UNIVERSITY OF COPENHAGEN
DEPARTMENT OF MATHEMATICAL SCIENCES

Master's Thesis

Frederik Madsen

House pricing with machine-learning

modelling the Copenhagen housing market 2016 - 2020

Supervisor: Rolf Poulsen

Submitted on: August 16th 2021

Abstract

This master thesis aims to examine, how machine learning methods can be utilized, when predicting housing prices in Copenhagen. Through crawling and scraping methods, information is parsed from websites dealing with Copenhagen housing prices, and a dataset of sales in 2016-2020 is constructed to constitute the foundation of the modelling work. This dataset is manually cleaned and preprocessed for preparation. An ordinary least squares linear model, a lasso linear model and a two-hidden-layer artificial neural network model are constructed, trained and utilized on the data, and the neural network model is found to be superior, predicting about 80 % within 15 % of the actual price on the test set. This is found to be due to its ability to incorporate non-linearity in the data. For comparison, other predictors are considered, and the models are compared to the AVM pricing model by Geomatic, and found to be superior. The thesis concludes that neural network models are a viable alternative for modelling housing prices in Copenhagen, although it shows weaknesses when predicting 2020 prices.

Contents

Chapter 1

Introduction

Who wouldn't like to be able to predict housing prices? Well, a lot of people would as a matter of fact. Apart from being one of the main topics of conversation for most adults, housing is a central part of the household, and not just because it provides a physical space for living. For those who are so fortunate to be participants in the housing market (unfortunately excluding the renting households) real estate is often the biggest asset in the portfolio of the household, making up the majority of the wealth of the household, and therefor crucial to financial opportunities. Thereby, real estate provides a cornerstone of the economy, and therefore many actors have an interest in the market of real estate, and the prices created by supply and demand. Real estate agents, banks, brokers, developers, investors, policy makers etc. all are keenly focused on where the housing market is headed, because it is a matter of so much value allocated. Here valuation of housing is essential in order to measure the worth of all these assets and to provide a quantitative measure of the benefits and liabilities accruing from the ownership.

This is no different for the city of Copenhagen, where all said interested parties could observe considerable increases in prices, and therefore growth in wealth, over the last 10 years as shown In figure (1.1). Valuation and price prediction is therefore as relevant here as in any of the many big cities experiencing this massive accruing of wealth, and this marks the outset of this thesis. A systematic approach to pricing real estate is of course always preferred, and rather than constructing elaborate macro- and microeconomic models we will here go in the direction of machine learning, which has garnered much attention in later years.
The idea here is to create models, that learn from big amounts of data, and then hopefully accurately can mimic the patterns of the market in order to predict realistic prices. It is the advances in computing power that has made this approach possible and even doable on a student laptop, where big data sets can be crunched through the algorithm to fit the model. It is therefore no surprise, that the key ingredient to success when following this approach is data, because the model will only be as good as the data it is fed. So when attempting to use machine learning tools, this thesis will put great emphasis on the data creation, to ensure a sound foundation.

In the work ahead, information will be gathered through scraping web portals that offer public information about housing sales. When pricing real estate, many aspects must be considered as would be done in a real world pricing situation. Therefore variables containing information about characteristics of the house, characteristics about the sale situation, the environmental/-societal setting and the distances to key places, all are vital for a model that can be expected to do well. And as housing prices has a great effect on the surrounding economy, so do the economic factors greatly affect pricing in the housing market. The data will therefore be supplemented by economic indicator variables, to ensure this aspect is included.

The thesis will of course have limits in scope and depth, because a topic as this most likely can and will be the subject of never ending research. Focus will only be on housing for living, and the physical area for the pricing models is limited to Copenhagen (including Frederiksberg) and the sales in question will be from the time frame 2016-2020. Naturally the study will only look at housing sales, which were free market deals, since pricing in other situations would be hard to predict. With regards to modelling, the thesis will study linear regression models and compare these to a artificial neural network model, which is a relatively new and popular approach within machine learning. Through a network, information flows forwards and backwards in order to set the weights and biases that make up the network function for pricing. This is a good example of how modern computing power becomes an important tool in the work of this thesis, and this will hopefully show promise in the quest of modelling housing prices in Copenhagen.
But as everything comes down to the data at hand, data gathering is where this thesis will have its outset, in the long process leading up to modelling.

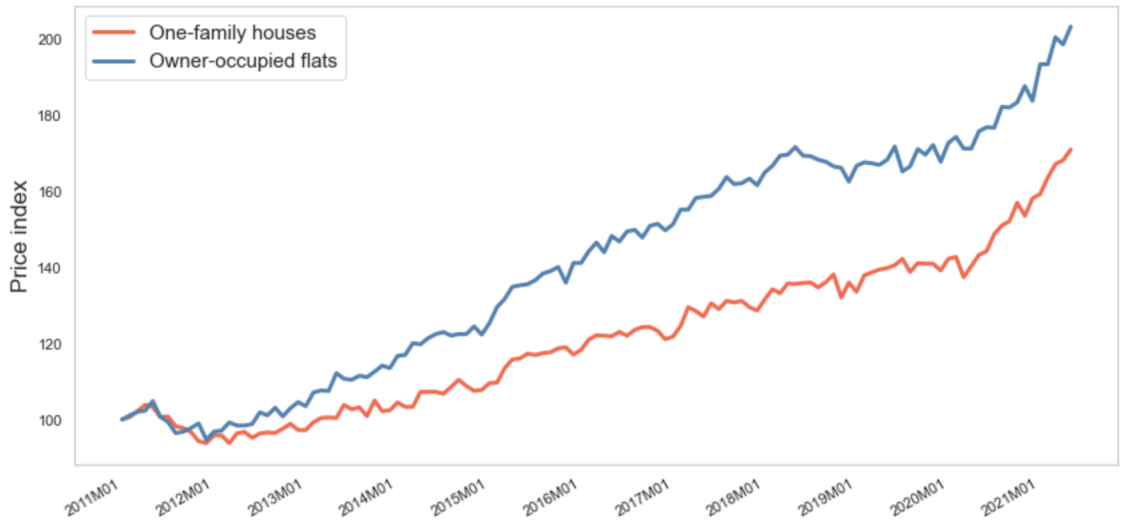

Figure 1.1: Development in Copenhagen housing prices 2011-2021 [Statistics Denmark, 2021].

Chapter 2

Data Scraping

In order to make a detailed model for the housing market in Copenhagen, having an extensive dataset with the relevant predictors is crucial. No single database suitable for this purpose is readily available, so the data basis for the model must be puzzled together by information that can be drawn from different sources on the internet. The sources of data will here be limited to four websites, which will all be dealt with in turn describing how information has been thoroughly 'scraped' to comprise a satisfactory data set.
The webpages in question are `www.boligsiden.dk`, `www.dingeo.dk`, `www.hvorlangterder.dk` and `www.statbank.dk`. The two first offer information about the address searched for, while the third provides information about distances to key places, which might be relevant for pricing. From the fourth, information about relevant economic indicators can readily be downloaded. Boligsiden is the starting point for scraping information, since it most accurately provides historical sales information, which is key to make a model of pricing prediction. Only addresses that have gone through a sale in the mentioned period of time, and are documented on Boligsiden, will be of interest when scraping the remaining websites. Furthermore, Boligsiden data will be chosen to have priority over other data with similar variables, and therefore will override in situations where the data disagrees. This choice is simply based on the fact that Boligsiden as mentioned is the basis for the scraping process, and because it turns out that Boligsiden provides the most information per address. For all three websites, there is often incomplete information, which gives challenges in comprising a complete data set. In many situations, there might be an error in the code of the website or the information might not be accessible in full for a certain address. Therefore the scraping process also in a large part comes down to choices of keeping or dropping addresses, when information is missing. In this thesis, the selection is quite strict, meaning the amount of addresses dropped is considerable, but since the amount of data was so big to begin with, this will hopefully still result in a decent size data set. Hopefully systemic exclusion of certain addresses sharing similar characteristics is avoided, but of course this is not for certain.
All scraping is done in Python on February 22^{nd} 2021, and the code (as is the case for the entire thesis) can be viewed on GitHub[1], where all functions can be inspected in their full length. Only a brief run-through will be given here of the functions, and some background helper functions will be omitted entirely.
In the code throughout, elements draw inspiration from previously publicized code [Knudsen, 2020], but only sporadically.

[1] `https://github.com/fredbuscma/Frederik_Madsen_Master_Thesis`

2.1 Boligsiden

Boligsiden is one of the biggest Danish sites dealing with real estate. It provides information about housing bought and sold, both historically and current in the market. It is run and owned by an association of real estate brokers, and provides information directly from the brokers. When a house is put on sale, the information will go directly to Boligsiden, and the site claims that the information on the site is always updated and trustworthy. It also contains updated information from Bygnings- og Boligregisteret (BBR). So it seems like a good starting point for obtaining data about housing prices in Copenhagen.
The scraping process can be divided into two parts: finding all the individual webpages of interest, and the actual scraping of every single page. The functions dealing with finding the links will be dealt with first. The main function collecting links for individual addresses is `get_all_links_boligsiden(kommune, startdate, enddate)`. The objective of this function is to get all links to pages on Boligsiden of addresses sold in a kommune (either Copenhagen or Frederiksberg) in the specified time frame (which as stated earlier is January 1^{st} 2016 to December 31^{st} 2020). The arguments are sent on to the helper function `get_url_boligsiden`, which sets up the direct link for the search. This will result in many search pages which the main `get_all_links_boligsiden` will loop through. For each search page, the function `get_all_urls_on_page_boligsiden` is built to obtain all the final links available for addresses with a sale. To do this, it uses the Selenium library, which utilises a chrome driver to crawl and parse the page, and finally collect all the wanted links. `get_all_links_boligsiden` loops over the search pages, and gathers all these sets of links.
Next up is the main function for scraping Boligsiden, which is `get_data_boligsiden(links)`. It takes the links for individual addresses sold as arguments, and then uses the helper function `get_simple_single_page_boligsiden` to do the scraping. `get_simple_single_page_boligsiden` utilises the BeautifulSoup library to set up a parser, which parses the page for each individual address sold. It turns out that the relevant information is to be found as a json string in the source code, which is gathered and stored in a data frame. `get_data_boligsiden` then concatenates all data frames, to finally hold a tentative data set for the data gathered from Boligsiden.
After deleting a considerable amount of adresses with faulty or incomplete information (which in many cases stems from the property still being for sale), there is information on 41,024 sales left for Copenhagen and Frederiksberg municipality. Another aspect which will have to be taken into consideration later when building the model, is that for each address only information on the newest sale is available in full. This means that only information on the most recent sale is included in the data, if one address has been sold multiple times in the time period. This will no doubt skew the information more towards recent years, which is something to be mindful of.

2.2 DinGeo

Now that Boligsiden has provided the sales information for all addresses that have gone through a sale in the given time period, `www.dingeo.dk` is where to find additional information that can be used for the model. The website has collected geo data from various different public agencies in areas as diverse as Radon risk and crime rates. It is financially independent of real estate brokers and banks, and has a goal of full transparency. The goal is to then scrape the website for all information relevant and available on the addresses provided by Boligsiden.
As for Boligsiden, the process of scraping the website can be divided into collecting the link for the web page for each individual address, and then gather data by scraping each individual web page. The process of gathering links is more complicated this time around, since it is now not just a matter of looping over results for a wider search over the time period. The addresses

of interest have already been found, so for each address the web page must be searched for manually. The functions dealing with this aspect will be dealt with first.
`get_geolinks1(df)` takes as its argument the data frame with information from Boligsiden, and has as its objective to put together a URL for each address. It loops through the rows in the data frame, and takes variables such as street address and zip code, and creates a string that follows the format of URLs on DinGeo for every address. There are many pitfalls here depending on the address name, since hyphens, dots and abbreviations such as "lejl.", makes the writing of a functional link cumbersome. The function then returns the data frame with the links included.
`add_dingeo(df)` is the main function for scraping DinGeo. It takes as argument the data frame with the web page links included, and it loops through these links and calls the helper function `dingeo_page` to scrape each page. This is a very computationally heavy and time consuming process, which is why multithreading is used for this part. What `dingeo_page` then does, is that it uses the Requests library and the BeautifulSoup library to create a parser, that parses the page. A dictionary is created to store all the information parsed from the page, and this is finally returned as a data frame. With `add_dingeo`, this new dataframe is then added to the respective address row, making a join over the URL for the web page.
DinGeo turned out to be a more complicated site to work with, since the the information for the addresses was in many instances incomplete. This will be dealt with later in the section for data cleaning, but there were many instances of faulty pages, where errors must have occurred on the site. Even though the problem of formatting address names in the URL was anticipated, still many had no page to be found under the correct name, and some did not exist on DinGeo entirely. Because of these problems, a significant loss of addresses reduced the dataset from 41024 observations to 40657 observations. This is still a very big data set, and therefore not deemed problematic for the model making.

2.3 Hvorlangterder.dk

As the name of the site might suggest, `www.hvorlangterder.dk` is the source of the distance variables which will be included in the model. It is a very simple site with a single search bar, which from a given address returns distances to various places of interest, whether this be the nearest doctor or S-train. This is of course of high interest, when one considers the price of housing. The site is a part of Viamap, a private company under Mølbak Landinspektører A/S creating solutions for geomapping.[2]
`add_hvorlangterder(df)` is the main function for scraping the site, which takes the data frame already constructed with the Boligsiden and DinGeo information and then adds the distance variables for each address. It loops over the rows in the data frame and extracts the addresses. `get_hvorlangterder` is then the helper function that actually does the scraping part. A variable called "Location" is made in the original data frame to put the addresses in a friendly format for the scraping. `get_hvorlangterder` takes this variable as its argument, and creates the URL which would amount to a search on the site. The Requests library is then used to get a response from the site, and the information can then readily be read out as a json file. The information is stored in the returned data frame, and this can then be merged with the existing data set by joining over the variable "Location".
Even though the process of scraping this site also involves creating a URL from the addresses, potentially creating problems from discrepancies in address names, this does not seem to be a source of problems leading to loss of many observations. Rather the loss of observations in this instance comes from errors on the page, for certain addresses where there simply is no information available. It lowers the observation count from 40,657 down to 40,606, which is a pretty modest loss.

[2] `https://www.viamap.net/kontakt/`

2.4 StatBank Denmark

Additional information is needed about the economic factors, which could reasonably be assumed to influence the housing prices in Copenhagen. For this, StatBank Denmark by Statistics Denmark provides time series for many economic indicators, that could be of interest. These data are readily downloaded as .csv files, which means the process of scraping is not needed for this section.
First up is the data set "EJ14", which provides price indexes for one-family houses and owner occupied flats, on monthly basis. It is reasonable to believe, that the general market pricing level will be an important indicator in the pricing process. From these two series a single variable is created, `priceIndex`, where the value is taken from the indexes, where the housing type, `address.itemTypeName`, matches. In stead of joining the new information to the data by month of the sale, the previous month is used. This means that for every address, `priceIndex` will actually be the index from the month before. This choice has been made, so that the index doesn't convey information about the time after the sale has actually been made. In stead the economic indicator as it was immediately before the sale is of interest, since this is the available knowledge in the pricing process. Finally, the indexes are reset, so that the start date of the scraping, January 2016, is set to 100.
Next up is the data set "MPK13", which provides data for the OMXC 20 Cap share index on monthly basis. How the local stock market is doing is often a clear reflection of the economic cycles. And since housing prices are very dependent on the swings of the economy, this is a relevant indicator. `OMXC20` is derived from the dataset and added to the scraped data, but again with the information from the previous month, for the reasons stated earlier.
From the set "DNRNURI", one can draw information about new domestic mortgage loans. Here the variable `mortgageRate` is made, which denotes the annualised agreed rate on a monthly basis. It seems reasonable to include mortgage rates as an economic indicator, since these set the cost of lending when one wants to buy a house. Again, the value is taken from the previous month, when inserted into the scraped data.
Finally, "AUS08" has been included, since it has information about unemployment. The unemployment is, as the stock market, an economic indicator of how the economy is doing, and can therefore be assumed to be a relevant predictor for housing prices. Here seasonally adjusted unemployed in percent of the labour force on a monthly basis is taken to create the variable `unemploymentRate`, again with a one month lag.

Chapter 3

Data Cleaning

To begin with, the raw data set comprises 40,606 observations with 289 variables gathered from the data scraping step. On closer inspection, it becomes clear, that these data are incomplete to say the least, and in need of further treatment before a model is anywhere near possible. All code is again to be found on the GitHub page.[1]

3.1 Rough data cleaning

A very large amount of information was gathered in the data scraping process, and to begin with the set is in need of a rough cleaning, since much of the information is irrelevant for the purpose of making the house pricing model. Upon inspection, many variables have many (in some cases all) values missing, and many of these variables are dropped right away with the function `remove_variables`. It is variables such as `bbrData` and `conservation.code` that are dropped, since most values are missing. In the first crude treatment by `remove_variables`, also variables which have unnecessary information like `imageLink100X80` and `nextOpenHouse` are dropped. Most of the variables dropped contain information from Boligsiden, which still remains the most represented web site in the data even after the first drop.
A second drop performed by the function `second_drop` further deletes some variables with a big amount of values missing, since other more complete variables contain the same information. These two steps bring the variable count down to 133.
Table (3.1) for the variable `address.latestSale.saleType` shows the division of sales by type, and only 35929 sales were "Fri handel", meaning sold in a free market setting. These are the only observations of interest for the model, since the model of this thesis is limited to free market pricing. This is of course because sales that are for example family deals can be very far off the actual value in a free market setting and would give a distorted picture in the model. Therefore these observation are deleted along with the now obsolete variable (all are free trade sales), bringing the data set down to 35929 rows and 132 variables.

address.latestSale.saleType	count
Fri handel	35929
Familiehandel	4255
I øvrigt	358
Auktion	39

Table 3.1: Observation count by address.latestSale.saleType category.

[1] `https://github.com/fredbuscma/Frederik_Madsen_Master_Thesis`

3.2 Fine data cleaning

The finer data cleaning, where all variables and numerous specific observations are dealt with in turn, is a lengthy process, and is therefore not included here in full. Rather, the walk-through will stick to focus points within some variable categories, exemplifying considerations that are general throughout the cleaning process. The order in which the data is handled is still the same as in the original code.
Throughout this part of the data process, it is a recurring theme, how information has to be gathered and compared from the different sources. In many cases, variables originating from different sites overlap in their subject. In some instances this is beneficial, since missing data for one variable can be gathered from another. In others, a problem might be that variables are in direct disagreement, and some rule must be specified for this situation. In this case of data duplicates, the simple rule that will be followed is that Boligsiden data has priority, since it is the basis of the information gathering, and the Boligsiden data makes up the bulk of the dataset. So to sum up, if DinGeo or hvorlangterder.dk bring duplicates of Boligsiden data to the table, then these additions will only act as supplements for the cases where the Boligsiden data is incomplete.

3.2.1 Sale date variables

The different time variables included in the data set primarily are regarding the sale or the address, which means that it is information gathered from Boligsiden. One thing to make sure here, is that the variable telling how long an address was on the market before it sold, `SalesPeriod`, is included for all observations, since this is deemed very relevant for the price. For instances where the value of this variable is missing, on way to get around the problem is to create the new variables `AddedRemoved` and `AnnouncedRemoved` which gives the time difference between `dateRemoved` and `dateAdded`, `dateRemoved` and `dateAnnounced`. Then the function `combine_first` is used when overwriting the original variable `SalesPeriod` - this prioritizes the original `SalesPeriod`, but if the value is missing it is filled by either `AddedRemoved` or `AnnouncedRemoved`, in that order of priority. Unfortunately, this still leads to missing `SalesPeriod` for 3381 observations, so these are deleted since this variable is deemed important. This results in a data set with 32548 observations.

3.2.2 Valuation variables

For valuation variables, `propertyValuation` and `valuationDate` are found to be relevant for the model. But again it turns out, that information is incomplete, in this case for 307 observation. Again the information is deemed important for the model, so these observations are deleted, resulting in the observation count going down to 32241 addresses.

3.2.3 Type and conservation variables

In the data set there are two variables (both from Boligsiden) coding for housing type. In table (3.2) these two variables are tabulated against each other, and give an overall impression that the two variables are in agreement, which is fortunate. There is a little discrepancy between whether something is a ”Rækkehus” or ”Villa”, but this is not deemed too important. None of the two variables have data missing, and so `address.itemTypeName` is chosen for simplicity, since it has fewer categories. Furthermore, categories as ”Andelsbolig” should not be included, since limitations are set to only deal with purely market based sales and in this regard cooperative pricing is a bit different. Without loss of much information, the data set is limited to only include observations with categories ”Ejerlejlighed”, ”Rækkehus” and ”Villa”. This only means that 14 rows are deleted, resulting in a data set of 32227 observation.

address.itemTypeName address.latestForSale.itemTypeName	Andelsbolig	Ejerlejlighed	Helårsgrund	Rækkehus	Rækkehus / ejerlejlighed	Villa	Villa / helårsgrund
Andelsbolig	1	2	0	1	0	0	0
Ejerlejlighed	3	27926	0	2	0	0	0
Ejerlejlighed / fritidsbolig	0	4	0	0	0	0	0
Ejerlejlighed / rækkehus	0	13	0	0	1	0	0
Ejerlejlighed / villa	0	4	0	0	0	0	0
Ejerlejlighed / villalejlighed	0	20	0	0	0	0	0
Helårsgrund	0	0	2	0	0	4	0
Helårsgrund / villa	0	0	0	0	0	1	0
Rækkehus	1	284	0	949	0	1	0
Rækkehus / ejerlejlighed	0	7	0	2	2	0	0
Rækkehus / villa	0	3	0	4	0	0	0
Villa	0	40	0	400	0	2453	0
Villa / ejerlejlighed	0	1	0	7	0	0	0
Villa / helårsgrund	0	0	0	0	0	17	1
Villa / rækkehus	0	0	0	4	0	0	0
Villa / villalejlighed	0	0	0	1	0	0	0
Villalejlighed	0	65	0	6	0	5	0
Villalejlighed / ejerlejlighed	0	2	0	0	0	0	0
Villalejlighed / villa	0	2	0	0	0	0	0

Table 3.2: Variables of housing type cross tabulated.

Another kind of variable with a similar content as housing type are variables that has information about the usage of the addresses. In table (3.3) the two variables `unitUsage.content` and `Usage` are tabulated against each other, and these variables too seem to be very much in agreement about the usage content. `Usage` is from DinGeo while `unitUsage.content` is from Boligsiden, and exemplifies well how much in agreement the information from the two websites in general was found to be. In this case, `unitUsage.content` has 973 observations missing, while `Usage` is only missing 4. As before, data from Boligsiden, in this case `unitUsage.content`, is still prioritized, but the missing values are filled in with data from `Usage`. Deleting 4 results in the data set now being 32223 rows long.

In both the case of Boligsiden and DinGeo, the scraping process yielded essentially no information about conservation of the addresses. For the case of Boligsiden, this information simply seems to not be provided, and DinGeo had a technical error on its site at the moment of scraping, meaning that all address searches gave no information on conservation.
Another way to get some kind of information of the kind of building at hand, is to look at the age and the year it was rebuild, and in this case, both sites were helpful in providing data. Only in 357 cases did they disagree on the year built, and again the Boligsiden data is prioritized.
Both sites also provide information of the rebuilding year, but with a majority of cases with observations missing. This is most likely due to that many buildings haven't been rebuild, and in these cases the year built is put as a standin for rebuiltyear, to make the variable complete.

Usage unitUsage.content	0	Anden enhed til helårsbeboelse	Bolig i etageejendom, flerfamiliehus eller to-familiehus	Dobbelthus	Fritliggende enfamiliehus	Kollegiebolig	Række-, kæde- eller dobbelthus	Række-, kæde- og kyngehus
Anden enhed til fritidsformål	0	0	0	0	0	0	1	0
Anden enhed til helårsbeboelse	0	6	0	0	0	0	0	0
Bolig i etageejendom (flerfamiliehus, herunder 2-familiehus)	1	0	27097	0	0	0	0	0
Dobbelthus	4	0	0	328	0	0	2	0
Fritliggende enfamilieshus (parcelhus)	0	0	0	1	2475	0	0	0
Kollegium	0	0	0	0	0	70	0	0
Række- og kædehus	6	0	0	0	0	0	0	1087
Række-, kæde- eller dobbelthus	0	0	0	0	0	0	172	0

Table 3.3: Variables of usage type cross tabulated.

3.2.4 Area variables

The variables containing information about the size of the sold real estate is arguably one of the most important aspects, when a price is determined. Here it is relevant to include both the housing area but also the weighted area which sets a wider frame for the accessible area for the address. Therefore this would include accessible addict or basement space and common areas, just to name some examples.
All aspects of housing area have similar variables from Boligsiden and DinGeo, and in all cases Boligsiden is given priority, using the `combine_first` function to let DinGeo data override missing values in the Boligsiden data. This is also the case for `areaWeighted`, but this variable is deemed extra important, so in cases of missing data from both sites, the missing values are filled in by the variable `unitAreaTotal`. This still leaves 135 addresses where `areaWeighted` is missing, and these are dropped giving a data set of 32092 observations.

3.2.5 Environmental/societal variables

What kind of effect the surroundings have on the address is of course of key importance when it comes to how the market prices it. This includes everything from burglary risk of the area to energy mark of the house, telling how much one would have to pay to heat the place up. DinGeo provides more general information on this front, and therefore DinGeo variables coding for radon risk, flooding risk, and area noise are kept. From Boligsiden, `energyMark` is kept, since it includes more different categories than the DinGeo equivalent. This variable is deemed important, so 1385 observations are deleted since data is missing for `energyMark`. This results in a data set of 30707 observations.

3.2.6 Coordinate variables

It is of course obvious that housing placement is key for pricing. The variables that hold addresses, more specifically zip codes, might be more of interest, when considering the surrounding area in the model. But coordinate variables turn out to be helpful, during the handling of data, which will become apparent. `Latitude` and `Longitude` are included from Boligsiden, but this still leaves 91 addressed with missing data. This is handled through manually inserting these from searches on Google Maps[2], and it turns out that the majority of these problematic addresses had missing coordinates because the housing was newly built.

3.2.7 Voting variables

For voting variables, DinGeo provides information about the local election area for each individual address. This is based on the 2019 general election results, and spans over information about election area, voter turnout and biggest party. This again leads to a situation of missing values, which can be divided between postal codes as done in table (3.4).

address.postalId	count
2100	696
2300	405
2200	392
2150	205
2400	61
2770	9

Table 3.4: Missing voting information by zip code.

Now the geo coordinates become of use, since using "hamstermap"[3] the addresses can be plotted, as is done for the zip code "2100" in figure (3.1). By comparing with the partition for election areas in figure (3.2), which was taken from "Københavnerkortet"[4], it is clear that all of the problematic addresses for "2100" lie within the same election area. It turns out that for every zip code, the addresses with missing data are within a single election area zone as with the problematic addresses in "2100". And by looking up these election areas at the election site provided by KMD[5], one can for each area find the voter turnout and biggest party, which are then inserted manually into the data set. An overview of the resulting inserted values can be seen in table (3.5).

address.postalId	Afstemningsområde	Valgdeltagelse	Største_parti	count
2100	1. Øst	88.7	venstre	696
2300	2. Øst	86.0	socialdemokratiet	405
2200	5. Nørrebrohallen	83.4	enhedslisten	392
2150	1. Øst	88.7	venstre	205
2400	6. Vest	75.6	enhedslisten	61
2770	2. Øst	86.0	socialdemokratiet	9

Table 3.5: Voting information manually inserted for missing values.

[2] `https://www.google.com/maps/`
[3] `http://www.hamstermap.com/`
[4] `https://kbhkort.kk.dk/spatialmap`
[5] `https://www.kmdvalg.dk/fv/2019/KMDValgFV.html`

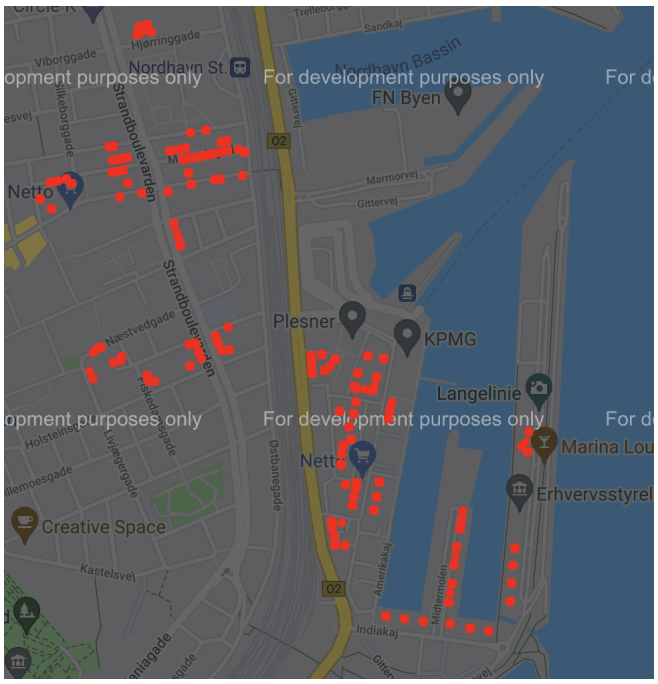

Figure 3.1: Addresses with missing voting data for zip code 2100 plotted in Hamstermap.

Figure 3.2: Election area 1. Øst from "Københavnerkortet".

3.2.8 Imputing variables

Missing variables are still an issue for the data set, and one way to patch up the holes is through imputing. This is done for several different variables, as can be seen in the code, but here it will only be exemplified by a few cases. It turns out, that there is 717 adresses with missing

bbr-data from Boligsiden. This is information such as the variables `propertyType.content` and `toilet.content` among others. For these 717 addresses we impute by inserting the most common value for `propertyType.content` for addresses with the same `address.postalId` and `numberOfRooms`. Likewise for these addresses `toilet.content` is imputed with the most common values among the addresses with the same `address.postalId` and `numberOfToilets`. And this approach is continued for other variables such as heating and wall materials, where it feels natural to assume that addresses with similar characteristics in the same area would share these variable values as well.
This still leaves 53 addresses with missing bbr-data, and these are then deleted, resulting in a dataset of 30,654 entries.

3.2.9 Cleaned data

Finally, 108 more observations were deleted, since their sales date turned out to be in 2021, leaving 30,546 observations and 69 variables. And this concludes the data cleaning procedure. All variables are at this point renamed and ordered, so that they can be displayed in tables (3.6) through (3.10), which carry a description for each. The lower case letter at the end of every variable name, indicates which website has provided the information. Boligsiden (b), DinGeo (d), mix of Boligsiden and DinGeo (bd), Hvorlangterder (h) and finally Statbank (s).

Variable name	**Description**	**Type**	**Unit**
	Boligsiden		
`address_b`	Address	Categorical	
`street_b`	Street name	Categorical	
`streetName_b`	Street name	Categorical	
`postalId_b`	Zip code	Categorical	
`city_b`	City part	Categorical	
`valuationDate_b`	Valuation Date	Categorical	
`propertyValuation_b`	Property valuation	Numeric	DKK
`salePrice_b`	Sale price	Numeric	DKK
`paymentCash_b`	Asking price	Numeric	DKK
`salesYear_b`	Sale year	Numeric	
`saleDate_b`	Sale date	Categorical	
`itemTypeName_b`	Housing type	Categorical	
`propertyType_b`	Property type	Categorical	
`buildYear_b`	Year built	Numeric	
`areaResidential_b`	Residential area	Numeric	m^2
`numberOfFloors_b`	Nr. of floors	Numeric	
`floor_b`	Floor	Numeric	
`toilet.content_b`	Toilet type	Categorical	
`bath.content_b`	Bath type	Categorical	
`numberOfBathrooms_b`	Nr. of bathrooms	Numeric	
`energyMark_b`	Energy mark	Categorical	
`SalesPeriod_b`	Sales period	Numeric	days
`latitude_b`	Latitude	Numeric	degrees
`longitude_b`	Longitude	Numeric	degrees

Table 3.6: Variables from Boligsiden.

Variable name	Description	Type	Unit
	Boligsiden and DinGeo		
`unitUsage_bd`	Usage type	Categorical	
`rebuildYear_bd`	Year rebuilt	Numeric	
`area_bd`	Area	Numeric	m^2
`areaBasement_bd`	Basement area	Numeric	m^2
`areaWeighted_bd`	Weighted area	Numeric	m^2
`areaTotal_bd`	Total area	Numeric	m^2
`numberOfRooms_bd`	Nr. of rooms	Numeric	
`numberOfToilets_bd`	Nr. of toilets	Numeric	

Table 3.7: Variables from Boligsiden and DinGeo combined.

Variable name	Description	Type	Unit
	DinGeo		
`AVM_pris_d`	AVM price	Numeric	DKK
`kitchen.content_d`	Kitchen type	Categorical	
`outerwall_d`	Outerwall material	Categorical	
`roof_d`	Roof type	Categorical	
`heating_d`	Heating type	Categorical	
`radonRisk_d`	Radon risk	Categorical	
`noise_d`	Noise	Categorical	
`floodingRisk_d`	Flooding risk	Categorical	
`aboveSea_d`	Above sea level	Numeric	m
`breakinRisk_d`	Break in risk	Categorical	
`biggestParty_d`	Biggest party	Categorical	
`turnoutVote_d`	Voter turnout	Numeric	%
`electionArea_d`	Voting area	Categorical	

Table 3.8: Variables from DinGeo.

Variable name	Description	Type	Unit
	hvorlangterder.dk		
school_h	Dist. to school	Numeric	m
roadtrain_h	Dist. to roadtrain	Numeric	m
junction_h	Dist. to junction	Numeric	m
daycare_h	Dist. to daycare	Numeric	m
metro_h	Dist. to metro	Numeric	m
doctor_h	Dist. to doctor	Numeric	m
soccerfield_h	Dist. to soccerfield	Numeric	m
hospital_h	Dist. to hospital	Numeric	m
busstop_h	Dist. to bus stop	Numeric	m
lake_h	Dist. to lake	Numeric	m
supermarket_h	Dist. to supermarket	Numeric	m
pharmacy_h	Dist. to pharmacy	Numeric	m
strain_h	Dist. to S-train	Numeric	m
airport_h	Dist. to airport	Numeric	m
train_h	Dist. to train	Numeric	m
library_h	Dist. to library	Numeric	m
publicbath_h	Dist. to public bath	Numeric	m
coast_h	Dist. to coast	Numeric	m
sportshall_h	Dist. to sportshall	Numeric	m
forest_h	Dist. to forest	Numeric	m

Table 3.9: Variables from hvorlangterder.dk.

Variable name	Description	Type	Unit
	Statbank		
priceIndex_s	Housing price index	Numeric	
OMXC20_s	Stock index	Numeric	
mortgageRate_s	Mortgage rate	Numeric	%
unemploymentRate_s	Unemployment rate	Numeric	%

Table 3.10: Variables from Statbank.

Chapter 4

Exploratory data analysis

From the now roughly cleaned data set of Copenhagen housing sales, comprising 30,546 observations and 69 variables, one can now look into the actual numbers, in order to decide, whether the data is ready to be the basis of the model. One should look into the structures of interest to appear in the data, and most importantly look for problematic patterns, which could constitute a problem for the model. Here outliers are of central importance, where few observations can skew the model in an obviously wrong direction. Initially, focus is on the target variable, sales price, with additional central developments and patterns of interest for the model.

4.1 Housing sales key takeaways

Of primary interest is of course the target variable `salePrice_b`, and how this variable behaves in the data. Below in figure (4.1), the distribution of the prices is illustrated in a histogram, where a kernel density estimate has been drawn on top. It shows how the the prices are very right skewed (a positive skew of 2.18), and very heavy tailed (a positive kurtosis of 9.41). This means that even with a median DKK 3,235,000 price, the mean sales price was DKK 3,705,967.

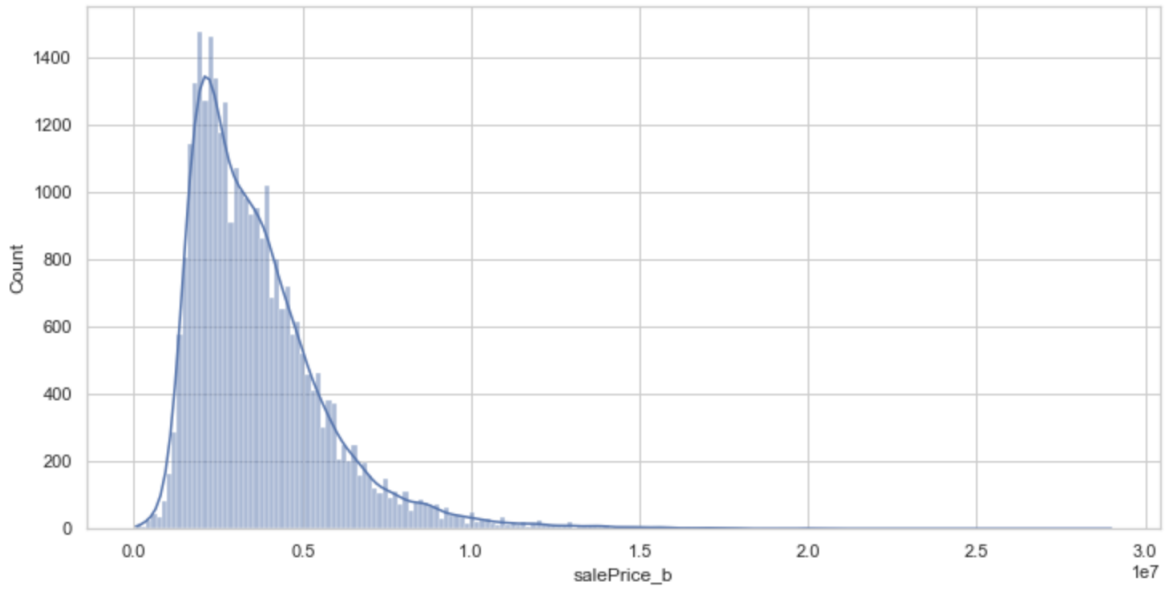

Figure 4.1: Distribution of sales price

So it is apparent that pricewise there are some serious outliers, with very high sales prices. This is not in itself a problem, since it reflects the true Copenhagen housing market, that some real estate is very expensive compared to the average, and one would of course want the model

to take this into account. The boxplot in figure (4.2) shows these outliers even more clearly, and shortly another approach will be used to determine which of these actually are problematic outliers.

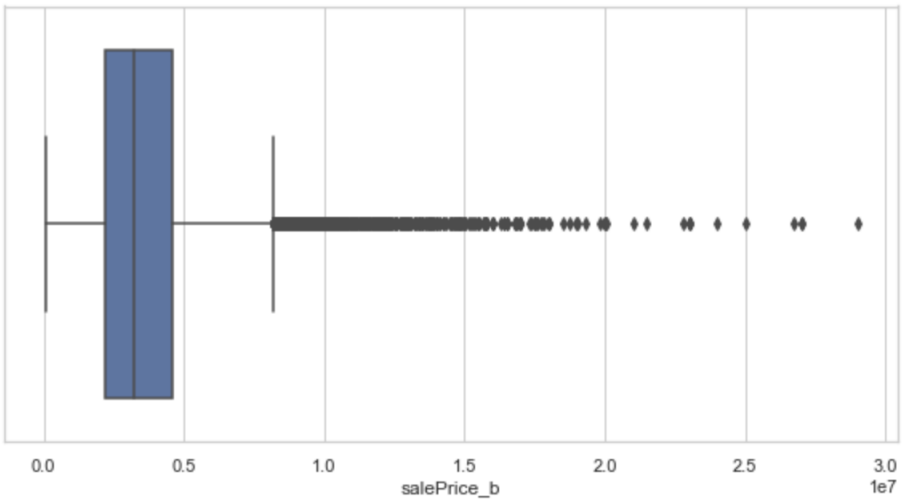

Figure 4.2: Boxplot of sales price

How the sales prices of Copenhagen housing have actually developed over the period of time can be found from the Statbank price index, which has been mentioned earlier. Figure (4.3) of the price index development for one-family houses and for owner-occupied flats in the capitol region of Denmark show a sure trend over the period 2016-2020: Housing prices have been rising markedly, with an increase of about 30% in the period.

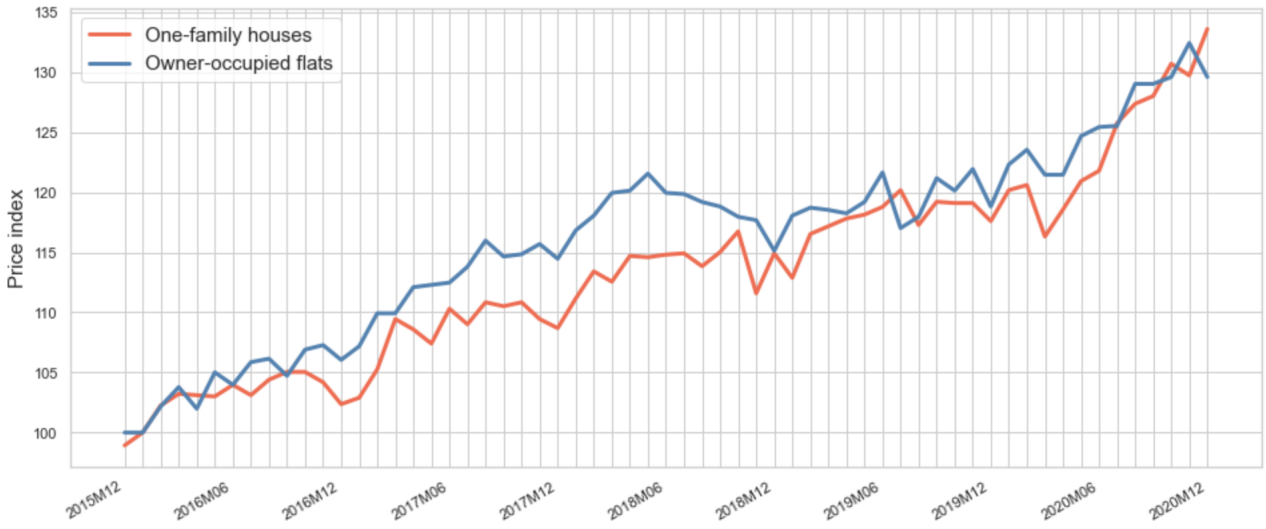

Figure 4.3: Price development in Capitol region of Copenhagen 2016-2020

The rise in housing prices in Copenhagen is clear in the figure (4.3), but this information comes from one of the additional economic indicators included in the model. One way to show, that this price increase over time is present in the sales data as well, is to introduce the variable `squareMetrePrice_bd`. This includes both the target `salePrice_b` and `areaWeighted_bd`, and therefore gives a more comparable price measure across housing of different sizes. The weighted area is chosen, since it presumably gives a more all around view of the space accessible for the address. The 30% price increase is visible in figure (4.4) of the average square meter price over

time, and thereby confirms the development in the price indices.

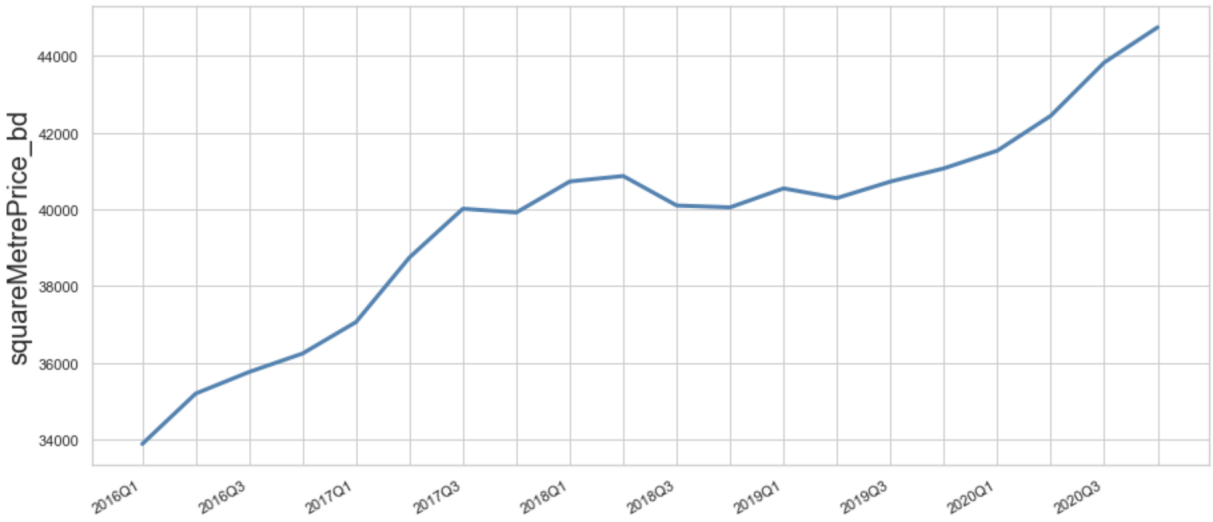

Figure 4.4: Square meter price development in 2016-2020

This is just one example of how development over time is a clear influence on the sales price. Therefore the model should reflect this, and therefore include this aspect in variables of time. Another example of how time variables will be of great importance in the model, is the cyclicality of the sales. The activity in the market will naturally affect pricing, and therefore time variables affecting sales activity is of interest. Figure (4.5) and (4.6) clearly show how the sales activity fluctuates over time, and the pattern can be captured on quarterly basis. Q4 and Q1 constitutes low sales, while Q2 and Q3 seem to give more activity. Therefore a new variable, `quarter_b`, is included in the data set, so that a model will be able to to capture these quarterly cycles.
Another takeaway from the figures is how sales are increasing over the period. Whether or not this is actually caused by a rise in activity in the period is uncertain, and external data would be needed to determine whether this is the case. A probable reason for this is the fact that the scraping process for every address only included the latest sale, and therefore every address is only included once. So as mentioned earlier, we are more likely to see sales in the latter years of the period, which is most likely why this increase can be observed in the figures.

When assessing the data set, it naturally becomes clear, that prices vary greatly from area to area in Copenhagen. To visualize this, figure (4.7) again includes square meter prices, and shows the distribution within each city area included in the data. It should be noted that many outliers have been left out from the picture in order to give a clearer view of the distributions. So this should give a clear picture, that one can expect prices to be higher in certain areas relative to other, even though this plot does not take into consideration what sizes of houses are prevalent in the different areas. But the figure serves its purpose by clearly showing, that variables determining city area will be important for pricing and should therefore be included in the model.

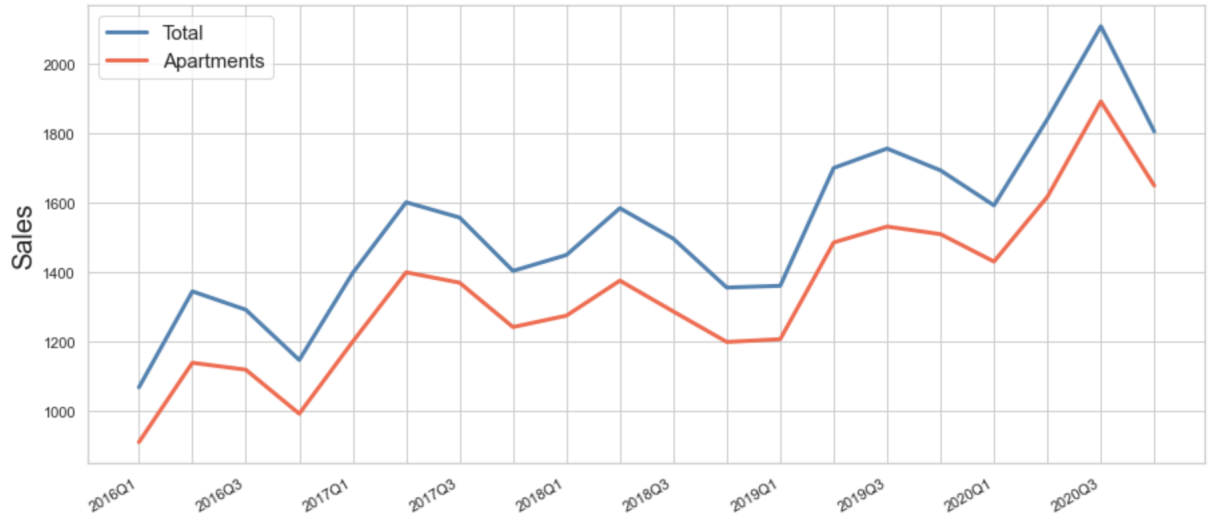

Figure 4.5: Development in total and apartment sales, 2016-2020

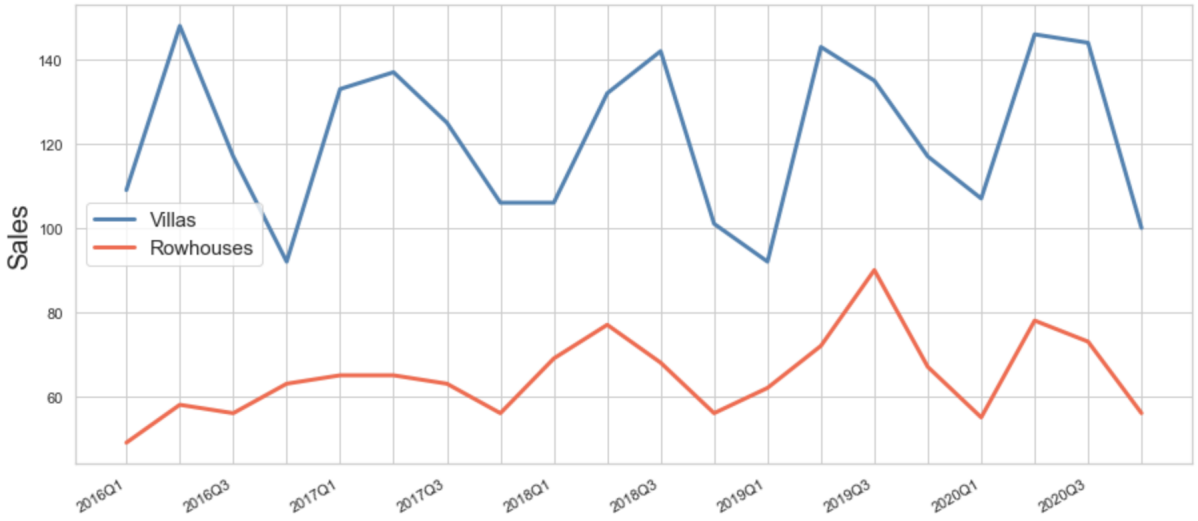

Figure 4.6: Development in row house and villa sales, 2016-2020

4.2 Outliers

Outliers in general can be problematic for the model fit, since they have the potential to skew the resulting model away from the true relationship. They can arise from a variety of reasons including mistakes during data collection or recording. In the previous section, figures (4.1) and (4.2) gave the clear impression, that the distribution of housing prices in the data set had a heavy tail with many outliers. Whether or not these were problematic outliers to be sorted out before fitting a model was less clear. Some addresses in reality are just excessively more expensive, and this should be reflected in the model. In order to get a more qualified view of the outliers, one cas use the newly created variable `squareMetrePrice_bd`, which naturally includes more information than just price. When looking at the distribution of square meter price in figure (4.8), there are still many outliers to take into consideration.

It is clear that especially one observations has an excessively high square meter price, which does not seem probable, and doesn't represent the market to be analysed. Using the square meter price, it is easier to determine, what seems realistic in Copenhagen, and thereby which outliers that should not make the cut to enter the model. The average of the square meter prices

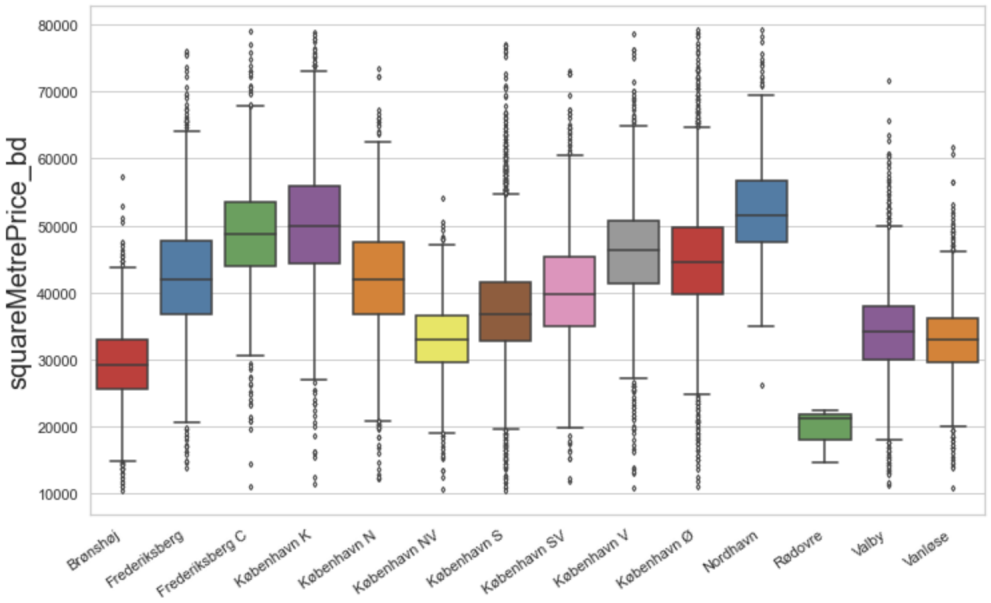

Figure 4.7: Square meter price distributions across city areas.

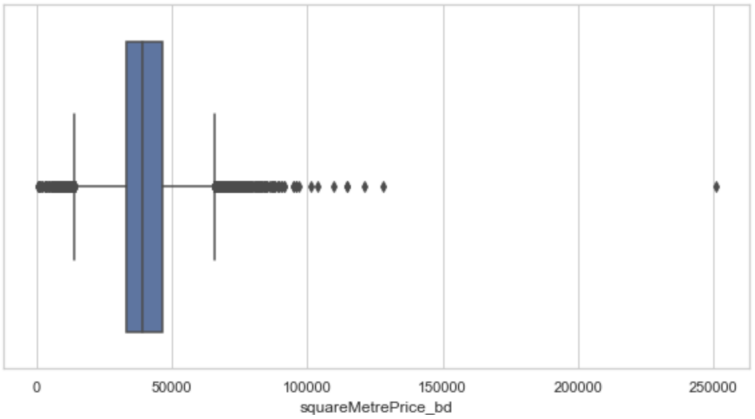

Figure 4.8: Square meter price distribution.

in the data set is DKK 40,055. Looking at the boxplot in figure (4.7), where the observations were split by cities, it was relatively clear, that most observations lie within DKK $\pm$ 20,000 of this average. Here, the rule is chosen for separating outliers, that square meter price may not be more than twice as big or 4 times as small as the mean. This seems to be quite conservative in terms of eliminating addresses, and it brings the observation count down to 30,414. In the scatterplot in figure (4.9), plotting price against weighted area, this rule of elimination becomes two straight lines, that separate the outliers. The red dots represent the addresses which have been excluded. It can also be seen in the boxplot in figure (4.10) that all the serious outliers have been rooted out.

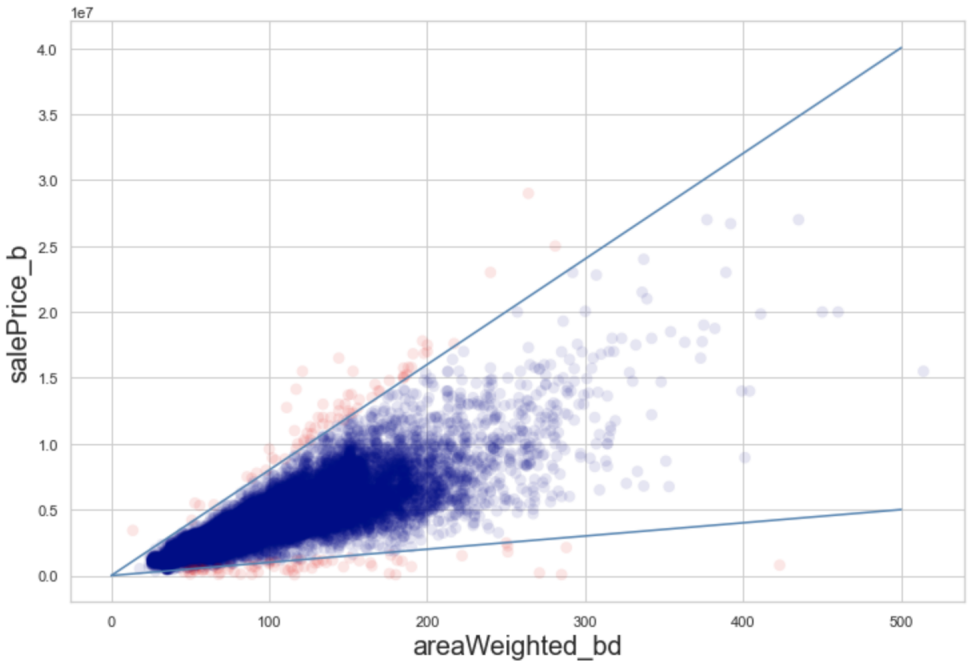

Figure 4.9: Sale price plotted against weighted area. Red dots are excluded outliers.

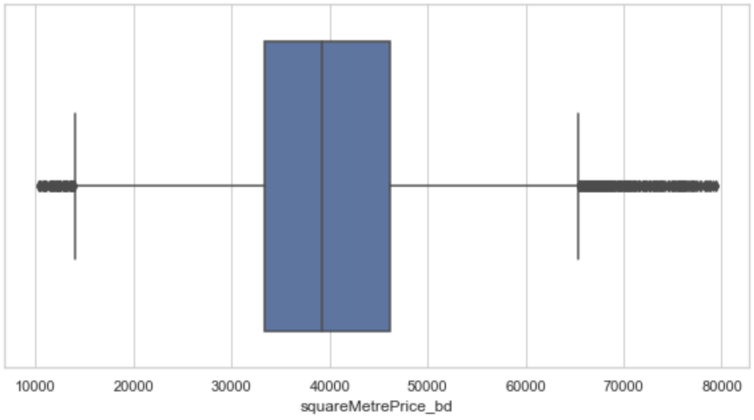

Figure 4.10: Square meter price distribution, excluding outliers.

4.3 Numerical variables

The numerical variables, are all the variables containing quantitative information, meaning the variables which are not categorical. In tables (3.6) through (3.10) in the previous section, an overview was given over the available variables in the dataset, including information about where the information originated. In the following, a new table (4.1) is included summing up mean, minimum and maximum for numeric variables from Boligsiden and DinGeo, excluding Hvorlangterder.dk and Statbank since they upon inspection were unproblematic. From this overview, several variables stand out to have some unrealistic values, which have the potential to have high leverage when fitting the model. These will be dealt with here.

- `propertyValuation_b`. First off, the property valuation can be seen to have a minimum value of 0. It turns out that 7 entries have a valuation of 0, and these are deleted, bringing the observation count down to 30,407. Secondly, there is a maximum valuation DKK 94,000,000 which seems to be way off, considering the maximum of the sales prices. So not to have many of these values with high leverage influencing the model, a rule is

	count	mean	min	max
propertyValuation_b	30,414.00	1,908,024.48	0.00	94,000,000.00
salePrice_b	30,414.00	3,692,396.27	345,000.00	27,000,000.00
paymentCash_b	30,414.00	3,780,124.75	400,000.00	27,500,000.00
salesYear_b	30,414.00	2,018.18	2,016.00	2,020.00
buildYear_b	30,414.00	1,943.14	1,200.00	2,024.00
areaResidential_b	30,414.00	88.98	18.00	410.00
numberOfFloors_b	30,414.00	1.20	0.00	32.00
floor_b	30,414.00	1.95	-1.00	29.00
numberOfBathrooms_b	30,414.00	1.09	1.00	5.00
SalesPeriod_b	30,414.00	88.81	0.00	993.00
latitude_b	30,414.00	55.68	55.62	55.73
longitude_b	30,414.00	12.55	12.45	12.64
rebuildYear_bd	30,414.00	1,952.03	1,200.00	2,024.00
area_bd	30,414.00	88.98	18.00	410.00
areaBasement_bd	30,414.00	6.73	0.00	4,807.00
areaWeighted_bd	30,414.00	91.44	18.00	514.00
areaTotal_bd	30,414.00	89.21	11.00	627.00
numberOfRooms_bd	30,414.00	3.09	0.00	14.00
numberOfToilets_bd	30,414.00	1.17	0.00	22.00
aboveSea_d	30,414.00	7.95	-1.60	37.30
turnoutVote_d	30,414.00	84.45	70.70	89.90

Table 4.1: Numerical variables from Boligsiden and DinGeo.

implemented, where the property valuation cannot be more than 4 times higher or 4 times lower than the sale price. 274 observation do not comply with this, bringing the count down to 30,133.

- `paymentCash_b`. The cash payment can be seen to have a higher max than the sale price. This gives the idea, that these two variables don't always follow each other. Therefore the above rule is used again, that the cash price cannot be more than 4 time greater than the sale price or 4 times lower. 11 observations do not comply with this, bringing the count down to 30,122.

- `buildYear_b` and `rebuildYear_bd`. It turns out that for both of these variables 6 addresses were built/rebuilt in the year 1200 or 2024, which seems very unrealistic. These are therefore deleted, bringing the count down to 30,116.

- `numberOfFloors_b`. With a minimum of zero floors and a maximum of 32 floors, there seems to be something obviously wrong with this variable. Upon further inspection, one can conclude that there must have been confusion in reporting the number of floors for each address. The vast majority does not surprisingly have one floor, but a considerable number of addresses have numbers which are not realistic for housing in Copenhagen. Given this situation, the variable is deleted from the data set.

- `floor_b`. The minimum of this variable is -1, and is not an uncommon value for the variable in the dataset. It must be assumed that this codes for basement level housing.

- `numberOfRooms_bd`. For this variable, a minimum value of 0 seems to be a mistake. 14 addresses are reported as having no rooms, so these are deleted, bringing the count down to 30,102 observations.

- `numberOfToilets_bd`. The maximum of 22 toilets comes from a single outlier observation, which is deleted bringing the count down to 30,101 observations.
- `areaBasement_bd`. For this variable, the maximum value of 4807 m^2 seems quite unrealistic, and one could imagine something like an underground common parking garage to have been counted as basement area. Nonetheless, it does not seem realistic, and a rule is introduced, which excludes all addresses with basements bigger than the weighted area of the address. This excludes 23 observations, bringing the count down to 30,078.

4.3.1 Correlations

In the exploratory data analysis, it is key to have an overview of the pairwise correlations between predictors and responses as well as correlations among predictors themselves. Table (4.2) lists the 12 predictor variables with the highest absolute correlation with the response `salePrice_b`, meaning that these will be most important to include in the model due to their explanatory capabilities.

	correlation
`salePrice_b`	1.00
`paymentCash_b`	0.97
`propertyValuation_b`	0.91
`area_bd`	0.87
`areaResidential_b`	0.87
`areaTotal_bd`	0.86
`areaWeighted_bd`	0.85
`numberOfRooms_bd`	0.75
`numberOfToilets_bd`	0.56
`numberOfBathrooms_b`	0.43
`areaBasement_bd`	0.38
`turnoutVote_d`	0.25
`coast_h`	-0.19

Table 4.2: Variables having highest absolute correlation with sale price.

Not surprisingly, `paymentCash_b` has an extremely high correlation with the target variable `salePrice_b`, since this is the asking price put out on the market. One could argue that this variable is too close to the selling price, and too close an indicator, making the model a bit obsolete. Furthermore, setting the asking price could be said to be part of the pricing process. And since the pricing process is the objective for the model to predict, including this variable could be seen as cutting corners. The variable `paymentCash_b` is therefore excluded from further exploratory analysis, but will return in model building for experimentation.
The other variables that are highly correlated with the target likewise deserve a quick run-through. `propertyValuation_b` has some similarity with the asking price, in that it could be seen as a short-cut to predicting the selling price. But the correlation is lower than for `paymentCash_b`, and from inspection of the data one can learn that the valuation is rarely spot on the selling price. And more importantly, the valuation has in most cases been made a considerable amount of time earlier, so it is realistic to have a valuation price at hand when entering the pricing process. This variable will therefore not be dropped.
From `area_bd` to `areaBasement_bd`, these variables that indicate the hosing size are naturally strongly correlated to the selling price. It seems reasonable, that what in the end has the biggest influence on the price is how big the unit is. The last two variables listed do not have

a particular strong correlation, but are still the highest correlated of the rest of the numerics. `turnoutVote_d` could be assumed to indicate something about the area that influences the price, so it turns out to be an important predictor. Lastly, `coast_h` being the only distance variable on the list is also the only one with a negative correlation to the selling price, meaning shorter distance to a coastline is associated with a higher price.
Scatter plots for these variables against the selling price are given in figure (4.11), and overall one can see how the correlation translates to in most cases a nice linear relationship, maybe with the exception of the `coast_h` variable. There is not much trace of non-linearity in these most highly correlated variables, which is gives an indication, that a linear regression model might be well suited for the job.

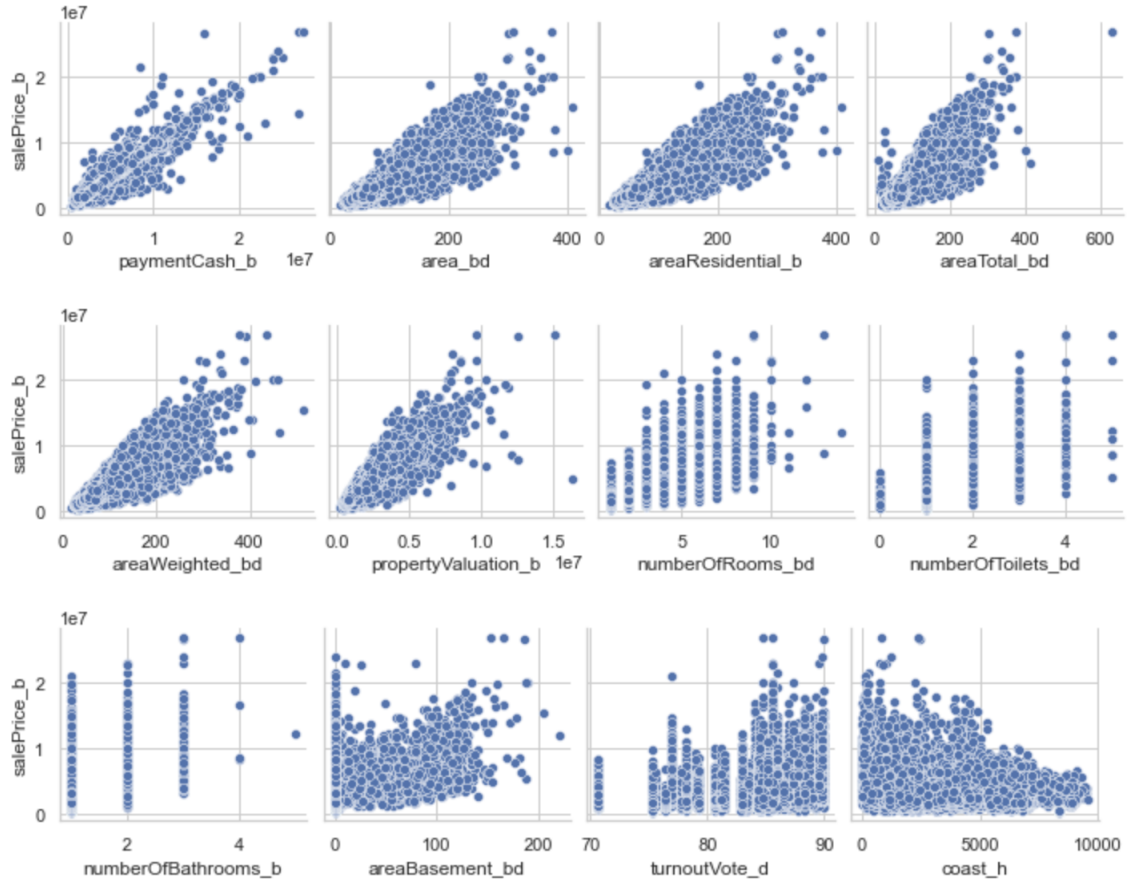

Figure 4.11: Scatterplots for variables most correlated with sale price.

The pairwise correlation between predictors is another concern when fitting a model for the sales price. Collinearity can occur, when predictors are internally correlated, and can pose a problem. Interpretability is compromised when two variables essentially carry the same information, since it becomes hard to determine which of the two has an effect on the target. Furthermore, collinearity results in unstable parameter estimates, since many parameter pairs could give a similar model [Hastie et al., 2001, p. 99]. This instability, or variance, in turn makes testing problematic, since it becomes harder to make hypothesis tests and determine whether there even is association between predictor and response. Hypothesis tests will be mentioned later in the theoretical section. In the heatmap in figure (4.12) the pairwise correlations are given for variables with absolute Pearsons correlation higher than 0.75, in order to look for potential collinearity between them. The 12 variables with substantial correlation with the target, except the cash payment, are likewise included in the map.

On the map, strong positive correlations are indicated by a strong red color, while a strong

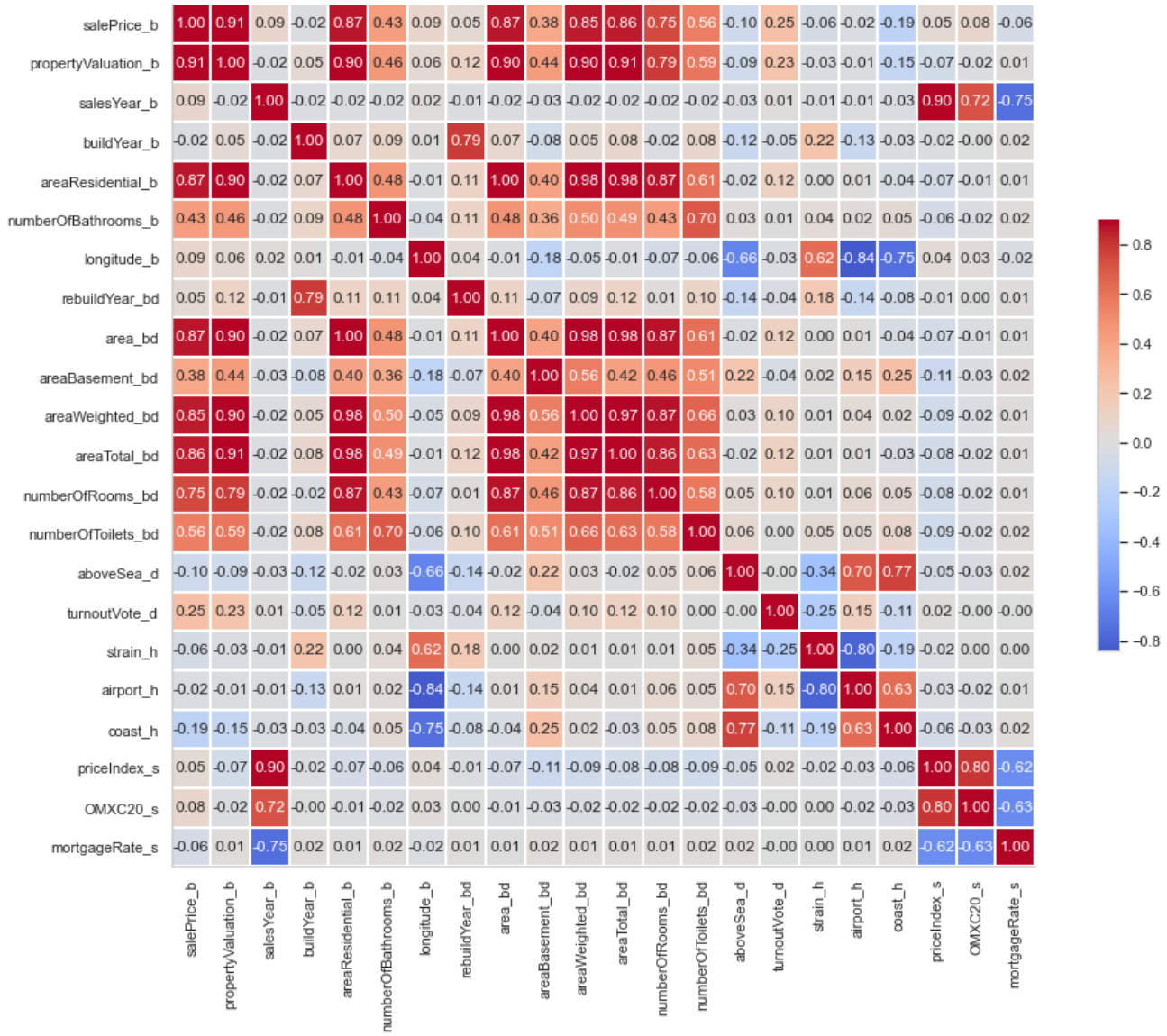

Figure 4.12: Pairwise correlations.

negative correlation is indicated by a strong blue color. What seems very clear from the outset, is that the strongest positive correlation is centred around the housing size variables. These are the same variables that were just discussed to have a high correlation with the target, and therefore of high importance for the regression model. But there are also other small clusters, which deserve attention. In the bottom right corner, the economic indicator variables have considerable correlation. `priceIndex_s` and `OMXC20_s` are relatively highly correlated, which can be attributed to the fact, that price fluctuations usually follow how the economy is going. They are both negatively correlated with `mortgageRate_s`, and here the relationship is less clear. It could be because the rise in housing prices was fuelled by cheaper lending. Regardless, all three economic indicator variables also have high correlations with the variable `salesYear_b`, which is not surprising, since these variables are time series. One could argue that `salesYear_b` could be deleted, since the economic variables, especially `priceIndex_s`, carry the time information, but for practical overview the year variable is kept. In other instances, absolute correlations above 0.90 will be considered a collinearity problem, so none of the economic indicator variables are dropped. Their correlation with the target is also not particularly high, so it is of less importance. Other variables, that are interesting to look into on the map, is for example the

few distance variables included. First of all, `longitude_b` is not surprisingly correlated with these. But since other categoricals will be used as housing placement variables, this variable will be dropped. Naturally, the distance variables are correlated with each other. For example `strain_h` and `airport_h` are negatively correlated due to the fact that S-trains and Copenhagen airport are in opposite ends of town. `coast_h` has high correlation with `aboveSea_d`, because distance to the coast normally would mean to be higher above sea level. None of the distance variables hit correlations of 0.90 or above, and furthermore not very correlated with the target, so they are all kept. And then finally, the highest correlations are as mentioned among the size variables. Only the area variables have almost perfect collinearity, which can be seen in the scatterplots of figure (4.13), and these are therefore dropped also due to their outsized importance for the target, as seen previously. `areaWeighted_bd` is the only one kept and thereby chosen to be the main area variable, since it gives an overall view of the area accessible for the address. `areaWeighted_bd` does not have so high correlations with the remaining sizing variables, such as `numberOfToilets_bd`, that it is a problem, so these variables are also kept. Only `propertyValuation_b` has a correlation with the weighted area at 0.90, but the valuation variable is kept, since it is deemed important and has high correlation with the target.

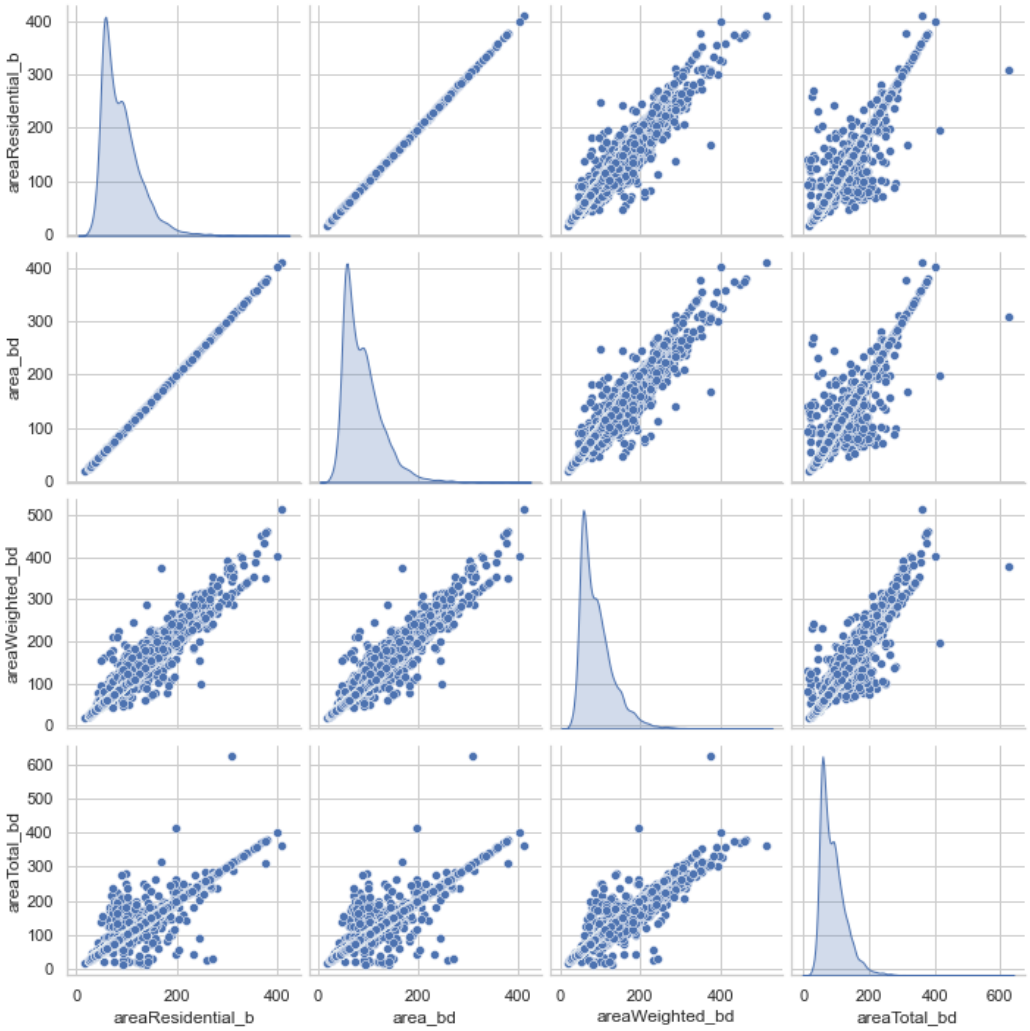

Figure 4.13: Scatterplots for area variables.

4.4 Categorical variables

As the numerical variables have now been evaluated, the rest of the data set still remains. This consists of categorical variables, which are characterized by having values that are not quantitative, but in stead have values drawn from a limited number of categories. The tables (3.6) through (3.10) from earlier gave an overview of which were the categorical variables, and a few of them will be dealt with here, before the encoding process begins.

- **Address variables**: The three variables `address_b`, `street_b` and `streetName_b` all are used to identify a particular address. So especially `address_b`, which is the unique identifier for each observation, will naturally be a categorical with as many categories as are observations, which makes it unusable as a predictor. The same applies for the two other variables, and these are therefore all removed as predictor variables.

- **Placement variables**: The variable `city_b` is usable when it comes to letting the placement of the housing be a predictor variable. Another possibility is to take `postalId_b`, which has the before mentioned problem of too many categories. `postalId_b_range` is created, which groups some of the zip codes, in order to make it a usable categorical variable. In this process, it is found that 248 observations are either in Kastrup or Hellerup, which is not deemed Copenhagen in the sense worked with here. These are therefore deleted, bringing the count down to 29,830 observations. The last placement variable in the data set, `electionArea_d`, is more indirect, but likewise groups together addresses in the same area. Naturally, these variables carry somewhat similar information, and therefore they pose a great potential for collinearity in the model. Therefore `postalId_b_range` is chosen to be kept as placement variable, since the number of categories for this variable is more than `city_b`, which would be too simplistic to use, but less than `electionArea_d`, which would be too complex.

- **Date variables**: `valuationDate_b` and `saleDate_b` likewise have the problem of an outsized number of categories, which is typical with date variables. Earlier it has been pointed out, how the aspect of time is important for the model, when one looks at the price development over the data period. Therefore `salesYear_b` has been included, but it is furthermore decided to include the new variable `'quarter_b'`, based on the knowledge from figures (4.5) and (4.6) which showed the cyclicality in the sales. Thereby these two variables will carry time information, without including the actual date variables.

Another problem that can be present in the categorical variables, is the fact that little variation in a predictor means that it essentially holds no information to be used in the model [Hansen, 2015, p. 65]. So when a predictor has a distribution which is overwhelmingly skewed to a single category, it becomes more or less impossible to detect whether the variable is associated with the response or not. In figures (4.14) and (4.15), six variables exemplify the difference between predictors that have satisfactory variation (the vertical bar plots), and those that do not (the horizontal bar plots). Variables that are dropped on this account include: `propertyType_b`, `toilet.content_b`, `bath.content_b` and `kitchen.content_d`.

4.4.1 Encoding

In order for a model to include categorical predictors, encoding is needed to transform the variables from from having qualitative values to having quantitative values. This is a part of what is called the preprocessing, which is the preperation of the dataset to be suitable for model making. Two different types of encoding will therefore be used, dependent on the nature of the individual predictor. Categorical variables that have categories following a natural order are

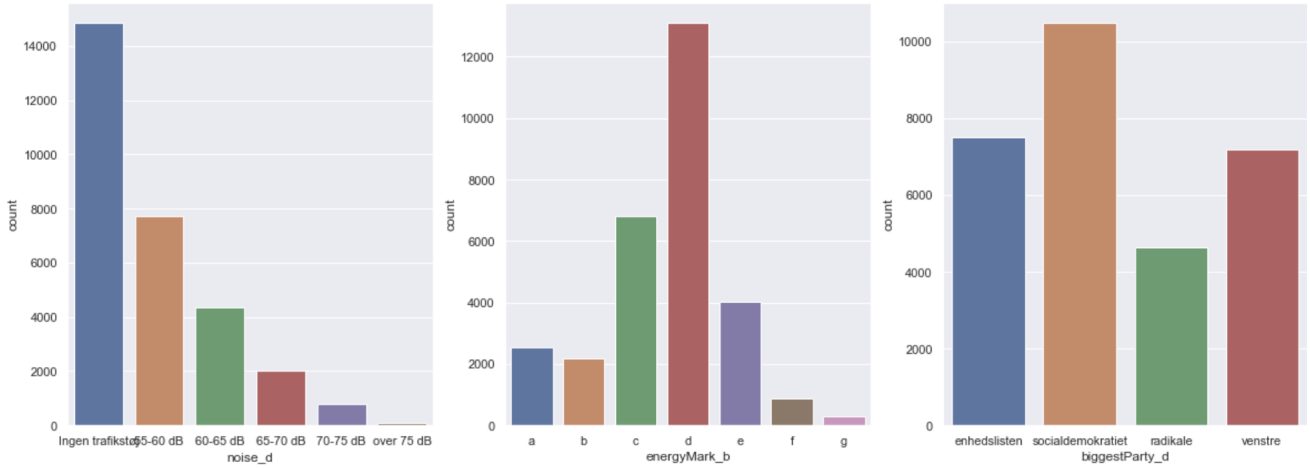

Figure 4.14: Categorical variables with satisfactory variation.

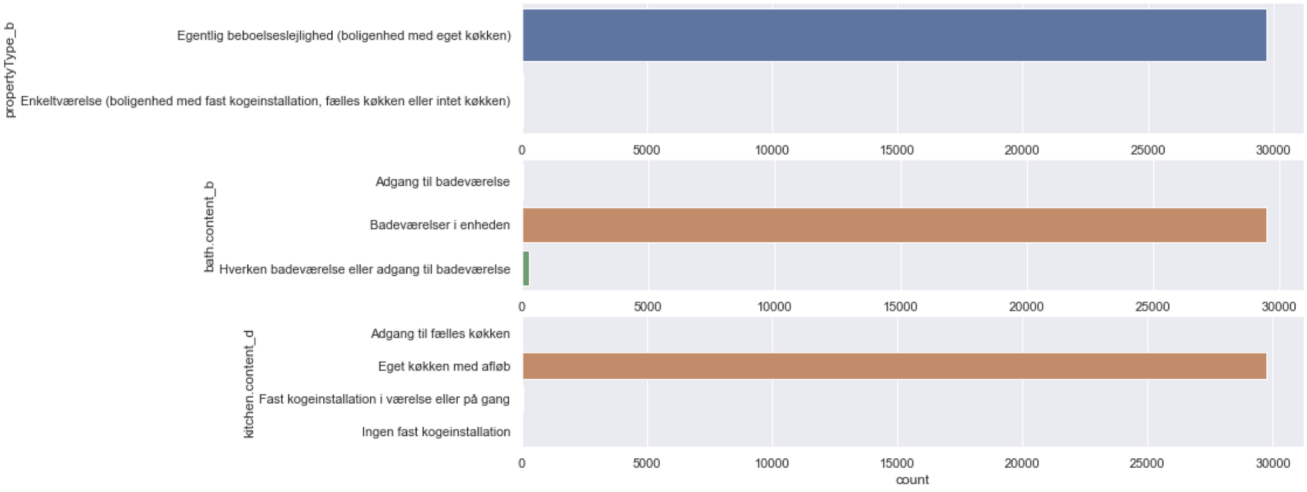

Figure 4.15: ategorical variables with unsatisfactory variation.

the easiest encodable. These are called **ordinal variables**, and as an example one could look at `energyMark_b`, where the order is:

$$a < b < c < d < e < f < g$$

In this case, mapping the predictor to numerical values manually is preferable, in a way that keeps the order intact [Raschka and Mirjalili, 2019, p. 116]. There is no superior way to choose this mapping in terms of values (and numerical difference between these values), only that the order is kept intact. So one way that could seem like an obvious choice could be to let the mapping be given by the dictionary:

$$\{\texttt{'a'}: 1, \texttt{'b'}: 2, \texttt{'c'}: 3, \texttt{'d'}: 4, \texttt{'e'}: 5, \texttt{'f'}: 6, \texttt{'g'}: 7\}$$

The variables `energyMark_b`, `radonRisk_d`, `noise_d`, `floodingRisk_d`, `breakinRisk_d`, `biggestParty_d` and `quarter0_b` are encoded in this way. The advantage of such ordinal categorical variables is that they are transformed directly, and do not result in more predictors in the dataset. This is not the case for the other type of encoding, **one-hot encoding**, performed on **nominal variables**, which are categorical predictors with no natural order. For each predictor, binary dummy variables are created, so that the number of dummy variables counts the number of categories with one subtracted [Raschka and Mirjalili, 2019, p. 120]. This means that a

dummy variable `itemTypeName_b_Villa` is 1 for observations where `itemTypeName_b` is "Villa", and 0 in all other cases. The reason for having one less dummy variable than the number of categories, is the risk of multicollinearity in an overparametrized model. This is especially a problem in a linear regression model that requires matrix inversion, where it can result in unstable parameter estimates as mentioned earlier. Dropping one dummy takes care of this problem without any loss of information, since an observation with value in the dropped category will just receive zeros in all dummy variables. This can be seen as the dropped group acting as a baseline, and the parameters for the dummy variables are differences from this baseline, also called contrasts [Hansen, 2015, p. 33]. Predictors which are encoded in this way are `postalId_b_range`, `itemTypeName_b`, `unitUsage_bd`, `outerwall_d'`, `roof_d` and `heating_d`, which raises the number of predictors from 50 to 96.

4.4.2 Correlations

The heatmap in figure (4.12) gave an overview of the correlations, when only the numerical variables were included. The categorical variables have now been encoded, and it is therefore possible to likewise create a heatmap in figure (4.16) for the correlations where these predictors are included. Similarly to what was done earlier, the variables with the highest absolute correlation with the target variable as well as variables with the strongest absolute pairwise correlation (minimum absolute value of 0.75) are included. What becomes clear, is that when it comes to the numerical variables the heatmap is more or less unchanged, meaning the upper left corner is as earlier. It is the same variables that are highly correlated with the target, and the same pairwise relations are kept. So size variables are still of great importance for the target, and the variables are highly correlated internally among each other. The same relations between economic indicator variables are present, and the same relationships between distance variables. Even though the picture is more or less the same, it is important to include the numerics, since there could be collinearity problems between numerical and categorical variables as well.
Looking at the new addition, what first catches the eye is that the dummies of `itemTypeName_b` and `unitUsage_bd` have considerable correlation with the target, as well as with housing size variables. An example is that `itemTypeName_b_Villa` and `unitUsage_bd_Fritliggende enfamilieshus (parcelhus)` both have a 0.75 correlation with `areaBasement_bd`. This seems sensible, since the two dummies are variables that indicate the type of housing, in which one would expect to have basement space. Furthermore, the two housing type dummy variables are perfectly correlated with a correlation of 1. Looking at all the dummies of `itemTypeName_b` and `unitUsage_bd` one sees how the strong correlations seem to be general, and that there is great risk of collinearity in the variables. On further inspection, this is not surprising, since the value groups for each categorical variable contain more or less the same information, as was the case with "Villa" and "Parcelhus". So to not risk collinearity, dummies from `unitUsage_bd` are deleted. As indicated earlier, information is not lost, since it is still kept by the `itemTypeName_b` dummies. Other correlations to note are `outerwall_d_Glas` and `roof_d_Glas`, which have a positive correlation of 0.81, indicating that housing with a glass roof often also have glass walls. It is not a candidate for excluding a variable, because the other dummies for `outerwall_d` and `roof_d` are not correlated in this way. Lastly, `postalId_b_range_2300` is the only placement variable dummy, which is highly correlated with other variables, this being the distance variables. This is because Amager is the only area which is clearly close to the airport and far from S-trains. Once again, the high correlation is only the case for one of the dummies of `postalId_b`, so this is also not a candidate for exclusion. In stead, this marks the end of the exploratory data analysis.

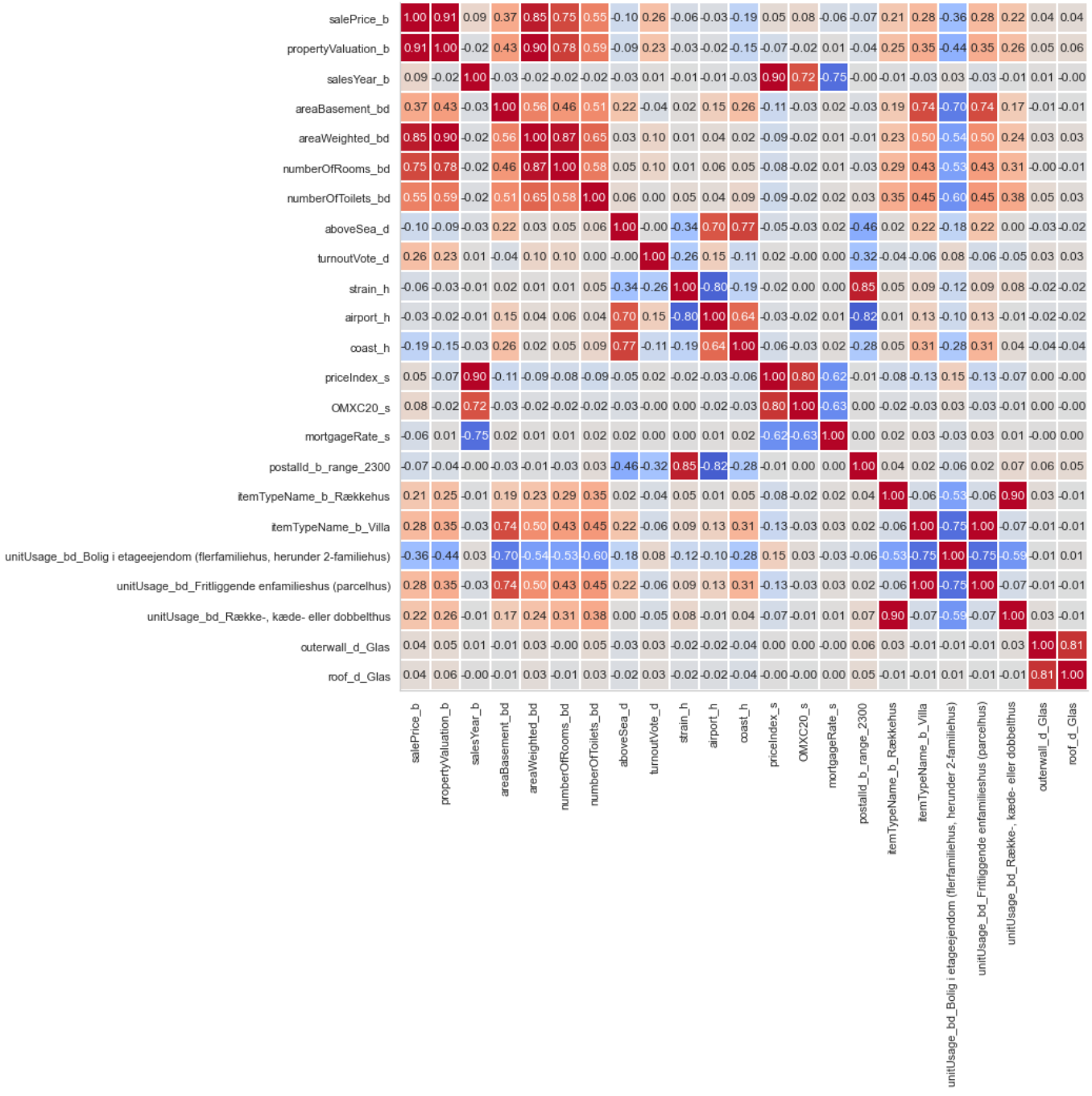

Figure 4.16: Pairwise correlations.

Chapter 5

Theoretical background

5.1 Supervised learning

When talking about machine learning, the more standard term of use is supervised learning. Supervised learning is in contrast to unsupervised learning the situation, where the observations at hand can be divided into input variables (predictors or features) and output variables (responses or labels), which we see as dependent on the input variables. Supervised learning can then be described as the exercise of creating a model based on paired observations, which then can predict the responses based on predictors. Intuitively, we are learning a model function $\hat{f}$ from the given data, which we want to approximate the assumed underlying function f that underlies the predictive relationship between inputs and outputs. Unsupervised learning deals with observation sets, where there are no obvious responses, which means the learning aspect deals with how to group the data, but this is not relevant here. When the response in the learning problem is quantitative, as is the case in real estate prices, the problem is known as regression, while qualitative responses calls for classification techniques.
This section will broadly be based on theory and notation taken from [Hastie et al., 2001], with additional notes drawn from [James et al., 2013].

5.1.1 Notation

The standard way to describe the setup within supervised learning is to let $X \in \mathbb{R}^p$ denote the predictors, or in other cases as a scalar input as well as matrix input. Then $Y \in \mathbb{R}$ will denote the response, where in other cases Y could likewise be a vector. Observation pairs in a training data set are $(x_i, y_i)_{i \in N}$ (lower case is used for observed variables), which are then used to train or learn the model, which is the objective of supervised learning. Let the model be denoted by f, so that

$$Y = f(X) + \epsilon \tag{5.1}$$

$\epsilon \sim N(0, \sigma_\epsilon^2)$ indicates that we assume some normally distributed noise in the observed responses (independent for each observation). This i.i.d. assumption for the data is not strictly necessary for supervised learning, and would be relaxed if working with sequential data. But it does make the model setup a lot more simple, and since there is no clear interdependence between house sales, the i.i.d. assumption is appropriate. Including the error term means that there will be a small variability in the output of the true model, which no f can account for. This is usually a useful approximation to the truth, since we then relax that input and output should have a deterministic relationship. So we take into account unmeasured variables that also contribute to the response, including measurement error. This is an example where an additive error is assumed, as is often standard. $\hat{Y}$ will then denote the estimate of the output given the estimated model function $\hat{f}$. In general, anything denoted with a hat signifies that this is an estimate.

5.1.2 The general problem of supervised learning

As stated above, the objective is to find a function f that underlies the predictive relationship between inputs and outputs. To do this on a more theoretical level, let X and Y be random variables with a joint distribution, so that we can look at the expected deviation of $f(X)$ from Y. This is because we are looking for an f with the best expected fit, meaning the best on average fit, and not just for a single observed case. A loss function penalizing prediction errors is needed for measure of fit, and the most common is squared error loss. This leads to the criteria when choosing f, that we want it to minimize expected squared prediction error (EPE).

$$\mathrm{EPE}(f) = \mathrm{E}_{(X,Y)}(Y - f(X))^2 \tag{5.2}$$

We have from [Hastie et al., 2001, p. 18] that this can be written as the following by conditioning:

$$\mathrm{EPE}(f) = \mathrm{E}_X \mathrm{E}_{Y|X}([Y - f(X)]^2|X) \tag{5.3}$$

It then suffices to minimize EPE pointwise

$$f(x) = \mathrm{argmin}_c \mathrm{E}_{Y|X}([Y - c]^2|X = x) \quad \Rightarrow \quad f(x) = \mathrm{E}(Y|X = x) \tag{5.4}$$

Because for each $X = x$, and any f

$$\mathrm{E}_{Y|X}([Y - f(X)]^2|X = x) = \mathrm{E}_Y([Y - f(X)]^2|X) = \mathrm{E}_Y([Y - \mathrm{E}(Y|X) + \mathrm{E}(Y|X) - f(X)]^2|X) \tag{5.5}$$

$$= \mathrm{E}_Y([Y - \mathrm{E}(Y|X)]^2|X) + \mathrm{E}_Y([\mathrm{E}(Y|X) - f(X)]^2|X) \tag{5.6}$$

$$+ 2\mathrm{E}_Y([Y - \mathrm{E}(Y|X)][\mathrm{E}(Y|X) - f(X)]|X) \tag{5.7}$$

Here $[\mathrm{E}(Y|X) - f(X)]$ is a deterministic factor for a given X and it then follows

$$= \mathrm{E}_Y([Y - \mathrm{E}(Y|X)]^2|X) + \mathrm{E}_Y([\mathrm{E}(Y|X) - f(X)]^2|X) \tag{5.8}$$

$$+ 2[\mathrm{E}(Y|X) - f(X)]\mathrm{E}_Y([Y - \mathrm{E}(Y|X)]|X) \tag{5.9}$$

$$= \mathrm{E}_Y([Y - \mathrm{E}(Y|X)]^2|X) + \mathrm{E}_Y([\mathrm{E}(Y|X) - f(X)]^2|X) + 0 \tag{5.10}$$

$$\tag{5.11}$$

And the last equality follows from the linearity of expectations. The inequality then immediately follows

$$\geq \mathrm{E}_Y([Y - \mathrm{E}(Y|X)]^2|X) \tag{5.12}$$

So when using this loss function, the solution to the minimization problem is the conditional expectation of Y given X. It seems very intuitive, that an ideal function f should have this property, that it outputs the average Y of the given situation $X = x$. f that solves this problem is called the *regression* function, and is an ideal that we want to approximate in supervised learning. In reality rather than expectations, we must use approximations, like averaging and assumption of model structure, in order to learn f from a given training data set. So EPE can be seen as the expected *test* or *generalization* error, since it deals with expectations, where the training error is only an approximation used for fitting.

Given the additive error assumption one can further look at EPE above and look at the squared error loss (Err) for an input $X = x_0$ (as is done in [Hastie et al., 2001, p. 223]):

$$\begin{aligned}
\mathrm{Err}(x_0) &= \mathrm{E}([Y - \hat{f}(x_0)]^2 | X = x_0) && (5.13)\\
&= \mathrm{E}([f(x_0) + \epsilon - \hat{f}(x_0)]^2) && (5.14)\\
&= \mathrm{E}([f(x_0) - \hat{f}(x_0)]^2) + 2\mathrm{E}([f(x_0) - \hat{f}(x_0)]\epsilon) + \mathrm{E}(\epsilon^2) && (5.15)\\
&= \mathrm{E}([f(x_0) - \hat{f}(x_0)]^2) + \sigma_\epsilon^2 && (5.16)
\end{aligned}$$

This follows the assumption of the error distribution, $\epsilon \sim N(0, \sigma_\epsilon^2)$, and the independence of the error term. Further one can divide the first term into a bias- and a variance term as follows:

$$\begin{aligned}
\mathrm{E}([f(x_0) - \hat{f}(x_0)]^2) &= \mathrm{E}\left[\left((f(x_0) - \mathrm{E}[\hat{f}(x_0)]) - (\hat{f}(x_0) - \mathrm{E}[\hat{f}(x_0)])\right)^2\right] && (5.17)\\
&= \mathrm{E}\left[\left(\mathrm{E}[\hat{f}(x_0)] - f(x_0)\right)^2\right] + \mathrm{E}\left[\left(\hat{f}(x_0) - \mathrm{E}[\hat{f}(x_0)]\right)^2\right] && (5.18)\\
&\quad - 2\mathrm{E}\left[\left(f(x_0) - \mathrm{E}[\hat{f}(x_0)]\right)\left(\hat{f}(x_0) - \mathrm{E}[\hat{f}(x_0)]\right)\right] && (5.19)
\end{aligned}$$

The first term is seen to be deterministic and in the third term there is likewise a deterministic factor.

$$\begin{aligned}
&= \left(\mathrm{E}[\hat{f}(x_0)] - f(x_0)\right)^2 + \mathrm{E}\left[\left(\hat{f}(x_0) - \mathrm{E}[\hat{f}(x_0)]\right)^2\right] && (5.20)\\
&- 2\left(f(x_0) - \mathrm{E}[\hat{f}(x_0)]\right)\mathrm{E}\left[\left(\hat{f}(x_0) - \mathrm{E}[\hat{f}(x_0)]\right)\right] && (5.21)\\
&= \left(\mathrm{E}[\hat{f}(x_0)] - f(x_0)\right)^2 + \mathrm{E}\left[\left(\hat{f}(x_0) - \mathrm{E}[\hat{f}(x_0)]\right)^2\right] - 0 && (5.22)\\
&= \mathrm{bias}(\hat{f}(x_0))^2 + \mathrm{var}(\hat{f}(x_0)) && (5.23)
\end{aligned}$$

This follows from linearity of expectations, and the two terms left is recognized as the *bias* and *variance* of the estimated model.

In the above calculations the unknown true model function f is fixed, while the estimated model function $\hat{f}$ is random, since it depends on realizations of the joint distribution of X and Y (which has noise), and therefore changes every time the experiment is done. So naturally we will not get the same $\hat{f}$ every time a new sample is given to train the model. Err then denotes the expected (squared) prediction error, EPE, of the random $\hat{f}$ on a point by point basis, meaning a measure of how much we expect $\hat{f}$ to deviate from the response in a single given input point. The first insight above is that we can break this measure of fit down into the expected mean squared error of $\hat{f}(x_0)$ estimating $f(x_0)$ and then an irreducible error given by σ^2 due to the error term. No matter if the random $\hat{f}$ would assume the true model function on average, we would still expect a squared error on average determined by the variance of the error term. The second insight is that the first term can be broken down into two terms. Firstly a squared bias term, measuring the difference between the mean of the fitted function $\hat{f}$ in the input point and the true mean in this point (since $f(x_0)$ is the mean of $f(x_0) + \epsilon$). For a high bias, the means would be very different from each other in the input point, signifying a bad fit. Secondly, the variance term measures the variance of the fitted model, meaning the variability in $\hat{f}$ coming from the different realizations of observations. So a high variance would mean that the fit is not very stable, since new data would give a possibly very different value of the fitted

model in this input point.
Ultimately, Err can be broken down into an irreducible error and then an error term that can be split into the bias and the variance of the model fit, know as the *bias-variance decomposition*. So since we are looking at measures of quality of fit, it is relevant to look at these two components when deciding if a given model structure is adequate. Oftentimes there is a trade-off between bias and variance, when one decides on the model complexity, as shown in figure (5.1). A very complex model might fit the observations closely in every realization, and would then tend to have a low bias, because the mean fitted value would be close to the true mean. But a complex model could still have a high Err, because every fit would be very different (high variance), and one fitted model would fit badly to the next batch of observations. This is the concept of *overfitting*. Conversely, a simple but rigid model further from reality would have low variance over different batches, but might fit the observations badly and therefore have a high bias resulting in a high Err. So both bias and variance are relevant for the quality of fit when choosing a model, which should be balanced in order to achieve a low Err, and thereby low EPE. Which model one should decide on then becomes a matter of assessing the data at hand with these criteria in mind.

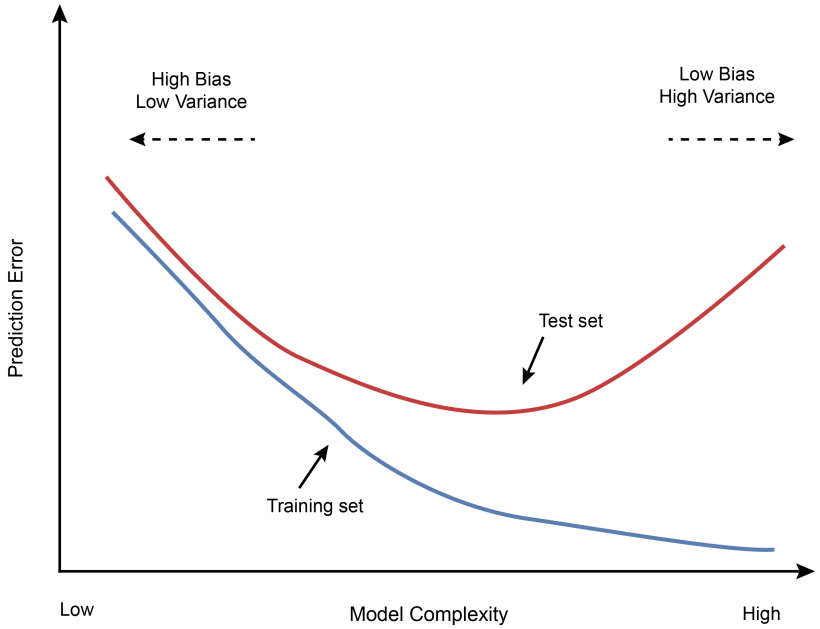

Figure 5.1: The Bias-Variance trade-off.

5.1.3 Cross validation for quality of fit

In order to assess the quality of fit or compare different models, a way to estimate the expected prediction error, or the expected test error, is needed. The training error is a poor estimate, since it is the same measure that is minimized when the model is fitted. So naturally the training error will always give the impression that the model estimates the data well, but that is of course the training data at hand. When assessing how close the model is fitted, one can use mean squared error as a measure given by

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{f}(x_i))^2 \tag{5.24}$$

So this gives an impression of the training error, while the expected mean squared error would be the equivalent of the EPE, which is what actually is of interest when deciding the best model. Because even though a hard fit gives a small training error, the fit might still be very bad on new data, because the variance aspect is not included in the training error. The expected test error on the other hand deals with the general (thereby generalization error) meaning: how well would we do with another training set and another test set drawn from the same distribution?
One would usually split up the given data at hand in a training set, a validation set, and then finally a test set. The training set is of course the data that the model is fitted over, and then the validation set is then used to validate the model. Validating the model entails trying the model on new data in order to get an estimate of the generalization error, but this can be done various times in order to get the best model of choice, by tweaking parameters in the model and so on. Therefore it is not actually a good estimate of the generalization error, since the model will be modified in order to do well on the validation set. Therefore the test data is left untouched, since one wants an unspoiled sample, which finally will give the best estimate of the generalization error, when a final model is chosen.
So the validation set has the role of estimating the expected test error, while still in the process of choosing the optimal model. But taking out data for validation gives the model a smaller set to be trained on, and therefore a method is desired, which can get around this hurdle, and still provide an estimate of the expected test error. It turns out that the method of cross validation is one such metod for estimating the expected test error in validation using training data, and here the focus will be on K-fold cross validation specifically. Let $\kappa : \{1, ..., N\} \to \{1, ..., K\}$ be an indexing function that divides the full training data into K groups. The model is fitted K times, where each time the observations in one of the K groups is left out and saved for validation. For each model fit, a validation error is then calculated using the separated validation set, giving us K validation sets. The average of these errors then is the cross validation estimate. More condensed, this can be written as follows:

$$\mathrm{CV}(\hat{f}) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{f}^{-\kappa(i)}(x_i))^2 \tag{5.25}$$

Here $\hat{f}^{-\kappa(i)}$ denotes the fitted model where group $\kappa(i)$ is removed. One can notice that the cross validation estimate is actually an average of MSE's on all the validation sets of the smaller fitted models.

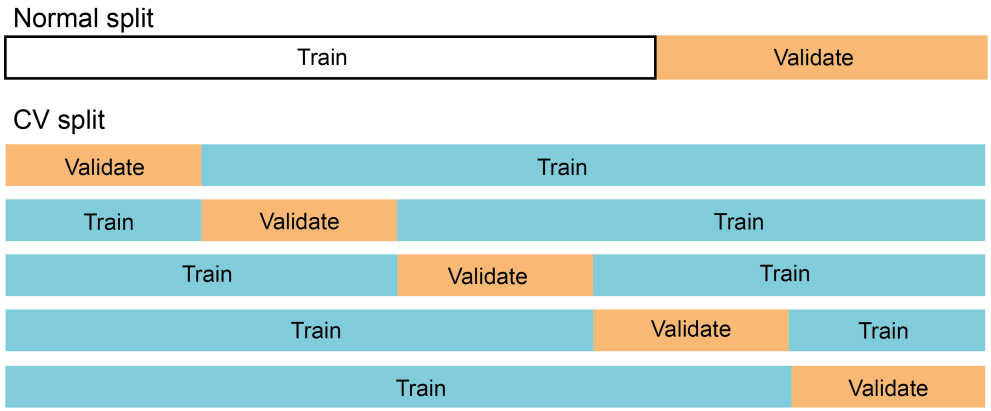

Figure 5.2: 5-fold Cross Validation.

To sum up, through cross validation we use the data at hand to estimate the expected test error, because we are averaging over different fitted models with different validation sets. It

aims to show how well we can expect the model to fit in general, when new data is drawn from the same distribution as our observations, such as the test set. So why not just let the observations be divided into one training set and one validation set, and let the error on the validation set estimate the generalization error? From this we would only obtain one estimate of the expected test error. We would only have trained on about half the observations, which in general would give a worse fit with significant bias, and therefore the estimate would overestimate the expected test error. At the same time, the estimate would be very dependent on how the partition was done, so high variability in the estimate can be expected. We get around these two problems by using cross validation, since we are fitting from bigger sets, and averaging gives more stability. That being said, K has to be chosen wisely. If K is too high, then the training set is almost the same for every group and the estimates will be very alike, and therefore very positively correlated. So the average would have high variance, since new training data could give something very different. When K is lower we don't get this problem, since the fitting of models are different and give different estimates. But in this case the training sets get smaller, which means more bias. It is empirically shown, that $K = 5$ and $K = 10$ are good choices, in order to balance these considerations.

5.2 Linear models

The obvious model to start with within supervised learning is to set up the simple linear model, which is a model based approach and therefore assumes a structure of f. It is additive, and the parameters are given by the vector β.

$$\hat{Y} = \hat{f}(X) = \hat{\beta}_0 + \sum_{i=1}^{p} X_i \hat{\beta}_i = X^T \hat{\beta} \tag{5.26}$$

The general linear model is linear in the parameters, so the model would not need to be linear in the predictors, as in the example above. But this is the most simple case of multiple linear regression, which will be applied in the model making. Here the last equality to the inner product notation can be made if we include $X_0 = 1$, so that the intercept β_0 does not have to be written separately. For this linear model the most standard form of regression used to fit the model to the data, meaning learning in the quantitative response setting, is ordinary least squares (OLS).

5.2.1 OLS

OLS corresponds to finding a β, which minimizes the residual sum of squares (RSS). In other words we are finding a linear plane, that best fits the noisy observations, and in the case of the linear model, OLS gives a closed form solution. Fitting other models can often times entail fitting through numerical methods.

$$\mathrm{RSS}(\beta) = \sum_{i=1}^{N} (y_i - x_i^T \beta)^2 = (\mathbf{y} - \mathbf{X}\beta)^T (\mathbf{y} - \mathbf{X}\beta) \tag{5.27}$$

Here $\mathbf{X}$ is an $N \times (p+1)$ matrix, where row i is x_i. Differentiating with regards to β is easily done in the matrix notation, and the minimization problem becomes

$$\frac{\partial}{\partial\beta}\text{RSS}(\beta) = 2(\mathbf{y} - \mathbf{X}\beta)^T\mathbf{X} = 0 \tag{5.28}$$

$$(\mathbf{y}^T - \beta^T\mathbf{X}^T)\mathbf{X} = \mathbf{y}^T\mathbf{X} - \beta^T\mathbf{X}^T\mathbf{X} = 0 \tag{5.29}$$

$$\beta^T\mathbf{X}^T\mathbf{X} = \mathbf{y}^T\mathbf{X} \tag{5.30}$$

$$\mathbf{X}^T\mathbf{X}\beta = \mathbf{X}^T\mathbf{y} \tag{5.31}$$

$$\hat{\beta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} \tag{5.32}$$

The unique solution to the problem exists if $(\mathbf{X}^T\mathbf{X})^{-1}$ exists, meaning if $\mathbf{X}^T\mathbf{X}$ is nonsingular.

From here, any input in the model will yield an output estimate as described in the assumed model above (5.26).

It turns out that this estimate gotten from performing OLS regression is actually the maximization estimate in the normal linear model. Cause if we further let (which we will in all instances of the thesis) the model error be additive, Gaussian and independent, and assume the true structure of f to be in fact linear, then we have the model

$$\mathbf{y} \sim N(\mathbf{X}\beta, \sigma_\epsilon^2\mathbf{I}) \tag{5.33}$$

on the vector space $\mathbf{y} \in \mathbb{R}^N$. Here the maximization estimator is the estimate $\hat{\beta}$, that would maximize the likelihoodfunction given our observations. This means that $\hat{\beta}$ gives the model, that is most in line with the observations. The maximization estimator is found through the orthogonal projection from the observed $\mathbf{y}$ down on the mean value subspace $L = \{\mathbf{X}\beta \mid \beta \in \mathbb{R}^{p+1}\}$, which is the columnspace of the design matrix $\mathbf{X}$ [Hansen, 2002, 10.19]. And the orthogonal projection is given by [Hansen, 2002, 10.8] as seen in figure (5.3).

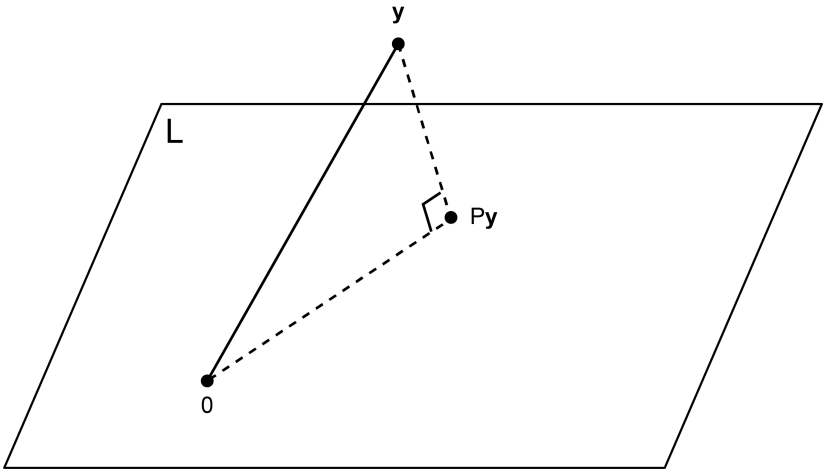

Figure 5.3: Maximization estimator as orthogonal projection.

$$P\mathbf{y} = \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} = \mathbf{X}\hat{\beta} \quad \Rightarrow \quad \hat{\beta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} \tag{5.34}$$

and this can be seen to be the same $\hat{\beta}$ as in the OLS case. So how come that minimizing RSS gives the maximization estimator? This becomes apparent if one takes a look at the log-likelihood function for $\mathbf{y}$ [Igel, 2020, 3.2.2].

$$-\ln p(\mathbf{y}|\mathbf{X}) = -\ln \prod_{i=1}^{N} \frac{1}{2\pi\sigma_\epsilon^2} e^{-\frac{(y_i - x_i^T\beta)^2}{2\sigma_\epsilon^2}} \tag{5.35}$$

$$= -\sum_{i=1}^{N} \ln\left(\frac{1}{2\pi\sigma_\epsilon^2} e^{-\frac{(y_i - x_i^T\beta)^2}{2\sigma_\epsilon^2}}\right) \tag{5.36}$$

$$= -\sum_{i=1}^{N} \ln\left(\frac{1}{2\pi\sigma_\epsilon^2}\right) + \frac{1}{2\sigma_\epsilon^2}\sum_{i=1}^{N}(y_i - x_i^T\beta)^2 \tag{5.37}$$

The first term does not depend on β, so minimizing the log-likelihood function (that give the maximization estimate) becomes a matter of minimizing the last term. This is equivalent to minimizing RSS in OLS regression, and thereby the connection becomes apparent between the two estimates in this model.

How does fitting a model through OLS relate to the original problem of minimizing EPE? It turns out that (5.32) is actually an approximation of the ideal regression function, but through heavy assumptions. Looking at (5.2), the similarity to RSS is apparent, since both expressions deals with a squared residual term. Inserting the assumption that $f(x) \approx x^T\beta$ into EPE:

$$\mathrm{EPE}(f) \approx \mathrm{E}(Y - X^T\beta)^2 \tag{5.38}$$

$$\frac{\partial}{\partial\beta}\mathrm{E}(Y - X^T\beta)^2 = \mathrm{E}\left(\frac{\partial}{\partial\beta}(Y - X^T\beta)^2\right) \tag{5.39}$$

$$= \mathrm{E}\left(2(Y - X^T\beta)^T(-X^T)\right) \tag{5.40}$$

$$= \mathrm{E}(2\beta^T XX^T) - \mathrm{E}(2Y^T X^T) \tag{5.41}$$

$$= 0 \tag{5.42}$$

$$\mathrm{E}(\beta^T XX^T) = \mathrm{E}(Y^T X^T) \tag{5.43}$$

$$\mathrm{E}(XX^T)\beta = \mathrm{E}(XY) \tag{5.44}$$

$$\beta = [\mathrm{E}(XX^T)]^{-1}\mathrm{E}(XY) \tag{5.45}$$

Here it's assumed that conditions of regularity are met for reversal of differentiation and expectation, following a dominated convergence argument [Hansen, 2006, 8.14].
This is not the regression function minimizing EPE, but an approximation of it, where we have made the linear assumption of the model structure. But it is now clear, that (5.32) is then a further approximation of the approximation to the solution to the original problem, where averaging is used rather than expectations. So when using OLS, we are essentially using what we have in order to come close to fitting a model, that minimizes EPE. Averaging over sample data (training data) approximates the expectations in the above expression, which is itself an approximation of the regression function, where linearity is assumed.
About linear regression using ordinary least squares we know that the approach is very stable, in other words this model has very low variance. On the other hand, if the true model is very far from the linear assumption made, then a linear fit will not be very accurate, in other words the model will have a high bias. This is particular to the OLS for the linear regression model, which is a smooth solution. The opposite is the flexible case for a regression method as k-nearest neighbours, which is not a method which will be dealt with further here. But in both cases, one would have to assess the data, in order to decide a model structure that would balance the bias-variance tradeoff and keep the expected error EPE low.

Given the assumed model setup (that f truly has the linear structure assumed), it can easily be shown, that the maximization estimator is normally distributed as $\hat{\beta} \sim N(\beta, (\mathbf{X}^T\mathbf{X})^{-1}\sigma_\epsilon^2)$. Another estimator conditioned on this assumed setup is given by

$$\hat{\sigma}^2 = \frac{1}{N-p-1}\sum_{i=1}^{N}(y_i - \hat{f}(x_i))^2 \tag{5.46}$$

which is not the maximization estimator for σ_ϵ^2, but the unbiased estimate, since it doesn't systematically underestimate the true variance. About this estimator it can be shown that $(N-p-1)\hat{\sigma}^2 \sim \sigma_\epsilon^2\chi^2_{N-p-1}$, and along with $\hat{\beta}$ it can therefore be used to construct the Z-score:

$$z_j = \frac{\hat{\beta}_j}{\hat{\sigma}\sqrt{v_j}} \tag{5.47}$$

where v_j is the jth diagonal element in $(\mathbf{X}^T\mathbf{X})^{-1}$. This score is a test size used for testing the hypothesis that $\beta_j = 0$, meaning there is no association between jth predictor and the response. Under this hypothesis $z_j \sim t_{N-p-1}$, where high values of $|z_j|$ are critical for the hypothesis, since low variance in the data and a $\hat{\beta}_j$ far from zero gives a clear indication that $\beta_j = 0$ is not the case. From the distribution can be derived a p-value, which is the probability under the hypothesis to get a given absolute Z-score or higher. This means that a very low p-value is critical for the hypothesis, and one can therefore reject it and assume significant association between predictor and response.

Since $\hat{\sigma}$ says something about the on average deviation from the model, it can also be used as a measure of quality of fit. Residual standard error, RSE = $\sqrt{\hat{\sigma}^2}$, is often used in this respect, and is exactly the average deviation in the observations. But this is an average in absolute terms, and sometimes one might want a measure of fit that is comparable in relative terms. This is what R^2 is used for, defined as

$$R^2 = \frac{\text{TSS} - \text{RSS}}{\text{TSS}} \tag{5.48}$$

where TSS = $\sum(y_i - \overline{y})^2$. R^2 is a measure of the proportion of the variability, that is explained by the model. TSS is a measure of the overall variability in the data. RSS is an expression of the deviation of the observations from the model, or the variability still remaining in the model. So if TSS is much larger than the RSS, then R^2 is close to 1, and we say that a lot of the variability is explained by the model, meaning a good fit. Alternatively, if they are of similar size, R^2 will be closer to 0, and we interpret that the variability around the model is not much different than the overall variability - so in this case the model did not catch much, and the quality of fit is poor. And furthermore, no assumptions about the model have to be made in order to use R^2, or RSE for that matter, as a measure of lack of fit, so we are free to use them in nonlinear regression as well.

5.2.2 The Lasso

When using the lasso method, the fitting procedure is slightly different from OLS, where the RSS are minimized, as written above. The lasso methods entails a penalty term whereby the fitting problem becomes minimizing the following [James et al., 2013, p. 219].

$$\sum_{i=1}^{N}(y_i - x_i^T\beta)^2 + \alpha\sum_{j=1}^{p}|\beta_j| \tag{5.49}$$

The penalty in the expression is controlled by the tuning parameter α, and for $\alpha = 0$ it becomes the regular least squares problem. Selecting the right size of α is critical to the fit, where lowering the expected test error is the objective. And since the cross validation method is one way to estimate the expected test error, this method is used to choose the α to yield the model with the smallest estimated test error.
The lasso is a shrinkage method, where the name is related to the shrinking of the parameters. This has the potential to lower variability in the parameter estimates, meaning lower variance in the fitted model. So in cases of high variance in the OLS fitting, the lasso can lower the variance with only a small increase in the bias. In other words, we are avoiding potential overfitting on the training data from having too many parameters. This in turn we know means lower expected test error, which is the objective. Shrinkage methods also have computational benefits, because they are more feasible when the dimension p is high, in contrast to subset selection.
The lasso uses an ℓ_1 penalty, which causes it to shrink some of the coefficient estimates to 0, when α is large enough. This is in contrast to the very similar ridge regression, which uses an ℓ_2 penalty, resulting in no coefficients shrunken to zero. In our case of large dimension p, it is beneficial for interpretation, to effectively perform variable selection through the lasso. This is why the lasso has been included rather than the ridge regression, even though ridge regression tends to give slightly lower coefficient estimates with slightly lower variance.
Another way to express the lasso regression problem is minimizing the following, subject to a condition:

$$\sum_{i=1}^{N}(y_i - x_i^T\beta)^2 \quad \text{subject to} \quad \sum_{j=1}^{p}|\beta_j| \leq t \tag{5.50}$$

It can be shown that the two ways mentioned of formulating the problem are equivalent, where the tuning parameter in the latter is t. The latter formulation gives an intuitive feel of why the lasso shrinks some coefficients to zero. Let $p = 2$, then the constraint becomes $|\beta_1| + |\beta_2| \leq t$, which is a region shaped like a diamond, seen in figure (5.4). Since the region has sharp corners, the level curve with the lowest error will most likely hit the region at a corner, letting one of the parameter estimates be 0. This logic generalizes to higher dimensions p, where visualization is not possible.

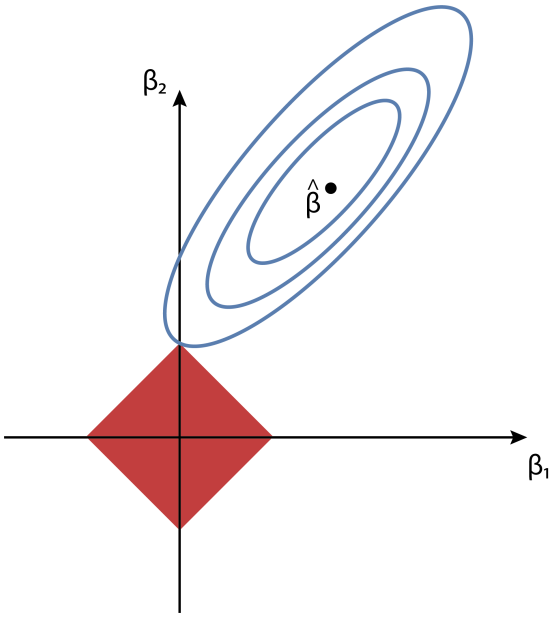

Figure 5.4: Contours of RSS and the constraint for the lasso.

5.3 Neural networks

Until now, focus has been on linear models for supervised learning, where the strength lies in the simplicity and the favourable computational properties. There is good conditions for inference, since one relatively easy can say something about the relationships between predictors and response from the model. But often, it can be hard for models of this simplicity to actually create a satisfactory fit, that works well for prediction. The linear model above could be expanded to include non-linearity in the predictors, but for this thesis the choice is to let non-linearity be reserved to the neural networks. The reason for this is that it would take careful assessment of the relationship with the response for each predictor, and due to the large number of predictors the linear model is kept simple. This could of course be a basis for further work, but for now neural networks will cover non-linearity, since these networks don't need the relationship assessments for each predictor in order to build the model.
Neural networks have become a very popular machine learning tool. Although the name suggests a connection to brain activity, there is no direct link, but the name originates from early models that were made for information processes in biological systems. The most simple of the kind, which will be dealt with here, is the feed forward neural network also known as the multilayer perceptron. The approach to the theoretical background of neural networks will in broad terms follow the approach of [Bishop, 2007].

5.3.1 Feed forward networks

As in all cases of supervised learning, the idea is still to start from the observation predictors and responses, and then fit a model describing the relationship. Often when dealing with neural networks, the terminology instead includes features and labels, but predictors and responses will be continued to be used here for coherence. When it comes to neural networks, the setup is that the model to be fitted can be written as the following:

$$\hat{Y} = \hat{f}(X) = \sum_{i=0}^{M} w_i \phi_i(X) \tag{5.51}$$

meaning that the output of the model is linear combination of non linear basis functions ϕ_i with coefficient weights w_i. An activation function around the linear combination of weighted basis functions is left out, which is typical in regression settings. This setup is common for many models, but in the case of neural networks, the basis functions ϕ_i are set to depend on parameters which are allowed to be adjusted along with the weights w_i during training. The relationship is then that the basis function is a non linear function on a linear combination of the inputs, where the coefficients as stated are adaptable. This gives non linearity in both predictors and parameters. The reason that models such as these go by the name of neural network becomes apparent when one tries to most easily visualize, how information is processed through the model. The figure (5.5) gives the simplest structure of such a model, which is a so-called one hidden layer neural network. This name comes from the fact that there is an input layer, which takes the predictors, and an output layer which has the response. Everything in between would be hidden layers and in this example there is only one.

This is a so called feed forward network because of how information transfers through to the output, also called forward propagation. Firstly, there is the input layer where the predictors along with a bias (given by index zero) make up the circles, which are called units. From there, a connection goes from each input to every of the M units in the next layer, which is the hidden layer. These connections are denoted by a weight $w_{ji}^{(1)}$, which are the first layer of adaptive weights mentioned earlier. ji signifies that this is weight from unit i in the input layer to unit j in the hidden layer. For each unit in the hidden layer there is an activation $a_j^{(1)}$, which takes

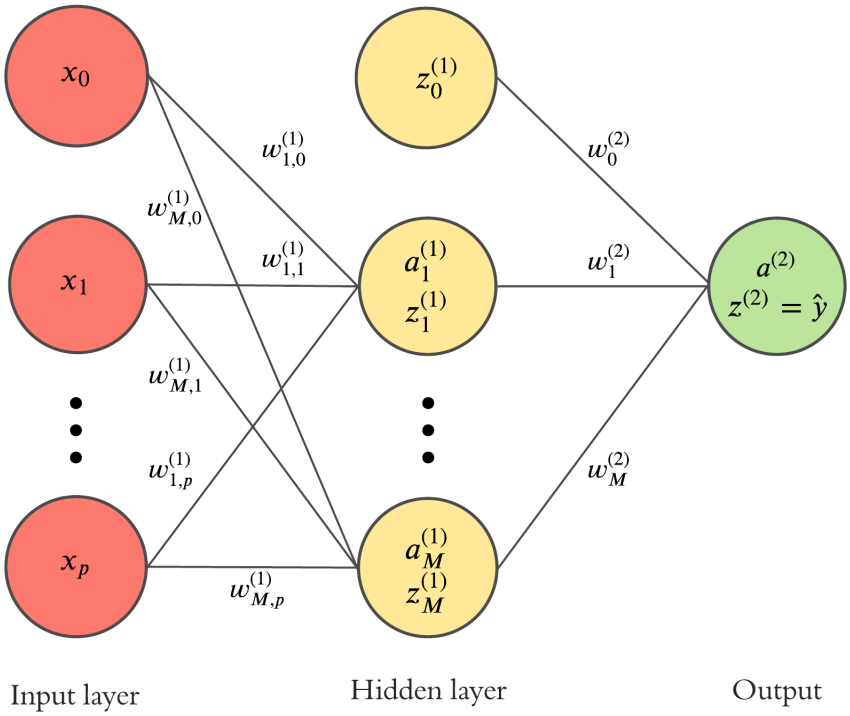

Figure 5.5: Single hidden layer feed forward neural network.

the information from the first input layer through a linear combination of all inputs with the weights. In every unit in the hidden layer this activation can be thought of as the input in the unit.

$$a_j^{(1)} = \sum_{i=0}^{p} w_{ji}^{(1)} x_i \tag{5.52}$$

Here the index goes from zero, so that the bias w_0 is included in the sum, which is handled analogously to the linear model earlier by letting $x_0 = 1$. For every unit in the hidden layer, the activation is then transformed by an activation function h, which is chosen to be a nonlinear continuous function. Previously many chose to use sigmoidal functions such as 'tanh', but presently the most popular is the rectified linear unit (ReLU) activation function, given by $h(x) = \max(0, x)$. This is because ReLU gets around the vanishing gradient problem, which slows down the learning process [Raschka and Mirjalili, 2019, p. 468], but this will not be dealt with further here. The activation then gives a sort of output in each unit given by

$$z_j^{(1)} = h(a_j^{(1)}) \tag{5.53}$$

where the superscript again signifies that it is in the first hidden layer. This way of processing information forward in the feed forward network can now continue in the same way to the next hidden layer, where $z_j^{(1)}$ will now act as input in the next connection. In this brief example there is only one hidden layer, so the next activation directly becomes the output of the network (since in the regression case the last activation function is the identity function).

$$a_1^{(2)} = \sum_{i=0}^{M} w_i^{(2)} z_i^{(1)} = \hat{y} \tag{5.54}$$

Now combine the stages of the process, and ultimately this gives an overall network function, where all weights and biases have been grouped in the vector $\mathbf{w}$:

$$\hat{y}(\mathbf{x}, \mathbf{w}) = \sum_{j=0}^{M} w_j^{(2)} h\left(\sum_{i=0}^{p} w_{ji}^{(1)} x_i\right) \tag{5.55}$$

And so, $\hat{y}(\mathbf{x}, \mathbf{w})$ is simply a nonlinear function of the inputs and the adjustable weights (and biases), which is also on the desired form (5.51). This is of course a very simplified example with only one hidden layer, but one can easily generalize to more layers, also resulting in this kind of nonlinear function.
Neural networks have been widely studied, and been found to be very general. They are said to be universal approximaters, since even a one hidden layer neural network with sufficiantly many hidden units can uniformly approximate any continuous function on a compact input domain to arbitrary accuracy [Bishop, 2007, p. 230].

5.3.2 Gradient descent

As in the case of the linear models, an error function is needed in the fitting process of approximating the true model function. The error function chosen to exemplify will again be based on the most common error term, RSS, which can be justified by a maximum likelihood argument, analogously to the earlier linear model example. That is of course if it is assumed (as will be done here) that the true model actually has the structure of the neural network function and the error term is additive Gaussian and independent, $\mathbf{y} \sim N(\hat{y}(\mathbf{x}, \mathbf{w}), \sigma_\epsilon^2 \mathbf{I})$. This leads to the following error function used for fitting, where $\mathbf{x}_i$ and y_i are the observed predictors and responses:

$$E(\mathbf{w}) = \frac{1}{2} \sum_{i=1}^{N} (\hat{y}(\mathbf{x}_i, \mathbf{w}) - y_i)^2 \tag{5.56}$$

Here multiplying by $\frac{1}{2}$ will have beneficial properties later on in the calculations. The objective is of course to find the weights and biases that minimize the error function when training the model, but since the network function $\hat{y}(\mathbf{x}, \mathbf{w})$ is nonlinear, the error function likewise becomes nonlinear and in general nonconvex. Therefore one will have to rely on local minima where the gradient vanishes, $\nabla E(\mathbf{w}) = 0$, when fitting the model, and reaching these through numerical methods. Here one can resort to the iterative numerical procedure known as gradient descent, which is the simplest approach to using the gradient information. Here the weight update comprises a small step in the direction of the negative gradient:

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E(\mathbf{w}^{(\tau)}) \tag{5.57}$$

where $\eta > 0$ is the learning rate. This technique means that for each iteration the weight vector is moved in the direction of the greatest rate of decrease of the error function, the steepest descent. The error function is defined with respect to a training set, so after each move, one has to re-evaluate the error function for the new weight vector using the entire training set. This kind of approach is called a batch method, since the entire data set is used in every step. Another approach is to cycle through the points in sequence (online learning), and then make the update based on the error for a single observation $\mathbf{x}_n$. This changes the iterative process to:

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E_n(\mathbf{w}^{(\tau)}), \quad \text{where} \quad E_n(\mathbf{w}^{(\tau)}) = \frac{1}{2}(\hat{y}(\mathbf{x}_n, \mathbf{w}^{(\tau)}) - y_i)^2 \tag{5.58}$$

Both for the batch method and this kind of sequential gradient descent it will often be necessary in order to find a sufficiently good minimum to run the algorithm multiple times with different starting points. But as it is obvious, the most important part is to be able to evaluate the gradient for every set of weights the algorithm goes through, which is the focus of the next section.

5.3.3 Error backpropagation

It is central to be able to evaluate the gradient of the error function in order to perform gradient descent. In this process it is also important that the calculations be feasible, meaning that an efficient method is available in order to have feasible computing time. Back propagation turns out to be one such method, which evaluates the gradient by sending information backwards in the feed forward network. To see how this process works, it is necessary first to review the setup of the neural network as described earlier.

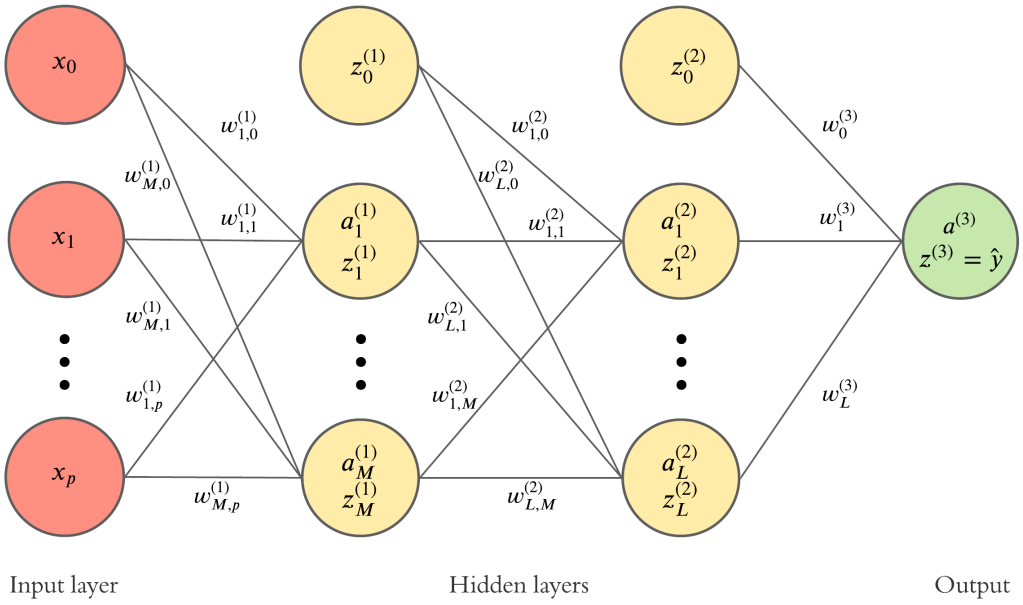

Figure 5.6: Two hidden layers feed forward neural network.

Figur (5.6) gives a bit more complexity than figure (5.5) by adding one extra hidden layer, which also makes it a bit more general as an example. Here one can see, how every unit in each of the two hidden layers and in the output layer have an activation $a_i^{(k)}$ and a transformation of this activation $z_i^{(k)}$. The hidden layers can all be described identically, and generally the following holds, with nonlinear differentiable function h:

$$a_j^{(k)} = \sum_i w_{ji}^{(k)} z_i^{(k-1)} \quad \text{and} \quad z_j^{(k)} = h(a_j^{(k)}) \tag{5.59}$$

which can again be understood as input a_j and output z_j within a particular unit. Here $k \in \{1, ..., K+1\}$, where K is the number of hidden layers. Layer $K+1$ is then the output layer, and in this special case the following holds:

$$a^{(K+1)} = \sum_i w_i^{(K+1)} z_i^{(K)} \quad \text{and} \quad z^{(K+1)} = h(a^{(K+1)}) = \hat{y} \tag{5.60}$$

Here there is no subscript in the last layer, since the output layer in this setup only has one unit, the output of the network function. The input layer is likewise a special case where one could say $k = 0$, but here $a_j^{(0)}$ does not exist, and $z_j^{(0)} = x_j$.
This is the structure of the feed forward network, in a framework which is a bit more general. Information flows forward from the inputs through the weights and biases of the hidden layer, and gives an output. This means that for every observation $\mathbf{x}_n$ the output and the error term can be calculated. The error function as stated earlier is a sum of individual observation errors, so therefore the gradient is likewise a sum of the individual observation error gradients.

$$E_n = \frac{1}{2}(\hat{y}(\mathbf{x}_n, \mathbf{w}) - y_n)^2 \tag{5.61}$$

So when looking at the error gradient it is sufficient to find a way to determine the individual error gradient. From here on, the subscript n will be omitted, but it is implicit, that it is the individual error gradient being determined.
The partial derivative of the error with regard to a weight/bias connecting to layer $k \in \{1, ..., K+1\}$ is given by:

$$\frac{\partial E}{\partial w_{ji}^{(k)}} = \frac{\partial E}{\partial a_j^{(k)}} \frac{\partial a_j^{(k)}}{\partial w_{ji}^{(k)}} = \frac{\partial E}{\partial a_j^{(k)}} z_i^{(k-1)} = \delta_j^{(k)} z_i^{(k-1)} \tag{5.62}$$

Where the two equalities follow directly from (5.59) and the chain rule for partial derivatives. This can be understood as that the error only depends on the given weight through the specific activation in which it is included. $z_i^{(k-1)}$ is already known through forward propagation, but the remaining partial derivative is not. This will now be defined as follows:

$$\delta_j^{(k)} \equiv \frac{\partial E}{\partial a_j^{(k)}} \tag{5.63}$$

For all weights, these $\delta_j^{(k)}$ are what need to be found in order to determine the gradient. An expression for this is first found for the output layer:

$$\delta^{(K+1)} = \frac{\partial}{\partial a^{(K+1)}} E = \frac{\partial}{\partial a^{(K+1)}} \frac{1}{2}(\hat{y} - y)^2 = \frac{\partial}{\partial a^{(K+1)}} \frac{1}{2}(a^{(K+1)} - y)^2 = a^{(K+1)} - y = \hat{y} - y \tag{5.64}$$

The last two equalities follow from the properties of the output layer (5.60), and the fact that the activation function in the output layer is taken to be the identity. It follows from the above expression, that it is fitting to call the δ's "errors". Now one can look at the errors for the hidden layer, meaning $k \in \{1, ..., K\}$, and here it is assumed that $\delta_j^{(k+1)}$ is known:

$$\delta_i^{(k)} = \frac{\partial E}{\partial a_i^{(k)}} = \sum_j \frac{\partial E}{\partial a_j^{(k+1)}} \frac{\partial a_j^{(k+1)}}{\partial a_i^{(k)}} = \sum_j \delta_j^{(k+1)} \frac{\partial a_j^{(k+1)}}{\partial a_i^{(k)}} \tag{5.65}$$

For the second equality above, the chain rule is again used, where the sum runs over all units j, which unit i sends connections to. Furthermore one can write:

$$\frac{\partial a_j^{(k+1)}}{\partial a_i^{(k)}} = \frac{\partial}{\partial a_i^{(k)}} \sum_i w_{ji}^{(k+1)} z_i^{(k)} = \frac{\partial}{\partial a_i^{(k)}} \sum_i w_{ji}^{(k+1)} h(a_i^{(k)}) = w_{ji}^{(k+1)} h'(a_i^{(k)}) \tag{5.66}$$

following the assumption that the activation function h is differentiable. This is not actually the case for the ReLU function in 0, but for practical purposes the method still works, since the argument very rarely is exactly 0. Now one can write the final expression for the "error", known as the back propagation formula:

$$\delta_i^{(k)} = \sum_j \delta_j^{(k+1)} \frac{\partial a_j^{(k+1)}}{\partial a_i^{(k)}} = \sum_j \delta_j^{(k+1)} w_{ji}^{(k+1)} h'(a_i^{(k)}) = h'(a_i^{(k)}) \sum_j \delta_j^{(k+1)} w_{ji}^{(k+1)} \tag{5.67}$$

At the time of evaluation of the gradient of the error function, all activations a_j, transformations z_j and weights w_{ji} are of course known through forward propagation. So the above expression reveals, that the "error" can be found through back propagation, meaning the information runs back through the network. If one knows the "errors" $\delta_j^{(k+1)}$ for one layer, then one can determine the "errors" for the layer right before, meaning $\delta_i^{(k)}$, and so forth. And since the last "error" is know (5.64), one can determine all "errors", and therefore all derivatives (5.62) to evaluate the gradient.
Error back propagation is a handy tool for neural networks, since it is computationally inexpensive. It determines the error function gradient, either for the single observation or the whole set in batch methods, and then enables methods like gradient descent in order to find local minima. One can show, that back propagation similarly works to find derivatives such as the Jacobian and Hessian matrices, which can be used in the evaluation of the stationary points. This will not be done here since it follows the lines of the above. But to sum up, it has now been shown that this kind of function, the neural network function, can be fitted efficiently to data through the methods of gradient descent and back propagation.

Chapter 6

Modelling

After having assembled a dataset of 29,830 observations of sales in the Copenhagen housing market in 2016-2020 consisting of 95 variables with characteristics of the particular housing, distance variables, economic indicator variables, and variables relating to the sale, it has finally come to the point where the set can be used for actual modelling. The set has been prepared in order to only include observations of full information, where the information consists of only the most relevant aspects around housing price prediction, which will be the objective of this modelling part of the thesis. The theoretical section has functioned as preparation for the upcoming work, and has listed the key aspects, which lay the ground for how models will be constructed and assessed. This includes the probability theoretic foundation of the statistics that will be performed, different measures of quality of fit, and of course a general overview of the models and their functionalities: Ordinary least squares linear regression, Lasso linear regression and finally a neural network. As has been described, the two first are similar in their setup, with the exception of a regularization term, while the neural network is quite different due to the functional form and the training. But for all of the models, preprocessing is beneficial for optimal performance, which among other things is why correlations in the variables were assessed in order to weed out the worst collinearity, and categorical variables have been encoded. To improve the performance further, scaling of the predictors will furthermore be done, since it improves the optimization algorithms in the models [Raschka and Mirjalili, 2019, p. 124]. To get the predictors somewhat on the same scale, standardization is performed following the formula:

$$x_{std}^{(i)} = \frac{x^{(i)} - \mu_x}{\sigma_x}$$

for all observations $i \in \{1, ..., N\}$. Here $x_{std}^{(i)}$ represents a standardized predictor for observation i, while μ_x and σ_x are the sample mean and sample standard deviation for the particular predictor column. This further has the benefit, that it centers the predictor columns, giving them mean and variance as a standard Gaussian distribution, which in particular helps weight learning for neural networks. This is the final step of preprocessing leading to modelling.
Below, the three models will be assessed, first and foremost in order to determine which model is superior in predicting housing prices. In order to assess the performance, the data set must be split in a training set and a test set. While the function of the training set obviously is to fit the model, the point of the test set is as mentioned earlier to estimate the expected test error - an estimate of the general quality of fit. So the data is split in two, where the test set is given 25% of the observations, which gives a training set of 22372 observations and test set of 7458 observations. The exact same split will be used for all models unless else has been indicated, in order to make a proper comparison.
It should be noted that open source machine learning libraries have been used to implement the models in Python: The Scikit-learn library [Pedregosa et al., 2011] has been used for the

linear models while the Keras interface for the TensorFlow library [Chollet et al., 2015] has been utilized for the neural network. These provide ready to use algorithms that can be suited for the modelling purpose at hand, which makes writing the whole program from scratch unnecessary. The python code behind the model making is not prioritized in this modelling section, but all code will as in previous sections be included on GitHub[1].

6.1 OLS linear model

A model is created by using a "pipeline"-object from the Scikit-learn library, which includes a standardization scaler and a modelling object `LinearRegression()`, which implements OLS regression. The model is then fitted on the training set, and response variables (housing prices) are predicted from the train and test set in order to be able to assess performance.

6.1.1 Diagnostics

Since one can not plot response against predictor in multiple linear regression, one must look at the residuals, which are found by subtracting the true response values from the predicted values. The residuals for both training and test data are plotted left in figure (6.1).

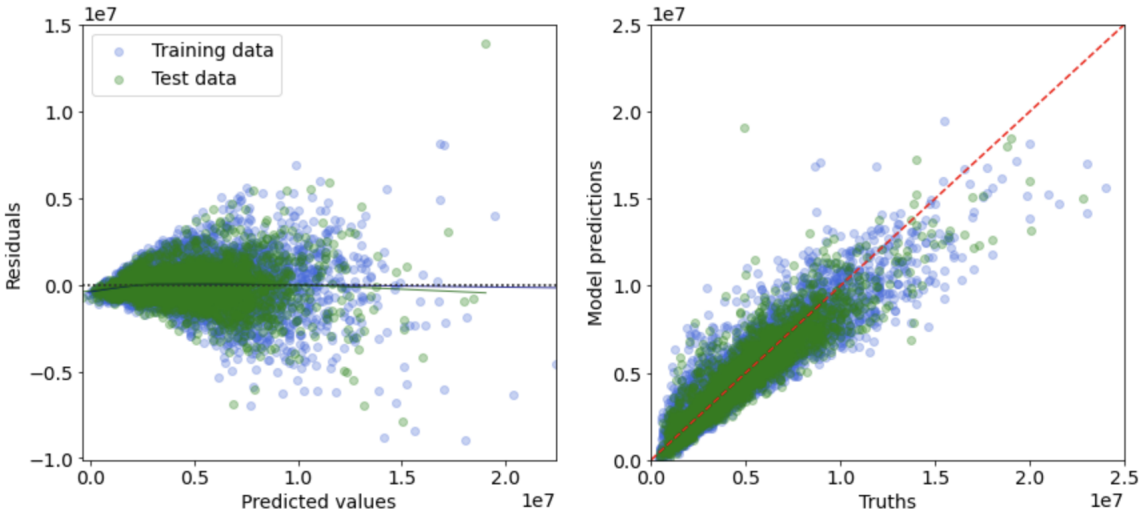

Figure 6.1: OLS linear model residuals (left) and predicted response plotted against true response (right).

Here it becomes apparent, that there is non-constant variance in the error, or heteroscedasticity, which is not in line with the model assumptions (5.33). While it might not influence the predictive power of the model, it does render hypothesis tests useless, because these rely on the assumption of constant variance.
The pattern is also visible in the right figure in (6.1), where one can see the high variance for higher housing prices, which means the model has a harder time pricing these correctly.
This also influences the distribution of the residuals, where the qq-plots of figure (6.2) shows how very heavy tails (and outliers) signifies a bad fit to the data. So the model doesn't fit the data well, since the model assumption of normally distributed errors is not fulfilled.

One solution to the above problem is to transform the response, in order to force the data to fit the linear model setup. Here the response is transformed using a natural-log transform, which will ensure a greater amount of shrinkage in the larger responses. Figure (6.3) shows the qq-plots after the transformation of the response, and there is a definite improvement, since the

[1] `https://github.com/fredbuscma/Frederik_Madsen_Master_Thesis`

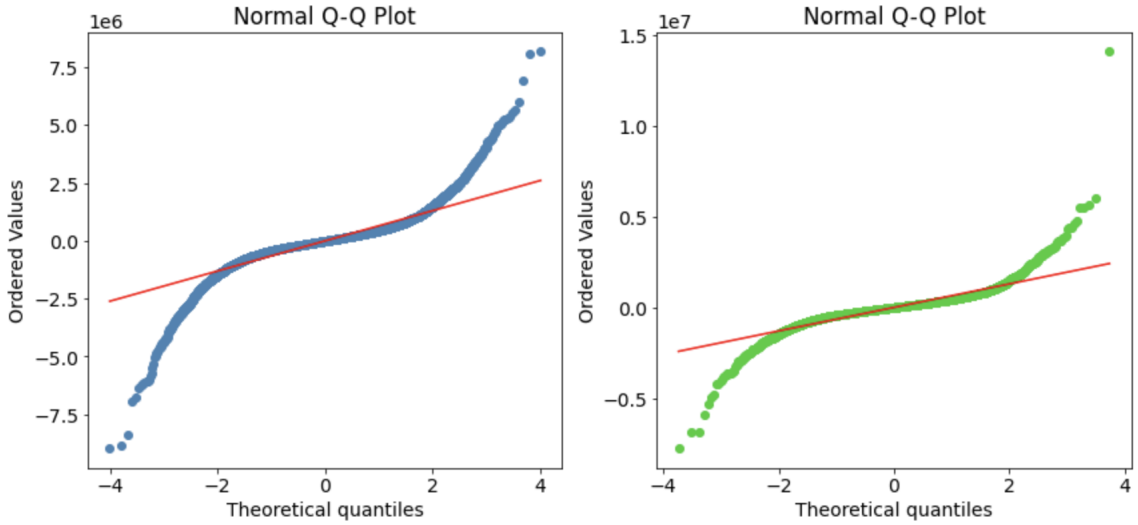

Figure 6.2: Q-Q plots for residuals in training (left) and test data (right).

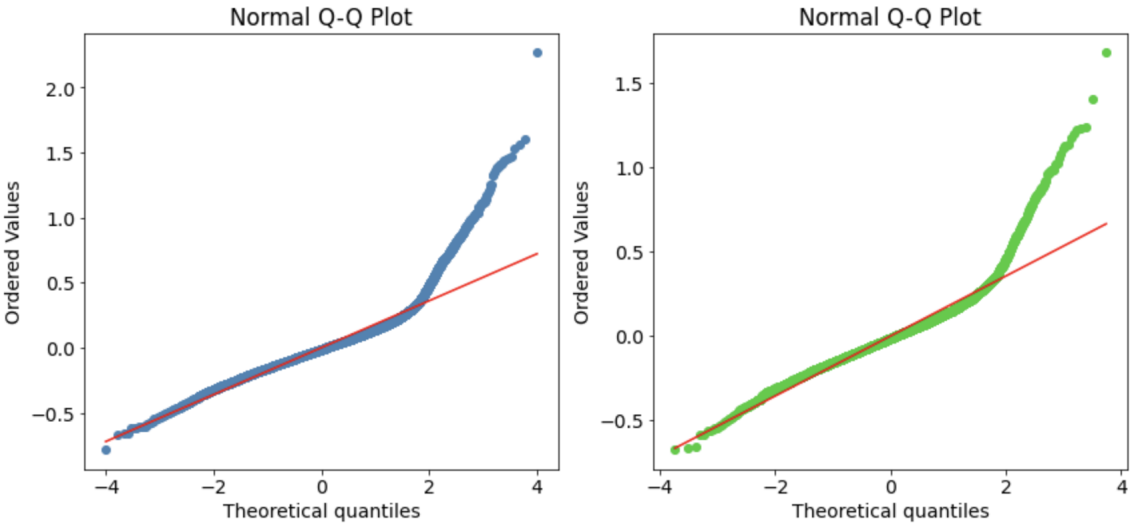

Figure 6.3: Q-Q plots for residuals in training (left) and test data (right) after log-transform of response.

lower tail has disappeared. There is still a heavy tail for the larger residual values.
This larger tail in the high residual values can also be seen in figure (6.4), that plots the residuals. The heteroscedasticity has been alleviated as intended, but now a non linear pattern is visible in the residuals and in the predicted responses plotted against the true responses. This is most likely what causes the heavy upper tail in the residuals, since one can see that the curvature forces more residuals to be positive. So it is already clear that the model has a high bias, since it is unable to catch the non linearity present in the data. One could introduce non linearity in the predictors and still use the linear model, but this will not be done here, since it is not clear which variables should be transformed.
From here on out in the thesis, the log-price will take the place as the response. The reason for this, is that the Lasso is known to work better when the error distribution is close to normal [Pedregosa et al., 2011], and also for comparability between the models.

Another thing which is interesting to look at, is how much relative significance the different predictors have in the model. The most obvious is of course to take a look at the size of the coefficients, or estimated parameters, as they are in the model, and this has been

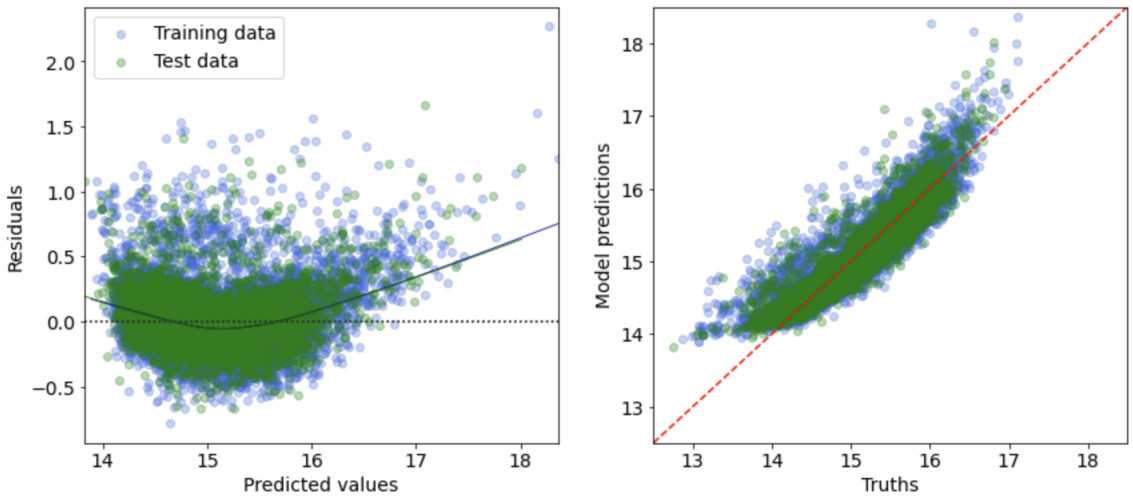

Figure 6.4: OLS linear model residuals (left) and predicted response plotted against true response (right) after log-transform of the response.

done in figure (6.5), where the highest absolute coefficients are presented. Here one should note that `areaWeighted_bd` by far seems to be the most significant predictor followed by `propertyValuation_b` and `numberOfRooms_bd`. This is not surprising, considering how positively correlated these predictors were found to be with the response in figure (4.12). Other than these three variables, there is not a clear analogous relationship to the correlations. Some distance variables have been given more influence in the model, `priceIndex_s` has come to take a prominent position, and `areaBasement_bd` has actualy come to have a negative coefficient, even though it was found earlier that it is positively correlated with the response.

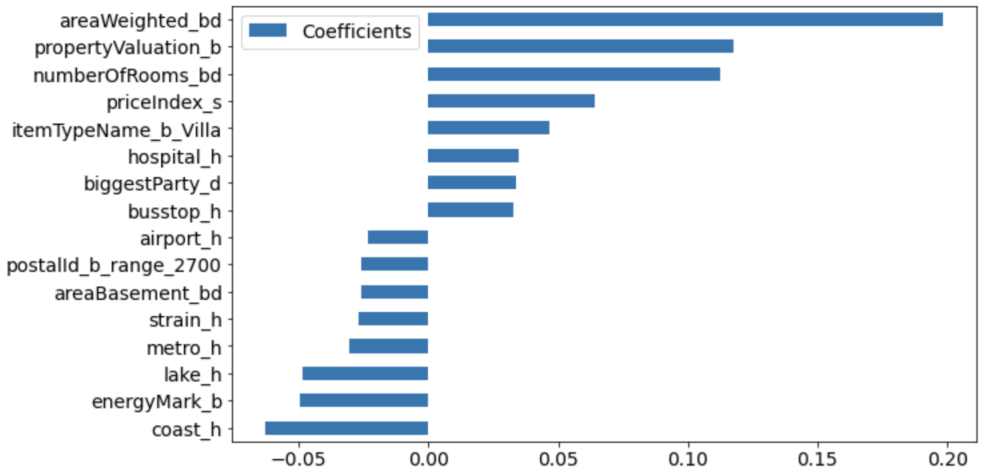

Figure 6.5: OLS most important predictors by highest absolute estimated parameter size.

In the other end of the spectrum, one can take a look at the predictors with the smallest coefficients, which have little impact in the model. Table (6.1) lists the variables with the 10 smallest coefficients in absolute terms, along with the p-values for the hypothesies that $\beta_i = 0$. It is clear, that for all of these variables, the p-value is high above the level of 5%, which means that the 0-hypothesies can't be rejected, meaning we can't assume any association between the predictors and the response. It turns out, that this can be said about 33 of the 92 variables in the model. This of course relies on assumptions which are not fully satisfied given the non-linearity, so it is uncertain how much these values can be relied upon. And furthermore, the p-value for the Z-scores say something about association of predictor and response, when the other

predictors aren't taken into consideration. But it still is noteworthy to have these very high p-values for many variables, since it means that some variables could potentially be dropped without the model changing much.

	Coefficients_abs	p values
rebuildYear_bd	0.000040	0.9942
postalId_b_range_2100	0.000183	0.9799
outerwall_d_Metal	0.000308	0.8673
outerwall_d_Glas	0.000474	0.7993
postalId_b_range_2150	0.000517	0.8262
roof_d_Fibercement uden asbest	0.000815	0.5977
OMXC20_s	0.000843	0.6131
roof_d_Glas	0.001015	0.6126
heating_d_Ovn til fast og flydende brændsel	0.001138	0.6462
postalId_b_range_1450-1500	0.001359	0.3875

Table 6.1: OLS least important predictors by highest absolute estimated parameter size, p-value included.

6.1.2 Assessment

For assessing the model, different measures of fit have been mentioned in the theoretical section. These included the mean squared error (MSE) (5.24), giving an average of squared error terms, and R^2 (5.48), which was a measure of how much of the variability that could be explained by the model. We furthermore introduce the mean absolute error (MAE), which takes the absolute error rather than squared error, and therefore is less sensitive to outliers.

$$\text{MAE} = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{f}(x_i)| \tag{6.1}$$

Figure (6.6) sketches a learning curve for how well the model performs, dependent on how many training examples it is being introduced to. Here it is very clear that the test R^2 measure improves the larger the training set becomes, and it converges to a level closer to the training R^2. Here it can be noted how the model in the beginning has the problem of overfitting, where it performs very well on the training data that it has been fitted to, but performs much worse on the validation set. The overfitting is alleviated as the training set grows.

Table (6.2) sums up the performance of the OLS linear model, and it looks promising, if one focuses on the R^2 measures. But these numbers are technical and are meant for comparison, since they don't make much sense by themselves. MSE and MAE is hard to conclude anything from by themselves, also because of the fact that these are made from the log-transformed response. It should be noted overall, that the results indicate the the mentioned overfitting is completely gone, since the model actually performs better on the test set than on the training set, which is quite surprising. There still might be a bias due to the non linearity of the data which drags the model down, but this will be more clear when comparing with other models.

	MSE, train	MAE, train	R^2, train	MSE, test	MAE, test	R^2, test
OLS	0.03586	0.132177	0.85974	0.034578	0.131064	0.863186

Table 6.2: Performance of the OLS linear model.

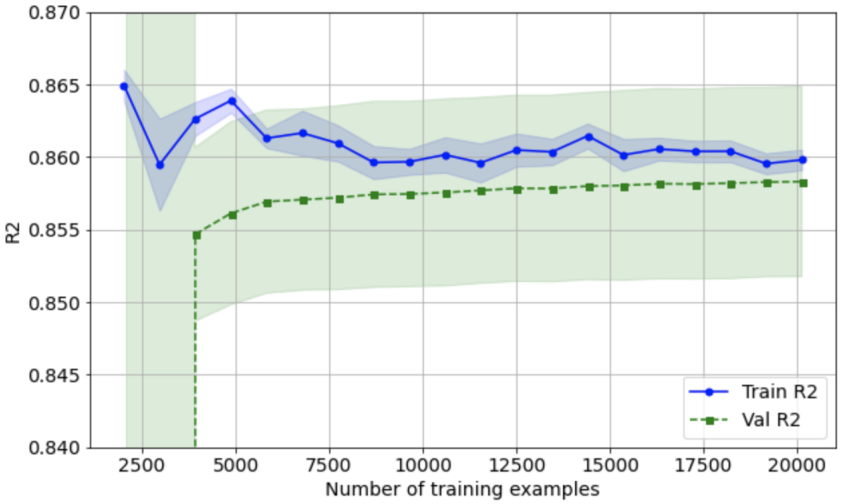

Figure 6.6: Learning curves for the OLS linear model.

To give an impression of how well the model does in terms of actual prices, table (6.3) sums up the mean and median relative deviation of the predicted housing prices from the true housing prices for the test set. The mean deviation is higher than the median deviation, which indicates that there is outliers with big deviations from the true housing price. The last column states, that just about 70 % of the test observations were predicted within 15 % of the actual price. Overall this is quite a satisfactory result for the first model.

	Mean deviation %	Median deviation %	Within 15 %
OLS	13.85	9.77	69.33

Table 6.3: Deviation of predicted prices from actual prices in test data for OLS linear model.

6.2 Lasso linear model

As it was described in the theoretical section, the Lasso model is a linear model, where in stead of just the OLS cost function, a penalty term is introduced for regularization. A model is created by using a standardization scaler and a modelling object `LassoCV()`, where 10-fold cross validation is chosen in order to choose an optimal α for the model. This model uses coordinate descent as the algorithm to fit the coefficients [Pedregosa et al., 2011]. Through the cross validation, the model is repeatedly fitted and validated in order to determine an optimal hyper-parameter α, and then finally fitted using this α.

6.2.1 Diagnostics

Figure (6.7) shows validation curves for the lasso model with the measures MSE and R^2. In the figure to the left, every colored line signifies one of the 10 validation set MSE's for each training-validation split, and with each partition the model is fitted over the range of possible values of α. The black line is then the average of these validation curves and is thereby the cross validation estimate for each alpha. In the figure to the right, only the cross validation average curve is included, but in this case also the R^2 measure for the training set. The validation curve is very close to the training curve for all α's, which shows that this model does not seem to experience overfitting either. But in both cases, the key take away is that the model performance on the validation set, does not change much for small values of α, until higher values weakens

the performance. The optimal value found is $\alpha = 0.0004$, which will then be used in fitting the model in order to evaluate on the test set. It is clearly a small value, so effectively one would expect, that not much regularization is optimal, meaning we have a model which is close to the OLS linear model. Presumably, the little need for regularization means that the 92 predictors were not too high a number to begin with, leading to overfitting, but this is to be expected since the number of predictors is small compared to the number of observations.

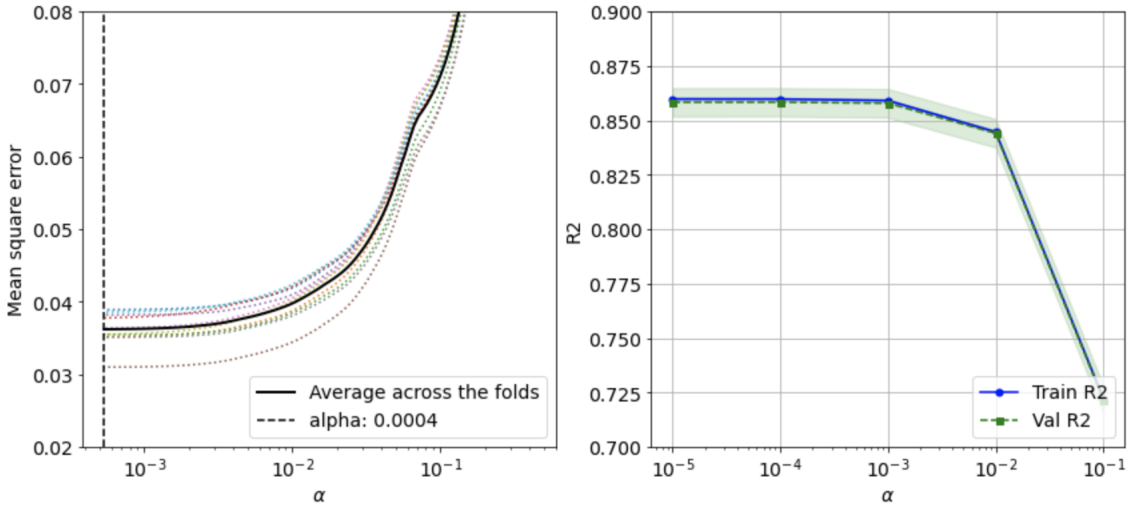

Figure 6.7: Validation curves for the Lasso linear model.

Figure (6.8) shows how the lasso, has not created much change in terms of which variables are more important. One can recognize the same predictors, and their parameter estimates are quite similar in relative sizes.

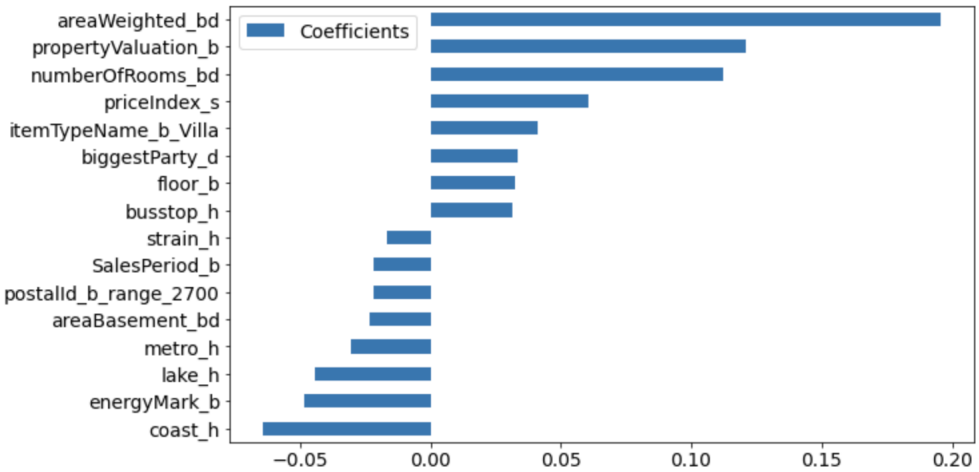

Figure 6.8: Lasso most important predictors by highest absolute estimated parameter size.

What is more interesting is seen in table (6.4), where the absolute value of the coefficients is given for the same predictors as in table (6.1). 5 of these coefficients have shrunk to zero in the lasso model, and this is also the case for 5 more variables. This is how lasso regression works as a shrinkage method compared to ridge regression as described in the theoretical section. But as was found earlier, the OLS linear model didn't show signs of overfitting, which is what the lasso is particulary good at alleviating. So this is why not many coefficients were set to 0, and the few that were already had very small coefficients in the OLS linear model. In short, there seems to be not much use for the lasso when trying to get a better performing linear model, cause it doesn't make much difference.

Figure (6.9) shows that the lasso performance gives virtually the same image as the OLS linear model when it comes to predictive power. As in the earlier model, the non-linearity is still clearly visible in the residuals, which is to be expected, since the lasso model at hand is not designed to handle the non-linearity in the data.

	OLS	Lasso
rebuildYear_bd	0.000040	0.000000
postalId_b_range_2100	0.000183	0.000000
outerwall_d_Metal	0.000308	0.000159
outerwall_d_Glas	0.000474	0.000000
postalId_b_range_2150	0.000517	0.000000
roof_d_Fibercement uden asbest	0.000815	0.001216
OMXC20_s	0.000843	0.000000
roof_d_Glas	0.001015	0.000575
heating_d_Ovn til fast og flydende brændsel	0.001138	0.002160
postalId_b_range_1450-1500	0.001359	0.000136

Table 6.4: OLS least important predictors by highest absolute estimated parameter size compared to lasso parameter sizes.

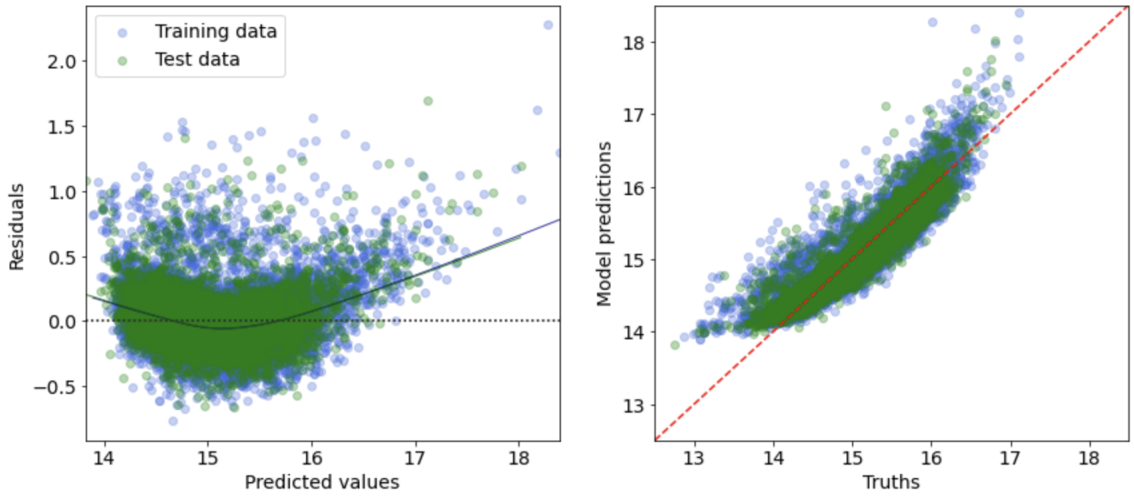

Figure 6.9: Lasso linear model residuals (left) and predicted response plotted against true response (right).

6.2.2 Assessment

The performance of the lasso linear model summed up by table (6.5) is virtually the same as for the OLS linear model. The scores are marginally worse for the train set and marginally better for the test set, and the peculiar situation where the test model performs better on the test set is repeated. The almost identical results is of course because the lasso didn't have much effect as explained earlier.

Table (6.6) tells the same story, where the relative deviations of the predicted prices from the true prices are only marginally different from the results in the OLS linear model. The median deviation is again lower than the mean due to outliers, and again just about 70 % of the test observations were predicted within 15 % of the actual price, which is still a satisfactory result.

	MSE, train	MAE, train	R^2, train	MSE, test	MAE, test	R^2, test
Lasso	0.035914	0.13242	0.85953	0.034576	0.131123	0.863192

Table 6.5: Performance of the Lasso linear model.

	Mean deviation %	Median deviation %	Within 15 %
Lasso	13.86	9.77	69.4

Table 6.6: Deviation of predicted prices from actual prices in test data for Lasso linear model.

6.3 Neural network model

Finally, the time has come to build a neural network model for housing prices in Copenhagen, which means that Keras for TensorFlow will be used to build the model. A model that only includes the predictor `areaWeighted_bd` is first presented in order to illustrate how the neural network model is able to accommodate the non-linearity in the data through the non-linear ReLU activation functions. So this model will only have one predictor, meaning one node in the input layer. To implement this a `keras.Sequential()` object is created, which specifies; a scaler for standardization of the predictor, how many hidden layers are needed and that ReLU activation functions are used. Furthermore the Adam optimizer algorithm is chosen, which performs stochastic gradient descent, or on-line learning [Chollet et al., 2015]. This means that the network will perform the update by gradient descent after each single observation has been fed forward in the network in stead of using batches when updating. This is of course computationally very heavy, which is why TensorFlow is useful, since it utilizes both CPU's and GPU's. The number of epochs, which is how many times the model goes through the whole training set, is set to 200, and 20 % of the observations is split from the training data for validation. MAE is chosen to be the loss function, since it is found to give more stability in the training and less dependency on outliers.
In this setup of the one predictor model, two cases are created. The first case has no hidden layers, meaning one weight and one bias to be fitted, which makes it a normal linear regression model. There are no non-linear activation functions, since the single output note takes the identity function as activation. The other case is a network with two hidden layers of 64 nodes each using ReLU activation functions, and this setup is chosen since it is found to give stable results. The number of parameters to be fitted is for each layer (except the input layer) given by the number of units in the previous layer, n_{-1}, and the number of units in the present layer, n_0, by the formula: $n_{-1} \times n_0 + n_0$. So in this case it gives $(1 \times 64 + 64) + (64 \times 64 + 64) + (64 \times 1 + 1) = 4,353$ parameters to be trained in the network.
After this example for illustration, the same setup was used for the whole data set, including all predictors in the input layer. For the case without hidden layers this again means linear regression with 92 weights and one bias, and for the deep neural network the number of trainable parameters has now increased to 10,177. The number of epochs is raised to 400 for the full data deep neural network to make sure a stable solution is reached, since this is the final model.
All four models can then be fitted to the training data and evaluated on the validation set. Afterwards it will be ready for assessment of its prediction capabilities.

6.3.1 Diagnostics

For the first cases, where `areaWeighted_bd` is the sole predictor in the model, the training progress is visualized in figure (6.10). Here one can see, that in both non-deep and deep neural network model, the model reaches a somewhat stable level of error right away in the training

process - and this level is not bad compared to MAE of the earlier models presented, considering that only one predictor is included. This is of course because `areaWeighted_bd` was the most significant variable in predicting log housing prices. Both cases show that the validation error oscillates a bit around the training error, but on such a small scale that no overfitting seems to be the case.

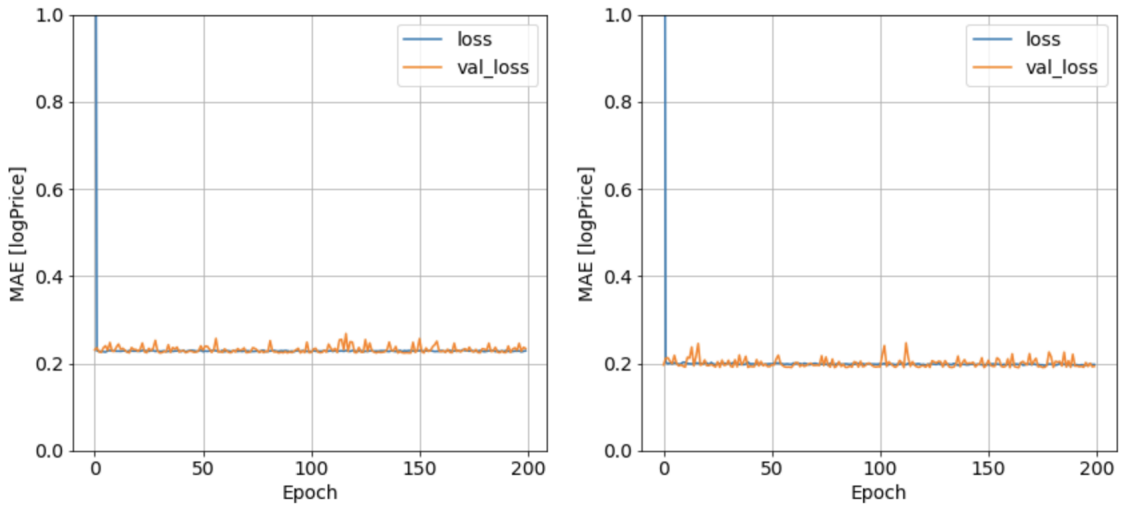

Figure 6.10: Training progress for single predictor neural networks, with no hidden layer (left) and two hidden layers (right).

Most noteworthy in figure (6.10) of the learning progress, is that the deep learning model reaches a lower level of error, than the model without hidden layers. The reason why becomes clear if one looks at figure (6.11), where a line visualizing the model prediction has been drawn over the training data that the models were fitted on. Here one can clearly see, that introducing non-linearity in the deep network enables the model to accommodate the non-linearity in the data, and therefore fit more closely to the data and achieve a lower error. Simply stated, introducing the hidden layers has gotten rid of the major bias that the linear model had.

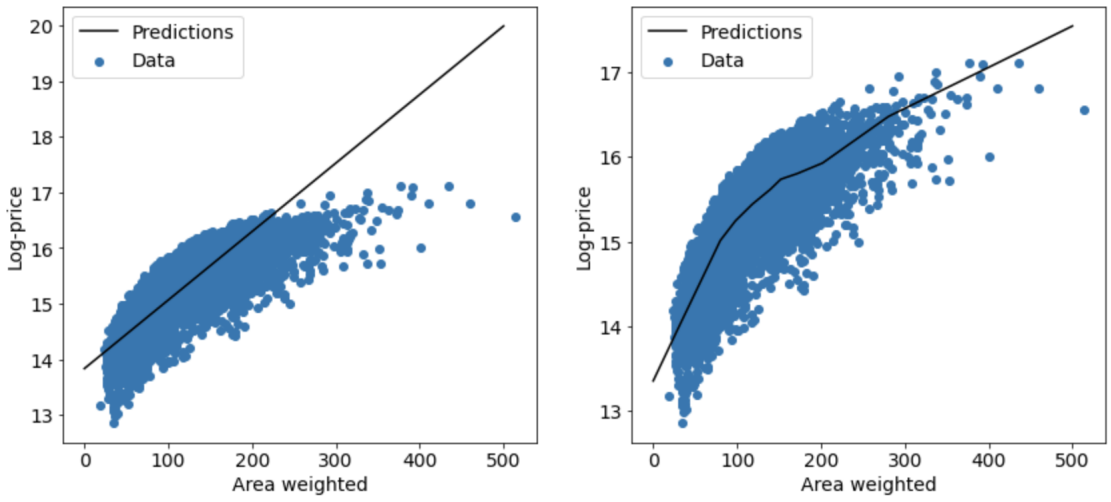

Figure 6.11: Log-price plotted against weighted area for true values and predictions, with no hidden layer (left) and two hidden layers (right).

Returning to the case where all predictors are included, the learning process is illustrated in figure (6.12), and here the graphs are not so similar. To start out with the model with no hidden layer, the model is seen to be very unstable. This is surprising since the model essentially

is linear regression again, but it seems the fitting process is unable to reach the results of the earlier linear regression model using the Scikit-learn library. This can most likely be attributed to the different optimization algorithms. The deep network on the other hand shows promising results. From the learning process it can be seen how the model initially learns slower than in the single predictor case, but it then quickly reaches a very low level of error, which is by far superior compared to the other neural network models. One can note how the validation error and the training error slightly diverge at the end of training, indicating some overfitting - but this is only in the slightest, and therefore not a worry.

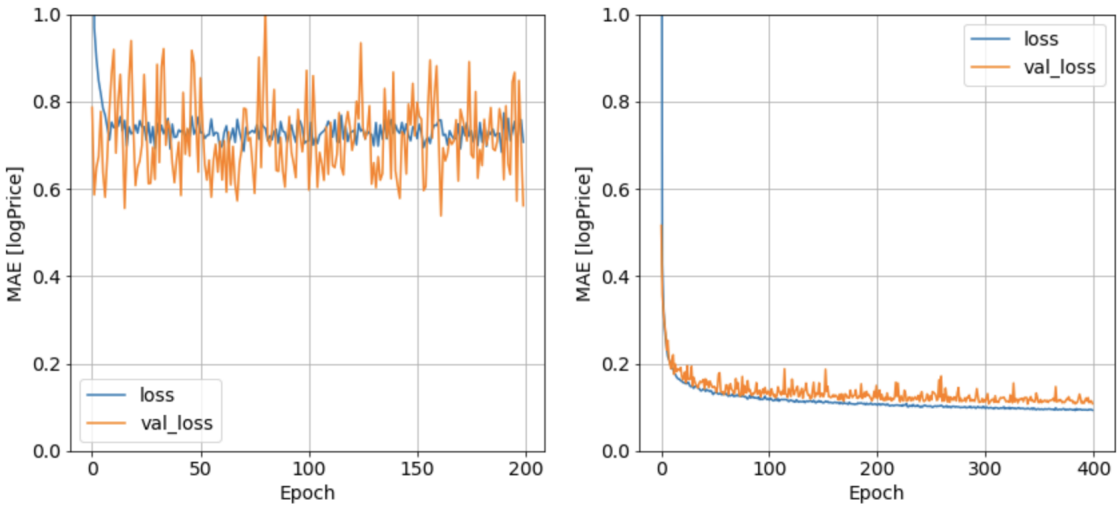

Figure 6.12: Training progress for full predictor neural networks, with no hidden layer (left) and two hidden layers (right).

Figure (6.13) shows the residuals and predicted response plotted against true response for the full deep neural network model, and it is clear how the errors show much less curvature compared to the linear models. So it seems, as expected, that the full neural network model really is able to accommodate the bulk of the non-linearity in the data, which most likely has resulted in a better fit. There still seems to be a tendency of a bigger tail for higher residual values, meaning that the model still has a problem with some observations where the price is predicted too high. And there also seems to be outliers when predicting, but it doesn't change the overall picture of a more satisfactory fit.

6.3.2 Assessment

Table (6.7) shows the performance of the deep neural network model, and in all measures one can see an improvement in comparison to the linear models from earlier. The performance on the test set is slightly worse than on the training set, and this is normal, and not necessarily a sign of overfitting. But it is clear that the neural network model has alleviated some of the bias, which was present for the linear models.

	MSE, train	MAE, train	R^2, train	MSE, test	MAE, test	R^2, test
NN	0.020428	0.088891	0.920101	0.027891	0.103964	0.889642

Table 6.7: Performance of the full deep neural network model.

The performance of the neural network model can also be seen in table (6.8) to be superior in the deviation of predicted prices from actual prices, where most noteworthy that about 80 % of sales prices in the test set now can be predicted within an error of 15 %.

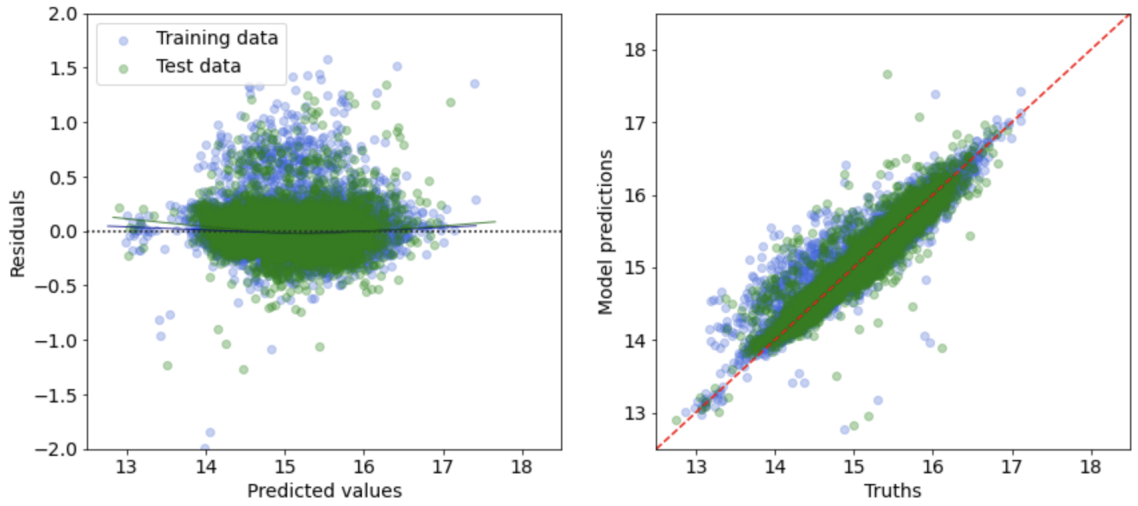

Figure 6.13: Deep neural network model residuals (left) and predicted response plotted against true response (right).

	Mean deviation %	Median deviation %	Within 15 %
NN	11.08	7.16	80.73

Table 6.8: Deviation of predicted prices from actual prices in test data for full deep neural network model.

6.4 Other predictors

6.4.1 Asking price

After having assessed three models for predicting housing prices in Copenhagen, it is worth looking at other possible ways of predicting for comparison. Firstly, the dataset is expanded by including the variable `paymentCash_b`, which is the asking price. This has not been used as a predictor previously, since the asking price very often is close to the actual sales price, and arguably this would make the modelling obsolete. Making a model is an attempt to mimic the pricing process, and the asking price can be said to be the result of this pricing process. But nonetheless, figure (6.14) shows the largest absolute parameter estimate where `paymentCash_b` is included. As before, the model still doesn't catch the non-linear aspect of the data as can be seen in figure (6.15), but the fit seems to be better than without `paymentCash_b`. So including the asking price does improve the model, and `paymentCash_b` can not surprisingly be seen to take the spot as most important predictor.

Tables (6.9) and (6.10) sum up the performance when including asking price in the OLS linear model, and not surprisingly there is improvement on all fronts. Impressively, the model now predicts 77 % of the test set within 15 % of the true price, which is more than 7 %-points better than without this predictor.

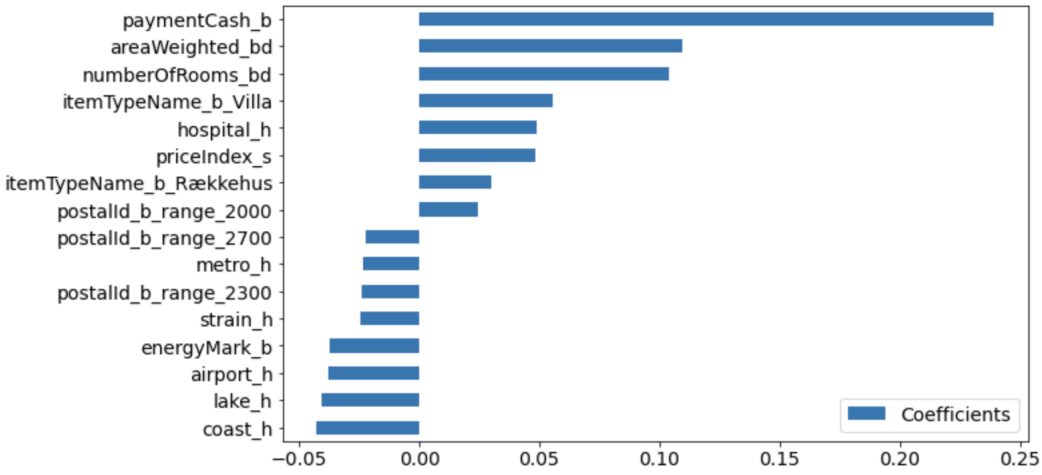

Figure 6.14: OLS with asking price most important predictors by highest absolute estimated parameter size.

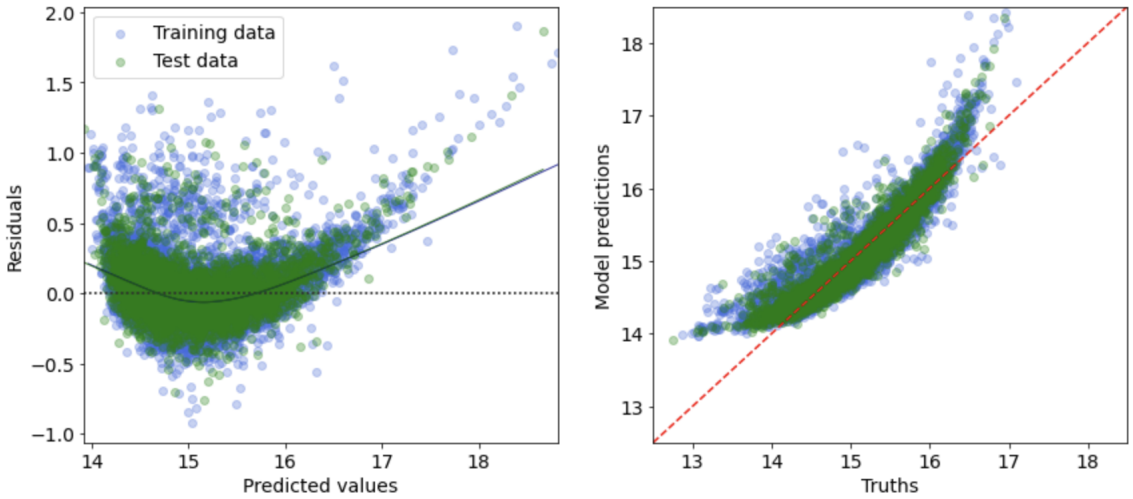

Figure 6.15: OLS linear model with asking price residuals (left) and predicted response plotted against true response (right).

	MSE, train	MAE, train	R^2, train	MSE, test	MAE, test	R^2, test
OLS Cash	0.028795	0.113101	0.887376	0.02686	0.112086	0.893722

Table 6.9: Performance of the OLS linear model with asking price.

	Mean deviation %	Median deviation %	Within 15 %
OLS Cash	11.88	8.17	77.45

Table 6.10: Deviation of predicted prices from actual prices in test data for OLS linear model with asking price.

Next up, the neural network setup is reused, where asking price once again is included as predictor, which gives one more node in the input layer. In figure (6.16) the training progress can be seen, which is very similar to the neural network without this extra predictor. But as one would expect, the error converges to a level, which is lower than the previous neural network model. The minor overfitting that one could see in figure (6.12) has also been alleviated. Also in figure (6.16), the same conclusions as earlier can be reached, since the non-linearity in the data visibly is accommodated in the neural network model.

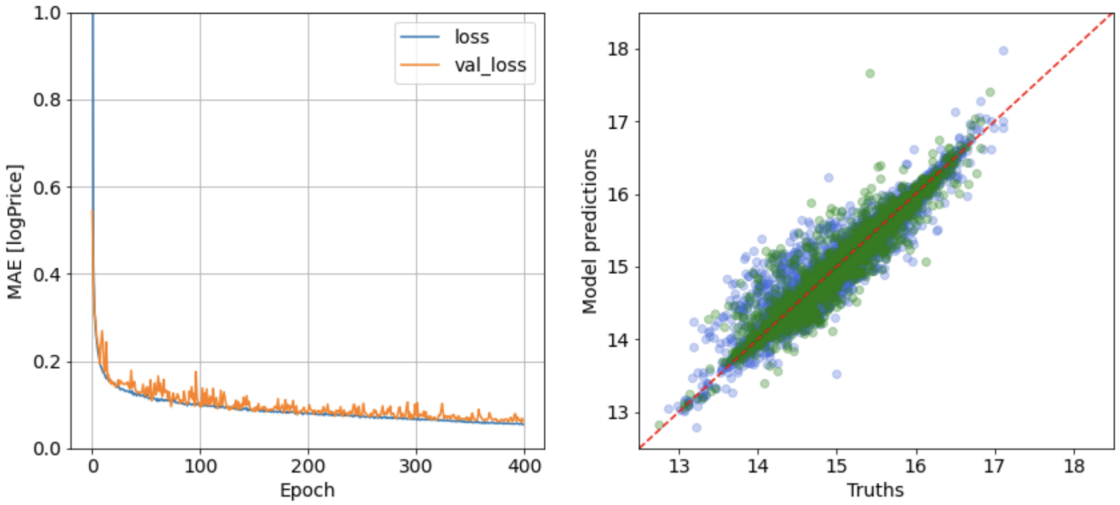

Figure 6.16: Training progress for full deep neural network with asking price (left), and predicted response plotted against true response (right).

Tables (6.11) and (6.12) show yet again an improvement, and also in this case one can see how the superiority of the neural network for this data was repeated. 92 % of the test data can be predicted within 15 %, which is almost 12 %-points better than the neural network without the extra predictor.

	MSE, train	MAE, train	R^2, train	MSE, test	MAE, test	R^2, test
NN Cash	0.011854	0.053992	0.953636	0.014259	0.060499	0.94358

Table 6.11: Performance of the full deep neural network model with asking price.

	Mean deviation %	Median deviation %	Within 15 %
NN Cash	6.35	3.23	92.12

Table 6.12: Deviation of predicted prices from actual prices in test data for full deep neural network model with asking price.

So including the asking price has definitively improved the performance of both the OLS linear model and the neural network model. As mentioned earlier, using this variable as predictor is counter intuitive, since the asking price is more like a result of the pricing process. But now that it has been included in the models for illustration, the question is, if the performance is better, if simply predicting the price to be the same as the asking price. Figure (6.17) shows, that using the asking price for prediction accommodates the non-linearity, and in general the residuals are spread evenly around the mean. It could be expected that a big amount of the data would have positive residuals, since real estate agents have an incentive to ask a higher price than what ends being the actual price, but there is not much evidence of this in figure (6.17).

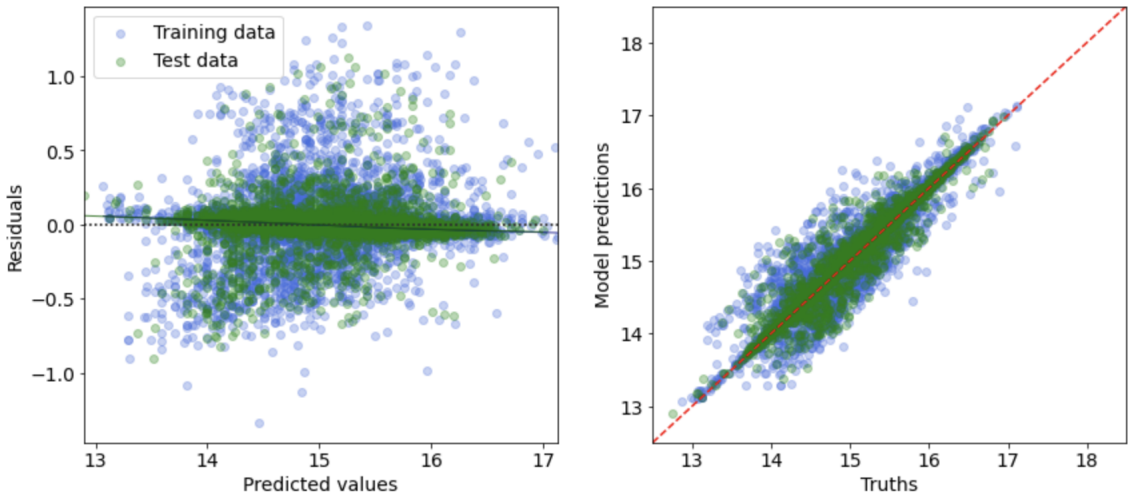

Figure 6.17: Asking price for prediction, residuals (left) and predicted response plotted against true response (right).

Tables (6.13) and (6.14) gives the predictive performance of using the asking price `paymentCash_b`, and it ends up being an improvement in all measures - it is even superior to the neural network including asking price. So one could argue that the best at predicting housing prices are the real estate agents, even though they themselves have great influence on the price level to begin with.

	MSE, train	MAE, train	R^2, train	MSE, test	MAE, test	R^2, test
Cash	0.015461	0.052441	0.939527	0.013447	0.050565	0.946794

Table 6.13: Performance of using asking price for prediction.

	Mean deviation %	Median deviation %	Within 15 %
Cash	5.3	2.41	92.89

Table 6.14: Deviation of predicted prices from actual prices in test data when using asking price for prediction.

6.4.2 AVM price

Analytical valuation model (AVM)[2] is a real estate pricing model for housing in Denmark, which is developed by Geomatic, a nordic data and analysis firm specializing in data science and data management. Geomatic lets the price estimates calculated by AVM be publicly available, and is therefore part of the data drawn from DinGeo. Thereby it is possible to evaluate the performance of AVM on the data at hand, and thereby compare the state of the art pricing model to the models created for this thesis. One problem, is that when pulling the data from DinGeo, not all housing had an AVM price, meaning there is missing data to be accounted for. The number of observations in the training and test set used for the models above is 22,371 and 7,458, while deleting observations without AVM price brings the numbers down to 19,202 and 6,400. So the number of dropped data is not devastating, and arguably the AVM price can still be assessed on the remaining data.
Figure (6.18) shows how well the model performs on the data when predicting housing prices by the AVM price. Clearly the AVM also accommodates the non-linearity in the data, since there seems to be no curvature in the residuals. But the distribution of the residuals does reveal, that AVM tends to overestimate the price, which might drag down the performance.

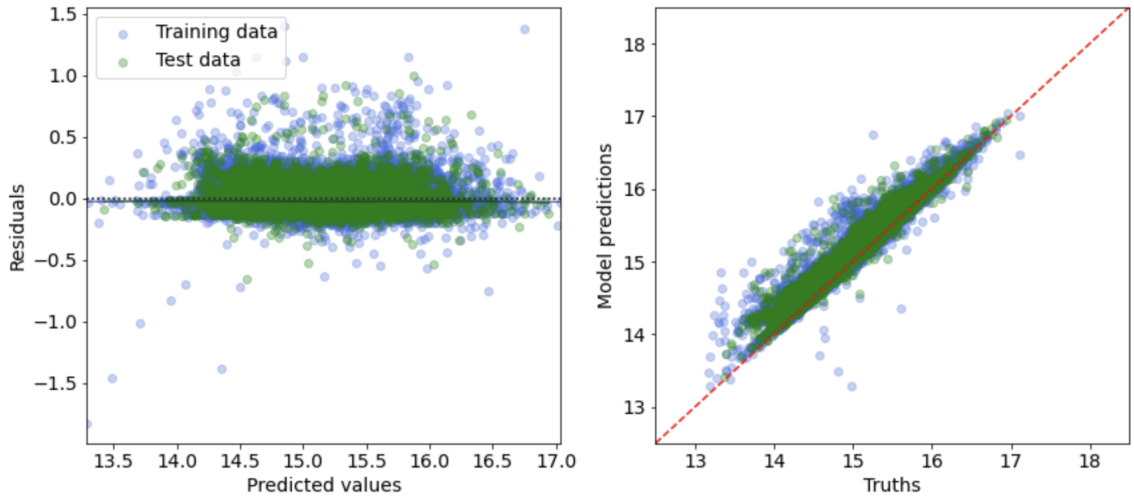

Figure 6.18: AVM residuals (left) and predicted response plotted against true response (right).

The performance of the AVM is given in tables (6.15) and (6.16), and gives results that are not up to par with the neural network model of this thesis. The quality of fit measures are of course almost identical for training and test set, since there is no actual training going on. The AVM only predicts just about 70 % of the test observations within 15 % of the actual price.

[2] `https://geomatic.dk/dk/services/data-lake/ejendomsvaerdimodel-avm/`

	MSE, train	MAE, train	R^2, train	MSE, test	MAE, test	R^2, test
AVM	0.031669	0.136752	0.870476	0.030802	0.13764	0.871138

Table 6.15: Performance of using AVM for prediction.

	Mean deviation %	Median deviation %	Within 15 %
AVM	15.48	12.06	69.42

Table 6.16: Deviation of predicted prices from actual prices in test data when using AVM for prediction.

6.5 Results

It is now time to conclude the modelling chapter, which has gone through the machine learning models chosen to model prices in the Copenhagen housing market in the period 2016-2020. This thesis has focused on simple linear models in comparison with neural network models in order to make the best possible fit, which could be used for predicting housing prices. The results of the section are summarised by tables (6.17) and (6.18), which give an overall picture of how the different models performed on the data set.
Table (6.17) sums up the different measures of fit, both on the training data and on the test data. Apart from estimating the expected prediction error by the test error measures for each model, this enabled us to see if any overfitting was present. This seemed not to be a big issue for any of the models, on the contrary the linear models, OLS and Lasso, performed better on the test set than on the training set. That being said, these two models were the worst performing of our own models, with a higher error and a lower R^2. Due to the size of the dataset compared to the number of predictors, there was little use for the Lasso regularization, and a very small α led to almost identical performance as in OLS.
Change came when using the neural network setup, where non-linearity was introduced through the ReLU activation functions. The results speak for themselves, and it was found that the non-linearity was much better accomodated by this model. The results could be seen to improve even further when including asking price as a predictor, but this is not surprising, since the actual price is often close to the asking price. Therefore the very best predictions were made, if one just let the asking price be the prediction. But the inclusion of the asking price is problematic, because in theory this is not readily available, when one wants to give a prediction of a sales price - only when a price has already been set by the real estate agent.
And lastly, the AVM from Geomatic was included, and the performance of this state of the art model can surprisingly be seen to be on par or worse than the other models. It only performs slightly better than the linear models, even though it was found that it does accommodate the non-linearity in the data. But it was found to have a tendency of overestimating sales prices.

Table (6.18) overall gives the same impression in term of performances, summing up deviation of the predictions from the real prices on the test set. The mean deviation is constantly higher than the median deviation, which leads one to think, that there are outliers in the test set. As before AVM underperforms, and there is even 10 %-points difference between the AVM and the neural network model, when it comes to the fraction predicted within 15 % from the truth. It has become clear in this thesis, that the neural network setup is well suited to the Copenhagen housing data, and in order to make even better predictions, this is a good direction to go in further development. .

	MSE, train	MAE, train	R^2, train	MSE, test	MAE, test	R^2, test
OLS	0.036	0.132	0.860	0.035	0.131	0.863
Lasso	0.036	0.132	0.860	0.035	0.131	0.863
NN	0.020	0.089	0.920	0.028	0.104	0.890
OLS Cash	0.029	0.113	0.887	0.027	0.112	0.894
NN Cash	0.012	0.054	0.954	0.014	0.060	0.944
Cash	0.015	0.052	0.940	0.013	0.051	0.947
AVM	0.032	0.137	0.870	0.031	0.138	0.871

Table 6.17: Performance of all models considered.

	Mean deviation %	Median deviation %	Within 15 %
OLS	13.85	9.77	69.33
Lasso	13.86	9.77	69.40
NN	11.08	7.16	80.73
OLS Cash	11.88	8.17	77.45
NN Cash	6.35	3.23	92.12
Cash	5.30	2.41	92.89
AVM	15.48	12.06	69.42

Table 6.18: Deviation of predicted prices from actual prices in test data for all models considered.

6.6 Prediction in times of Corona

As we all know, COVID-19 got a firm grip of the world in 2020. Other than being devastating for so many on a personal level, it left the world economy in a state of uncertainty, not seen since the financial crisis of 2008. Even though the outcome was not as dire for the economy, and no full scale meltdown of the housing market occurred, it still created an interesting example for housing price prediction using the model of this thesis. Since there was a clear shift in circumstances from 2019 to 2020, it seems an obvious chance to test the predictive capabilities of the models, to see how well they do when circumstances are changed. So in this example, the train/test split are set according to year, letting the training set consist of observation from 2016-2019, and the test set consist of observations from 2020.
Figure (6.19) shows the development in the price indecies for housing in the capital region of Copenhagen in the year 2020. Surprisingly, there was no slow-down in the growth in prices, which has been seen almost over the entire period of the data since 2016. Therefore, using 2020 observations as test set might not create as big of a challenge as one might think.

Figure (6.20) shows how the the models fit, when this new Corona-split was used. Again, the OLS linear model has the problem of accommodating the non-linearity in the data, while the neural network model gets around that problem. What is noteworthy is that both models seem to tend to overestimate prices for the test set. This could be because the models take account of the higher pricing index in the test set, but don't take account of some other variable, which resulted in a smaller increase in prices than predicted. In both cases, the models are extrapolating into a new year, for which they haven't seen new data, and this apparently can be quite challenging for the models.

The assessment of the performance of the two models speaks for itself in tables (6.19) and (6.20). It really is not an easy task for models to look into the future, maybe especially not when a situation like a pandemic comes into play. The models are fitted to the training data, so that the performance is more or less equivalent to the training set performance with the earlier train/test-split. But when the fitted mode is then used on the 2020 test data, it goes all wrong.

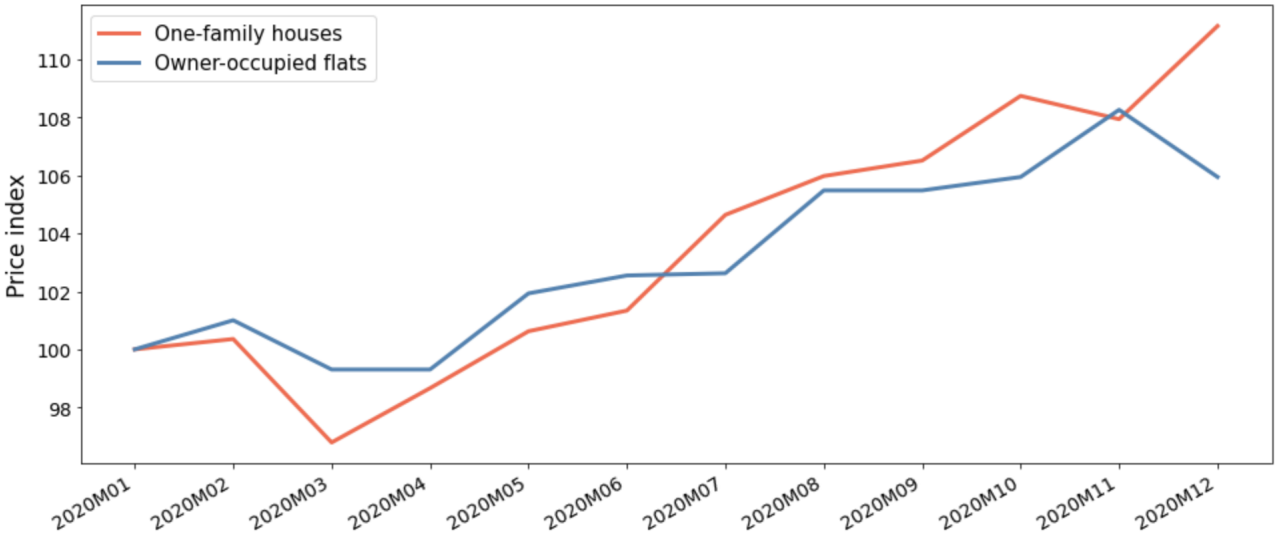

Figure 6.19: Development in housing prices in the capital region of Copenhagen in 2020.

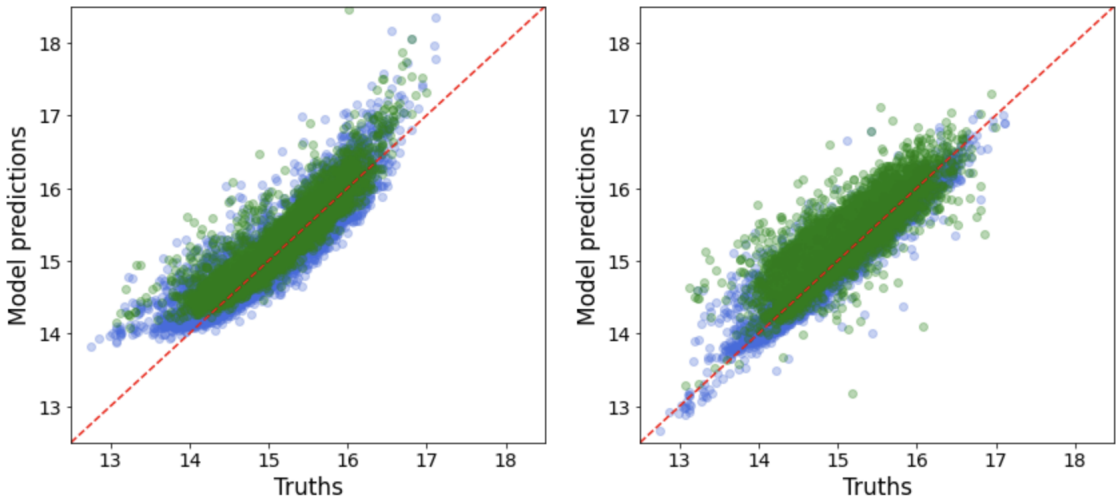

Figure 6.20: Predicted response plotted against true response for OLS linear model (left) and neural network model (right).

We see much worse quality of fit measurements, and the deviation from true prices are through the roof. Interestingly, the OLS linear model is better at extrapolating into the future, since it does significantly better than the neural network model, which only get 33 % of the predictions on the test set within 15 % of the true values.
To sum up, the models are clearly not very helpful as forecasting tools, and therefore this leaves room for a lot of work. But one must also conclude, that forecasting is not an easy task.

	MSE, train	MAE, train	R^2, train	MSE, test	MAE, test	R^2, test
OLS Corona	0.035	0.13	0.864	0.062	0.185	0.754
NN Corona	0.019	0.09	0.927	0.133	0.287	0.470

Table 6.19: Performance of models with test set in 2020.

	Mean deviation %	Median deviation %	Within 15 %
OLS Corona	22.00	16.03	47.22
NN Corona	36.18	27.22	33.11

Table 6.20: Deviation of predicted prices from actual prices in 2020 test data.

Chapter 7

Conclusion

In this thesis, we set out to describe the hot topic of Copenhagen housing prices using machine learning techniques. The initial search for the data needed was extensive and comprised four different sources of data (Boligsiden, DinGeo, Hvorlangerder.dk and Statbank), which all provided a great deal of information about housing sales in the period 2016-2020. The websites were crawled and scraped using the libraries BeautifulSoup, Selenium and Requests, resulting in information about 40,606 sales. The sales were not equally distributed over the period, since the scraping from the outset could not accommodate multiple sales for a single address, so only the latest free market sale per address was included. At the outset, 289 variables made up the data for every single address, containing information such as housing and sales specifics, environmental/societal aspects, distances to key places and economic indicators. But this vast amount of information needed a great deal of work to be ready for modelling work. This was the focus of the data cleaning section of the thesis, where rough cleaning sorted out irrelevant variables, and fine cleaning in broad terms dealt with missing data.
Exploratory analysis revealed further need for cleaning out observations, since outliers of for example unrealistic m^2-prices would be potentially harmful when used in modelling. It also revealed the need to for example delete area variables to avoid collinearity, and to delete categoricals with too little or too much variation. When all was said and done, the final data set contained 29,830 observations and 95 variables, which is quite a reduction, but nonetheless found necessary for a good modelling foundation.
As stated, machine learning was the approach of the thesis, or supervised learning to use another term: the process of learning a model by fitting it to the data available. Therefore, the data was split in a training set, comprising 75 % of the observations at random, while leaving out a test set. The point of the test set is to assess the quality of a model, since one basically approximates the expected test error, that is sought minimized. The best model is the one which best approximates the underlying 'true' model describing the distributions of predictors and response. For such a model, considerations of possible bias and variance must be balanced in order to have a strong predictive model that doesn't overfit. Different measures for quality of fit have been presented (MSE, MAE and R^2), and set to be the baseline upon which different models were to be judged.
The models that were utilized could be divided into two groups: linear models and artificial neural networks. The first being the more standard approach to regression problems, while the other being a relatively new field with much attention and potential. The linear models were created using the Scikit-learn library for Python, while the Keras library for TensorFlow in Python was used for the neural network model.
The first model for the regression problem in question was the OLS linear model. This model is linear in the parameters, and for simplicity was also chosen to be linear in the predictors. It has attractive properties such as a closed form solution when fitting, and the parameter estimate is also the maximization estimator for the linear model. Furthermore it has the advantage of

the relative ease of interpretation of the parameter estimates, and in our case it showed that weighted area and property valuation were the key predictors. It also showed that a surprisingly big amount of predictors had very low influence by size of parameter.
The model was found to do relatively well, when the sale price as the response had been log-transformed to avoid heteroscedasticity. On the test set it was able to predict about 70 % within 15 % of the true price, and it actually performed better on the test set than on the set on which it was trained. From the residuals, one could see that non-linearity in the data was not accommodated by the model, and this bias of the model affected the performance negatively.
The next model to be tried out was the Lasso linear model. As a regularization model, it adds a penalty term to the loss function of the regular OLS, in order to shrink some of the parameter estimates to zero. This was also the case when using the model, but only for few predictors. The lasso has its benefits, when there is a tendency to overfitting - oftentimes when there are many predictors compared to observations. But since this was not the case, the tuning parameter α was set very low through cross validation, and this essentially meant that the lasso model became very similar to OLS linear model. The performance of the model on both training and test sets was therefore almost identical to OLS and was therefore not deemed very interesting in our case.
Lastly among our own models was the neural network model, which was set to be a two hidden layer feed forward network with non-linear ReLU activation functions and an online learning optimizer through 400 epochs. Through back-propagation and gradient descent the model was trained by correcting the weights and biases, and this payed off. The model performed much better on both training and test sets, and could predict 80 % within 15 % of actual price on the test set. This improvement was attributed to the fact that the neural network model visibly was able to accommodate the non-linearity in the data, and thereby avoiding the bias present in the linear models.
For comparison, the models were expanded to include asking price, and in all cases seen to do much better. The best performance turned out to just be using asking price for prediction, and this predicted 93 % within 15 % of actual price on the test set. But it was found that the relevance of this comparison is questionable, since the asking price itself can be said to be a result of the pricing process.
Another comparison was done with a state of the art pricing model, the AVM from Geomatic, which turned out to perform worse than our neural network model, mostly due to predicting prices too high. It therefore was able to predict about 70 % within 15 % of actual price on the test set.
So when modelling the Copenhagen housing market, this thesis showed how neural networks hold great promise for further development, since even this simple version was able to compete, and outperform, AVM.
The thesis culminated with a new split of the data to investigate the predictive power of the models in the time of Corona. A 2020 test set describes a year which was anything but normal, though the housing prices continued to rise as before through the year. Using this new split, the performances of both the OLS linear model and the neural network model were very disappointing, and OLS even did better in this case of extrapolation than the neural network, which only predicted 33 % within 15 % of actual price on the test set. Whether this can be attributed to a year, which differed wildly, or to a weakness in the model (that it needs to include data from all periods to fit properly) is not certain. But it would be an interesting area of further research when regarding the many facets of the housing market of Copenhagen.

Appendix A

Appendix

- **All code written in Python available in notebook form on GitHub:** `https://github.com/fredbuscma/Frederik_Madsen_Master_Thesis`

References

[Bishop, 2007] Bishop, C. M. (2007). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer, 1 edition.

[Boligsiden A/S, 2021] Boligsiden A/S (2021). Boligsiden. `https://www.boligsiden.dk/`.

[Chollet et al., 2015] Chollet, F. et al. (2015). Keras. `https://github.com/fchollet/keras`.

[Dingeo.dk Aps, 2021] Dingeo.dk Aps (2021). Dingeo. `https://www.dingeo.dk/`.

[Geomatic, 2021] Geomatic (2021). Avm. `https://geomatic.dk/dk/services/data-lake/ejendomsvaerdimodel-avm/`.

[Google, 2021] Google (2021). Google maps. `https://www.google.com/maps/`.

[Hamster, 2021] Hamster (2021). Hamster map. `http://www.hamstermap.com/`.

[Hansen, 2002] Hansen, E. (2002). *Introduktion til Matematisk Statistik*. Afdeling for Teoretisk Statistik, K.U.,.

[Hansen, 2006] Hansen, E. (2006). *Measure Theory*. Afdeling for Teoretisk Statistik, K.U.,.

[Hansen, 2015] Hansen, N. (2015). *Regression with R*. H.C.Ø.-Tryk.

[Hastie et al., 2001] Hastie, T., Tibshirani, R., and Friedman, J. (2001). *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., New York, NY, USA.

[Igel, 2020] Igel, C. (2020). Lecture notes on machine learning: Kernel-based methods.

[James et al., 2013] James, G., Witten, D., Hastie, T., and Tibshirani, R. (2013). *An Introduction to Statistical Learning: with Applications in R*. Springer.

[KMD, 2021] KMD (2021). Folketingsvalg. `https://www.kmdvalg.dk/fv/2019/KMDValgFV.html`.

[Knudsen, 2020] Knudsen, J. S. (2020). Predicting-house-prices-in-copenhagen-using-machine-learning. `https://github.com/JohanneSihmKnudsen/Predicting-House-Prices-in-Copenhagen-using-Machine-Learning`.

[Københavns Kommune, 2021] Københavns Kommune (2021). Københavnerkortet. `https://kbhkort.kk.dk/spatialmap`.

[Pedregosa et al., 2011] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

[Raschka and Mirjalili, 2019] Raschka, S. and Mirjalili, V. (2019). *Python Machine Learning, 3rd Ed.* Packt Publishing, Birmingham, UK.

[Statistics Denmark, 2021] Statistics Denmark (2021). Statbank. `https://www.statbank.dk/`.

[Viamap, 2021] Viamap (2021). Hvorlangterder.dk. `https://hvorlangterder.dk/`.