485    75,5 + 21
958      89 + 50
Name: area (m²), dtype: object
```

[123]:
```
regular_values = irregular_values.str.split('+').str[0]
regular_values.head()
```

[123]:
```
69       60
91       44
154    48,4
485    75,5
958      89
Name: area (m²), dtype: object
```

[124]:
```python
irregular_index = apart_df[apart_df['area (m²)'].str.contains('+',␣
↪regex=False)].index.values

for ind in irregular_index:
    area = area.replace(area[ind], regular_values[ind])
```

[125]:
```python
area[91]
```

[125]:
```
'44 '
```

[126]:
```python
# We also replace comma with period.

area = area.str.replace(',','.')
area.head()
```

[126]:
```
0      43
1      34
2    87.4
4      78
5      59
Name: area (m²), dtype: object
```

[160]:
```python
# Now replace the column with the cleaned values.

pd.options.mode.chained_assignment = None  # default='warn'
apart_df['area (m²)'] = area
# apart_df = apart_df[apart_df['area (m²)'] == area] # sth wrong with this␣
↪line of code which changes apart_df shape.
apart_df.head()
```

[160]:
```
               Addresses     Types area (m²)  # of rooms  Monthly Fees (Kr)  \
0       Flormansgatan 2A  Lägenhet        43         1.5             2767.0
1      Kastanjegatan 19F  Lägenhet        34         2.0             2415.0
2       Karl XI gatan 47  Lägenhet      87.4         3.0             5787.0
4      Margaretavägen 3K  Lägenhet        78         3.0             4584.0
5   Qvantenborgsvägen 4B  Lägenhet        59         2.0             3125.0

           Sold Dates                                             Links  \
0  30 september 2021  https://www.hemnet.se/salda/lagenhet-1,5rum-ce...
1  30 september 2021  https://www.hemnet.se/salda/lagenhet-2rum-jarn...
2  30 september 2021  https://www.hemnet.se/salda/lagenhet-3rum-lund...
4  30 september 2021  https://www.hemnet.se/salda/lagenhet-3rum-moll...
5  29 september 2021  https://www.hemnet.se/salda/lagenhet-2rum-kobj...

   Prices (tKr)
0          2370
1          1745
2          4700
4          2750
5          2250
```

[161]:
```
apart_df.shape
```

[161]:
```
(1972, 8)
```

### E.2.3 Change the format of Sold Dates

[162]:
```
Dates = apart_df['Sold Dates']
Dates=Dates.str.replace(' januari ','/01/').str.replace(' februari ','/02/').
↪str.replace(' mars ','/03/').str.replace(' april ','/04/').str.replace('␣
↪maj ','/05/').str.replace(' juni ','/06/').str.replace(' juli ','/07/').
↪str.replace(' augusti ','/08/').str.replace(' september ','/09/').str.
↪replace(' oktober ','/10/').str.replace(' november ','/11/').str.replace('␣
↪december ','/12/')
Dates.head()
```

[162]:
```
0    30/09/2021
1    30/09/2021
2    30/09/2021
4    30/09/2021
5    29/09/2021
Name: Sold Dates, dtype: object
```

[163]:
```
Dates = pd.to_datetime(Dates)
apart_df['Sold Dates'] = Dates.values
apart_df.head()
```

[163]:
```
                Addresses     Types area (m²)  # of rooms  Monthly Fees (Kr)  \
0      Flormansgatan 2A  Lägenhet         43         1.5             2767.0
1     Kastanjegatan 19F  Lägenhet         34         2.0             2415.0
2      Karl XI gatan 47  Lägenhet       87.4         3.0             5787.0
4    Margaretavägen 3K  Lägenhet         78         3.0             4584.0
5  Qvantenborgsvägen 4B  Lägenhet         59         2.0             3125.0

  Sold Dates                                              Links  Prices (tKr)
0 2021-09-30  https://www.hemnet.se/salda/lagenhet-1,5rum-ce...          2370
1 2021-09-30  https://www.hemnet.se/salda/lagenhet-2rum-jarn...          1745
2 2021-09-30  https://www.hemnet.se/salda/lagenhet-3rum-lund...          4700
4 2021-09-30  https://www.hemnet.se/salda/lagenhet-3rum-moll...          2750
5 2021-09-29  https://www.hemnet.se/salda/lagenhet-2rum-kobj...          2250
```

[164]:
```
apart_df.shape
```

[164]:
```
(1972, 8)
```

### E.2.4 Drop the columns 'Types' and 'Links'

[165]:
```
apart_df = apart_df.drop(columns='Types')
```

[166]: 
```
apart_df = apart_df.drop(columns='Links')
```

[167]: 
```
apart_df.head()
```

[167]:
```
               Addresses area (m²)  # of rooms  Monthly Fees (Kr) Sold Dates ␣
↪\
0      Flormansgatan 2A        43         1.5             2767.0 2021-09-30
1     Kastanjegatan 19F        34         2.0             2415.0 2021-09-30
2      Karl XI gatan 47      87.4         3.0             5787.0 2021-09-30
4     Margaretavägen 3K        78         3.0             4584.0 2021-09-30
5  Qvantenborgsvägen 4B        59         2.0             3125.0 2021-09-29

   Prices (tKr)
0          2370
1          1745
2          4700
4          2750
5          2250
```

### E.2.5 Reorder the rows as the increasing sold prices

[168]: 
```
apart_df = apart_df.sort_values(['Prices (tKr)'], ascending=1)
apart_df.head()
```

[168]:
```
               Addresses area (m²)  # of rooms  Monthly Fees (Kr) Sold Dates ␣
↪\
565   Veberödsvägen 22C        23         1.0             1287.0 2021-04-09
319     Idalavägen 47 f        50         2.0             4011.0 2020-10-21
2007        Allégatan 3F      23.5         1.0             1836.0 2020-11-15
259        Horstgatan 4H        31         1.0             1770.0 2020-10-29
2411        Allégatan 3F      23.5         1.0             1786.0 2021-03-01

      Prices (tKr)
565            750
319            795
2007           800
259            810
2411           810
```

[169]: 
```
apart_df.shape
```

[169]: (1972, 6)

### E.2.6 Drop the index and add other features from second layer info

[170]: 
```
new_apart_df = apart_df.reset_index()
```

[171]: 
```
new_apart_df.head(3)
```

```
[171]:    index          Addresses area (m²)  # of rooms  Monthly Fees (Kr)  \
       0    565  Veberödsvägen 22C        23         1.0             1287.0
       1    319    Idalavägen 47 f        50         2.0             4011.0
       2   2007       Allégatan 3F      23.5         1.0             1836.0

         Sold Dates  Prices (tKr)
       0 2021-04-09           750
       1 2020-10-21           795
       2 2020-11-15           800
```

```
[172]: agent_df = pd.read_csv('agent.csv')
       snd_info_df = pd.read_csv('snd_layer_df_info.csv')
```

```
[213]: # Join 3 dataframes

       new_df = pd.concat([new_apart_df, snd_info_df, agent_df], axis=1)
```

```
[214]: # Get rid of NaN

       new_df = new_df.fillna('')
       new_df.head()
```

```
[214]:    index          Addresses area (m²)  # of rooms  Monthly Fees (Kr)  \
       0    565  Veberödsvägen 22C        23         1.0             1287.0
       1    319    Idalavägen 47 f        50         2.0             4011.0
       2   2007       Allégatan 3F      23.5         1.0             1836.0
       3    259      Horstgatan 4H        31         1.0             1770.0
       4   2411       Allégatan 3F      23.5         1.0             1786.0

         Sold Dates  Prices (tKr)  Average Asking price Balcony Patio Build year  \
       0 2021-04-09           750  32609.0       695000     Nej            1956
       1 2020-10-21           795  15900.0       795000                    2004
       2 2020-11-15           800  34043.0       725000                    1957
       3 2020-10-29           810  26129.0       795000                    2018
       4 2021-03-01           810  34468.0       750000                    1957

         Floor number Total floor Elevator          Agents  \
       0            1           3       No   Karin Ekström
       1            2           2       No  Rickard Saltin
       2            2           2       No
       3            2           2
       4            1           2       No

                                Agencies
       0  Erik Olsson Fastighetsförmedling
       1              Fastighetsbyrån Lund
       2              Fastighetsbyrån Lund
       3  Svensk Fastighetsförmedling Lund
       4              Fastighetsbyrån Lund
```

[99]:
```
# 'Average': average, 'Asking price': asking_price,'Balcony': is_balcony,
#'Patio': is_patio,'Build year': build_year, 'Floor number': floor_number,
#'Total floor': total_floor, 'Elevator': is_elevator}
```

### E.2.7 Change data type/reorder columns

**We change the column ‘asking price’ also to ‘asking price (tkr)’**

[210]:
```
# Two apartments do not have asking prices

none_index=[i for i,v in enumerate(new_df['Asking price']) if v == '']
none_index
```

[210]:
```
[1399, 1473]
```

[211]:
```
# We replace these 2 values to 0. Remember to drop these 2 values in␣
↪analysis using this feature!

ask_price = new_df['Asking price'].replace('',0)
ask_price_tkr = ask_price/1000
ask_price_tkr = ask_price_tkr.astype(int)
```

[215]:
```
new_df = new_df.drop(columns='index')
new_df = new_df.drop(columns='Asking price')
new_df['Asking (tKr)'] = ask_price_tkr
```

[221]:
```
apartment_df = new_df.reindex(columns=['Addresses', 'area (m²)', '# of␣
↪rooms', 'Balcony', 'Patio', 'Elevator', 'Floor number', 'Total floor',␣
↪'Monthly Fees (Kr)', 'Build year', 'Asking (tKr)', 'Prices (tKr)',␣
↪'Average', 'Agents', 'Agencies', 'Sold Dates'])
```

[222]:
```
apartment_df.head(3)
```

[222]:
```
            Addresses area (m²)  # of rooms Balcony Patio Elevator  \
0  Veberödsvägen 22C        23         1.0     Nej              No
1     Idalavägen 47 f       50         2.0                      No
2        Allégatan 3F      23.5        1.0                      No

  Floor number Total floor  Monthly Fees (Kr) Build year  Asking (tKr)  \
0            1           3             1287.0       1956           695
1            2           2             4011.0       2004           795
2            2           2             1836.0       1957           725

   Prices (tKr)  Average         Agents                       Agencies  \
0           750  32609.0  Karin Ekström  Erik Olsson Fastighetsförmedling
1           795  15900.0 Rickard Saltin            Fastighetsbyrån Lund
2           800  34043.0                           Fastighetsbyrån Lund

  Sold Dates
0 2021-04-09
```

1 2020-10-21
2 2020-11-15

[ ]:

```

```

[223]:

```
apartment_df.to_csv('apartment_df.csv')
```

# F Feature Engineering and Model pre-training

## F.1 Overlay data on dynamic Google map

### F.1.1 Use Geopy to get lat-lon coordinates

[1]:

```
from geopy.geocoders import Nominatim
```

[2]:

```
# As a test, we use geopy to Get the lat-lon coordinate
# of the Kemicentrum, Lund University

geolocator = Nominatim(user_agent='myGeocoder')
location = geolocator.geocode("Kemicentrum, Lund")
print(location.latitude, location.longitude)
```

55.7164444 13.2094047

[ ]:

```
# Read CSV file
df = pd.read_csv('apartment_df.csv')
```

[ ]:

```
# We add string ', Lund' to all addresses to avoid confusion
# as other cities may have the same addresses.

length = len(df['Addresses'])
address_lund = [None for _ in range(length)]

for i in range(length):
    address_lund[i] = df['Addresses'][i] + ', Lund'
```

[ ]:

```
# Now we may get lat-lon coordinates for all modified addresses

lat = [None for _ in range(length)]
lon = [None for _ in range(length)]

for i in range(length):
    location = geolocator.geocode(address_lund[i])
    if location: # there are several Nones
        lat[i] = location.latitude
        lon[i] = location.longitude
```

[ ]:
```
# Deal with the Nones

l=[i for i,v in enumerate(lat) if v == None]
```

[ ]:
```
# We find that some addresses have the prefix 'Aromalund -'
# and some have room number within brackets e.g.(lgh 1003) which need to be␣
↪removed.

import re

for ind in l:
    address_lund[ind] = address_lund[ind].replace('Aromalund - ','').
↪replace(' - Kulturkvarteret','')
    address_lund[ind] = re.sub(r"[\(\[].*?[\)\]]", "", address_lund[ind])
```

[ ]:
```
# Let's check again and now there are 27 Nones

l2=[i for i,v in enumerate(lat) if v == None]
print(len(l2))
print(l2)
```

```
27
[238, 332, 333, 372, 394, 395, 617, 618, 619, 750, 942, 1036, 1093, 1260,␣
↪1309,
1429, 1597, 1599, 1631, 1686, 1697, 1759, 1832, 1849, 1908, 1917, 1926]
```

[ ]:
```
# We print them out:

for ind2 in l2:
    print(address_lund[ind2])
```

```
Prins August gata, Lund
Parternas Gränd 55 LGH 1002, Lund
Måsvägen 3 C lgh 1003, Lund
Prins August gata, Lund
Dalbyvägen 20 - lgh 2106, Lund
Tellusgatan 3 - lgh 2106, Lund
Arkitektritade taklägenheter, Lund
Arkitektritade taklägenheter, Lund
Arkitektritade taklägenheter, Lund
Sofiavägen 3 A lgh 1203, Lund
Prins August gata, Lund
Prins Augusts gatan 14, Lund
Prins August gata, Lund
Prins August gata 14, Lund
Prins Augusts gatan 10, Lund
Prins August gatan 14, Lund
Prins August Gata 8, Lund
Prins August gata, Lund
Spolebacken 2, Lund
Prins August gata, Lund
```

```
Prins August gata, Lund
Spolebacken 2, Lund
Prins August gata, Lund
Prins August gata, Lund
Prins August gatan 12, Lund
Prins August gata, Lund
Karl XII gata 9, Lund
```

For these strings, we may manually correct them. For the items with number 6,7,8,18,21, the address info were not provided, which we will delete them in the following analysis.

[ ]:
```
address_lund[l2[0]] = 'Prins Augusts Gata, Lund'
address_lund[l2[1]] = 'Parternas Gränd 55, Lund'
address_lund[l2[2]] = 'Måsvägen 3C, Lund'
address_lund[l2[3]] = 'Prins Augusts Gata, Lund'
address_lund[l2[4]] = 'Dalbyvägen 20, Lund'
address_lund[l2[5]] = 'Tellusgatan 3, Lund'
#address_lund[l2[6]] = 'Prins Augusts Gata, Lund'
#address_lund[l2[7]] = 'Prins Augusts Gata, Lund'
#address_lund[l2[8]] = 'Prins Augusts Gata, Lund'

address_lund[l2[9]] = 'Sofiavägen 3 A, Lund'
address_lund[l2[10]] = 'Prins Augusts Gata, Lund'
address_lund[l2[11]] = 'Prins Augusts Gata, Lund'
address_lund[l2[12]] = 'Prins Augusts Gata, Lund'
address_lund[l2[13]] = 'Prins Augusts Gata, Lund'
address_lund[l2[14]] = 'Prins Augusts Gata, Lund'
address_lund[l2[15]] = 'Prins Augusts Gata, Lund'
address_lund[l2[16]] = 'Prins Augusts Gata, Lund'
address_lund[l2[17]] = 'Prins Augusts Gata, Lund'

#address_lund[l2[18]] = 'Prins Augusts Gata, Lund'
address_lund[l2[19]] = 'Prins Augusts Gata, Lund'
address_lund[l2[20]] = 'Prins Augusts Gata, Lund'
#address_lund[l2[21]] = 'Prins Augusts Gata, Lund'
address_lund[l2[22]] = 'Prins Augusts Gata, Lund'
address_lund[l2[23]] = 'Prins Augusts Gata, Lund'
address_lund[l2[24]] = 'Prins Augusts Gata, Lund'
address_lund[l2[25]] = 'Prins Augusts Gata, Lund'
address_lund[l2[26]] = 'Karl XII gatan 9, Lund'
```

[ ]:
```
# But we first add these coordinates and save as a new csv file

df['Lat']=lat
df['Lon']=lon

df.to_csv('apartment_df_latlon.csv')
```

### F.1.2 Display data on Google map

[7]:
```
# import necessary libraries

import os
import pandas as pd
from bokeh.io import output_notebook
output_notebook()
```

[4]:
```
# Google map API key (not displayed)
api_key = 'xxxxxxx'
```

[ ]:
```
# Drop the 5 None values, fill other NANs, reset the index.
df = pd.read_csv('apartment_df_latlon.csv')
new_df = df.dropna(subset=['Lat'])
new_df = new_df.drop(columns=['Unnamed: 0', 'Unnamed: 0.1'])
new_df = new_df.fillna('')
new_df = new_df.reset_index()
```

[ ]:
```
# Define the size and display center of the map

bokeh_width, bokeh_height = 500,400
center_lat = new_df['Lat'][1966] # the most expensive one as center
center_lon = new_df['Lon'][1966]
```

[ ]:
```
# Plot a static map with the center point

from bokeh.io import show
from bokeh.plotting import gmap
from bokeh.models import GMapOptions

def plot(lat, lng, zoom=10, map_type='roadmap'):
    gmap_options = GMapOptions(lat=lat, lng=lng,
                               map_type=map_type, zoom=zoom)
    p = gmap(api_key, gmap_options, title='Lund',
             width=bokeh_width, height=bokeh_height)

    center = p.circle([lng], [lat], size=10, alpha=0.5, color='red')
    show(p)
    return p
```

[ ]:
```
p = plot(center_lat, center_lon)
```

[ ]:
```
# Now let's plot the data points with hover tools

from bokeh.io import show
from bokeh.plotting import gmap
from bokeh.models import GMapOptions
from bokeh.models import ColumnDataSource
from bokeh.models import HoverTool
```

```python
from bokeh.transform import linear_cmap
from bokeh.palettes import Plasma256 as palette
from bokeh.models import ColorBar

def plot(lat, lng, zoom=12, map_type='roadmap'):
    gmap_options = GMapOptions(lat=lat, lng=lng,
                               map_type=map_type, zoom=zoom)

    p = gmap(api_key, gmap_options, title='Lund',
            width=bokeh_width, height=bokeh_height,
            tools=['hover', 'reset', 'wheel_zoom', 'pan'])

    source = ColumnDataSource(new_df)

    center = p.circle('Lon', 'Lat', size=4, alpha=0.2,
                      color='red', source=source)

    show(p)
    return p
```

[ ]:
```python
p = plot(center_lat, center_lon, map_type='roadmap', zoom=12)
```

[ ]:
```python
# We need to replace the columns' name with spaces. We should aovid
# use such notation later!

df2 = new_df.copy()
df2.columns = [c.replace(' ', '_') for c in df2.columns]
df2 = df2.rename(columns={'Prices_(tKr)': 'price','area_(m²)':'Size',␣
 ↪'Monthly_Fees_(Kr)':'Avgift', 'Sold_Dates':'SoldDate'})
```

[ ]:
```python
def plot(lat, lng, zoom=12, map_type='roadmap'):
    gmap_options = GMapOptions(lat=lat, lng=lng,
                               map_type=map_type, zoom=zoom)

    hover = HoverTool(
        tooltips = [
            ('Addresses', '@Addresses'),
            ('Prices (tKr)', '@price'),
            ('Average (Kr)', '@Average{0.}'),
            ('Size_m2', '@Size{0.0}'),
            ('Floor', '@Floor_number'),
            ('Avgift (Kr)', '@Avgift{0.}'),
            ('BuildYear', '@Build_year'),
            ('SoldDate', '@SoldDate'),
        ]
    )

    p = gmap(api_key, gmap_options, title='Lund',
            width=bokeh_width, height=bokeh_height,
            tools=[hover, 'reset', 'wheel_zoom', 'pan'])
```

```python
    # center = p.circle([lng], [lat], size=10, alpha=0.5, color='red')

    source = ColumnDataSource(df2)

    mapper = linear_cmap('price', palette, 700., 12000.)

    center = p.circle('Lon', 'Lat', size=4, alpha=0.2,
                      color=mapper, source=source)

    color_bar = ColorBar(color_mapper=mapper['transform'],
                         location=(0,0))
    p.add_layout(color_bar, 'right')

    show(p)
    return p
```

[ ]:
```python
p = plot(center_lat, center_lon, map_type='roadmap', zoom=11)
```

[ ]:
```python
# We can also plot as average price (kr/m^2)

df2['Average'].max()
df2['Average'].min()
```

[ ]:
```python
def plot2(lat, lng, zoom=12, map_type='roadmap'):
    gmap_options = GMapOptions(lat=lat, lng=lng,
                               map_type=map_type, zoom=zoom)

    hover = HoverTool(
        tooltips = [
            ('Addresses', '@Addresses'),
            ('Prices (tKr)', '@price'),
            ('Average (Kr)', '@Average{0.}'),
            ('Size_m2', '@Size{0.0}'),
            ('Floor', '@Floor_number'),
            ('Avgift (Kr)', '@Avgift{0.}'),
            ('BuildYear', '@Build_year'),
            ('SoldDate', '@SoldDate'),
        ]
    )

    p = gmap(api_key, gmap_options, title='Lund',
             width=bokeh_width, height=bokeh_height,
             tools=[hover, 'reset', 'wheel_zoom', 'pan'])

    source = ColumnDataSource(df2)

    mapper = linear_cmap('Average', palette, 15000., 130000.)

    center = p.circle('Lon', 'Lat', size=4, alpha=0.2,
                      color=mapper, source=source)
```

```
    color_bar = ColorBar(color_mapper=mapper['transform'],
                         location=(0,0))
    p.add_layout(color_bar, 'right')

    show(p)
    return p
```

[ ]:
```
p = plot2(center_lat, center_lon, map_type='roadmap', zoom=11)
```

[184]:
```
# Import necessary libiraries

import pandas as pd
import numpy as np
import math
import matplotlib.pyplot as plt
import seaborn as sns
import pickle

from geopy import distance
from scipy.stats import norm, skew
from scipy import stats

from sklearn.model_selection import train_test_split
from xgboost import XGBRegressor
from sklearn.ensemble import RandomForestRegressor,␣
↪GradientBoostingRegressor, AdaBoostRegressor, BaggingRegressor
from sklearn.metrics import mean_absolute_error
```

[ ]:
```
# Read csv file with lat-lon coordinates

df = pd.read_csv('apartment_df_latlon.csv')
df = df.drop(columns=['Unnamed: 0', 'Unnamed: 0.1'])
df = df.reset_index()
```

[ ]:
```
# Remove 4 columns that are not needed for the model prediction

df = df.drop(columns=['Asking (tKr)', 'Agents', 'Agencies', 'Sold Dates',␣
↪'Average'])
```

## F.2 Missing data

[46]:
```
# Check missing data
total = df.isnull().sum().sort_values(ascending=False)
percent = (df.isnull().sum()/df.isnull().count()).
↪sort_values(ascending=False)
missing_data = pd.concat([total, percent], axis=1, keys=['Total', 'Percent'])
missing_data.head(20)
```

[46]:

| | Total | Percent |
|---|---|---|
| Patio | 1762 | 0.893509 |
| Elevator | 304 | 0.154158 |
| Build year | 296 | 0.150101 |
| Balcony | 242 | 0.122718 |
| Floor number | 221 | 0.112069 |
| Total floor | 221 | 0.112069 |
| Lat | 5 | 0.002535 |
| Lon | 5 | 0.002535 |
| index | 0 | 0.000000 |

```
Addresses              0  0.000000
area (m²)              0  0.000000
# of rooms             0  0.000000
Monthly Fees (Kr)      0  0.000000
Prices (tKr)           0  0.000000
Average                0  0.000000
Month                  0  0.000000
```

[47]:
```
# Remove the feature 'patio' as too many values are missing

df = df.drop(columns=['Patio'])
```

[47]:
```
   index          Addresses  area (m²)  # of rooms Balcony Elevator  \
0      0  Veberödsvägen 22C       23.0         1.0     Nej       No
1      1    Idalavägen 47 f       50.0         2.0     NaN       No

  Floor number  Total floor  Monthly Fees (Kr)  Build year  Prices (tKr)  \
0            1          3.0             1287.0      1956.0           750
1            2          2.0             4011.0      2004.0           795

   Month        Lat        Lon
0      4  55.662566  13.352051
1     10  55.621239  13.500930
```

## F.3 More feature engineering

[63]:
```
# make a df copy

test_df = df.copy()
```

[65]:
```
# Fill Nans to 0 as Nan means no balcony/elevator,
# then we transform "Balcony" & "Elevator" from categorical to numerical

test_df['Balcony']=test_df['Balcony'].fillna('0')
test_df['Balcony']=test_df['Balcony'].replace('Nej', '0')
test_df['Balcony']=test_df['Balcony'].replace('Ja', '1')

test_df['Elevator']=test_df['Elevator'].fillna('0')
test_df['Elevator']=test_df['Elevator'].replace('No', '0')
test_df['Elevator']=test_df['Elevator'].replace('Yes', '1')
```

[68]:
```
# Fill Nans to 1 as Nan means the buidling has only 1 floor.

test_df['Floor number']=test_df['Floor number'].fillna('1')
test_df['Total floor']=test_df['Total floor'].fillna('1')
```

[69]:
```
# Replace Nans to the mean value for the "Build year" and set the data type␣
↪as integer
```

```
test_df['Build year']=test_df['Build year'].fillna(test_df['Build year'].
 ↪mean())
test_df['Build year']=test_df['Build year'].astype(int)
```

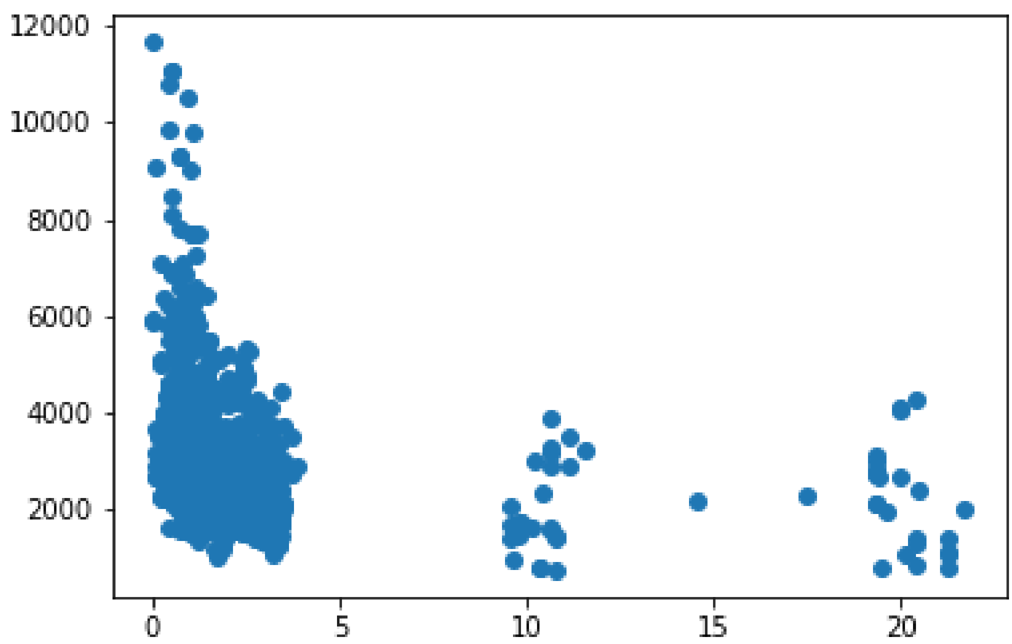
[71]:
```
# Replace Nans with the most common values for the lat-lon coordinates.

test_df['Lat']=test_df['Lat'].fillna(test_df['Lat'].mode()[0])
test_df['Lon']=test_df['Lon'].fillna(test_df['Lon'].mode()[0])
```

### F.4 Outliers

[72]:
```
# Set the location of the apartment with highest sold price as reference
# and calculate the distances of the other apartments to this reference␣
 ↪point.

length = len(test_df['Lat'])
dist = [None for _ in range(length)]

for i in range(length):
    ref = (test_df['Lat'][1971], test_df['Lon'][1971])
    coordi = (test_df['Lat'][i], test_df['Lon'][i])
    dist[i] = distance.distance(coordi, ref).km

test_df['distance'] = dist
```

[77]:
```
# Scatter plot: sold price vs distances.

plt.scatter(test_df['distance'], test_df['Prices (tKr)'])
```

[77]:
```
<matplotlib.collections.PathCollection at 0x7fc068618790>
```



We noticed that most data points (1919 out of 1972) are clustered within 5 km radius which display some linearities (decay as we expected). There are several apartments sold in Södra Sandby, Dalby, and Veberöd. We may treat them as outliers, delete these points and focus our model prediction within the Lund city.

[113]:
```
df_city = test_df[test_df['distance']<5]
df_city.reset_index(drop=True, inplace=True)
```

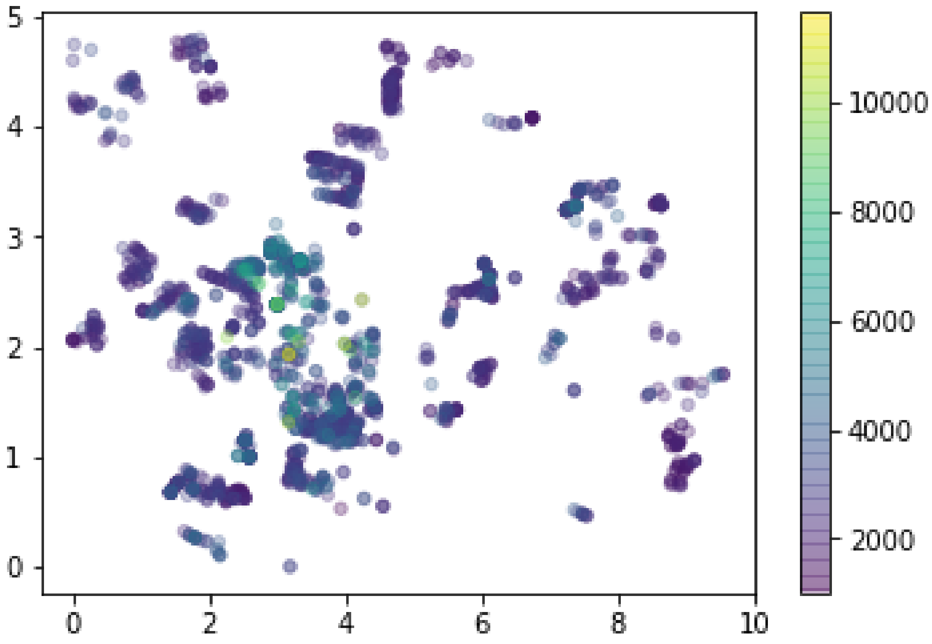
### F.5 Data transformation

[115]:
```
# Convert lat-lon to radius

radi_Lat = df_city['Lat']*math.pi/180
radi_Lon = df_city['Lon']*math.pi/180
```

[116]:
```
# Convert radius to X, Y coordinates (neglect Z as the area is small)

R = 6371

length_city = len(df_city['Lat'])

X = [0 for _ in range(length_city)]
Y = [0 for _ in range(length_city)]

for i in range(length_city):
    X[i] = R*math.cos(radi_Lat[i])*math.cos(radi_Lat[i])
    Y[i] = R*math.cos(radi_Lon[i])*math.sin(radi_Lon[i])
```

[117]:
```
# Scaling the coordinate data

df_city['X'] = Y
df_city['Y'] = X
pd.options.mode.chained_assignment = None  # default='warn'

df_city['X'] = df_city['X']-df_city['X'].min()
df_city['Y'] = -df_city['Y']+df_city['Y'].max()
```

[283]:
```
plt.scatter(df_city['X'],df_city['Y'],c=df_city['Prices (tKr)'], s=20,␣
↪alpha=0.3,
            cmap='viridis')
plt.colorbar();
```




[412]:
```
# Dropping lat-lon coordinates

df_city_clean = df_city.copy()
df_city_clean = df_city_clean.drop(columns=['Lat', 'Lon'])
```

[414]:
```
# Woops! Found 2 uncleaned data!

df_city_clean['Floor number'] = df_city_clean['Floor number'].str.
 ↪replace(',','.').astype(float)
df_city_clean['Total floor'] = df_city_clean['Total floor'].astype(float)
df_city_clean['Elevator'] = df_city_clean['Elevator'].replace('5 av 3', '0')
```

### F.6 Model pre-training

[415]:
```
# Train-test data split

cols_to_use = ['index','area (m²)', '# of rooms', 'Balcony', 'Elevator',␣
 ↪'Floor number', 'Total floor', 'Monthly Fees (Kr)', 'Build year', 'Month',␣
 ↪'distance', 'X', 'Y']
X = df_city_clean[cols_to_use]
y = df_city_clean['Prices (tKr)']
train_pre_X, test_pre_X, train_y, test_y = train_test_split(X, y,␣
 ↪test_size=0.3)
```

[416]:
```
train_pre_X.shape
```

[416]: (1343, 13)

[417]:
```
test_pre_X.shape
```

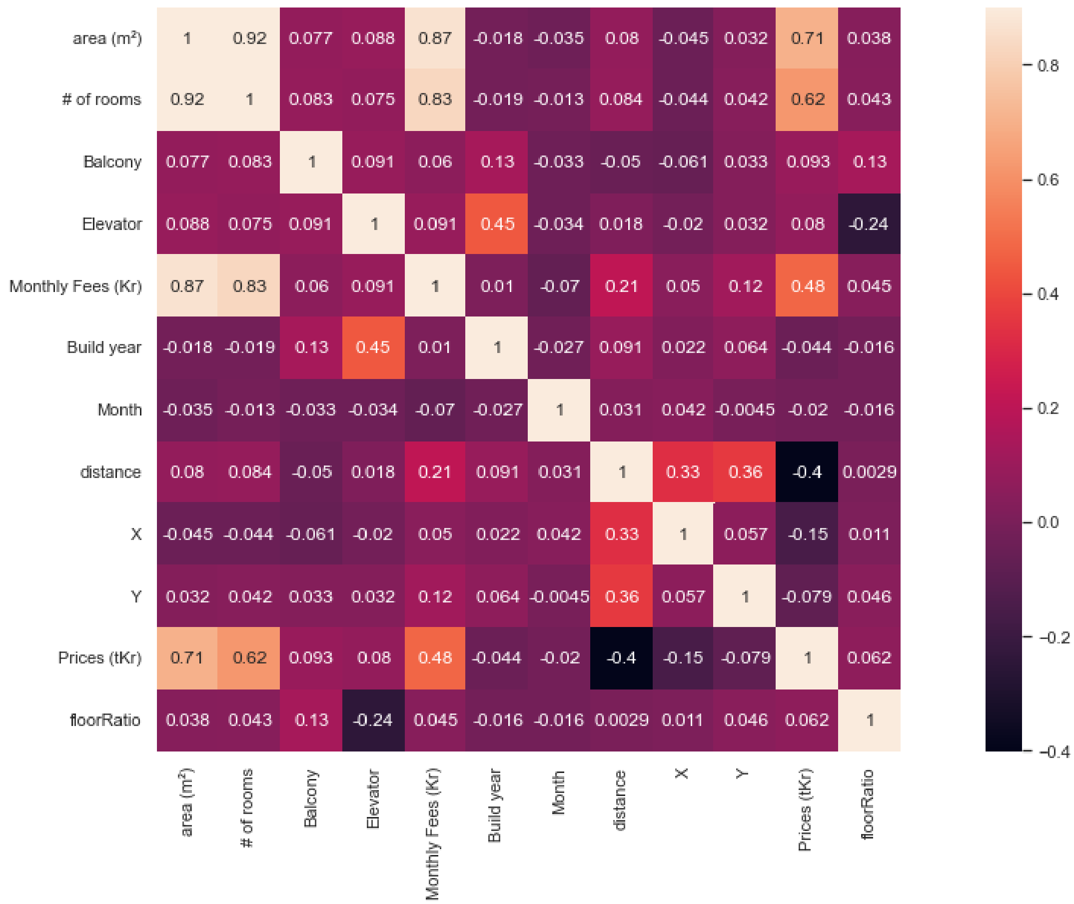
[417]: (576, 13)

[ ]:
```
# Combine train_X and train_Y into one data frame to check the correlations.

train_df = train_pre_X.copy()
train_df['Prices (tKr)'] = train_y
```

[419]:
```
# Do some more data transformation.

train_df['floorRatio']= train_df['Floor number']/train_df['Total floor']
test_pre_X['floorRatio']= test_pre_X['Floor number']/test_pre_X['Total␣
↪floor']

train_df = train_df.drop(columns=['Floor number', 'Total floor'])
test_pre_X = test_pre_X.drop(columns=['Floor number', 'Total floor'])

train_df['index'] = train_df['index'].astype(str)
test_pre_X['index'] = test_pre_X['index'].astype(str)

train_df['Balcony'] = train_df['Balcony'].astype(int)
test_pre_X['Balcony'] = test_pre_X['Balcony'].astype(int)

train_df['Elevator'] = train_df['Elevator'].astype(int)
test_pre_X['Elevator'] = test_pre_X['Elevator'].astype(int)
```

[420]:
```
# Plot heatmap to check the coorelation between different features

plt.subplots(figsize=(20,9))
sns.heatmap(train_df.corr(), cbar=True, annot=True, vmax=0.9, square=True)
```

[420]: <AxesSubplot:>

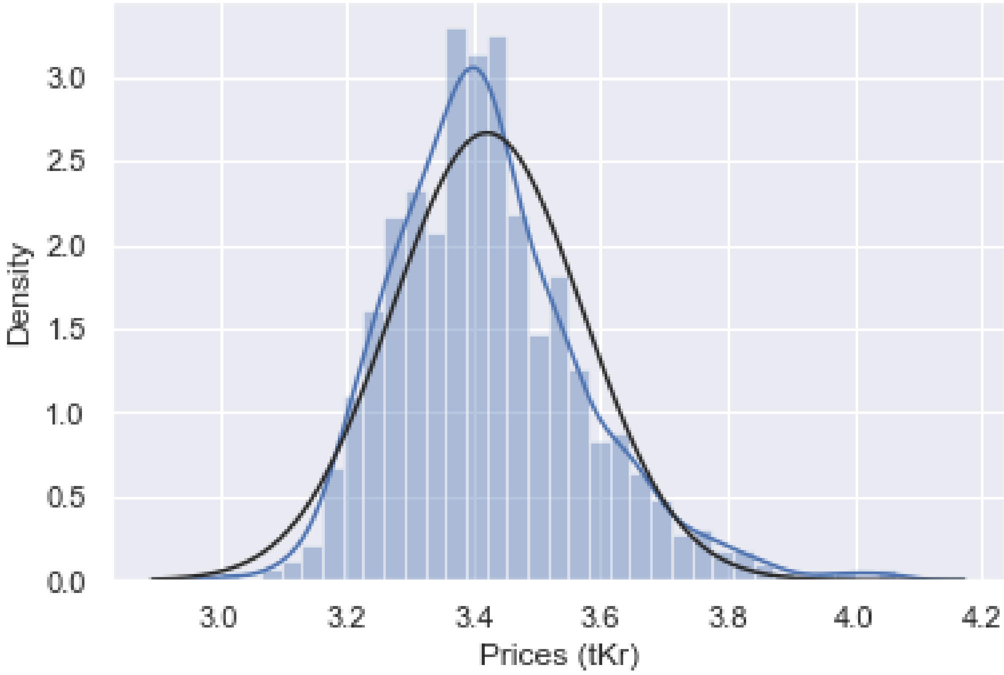



```
[421]: # Drop "# of rooms" and "Avgift" as they are highly correlated to apartment␣
       ↪size

       train_df= train_df.drop(columns=['# of rooms', 'Monthly Fees (Kr)'])
       test_pre_X= test_pre_X.drop(columns=['# of rooms', 'Monthly Fees (Kr)'])
```

```
[422]: # Transform skewed distribution using log10

       train_df['Prices (tKr)'] = np.log10(train_df['Prices (tKr)'])
       # test = np.power(train_df['Prices (tKr)'], 0.0025)
       # test = boxcox1p(train_df['Prices (tKr)'], 0.05)
       sns.distplot(train_df['Prices (tKr)'], fit=norm)
       print("Skewness: %f" % train_df['Prices (tKr)'].skew())
       print("Kurtosis: %f" % train_df['Prices (tKr)'].kurt())
```

```
/Library/Frameworks/Python.framework/Versions/3.9/lib/python3.9/site-
packages/seaborn/distributions.py:2619: FutureWarning: `distplot` is a
deprecated function and will be removed in a future version. Please adapt your
code to use either `displot` (a figure-level function with similar␣
↪flexibility)
or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

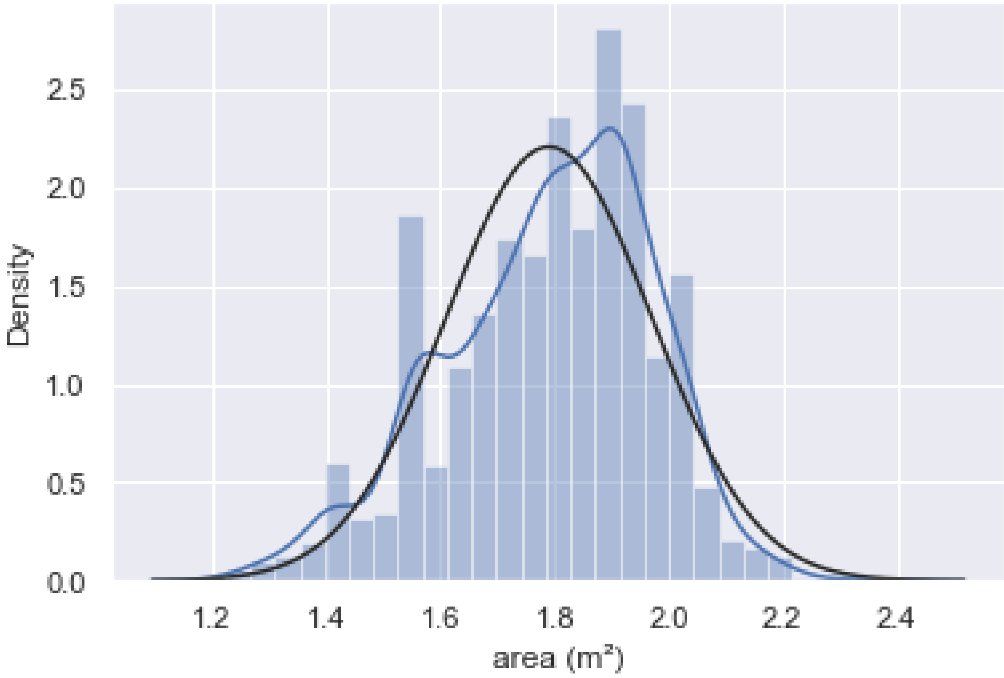
```
Skewness: 0.770440
Kurtosis: 1.195377
```




[424]:
```python
train_df['area (m²)'] = np.log10(train_df['area (m²)'])
# test = np.power(train_df['Prices (tKr)'], 0.0025)
# test = boxcox1p(train_df['Prices (tKr)'], 0.05)
sns.distplot(train_df['area (m²)'], fit=norm)
print("Skewness: %f" % train_df['area (m²)'].skew())
print("Kurtosis: %f" % train_df['area (m²)'].kurt())
```

```
/Library/Frameworks/Python.framework/Versions/3.9/lib/python3.9/site-
packages/seaborn/distributions.py:2619: FutureWarning: `distplot` is a
deprecated function and will be removed in a future version. Please adapt your
code to use either `displot` (a figure-level function with similar␣
↪flexibility)
or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

```
Skewness: -0.411472
Kurtosis: -0.120528
```




[425]:
```python
# Check the skew of all numerical features

temp_df = train_df
numeric_feats = temp_df.dtypes[temp_df.dtypes != "object"].index

skewed_feats = temp_df[numeric_feats].apply(lambda x: skew(x.dropna())).
 ↪sort_values(ascending=False)
print("\nSkew in numerical features: \n")
skewness = pd.DataFrame({'Skew' :skewed_feats})
skewness.head(20)
```

```
Skew in numerical features:
```

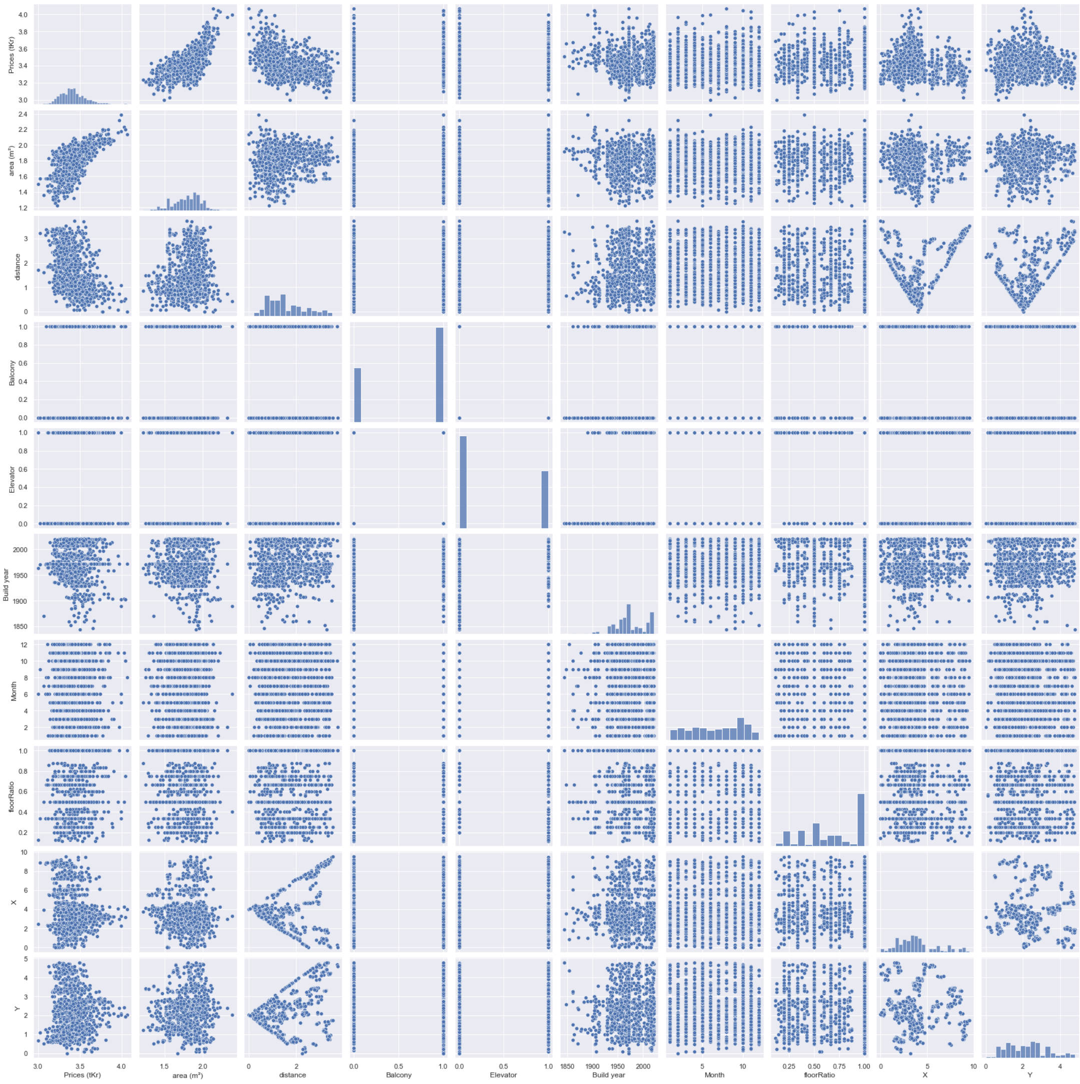
[425]:

| | Skew |
|---|---|
| X | 0.770137 |
| Prices (tKr) | 0.769580 |
| distance | 0.572442 |
| Elevator | 0.472850 |
| Y | 0.359745 |
| floorRatio | -0.178422 |
| Month | -0.190101 |
| area (m²) | -0.411013 |
| Build year | -0.482272 |
| Balcony | -0.561480 |

[ ]:
```
# We do similar transformation to the test data.

test_X = test_pre_X.drop(columns='index')
test_X['area (m²)'] = np.log10(test_X['area (m²)'])
```

[426]:
```
# Pair plot

sns.set()
cols = ['Prices (tKr)', 'area (m²)', 'distance','Balcony', 'Elevator',␣
↪'Build year', 'Month', 'floorRatio', 'X', 'Y']
sns.pairplot(train_df[cols], height=2.5)
```

[426]: <seaborn.axisgrid.PairGrid at 0x7fc07d094100>




[434]:
```
# reset train_X and train_y data.

train_X = train_df.drop(columns=['Prices (tKr)','index'])
train_y = train_df['Prices (tKr)']
```

[435]:
```
train_X.columns
```

[435]:
```
Index(['area (m²)', 'Balcony', 'Elevator', 'Build year', 'Month', 'distance',
       'X', 'Y', 'floorRatio'],
      dtype='object')
```

[ ]:
```
# From xgboost import XGBRegressor
xgb_model = XGBRegressor()
xgb_model.fit(train_X, train_y, verbose=False)
xgb_pred = xgb_model.predict(test_X)
xgb_predict = 10**xgb_pred
```

[458]:
```
# Using random forest
rf_model = RandomForestRegressor(random_state=1)
rf_model.fit(train_X, train_y)
rf_pred = rf_model.predict(test_X)
rf_predict = 10 ** rf_pred
```

[462]:
```
print("Mean Absolute Error for xgb: " + str(mean_absolute_error(xgb_predict,␣
 ↪test_y)))
print("Mean Absolute Error for rf: " + str(mean_absolute_error(rf_predict,␣
 ↪test_y)))
```

```
Mean Absolute Error for xgb: 277.014625761244
Mean Absolute Error for rf: 273.9269412287807
```

[ ]:
```
# Check gradient boosting

Gboost_model = GradientBoostingRegressor()
Gboost_model.fit(train_X, train_y)
Gboost_pred = Gboost_model.predict(test_X)
Gboost_predict = 10**Gboost_pred
```

[ ]:
```
mean_absolute_error(Gboost_predict, test_y)
```

```
286.02924602645743
```

[463]:
```
# Test the effect of two most important features: location and size

train_simple_X = train_X.drop(columns=['Balcony', 'Elevator', 'Build year',␣
 ↪'Month',
       'X', 'Y', 'floorRatio'])
test_simple_X = test_X.drop(columns=['Balcony', 'Elevator', 'Build year',␣
 ↪'Month',
       'X', 'Y', 'floorRatio'])
```

[464]:
```
train_simple_X.columns
```

[464]:
```
Index(['area (m²)', 'distance'], dtype='object')
```

[465]:
```python
xgb_simple_model = XGBRegressor()
xgb_simple_model.fit(train_simple_X, train_y, verbose=False)
xgb_simple_pred = xgb_simple_model.predict(test_simple_X)
xgb_simple_predict = 10**xgb_simple_pred

rf_simple_model = RandomForestRegressor(random_state=1)
rf_simple_model.fit(train_simple_X, train_y)
rf_simple_pred = rf_simple_model.predict(test_simple_X)
rf_simple_predict = 10 ** rf_simple_pred
```

[468]:
```python
print("Mean Absolute Error for xgb_simple: " +␣
 ↪str(mean_absolute_error(xgb_simple_predict, test_y)))
print("Mean Absolute Error for rf_simple: " +␣
 ↪str(mean_absolute_error(rf_simple_predict, test_y)))
```

```
Mean Absolute Error for xgb_simple: 301.2202042473687
Mean Absolute Error for rf_simple: 300.63847541328437
```

[469]:
```python
orig_df = pd.read_csv('apartment_df_latlon.csv')
mean_absolute_error(orig_df['Asking (tKr)'], orig_df['Prices (tKr)'])
# orig_df.head(2)
```

[469]: 184.65314401622717

[472]:
```python
Gboost_simple_model = GradientBoostingRegressor()
Gboost_simple_model.fit(train_simple_X, train_y)
Gboost_simple_pred = Gboost_simple_model.predict(test_simple_X)
Gboost_simple_predict = 10**Gboost_simple_pred
mean_absolute_error(Gboost_simple_predict, test_y)
```

[472]: 312.32285192691677

[478]:
```python
# Check the mean absolute error for the asking price in the test set.

asking_test = [0 for _ in test_y]

for ind in range(len(test_y)):
    asking_test[ind] = orig_df['Asking (tKr)'][test_y.index[ind]]

mean_absolute_error(asking_test, test_y)
```

[478]: 287.1197916666667

Apparently we could improve the model by doing the follow:

- Fine tuning the model
- More feature engineering
- Feature scalling
- dealing more outliers
- Blending prediction for different models

inherent flaw: (1) no renovation data; (2) bidding is unpredictable

[481]:
```
train_X.to_csv('train_X.csv')
train_y.to_csv('train_y.csv')
test_X.to_csv('test_X.csv')
test_y.to_csv('test_y.csv')
```

[ ]:
```
# Save model

pickle.dump(rf_model, open('hemnet_rf_pre.sav', 'wb'))
```

[ ]:
```
rf_load = pickle.load(open('hemnet_rf_pre.sav', 'rb'))
result = rf_load.predict(test_X)
# print(10**result)
```

[ ]:
```

```